DATA MINING AND ANALYSIS

The fundamental algorithms in data mining and analysis form the basis for the emerging field of data science, which includes automated methods to analyze patterns and models for all kinds of data, with applications ranging from scientific discovery to business intelligence and analytics. This textbook for senior undergraduate and graduate data mining courses provides a broad yet in-depth overview of data mining, integrating related concepts from machine learning and statistics. The main parts of the book include exploratory data analysis, pattern mining, clustering, and classification. The book lays the basic foundations of these tasks and also covers cutting-edge topics such as kernel methods, high-dimensional data analysis, and complex graphs and networks. With its comprehensive coverage, algorithmic perspective, and wealth of examples, this book offers solid guidance in data mining for students, researchers, and practitioners alike.

Key Features:

- Covers both core methods and cutting-edge research
- Algorithmic approach with open-source implementations
- Minimal prerequisites, as all key mathematical concepts are presented, as is the intuition behind the formulas
- Short, self-contained chapters with class-tested examples and exercises that allow for flexibility in designing a course and for easy reference
- Supplementary online resource containing lecture slides, videos, project ideas, and more

Mohammed J. Zaki is a Professor of Computer Science at Rensselaer Polytechnic Institute, Troy, New York.

Wagner Meira Jr. is a Professor of Computer Science at Universidade Federal de Minas Gerais, Brazil.

DATA MINING AND ANALYSIS

Fundamental Concepts and Algorithms

MOHAMMED J. ZAKI
Rensselaer Polytechnic Institute, Troy, New York

WAGNER MEIRA JR.
Universidade Federal de Minas Gerais, Brazil

CAMBRIDGE
UNIVERSITY PRESS

CAMBRIDGE
UNIVERSITY PRESS

32 Avenue of the Americas, New York, NY 10013-2473, USA

Cambridge University Press is part of the University of Cambridge.

It furthers the University's mission by disseminating knowledge in the pursuit of education, learning, and research at the highest international levels of excellence.

www.cambridge.org
Information on this title: www.cambridge.org/9780521766333

First published 2014

Printed in the United States of America

A catalog record for this publication is available from the British Library.

Library of Congress Cataloging in Publication Data
Zaki, Mohammed J., 1971–
Data mining and analysis: fundamental concepts and algorithms / Mohammed J. Zaki, Rensselaer Polytechnic Institute, Troy, New York, Wagner Meira Jr., Universidade Federal de Minas Gerais, Brazil.
pages cm
Includes bibliographical references and index.
ISBN 978-0-521-76633-3 (hardback)
1. Data mining. I. Meira, Wagner, 1967– II. Title.
QA76.9.D343Z36 2014
006.3′12–dc23 2013037544

ISBN 978-0-521-76633-3 Hardback

Contents

Preface

This book is an outgrowth of data mining courses at Rensselaer Polytechnic Institute (RPI) and Universidade Federal de Minas Gerais (UFMG); the RPI course has been offered every Fall since 1998, whereas the UFMG course has been offered since 2002. Although there are several good books on data mining and related topics, we felt that many of them are either too high-level or too advanced. Our goal was to write an introductory text that focuses on the fundamental algorithms in data mining and analysis. It lays the mathematical foundations for the core data mining methods, with key concepts explained when first encountered; the book also tries to build the intuition behind the formulas to aid understanding.

The main parts of the book include exploratory data analysis, frequent pattern mining, clustering, and classification. The book lays the basic foundations of these tasks, and it also covers cutting-edge topics such as kernel methods, high-dimensional data analysis, and complex graphs and networks. It integrates concepts from related disciplines such as machine learning and statistics and is also ideal for a course on data analysis. Most of the prerequisite material is covered in the text, especially on linear algebra, and probability and statistics.

The book includes many examples to illustrate the main technical concepts. It also has end-of-chapter exercises, which have been used in class. All of the algorithms in the book have been implemented by the authors. We suggest that readers use their favorite data analysis and mining software to work through our examples and to implement the algorithms we describe in text; we recommend the R software or the Python language with its NumPy package. The datasets used and other supplementary material such as project ideas and slides are available online at the book's companion site and its mirrors at RPI and UFMG:

- `http://dataminingbook.info`
- `http://www.cs.rpi.edu/~zaki/dataminingbook`
- `http://www.dcc.ufmg.br/dataminingbook`

Having understood the basic principles and algorithms in data mining and data analysis, readers will be well equipped to develop their own methods or use more advanced techniques.

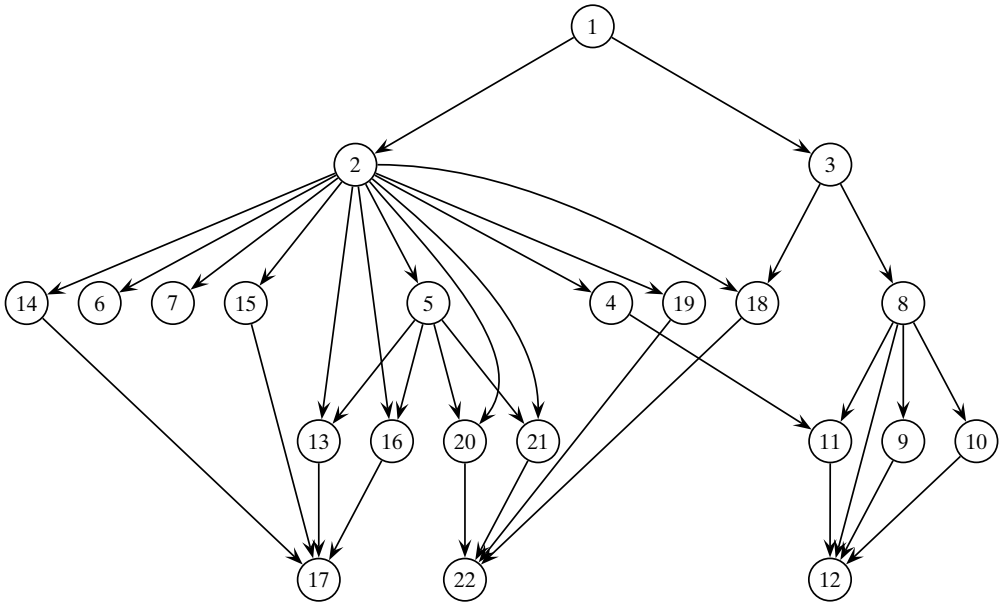


Figure 0.1. Chapter dependencies

Suggested Roadmaps

The chapter dependency graph is shown in Figure 0.1. We suggest some typical roadmaps for courses and readings based on this book. For an undergraduate-level course, we suggest the following chapters: 1–3, 8, 10, 12–15, 17–19, and 21–22. For an undergraduate course without exploratory data analysis, we recommend Chapters 1, 8–15, 17–19, and 21–22. For a graduate course, one possibility is to quickly go over the material in Part I or to assume it as background reading and to directly cover Chapters 9–22; the other parts of the book, namely frequent pattern mining (Part II), clustering (Part III), and classification (Part IV), can be covered in any order. For a course on data analysis the chapters covered must include 1–7, 13–14, 15 (Section 2), and 20. Finally, for a course with an emphasis on graphs and kernels we suggest Chapters 4, 5, 7 (Sections 1–3), 11–12, 13 (Sections 1–2), 16–17, and 20–22.

Acknowledgments

Initial drafts of this book have been used in several data mining courses. We received many valuable comments and corrections from both the faculty and students. Our thanks go to

- Muhammad Abulaish, Jamia Millia Islamia, India
- Mohammad Al Hasan, Indiana University Purdue University at Indianapolis
- Marcio Luiz Bunte de Carvalho, Universidade Federal de Minas Gerais, Brazil
- Loïc Cerf, Universidade Federal de Minas Gerais, Brazil
- Ayhan Demiriz, Sakarya University, Turkey
- Murat Dundar, Indiana University Purdue University at Indianapolis
- Jun Luke Huan, University of Kansas
- Ruoming Jin, Kent State University
- Latifur Khan, University of Texas, Dallas

- Pauli Miettinen, Max-Planck-Institut für Informatik, Germany
- Suat Ozdemir, Gazi University, Turkey
- Naren Ramakrishnan, Virginia Polytechnic and State University
- Leonardo Chaves Dutra da Rocha, Universidade Federal de São João del-Rei, Brazil
- Saeed Salem, North Dakota State University
- Ankur Teredesai, University of Washington, Tacoma
- Hannu Toivonen, University of Helsinki, Finland
- Adriano Alonso Veloso, Universidade Federal de Minas Gerais, Brazil
- Jason T.L. Wang, New Jersey Institute of Technology
- Jianyong Wang, Tsinghua University, China
- Jiong Yang, Case Western Reserve University
- Jieping Ye, Arizona State University

We would like to thank all the students enrolled in our data mining courses at RPI and UFMG, as well as the anonymous reviewers who provided technical comments on various chapters. We appreciate the collegial and supportive environment within the computer science departments at RPI and UFMG and at the Qatar Computing Research Institute. In addition, we thank NSF, CNPq, CAPES, FAPEMIG, Inweb – the National Institute of Science and Technology for the Web, and Brazil's Science without Borders program for their support. We thank Lauren Cowles, our editor at Cambridge University Press, for her guidance and patience in realizing this book.

Finally, on a more personal front, MJZ dedicates the book to his wife, Amina, for her love, patience and support over all these years, and to his children, Abrar and Afsah, and his parents. WMJ gratefully dedicates the book to his wife Patricia; to his children, Gabriel and Marina; and to his parents, Wagner and Marlene, for their love, encouragement, and inspiration.

CHAPTER 1 Data Mining and Analysis

Data mining is the process of discovering insightful, interesting, and novel patterns, as well as descriptive, understandable, and predictive models from large-scale data. We begin this chapter by looking at basic properties of data modeled as a data matrix. We emphasize the geometric and algebraic views, as well as the probabilistic interpretation of data. We then discuss the main data mining tasks, which span exploratory data analysis, frequent pattern mining, clustering, and classification, laying out the roadmap for the book.

1.1 DATA MATRIX

Data can often be represented or abstracted as an $n \times d$ *data matrix*, with n rows and d columns, where rows correspond to entities in the dataset, and columns represent attributes or properties of interest. Each row in the data matrix records the observed attribute values for a given entity. The $n \times d$ data matrix is given as

$$\mathbf{D} = \begin{pmatrix} & X_1 & X_2 & \cdots & X_d \\ \mathbf{x}_1 & x_{11} & x_{12} & \cdots & x_{1d} \\ \mathbf{x}_2 & x_{21} & x_{22} & \cdots & x_{2d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{x}_n & x_{n1} & x_{n2} & \cdots & x_{nd} \end{pmatrix}$$

where $\mathbf{x}_i$ denotes the ith row, which is a d-tuple given as

$$\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{id})$$

and X_j denotes the jth column, which is an n-tuple given as

$$X_j = (x_{1j}, x_{2j}, \ldots, x_{nj})$$

Depending on the application domain, rows may also be referred to as *entities*, *instances*, *examples*, *records*, *transactions*, *objects*, *points*, *feature-vectors*, *tuples*, and so on. Likewise, columns may also be called *attributes*, *properties*, *features*, *dimensions*, *variables*, *fields*, and so on. The number of instances n is referred to as the *size* of

Table 1.1. Extract from the Iris dataset

	Sepal length X_1	Sepal width X_2	Petal length X_3	Petal width X_4	Class X_5
$\mathbf{x}_1$	5.9	3.0	4.2	1.5	Iris-versicolor
$\mathbf{x}_2$	6.9	3.1	4.9	1.5	Iris-versicolor
$\mathbf{x}_3$	6.6	2.9	4.6	1.3	Iris-versicolor
$\mathbf{x}_4$	4.6	3.2	1.4	0.2	Iris-setosa
$\mathbf{x}_5$	6.0	2.2	4.0	1.0	Iris-versicolor
$\mathbf{x}_6$	4.7	3.2	1.3	0.2	Iris-setosa
$\mathbf{x}_7$	6.5	3.0	5.8	2.2	Iris-virginica
$\mathbf{x}_8$	5.8	2.7	5.1	1.9	Iris-virginica
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\mathbf{x}_{149}$	7.7	3.8	6.7	2.2	Iris-virginica
$\mathbf{x}_{150}$	5.1	3.4	1.5	0.2	Iris-setosa

the data, whereas the number of attributes d is called the *dimensionality* of the data. The analysis of a single attribute is referred to as *univariate analysis*, whereas the simultaneous analysis of two attributes is called *bivariate analysis* and the simultaneous analysis of more than two attributes is called *multivariate analysis*.

Example 1.1. Table 1.1 shows an extract of the Iris dataset; the complete data forms a 150×5 data matrix. Each entity is an Iris flower, and the attributes include `sepal length`, `sepal width`, `petal length`, and `petal width` in centimeters, and the type or `class` of the Iris flower. The first row is given as the 5-tuple

$$\mathbf{x}_1 = (5.9, 3.0, 4.2, 1.5, \texttt{Iris-versicolor})$$

Not all datasets are in the form of a data matrix. For instance, more complex datasets can be in the form of sequences (e.g., DNA and protein sequences), text, time-series, images, audio, video, and so on, which may need special techniques for analysis. However, in many cases even if the raw data is not a data matrix it can usually be transformed into that form via feature extraction. For example, given a database of images, we can create a data matrix in which rows represent images and columns correspond to image features such as color, texture, and so on. Sometimes, certain attributes may have special semantics associated with them requiring special treatment. For instance, temporal or spatial attributes are often treated differently. It is also worth noting that traditional data analysis assumes that each entity or instance is independent. However, given the interconnected nature of the world we live in, this assumption may not always hold. Instances may be connected to other instances via various kinds of relationships, giving rise to a *data graph*, where a node represents an entity and an edge represents the relationship between two entities.

1.2 ATTRIBUTES

Attributes may be classified into two main types depending on their domain, that is, depending on the types of values they take on.

Numeric Attributes

A *numeric* attribute is one that has a real-valued or integer-valued domain. For example, Age with $domain(\texttt{Age}) = \mathbb{N}$, where $\mathbb{N}$ denotes the set of natural numbers (non-negative integers), is numeric, and so is petal length in Table 1.1, with $domain(\texttt{petal length}) = \mathbb{R}^+$ (the set of all positive real numbers). Numeric attributes that take on a finite or countably infinite set of values are called *discrete*, whereas those that can take on any real value are called *continuous*. As a special case of discrete, if an attribute has as its domain the set $\{0, 1\}$, it is called a *binary* attribute. Numeric attributes can be classified further into two types:

- *Interval-scaled*: For these kinds of attributes only differences (addition or subtraction) make sense. For example, attribute temperature measured in °C or °F is interval-scaled. If it is 20 °C on one day and 10 °C on the following day, it is meaningful to talk about a temperature drop of 10 °C, but it is not meaningful to say that it is twice as cold as the previous day.
- *Ratio-scaled*: Here one can compute both differences as well as ratios between values. For example, for attribute Age, we can say that someone who is 20 years old is twice as old as someone who is 10 years old.

Categorical Attributes

A *categorical* attribute is one that has a set-valued domain composed of a set of symbols. For example, Sex and Education could be categorical attributes with their domains given as

$$domain(\texttt{Sex}) = \{\texttt{M}, \texttt{F}\}$$

$$domain(\texttt{Education}) = \{\texttt{HighSchool}, \texttt{BS}, \texttt{MS}, \texttt{PhD}\}$$

Categorical attributes may be of two types:

- *Nominal*: The attribute values in the domain are unordered, and thus only equality comparisons are meaningful. That is, we can check only whether the value of the attribute for two given instances is the same or not. For example, Sex is a nominal attribute. Also class in Table 1.1 is a nominal attribute with $domain(\texttt{class}) = \{\texttt{iris-setosa}, \texttt{iris-versicolor}, \texttt{iris-virginica}\}$.
- *Ordinal*: The attribute values are ordered, and thus both equality comparisons (is one value equal to another?) and inequality comparisons (is one value less than or greater than another?) are allowed, though it may not be possible to quantify the difference between values. For example, Education is an ordinal attribute because its domain values are ordered by increasing educational qualification.

1.3 DATA: ALGEBRAIC AND GEOMETRIC VIEW

If the d attributes or dimensions in the data matrix $\mathbf{D}$ are all numeric, then each row can be considered as a d-dimensional point:

$$\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{id}) \in \mathbb{R}^d$$

or equivalently, each row may be considered as a d-dimensional column vector (all vectors are assumed to be column vectors by default):

$$\mathbf{x}_i = \begin{pmatrix} x_{i1} \\ x_{i2} \\ \vdots \\ x_{id} \end{pmatrix} = \begin{pmatrix} x_{i1} & x_{i2} & \cdots & x_{id} \end{pmatrix}^T \in \mathbb{R}^d$$

where T is the *matrix transpose* operator.

The d-dimensional Cartesian coordinate space is specified via the d unit vectors, called the standard basis vectors, along each of the axes. The jth *standard basis vector* $\mathbf{e}_j$ is the d-dimensional unit vector whose jth component is 1 and the rest of the components are 0

$$\mathbf{e}_j = (0, \ldots, 1_j, \ldots, 0)^T$$

Any other vector in $\mathbb{R}^d$ can be written as *linear combination* of the standard basis vectors. For example, each of the points $\mathbf{x}_i$ can be written as the linear combination

$$\mathbf{x}_i = x_{i1}\mathbf{e}_1 + x_{i2}\mathbf{e}_2 + \cdots + x_{id}\mathbf{e}_d = \sum_{j=1}^{d} x_{ij}\mathbf{e}_j$$

where the scalar value x_{ij} is the coordinate value along the jth axis or attribute.

Example 1.2. Consider the Iris data in Table 1.1. If we *project* the entire data onto the first two attributes, then each row can be considered as a point or a vector in 2-dimensional space. For example, the projection of the 5-tuple $\mathbf{x}_1 = (5.9, 3.0, 4.2, 1.5, \texttt{Iris-versicolor})$ on the first two attributes is shown in Figure 1.1a. Figure 1.2 shows the scatterplot of all the $n = 150$ points in the 2-dimensional space spanned by the first two attributes. Likewise, Figure 1.1b shows $\mathbf{x}_1$ as a point and vector in 3-dimensional space, by projecting the data onto the first three attributes. The point $(5.9, 3.0, 4.2)$ can be seen as specifying the coefficients in the linear combination of the standard basis vectors in $\mathbb{R}^3$:

$$\mathbf{x}_1 = 5.9\mathbf{e}_1 + 3.0\mathbf{e}_2 + 4.2\mathbf{e}_3 = 5.9\begin{pmatrix}1\\0\\0\end{pmatrix} + 3.0\begin{pmatrix}0\\1\\0\end{pmatrix} + 4.2\begin{pmatrix}0\\0\\1\end{pmatrix} = \begin{pmatrix}5.9\\3.0\\4.2\end{pmatrix}$$

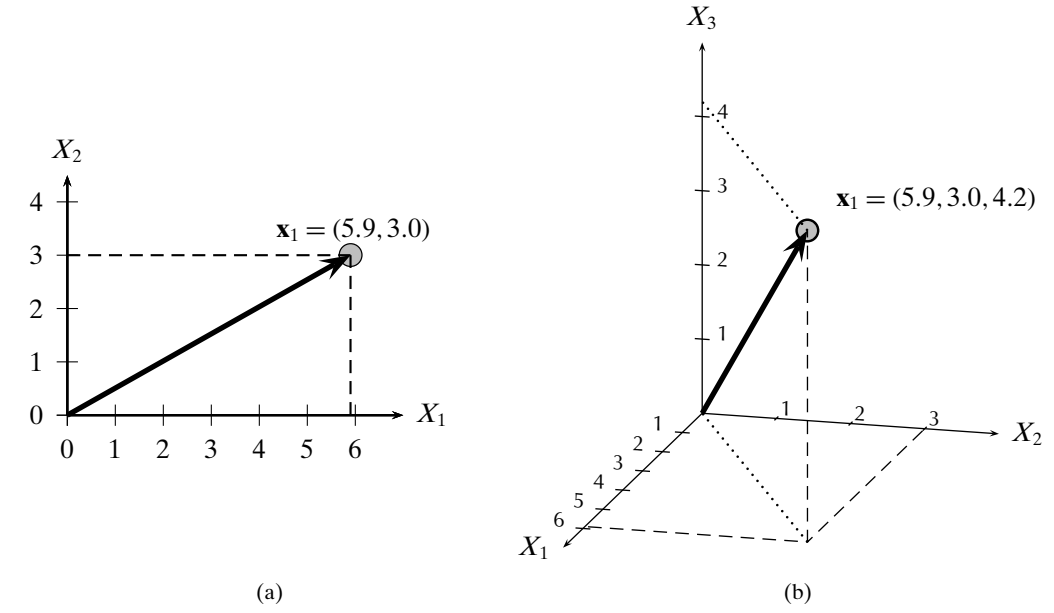


Figure 1.1. Row $\mathbf{x}_1$ as a point and vector in (a) $\mathbb{R}^2$ and (b) $\mathbb{R}^3$.

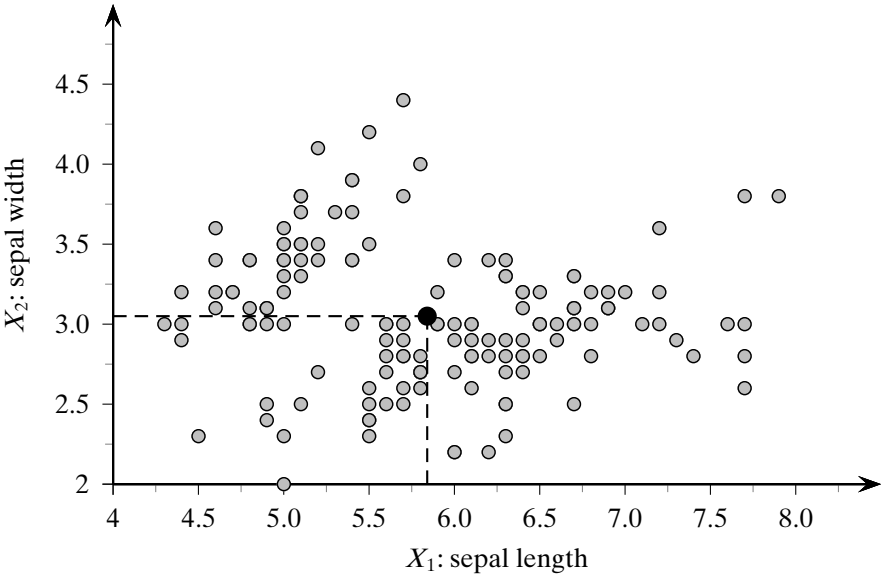


Figure 1.2. Scatterplot: `sepal length` versus `sepal width`. The solid circle shows the mean point.

Each numeric column or attribute can also be treated as a vector in an n-dimensional space $\mathbb{R}^n$:

$$X_j = \begin{pmatrix} x_{1j} \\ x_{2j} \\ \vdots \\ x_{nj} \end{pmatrix}$$

If all attributes are numeric, then the data matrix $\mathbf{D}$ is in fact an $n \times d$ matrix, also written as $\mathbf{D} \in \mathbb{R}^{n \times d}$, given as

$$\mathbf{D} = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1d} \\ x_{21} & x_{22} & \cdots & x_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nd} \end{pmatrix} = \begin{pmatrix} — \mathbf{x}_1^T — \\ — \mathbf{x}_2^T — \\ \vdots \\ — \mathbf{x}_n^T — \end{pmatrix} = \begin{pmatrix} | & | & & | \\ X_1 & X_2 & \cdots & X_d \\ | & | & & | \end{pmatrix}$$

As we can see, we can consider the entire dataset as an $n \times d$ matrix, or equivalently as a set of n row vectors $\mathbf{x}_i^T \in \mathbb{R}^d$ or as a set of d column vectors $X_j \in \mathbb{R}^n$.

1.3.1 Distance and Angle

Treating data instances and attributes as vectors, and the entire dataset as a matrix, enables one to apply both geometric and algebraic methods to aid in the data mining and analysis tasks.

Let $\mathbf{a}, \mathbf{b} \in \mathbb{R}^m$ be two m-dimensional vectors given as

$$\mathbf{a} = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{pmatrix} \qquad \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}$$

Dot Product

The *dot product* between $\mathbf{a}$ and $\mathbf{b}$ is defined as the scalar value

$$\begin{aligned} \mathbf{a}^T\mathbf{b} &= \begin{pmatrix} a_1 & a_2 & \cdots & a_m \end{pmatrix} \times \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix} \\ &= a_1 b_1 + a_2 b_2 + \cdots + a_m b_m \\ &= \sum_{i=1}^{m} a_i b_i \end{aligned}$$

Length

The *Euclidean norm* or *length* of a vector $\mathbf{a} \in \mathbb{R}^m$ is defined as

$$\|\mathbf{a}\| = \sqrt{\mathbf{a}^T\mathbf{a}} = \sqrt{a_1^2 + a_2^2 + \cdots + a_m^2} = \sqrt{\sum_{i=1}^{m} a_i^2}$$

The *unit vector* in the direction of $\mathbf{a}$ is given as

$$\mathbf{u} = \frac{\mathbf{a}}{\|\mathbf{a}\|} = \left(\frac{1}{\|\mathbf{a}\|}\right)\mathbf{a}$$

By definition $\mathbf{u}$ has length $\|\mathbf{u}\| = 1$, and it is also called a *normalized* vector, which can be used in lieu of $\mathbf{a}$ in some analysis tasks.

The Euclidean norm is a special case of a general class of norms, known as L_p*-norm*, defined as

$$\|\mathbf{a}\|_p = \left(|a_1|^p + |a_2|^p + \cdots + |a_m|^p\right)^{\frac{1}{p}} = \left(\sum_{i=1}^{m} |a_i|^p\right)^{\frac{1}{p}}$$

for any $p \neq 0$. Thus, the Euclidean norm corresponds to the case when $p = 2$.

Distance

From the Euclidean norm we can define the *Euclidean distance* between $\mathbf{a}$ and $\mathbf{b}$, as follows

$$\delta(\mathbf{a}, \mathbf{b}) = \|\mathbf{a} - \mathbf{b}\| = \sqrt{(\mathbf{a} - \mathbf{b})^T(\mathbf{a} - \mathbf{b})} = \sqrt{\sum_{i=1}^{m} (a_i - b_i)^2} \tag{1.1}$$

Thus, the length of a vector is simply its distance from the zero vector $\mathbf{0}$, all of whose elements are 0, that is, $\|\mathbf{a}\| = \|\mathbf{a} - \mathbf{0}\| = \delta(\mathbf{a}, \mathbf{0})$.

From the general L_p-norm we can define the corresponding L_p-distance function, given as follows

$$\delta_p(\mathbf{a}, \mathbf{b}) = \|\mathbf{a} - \mathbf{b}\|_p \tag{1.2}$$

If p is unspecified, as in Eq. (1.1), it is assumed to be $p = 2$ by default.

Angle

The cosine of the smallest angle between vectors $\mathbf{a}$ and $\mathbf{b}$, also called the *cosine similarity*, is given as

$$\cos\theta = \frac{\mathbf{a}^T\mathbf{b}}{\|\mathbf{a}\|\,\|\mathbf{b}\|} = \left(\frac{\mathbf{a}}{\|\mathbf{a}\|}\right)^T \left(\frac{\mathbf{b}}{\|\mathbf{b}\|}\right) \tag{1.3}$$

Thus, the cosine of the angle between $\mathbf{a}$ and $\mathbf{b}$ is given as the dot product of the unit vectors $\frac{\mathbf{a}}{\|\mathbf{a}\|}$ and $\frac{\mathbf{b}}{\|\mathbf{b}\|}$.

The *Cauchy–Schwartz* inequality states that for any vectors $\mathbf{a}$ and $\mathbf{b}$ in $\mathbb{R}^m$

$$|\mathbf{a}^T\mathbf{b}| \leq \|\mathbf{a}\| \cdot \|\mathbf{b}\|$$

It follows immediately from the Cauchy–Schwartz inequality that

$$-1 \leq \cos\theta \leq 1$$

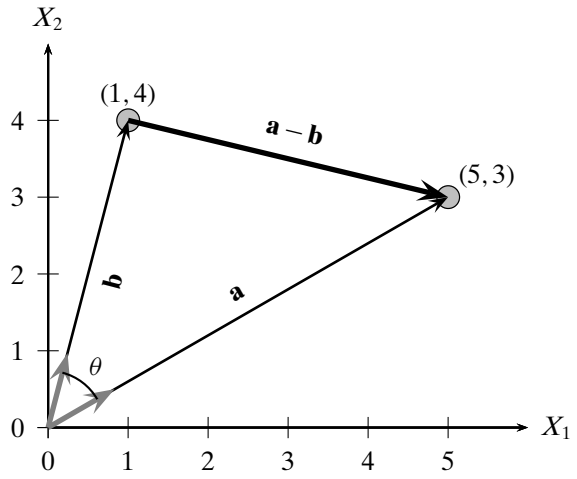


Figure 1.3. Distance and angle. Unit vectors are shown in gray.

Because the smallest angle $\theta \in [0^\circ, 180^\circ]$ and because $\cos\theta \in [-1, 1]$, the cosine similarity value ranges from $+1$, corresponding to an angle of 0°, to -1, corresponding to an angle of 180° (or π radians).

Orthogonality

Two vectors $\mathbf{a}$ and $\mathbf{b}$ are said to be *orthogonal* if and only if $\mathbf{a}^T\mathbf{b} = 0$, which in turn implies that $\cos\theta = 0$, that is, the angle between them is 90° or $\frac{\pi}{2}$ radians. In this case, we say that they have no similarity.

Example 1.3 (Distance and Angle). Figure 1.3 shows the two vectors

$$\mathbf{a} = \begin{pmatrix} 5 \\ 3 \end{pmatrix} \text{ and } \mathbf{b} = \begin{pmatrix} 1 \\ 4 \end{pmatrix}$$

Using Eq. (1.1), the Euclidean distance between them is given as

$$\delta(\mathbf{a}, \mathbf{b}) = \sqrt{(5-1)^2 + (3-4)^2} = \sqrt{16+1} = \sqrt{17} = 4.12$$

The distance can also be computed as the magnitude of the vector:

$$\mathbf{a} - \mathbf{b} = \begin{pmatrix} 5 \\ 3 \end{pmatrix} - \begin{pmatrix} 1 \\ 4 \end{pmatrix} = \begin{pmatrix} 4 \\ -1 \end{pmatrix}$$

because $\|\mathbf{a} - \mathbf{b}\| = \sqrt{4^2 + (-1)^2} = \sqrt{17} = 4.12$.

The unit vector in the direction of $\mathbf{a}$ is given as

$$\mathbf{u}_a = \frac{\mathbf{a}}{\|\mathbf{a}\|} = \frac{1}{\sqrt{5^2+3^2}} \begin{pmatrix} 5 \\ 3 \end{pmatrix} = \frac{1}{\sqrt{34}} \begin{pmatrix} 5 \\ 3 \end{pmatrix} = \begin{pmatrix} 0.86 \\ 0.51 \end{pmatrix}$$

The unit vector in the direction of **b** can be computed similarly:

$$\mathbf{u}_b = \begin{pmatrix} 0.24 \\ 0.97 \end{pmatrix}$$

These unit vectors are also shown in gray in Figure 1.3.

By Eq. (1.3) the cosine of the angle between **a** and **b** is given as

$$\cos\theta = \frac{\begin{pmatrix} 5 \\ 3 \end{pmatrix}^T \begin{pmatrix} 1 \\ 4 \end{pmatrix}}{\sqrt{5^2+3^2}\sqrt{1^2+4^2}} = \frac{17}{\sqrt{34 \times 17}} = \frac{1}{\sqrt{2}}$$

We can get the angle by computing the inverse of the cosine:

$$\theta = \cos^{-1}\left(1/\sqrt{2}\right) = 45^\circ$$

Let us consider the L_p-norm for **a** with $p = 3$; we get

$$\|\mathbf{a}\|_3 = \left(5^3 + 3^3\right)^{1/3} = (153)^{1/3} = 5.34$$

The distance between **a** and **b** using Eq. (1.2) for the L_p-norm with $p = 3$ is given as

$$\|\mathbf{a} - \mathbf{b}\|_3 = \left\|(4,-1)^T\right\|_3 = \left(4^3 + (-1)^3\right)^{1/3} = (63)^{1/3} = 3.98$$

1.3.2 Mean and Total Variance

Mean

The *mean* of the data matrix **D** is the vector obtained as the average of all the points:

$$mean(\mathbf{D}) = \boldsymbol{\mu} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i$$

Total Variance

The *total variance* of the data matrix **D** is the average squared distance of each point from the mean:

$$var(\mathbf{D}) = \frac{1}{n}\sum_{i=1}^{n} \delta(\mathbf{x}_i, \boldsymbol{\mu})^2 = \frac{1}{n}\sum_{i=1}^{n} \|\mathbf{x}_i - \boldsymbol{\mu}\|^2 \tag{1.4}$$

Simplifying Eq. (1.4) we obtain

$$\begin{aligned} var(\mathbf{D}) &= \frac{1}{n}\sum_{i=1}^{n} \left(\|\mathbf{x}_i\|^2 - 2\mathbf{x}_i^T\boldsymbol{\mu} + \|\boldsymbol{\mu}\|^2\right) \\ &= \frac{1}{n}\left(\sum_{i=1}^{n} \|\mathbf{x}_i\|^2 - 2n\boldsymbol{\mu}^T\left(\frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i\right) + n\|\boldsymbol{\mu}\|^2\right) \end{aligned}$$

$$= \frac{1}{n}\left(\sum_{i=1}^{n} \|\mathbf{x}_i\|^2 - 2n\boldsymbol{\mu}^T\boldsymbol{\mu} + n\|\boldsymbol{\mu}\|^2\right)$$
$$= \frac{1}{n}\left(\sum_{i=1}^{n} \|\mathbf{x}_i\|^2\right) - \|\boldsymbol{\mu}\|^2$$

The total variance is thus the difference between the average of the squared magnitude of the data points and the squared magnitude of the mean (average of the points).

Centered Data Matrix

Often we need to center the data matrix by making the mean coincide with the origin of the data space. The *centered data matrix* is obtained by subtracting the mean from all the points:

$$\mathbf{Z} = \mathbf{D} - \mathbf{1}\cdot\boldsymbol{\mu}^T = \begin{pmatrix}\mathbf{x}_1^T\\ \mathbf{x}_2^T\\ \vdots\\ \mathbf{x}_n^T\end{pmatrix} - \begin{pmatrix}\boldsymbol{\mu}^T\\ \boldsymbol{\mu}^T\\ \vdots\\ \boldsymbol{\mu}^T\end{pmatrix} = \begin{pmatrix}\mathbf{x}_1^T - \boldsymbol{\mu}^T\\ \mathbf{x}_2^T - \boldsymbol{\mu}^T\\ \vdots\\ \mathbf{x}_n^T - \boldsymbol{\mu}^T\end{pmatrix} = \begin{pmatrix}\mathbf{z}_1^T\\ \mathbf{z}_2^T\\ \vdots\\ \mathbf{z}_n^T\end{pmatrix} \tag{1.5}$$

where $\mathbf{z}_i = \mathbf{x}_i - \boldsymbol{\mu}$ represents the centered point corresponding to $\mathbf{x}_i$, and $\mathbf{1} \in \mathbb{R}^n$ is the n-dimensional vector all of whose elements have value 1. The mean of the centered data matrix $\mathbf{Z}$ is $\mathbf{0} \in \mathbb{R}^d$, because we have subtracted the mean $\boldsymbol{\mu}$ from all the points $\mathbf{x}_i$.

1.3.3 Orthogonal Projection

Often in data mining we need to project a point or vector onto another vector, for example, to obtain a new point after a change of the basis vectors. Let $\mathbf{a}, \mathbf{b} \in \mathbb{R}^m$ be two m-dimensional vectors. An *orthogonal decomposition* of the vector $\mathbf{b}$ in the direction

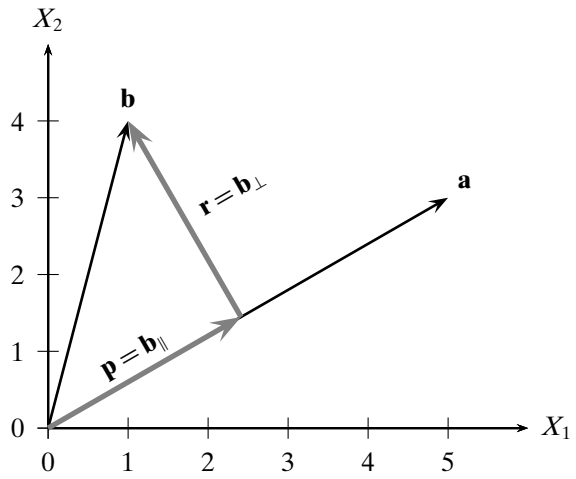


Figure 1.4. Orthogonal projection.

of another vector $\mathbf{a}$, illustrated in Figure 1.4, is given as

$$\mathbf{b} = \mathbf{b}_{\parallel} + \mathbf{b}_{\perp} = \mathbf{p} + \mathbf{r} \tag{1.6}$$

where $\mathbf{p} = \mathbf{b}_{\parallel}$ is parallel to $\mathbf{a}$, and $\mathbf{r} = \mathbf{b}_{\perp}$ is perpendicular or orthogonal to $\mathbf{a}$. The vector $\mathbf{p}$ is called the *orthogonal projection* or simply projection of $\mathbf{b}$ on the vector $\mathbf{a}$. Note that the point $\mathbf{p} \in \mathbb{R}^m$ is the point closest to $\mathbf{b}$ on the line passing through $\mathbf{a}$. Thus, the magnitude of the vector $\mathbf{r} = \mathbf{b} - \mathbf{p}$ gives the *perpendicular distance* between $\mathbf{b}$ and $\mathbf{a}$, which is often interpreted as the residual or error vector between the points $\mathbf{b}$ and $\mathbf{p}$.

We can derive an expression for $\mathbf{p}$ by noting that $\mathbf{p} = c\mathbf{a}$ for some scalar c, as $\mathbf{p}$ is parallel to $\mathbf{a}$. Thus, $\mathbf{r} = \mathbf{b} - \mathbf{p} = \mathbf{b} - c\mathbf{a}$. Because $\mathbf{p}$ and $\mathbf{r}$ are orthogonal, we have

$$\mathbf{p}^T\mathbf{r} = (c\mathbf{a})^T(\mathbf{b} - c\mathbf{a}) = c\mathbf{a}^T\mathbf{b} - c^2\mathbf{a}^T\mathbf{a} = 0$$

which implies that

$$c = \frac{\mathbf{a}^T\mathbf{b}}{\mathbf{a}^T\mathbf{a}}$$

Therefore, the projection of $\mathbf{b}$ on $\mathbf{a}$ is given as

$$\mathbf{p} = \mathbf{b}_{\parallel} = c\mathbf{a} = \left(\frac{\mathbf{a}^T\mathbf{b}}{\mathbf{a}^T\mathbf{a}}\right)\mathbf{a} \tag{1.7}$$

Example 1.4. Restricting the Iris dataset to the first two dimensions, `sepal length` and `sepal width`, the mean point is given as

$$mean(\mathbf{D}) = \begin{pmatrix} 5.843 \\ 3.054 \end{pmatrix}$$

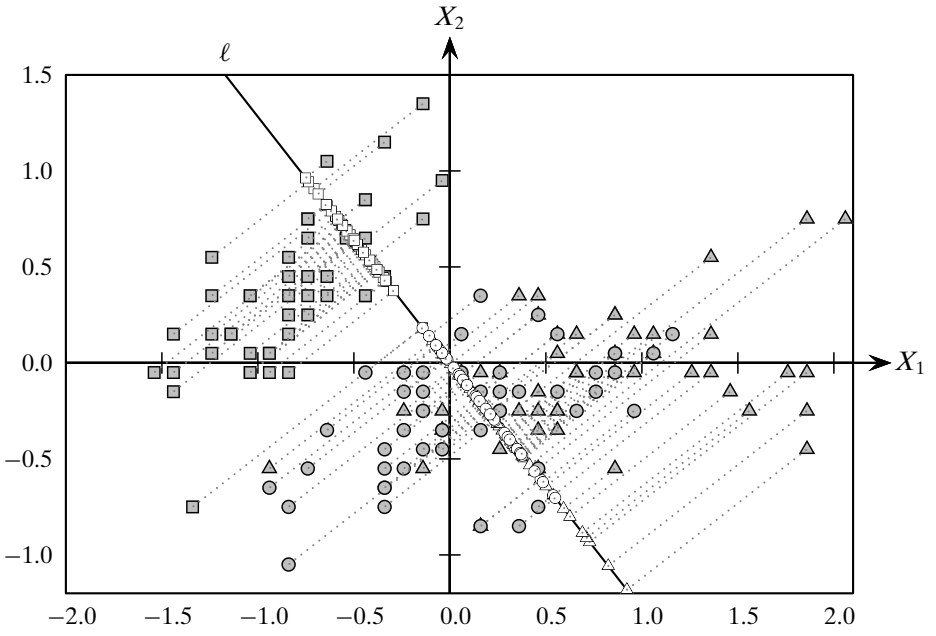


Figure 1.5. Projecting the centered data onto the line ℓ.

which is shown as the black circle in Figure 1.2. The corresponding centered data is shown in Figure 1.5, and the total variance is $var(\mathbf{D}) = 0.868$ (centering does not change this value).

Figure 1.5 shows the projection of each point onto the line ℓ, which is the line that maximizes the separation between the class `iris-setosa` (squares) from the other two classes, namely `iris-versicolor` (circles) and `iris-virginica` (triangles). The line ℓ is given as the set of all the points $(x_1, x_2)^T$ satisfying the constraint $\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = c \begin{pmatrix} -2.15 \\ 2.75 \end{pmatrix}$ for all scalars $c \in \mathbb{R}$.

1.3.4 Linear Independence and Dimensionality

Given the data matrix

$$\mathbf{D} = \begin{pmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_n \end{pmatrix}^T = \begin{pmatrix} X_1 & X_2 & \cdots & X_d \end{pmatrix}$$

we are often interested in the linear combinations of the rows (points) or the columns (attributes). For instance, different linear combinations of the original d attributes yield new derived attributes, which play a key role in feature extraction and dimensionality reduction.

Given any set of vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$ in an m-dimensional vector space $\mathbb{R}^m$, their *linear combination* is given as

$$c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + \cdots + c_k\mathbf{v}_k$$

where $c_i \in \mathbb{R}$ are scalar values. The set of all possible linear combinations of the k vectors is called the *span*, denoted as $span(\mathbf{v}_1, \ldots, \mathbf{v}_k)$, which is itself a vector space being a *subspace* of $\mathbb{R}^m$. If $span(\mathbf{v}_1, \ldots, \mathbf{v}_k) = \mathbb{R}^m$, then we say that $\mathbf{v}_1, \ldots, \mathbf{v}_k$ is a *spanning set* for $\mathbb{R}^m$.

Row and Column Space

There are several interesting vector spaces associated with the data matrix $\mathbf{D}$, two of which are the column space and row space of $\mathbf{D}$. The *column space* of $\mathbf{D}$, denoted $col(\mathbf{D})$, is the set of all linear combinations of the d attributes $X_j \in \mathbb{R}^n$, that is,

$$col(\mathbf{D}) = span(X_1, X_2, \ldots, X_d)$$

By definition $col(\mathbf{D})$ is a subspace of $\mathbb{R}^n$. The *row space* of $\mathbf{D}$, denoted $row(\mathbf{D})$, is the set of all linear combinations of the n points $\mathbf{x}_i \in \mathbb{R}^d$, that is,

$$row(\mathbf{D}) = span(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$$

By definition $row(\mathbf{D})$ is a subspace of $\mathbb{R}^d$. Note also that the row space of $\mathbf{D}$ is the column space of $\mathbf{D}^T$:

$$row(\mathbf{D}) = col(\mathbf{D}^T)$$

Linear Independence

We say that the vectors $\mathbf{v}_1, \ldots, \mathbf{v}_k$ are *linearly dependent* if at least one vector can be written as a linear combination of the others. Alternatively, the k vectors are linearly dependent if there are scalars $c_1, c_2, \ldots, c_k$, at least one of which is not zero, such that

$$c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + \cdots + c_k\mathbf{v}_k = \mathbf{0}$$

On the other hand, $\mathbf{v}_1, \cdots, \mathbf{v}_k$ are *linearly independent* if and only if

$$c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + \cdots + c_k\mathbf{v}_k = \mathbf{0} \text{ implies } c_1 = c_2 = \cdots = c_k = 0$$

Simply put, a set of vectors is linearly independent if none of them can be written as a linear combination of the other vectors in the set.

Dimension and Rank

Let S be a subspace of $\mathbb{R}^m$. A *basis* for S is a set of vectors in S, say $\mathbf{v}_1, \ldots, \mathbf{v}_k$, that are linearly independent and they span S, that is, $span(\mathbf{v}_1, \ldots, \mathbf{v}_k) = S$. In fact, a basis is a minimal spanning set. If the vectors in the basis are pairwise orthogonal, they are said to form an *orthogonal basis* for S. If, in addition, they are also normalized to be unit vectors, then they make up an *orthonormal basis* for S. For instance, the *standard basis* for $\mathbb{R}^m$ is an orthonormal basis consisting of the vectors

$$\mathbf{e}_1 = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \qquad \mathbf{e}_2 = \begin{pmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix} \qquad \cdots \qquad \mathbf{e}_m = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix}$$

Any two bases for S must have the same number of vectors, and the number of vectors in a basis for S is called the *dimension* of S, denoted as $dim(S)$. Because S is a subspace of $\mathbb{R}^m$, we must have $dim(S) \leq m$.

It is a remarkable fact that, for any matrix, the dimension of its row and column space is the same, and this dimension is also called the *rank* of the matrix. For the data matrix $\mathbf{D} \in \mathbb{R}^{n \times d}$, we have $rank(\mathbf{D}) \leq \min(n, d)$, which follows from the fact that the column space can have dimension at most d, and the row space can have dimension at most n. Thus, even though the data points are ostensibly in a d dimensional attribute space (the *extrinsic dimensionality*), if $rank(\mathbf{D}) < d$, then the data points reside in a lower dimensional subspace of $\mathbb{R}^d$, and in this case $rank(\mathbf{D})$ gives an indication about the *intrinsic* dimensionality of the data. In fact, with dimensionality reduction methods it is often possible to approximate $\mathbf{D} \in \mathbb{R}^{n \times d}$ with a derived data matrix $\mathbf{D}' \in \mathbb{R}^{n \times k}$, which has much lower dimensionality, that is, $k \ll d$. In this case k may reflect the "true" intrinsic dimensionality of the data.

Example 1.5. The line ℓ in Figure 1.5 is given as $\ell = span\left((-2.15 \quad 2.75)^T\right)$, with $dim(\ell) = 1$. After normalization, we obtain the orthonormal basis for ℓ as the unit vector

$$\frac{1}{\sqrt{12.19}}\begin{pmatrix} -2.15 \\ 2.75 \end{pmatrix} = \begin{pmatrix} -0.615 \\ 0.788 \end{pmatrix}$$

Table 1.2. Iris dataset: `sepal length` (in centimeters).

5.9	6.9	6.6	4.6	6.0	4.7	6.5	5.8	6.7	6.7	5.1	5.1	5.7	6.1	4.9
5.0	5.0	5.7	5.0	7.2	5.9	6.5	5.7	5.5	4.9	5.0	5.5	4.6	7.2	6.8
5.4	5.0	5.7	5.8	5.1	5.6	5.8	5.1	6.3	6.3	5.6	6.1	6.8	7.3	5.6
4.8	7.1	5.7	5.3	5.7	5.7	5.6	4.4	6.3	5.4	6.3	6.9	7.7	6.1	5.6
6.1	6.4	5.0	5.1	5.6	5.4	5.8	4.9	4.6	5.2	7.9	7.7	6.1	5.5	4.6
4.7	4.4	6.2	4.8	6.0	6.2	5.0	6.4	6.3	6.7	5.0	5.9	6.7	5.4	6.3
4.8	4.4	6.4	6.2	6.0	7.4	4.9	7.0	5.5	6.3	6.8	6.1	6.5	6.7	6.7
4.8	4.9	6.9	4.5	4.3	5.2	5.0	6.4	5.2	5.8	5.5	7.6	6.3	6.4	6.3
5.8	5.0	6.7	6.0	5.1	4.8	5.7	5.1	6.6	6.4	5.2	6.4	7.7	5.8	4.9
5.4	5.1	6.0	6.5	5.5	7.2	6.9	6.2	6.5	6.0	5.4	5.5	6.7	7.7	5.1

1.4 DATA: PROBABILISTIC VIEW

The probabilistic view of the data assumes that each numeric attribute X is a *random variable*, defined as a function that assigns a real number to each outcome of an experiment (i.e., some process of observation or measurement). Formally, X is a function $X: \mathcal{O} \to \mathbb{R}$, where $\mathcal{O}$, the domain of X, is the set of all possible outcomes of the experiment, also called the *sample space*, and $\mathbb{R}$, the *range* of X, is the set of real numbers. If the outcomes are numeric, and represent the observed values of the random variable, then $X: \mathcal{O} \to \mathcal{O}$ is simply the identity function: $X(v) = v$ for all $v \in \mathcal{O}$. The distinction between the outcomes and the value of the random variable is important, as we may want to treat the observed values differently depending on the context, as seen in Example 1.6.

A random variable X is called a *discrete random variable* if it takes on only a finite or countably infinite number of values in its range, whereas X is called a *continuous random variable* if it can take on any value in its range.

Example 1.6. Consider the `sepal length` attribute (X_1) for the Iris dataset in Table 1.1. All $n = 150$ values of this attribute are shown in Table 1.2, which lie in the range $[4.3, 7.9]$, with centimeters as the unit of measurement. Let us assume that these constitute the set of all possible outcomes $\mathcal{O}$.

By default, we can consider the attribute X_1 to be a continuous random variable, given as the identity function $X_1(v) = v$, because the outcomes (sepal length values) are all numeric.

On the other hand, if we want to distinguish between Iris flowers with short and long sepal lengths, with long being, say, a length of 7 cm or more, we can define a discrete random variable A as follows:

$$A(v) = \begin{cases} 0 & \text{if } v < 7 \\ 1 & \text{if } v \geq 7 \end{cases}$$

In this case the domain of A is $[4.3, 7.9]$, and its range is $\{0, 1\}$. Thus, A assumes nonzero probability only at the discrete values 0 and 1.

Probability Mass Function

If X is discrete, the *probability mass function* of X is defined as

$$f(x) = P(X = x) \qquad \text{for all } x \in \mathbb{R}$$

In other words, the function f gives the probability $P(X = x)$ that the random variable X has the exact value x. The name "probability mass function" intuitively conveys the fact that the probability is concentrated or massed at only discrete values in the range of X, and is zero for all other values. f must also obey the basic rules of probability. That is, f must be non-negative:

$$f(x) \geq 0$$

and the sum of all probabilities should add to 1:

$$\sum_x f(x) = 1$$

Example 1.7 (Bernoulli and Binomial Distribution). In Example 1.6, A was defined as a discrete random variable representing long sepal length. From the sepal length data in Table 1.2 we find that only 13 Irises have sepal length of at least 7 cm. We can thus estimate the probability mass function of A as follows:

$$f(1) = P(A = 1) = \frac{13}{150} = 0.087 = p$$

and

$$f(0) = P(A = 0) = \frac{137}{150} = 0.913 = 1 - p$$

In this case we say that A has a *Bernoulli distribution* with parameter $p \in [0, 1]$, which denotes the probability of a *success*, that is, the probability of picking an Iris with a long sepal length at random from the set of all points. On the other hand, $1 - p$ is the probability of a *failure*, that is, of not picking an Iris with long sepal length.

Let us consider another discrete random variable B, denoting the number of Irises with long sepal length in m independent Bernoulli trials with probability of success p. In this case, B takes on the discrete values $[0, m]$, and its probability mass function is given by the *Binomial distribution*

$$f(k) = P(B = k) = \binom{m}{k} p^k (1 - p)^{m-k}$$

The formula can be understood as follows. There are $\binom{m}{k}$ ways of picking k long sepal length Irises out of the m trials. For each selection of k long sepal length Irises, the total probability of the k successes is p^k, and the total probability of $m - k$ failures is $(1 - p)^{m-k}$. For example, because $p = 0.087$ from above, the probability of observing exactly $k = 2$ Irises with long sepal length in $m = 10$ trials is given as

$$f(2) = P(B = 2) = \binom{10}{2} (0.087)^2 (0.913)^8 = 0.164$$

Figure 1.6 shows the full probability mass function for different values of k for $m = 10$. Because p is quite small, the probability of k successes in so few a trials falls off rapidly as k increases, becoming practically zero for values of $k \geq 6$.

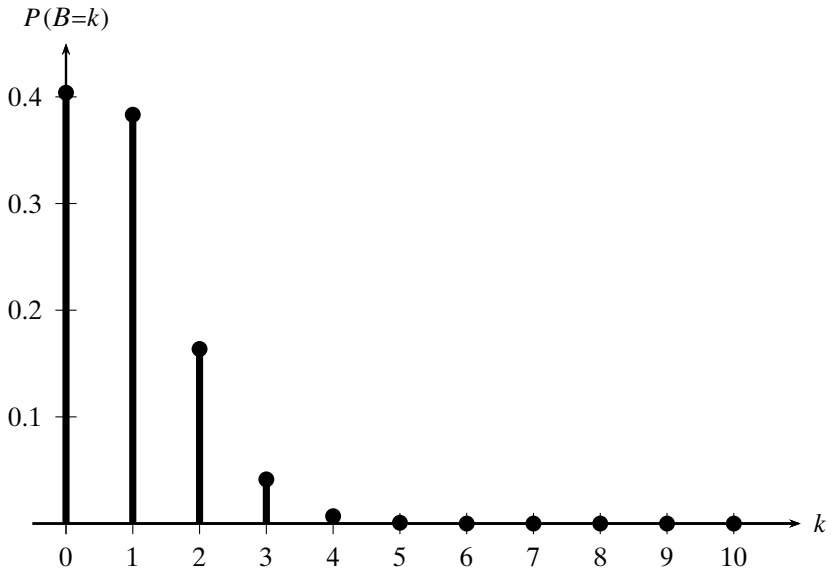


Figure 1.6. Binomial distribution: probability mass function ($m = 10, p = 0.087$).

Probability Density Function

If X is continuous, its range is the entire set of real numbers $\mathbb{R}$. The probability of any specific value x is only one out of the infinitely many possible values in the range of X, which means that $P(X = x) = 0$ for all $x \in \mathbb{R}$. However, this does not mean that the value x is impossible, because in that case we would conclude that all values are impossible! What it means is that the probability mass is spread so thinly over the range of values that it can be measured only over intervals $[a, b] \subset \mathbb{R}$, rather than at specific points. Thus, instead of the probability mass function, we define the *probability density function*, which specifies the probability that the variable X takes on values in any interval $[a, b] \subset \mathbb{R}$:

$$P\big(X \in [a, b]\big) = \int_a^b f(x)\, dx$$

As before, the density function f must satisfy the basic laws of probability:

$$f(x) \geq 0, \qquad \text{for all } x \in \mathbb{R}$$

and

$$\int_{-\infty}^{\infty} f(x)\, dx = 1$$

We can get an intuitive understanding of the density function f by considering the probability density over a small interval of width $2\epsilon > 0$, centered at x, namely

$[x-\epsilon, x+\epsilon]$:

$$P\big(X \in [x-\epsilon, x+\epsilon]\big) = \int_{x-\epsilon}^{x+\epsilon} f(x)\, dx \simeq 2\epsilon \cdot f(x)$$

$$f(x) \simeq \frac{P\big(X \in [x-\epsilon, x+\epsilon]\big)}{2\epsilon} \tag{1.8}$$

$f(x)$ thus gives the probability density at x, given as the ratio of the probability mass to the width of the interval, that is, the probability mass per unit distance. Thus, it is important to note that $P(X=x) \neq f(x)$.

Even though the probability density function $f(x)$ does not specify the probability $P(X=x)$, it can be used to obtain the relative probability of one value x_1 over another x_2 because for a given $\epsilon > 0$, by Eq. (1.8), we have

$$\frac{P(X \in [x_1-\epsilon, x_1+\epsilon])}{P(X \in [x_2-\epsilon, x_2+\epsilon])} \simeq \frac{2\epsilon \cdot f(x_1)}{2\epsilon \cdot f(x_2)} = \frac{f(x_1)}{f(x_2)} \tag{1.9}$$

Thus, if $f(x_1)$ is larger than $f(x_2)$, then values of X close to x_1 are more probable than values close to x_2, and vice versa.

Example 1.8 (Normal Distribution). Consider again the `sepal length` values from the Iris dataset, as shown in Table 1.2. Let us assume that these values follow a *Gaussian* or *normal* density function, given as

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{\frac{-(x-\mu)^2}{2\sigma^2}\right\}$$

There are two parameters of the normal density distribution, namely, μ, which represents the mean value, and σ^2, which represents the variance of the values (these parameters are discussed in Chapter 2). Figure 1.7 shows the characteristic "bell" shape plot of the normal distribution. The parameters, $\mu = 5.84$ and $\sigma^2 = 0.681$, were estimated directly from the data for `sepal length` in Table 1.2.

Whereas $f(x=\mu) = f(5.84) = \frac{1}{\sqrt{2\pi \cdot 0.681}} \exp\{0\} = 0.483$, we emphasize that the probability of observing $X = \mu$ is zero, that is, $P(X=\mu) = 0$. Thus, $P(X=x)$ is not given by $f(x)$, rather, $P(X=x)$ is given as the area under the curve for an infinitesimally small interval $[x-\epsilon, x+\epsilon]$ centered at x, with $\epsilon > 0$. Figure 1.7 illustrates this with the shaded region centered at $\mu = 5.84$. From Eq. (1.8), we have

$$P(X=\mu) \simeq 2\epsilon \cdot f(\mu) = 2\epsilon \cdot 0.483 = 0.967\epsilon$$

As $\epsilon \to 0$, we get $P(X=\mu) \to 0$. However, based on Eq. (1.9) we can claim that the probability of observing values close to the mean value $\mu = 5.84$ is 2.69 times the probability of observing values close to $x = 7$, as

$$\frac{f(5.84)}{f(7)} = \frac{0.483}{0.18} = 2.69$$

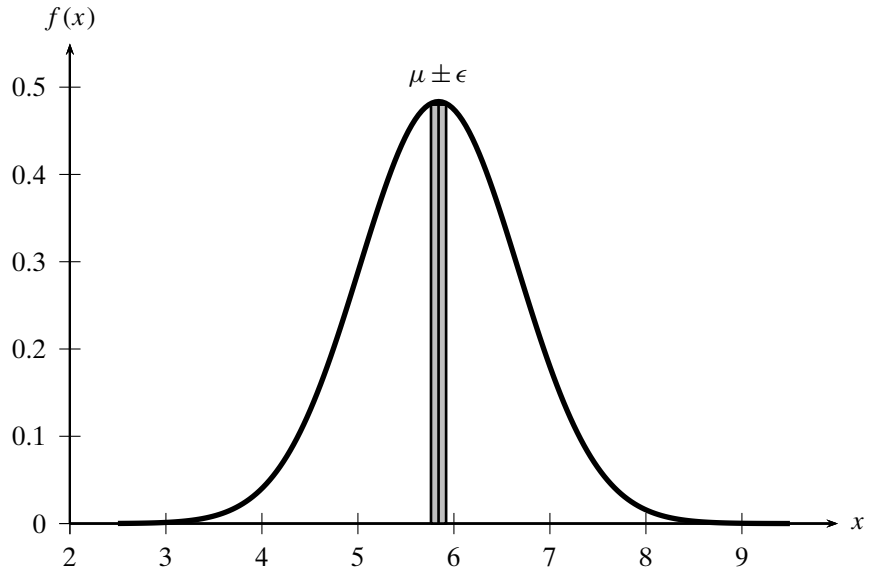


Figure 1.7. Normal distribution: probability density function ($\mu = 5.84$, $\sigma^2 = 0.681$).

Cumulative Distribution Function

For any random variable X, whether discrete or continuous, we can define the *cumulative distribution function (CDF)* $F : \mathbb{R} \to [0, 1]$, which gives the probability of observing a value at most some given value x:

$$F(x) = P(X \leq x) \qquad \text{for all } -\infty < x < \infty$$

When X is discrete, F is given as

$$F(x) = P(X \leq x) = \sum_{u \leq x} f(u)$$

and when X is continuous, F is given as

$$F(x) = P(X \leq x) = \int_{-\infty}^{x} f(u)\, du$$

Example 1.9 (Cumulative Distribution Function). Figure 1.8 shows the cumulative distribution function for the binomial distribution in Figure 1.6. It has the characteristic step shape (right continuous, non-decreasing), as expected for a discrete random variable. $F(x)$ has the same value $F(k)$ for all $x \in [k, k+1)$ with $0 \leq k < m$, where m is the number of trials and k is the number of successes. The closed (filled) and open circles demarcate the corresponding closed and open interval $[k, k+1)$. For instance, $F(x) = 0.404 = F(0)$ for all $x \in [0, 1)$.

Figure 1.9 shows the cumulative distribution function for the normal density function shown in Figure 1.7. As expected, for a continuous random variable, the CDF is also continuous, and non-decreasing. Because the normal distribution is symmetric about the mean, we have $F(\mu) = P(X \leq \mu) = 0.5$.

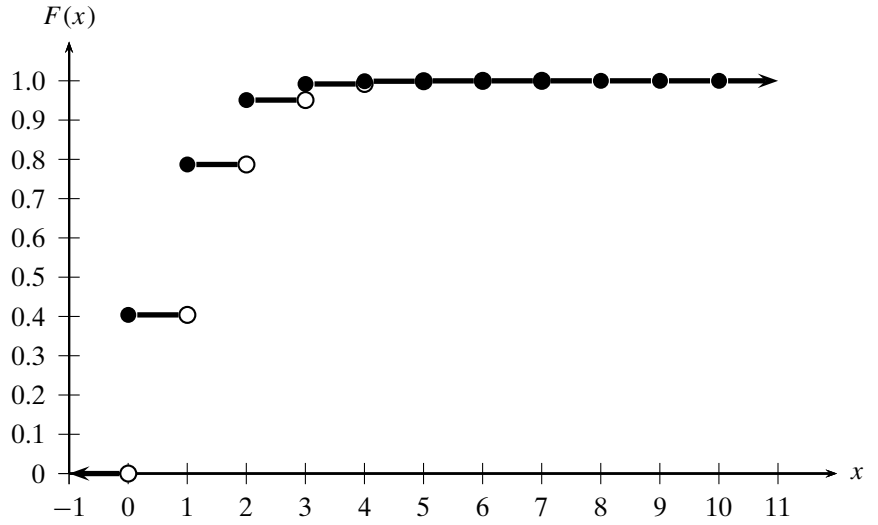


Figure 1.8. Cumulative distribution function for the binomial distribution.

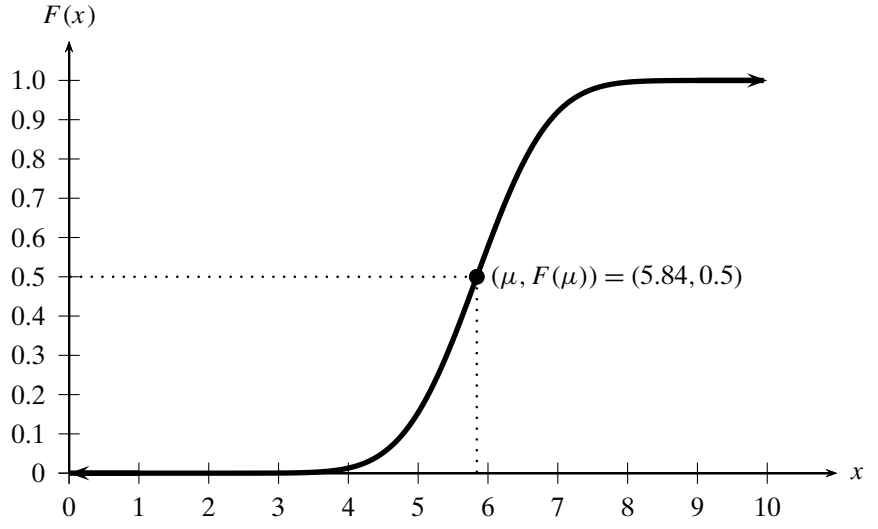


Figure 1.9. Cumulative distribution function for the normal distribution.

1.4.1 Bivariate Random Variables

Instead of considering each attribute as a random variable, we can also perform pair-wise analysis by considering a pair of attributes, X_1 and X_2, as a *bivariate random variable*:

$$\mathbf{X} = \begin{pmatrix} X_1 \\ X_2 \end{pmatrix}$$

$\mathbf{X} : \mathcal{O} \to \mathbb{R}^2$ is a function that assigns to each outcome in the sample space, a pair of real numbers, that is, a 2-dimensional vector $\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \in \mathbb{R}^2$. As in the univariate case,

if the outcomes are numeric, then the default is to assume $\mathbf{X}$ to be the identity function.

Joint Probability Mass Function

If X_1 and X_2 are both discrete random variables then $\mathbf{X}$ has a *joint probability mass function* given as follows:

$$f(\mathbf{x}) = f(x_1, x_2) = P(X_1 = x_1, X_2 = x_2) = P(\mathbf{X} = \mathbf{x})$$

f must satisfy the following two conditions:

$$f(\mathbf{x}) = f(x_1, x_2) \geq 0 \qquad \text{for all } -\infty < x_1, x_2 < \infty$$

$$\sum_{\mathbf{x}} f(\mathbf{x}) = \sum_{x_1} \sum_{x_2} f(x_1, x_2) = 1$$

Joint Probability Density Function

If X_1 and X_2 are both continuous random variables then $\mathbf{X}$ has a *joint probability density function* f given as follows:

$$P(\mathbf{X} \in W) = \int_{\mathbf{x} \in W} \int f(\mathbf{x})\, d\mathbf{x} = \int_{(x_1, x_2)^T \in W} \int f(x_1, x_2)\, dx_1\, dx_2$$

where $W \subset \mathbb{R}^2$ is some subset of the 2-dimensional space of reals. f must also satisfy the following two conditions:

$$f(\mathbf{x}) = f(x_1, x_2) \geq 0 \qquad \text{for all } -\infty < x_1, x_2 < \infty$$

$$\int_{\mathbb{R}^2} f(\mathbf{x})\, d\mathbf{x} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x_1, x_2)\, dx_1\, dx_2 = 1$$

As in the univariate case, the probability mass $P(\mathbf{x}) = P\big((x_1, x_2)^T\big) = 0$ for any particular point $\mathbf{x}$. However, we can use f to compute the probability density at $\mathbf{x}$. Consider the square region $W = \big([x_1 - \epsilon, x_1 + \epsilon], [x_2 - \epsilon, x_2 + \epsilon]\big)$, that is, a 2-dimensional window of width 2ϵ centered at $\mathbf{x} = (x_1, x_2)^T$. The probability density at $\mathbf{x}$ can be approximated as

$$\begin{aligned} P(\mathbf{X} \in W) &= P\Big(\mathbf{X} \in \big([x_1 - \epsilon, x_1 + \epsilon], [x_2 - \epsilon, x_2 + \epsilon]\big)\Big) \\ &= \int_{x_1 - \epsilon}^{x_1 + \epsilon} \int_{x_2 - \epsilon}^{x_2 + \epsilon} f(x_1, x_2)\, dx_1\, dx_2 \\ &\simeq 2\epsilon \cdot 2\epsilon \cdot f(x_1, x_2) \end{aligned}$$

which implies that

$$f(x_1, x_2) = \frac{P(\mathbf{X} \in W)}{(2\epsilon)^2}$$

The relative probability of one value (a_1, a_2) versus another (b_1, b_2) can therefore be computed via the probability density function:

$$\frac{P(\mathbf{X} \in \big([a_1 - \epsilon, a_1 + \epsilon], [a_2 - \epsilon, a_2 + \epsilon]\big))}{P(\mathbf{X} \in \big([b_1 - \epsilon, b_1 + \epsilon], [b_2 - \epsilon, b_2 + \epsilon]\big))} \simeq \frac{(2\epsilon)^2 \cdot f(a_1, a_2)}{(2\epsilon)^2 \cdot f(b_1, b_2)} = \frac{f(a_1, a_2)}{f(b_1, b_2)}$$

Example 1.10 (Bivariate Distributions). Consider the `sepal length` and `sepal width` attributes in the Iris dataset, plotted in Figure 1.2. Let A denote the Bernoulli random variable corresponding to long sepal length (at least 7 cm), as defined in Example 1.7.

Define another Bernoulli random variable B corresponding to long sepal width, say, at least 3.5 cm. Let $\mathbf{X} = \begin{pmatrix} A \\ B \end{pmatrix}$ be a discrete bivariate random variable; then the joint probability mass function of $\mathbf{X}$ can be estimated from the data as follows:

$$f(0,0) = P(A=0, B=0) = \frac{116}{150} = 0.773$$
$$f(0,1) = P(A=0, B=1) = \frac{21}{150} = 0.140$$
$$f(1,0) = P(A=1, B=0) = \frac{10}{150} = 0.067$$
$$f(1,1) = P(A=1, B=1) = \frac{3}{150} = 0.020$$

Figure 1.10 shows a plot of this probability mass function.

Treating attributes X_1 and X_2 in the Iris dataset (see Table 1.1) as continuous random variables, we can define a continuous bivariate random variable $\mathbf{X} = \begin{pmatrix} X_1 \\ X_2 \end{pmatrix}$. Assuming that $\mathbf{X}$ follows a *bivariate normal distribution*, its joint probability density function is given as

$$f(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{2\pi\sqrt{|\boldsymbol{\Sigma}|}} \exp\left\{-\frac{(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x}-\boldsymbol{\mu})}{2}\right\}$$

Here $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are the parameters of the bivariate normal distribution, representing the 2-dimensional mean vector and covariance matrix, which are discussed in detail

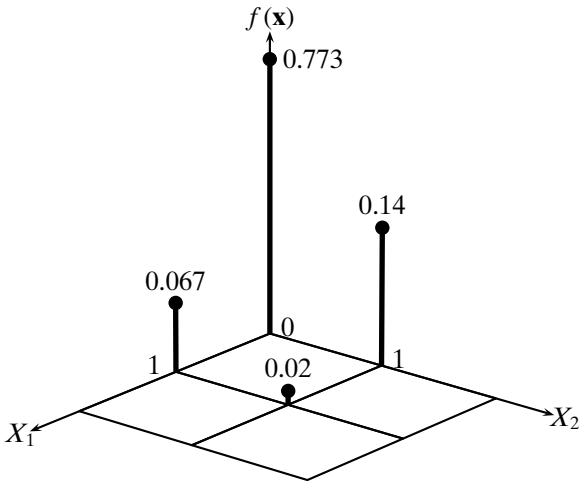


Figure 1.10. Joint probability mass function: X_1 (long sepal length), X_2 (long sepal width).

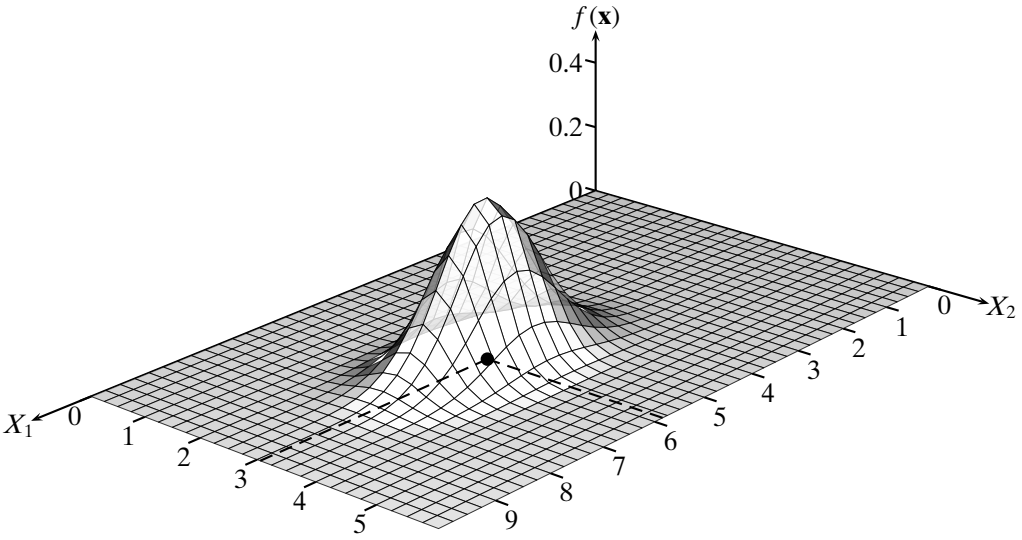


Figure 1.11. Bivariate normal density: $\boldsymbol{\mu} = (5.843, 3.054)^T$ (solid circle).

in Chapter 2. Further, $|\boldsymbol{\Sigma}|$ denotes the determinant of $\boldsymbol{\Sigma}$. The plot of the bivariate normal density is given in Figure 1.11, with mean

$$\boldsymbol{\mu} = (5.843, 3.054)^T$$

and covariance matrix

$$\boldsymbol{\Sigma} = \begin{pmatrix} 0.681 & -0.039 \\ -0.039 & 0.187 \end{pmatrix}$$

It is important to emphasize that the function $f(\mathbf{x})$ specifies only the probability density at $\mathbf{x}$, and $f(\mathbf{x}) \neq P(\mathbf{X} = \mathbf{x})$. As before, we have $P(\mathbf{X} = \mathbf{x}) = 0$.

Joint Cumulative Distribution Function

The *joint cumulative distribution function* for two random variables X_1 and X_2 is defined as the function F, such that for all values $x_1, x_2 \in (-\infty, \infty)$,

$$F(\mathbf{x}) = F(x_1, x_2) = P(X_1 \leq x_1 \text{ and } X_2 \leq x_2) = P(\mathbf{X} \leq \mathbf{x})$$

Statistical Independence

Two random variables X_1 and X_2 are said to be (statistically) *independent* if, for every $W_1 \subset \mathbb{R}$ and $W_2 \subset \mathbb{R}$, we have

$$P(X_1 \in W_1 \text{ and } X_2 \in W_2) = P(X_1 \in W_1) \cdot P(X_2 \in W_2)$$

Furthermore, if X_1 and X_2 are independent, then the following two conditions are also satisfied:

$$F(\mathbf{x}) = F(x_1, x_2) = F_1(x_1) \cdot F_2(x_2)$$

$$f(\mathbf{x}) = f(x_1, x_2) = f_1(x_1) \cdot f_2(x_2)$$

where F_i is the cumulative distribution function, and f_i is the probability mass or density function for random variable X_i.

1.4.2 Multivariate Random Variable

A d-dimensional *multivariate random variable* $\mathbf{X} = (X_1, X_2, \ldots, X_d)^T$, also called a *vector random variable*, is defined as a function that assigns a vector of real numbers to each outcome in the sample space, that is, $\mathbf{X} : \mathcal{O} \to \mathbb{R}^d$. The range of $\mathbf{X}$ can be denoted as a vector $\mathbf{x} = (x_1, x_2, \ldots, x_d)^T$. In case all X_j are numeric, then $\mathbf{X}$ is by default assumed to be the identity function. In other words, if all attributes are numeric, we can treat each outcome in the sample space (i.e., each point in the data matrix) as a vector random variable. On the other hand, if the attributes are not all numeric, then $\mathbf{X}$ maps the outcomes to numeric vectors in its range.

If all X_j are discrete, then $\mathbf{X}$ is jointly discrete and its joint probability mass function f is given as

$$f(\mathbf{x}) = P(\mathbf{X} = \mathbf{x})$$
$$f(x_1, x_2, \ldots, x_d) = P(X_1 = x_1, X_2 = x_2, \ldots, X_d = x_d)$$

If all X_j are continuous, then $\mathbf{X}$ is jointly continuous and its joint probability density function is given as

$$P(\mathbf{X} \in W) = \int_{\mathbf{x} \in W} \cdots \int f(\mathbf{x})\, d\mathbf{x}$$
$$P\left((X_1, X_2, \ldots, X_d)^T \in W\right) = \int_{(x_1, x_2, \ldots, x_d)^T \in W} \cdots \int f(x_1, x_2, \ldots, x_d)\, dx_1\, dx_2 \ldots dx_d$$

for any d-dimensional region $W \subseteq \mathbb{R}^d$.

The laws of probability must be obeyed as usual, that is, $f(\mathbf{x}) \geq 0$ and sum of f over all $\mathbf{x}$ in the range of $\mathbf{X}$ must be 1. The joint cumulative distribution function of $\mathbf{X} = (X_1, \ldots, X_d)^T$ is given as

$$F(\mathbf{x}) = P(\mathbf{X} \leq \mathbf{x})$$
$$F(x_1, x_2, \ldots, x_d) = P(X_1 \leq x_1, X_2 \leq x_2, \ldots, X_d \leq x_d)$$

for every point $\mathbf{x} \in \mathbb{R}^d$.

We say that $X_1, X_2, \ldots, X_d$ are independent random variables if and only if, for every region $W_i \subset \mathbb{R}$, we have

$$P(X_1 \in W_1 \text{ and } X_2 \in W_2 \cdots \text{ and } X_d \in W_d) = P(X_1 \in W_1) \cdot P(X_2 \in W_2) \cdot \cdots \cdot P(X_d \in W_d) \quad (1.10)$$

If $X_1, X_2, \ldots, X_d$ are independent then the following conditions are also satisfied

$$F(\mathbf{x}) = F(x_1, \ldots, x_d) = F_1(x_1) \cdot F_2(x_2) \cdot \ldots \cdot F_d(x_d)$$
$$f(\mathbf{x}) = f(x_1, \ldots, x_d) = f_1(x_1) \cdot f_2(x_2) \cdot \ldots \cdot f_d(x_d) \quad (1.11)$$

where F_i is the cumulative distribution function, and f_i is the probability mass or density function for random variable X_i.

1.4.3 Random Sample and Statistics

The probability mass or density function of a random variable X may follow some known form, or as is often the case in data analysis, it may be unknown. When the probability function is not known, it may still be convenient to assume that the values follow some known distribution, based on the characteristics of the data. However, even in this case, the parameters of the distribution may still be unknown. Thus, in general, either the parameters, or the entire distribution, may have to be estimated from the data.

In statistics, the word *population* is used to refer to the set or universe of all entities under study. Usually we are interested in certain characteristics or parameters of the entire population (e.g., the mean age of all computer science students in the United States). However, looking at the entire population may not be feasible or may be too expensive. Instead, we try to make inferences about the population parameters by drawing a random sample from the population, and by computing appropriate *statistics* from the sample that give estimates of the corresponding population parameters of interest.

Univariate Sample

Given a random variable X, a *random sample* of size n from X is defined as a set of n *independent and identically distributed (IID)* random variables $S_1, S_2, \ldots, S_n$, that is, all of the S_i's are statistically independent of each other, and follow the same probability mass or density function as X.

If we treat attribute X as a random variable, then each of the observed values of X, namely, x_i ($1 \leq i \leq n$), are themselves treated as identity random variables, and the observed data is assumed to be a random sample drawn from X. That is, all x_i are considered to be mutually independent and identically distributed as X. By Eq. (1.11) their joint probability function is given as

$$f(x_1, \ldots, x_n) = \prod_{i=1}^{n} f_X(x_i)$$

where $f_{\mathbf{X}}$ is the probability mass or density function for $\mathbf{X}$.

Multivariate Sample

For multivariate parameter estimation, the n data points $\mathbf{x}_i$ (with $1 \leq i \leq n$) constitute a d-dimensional multivariate random sample drawn from the vector random variable $\mathbf{X} = (X_1, X_2, \ldots, X_d)$. That is, $\mathbf{x}_i$ are assumed to be independent and identically distributed, and thus their joint distribution is given as

$$f(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n) = \prod_{i=1}^{n} f_{\mathbf{X}}(\mathbf{x}_i) \tag{1.12}$$

where $f_{\mathbf{X}}$ is the probability mass or density function for $\mathbf{X}$.

Estimating the parameters of a multivariate joint probability distribution is usually difficult and computationally intensive. One simplifying assumption that is typically made is that the d attributes $X_1, X_2, \ldots, X_d$ are statistically independent. However, we do not assume that they are identically distributed, because that is almost never justified. Under the attribute independence assumption Eq. (1.12) can be rewritten as

$$f(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n) = \prod_{i=1}^{n} f(\mathbf{x}_i) = \prod_{i=1}^{n} \prod_{j=1}^{d} f_{X_j}(x_{ij})$$

Statistic

We can estimate a parameter of the population by defining an appropriate sample *statistic*, which is defined as a function of the sample. More precisely, let $\{\mathbf{S}_i\}_{i=1}^{m}$ denote the random sample of size m drawn from a (multivariate) random variable $\mathbf{X}$. A statistic $\hat{\theta}$ is a function $\hat{\theta}: (\mathbf{S}_1, \mathbf{S}_2, \ldots, \mathbf{S}_m) \to \mathbb{R}$. The statistic is an estimate of the corresponding population parameter θ. As such, the statistic $\hat{\theta}$ is itself a random variable. If we use the value of a statistic to estimate a population parameter, this value is called a *point estimate* of the parameter, and the statistic is called an *estimator* of the parameter. In Chapter 2 we will study different estimators for population parameters that reflect the location (or centrality) and dispersion of values.

Example 1.11 (Sample Mean). Consider attribute `sepal length` (X_1) in the Iris dataset, whose values are shown in Table 1.2. Assume that the mean value of X_1 is not known. Let us assume that the observed values $\{x_i\}_{i=1}^{n}$ constitute a random sample drawn from X_1.

The *sample mean* is a statistic, defined as the average

$$\hat{\mu} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

Plugging in values from Table 1.2, we obtain

$$\hat{\mu} = \frac{1}{150}(5.9 + 6.9 + \cdots + 7.7 + 5.1) = \frac{876.5}{150} = 5.84$$

The value $\hat{\mu} = 5.84$ is a point estimate for the unknown population parameter μ, the (true) mean value of variable X_1.

1.5 DATA MINING

Data mining comprises the core algorithms that enable one to gain fundamental insights and knowledge from massive data. It is an interdisciplinary field merging concepts from allied areas such as database systems, statistics, machine learning, and pattern recognition. In fact, data mining is part of a larger knowledge discovery process, which includes pre-processing tasks such as data extraction, data cleaning, data fusion, data reduction and feature construction, as well as post-processing steps

such as pattern and model interpretation, hypothesis confirmation and generation, and so on. This knowledge discovery and data mining process tends to be highly iterative and interactive.

The algebraic, geometric, and probabilistic viewpoints of data play a key role in data mining. Given a dataset of n points in a d-dimensional space, the fundamental analysis and mining tasks covered in this book include exploratory data analysis, frequent pattern discovery, data clustering, and classification models, which are described next.

1.5.1 Exploratory Data Analysis

Exploratory data analysis aims to explore the numeric and categorical attributes of the data individually or jointly to extract key characteristics of the data sample via statistics that give information about the centrality, dispersion, and so on. Moving away from the IID assumption among the data points, it is also important to consider the statistics that deal with the data as a graph, where the nodes denote the points and weighted edges denote the connections between points. This enables one to extract important topological attributes that give insights into the structure and models of networks and graphs. Kernel methods provide a fundamental connection between the independent pointwise view of data, and the viewpoint that deals with pairwise similarities between points. Many of the exploratory data analysis and mining tasks can be cast as kernel problems via the *kernel trick*, that is, by showing that the operations involve only dot-products between pairs of points. However, kernel methods also enable us to perform nonlinear analysis by using familiar linear algebraic and statistical methods in high-dimensional spaces comprising "nonlinear" dimensions. They further allow us to mine complex data as long as we have a way to measure the pairwise similarity between two abstract objects. Given that data mining deals with massive datasets with thousands of attributes and millions of points, another goal of exploratory analysis is to reduce the amount of data to be mined. For instance, feature selection and dimensionality reduction methods are used to select the most important dimensions, discretization methods can be used to reduce the number of values of an attribute, data sampling methods can be used to reduce the data size, and so on.

Part I of this book begins with basic statistical analysis of univariate and multivariate numeric data in Chapter 2. We describe measures of central tendency such as mean, median, and mode, and then we consider measures of dispersion such as range, variance, and covariance. We emphasize the dual algebraic and probabilistic views, and highlight the geometric interpretation of the various measures. We especially focus on the multivariate normal distribution, which is widely used as the default parametric model for data in both classification and clustering. In Chapter 3 we show how categorical data can be modeled via the multivariate binomial and the multinomial distributions. We describe the contingency table analysis approach to test for dependence between categorical attributes. Next, in Chapter 4 we show how to analyze graph data in terms of the topological structure, with special focus on various graph centrality measures such as closeness, betweenness, prestige, PageRank, and so on. We also study basic topological properties of real-world networks such as the *small*

world property, which states that real graphs have small average path length between pairs of nodes, the *clustering effect*, which indicates local clustering around nodes, and the *scale-free property*, which manifests itself in a *power-law* degree distribution. We describe models that can explain some of these characteristics of real-world graphs; these include the Erdös–Rényi random graph model, the Watts–Strogatz model, and the Barabási–Albert model. Kernel methods are then introduced in Chapter 5, which provide new insights and connections between linear, nonlinear, graph, and complex data mining tasks. We briefly highlight the theory behind kernel functions, with the key concept being that a positive semidefinite kernel corresponds to a dot product in some high-dimensional feature space, and thus we can use familiar numeric analysis methods for nonlinear or complex object analysis provided we can compute the pairwise kernel matrix of similarities between object instances. We describe various kernels for numeric or vector data, as well as sequence and graph data. In Chapter 6 we consider the peculiarities of high-dimensional space, colorfully referred to as *the curse of dimensionality*. In particular, we study the scattering effect, that is, the fact that data points lie along the surface and corners in high dimensions, with the "center" of the space being virtually empty. We show the proliferation of orthogonal axes and also the behavior of the multivariate normal distribution in high dimensions. Finally, in Chapter 7 we describe the widely used dimensionality reduction methods such as principal component analysis (PCA) and singular value decomposition (SVD). PCA finds the optimal k-dimensional subspace that captures most of the variance in the data. We also show how kernel PCA can be used to find nonlinear directions that capture the most variance. We conclude with the powerful SVD spectral decomposition method, studying its geometry, and its relationship to PCA.

1.5.2 Frequent Pattern Mining

Frequent pattern mining refers to the task of extracting informative and useful patterns in massive and complex datasets. Patterns comprise sets of co-occurring attribute values, called *itemsets*, or more complex patterns, such as sequences, which consider explicit precedence relationships (either positional or temporal), and graphs, which consider arbitrary relationships between points. The key goal is to discover hidden trends and behaviors in the data to understand better the interactions among the points and attributes.

Part II begins by presenting efficient algorithms for frequent itemset mining in Chapter 8. The key methods include the level-wise Apriori algorithm, the "vertical" intersection based Eclat algorithm, and the frequent pattern tree and projection based FPGrowth method. Typically the mining process results in too many frequent patterns that can be hard to interpret. In Chapter 9 we consider approaches to summarize the mined patterns; these include maximal (GenMax algorithm), closed (Charm algorithm), and non-derivable itemsets. We describe effective methods for frequent sequence mining in Chapter 10, which include the level-wise GSP method, the vertical SPADE algorithm, and the projection-based PrefixSpan approach. We also describe how consecutive subsequences, also called substrings, can be mined much more efficiently via Ukkonen's linear time and space suffix tree method. Moving

beyond sequences to arbitrary graphs, we describe the popular and efficient gSpan algorithm for frequent subgraph mining in Chapter 11. Graph mining involves two key steps, namely graph isomorphism checks to eliminate duplicate patterns during pattern enumeration and subgraph isomorphism checks during frequency computation. These operations can be performed in polynomial time for sets and sequences, but for graphs it is known that subgraph isomorphism is NP-hard, and thus there is no polynomial time method possible unless P = NP. The gSpan method proposes a new canonical code and a systematic approach to subgraph extension, which allow it to efficiently detect duplicates and to perform several subgraph isomorphism checks much more efficiently than performing them individually. Given that pattern mining methods generate many output results it is very important to assess the mined patterns. We discuss strategies for assessing both the frequent patterns and rules that can be mined from them in Chapter 12, emphasizing methods for significance testing.

1.5.3 Clustering

Clustering is the task of partitioning the points into *natural groups* called clusters, such that points within a group are very similar, whereas points across clusters are as dissimilar as possible. Depending on the data and desired cluster characteristics, there are different types of clustering paradigms such as representative-based, hierarchical, density-based, graph-based, and spectral clustering.

Part III starts with representative-based clustering methods (Chapter 13), which include the K-means and Expectation-Maximization (EM) algorithms. K-means is a greedy algorithm that minimizes the squared error of points from their respective cluster means, and it performs hard clustering, that is, each point is assigned to only one cluster. We also show how kernel K-means can be used for nonlinear clusters. EM generalizes K-means by modeling the data as a mixture of normal distributions, and it finds the cluster parameters (the mean and covariance matrix) by maximizing the likelihood of the data. It is a soft clustering approach, that is, instead of making a hard assignment, it returns the probability that a point belongs to each cluster. In Chapter 14 we consider various agglomerative hierarchical clustering methods, which start from each point in its own cluster, and successively merge (or agglomerate) pairs of clusters until the desired number of clusters have been found. We consider various cluster proximity measures that distinguish the different hierarchical methods. There are some datasets where the points from different clusters may in fact be closer in distance than points from the same cluster; this usually happens when the clusters are nonconvex in shape. Density-based clustering methods described in Chapter 15 use the density or connectedness properties to find such nonconvex clusters. The two main methods are DBSCAN and its generalization DENCLUE, which is based on kernel density estimation. We consider graph clustering methods in Chapter 16, which are typically based on spectral analysis of graph data. Graph clustering can be considered as an optimization problem over a k-way cut in a graph; different objectives can be cast as spectral decomposition of different graph matrices, such as the (normalized) adjacency matrix, Laplacian matrix, and so on, derived from the original graph data or from the kernel matrix. Finally, given the proliferation of different types of clustering methods,

it is important to assess the mined clusters as to how good they are in capturing the natural groups in data. In Chapter 17, we describe various clustering validation and evaluation strategies, spanning external and internal measures to compare a clustering with the ground-truth if it is available, or to compare two clusterings. We also highlight methods for clustering stability, that is, the sensitivity of the clustering to data perturbation, and clustering tendency, that is, the clusterability of the data. We also consider methods to choose the parameter k, which is the user-specified value for the number of clusters to discover.

1.5.4 Classification

The classification task is to predict the label or class for a given unlabeled point. Formally, a classifier is a model or function M that predicts the class label $\hat{y}$ for a given input example $\mathbf{x}$, that is, $\hat{y} = M(\mathbf{x})$, where $\hat{y} \in \{c_1, c_2, \ldots, c_k\}$ and each c_i is a class label (a categorical attribute value). To build the model we require a set of points with their correct class labels, which is called a *training set*. After learning the model M, we can automatically predict the class for any new point. Many different types of classification models have been proposed such as decision trees, probabilistic classifiers, support vector machines, and so on.

Part IV starts with the powerful Bayes classifier, which is an example of the probabilistic classification approach (Chapter 18). It uses the Bayes theorem to predict the class as the one that maximizes the posterior probability $P(c_i|\mathbf{x})$. The main task is to estimate the joint probability density function $f(\mathbf{x})$ for each class, which is modeled via a multivariate normal distribution. One limitation of the Bayes approach is the number of parameters to be estimated which scales as $O(d^2)$. The naive Bayes classifier makes the simplifying assumption that all attributes are independent, which requires the estimation of only $O(d)$ parameters. It is, however, surprisingly effective for many datasets. In Chapter 19 we consider the popular decision tree classifier, one of whose strengths is that it yields models that are easier to understand compared to other methods. A decision tree recursively partitions the data space into "pure" regions that contain data points from only one class, with relatively few exceptions. Next, in Chapter 20, we consider the task of finding an optimal direction that separates the points from two classes via linear discriminant analysis. It can be considered as a dimensionality reduction method that also takes the class labels into account, unlike PCA, which does not consider the class attribute. We also describe the generalization of linear to kernel discriminant analysis, which allows us to find nonlinear directions via the kernel trick. In Chapter 21 we describe the support vector machine (SVM) approach in detail, which is one of the most effective classifiers for many different problem domains. The goal of SVMs is to find the optimal hyperplane that maximizes the *margin* between the classes. Via the kernel trick, SVMs can be used to find nonlinear boundaries, which nevertheless correspond to some linear hyperplane in some high-dimensional "nonlinear" space. One of the important tasks in classification is to assess how good the models are. We conclude this part with Chapter 22, which presents the various methodologies for assessing classification models. We define various classification performance measures including ROC analysis. We then describe the bootstrap and cross-validation approaches for classifier evaluation. Finally, we

discuss the bias–variance tradeoff in classification, and how ensemble classifiers can help improve the variance or the bias of a classifier.

1.6 FURTHER READING

For a review of the linear algebra concepts see Strang (2006) and Poole (2010), and for the probabilistic view see Evans and Rosenthal (2011). There are several good books on data mining, and machine and statistical learning; these include Hand, Mannila, and Smyth (2001); Han, Kamber, and Pei (2006); Witten, Frank, and Hall (2011); Tan, Steinbach, and Kumar (2013); Bishop (2006) and Hastie, Tibshirani, and Friedman (2009).

Bishop, C. (2006). *Pattern Recognition and Machine Learning*. Information Science and Statistics. New York: Springer Science+Business Media.

Evans, M. and Rosenthal, J. (2011). *Probability and Statistics: The Science of Uncertainty*, 2nd ed. New York: W. H. Freeman.

Han, J., Kamber, M., and Pei, J. (2006). *Data Mining: Concepts and Techniques,* 2nd ed. The Morgan Kaufmann Series in Data Management Systems. Philadelphia: Elsevier Science.

Hand, D., Mannila, H., and Smyth, P. (2001). *Principles of Data Mining*. Adaptative Computation and Machine Learning Series. Cambridge, MA: MIT Press.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning*, 2nd ed. Springer Series in Statistics. NewYork: Springer Science + Business Media.

Poole, D. (2010). *Linear Algebra: A Modern Introduction*, 3rd ed. Independence, KY: Cengage Learning.

Strang, G. (2006). *Linear Algebra and Its Applications*, 4th ed. Independence, KY: Thomson Brooks/Cole, Cengage Learning.

Tan, P., Steinbach, M., and Kumar, V. (2013). *Introduction to Data Mining*, 2nd ed. Upper Saddle River, NJ: Prentice Hall.

Witten, I., Frank, E., and Hall, M. (2011). *Data Mining: Practical Machine Learning Tools and Techniques: Practical Machine Learning Tools and Techniques*, 3rd ed. The Morgan Kaufmann Series in Data Management Systems. Philadelphia: Elsevier Science.

1.7 EXERCISES

Q1. Show that the mean of the centered data matrix $\mathbf{Z}$ in (1.5) is $\mathbf{0}$.

Q2. Prove that for the L_p-distance in Eq. (1.2), we have

$$\delta_\infty(\mathbf{x}, \mathbf{y}) = \lim_{p \to \infty} \delta_p(\mathbf{x}, \mathbf{y}) = \max_{i=1}^{d} \{|x_i - y_i|\}$$

for $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$.

PART ONE

DATA ANALYSIS FOUNDATIONS

CHAPTER 2 Numeric Attributes

In this chapter, we discuss basic statistical methods for exploratory data analysis of numeric attributes. We look at measures of central tendency or location, measures of dispersion, and measures of linear dependence or association between attributes. We emphasize the connection between the probabilistic and the geometric and algebraic views of the data matrix.

2.1 UNIVARIATE ANALYSIS

Univariate analysis focuses on a single attribute at a time; thus the data matrix $\mathbf{D}$ can be thought of as an $n \times 1$ matrix, or simply a column vector, given as

$$\mathbf{D} = \begin{pmatrix} X \\ \hline x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

where X is the numeric attribute of interest, with $x_i \in \mathbb{R}$. X is assumed to be a random variable, with each point x_i $(1 \leq i \leq n)$ itself treated as an identity random variable. We assume that the observed data is a random sample drawn from X, that is, each variable x_i is independent and identically distributed as X. In the vector view, we treat the sample as an n-dimensional vector, and write $X \in \mathbb{R}^n$.

In general, the probability density or mass function $f(x)$ and the cumulative distribution function $F(x)$, for attribute X, are both unknown. However, we can estimate these distributions directly from the data sample, which also allow us to compute statistics to estimate several important population parameters.

Empirical Cumulative Distribution Function

The *empirical cumulative distribution function (CDF)* of X is given as

$$\hat{F}(x) = \frac{1}{n} \sum_{i=1}^{n} I(x_i \leq x) \tag{2.1}$$

where

$$I(x_i \leq x) = \begin{cases} 1 & \text{if } x_i \leq x \\ 0 & \text{if } x_i > x \end{cases}$$

is a binary *indicator variable* that indicates whether the given condition is satisfied or not. Intuitively, to obtain the empirical CDF we compute, for each value $x \in \mathbb{R}$, how many points in the sample are less than or equal to x. The empirical CDF puts a probability mass of $\frac{1}{n}$ at each point x_i. Note that we use the notation $\hat{F}$ to denote the fact that the empirical CDF is an estimate for the unknown population CDF F.

Inverse Cumulative Distribution Function

Define the *inverse cumulative distribution function* or *quantile function* for a random variable X as follows:

$$F^{-1}(q) = \min\{x \mid \hat{F}(x) \geq q\} \qquad \text{for } q \in [0, 1] \tag{2.2}$$

That is, the inverse CDF gives the least value of X, for which q fraction of the values are higher, and $1 - q$ fraction of the values are lower. The *empirical inverse cumulative distribution function* $\hat{F}^{-1}$ can be obtained from Eq. (2.1).

Empirical Probability Mass Function

The *empirical probability mass function (PMF)* of X is given as

$$\hat{f}(x) = P(X = x) = \frac{1}{n} \sum_{i=1}^{n} I(x_i = x) \tag{2.3}$$

where

$$I(x_i = x) = \begin{cases} 1 & \text{if } x_i = x \\ 0 & \text{if } x_i \neq x \end{cases}$$

The empirical PMF also puts a probability mass of $\frac{1}{n}$ at each point x_i.

2.1.1 Measures of Central Tendency

These measures given an indication about the concentration of the probability mass, the "middle" values, and so on.

Mean

The *mean*, also called the *expected value*, of a random variable X is the arithmetic average of the values of X. It provides a one-number summary of the *location* or *central tendency* for the distribution of X.

The mean or expected value of a discrete random variable X is defined as

$$\mu = E[X] = \sum_{x} x f(x) \tag{2.4}$$

where $f(x)$ is the probability mass function of X.

The expected value of a continuous random variable X is defined as

$$\mu = E[X] = \int_{-\infty}^{\infty} x f(x)\, dx$$

where $f(x)$ is the probability density function of X.

Sample Mean The *sample mean* is a statistic, that is, a function $\hat{\mu} : \{x_1, x_2, \ldots, x_n\} \to \mathbb{R}$, defined as the average value of x_i's:

$$\hat{\mu} = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{2.5}$$

It serves as an estimator for the unknown mean value μ of X. It can be derived by plugging in the empirical PMF $\hat{f}(x)$ in Eq. (2.4):

$$\hat{\mu} = \sum_x x\, \hat{f}(x) = \sum_x x \left(\frac{1}{n} \sum_{i=1}^{n} I(x_i = x) \right) = \frac{1}{n} \sum_{i=1}^{n} x_i$$

Sample Mean Is Unbiased An estimator $\hat{\theta}$ is called an *unbiased estimator* for parameter θ if $E[\hat{\theta}] = \theta$ for every possible value of θ. The sample mean $\hat{\mu}$ is an unbiased estimator for the population mean μ, as

$$E[\hat{\mu}] = E\left[\frac{1}{n} \sum_{i=1}^{n} x_i \right] = \frac{1}{n} \sum_{i=1}^{n} E[x_i] = \frac{1}{n} \sum_{i=1}^{n} \mu = \mu \tag{2.6}$$

where we use the fact that the random variables x_i are IID according to X, which implies that they have the same mean μ as X, that is, $E[x_i] = \mu$ for all x_i. We also used the fact that the expectation function E is a *linear operator*, that is, for any two random variables X and Y, and real numbers a and b, we have $E[aX + bY] = aE[X] + bE[Y]$.

Robustness We say that a statistic is *robust* if it is not affected by extreme values (such as outliers) in the data. The sample mean is unfortunately not robust because a single large value (an outlier) can skew the average. A more robust measure is the *trimmed mean* obtained after discarding a small fraction of extreme values on one or both ends. Furthermore, the mean can be somewhat misleading in that it is typically not a value that occurs in the sample, and it may not even be a value that the random variable can actually assume (for a discrete random variable). For example, the number of cars per capita is an integer-valued random variable, but according to the US Bureau of Transportation Studies, the average number of passenger cars in the United States was 0.45 in 2008 (137.1 million cars, with a population size of 304.4 million). Obviously, one cannot own 0.45 cars; it can be interpreted as saying that on average there are 45 cars per 100 people.

Median

The *median* of a random variable is defined as the value m such that

$$P(X \leq m) \geq \frac{1}{2} \text{ and } P(X \geq m) \geq \frac{1}{2}$$

In other words, the median m is the "middle-most" value; half of the values of X are less and half of the values of X are more than m. In terms of the (inverse) cumulative distribution function, the median is therefore the value m for which

$$F(m) = 0.5 \text{ or } m = F^{-1}(0.5)$$

The *sample median* can be obtained from the empirical CDF [Eq. (2.1)] or the empirical inverse CDF [Eq. (2.2)] by computing

$$\hat{F}(m) = 0.5 \text{ or } m = \hat{F}^{-1}(0.5)$$

A simpler approach to compute the sample median is to first sort all the values x_i ($i \in [1, n]$) in increasing order. If n is odd, the median is the value at position $\frac{n+1}{2}$. If n is even, the values at positions $\frac{n}{2}$ and $\frac{n}{2}+1$ are both medians.

Unlike the mean, median is robust, as it is not affected very much by extreme values. Also, it is a value that occurs in the sample and a value the random variable can actually assume.

Mode

The *mode* of a random variable X is the value at which the probability mass function or the probability density function attains its maximum value, depending on whether X is discrete or continuous, respectively.

The *sample mode* is a value for which the empirical probability mass function [Eq. (2.3)] attains its maximum, given as

$$\text{mode}(X) = \arg\max_x \hat{f}(x)$$

The mode may not be a very useful measure of central tendency for a sample because by chance an unrepresentative element may be the most frequent element. Furthermore, if all values in the sample are distinct, each of them will be the mode.

Example 2.1 (Sample Mean, Median, and Mode). Consider the attribute `sepal length` (X_1) in the Iris dataset, whose values are shown in Table 1.2. The sample mean is given as follows:

$$\hat{\mu} = \frac{1}{150}(5.9 + 6.9 + \cdots + 7.7 + 5.1) = \frac{876.5}{150} = 5.843$$

Figure 2.1 shows all 150 values of `sepal length`, and the sample mean. Figure 2.2a shows the empirical CDF and Figure 2.2b shows the empirical inverse CDF for `sepal length`.

Because $n = 150$ is even, the sample median is the value at positions $\frac{n}{2} = 75$ and $\frac{n}{2}+1 = 76$ in sorted order. For `sepal length` both these values are 5.8; thus the sample median is 5.8. From the inverse CDF in Figure 2.2b, we can see that

$$\hat{F}(5.8) = 0.5 \text{ or } 5.8 = \hat{F}^{-1}(0.5)$$

The sample mode for `sepal length` is 5, which can be observed from the frequency of 5 in Figure 2.1. The empirical probability mass at $x = 5$ is

$$\hat{f}(5) = \frac{10}{150} = 0.067$$

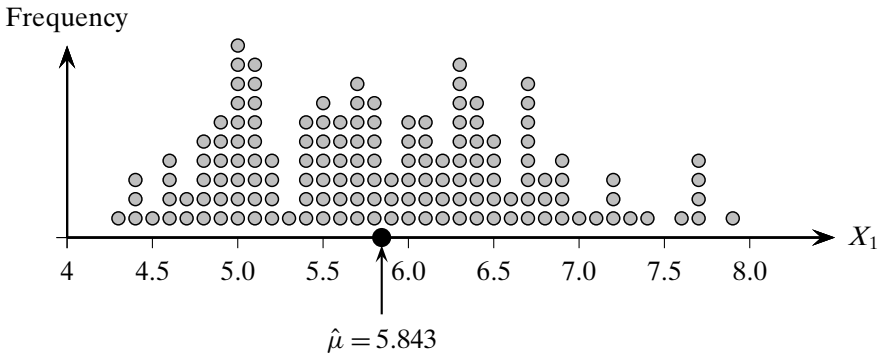
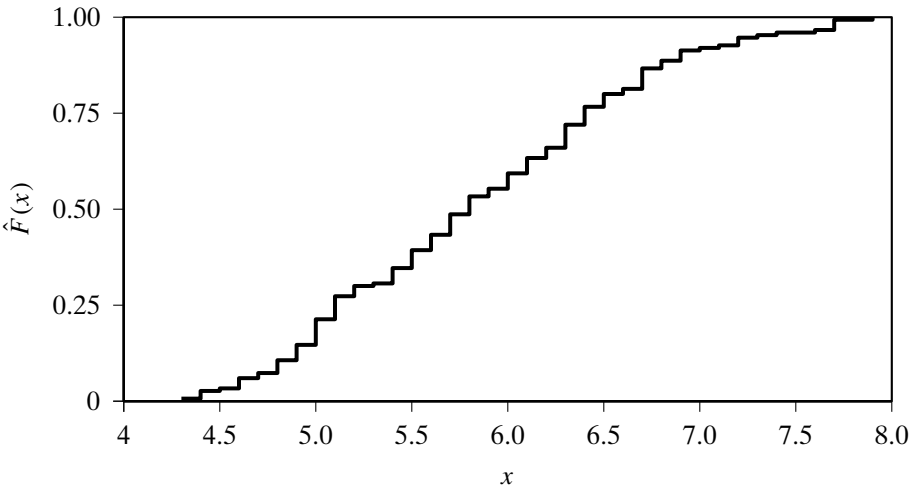
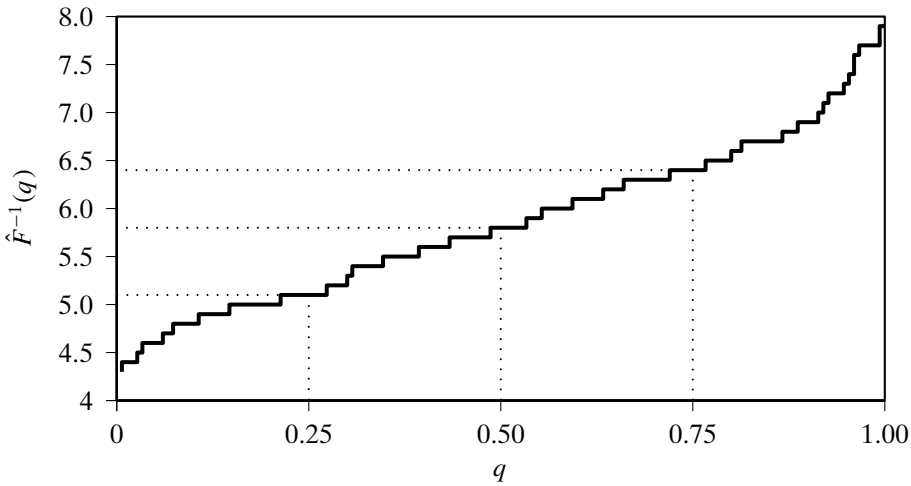


Figure 2.1. Sample mean for `sepal length`. Multiple occurrences of the same value are shown stacked.



(a) Empirical CDF



(b) Empirical inverse CDF

Figure 2.2. Empirical CDF and inverse CDF: `sepal length`.

2.1.2 Measures of Dispersion

The measures of dispersion give an indication about the spread or variation in the values of a random variable.

Range

The *value range* or simply *range* of a random variable X is the difference between the maximum and minimum values of X, given as

$$r = \max\{X\} - \min\{X\}$$

The (value) range of X is a population parameter, not to be confused with the range of the function X, which is the set of all the values X can assume. Which range is being used should be clear from the context.

The *sample range* is a statistic, given as

$$\hat{r} = \max_{i=1}^{n}\{x_i\} - \min_{i=1}^{n}\{x_i\}$$

By definition, range is sensitive to extreme values, and thus is not robust.

Interquartile Range

Quartiles are special values of the quantile function [Eq. (2.2)] that divide the data into four equal parts. That is, quartiles correspond to the quantile values of 0.25, 0.5, 0.75, and 1.0. The *first quartile* is the value $q_1 = F^{-1}(0.25)$, to the left of which 25% of the points lie; the *second quartile* is the same as the median value $q_2 = F^{-1}(0.5)$, to the left of which 50% of the points lie; the third quartile $q_3 = F^{-1}(0.75)$ is the value to the left of which 75% of the points lie; and the fourth quartile is the maximum value of X, to the left of which 100% of the points lie.

A more robust measure of the dispersion of X is the *interquartile range (IQR)*, defined as

$$IQR = q_3 - q_1 = F^{-1}(0.75) - F^{-1}(0.25) \tag{2.7}$$

IQR can also be thought of as a *trimmed range*, where we discard 25% of the low and high values of X. Or put differently, it is the range for the middle 50% of the values of X. IQR is robust by definition.

The *sample IQR* can be obtained by plugging in the empirical inverse CDF in Eq. (2.7):

$$\widehat{IQR} = \hat{q}_3 - \hat{q}_1 = \hat{F}^{-1}(0.75) - \hat{F}^{-1}(0.25)$$

Variance and Standard Deviation

The *variance* of a random variable X provides a measure of how much the values of X deviate from the mean or expected value of X. More formally, variance is the expected

value of the squared deviation from the mean, defined as

$$\sigma^2 = var(X) = E[(X-\mu)^2] = \begin{cases} \sum_x (x-\mu)^2 \, f(x) & \text{if } X \text{ is discrete} \\ \int_{-\infty}^{\infty} (x-\mu)^2 \, f(x)\, dx & \text{if } X \text{ is continuous} \end{cases} \tag{2.8}$$

The *standard deviation*, σ, is defined as the positive square root of the variance, σ^2.

We can also write the variance as the difference between the expectation of X^2 and the square of the expectation of X:

$$\begin{aligned} \sigma^2 = var(X) &= E[(X-\mu)^2] = E[X^2 - 2\mu X + \mu^2] \\ &= E[X^2] - 2\mu E[X] + \mu^2 = E[X^2] - 2\mu^2 + \mu^2 \\ &= E[X^2] - (E[X])^2 \end{aligned} \tag{2.9}$$

It is worth noting that variance is in fact the *second moment about the mean*, corresponding to $r = 2$, which is a special case of the *rth moment about the mean* for a random variable X, defined as $E[(\mathbf{x} - \mu)^r]$.

Sample Variance The *sample variance* is defined as

$$\hat{\sigma}^2 = \frac{1}{n} \sum_{i=1}^{n} (x_i - \hat{\mu})^2 \tag{2.10}$$

It is the average squared deviation of the data values x_i from the sample mean $\hat{\mu}$, and can be derived by plugging in the empirical probability function $\hat{f}$ from Eq. (2.3) into Eq. (2.8), as

$$\hat{\sigma}^2 = \sum_x (x-\hat{\mu})^2 \hat{f}(x) = \sum_x (x-\hat{\mu})^2 \left(\frac{1}{n} \sum_{i=1}^{n} I(x_i = x) \right) = \frac{1}{n} \sum_{i=1}^{n} (x_i - \hat{\mu})^2$$

The *sample standard deviation* is given as the positive square root of the sample variance:

$$\hat{\sigma} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (x_i - \hat{\mu})^2}$$

The *standard score*, also called the *z-score*, of a sample value x_i is the number of standard deviations the value is away from the mean:

$$z_i = \frac{x_i - \hat{\mu}}{\hat{\sigma}}$$

Put differently, the z-score of x_i measures the deviation of x_i from the mean value $\hat{\mu}$, in units of $\hat{\sigma}$.

Geometric Interpretation of Sample Variance We can treat the data sample for attribute X as a vector in n-dimensional space, where n is the sample size. That is, we write $X = (x_1, x_2, \ldots, x_n)^T \in \mathbb{R}^n$. Further, let

$$Z = X - \mathbf{1} \cdot \hat{\mu} = \begin{pmatrix} x_1 - \hat{\mu} \\ x_2 - \hat{\mu} \\ \vdots \\ x_n - \hat{\mu} \end{pmatrix}$$

denote the mean subtracted attribute vector, where $\mathbf{1} \in \mathbb{R}^n$ is the n-dimensional vector all of whose elements have value 1. We can rewrite Eq. (2.10) in terms of the magnitude of Z, that is, the dot product of Z with itself:

$$\hat{\sigma}^2 = \frac{1}{n} \|Z\|^2 = \frac{1}{n} Z^T Z = \frac{1}{n} \sum_{i=1}^{n} (x_i - \hat{\mu})^2 \tag{2.11}$$

The sample variance can thus be interpreted as the squared magnitude of the centered attribute vector, or the dot product of the centered attribute vector with itself, normalized by the sample size.

Example 2.2. Consider the data sample for `sepal length` shown in Figure 2.1. We can see that the sample range is given as

$$\max_i\{x_i\} - \min_i\{x_i\} = 7.9 - 4.3 = 3.6$$

From the inverse CDF for `sepal length` in Figure 2.2b, we can find the sample IQR as follows:

$$\hat{q}_1 = \hat{F}^{-1}(0.25) = 5.1$$
$$\hat{q}_3 = \hat{F}^{-1}(0.75) = 6.4$$
$$\widehat{IQR} = \hat{q}_3 - \hat{q}_1 = 6.4 - 5.1 = 1.3$$

The sample variance can be computed from the centered data vector via Eq. (2.11):

$$\hat{\sigma}^2 = \frac{1}{n}(X - \mathbf{1} \cdot \hat{\mu})^T (X - \mathbf{1} \cdot \hat{\mu}) = 102.168/150 = 0.681$$

The sample standard deviation is then

$$\hat{\sigma} = \sqrt{0.681} = 0.825$$

Variance of the Sample Mean Because the sample mean $\hat{\mu}$ is itself a statistic, we can compute its mean value and variance. The expected value of the sample mean is simply μ, as we saw in Eq. (2.6). To derive an expression for the variance of the sample mean,

we utilize the fact that the random variables x_i are all independent, and thus

$$var\left(\sum_{i=1}^{n} x_i\right) = \sum_{i=1}^{n} var(x_i)$$

Further, because all the x_i's are identically distributed as X, they have the same variance as X, that is,

$$var(x_i) = \sigma^2 \text{ for all } i$$

Combining the above two facts, we get

$$var\left(\sum_{i=1}^{n} x_i\right) = \sum_{i=1}^{n} var(x_i) = \sum_{i=1}^{n} \sigma^2 = n\sigma^2 \tag{2.12}$$

Further, note that

$$E\left[\sum_{i=1}^{n} x_i\right] = n\mu \tag{2.13}$$

Using Eqs. (2.9), (2.12), and (2.13), the variance of the sample mean $\hat{\mu}$ can be computed as

$$\begin{aligned} var(\hat{\mu}) &= E[(\hat{\mu} - \mu)^2] = E[\hat{\mu}^2] - \mu^2 = E\left[\left(\frac{1}{n}\sum_{i=1}^{n} x_i\right)^2\right] - \frac{1}{n^2} E\left[\sum_{i=1}^{n} x_i\right]^2 \\ &= \frac{1}{n^2}\left(E\left[\left(\sum_{i=1}^{n} x_i\right)^2\right] - E\left[\sum_{i=1}^{n} x_i\right]^2\right) = \frac{1}{n^2}\, var\left(\sum_{i=1}^{n} x_i\right) \\ &= \frac{\sigma^2}{n} \end{aligned} \tag{2.14}$$

In other words, the sample mean $\hat{\mu}$ varies or deviates from the mean μ in proportion to the population variance σ^2. However, the deviation can be made smaller by considering larger sample size n.

Sample Variance Is Biased, but Is Asymptotically Unbiased The sample variance in Eq. (2.10) is a *biased estimator* for the true population variance, σ^2, that is, $E[\hat{\sigma}^2] \neq \sigma^2$. To show this we make use of the identity

$$\sum_{i=1}^{n} (x_i - \mu)^2 = n(\hat{\mu} - \mu)^2 + \sum_{i=1}^{n} (x_i - \hat{\mu})^2 \tag{2.15}$$

Computing the expectation of $\hat{\sigma}^2$ by using Eq. (2.15) in the first step, we get

$$E[\hat{\sigma}^2] = E\left[\frac{1}{n}\sum_{i=1}^{n} (x_i - \hat{\mu})^2\right] = E\left[\frac{1}{n}\sum_{i=1}^{n} (x_i - \mu)^2\right] - E[(\hat{\mu} - \mu)^2] \tag{2.16}$$

Recall that the random variables x_i are IID according to X, which means that they have the same mean μ and variance σ^2 as X. This means that

$$E[(x_i - \mu)^2] = \sigma^2$$

Further, from Eq. (2.14) the sample mean $\hat{\mu}$ has variance $E[(\hat{\mu} - \mu)^2] = \frac{\sigma^2}{n}$. Plugging these into the Eq. (2.16) we get

$$\begin{aligned} E[\hat{\sigma}^2] &= \frac{1}{n} n\sigma^2 - \frac{\sigma^2}{n} \\ &= \left(\frac{n-1}{n}\right)\sigma^2 \end{aligned}$$

The sample variance $\hat{\sigma}^2$ is a biased estimator of σ^2, as its expected value differs from the population variance by a factor of $\frac{n-1}{n}$. However, it is *asymptotically unbiased*, that is, the bias vanishes as $n \to \infty$ because

$$\lim_{n\to\infty} \frac{n-1}{n} = \lim_{n\to\infty} 1 - \frac{1}{n} = 1$$

Put differently, as the sample size increases, we have

$$E[\hat{\sigma}^2] \to \sigma^2 \qquad \text{as } n \to \infty$$

2.2 BIVARIATE ANALYSIS

In bivariate analysis, we consider two attributes at the same time. We are specifically interested in understanding the association or dependence between them, if any. We thus restrict our attention to the two numeric attributes of interest, say X_1 and X_2, with the data $\mathbf{D}$ represented as an $n \times 2$ matrix:

$$\mathbf{D} = \begin{pmatrix} X_1 & X_2 \\ \hline x_{11} & x_{12} \\ x_{21} & x_{22} \\ \vdots & \vdots \\ x_{n1} & x_{n2} \end{pmatrix}$$

Geometrically, we can think of $\mathbf{D}$ in two ways. It can be viewed as n points or vectors in 2-dimensional space over the attributes X_1 and X_2, that is, $\mathbf{x}_i = (x_{i1}, x_{i2})^T \in \mathbb{R}^2$. Alternatively, it can be viewed as two points or vectors in an n-dimensional space comprising the points, that is, each column is a vector in $\mathbb{R}^n$, as follows:

$$\begin{aligned} X_1 &= (x_{11}, x_{21}, \ldots, x_{n1})^T \\ X_2 &= (x_{12}, x_{22}, \ldots, x_{n2})^T \end{aligned}$$

In the probabilistic view, the column vector $\mathbf{X} = (X_1, X_2)^T$ is considered a bivariate vector random variable, and the points $\mathbf{x}_i$ $(1 \le i \le n)$ are treated as a random sample drawn from $\mathbf{X}$, that is, $\mathbf{x}_i$'s are considered independent and identically distributed as $\mathbf{X}$.

Empirical Joint Probability Mass Function

The *empirical joint probability mass function* for $\mathbf{X}$ is given as

$$\hat{f}(\mathbf{x}) = P(\mathbf{X} = \mathbf{x}) = \frac{1}{n} \sum_{i=1}^{n} I(\mathbf{x}_i = \mathbf{x}) \tag{2.17}$$

$$\hat{f}(x_1, x_2) = P(X_1 = x_1, X_2 = x_2) = \frac{1}{n} \sum_{i=1}^{n} I(x_{i1} = x_1, x_{i2} = x_2)$$

where $\mathbf{x} = (x_1, x_2)^T$ and I is a indicator variable that takes on the value 1 only when its argument is true:

$$I(\mathbf{x}_i = \mathbf{x}) = \begin{cases} 1 & \text{if } x_{i1} = x_1 \text{ and } x_{i2} = x_2 \\ 0 & \text{otherwise} \end{cases}$$

As in the univariate case, the probability function puts a probability mass of $\frac{1}{n}$ at each point in the data sample.

2.2.1 Measures of Location and Dispersion

Mean

The bivariate mean is defined as the expected value of the vector random variable $\mathbf{X}$, defined as follows:

$$\boldsymbol{\mu} = E[\mathbf{X}] = E\left[\begin{pmatrix} X_1 \\ X_2 \end{pmatrix}\right] = \begin{pmatrix} E[X_1] \\ E[X_2] \end{pmatrix} = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix} \tag{2.18}$$

In other words, the bivariate mean vector is simply the vector of expected values along each attribute.

The sample mean vector can be obtained from $\hat{f}_{X_1}$ and $\hat{f}_{X_2}$, the empirical probability mass functions of X_1 and X_2, respectively, using Eq. (2.5). It can also be computed from the joint empirical PMF in Eq. (2.17)

$$\hat{\boldsymbol{\mu}} = \sum_{\mathbf{x}} \mathbf{x} \hat{f}(\mathbf{x}) = \sum_{\mathbf{x}} \mathbf{x} \left(\frac{1}{n} \sum_{i=1}^{n} I(\mathbf{x}_i = \mathbf{x}) \right) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i \tag{2.19}$$

Variance

We can compute the variance along each attribute, namely σ_1^2 for X_1 and σ_2^2 for X_2 using Eq. (2.8). The *total variance* [Eq. (1.4)] is given as

$$var(\mathbf{D}) = \sigma_1^2 + \sigma_2^2$$

The sample variances $\hat{\sigma}_1^2$ and $\hat{\sigma}_2^2$ can be estimated using Eq. (2.10), and the *sample total variance* is simply $\hat{\sigma}_1^2 + \hat{\sigma}_2^2$.

2.2.2 Measures of Association

Covariance

The *covariance* between two attributes X_1 and X_2 provides a measure of the association or linear dependence between them, and is defined as

$$\sigma_{12} = E[(X_1 - \mu_1)(X_2 - \mu_2)] \tag{2.20}$$

By linearity of expectation, we have

$$\begin{aligned}\sigma_{12} &= E[(X_1 - \mu_1)(X_2 - \mu_2)] \\ &= E[X_1X_2 - X_1\mu_2 - X_2\mu_1 + \mu_1\mu_2] \\ &= E[X_1X_2] - \mu_2 E[X_1] - \mu_1 E[X_2] + \mu_1\mu_2 \\ &= E[X_1X_2] - \mu_1\mu_2 \\ &= E[X_1X_2] - E[X_1]E[X_2] \end{aligned} \tag{2.21}$$

Eq. (2.21) can be seen as a generalization of the univariate variance [Eq. (2.9)] to the bivariate case.

If X_1 and X_2 are independent random variables, then we conclude that their covariance is zero. This is because if X_1 and X_2 are independent, then we have

$$E[X_1X_2] = E[X_1] \cdot E[X_2]$$

which in turn implies that

$$\sigma_{12} = 0$$

However, the converse is not true. That is, if $\sigma_{12} = 0$, one cannot claim that X_1 and X_2 are independent. All we can say is that there is no linear dependence between them, but we cannot rule out that there might be a higher order relationship or dependence between the two attributes.

The *sample covariance* between X_1 and X_2 is given as

$$\hat{\sigma}_{12} = \frac{1}{n}\sum_{i=1}^{n}(x_{i1} - \hat{\mu}_1)(x_{i2} - \hat{\mu}_2) \tag{2.22}$$

It can be derived by substituting the empirical joint probability mass function $\hat{f}(x_1, x_2)$ from Eq. (2.17) into Eq. (2.20), as follows:

$$\begin{aligned}\hat{\sigma}_{12} &= E[(X_1 - \hat{\mu}_1)(X_2 - \hat{\mu}_2)] \\ &= \sum_{\mathbf{x}=(x_1,x_2)^T}(x_1 - \hat{\mu}_1)(x_2 - \hat{\mu}_2)\hat{f}(x_1, x_2) \\ &= \frac{1}{n}\sum_{\mathbf{x}=(x_1,x_2)^T}\sum_{i=1}^{n}(x_1 - \hat{\mu}_1) \cdot (x_2 - \hat{\mu}_2) \cdot I(x_{i1} = x_1, x_{i2} = x_2) \\ &= \frac{1}{n}\sum_{i=1}^{n}(x_{i1} - \hat{\mu}_1)(x_{i2} - \hat{\mu}_2)\end{aligned}$$

Notice that sample covariance is a generalization of the sample variance [Eq. (2.10)] because

$$\hat{\sigma}_{11} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_1)(x_i - \mu_1) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_1)^2 = \hat{\sigma}_1^2$$

and similarly, $\hat{\sigma}_{22} = \hat{\sigma}_2^2$.

Correlation

The *correlation* between variables X_1 and X_2 is the *standardized covariance*, obtained by normalizing the covariance with the standard deviation of each variable, given as

$$\rho_{12} = \frac{\sigma_{12}}{\sigma_1 \sigma_2} = \frac{\sigma_{12}}{\sqrt{\sigma_1^2 \sigma_2^2}} \tag{2.23}$$

The *sample correlation* for attributes X_1 and X_2 is given as

$$\hat{\rho}_{12} = \frac{\hat{\sigma}_{12}}{\hat{\sigma}_1 \hat{\sigma}_2} = \frac{\sum_{i=1}^{n}(x_{i1} - \hat{\mu}_1)(x_{i2} - \hat{\mu}_2)}{\sqrt{\sum_{i=1}^{n}(x_{i1} - \hat{\mu}_1)^2 \sum_{i=1}^{n}(x_{i2} - \hat{\mu}_2)^2}} \tag{2.24}$$

Geometric Interpretation of Sample Covariance and Correlation

Let Z_1 and Z_2 denote the centered attribute vectors in $\mathbb{R}^n$, given as follows:

$$Z_1 = X_1 - \mathbf{1} \cdot \hat{\mu}_1 = \begin{pmatrix} x_{11} - \hat{\mu}_1 \\ x_{21} - \hat{\mu}_1 \\ \vdots \\ x_{n1} - \hat{\mu}_1 \end{pmatrix} \qquad Z_2 = X_2 - \mathbf{1} \cdot \hat{\mu}_2 = \begin{pmatrix} x_{12} - \hat{\mu}_2 \\ x_{22} - \hat{\mu}_2 \\ \vdots \\ x_{n2} - \hat{\mu}_2 \end{pmatrix}$$

The sample covariance [Eq. (2.22)] can then be written as

$$\hat{\sigma}_{12} = \frac{Z_1^T Z_2}{n}$$

In other words, the covariance between the two attributes is simply the dot product between the two centered attribute vectors, normalized by the sample size. The above can be seen as a generalization of the univariate sample variance given in Eq. (2.11).

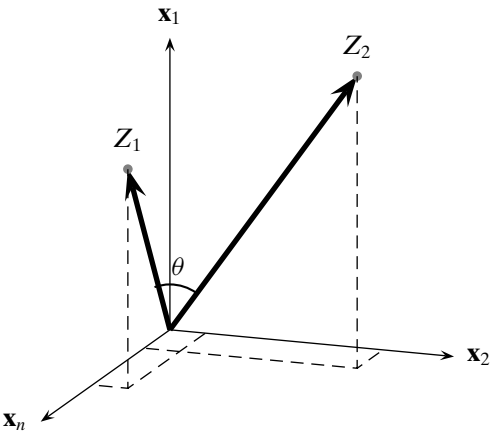


Figure 2.3. Geometric interpretation of covariance and correlation. The two centered attribute vectors are shown in the (conceptual) n-dimensional space $\mathbb{R}^n$ spanned by the n points.

The sample correlation [Eq. (2.24)] can be written as

$$\hat{\rho}_{12} = \frac{Z_1^T Z_2}{\sqrt{Z_1^T Z_1}\sqrt{Z_2^T Z_2}} = \frac{Z_1^T Z_2}{\|Z_1\| \, \|Z_2\|} = \left(\frac{Z_1}{\|Z_1\|}\right)^T \left(\frac{Z_2}{\|Z_2\|}\right) = \cos\theta \tag{2.25}$$

Thus, the correlation coefficient is simply the cosine of the angle [Eq. (1.3)] between the two centered attribute vectors, as illustrated in Figure 2.3.

Covariance Matrix

The variance–covariance information for the two attributes X_1 and X_2 can be summarized in the square 2×2 *covariance matrix*, given as

$$\begin{aligned}
\boldsymbol{\Sigma} &= E[(\mathbf{X} - \boldsymbol{\mu})(\mathbf{X} - \boldsymbol{\mu})^T] \\
&= E\left[\begin{pmatrix} X_1 - \mu_1 \\ X_2 - \mu_2 \end{pmatrix} \begin{pmatrix} X_1 - \mu_1 & X_2 - \mu_2 \end{pmatrix}\right] \\
&= \begin{pmatrix} E[(X_1 - \mu_1)(X_1 - \mu_1)] & E[(X_1 - \mu_1)(X_2 - \mu_2)] \\ E[(X_2 - \mu_2)(X_1 - \mu_1)] & E[(X_2 - \mu_2)(X_2 - \mu_2)] \end{pmatrix} \\
&= \begin{pmatrix} \sigma_1^2 & \sigma_{12} \\ \sigma_{21} & \sigma_2^2 \end{pmatrix}
\end{aligned} \tag{2.26}$$

Because $\sigma_{12} = \sigma_{21}$, $\boldsymbol{\Sigma}$ is a *symmetric* matrix. The covariance matrix records the attribute specific variances on the main diagonal, and the covariance information on the off-diagonal elements.

The *total variance* of the two attributes is given as the sum of the diagonal elements of $\boldsymbol{\Sigma}$, which is also called the *trace* of $\boldsymbol{\Sigma}$, given as

$$var(\mathbf{D}) = tr(\boldsymbol{\Sigma}) = \sigma_1^2 + \sigma_2^2$$

We immediately have $tr(\boldsymbol{\Sigma}) \geq 0$.

The *generalized variance* of the two attributes also considers the covariance, in addition to the attribute variances, and is given as the *determinant* of the covariance matrix $\boldsymbol{\Sigma}$, denoted as $|\boldsymbol{\Sigma}|$ or $\det(\boldsymbol{\Sigma})$. The generalized covariance is non-negative, because

$$|\boldsymbol{\Sigma}| = \det(\boldsymbol{\Sigma}) = \sigma_1^2\sigma_2^2 - \sigma_{12}^2 = \sigma_1^2\sigma_2^2 - \rho_{12}^2\sigma_1^2\sigma_2^2 = (1 - \rho_{12}^2)\sigma_1^2\sigma_2^2$$

where we used Eq. (2.23), that is, $\sigma_{12} = \rho_{12}\sigma_1\sigma_2$. Note that $|\rho_{12}| \leq 1$ implies that $\rho_{12}^2 \leq 1$, which in turn implies that $\det(\boldsymbol{\Sigma}) \geq 0$, that is, the determinant is non-negative.

The *sample covariance matrix* is given as

$$\widehat{\boldsymbol{\Sigma}} = \begin{pmatrix} \hat{\sigma}_1^2 & \hat{\sigma}_{12} \\ \hat{\sigma}_{12} & \hat{\sigma}_2^2 \end{pmatrix}$$

The sample covariance matrix $\widehat{\boldsymbol{\Sigma}}$ shares the same properties as $\boldsymbol{\Sigma}$, that is, it is symmetric and $|\widehat{\boldsymbol{\Sigma}}| \geq 0$, and it can be used to easily obtain the sample total and generalized variance.

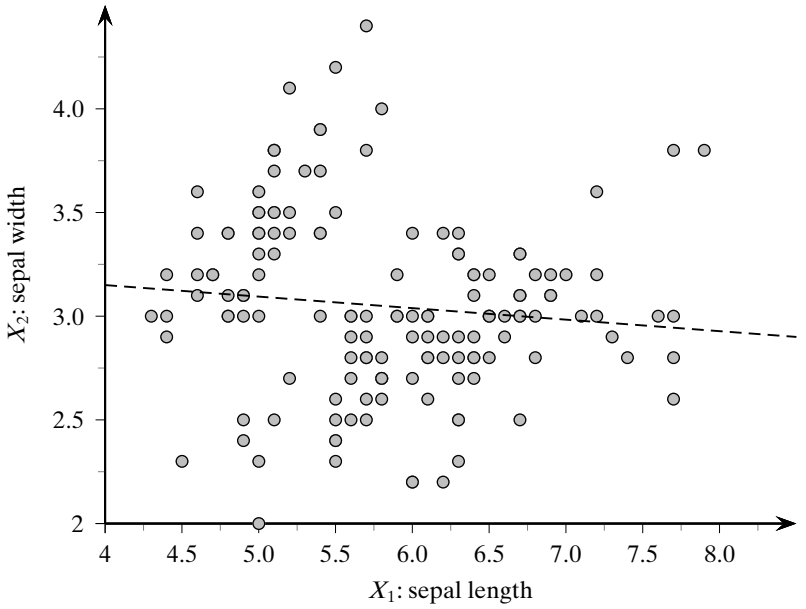


Figure 2.4. Correlation between `sepal length` and `sepal width`.

Example 2.3 (Sample Mean and Covariance). Consider the `sepal length` and `sepal width` attributes for the Iris dataset, plotted in Figure 2.4. There are $n = 150$ points in the $d = 2$ dimensional attribute space. The sample mean vector is given as

$$\hat{\boldsymbol{\mu}} = \begin{pmatrix} 5.843 \\ 3.054 \end{pmatrix}$$

The sample covariance matrix is given as

$$\widehat{\boldsymbol{\Sigma}} = \begin{pmatrix} 0.681 & -0.039 \\ -0.039 & 0.187 \end{pmatrix}$$

The variance for `sepal length` is $\hat{\sigma}_1^2 = 0.681$, and that for `sepal width` is $\hat{\sigma}_2^2 = 0.187$. The covariance between the two attributes is $\hat{\sigma}_{12} = -0.039$, and the correlation between them is

$$\hat{\rho}_{12} = \frac{-0.039}{\sqrt{0.681 \cdot 0.187}} = -0.109$$

Thus, there is a very weak negative correlation between these two attributes, as evidenced by the best linear fit line in Figure 2.4. Alternatively, we can consider the attributes `sepal length` and `sepal width` as two points in $\mathbb{R}^n$. The correlation is then the cosine of the angle between them; we have

$$\hat{\rho}_{12} = \cos\theta = -0.109, \text{ which implies that } \theta = \cos^{-1}(-0.109) = 96.26^\circ$$

The angle is close to 90°, that is, the two attribute vectors are almost orthogonal, indicating weak correlation. Further, the angle being greater than 90° indicates negative correlation.

The sample total variance is given as

$$tr(\widehat{\mathbf{\Sigma}}) = 0.681 + 0.187 = 0.868$$

and the sample generalized variance is given as

$$|\widehat{\mathbf{\Sigma}}| = \det(\widehat{\mathbf{\Sigma}}) = 0.681 \cdot 0.187 - (-0.039)^2 = 0.126$$

2.3 MULTIVARIATE ANALYSIS

In multivariate analysis, we consider all the d numeric attributes $X_1, X_2, \ldots, X_d$. The full data is an $n \times d$ matrix, given as

$$\mathbf{D} = \begin{pmatrix} X_1 & X_2 & \cdots & X_d \\ \hline x_{11} & x_{12} & \cdots & x_{1d} \\ x_{21} & x_{22} & \cdots & x_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nd} \end{pmatrix}$$

In the row view, the data can be considered as a set of n points or vectors in the d-dimensional attribute space

$$\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{id})^T \in \mathbb{R}^d$$

In the column view, the data can be considered as a set of d points or vectors in the n-dimensional space spanned by the data points

$$X_j = (x_{1j}, x_{2j}, \ldots, x_{nj})^T \in \mathbb{R}^n$$

In the probabilistic view, the d attributes are modeled as a vector random variable, $\mathbf{X} = (X_1, X_2, \ldots, X_d)^T$, and the points $\mathbf{x}_i$ are considered to be a random sample drawn from $\mathbf{X}$, that is, they are independent and identically distributed as $\mathbf{X}$.

Mean

Generalizing Eq. (2.18), the *multivariate mean vector* is obtained by taking the mean of each attribute, given as

$$\boldsymbol{\mu} = E[\mathbf{X}] = \begin{pmatrix} E[X_1] \\ E[X_2] \\ \vdots \\ E[X_d] \end{pmatrix} = \begin{pmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_d \end{pmatrix}$$

Generalizing Eq. (2.19), the *sample mean* is given as

$$\hat{\boldsymbol{\mu}} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i$$

Covariance Matrix

Generalizing Eq. (2.26) to d dimensions, the multivariate covariance information is captured by the $d \times d$ (square) symmetric *covariance matrix* that gives the covariance for each pair of attributes:

$$\mathbf{\Sigma} = E[(\mathbf{X} - \boldsymbol{\mu})(\mathbf{X} - \boldsymbol{\mu})^T] = \begin{pmatrix} \sigma_1^2 & \sigma_{12} & \cdots & \sigma_{1d} \\ \sigma_{21} & \sigma_2^2 & \cdots & \sigma_{2d} \\ \cdots & \cdots & \cdots & \cdots \\ \sigma_{d1} & \sigma_{d2} & \cdots & \sigma_d^2 \end{pmatrix}$$

The diagonal element σ_i^2 specifies the attribute variance for X_i, whereas the off-diagonal elements $\sigma_{ij} = \sigma_{ji}$ represent the covariance between attribute pairs X_i and X_j.

Covariance Matrix Is Positive Semidefinite

It is worth noting that $\mathbf{\Sigma}$ is a *positive semidefinite* matrix, that is,

$$\mathbf{a}^T \mathbf{\Sigma} \mathbf{a} \geq 0 \text{ for any } d\text{-dimensional vector } \mathbf{a}$$

To see this, observe that

$$\begin{aligned} \mathbf{a}^T \mathbf{\Sigma} \mathbf{a} &= \mathbf{a}^T E[(\mathbf{X} - \boldsymbol{\mu})(\mathbf{X} - \boldsymbol{\mu})^T]\mathbf{a} \\ &= E[\mathbf{a}^T (\mathbf{X} - \boldsymbol{\mu})(\mathbf{X} - \boldsymbol{\mu})^T \mathbf{a}] \\ &= E[Y^2] \\ &\geq 0 \end{aligned}$$

where Y is the random variable $Y = \mathbf{a}^T(\mathbf{X} - \boldsymbol{\mu}) = \sum_{i=1}^{d} a_i(X_i - \mu_i)$, and we use the fact that the expectation of a squared random variable is non-negative.

Because $\mathbf{\Sigma}$ is also symmetric, this implies that all the eigenvalues of $\mathbf{\Sigma}$ are real and non-negative. In other words the d eigenvalues of $\mathbf{\Sigma}$ can be arranged from the largest to the smallest as follows: $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d \geq 0$. A consequence is that the determinant of $\mathbf{\Sigma}$ is non-negative:

$$\det(\mathbf{\Sigma}) = \prod_{i=1}^{d} \lambda_i \geq 0 \tag{2.27}$$

Total and Generalized Variance

The total variance is given as the trace of the covariance matrix:

$$var(\mathbf{D}) = tr(\mathbf{\Sigma}) = \sigma_1^2 + \sigma_2^2 + \cdots + \sigma_d^2 \tag{2.28}$$

Being a sum of squares, the total variance must be non-negative.

The generalized variance is defined as the determinant of the covariance matrix, $\det(\mathbf{\Sigma})$, also denoted as $|\mathbf{\Sigma}|$. It gives a single value for the overall multivariate scatter. From Eq. (2.27) we have $\det(\mathbf{\Sigma}) \geq 0$.

Sample Covariance Matrix

The *sample covariance matrix* is given as

$$\widehat{\mathbf{\Sigma}} = E[(\mathbf{X} - \hat{\boldsymbol{\mu}})(\mathbf{X} - \hat{\boldsymbol{\mu}})^T] = \begin{pmatrix} \hat{\sigma}_1^2 & \hat{\sigma}_{12} & \cdots & \hat{\sigma}_{1d} \\ \hat{\sigma}_{21} & \hat{\sigma}_2^2 & \cdots & \hat{\sigma}_{2d} \\ \cdots & \cdots & \cdots & \cdots \\ \hat{\sigma}_{d1} & \hat{\sigma}_{d2} & \cdots & \hat{\sigma}_d^2 \end{pmatrix} \tag{2.29}$$

Instead of computing the sample covariance matrix element-by-element, we can obtain it via matrix operations. Let $\mathbf{Z}$ represent the centered data matrix, given as the matrix of centered attribute vectors $Z_i = X_i - \mathbf{1} \cdot \hat{\mu}_i$, where $\mathbf{1} \in \mathbb{R}^n$:

$$\mathbf{Z} = \mathbf{D} - \mathbf{1} \cdot \hat{\boldsymbol{\mu}}^T = \begin{pmatrix} | & | & & | \\ Z_1 & Z_2 & \cdots & Z_d \\ | & | & & | \end{pmatrix}$$

Alternatively, the centered data matrix can also be written in terms of the centered points $\mathbf{z}_i = \mathbf{x}_i - \hat{\boldsymbol{\mu}}$:

$$\mathbf{Z} = \mathbf{D} - \mathbf{1} \cdot \hat{\boldsymbol{\mu}}^T = \begin{pmatrix} \mathbf{x}_1^T - \hat{\boldsymbol{\mu}}^T \\ \mathbf{x}_2^T - \hat{\boldsymbol{\mu}}^T \\ \vdots \\ \mathbf{x}_n^T - \hat{\boldsymbol{\mu}}^T \end{pmatrix} = \begin{pmatrix} — & \mathbf{z}_1^T & — \\ — & \mathbf{z}_2^T & — \\ & \vdots & \\ — & \mathbf{z}_n^T & — \end{pmatrix}$$

In matrix notation, the sample covariance matrix can be written as

$$\widehat{\mathbf{\Sigma}} = \frac{1}{n}\left(\mathbf{Z}^T\mathbf{Z}\right) = \frac{1}{n}\begin{pmatrix} Z_1^T Z_1 & Z_1^T Z_2 & \cdots & Z_1^T Z_d \\ Z_2^T Z_1 & Z_2^T Z_2 & \cdots & Z_2^T Z_d \\ \vdots & \vdots & \ddots & \vdots \\ Z_d^T Z_1 & Z_d^T Z_2 & \cdots & Z_d^T Z_d \end{pmatrix} \tag{2.30}$$

The sample covariance matrix is thus given as the pairwise *inner or dot products* of the centered attribute vectors, normalized by the sample size.

In terms of the centered points $\mathbf{z}_i$, the sample covariance matrix can also be written as a sum of rank-one matrices obtained as the *outer product* of each centered point:

$$\widehat{\mathbf{\Sigma}} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{z}_i \cdot \mathbf{z}_i^T \tag{2.31}$$

Example 2.4 (Sample Mean and Covariance Matrix). Let us consider all four numeric attributes for the Iris dataset, namely `sepal length`, `sepal width`, `petal length`, and `petal width`. The multivariate sample mean vector is given as

$$\hat{\boldsymbol{\mu}} = \begin{pmatrix} 5.843 & 3.054 & 3.759 & 1.199 \end{pmatrix}^T$$

and the sample covariance matrix is given as

$$\widehat{\mathbf{\Sigma}} = \begin{pmatrix} 0.681 & -0.039 & 1.265 & 0.513 \\ -0.039 & 0.187 & -0.320 & -0.117 \\ 1.265 & -0.320 & 3.092 & 1.288 \\ 0.513 & -0.117 & 1.288 & 0.579 \end{pmatrix}$$

The sample total variance is

$$var(\mathbf{D}) = tr(\widehat{\mathbf{\Sigma}}) = 0.681 + 0.187 + 3.092 + 0.579 = 4.539$$

and the generalized variance is

$$\det(\widehat{\mathbf{\Sigma}}) = 1.853 \times 10^{-3}$$

Example 2.5 (Inner and Outer Product). To illustrate the inner and outer product–based computation of the sample covariance matrix, consider the 2-dimensional dataset

$$\mathbf{D} = \begin{pmatrix} A_1 & A_2 \\ \hline 1 & 0.8 \\ 5 & 2.4 \\ 9 & 5.5 \end{pmatrix}$$

The mean vector is as follows:

$$\hat{\mu} = \begin{pmatrix} \hat{\mu}_1 \\ \hat{\mu}_2 \end{pmatrix} = \begin{pmatrix} 15/3 \\ 8.7/3 \end{pmatrix} = \begin{pmatrix} 5 \\ 2.9 \end{pmatrix}$$

and the centered data matrix is then given as

$$\mathbf{Z} = \mathbf{D} - \mathbf{1} \cdot \boldsymbol{\mu}^T = \begin{pmatrix} 1 & 0.8 \\ 5 & 2.4 \\ 9 & 5.5 \end{pmatrix} - \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \begin{pmatrix} 5 & 2.9 \end{pmatrix} = \begin{pmatrix} -4 & -2.1 \\ 0 & -0.5 \\ 4 & 2.6 \end{pmatrix}$$

The inner-product approach [Eq. (2.30)] to compute the sample covariance matrix gives

$$\widehat{\mathbf{\Sigma}} = \frac{1}{n}\mathbf{Z}^T\mathbf{Z} = \frac{1}{3}\begin{pmatrix} -4 & 0 & 4 \\ -2.1 & -0.5 & 2.6 \end{pmatrix} \cdot \begin{pmatrix} -4 & -2.1 \\ 0 & -0.5 \\ 4 & 2.6 \end{pmatrix}$$

$$= \frac{1}{3}\begin{pmatrix} 32 & 18.8 \\ 18.8 & 11.42 \end{pmatrix} = \begin{pmatrix} 10.67 & 6.27 \\ 6.27 & 3.81 \end{pmatrix}$$

Alternatively, the outer-product approach [Eq. (2.31)] gives

$$\widehat{\mathbf{\Sigma}} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{z}_i \cdot \mathbf{z}_i^T$$

$$= \frac{1}{3}\left[\begin{pmatrix} -4 \\ -2.1 \end{pmatrix} \cdot \begin{pmatrix} -4 & -2.1 \end{pmatrix} + \begin{pmatrix} 0 \\ -0.5 \end{pmatrix} \cdot \begin{pmatrix} 0 & -0.5 \end{pmatrix} + \begin{pmatrix} 4 \\ 2.6 \end{pmatrix} \cdot \begin{pmatrix} 4 & 2.6 \end{pmatrix} \right]$$

$$= \frac{1}{3}\left[\begin{pmatrix}16.0 & 8.4\\ 8.4 & 4.41\end{pmatrix} + \begin{pmatrix}0.0 & 0.0\\ 0.0 & 0.25\end{pmatrix} + \begin{pmatrix}16.0 & 10.4\\ 10.4 & 6.76\end{pmatrix}\right]$$

$$= \frac{1}{3}\begin{pmatrix}32.0 & 18.8\\ 18.8 & 11.42\end{pmatrix} = \begin{pmatrix}10.67 & 6.27\\ 6.27 & 3.81\end{pmatrix}$$

where the centered points $\mathbf{z}_i$ are the rows of $\mathbf{Z}$. We can see that both the inner and outer product approaches yield the same sample covariance matrix.

2.4 DATA NORMALIZATION

When analyzing two or more attributes it is often necessary to normalize the values of the attributes, especially in those cases where the values are vastly different in scale.

Range Normalization

Let X be an attribute and let $x_1, x_2, \ldots, x_n$ be a random sample drawn from X. In *range normalization* each value is scaled by the sample range $\hat{r}$ of X:

$$x_i' = \frac{x_i - \min_i\{x_i\}}{\hat{r}} = \frac{x_i - \min_i\{x_i\}}{\max_i\{x_i\} - \min_i\{x_i\}}$$

After transformation the new attribute takes on values in the range $[0, 1]$.

Standard Score Normalization

In *standard score normalization*, also called z-normalization, each value is replaced by its z-score:

$$x_i' = \frac{x_i - \hat{\mu}}{\hat{\sigma}}$$

where $\hat{\mu}$ is the sample mean and $\hat{\sigma}^2$ is the sample variance of X. After transformation, the new attribute has mean $\hat{\mu}' = 0$, and standard deviation $\hat{\sigma}' = 1$.

Example 2.6. Consider the example dataset shown in Table 2.1. The attributes `Age` and `Income` have very different scales, with the latter having much larger values. Consider the distance between $\mathbf{x}_1$ and $\mathbf{x}_2$:

$$\|\mathbf{x}_1 - \mathbf{x}_2\| = \|(2, 200)^T\| = \sqrt{2^2 + 200^2} = \sqrt{40004} = 200.01$$

As we can observe, the contribution of `Age` is overshadowed by the value of `Income`.

The sample range for `Age` is $\hat{r} = 40 - 12 = 28$, with the minimum value 12. After range normalization, the new attribute is given as

$$\texttt{Age}' = (0, 0.071, 0.214, 0.393, 0.536, 0.571, 0.786, 0.893, 0.964, 1)^T$$

For example, for the point $\mathbf{x}_2 = (x_{21}, x_{22}) = (14, 500)$, the value $x_{21} = 14$ is transformed into

$$x_{21}' = \frac{14 - 12}{28} = \frac{2}{28} = 0.071$$

Table 2.1. Dataset for normalization

$\mathbf{x}_i$	Age (X_1)	Income (X_2)
$\mathbf{x}_1$	12	300
$\mathbf{x}_2$	14	500
$\mathbf{x}_3$	18	1000
$\mathbf{x}_4$	23	2000
$\mathbf{x}_5$	27	3500
$\mathbf{x}_6$	28	4000
$\mathbf{x}_7$	34	4300
$\mathbf{x}_8$	37	6000
$\mathbf{x}_9$	39	2500
$\mathbf{x}_{10}$	40	2700

Likewise, the sample range for `Income` is $2700 - 300 = 2400$, with a minimum value of 300; `Income` is therefore transformed into

$$\texttt{Income}' = (0, 0.035, 0.123, 0.298, 0.561, 0.649, 0.702, 1, 0.386, 0.421)^T$$

so that $x_{22} = 0.035$. The distance between $\mathbf{x}_1$ and $\mathbf{x}_2$ after range normalization is given as

$$\left\|\mathbf{x}_1' - \mathbf{x}_2'\right\| = \left\|(0,0)^T - (0.071, 0.035)^T\right\| = \left\|(-0.071, -0.035)^T\right\| = 0.079$$

We can observe that `Income` no longer skews the distance.

For z-normalization, we first compute the mean and standard deviation of both attributes:

	Age	Income
$\hat{\mu}$	27.2	2680
$\hat{\sigma}$	9.77	1726.15

`Age` is transformed into

$$\texttt{Age}' = (-1.56, -1.35, -0.94, -0.43, -0.02, 0.08, 0.70, 1.0, 1.21, 1.31)^T$$

For instance, the value $x_{21} = 14$, for the point $\mathbf{x}_2 = (x_{21}, x_{22}) = (14, 500)$, is transformed as

$$x_{21}' = \frac{14 - 27.2}{9.77} = -1.35$$

Likewise, `Income` is transformed into

$$\texttt{Income}' = (-1.38, -1.26, -0.97, -0.39, 0.48, 0.77, 0.94, 1.92, -0.10, 0.01)^T$$

so that $x_{22} = -1.26$. The distance between $\mathbf{x}_1$ and $\mathbf{x}_2$ after z-normalization is given as

$$\left\|\mathbf{x}_1' - \mathbf{x}_2'\right\| = \left\|(-1.56, -1.38)^T - (1.35, -1.26)^T\right\| = \left\|(-0.18, -0.12)^T\right\| = 0.216$$

2.5 NORMAL DISTRIBUTION

The normal distribution is one of the most important probability density functions, especially because many physically observed variables follow an approximately normal distribution. Furthermore, the sampling distribution of the mean of any arbitrary probability distribution follows a normal distribution. The normal distribution also plays an important role as the parametric distribution of choice in clustering, density estimation, and classification.

2.5.1 Univariate Normal Distribution

A random variable X has a normal distribution, with the parameters mean μ and variance σ^2, if the probability density function of X is given as follows:

$$f(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{(x-\mu)^2}{2\sigma^2}\right\}$$

The term $(x-\mu)^2$ measures the distance of a value x from the mean μ of the distribution, and thus the probability density decreases exponentially as a function of the distance from the mean. The maximum value of the density occurs at the mean value $x = \mu$, given as $f(\mu) = \frac{1}{\sqrt{2\pi\sigma^2}}$, which is inversely proportional to the standard deviation σ of the distribution.

Example 2.7. Figure 2.5 plots the standard normal distribution, which has the parameters $\mu = 0$ and $\sigma^2 = 1$. The normal distribution has a characteristic *bell* shape, and it is symmetric about the mean. The figure also shows the effect of different values of standard deviation on the shape of the distribution. A smaller value (e.g., $\sigma = 0.5$) results in a more "peaked" distribution that decays faster, whereas a larger value (e.g., $\sigma = 2$) results in a flatter distribution that decays slower. Because the normal distribution is symmetric, the mean μ is also the median, as well as the mode, of the distribution.

Probability Mass

Given an interval $[a,b]$ the probability mass of the normal distribution within that interval is given as

$$P(a \le x \le b) = \int_a^b f(x|\,\mu,\sigma^2)\,dx$$

In particular, we are often interested in the probability mass concentrated within k standard deviations from the mean, that is, for the interval $[\mu - k\sigma, \mu + k\sigma]$, which can be computed as

$$P\big(\mu - k\sigma \le x \le \mu + k\sigma\big) = \frac{1}{\sqrt{2\pi}\sigma} \int_{\mu-k\sigma}^{\mu+k\sigma} \exp\left\{-\frac{(x-\mu)^2}{2\sigma^2}\right\} dx$$

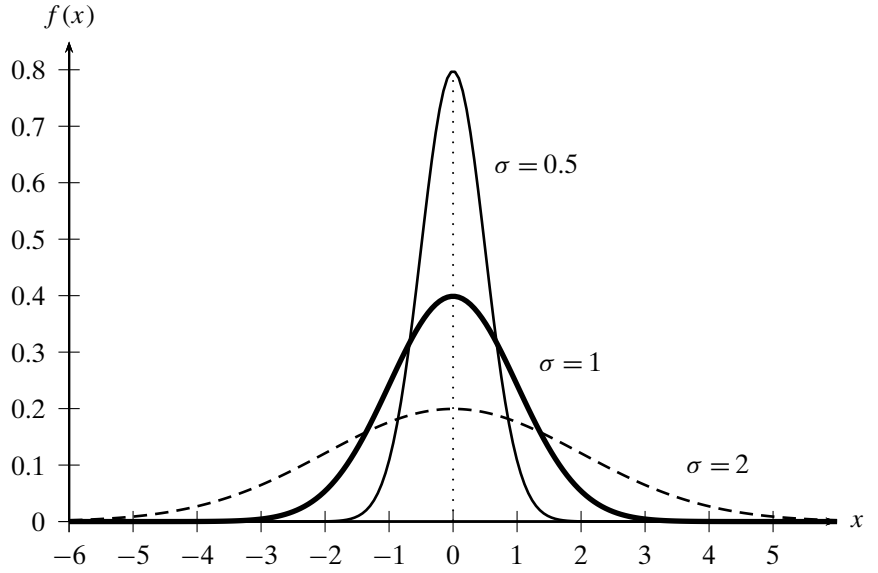


Figure 2.5. Normal distribution: $\mu = 0$, and different variances.

Via a change of variable $z = \frac{x-\mu}{\sigma}$, we get an equivalent formulation in terms of the standard normal distribution:

$$P(-k \leq z \leq k) = \frac{1}{\sqrt{2\pi}} \int_{-k}^{k} \mathrm{e}^{-\frac{1}{2}z^2} dz$$

$$= \frac{2}{\sqrt{2\pi}} \int_{0}^{k} \mathrm{e}^{-\frac{1}{2}z^2} dz$$

The last step follows from the fact that $\mathrm{e}^{-\frac{1}{2}z^2}$ is symmetric, and thus the integral over the range $[-k, k]$ is equivalent to 2 times the integral over the range $[0, k]$. Finally, via another change of variable $t = \frac{z}{\sqrt{2}}$, we get

$$P(-k \leq z \leq k) = P\big(0 \leq t \leq k/\sqrt{2}\big) = \frac{2}{\sqrt{\pi}} \int_{0}^{k/\sqrt{2}} \mathrm{e}^{-t^2} dt = \mathrm{erf}\left(k/\sqrt{2}\right) \tag{2.32}$$

where erf is the *Gauss error function*, defined as

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_{0}^{x} \mathrm{e}^{-t^2} dt$$

Using Eq. (2.32) we can compute the probability mass within k standard deviations of the mean. In particular, for $k = 1$, we have

$$P(\mu - \sigma \leq x \leq \mu + \sigma) = \mathrm{erf}(1/\sqrt{2}) = 0.6827$$

which means that 68.27% of all points lie within 1 standard deviation from the mean. For $k = 2$, we have $\text{erf}(2/\sqrt{2}) = 0.9545$, and for $k = 3$ we have $\text{erf}(3/\sqrt{2}) = 0.9973$. Thus, almost the entire probability mass (i.e., 99.73%) of a normal distribution is within $\pm 3\sigma$ from the mean μ.

2.5.2 Multivariate Normal Distribution

Given the d-dimensional vector random variable $\mathbf{X} = (X_1, X_2, \ldots, X_d)^T$, we say that $\mathbf{X}$ has a multivariate normal distribution, with the parameters mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$, if its joint multivariate probability density function is given as follows:

$$f(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(\sqrt{2\pi})^d \sqrt{|\boldsymbol{\Sigma}|}} \exp\left\{-\frac{(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x}-\boldsymbol{\mu})}{2}\right\} \tag{2.33}$$

where $|\boldsymbol{\Sigma}|$ is the determinant of the covariance matrix. As in the univariate case, the term

$$(\mathbf{x}_i - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x}_i - \boldsymbol{\mu}) \tag{2.34}$$

measures the distance, called the *Mahalanobis distance*, of the point $\mathbf{x}$ from the mean $\boldsymbol{\mu}$ of the distribution, taking into account all of the variance–covariance information between the attributes. The Mahalanobis distance is a generalization of Euclidean distance because if we set $\boldsymbol{\Sigma} = \mathbf{I}$, where $\mathbf{I}$ is the $d \times d$ identity matrix (with diagonal elements as 1's and off-diagonal elements as 0's), we get

$$(\mathbf{x}_i - \boldsymbol{\mu})^T \mathbf{I}^{-1} (\mathbf{x}_i - \boldsymbol{\mu}) = \|\mathbf{x}_i - \boldsymbol{\mu}\|^2$$

The Euclidean distance thus ignores the covariance information between the attributes, whereas the Mahalanobis distance explicitly takes it into consideration.

The *standard multivariate normal distribution* has parameters $\mu = \mathbf{0}$ and $\boldsymbol{\Sigma} = \mathbf{I}$. Figure 2.6a plots the probability density of the standard bivariate ($d = 2$) normal distribution, with parameters

$$\boldsymbol{\mu} = \mathbf{0} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

and

$$\boldsymbol{\Sigma} = \mathbf{I} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

This corresponds to the case where the two attributes are independent, and both follow the standard normal distribution. The symmetric nature of the standard normal distribution can be clearly seen in the contour plot shown in Figure 2.6b. Each level curve represents the set of points $\mathbf{x}$ with a fixed density value $f(\mathbf{x})$.

Geometry of the Multivariate Normal

Let us consider the geometry of the multivariate normal distribution for an arbitrary mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$. Compared to the standard normal distribution, we can expect the density contours to be shifted, scaled, and rotated. The shift or translation comes from the fact that the mean $\boldsymbol{\mu}$ is not necessarily the origin $\mathbf{0}$. The

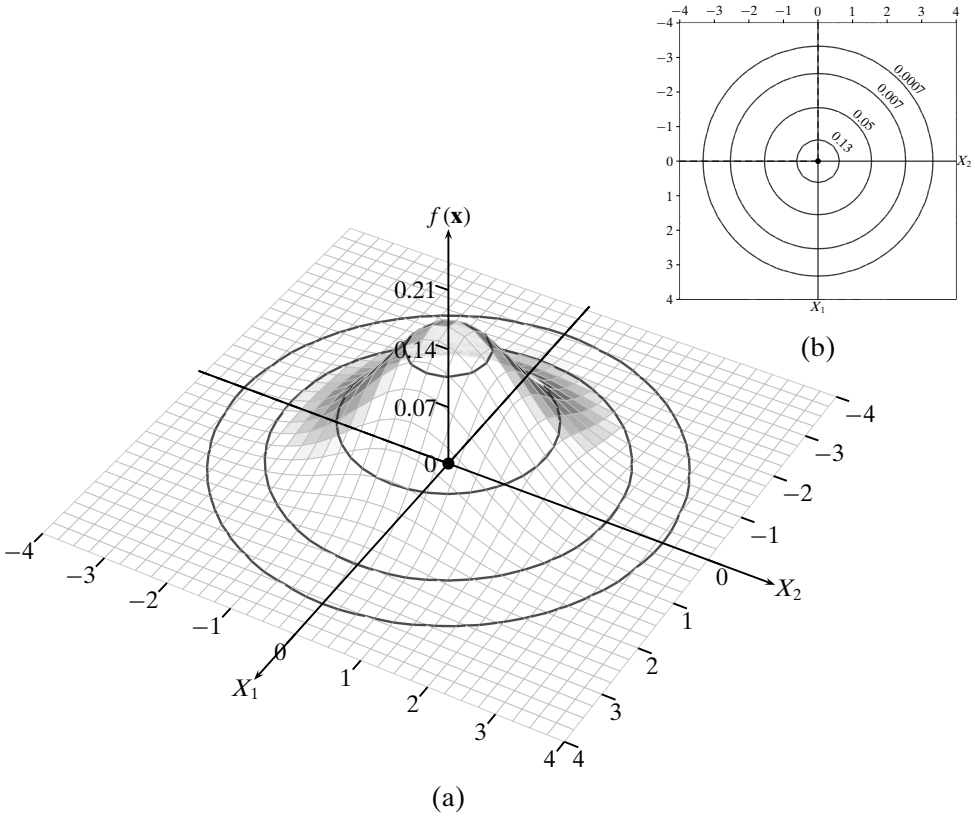


Figure 2.6. (a) Standard bivariate normal density and (b) its contour plot. Parameters: $\boldsymbol{\mu} = (0, 0)^T$, $\boldsymbol{\Sigma} = \mathbf{I}$.

scaling or skewing is a result of the attribute variances, and the rotation is a result of the covariances.

The shape or geometry of the normal distribution becomes clear by considering the eigen-decomposition of the covariance matrix. Recall that $\boldsymbol{\Sigma}$ is a $d \times d$ symmetric positive semidefinite matrix. The eigenvector equation for $\boldsymbol{\Sigma}$ is given as

$$\boldsymbol{\Sigma}\mathbf{u}_i = \lambda_i \mathbf{u}_i$$

Here λ_i is an eigenvalue of $\boldsymbol{\Sigma}$ and the vector $\mathbf{u}_i \in \mathbb{R}^d$ is the eigenvector corresponding to λ_i. Because $\boldsymbol{\Sigma}$ is symmetric and positive semidefinite it has d real and non-negative eigenvalues, which can be arranged in order from the largest to the smallest as follows: $\lambda_1 \geq \lambda_2 \geq \ldots \lambda_d \geq 0$. The diagonal matrix $\boldsymbol{\Lambda}$ is used to record these eigenvalues:

$$\boldsymbol{\Lambda} = \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_d \end{pmatrix}$$

Further, the eigenvectors are unit vectors (normal) and are mutually orthogonal, that is, they are orthonormal:

$$\mathbf{u}_i^T \mathbf{u}_i = 1 \quad \text{for all } i$$
$$\mathbf{u}_i^T \mathbf{u}_j = 0 \quad \text{for all } i \neq j$$

The eigenvectors can be put together into an orthogonal matrix $\mathbf{U}$, defined as a matrix with normal and mutually orthogonal columns:

$$\mathbf{U} = \begin{pmatrix} | & | & & | \\ \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_d \\ | & | & & | \end{pmatrix}$$

The eigen-decomposition of $\mathbf{\Sigma}$ can then be expressed compactly as follows:

$$\mathbf{\Sigma} = \mathbf{U} \mathbf{\Lambda} \mathbf{U}^T$$

This equation can be interpreted geometrically as a change in basis vectors. From the original d dimensions corresponding to the d attributes X_j, we derive d new dimensions $\mathbf{u}_i$. $\mathbf{\Sigma}$ is the covariance matrix in the original space, whereas $\mathbf{\Lambda}$ is the covariance matrix in the new coordinate space. Because $\mathbf{\Lambda}$ is a diagonal matrix, we can immediately conclude that after the transformation, each new dimension $\mathbf{u}_i$ has variance λ_i, and further that all covariances are zero. In other words, in the new space, the normal distribution is axis aligned (has no rotation component), but is skewed in each axis proportional to the eigenvalue λ_i, which represents the variance along that dimension (further details are given in Section 7.2.4).

Total and Generalized Variance

The determinant of the covariance matrix is is given as $\det(\mathbf{\Sigma}) = \prod_{i=1}^{d} \lambda_i$. Thus, the generalized variance of $\mathbf{\Sigma}$ is the product of its eigenvectors.

Given the fact that the trace of a square matrix is invariant to similarity transformation, such as a change of basis, we conclude that the total variance $var(\mathbf{D})$ for a dataset $\mathbf{D}$ is invariant, that is,

$$var(\mathbf{D}) = tr(\mathbf{\Sigma}) = \sum_{i=1}^{d} \sigma_i^2 = \sum_{i=1}^{d} \lambda_i = tr(\Lambda)$$

In other words $\sigma_1^2 + \cdots + \sigma_d^2 = \lambda_1 + \cdots + \lambda_d$.

Example 2.8 (Bivariate Normal Density). Treating attributes `sepal length` (X_1) and `sepal width` (X_2) in the Iris dataset (see Table 1.1) as continuous random variables, we can define a continuous bivariate random variable $\mathbf{X} = \begin{pmatrix} X_1 \\ X_2 \end{pmatrix}$. Assuming that $\mathbf{X}$ follows a bivariate normal distribution, we can estimate its parameters from the sample. The sample mean is given as

$$\hat{\boldsymbol{\mu}} = (5.843, 3.054)^T$$

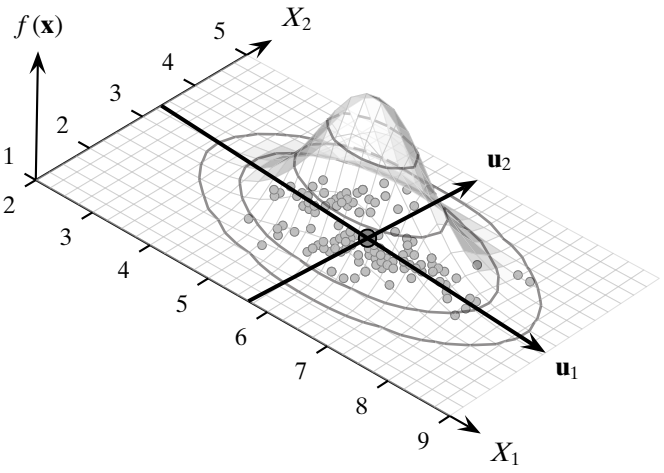


Figure 2.7. Iris: `sepal length` and `sepal width`, bivariate normal density and contours.

and the sample covariance matrix is given as

$$\widehat{\mathbf{\Sigma}} = \begin{pmatrix} 0.681 & -0.039 \\ -0.039 & 0.187 \end{pmatrix}$$

The plot of the bivariate normal density for the two attributes is shown in Figure 2.7. The figure also shows the contour lines and the data points.

Consider the point $\mathbf{x}_2 = (6.9, 3.1)^T$. We have

$$\mathbf{x}_2 - \hat{\boldsymbol{\mu}} = \begin{pmatrix} 6.9 \\ 3.1 \end{pmatrix} - \begin{pmatrix} 5.843 \\ 3.054 \end{pmatrix} = \begin{pmatrix} 1.057 \\ 0.046 \end{pmatrix}$$

The Mahalanobis distance between $\mathbf{x}_2$ and $\hat{\boldsymbol{\mu}}$ is

$$\begin{aligned} (\mathbf{x}_i - \hat{\boldsymbol{\mu}})^T \, \widehat{\mathbf{\Sigma}}^{-1} \, (\mathbf{x}_i - \hat{\boldsymbol{\mu}}) &= \begin{pmatrix} 1.057 & 0.046 \end{pmatrix} \begin{pmatrix} 0.681 & -0.039 \\ -0.039 & 0.187 \end{pmatrix}^{-1} \begin{pmatrix} 1.057 \\ 0.046 \end{pmatrix} \\ &= \begin{pmatrix} 1.057 & 0.046 \end{pmatrix} \begin{pmatrix} 1.486 & 0.31 \\ 0.31 & 5.42 \end{pmatrix} \begin{pmatrix} 1.057 \\ 0.046 \end{pmatrix} \\ &= 1.701 \end{aligned}$$

whereas the squared Euclidean distance between them is

$$\|(\mathbf{x}_2 - \hat{\boldsymbol{\mu}})\|^2 = \begin{pmatrix} 1.057 & 0.046 \end{pmatrix} \begin{pmatrix} 1.057 \\ 0.046 \end{pmatrix} = 1.119$$

The eigenvalues and the corresponding eigenvectors of $\widehat{\mathbf{\Sigma}}$ are as follows:

$$\lambda_1 = 0.684 \qquad \mathbf{u}_1 = (-0.997, 0.078)^T$$
$$\lambda_2 = 0.184 \qquad \mathbf{u}_2 = (-0.078, -0.997)^T$$

These two eigenvectors define the new axes in which the covariance matrix is given as

$$\mathbf{\Lambda} = \begin{pmatrix} 0.684 & 0 \\ 0 & 0.184 \end{pmatrix}$$

The angle between the original axes $\mathbf{e}_1 = (1,0)^T$ and $\mathbf{u}_1$ specifies the rotation angle for the multivariate normal:

$$\cos\theta = \mathbf{e}_1^T \mathbf{u}_1 = -0.997$$
$$\theta = \cos^{-1}(-0.997) = 175.5^\circ$$

Figure 2.7 illustrates the new coordinate axes and the new variances. We can see that in the original axes, the contours are only slightly rotated by angle 175.5° (or -4.5°).

2.6 FURTHER READING

There are several good textbooks that cover the topics discussed in this chapter in more depth; see Evans and Rosenthal (2011); Wasserman (2004) and Rencher and Christensen (2012).

Evans, M. and Rosenthal, J. (2011). *Probability and Statistics: The Science of Uncertainty*, 2nd ed. New York: W. H. Freeman.

Rencher, A. C. and Christensen, W. F. (2012). *Methods of Multivariate Analysis*, 3rd ed. Hoboken, NJ: John Wiley & Sons.

Wasserman, L. (2004). *All of Statistics: A Concise Course in Statistical Inference.* New York: Springer Science+Business Media.

2.7 EXERCISES

Q1. True or False:

(a) Mean is robust against outliers.

(b) Median is robust against outliers.

(c) Standard deviation is robust against outliers.

Q2. Let X and Y be two random variables, denoting age and weight, respectively. Consider a random sample of size $n = 20$ from these two variables

$$X = (69, 74, 68, 70, 72, 67, 66, 70, 76, 68, 72, 79, 74, 67, 66, 71, 74, 75, 75, 76)$$
$$Y = (153, 175, 155, 135, 172, 150, 115, 137, 200, 130, 140, 265, 185, 112, 140, 150, 165, 185, 210, 220)$$

(a) Find the mean, median, and mode for X.

(b) What is the variance for Y?

(c) Plot the normal distribution for X.
(d) What is the probability of observing an age of 80 or higher?
(e) Find the 2-dimensional mean $\hat{\boldsymbol{\mu}}$ and the covariance matrix $\widehat{\boldsymbol{\Sigma}}$ for these two variables.
(f) What is the correlation between age and weight?
(g) Draw a scatterplot to show the relationship between age and weight.

Q3. Show that the identity in Eq. (2.15) holds, that is,

$$\sum_{i=1}^{n}(x_i-\mu)^2 = n(\hat{\mu}-mu)^2 + \sum_{i=1}^{n}(x_i-\hat{\mu})^2$$

Q4. Prove that if x_i are independent random variables, then

$$var\left(\sum_{i=1}^{n} x_i\right) = \sum_{i=1}^{n} var(x_i)$$

This fact was used in Eq. (2.12).

Q5. Define a measure of deviation called *mean absolute deviation* for a random variable X as follows:

$$\frac{1}{n}\sum_{i=1}^{n}|x_i-\mu|$$

Is this measure robust? Why or why not?

Q6. Prove that the expected value of a vector random variable $\mathbf{X} = (X_1, X_2)^T$ is simply the vector of the expected value of the individual random variables X_1 and X_2 as given in Eq. (2.18).

Q7. Show that the correlation [Eq. (2.23)] between any two random variables X_1 and X_2 lies in the range $[-1, 1]$.

Q8. Given the dataset in Table 2.2, compute the covariance matrix and the generalized variance.

Table 2.2. Dataset for Q8

	X_1	X_2	X_3
$\mathbf{x}_1$	17	17	12
$\mathbf{x}_2$	11	9	13
$\mathbf{x}_3$	11	8	19

Q9. Show that the outer-product in Eq. (2.31) for the sample covariance matrix is equivalent to Eq. (2.29).

Q10. Assume that we are given two univariate normal distributions, N_A and N_B, and let their mean and standard deviation be as follows: $\mu_A = 4$, $\sigma_A = 1$ and $\mu_B = 8, \sigma_B = 2$.
(a) For each of the following values $x_i \in \{5, 6, 7\}$ find out which is the more likely normal distribution to have produced it.
(b) Derive an expression for the point for which the probability of having been produced by both the normals is the same.

Q11. Consider Table 2.3. Assume that both the attributes X and Y are numeric, and the table represents the entire population. If we know that the correlation between X and Y is zero, what can you infer about the values of Y?

Table 2.3. Dataset for Q11

X	Y
1	a
0	b
1	c
0	a
0	c

Q12. Under what conditions will the covariance matrix $\mathbf{\Sigma}$ be identical to the correlation matrix, whose (i, j) entry gives the correlation between attributes X_i and X_j? What can you conclude about the two variables?

CHAPTER 3 Categorical Attributes

In this chapter we present methods to analyze categorical attributes. Because categorical attributes have only symbolic values, many of the arithmetic operations cannot be performed directly on the symbolic values. However, we can compute the frequencies of these values and use them to analyze the attributes.

3.1 UNIVARIATE ANALYSIS

We assume that the data consists of values for a single categorical attribute, X. Let the domain of X consist of m symbolic values $dom(X) = \{a_1, a_2, \ldots, a_m\}$. The data $\mathbf{D}$ is thus an $n \times 1$ symbolic data matrix given as

$$\mathbf{D} = \begin{pmatrix} X \\ \hline x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

where each point $x_i \in dom(X)$.

3.1.1 Bernoulli Variable

Let us first consider the case when the categorical attribute X has domain $\{a_1, a_2\}$, with $m = 2$. We can model X as a Bernoulli random variable, which takes on two distinct values, 1 and 0, according to the mapping

$$X(v) = \begin{cases} 1 & \text{if } v = a_1 \\ 0 & \text{if } v = a_2 \end{cases}$$

The probability mass function (PMF) of X is given as

$$P(X = x) = f(x) = \begin{cases} p_1 & \text{if } x = 1 \\ p_0 & \text{if } x = 0 \end{cases}$$

where p_1 and p_0 are the parameters of the distribution, which must satisfy the condition

$$p_1 + p_0 = 1$$

Because there is only one free parameter, it is customary to denote $p_1 = p$, from which it follows that $p_0 = 1 - p$. The PMF of Bernoulli random variable X can then be written compactly as

$$P(X = x) = f(x) = p^x (1-p)^{1-x}$$

We can see that $P(X = 1) = p^1(1-p)^0 = p$ and $P(X = 0) = p^0(1-p)^1 = 1 - p$, as desired.

Mean and Variance

The expected value of X is given as

$$\mu = E[X] = 1 \cdot p + 0 \cdot (1-p) = p$$

and the variance of X is given as

$$\begin{aligned} \sigma^2 = var(X) &= E[X^2] - (E[X])^2 \\ &= (1^2 \cdot p + 0^2 \cdot (1-p)) - p^2 = p - p^2 = p(1-p) \end{aligned} \tag{3.1}$$

Sample Mean and Variance

To estimate the parameters of the Bernoulli variable X, we assume that each symbolic point has been mapped to its binary value. Thus, the set $\{x_1, x_2, \ldots, x_n\}$ is assumed to be a random sample drawn from X (i.e., each x_i is IID with X).

The sample mean is given as

$$\hat{\mu} = \frac{1}{n} \sum_{i=1}^{n} x_i = \frac{n_1}{n} = \hat{p} \tag{3.2}$$

where n_1 is the number of points with $x_i = 1$ in the random sample (equal to the number of occurrences of symbol a_1).

Let $n_0 = n - n_1$ denote the number of points with $x_i = 0$ in the random sample. The sample variance is given as

$$\begin{aligned} \hat{\sigma}^2 &= \frac{1}{n} \sum_{i=1}^{n} (x_i - \hat{\mu})^2 \\ &= \frac{n_1}{n}(1-\hat{p})^2 + \frac{n - n_1}{n}(-\hat{p})^2 \\ &= \hat{p}(1-\hat{p})^2 + (1-\hat{p})\hat{p}^2 \\ &= \hat{p}(1-\hat{p})(1-\hat{p}+\hat{p}) \\ &= \hat{p}(1-\hat{p}) \end{aligned}$$

The sample variance could also have been obtained directly from Eq. (3.1), by substituting $\hat{p}$ for p.

Example 3.1. Consider the `sepal length` attribute (X_1) for the Iris dataset in Table 1.1. Let us define an Iris flower as `Long` if its sepal length is in the range $[7, \infty]$, and `Short` if its sepal length is in the range $[-\infty, 7)$. Then X_1 can be treated as a categorical attribute with domain {`Long`, `Short`}. From the observed sample of size $n = 150$, we find 13 long Irises. The sample mean of X_1 is

$$\hat{\mu} = \hat{p} = 13/150 = 0.087$$

and its variance is

$$\hat{\sigma}^2 = \hat{p}(1 - \hat{p}) = 0.087(1 - 0.087) = 0.087 \cdot 0.913 = 0.079$$

Binomial Distribution: Number of Occurrences

Given the Bernoulli variable X, let $\{x_1, x_2, \ldots, x_n\}$ denote a random sample of size n drawn from X. Let N be the random variable denoting the number of occurrences of the symbol a_1 (value $X = 1$) in the sample. N has a binomial distribution, given as

$$f(N = n_1 \,|\, n, p) = \binom{n}{n_1} p^{n_1} (1 - p)^{n - n_1} \tag{3.3}$$

In fact, N is the sum of the n independent Bernoulli random variables x_i IID with X, that is, $N = \sum_{i=1}^{n} x_i$. By linearity of expectation, the mean or expected number of occurrences of symbol a_1 is given as

$$\mu_N = E[N] = E\left[\sum_{i=1}^{n} x_i\right] = \sum_{i=1}^{n} E[x_i] = \sum_{i=1}^{n} p = np$$

Because x_i are all independent, the variance of N is given as

$$\sigma_N^2 = var(N) = \sum_{i=1}^{n} var(x_i) = \sum_{i=1}^{n} p(1 - p) = np(1 - p)$$

Example 3.2. Continuing with Example 3.1, we can use the estimated parameter $\hat{p} = 0.087$ to compute the expected number of occurrences N of `Long` sepal length Irises via the binomial distribution:

$$E[N] = n\hat{p} = 150 \cdot 0.087 = 13$$

In this case, because p is estimated from the sample via $\hat{p}$, it is not surprising that the expected number of occurrences of long Irises coincides with the actual occurrences. However, what is more interesting is that we can compute the variance in the number of occurrences:

$$var(N) = n\hat{p}(1 - \hat{p}) = 150 \cdot 0.079 = 11.9$$

As the sample size increases, the binomial distribution given in Eq. 3.3 tends to a normal distribution with $\mu = 13$ and $\sigma = \sqrt{11.9} = 3.45$ for our example. Thus, with confidence greater than 95% we can claim that the number of occurrences of a_1 will lie in the range $\mu \pm 2\sigma = [9.55, 16.45]$, which follows from the fact that for a normal distribution 95.45% of the probability mass lies within two standard deviations from the mean (see Section 2.5.1).

3.1.2 Multivariate Bernoulli Variable

We now consider the general case when X is a categorical attribute with domain $\{a_1, a_2, \ldots, a_m\}$. We can model X as an m-dimensional Bernoulli random variable $\mathbf{X} = (A_1, A_2, \ldots, A_m)^T$, where each A_i is a Bernoulli variable with parameter p_i denoting the probability of observing symbol a_i. However, because X can assume only one of the symbolic values at any one time, if $X = a_i$, then $A_i = 1$, and $A_j = 0$ for all $j \neq i$. The range of the random variable $\mathbf{X}$ is thus the set $\{0, 1\}^m$, with the further restriction that if $X = a_i$, then $\mathbf{X} = \mathbf{e}_i$, where $\mathbf{e}_i$ is the ith standard basis vector $\mathbf{e}_i \in \mathbb{R}^m$ given as

$$\mathbf{e}_i = (\overbrace{0, \ldots, 0}^{i-1}, 1, \overbrace{0, \ldots, 0}^{m-i})^T$$

In $\mathbf{e}_i$, only the ith element is 1 ($e_{ii} = 1$), whereas all other elements are zero ($e_{ij} = 0, \forall j \neq i$).

This is precisely the definition of a *multivariate Bernoulli variable*, which is a generalization of a Bernoulli variable from two outcomes to m outcomes. We thus model the categorical attribute X as a multivariate Bernoulli variable $\mathbf{X}$ defined as

$$\mathbf{X}(v) = \mathbf{e}_i \text{ if } v = a_i$$

The range of $\mathbf{X}$ consists of m distinct vector values $\{\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_m\}$, with the PMF of $\mathbf{X}$ given as

$$P(\mathbf{X} = \mathbf{e}_i) = f(\mathbf{e}_i) = p_i$$

where p_i is the probability of observing value a_i. These parameters must satisfy the condition

$$\sum_{i=1}^{m} p_i = 1$$

The PMF can be written compactly as follows:

$$P(\mathbf{X} = \mathbf{e}_i) = f(\mathbf{e}_i) = \prod_{j=1}^{m} p_j^{e_{ij}} \tag{3.4}$$

Because $e_{ii} = 1$, and $e_{ij} = 0$ for $j \neq i$, we can see that, as expected, we have

$$f(\mathbf{e}_i) = \prod_{j=1}^{m} p_j^{e_{ij}} = p_1^{e_{i0}} \times \cdots p_i^{e_{ii}} \cdots \times p_m^{e_{im}} = p_1^0 \times \cdots p_i^1 \cdots \times p_m^0 = p_i$$

Table 3.1. Discretized `sepal length` attribute

Bins	Domain	Counts
[4.3, 5.2]	Very Short (a_1)	$n_1 = 45$
(5.2, 6.1]	Short (a_2)	$n_2 = 50$
(6.1, 7.0]	Long (a_3)	$n_3 = 43$
(7.0, 7.9]	Very Long (a_4)	$n_4 = 12$

Example 3.3. Let us consider the `sepal length` attribute (X_1) for the Iris dataset shown in Table 1.2. We divide the sepal length into four equal-width intervals, and give each interval a name as shown in Table 3.1. We consider X_1 as a categorical attribute with domain

$$\{a_1 = \texttt{VeryShort}, a_2 = \texttt{Short}, a_3 = \texttt{Long}, a_4 = \texttt{VeryLong}\}$$

We model the categorical attribute X_1 as a multivariate Bernoulli variable $\mathbf{X}$, defined as

$$\mathbf{X}(v) = \begin{cases} \mathbf{e}_1 = (1,0,0,0) & \text{if } v = a_1 \\ \mathbf{e}_2 = (0,1,0,0) & \text{if } v = a_2 \\ \mathbf{e}_3 = (0,0,1,0) & \text{if } v = a_3 \\ \mathbf{e}_4 = (0,0,0,1) & \text{if } v = a_4 \end{cases}$$

For example, the symbolic point $x_1 = \texttt{Short} = a_2$ is represented as the vector $(0,1,0,0)^T = \mathbf{e}_2$.

Mean

The mean or expected value of $\mathbf{X}$ can be obtained as

$$\boldsymbol{\mu} = E[\mathbf{X}] = \sum_{i=1}^{m} \mathbf{e}_i f(\mathbf{e}_i) = \sum_{i=1}^{m} \mathbf{e}_i p_i = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} p_1 + \cdots + \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix} p_m = \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_m \end{pmatrix} = \mathbf{p} \tag{3.5}$$

Sample Mean

Assume that each symbolic point $x_i \in \mathbf{D}$ is mapped to the variable $\mathbf{x}_i = \mathbf{X}(x_i)$. The mapped dataset $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ is then assumed to be a random sample IID with $\mathbf{X}$. We can compute the sample mean by placing a probability mass of $\frac{1}{n}$ at each point

$$\hat{\boldsymbol{\mu}} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i = \sum_{i=1}^{m} \frac{n_i}{n} \mathbf{e}_i = \begin{pmatrix} n_1/n \\ n_2/n \\ \vdots \\ n_m/n \end{pmatrix} = \begin{pmatrix} \hat{p}_1 \\ \hat{p}_2 \\ \vdots \\ \hat{p}_m \end{pmatrix} = \hat{\mathbf{p}} \tag{3.6}$$

where n_i is the number of occurrences of the vector value $\mathbf{e}_i$ in the sample, which is equivalent to the number of occurrences of the symbol a_i. Furthermore, we have

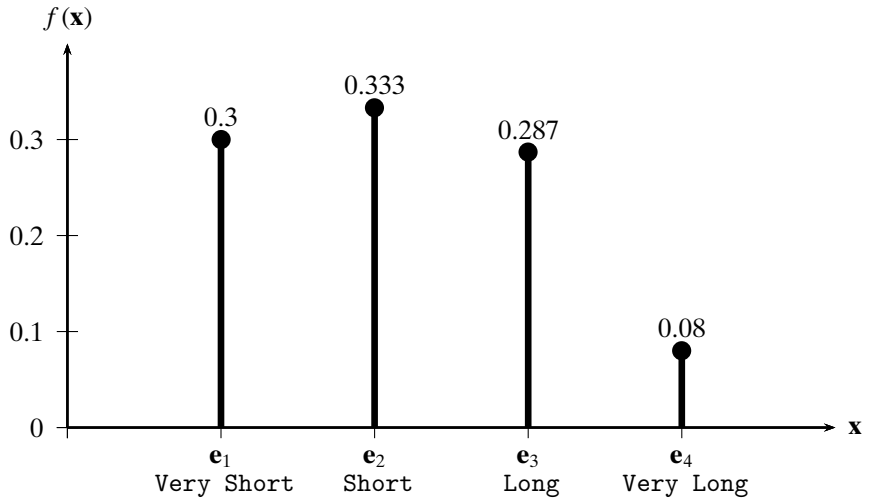


Figure 3.1. Probability mass function: `sepal length`.

$\sum_{i=1}^{m} n_i = n$, which follows from the fact that $\mathbf{X}$ can take on only m distinct values $\mathbf{e}_i$, and the counts for each value must add up to the sample size n.

Example 3.4 (Sample Mean). Consider the observed counts n_i for each of the values a_i ($\mathbf{e}_i$) of the discretized `sepal length` attribute, shown in Table 3.1. Because the total sample size is $n = 150$, from these we can obtain the estimates $\hat{p}_i$ as follows:

$$\hat{p}_1 = 45/150 = 0.3$$
$$\hat{p}_2 = 50/150 = 0.333$$
$$\hat{p}_3 = 43/150 = 0.287$$
$$\hat{p}_4 = 12/150 = 0.08$$

The PMF for $\mathbf{X}$ is plotted in Figure 3.1, and the sample mean for $\mathbf{X}$ is given as

$$\hat{\boldsymbol{\mu}} = \hat{\mathbf{p}} = \begin{pmatrix} 0.3 \\ 0.333 \\ 0.287 \\ 0.08 \end{pmatrix}$$

Covariance Matrix

Recall that an m-dimensional multivariate Bernoulli variable is simply a vector of m Bernoulli variables. For instance, $\mathbf{X} = (A_1, A_2, \ldots, A_m)^T$, where A_i is the Bernoulli variable corresponding to symbol a_i. The variance–covariance information between the constituent Bernoulli variables yields a covariance matrix for $\mathbf{X}$.

Let us first consider the variance along each Bernoulli variable A_i. By Eq. (3.1), we immediately have

$$\sigma_i^2 = var(A_i) = p_i(1 - p_i)$$

Next consider the covariance between A_i and A_j. Utilizing the identity in Eq. (2.21), we have

$$\sigma_{ij} = E[A_i A_j] - E[A_i] \cdot E[A_j] = 0 - p_i p_j = -p_i p_j$$

which follows from the fact that $E[A_i A_j] = 0$, as A_i and A_j cannot both be 1 at the same time, and thus their product $A_i A_j = 0$. This same fact leads to the negative relationship between A_i and A_j. What is interesting is that the degree of negative association is proportional to the product of the mean values for A_i and A_j.

From the preceding expressions for variance and covariance, the $m \times m$ covariance matrix for $\mathbf{X}$ is given as

$$\mathbf{\Sigma} = \begin{pmatrix} \sigma_1^2 & \sigma_{12} & \ldots & \sigma_{1m} \\ \sigma_{12} & \sigma_2^2 & \ldots & \sigma_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{1m} & \sigma_{2m} & \ldots & \sigma_m^2 \end{pmatrix} = \begin{pmatrix} p_1(1-p_1) & -p_1 p_2 & \cdots & -p_1 p_m \\ -p_1 p_2 & p_2(1-p_2) & \cdots & -p_2 p_m \\ \vdots & \vdots & \ddots & \vdots \\ -p_1 p_m & -p_2 p_m & \cdots & p_m(1-p_m) \end{pmatrix}$$

Notice how each row in $\mathbf{\Sigma}$ sums to zero. For example, for row i, we have

$$-p_i p_1 - p_i p_2 - \cdots + p_i(1 - p_i) - \cdots - p_i p_m = p_i - p_i \sum_{j=1}^{m} p_j = p_i - p_i = 0 \tag{3.7}$$

Because $\mathbf{\Sigma}$ is symmetric, it follows that each column also sums to zero.

Define $\mathbf{P}$ as the $m \times m$ diagonal matrix:

$$\mathbf{P} = \text{diag}(\mathbf{p}) = \text{diag}(p_1, p_2, \ldots, p_m) = \begin{pmatrix} p_1 & 0 & \cdots & 0 \\ 0 & p_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & p_m \end{pmatrix}$$

We can compactly write the covariance matrix of $\mathbf{X}$ as

$$\mathbf{\Sigma} = \mathbf{P} - \mathbf{p} \cdot \mathbf{p}^T \tag{3.8}$$

Sample Covariance Matrix

The sample covariance matrix can be obtained from Eq. (3.8) in a straightforward manner:

$$\widehat{\mathbf{\Sigma}} = \widehat{\mathbf{P}} - \hat{\mathbf{p}} \cdot \hat{\mathbf{p}}^T \tag{3.9}$$

where $\widehat{\mathbf{P}} = \text{diag}(\hat{\mathbf{p}})$, and $\hat{\mathbf{p}} = \hat{\boldsymbol{\mu}} = (\hat{p}_1, \hat{p}_2, \ldots, \hat{p}_m)^T$ denotes the empirical probability mass function for $\mathbf{X}$.

Example 3.5. Returning to the discretized `sepal length` attribute in Example 3.4, we have $\hat{\boldsymbol{\mu}} = \hat{\mathbf{p}} = (0.3, 0.333, 0.287, 0.08)^T$. The sample covariance matrix is given as

$$\begin{aligned}
\widehat{\boldsymbol{\Sigma}} &= \widehat{\mathbf{P}} - \hat{\mathbf{p}} \cdot \hat{\mathbf{p}}^T \\
&= \begin{pmatrix} 0.3 & 0 & 0 & 0 \\ 0 & 0.333 & 0 & 0 \\ 0 & 0 & 0.287 & 0 \\ 0 & 0 & 0 & 0.08 \end{pmatrix} - \begin{pmatrix} 0.3 \\ 0.333 \\ 0.287 \\ 0.08 \end{pmatrix} \begin{pmatrix} 0.3 & 0.333 & 0.287 & 0.08 \end{pmatrix} \\
&= \begin{pmatrix} 0.3 & 0 & 0 & 0 \\ 0 & 0.333 & 0 & 0 \\ 0 & 0 & 0.287 & 0 \\ 0 & 0 & 0 & 0.08 \end{pmatrix} - \begin{pmatrix} 0.09 & 0.1 & 0.086 & 0.024 \\ 0.1 & 0.111 & 0.096 & 0.027 \\ 0.086 & 0.096 & 0.082 & 0.023 \\ 0.024 & 0.027 & 0.023 & 0.006 \end{pmatrix} \\
&= \begin{pmatrix} 0.21 & -0.1 & -0.086 & -0.024 \\ -0.1 & 0.222 & -0.096 & -0.027 \\ -0.086 & -0.096 & 0.204 & -0.023 \\ -0.024 & -0.027 & -0.023 & 0.074 \end{pmatrix}
\end{aligned}$$

One can verify that each row (and column) in $\widehat{\boldsymbol{\Sigma}}$ sums to zero.

It is worth emphasizing that whereas the modeling of categorical attribute X as a multivariate Bernoulli variable, $\mathbf{X} = (A_1, A_2, \ldots, A_m)^T$, makes the structure of the mean and covariance matrix explicit, the same results would be obtained if we simply treat the mapped values $\mathbf{X}(x_i)$ as a new $n \times m$ binary data matrix, and apply the standard definitions of the mean and covariance matrix from multivariate numeric attribute analysis (see Section 2.3). In essence, the mapping from symbols a_i to binary vectors $\mathbf{e}_i$ is the key idea in categorical attribute analysis.

Example 3.6. Consider the sample $\mathbf{D}$ of size $n = 5$ for the `sepal length` attribute X_1 in the Iris dataset, shown in Table 3.2a. As in Example 3.1, we assume that X_1 has only two categorical values {`Long`, `Short`}. We model X_1 as the multivariate Bernoulli variable $\mathbf{X}_1$ defined as

$$\mathbf{X}_1(v) = \begin{cases} \mathbf{e}_1 = (1,0)^T & \text{if } v = \texttt{Long}(a_1) \\ \mathbf{e}_2 = (0,1)^T & \text{if } v = \texttt{Short}(a_2) \end{cases}$$

The sample mean [Eq. (3.6)] is

$$\hat{\boldsymbol{\mu}} = \hat{\mathbf{p}} = (2/5, 3/5)^T = (0.4, 0.6)^T$$

and the sample covariance matrix [Eq. (3.9)] is

$$\begin{aligned}
\widehat{\boldsymbol{\Sigma}} = \widehat{\mathbf{P}} - \hat{\mathbf{p}}\hat{\mathbf{p}}^T &= \begin{pmatrix} 0.4 & 0 \\ 0 & 0.6 \end{pmatrix} - \begin{pmatrix} 0.4 \\ 0.6 \end{pmatrix} \begin{pmatrix} 0.4 & 0.6 \end{pmatrix} \\
&= \begin{pmatrix} 0.4 & 0 \\ 0 & 0.6 \end{pmatrix} - \begin{pmatrix} 0.16 & 0.24 \\ 0.24 & 0.36 \end{pmatrix} = \begin{pmatrix} 0.24 & -0.24 \\ -0.24 & 0.24 \end{pmatrix}
\end{aligned}$$

Table 3.2. (a) Categorical dataset. (b) Mapped binary dataset. (c) Centered dataset.

(a)

	X
x_1	Short
x_2	Short
x_3	Long
x_4	Short
x_5	Long

(b)

	A_1	A_2
$\mathbf{x}_1$	0	1
$\mathbf{x}_2$	0	1
$\mathbf{x}_3$	1	0
$\mathbf{x}_4$	0	1
$\mathbf{x}_5$	1	0

(c)

	Z_1	Z_2
$\mathbf{z}_1$	−0.4	0.4
$\mathbf{z}_2$	−0.4	0.4
$\mathbf{z}_3$	0.6	−0.6
$\mathbf{z}_4$	−0.4	0.4
$\mathbf{z}_5$	0.6	−0.6

To show that the same result would be obtained via standard numeric analysis, we map the categorical attribute X to the two Bernoulli attributes A_1 and A_2 corresponding to symbols Long and Short, respectively. The mapped dataset is shown in Table 3.2b. The sample mean is simply

$$\hat{\boldsymbol{\mu}} = \frac{1}{5}\sum_{i=1}^{5}\mathbf{x}_i = \frac{1}{5}(2,3)^T = (0.4, 0.6)^T$$

Next, we center the dataset by subtracting the mean value from each attribute. After centering, the mapped dataset is as shown in Table 3.2c, with attribute Z_i as the centered attribute A_i. We can compute the covariance matrix using the inner-product form [Eq. (2.30)] on the centered column vectors. We have

$$\sigma_1^2 = \frac{1}{5}Z_1^T Z_1 = 1.2/5 = 0.24$$

$$\sigma_2^2 = \frac{1}{5}Z_2^T Z_2 = 1.2/5 = 0.24$$

$$\sigma_{12} = \frac{1}{5}Z_1^T Z_2 = -1.2/5 = -0.24$$

Thus, the sample covariance matrix is given as

$$\hat{\boldsymbol{\Sigma}} = \begin{pmatrix} 0.24 & -0.24 \\ -0.24 & 0.24 \end{pmatrix}$$

which matches the result obtained by using the multivariate Bernoulli modeling approach.

Multinomial Distribution: Number of Occurrences

Given a multivariate Bernoulli variable $\mathbf{X}$ and a random sample $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ drawn from $\mathbf{X}$. Let N_i be the random variable corresponding to the number of occurrences of symbol a_i in the sample, and let $\mathbf{N} = (N_1, N_2, \ldots, N_m)^T$ denote the vector random variable corresponding to the joint distribution of the number of occurrences over all the symbols. Then $\mathbf{N}$ has a multinomial distribution, given as

$$f\big(\mathbf{N} = (n_1, n_2, \ldots, n_m) \mid \mathbf{p}\big) = \binom{n}{n_1 n_2 \ldots n_m}\prod_{i=1}^{m} p_i^{n_i}$$

We can see that this is a direct generalization of the binomial distribution in Eq. (3.3). The term

$$\binom{n}{n_1 n_2 \ldots n_m} = \frac{n!}{n_1! n_2! \ldots n_m!}$$

denotes the number of ways of choosing n_i occurrences of each symbol a_i from a sample of size n, with $\sum_{i=1}^{m} n_i = n$.

The mean and covariance matrix of $\mathbf{N}$ are given as n times the mean and covariance matrix of $\mathbf{X}$. That is, the mean of $\mathbf{N}$ is given as

$$\boldsymbol{\mu}_{\mathbf{N}} = E[\mathbf{N}] = nE[\mathbf{X}] = n \cdot \boldsymbol{\mu} = n \cdot \mathbf{p} = \begin{pmatrix} np_1 \\ \vdots \\ np_m \end{pmatrix}$$

and its covariance matrix is given as

$$\boldsymbol{\Sigma}_{\mathbf{N}} = n \cdot (\mathbf{P} - \mathbf{p}\mathbf{p}^T) = \begin{pmatrix} np_1(1-p_1) & -np_1p_2 & \cdots & -np_1p_m \\ -np_1p_2 & np_2(1-p_2) & \cdots & -np_2p_m \\ \vdots & \vdots & \ddots & \vdots \\ -np_1p_m & -np_2p_m & \cdots & np_m(1-p_m) \end{pmatrix}$$

Likewise the sample mean and covariance matrix for $\mathbf{N}$ are given as

$$\hat{\boldsymbol{\mu}}_{\mathbf{N}} = n\hat{\mathbf{p}} \qquad\qquad \hat{\boldsymbol{\Sigma}}_{\mathbf{N}} = n\left(\hat{\mathbf{P}} - \hat{\mathbf{p}}\hat{\mathbf{p}}^T\right)$$

3.2 BIVARIATE ANALYSIS

Assume that the data comprises two categorical attributes, X_1 and X_2, with

$$dom(X_1) = \{a_{11}, a_{12}, \ldots, a_{1m_1}\}$$
$$dom(X_2) = \{a_{21}, a_{22}, \ldots, a_{2m_2}\}$$

We are given n categorical points of the form $\mathbf{x}_i = (x_{i1}, x_{i2})^T$ with $x_{i1} \in dom(X_1)$ and $x_{i2} \in dom(X_2)$. The dataset is thus an $n \times 2$ symbolic data matrix:

$$\mathbf{D} = \begin{pmatrix} X_1 & X_2 \\ \hline x_{11} & x_{12} \\ x_{21} & x_{22} \\ \vdots & \vdots \\ x_{n1} & x_{n2} \end{pmatrix}$$

We can model X_1 and X_2 as multivariate Bernoulli variables $\mathbf{X}_1$ and $\mathbf{X}_2$ with dimensions m_1 and m_2, respectively. The probability mass functions for $\mathbf{X}_1$ and $\mathbf{X}_2$ are

given according to Eq. (3.4):

$$P(\mathbf{X}_1 = \mathbf{e}_{1i}) = f_1(\mathbf{e}_{1i}) = p_i^1 = \prod_{k=1}^{m_1} (p_i^1)^{e_{ik}^1}$$

$$P(\mathbf{X}_2 = \mathbf{e}_{2j}) = f_2(\mathbf{e}_{2j}) = p_j^2 = \prod_{k=1}^{m_2} (p_j^2)^{e_{jk}^2}$$

where $\mathbf{e}_{1i}$ is the ith standard basis vector in $\mathbb{R}^{m_1}$ (for attribute X_1) whose kth component is e_{ik}^1, and $\mathbf{e}_{2j}$ is the jth standard basis vector in $\mathbb{R}^{m_2}$ (for attribute X_2) whose kth component is e_{jk}^2. Further, the parameter p_i^1 denotes the probability of observing symbol a_{1i}, and p_j^2 denotes the probability of observing symbol a_{2j}. Together they must satisfy the conditions: $\sum_{i=1}^{m_1} p_i^1 = 1$ and $\sum_{j=1}^{m_2} p_j^2 = 1$.

The joint distribution of $\mathbf{X}_1$ and $\mathbf{X}_2$ is modeled as the $d' = m_1 + m_2$ dimensional vector variable $\mathbf{X} = \begin{pmatrix} \mathbf{X}_1 \\ \mathbf{X}_2 \end{pmatrix}$, specified by the mapping

$$\mathbf{X}\left((v_1, v_2)^T\right) = \begin{pmatrix} \mathbf{X}_1(v_1) \\ \mathbf{X}_2(v_2) \end{pmatrix} = \begin{pmatrix} \mathbf{e}_{1i} \\ \mathbf{e}_{2j} \end{pmatrix}$$

provided that $v_1 = a_{1i}$ and $v_2 = a_{2j}$. The range of $\mathbf{X}$ thus consists of $m_1 \times m_2$ distinct pairs of vector values $\left\{(\mathbf{e}_{1i}, \mathbf{e}_{2j})^T\right\}$, with $1 \le i \le m_1$ and $1 \le j \le m_2$. The joint PMF of $\mathbf{X}$ is given as

$$P\left(\mathbf{X} = (\mathbf{e}_{1i}, \mathbf{e}_{2j})^T\right) = f(\mathbf{e}_{1i}, \mathbf{e}_{2j}) = p_{ij} = \prod_{r=1}^{m_1} \prod_{s=1}^{m_2} p_{ij}^{e_{ir}^1 \cdot e_{js}^2}$$

where p_{ij} the probability of observing the symbol pair (a_{1i}, a_{2j}). These probability parameters must satisfy the condition $\sum_{i=1}^{m_1} \sum_{j=1}^{m_2} p_{ij} = 1$. The joint PMF for $\mathbf{X}$ can be expressed as the $m_1 \times m_2$ matrix

$$\mathbf{P}_{12} = \begin{pmatrix} p_{11} & p_{12} & \cdots & p_{1m_2} \\ p_{21} & p_{22} & \cdots & p_{2m_2} \\ \vdots & \vdots & \ddots & \vdots \\ p_{m_1 1} & p_{m_1 2} & \cdots & p_{m_1 m_2} \end{pmatrix} \tag{3.10}$$

Example 3.7. Consider the discretized `sepal length` attribute (X_1) in Table 3.1. We also discretize the `sepal width` attribute (X_2) into three values as shown in Table 3.3. We thus have

$$dom(X_1) = \{a_{11} = \texttt{VeryShort}, a_{12} = \texttt{Short}, a_{13} = \texttt{Long}, a_{14} = \texttt{VeryLong}\}$$

$$dom(X_2) = \{a_{21} = \texttt{Short}, a_{22} = \texttt{Medium}, a_{23} = \texttt{Long}\}$$

The symbolic point $\mathbf{x} = (\texttt{Short}, \texttt{Long}) = (a_{12}, a_{23})$, is mapped to the vector

$$\mathbf{X}(\mathbf{x}) = \begin{pmatrix} \mathbf{e}_{12} \\ \mathbf{e}_{23} \end{pmatrix} = (0, 1, 0, 0 \mid 0, 0, 1)^T \in \mathbb{R}^7$$

Table 3.3. Discretized `sepal width` attribute

Bins	Domain	Counts
[2.0, 2.8]	Short (a_1)	47
(2.8, 3.6]	Medium (a_2)	88
(3.6, 4.4]	Long (a_3)	15

where we use | to demarcate the two subvectors $\mathbf{e}_{12} = (0,1,0,0)^T \in \mathbb{R}^4$ and $\mathbf{e}_{23} = (0,0,1)^T \in \mathbb{R}^3$, corresponding to symbolic attributes `sepal length` and `sepal width`, respectively. Note that $\mathbf{e}_{12}$ is the second standard basis vector in $\mathbb{R}^4$ for $\mathbf{X}_1$, and $\mathbf{e}_{23}$ is the third standard basis vector in $\mathbb{R}^3$ for $\mathbf{X}_2$.

Mean

The bivariate mean can easily be generalized from Eq. (3.5), as follows:

$$\boldsymbol{\mu} = E[\mathbf{X}] = E\left[\begin{pmatrix}\mathbf{X}_1 \\ \mathbf{X}_2\end{pmatrix}\right] = \begin{pmatrix}E[\mathbf{X}_1] \\ E[\mathbf{X}_2]\end{pmatrix} = \begin{pmatrix}\boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2\end{pmatrix} = \begin{pmatrix}\mathbf{p}_1 \\ \mathbf{p}_2\end{pmatrix}$$

where $\boldsymbol{\mu}_1 = \mathbf{p}_1 = (p_1^1, \ldots, p_{m_1}^1)^T$ and $\boldsymbol{\mu}_2 = \mathbf{p}_2 = (p_1^2, \ldots, p_{m_2}^2)^T$ are the mean vectors for $\mathbf{X}_1$ and $\mathbf{X}_2$. The vectors $\mathbf{p}_1$ and $\mathbf{p}_2$ also represent the probability mass functions for $\mathbf{X}_1$ and $\mathbf{X}_2$, respectively.

Sample Mean

The sample mean can also be generalized from Eq. (3.6), by placing a probability mass of $\frac{1}{n}$ at each point:

$$\hat{\boldsymbol{\mu}} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i = \frac{1}{n}\begin{pmatrix}\sum_{i=1}^{m_1} n_i^1 \mathbf{e}_{1i} \\ \sum_{j=1}^{m_2} n_j^2 \mathbf{e}_{2j}\end{pmatrix} = \frac{1}{n}\begin{pmatrix}n_1^1 \\ \vdots \\ n_{m_1}^1 \\ n_1^2 \\ \vdots \\ n_{m_2}^2\end{pmatrix} = \begin{pmatrix}\hat{p}_1^1 \\ \vdots \\ \hat{p}_{m_1}^1 \\ \hat{p}_1^2 \\ \vdots \\ \hat{p}_{m_2}^2\end{pmatrix} = \begin{pmatrix}\hat{\mathbf{p}}_1 \\ \hat{\mathbf{p}}_2\end{pmatrix} = \begin{pmatrix}\hat{\boldsymbol{\mu}}_1 \\ \hat{\boldsymbol{\mu}}_2\end{pmatrix}$$

where n_j^i is the observed frequency of symbol a_{ij} in the sample of size n, and $\hat{\boldsymbol{\mu}}_i = \hat{\mathbf{p}}_i = (p_1^i, p_2^i, \ldots, p_{m_i}^i)^T$ is the sample mean vector for $\mathbf{X}_i$, which is also the empirical PMF for attribute $\mathbf{X}_i$.

Covariance Matrix

The covariance matrix for $\mathbf{X}$ is the $d' \times d' = (m_1 + m_2) \times (m_1 + m_2)$ matrix given as

$$\boldsymbol{\Sigma} = \begin{pmatrix}\boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{12}^T & \boldsymbol{\Sigma}_{22}\end{pmatrix} \tag{3.11}$$

where $\boldsymbol{\Sigma}_{11}$ is the $m_1 \times m_1$ covariance matrix for $\mathbf{X}_1$, and $\boldsymbol{\Sigma}_{22}$ is the $m_2 \times m_2$ covariance matrix for $\mathbf{X}_2$, which can be computed using Eq. (3.8). That is,

$$\boldsymbol{\Sigma}_{11} = \mathbf{P}_1 - \mathbf{p}_1\mathbf{p}_1^T$$
$$\boldsymbol{\Sigma}_{22} = \mathbf{P}_2 - \mathbf{p}_2\mathbf{p}_2^T$$

where $\mathbf{P}_1 = \text{diag}(\mathbf{p}_1)$ and $\mathbf{P}_2 = \text{diag}(\mathbf{p}_2)$. Further, $\mathbf{\Sigma}_{12}$ is the $m_1 \times m_2$ covariance matrix between variables $\mathbf{X}_1$ and $\mathbf{X}_2$, given as

$$\begin{aligned}
\mathbf{\Sigma}_{12} &= E[(\mathbf{X}_1 - \boldsymbol{\mu}_1)(\mathbf{X}_2 - \boldsymbol{\mu}_2)^T] \\
&= E[\mathbf{X}_1\mathbf{X}_2^T] - E[\mathbf{X}_1]E[\mathbf{X}_2]^T \\
&= \mathbf{P}_{12} - \boldsymbol{\mu}_1\boldsymbol{\mu}_2^T \\
&= \mathbf{P}_{12} - \mathbf{p}_1\mathbf{p}_2^T \\
&= \begin{pmatrix} p_{11} - p_1^1 p_1^2 & p_{12} - p_1^1 p_2^2 & \cdots & p_{1m_2} - p_1^1 p_{m_2}^2 \\ p_{21} - p_2^1 p_1^2 & p_{22} - p_2^1 p_2^2 & \cdots & p_{2m_2} - p_2^1 p_{m_2}^2 \\ \vdots & \vdots & \ddots & \vdots \\ p_{m_1 1} - p_{m_1}^1 p_1^2 & p_{m_1 2} - p_{m_1}^1 p_2^2 & \cdots & p_{m_1 m_2} - p_{m_1}^1 p_{m_2}^2 \end{pmatrix}
\end{aligned}$$

where $\mathbf{P}_{12}$ represents the joint PMF for $\mathbf{X}$ given in Eq. (3.10).

Incidentally, each row and each column of $\mathbf{\Sigma}_{12}$ sums to zero. For example, consider row i and column j:

$$\sum_{k=1}^{m_2}(p_{ik} - p_i^1 p_k^2) = \left(\sum_{k=1}^{m_2} p_{ik}\right) - p_i^1 = p_i^1 - p_i^1 = 0$$

$$\sum_{k=1}^{m_1}(p_{kj} - p_k^1 p_j^2) = \left(\sum_{k=1}^{m_1} p_{kj}\right) - p_j^2 = p_j^2 - p_j^2 = 0$$

which follows from the fact that summing the joint mass function over all values of $\mathbf{X}_2$, yields the marginal distribution of $\mathbf{X}_1$, and summing it over all values of $\mathbf{X}_1$ yields the marginal distribution for $\mathbf{X}_2$. Note that p_j^2 is the probability of observing symbol a_{2j}; it should not be confused with the square of p_j. Combined with the fact that $\mathbf{\Sigma}_{11}$ and $\mathbf{\Sigma}_{22}$ also have row and column sums equal to zero via Eq. (3.7), the full covariance matrix $\mathbf{\Sigma}$ has rows and columns that sum up to zero.

Sample Covariance Matrix

The sample covariance matrix is given as

$$\widehat{\mathbf{\Sigma}} = \begin{pmatrix} \widehat{\mathbf{\Sigma}}_{11} & \widehat{\mathbf{\Sigma}}_{12} \\ \widehat{\mathbf{\Sigma}}_{12}^T & \widehat{\mathbf{\Sigma}}_{22} \end{pmatrix} \tag{3.12}$$

where

$$\begin{aligned}
\widehat{\mathbf{\Sigma}}_{11} &= \widehat{\mathbf{P}}_1 - \hat{\mathbf{p}}_1\hat{\mathbf{p}}_1^T \\
\widehat{\mathbf{\Sigma}}_{22} &= \widehat{\mathbf{P}}_2 - \hat{\mathbf{p}}_2\hat{\mathbf{p}}_2^T \\
\widehat{\mathbf{\Sigma}}_{12} &= \widehat{\mathbf{P}}_{12} - \hat{\mathbf{p}}_1\hat{\mathbf{p}}_2^T
\end{aligned}$$

Here $\widehat{\mathbf{P}}_1 = \text{diag}(\hat{\mathbf{p}}_1)$ and $\widehat{\mathbf{P}}_2 = \text{diag}(\hat{\mathbf{p}}_2)$, and $\hat{\mathbf{p}}_1$ and $\hat{\mathbf{p}}_2$ specify the empirical probability mass functions for $\mathbf{X}_1$, and $\mathbf{X}_2$, respectively. Further, $\widehat{\mathbf{P}}_{12}$ specifies the empirical joint PMF for $\mathbf{X}_1$ and $\mathbf{X}_2$, given as

$$\widehat{\mathbf{P}}_{12}(i, j) = \hat{f}(\mathbf{e}_{1i}, \mathbf{e}_{2j}) = \frac{1}{n}\sum_{k=1}^{n} I_{ij}(\mathbf{x}_k) = \frac{n_{ij}}{n} = \hat{p}_{ij} \tag{3.13}$$

where I_{ij} is the indicator variable

$$I_{ij}(\mathbf{x}_k) = \begin{cases} 1 & \text{if } \mathbf{x}_{k1} = \mathbf{e}_{1i} \text{ and } \mathbf{x}_{k2} = \mathbf{e}_{2j} \\ 0 & \text{otherwise} \end{cases}$$

Taking the sum of $I_{ij}(\mathbf{x}_k)$ over all the n points in the sample yields the number of occurrences, n_{ij}, of the symbol pair (a_{1i}, a_{2j}) in the sample. One issue with the cross-attribute covariance matrix $\widehat{\boldsymbol{\Sigma}}_{12}$ is the need to estimate a quadratic number of parameters. That is, we need to obtain reliable counts n_{ij} to estimate the parameters p_{ij}, for a total of $O(m_1 \times m_2)$ parameters that have to be estimated, which can be a problem if the categorical attributes have many symbols. On the other hand, estimating $\widehat{\boldsymbol{\Sigma}}_{11}$ and $\widehat{\boldsymbol{\Sigma}}_{22}$ requires that we estimate m_1 and m_2 parameters, corresponding to p_i^1 and p_j^2, respectively. In total, computing $\boldsymbol{\Sigma}$ requires the estimation of $m_1 m_2 + m_1 + m_2$ parameters.

Example 3.8. We continue with the bivariate categorical attributes X_1 and X_2 in Example 3.7. From Example 3.4, and from the occurrence counts for each of the values of `sepal width` in Table 3.3, we have

$$\hat{\boldsymbol{\mu}}_1 = \hat{\mathbf{p}}_1 = \begin{pmatrix} 0.3 \\ 0.333 \\ 0.287 \\ 0.08 \end{pmatrix} \qquad \hat{\boldsymbol{\mu}}_2 = \hat{\mathbf{p}}_2 = \frac{1}{150}\begin{pmatrix} 47 \\ 88 \\ 15 \end{pmatrix} = \begin{pmatrix} 0.313 \\ 0.587 \\ 0.1 \end{pmatrix}$$

Thus, the mean for $\mathbf{X} = \begin{pmatrix} \mathbf{X}_1 \\ \mathbf{X}_2 \end{pmatrix}$ is given as

$$\hat{\boldsymbol{\mu}} = \begin{pmatrix} \hat{\boldsymbol{\mu}}_1 \\ \hat{\boldsymbol{\mu}}_2 \end{pmatrix} = \begin{pmatrix} \hat{\mathbf{p}}_1 \\ \hat{\mathbf{p}}_2 \end{pmatrix} = (0.3, 0.333, 0.287, 0.08 \mid 0.313, 0.587, 0.1)^T$$

From Example 3.5 we have

$$\widehat{\boldsymbol{\Sigma}}_{11} = \begin{pmatrix} 0.21 & -0.1 & -0.086 & -0.024 \\ -0.1 & 0.222 & -0.096 & -0.027 \\ -0.086 & -0.096 & 0.204 & -0.023 \\ -0.024 & -0.027 & -0.023 & 0.074 \end{pmatrix}$$

In a similar manner we can obtain

$$\widehat{\boldsymbol{\Sigma}}_{22} = \begin{pmatrix} 0.215 & -0.184 & -0.031 \\ -0.184 & 0.242 & -0.059 \\ -0.031 & -0.059 & 0.09 \end{pmatrix}$$

Next, we use the observed counts in Table 3.4 to obtain the empirical joint PMF for $\mathbf{X}_1$ and $\mathbf{X}_2$ using Eq. (3.13), as plotted in Figure 3.2. From these probabilities we get

$$E[\mathbf{X}_1 \mathbf{X}_2^T] = \widehat{\mathbf{P}}_{12} = \frac{1}{150}\begin{pmatrix} 7 & 33 & 5 \\ 24 & 18 & 8 \\ 13 & 30 & 0 \\ 3 & 7 & 2 \end{pmatrix} = \begin{pmatrix} 0.047 & 0.22 & 0.033 \\ 0.16 & 0.12 & 0.053 \\ 0.087 & 0.2 & 0 \\ 0.02 & 0.047 & 0.013 \end{pmatrix}$$

Table 3.4. Observed Counts (n_{ij}): `sepal length` and `sepal width`

		X_2		
		`Short` ($\mathbf{e}_{21}$)	`Medium` ($\mathbf{e}_{22}$)	`Long` ($\mathbf{e}_{23}$)
X_1	`Very Short` ($\mathbf{e}_{11}$)	7	33	5
	`Short` ($\mathbf{e}_{22}$)	24	18	8
	`Long` ($\mathbf{e}_{13}$)	13	30	0
	`Very Long` ($\mathbf{e}_{14}$)	3	7	2

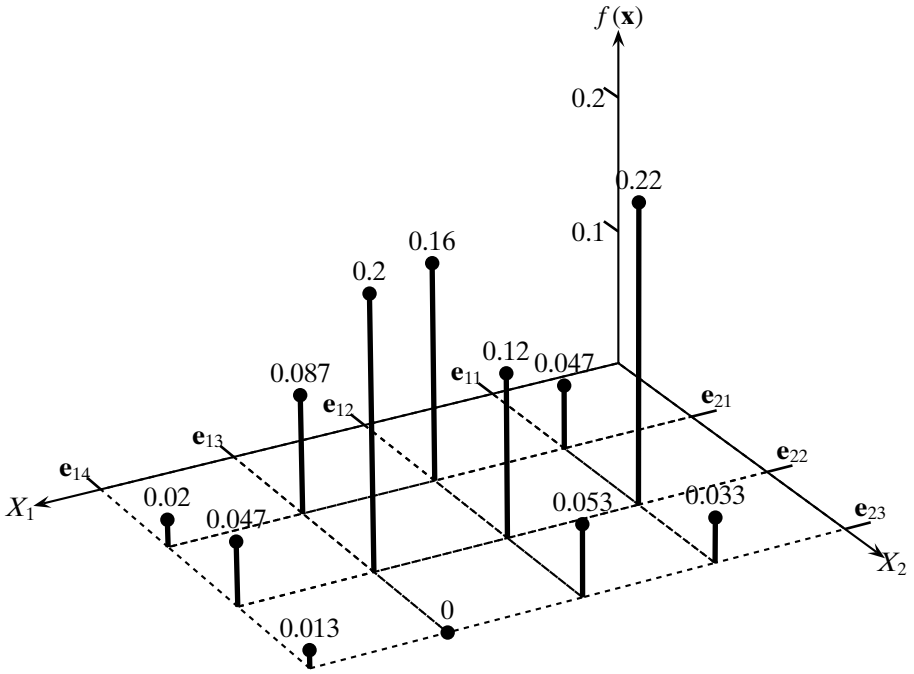


Figure 3.2. Empirical joint probability mass function: `sepal length` and `sepal width`.

Further, we have

$$E[\mathbf{X}_1]E[\mathbf{X}_2]^T = \hat{\boldsymbol{\mu}}_1\hat{\boldsymbol{\mu}}_2^T = \hat{\mathbf{p}}_1\hat{\mathbf{p}}_2^T$$

$$= \begin{pmatrix} 0.3 \\ 0.333 \\ 0.287 \\ 0.08 \end{pmatrix} \begin{pmatrix} 0.313 & 0.587 & 0.1 \end{pmatrix}$$

$$= \begin{pmatrix} 0.094 & 0.176 & 0.03 \\ 0.104 & 0.196 & 0.033 \\ 0.09 & 0.168 & 0.029 \\ 0.025 & 0.047 & 0.008 \end{pmatrix}$$

We can now compute the across-attribute sample covariance matrix $\widehat{\boldsymbol{\Sigma}}_{12}$ for $\mathbf{X}_1$ and $\mathbf{X}_2$ using Eq. (3.11), as follows:

$$\widehat{\boldsymbol{\Sigma}}_{12} = \widehat{\mathbf{P}}_{12} - \hat{\mathbf{p}}_1 \hat{\mathbf{p}}_2^T = \begin{pmatrix} -0.047 & 0.044 & 0.003 \\ 0.056 & -0.076 & 0.02 \\ -0.003 & 0.032 & -0.029 \\ -0.005 & 0 & 0.005 \end{pmatrix}$$

One can observe that each row and column in $\widehat{\boldsymbol{\Sigma}}_{12}$ sums to zero. Putting it all together, from $\widehat{\boldsymbol{\Sigma}}_{11}$, $\widehat{\boldsymbol{\Sigma}}_{22}$ and $\widehat{\boldsymbol{\Sigma}}_{12}$ we obtain the sample covariance matrix as follows

$$\widehat{\boldsymbol{\Sigma}} = \begin{pmatrix} \widehat{\boldsymbol{\Sigma}}_{11} & \widehat{\boldsymbol{\Sigma}}_{12} \\ \widehat{\boldsymbol{\Sigma}}_{12}^T & \widehat{\boldsymbol{\Sigma}}_{22} \end{pmatrix} = \left(\begin{array}{cccc|ccc} 0.21 & -0.1 & -0.086 & -0.024 & -0.047 & 0.044 & 0.003 \\ -0.1 & 0.222 & -0.096 & -0.027 & 0.056 & -0.076 & 0.02 \\ -0.086 & -0.096 & 0.204 & -0.023 & -0.003 & 0.032 & -0.029 \\ -0.024 & -0.027 & -0.023 & 0.074 & -0.005 & 0 & 0.005 \\ \hline -0.047 & 0.056 & -0.003 & -0.005 & 0.215 & -0.184 & -0.031 \\ 0.044 & -0.076 & 0.032 & 0 & -0.184 & 0.242 & -0.059 \\ 0.003 & 0.02 & -0.029 & 0.005 & -0.031 & -0.059 & 0.09 \end{array}\right)$$

In $\widehat{\boldsymbol{\Sigma}}$, each row and column also sums to zero.

3.2.1 Attribute Dependence: Contingency Analysis

Testing for the independence of the two categorical random variables X_1 and X_2 can be done via *contingency table analysis*. The main idea is to set up a hypothesis testing framework, where the null hypothesis H_0 is that $\mathbf{X}_1$ and $\mathbf{X}_2$ are independent, and the alternative hypothesis H_1 is that they are dependent. We then compute the value of the chi-square statistic χ^2 under the null hypothesis. Depending on the p-value, we either accept or reject the null hypothesis; in the latter case the attributes are considered to be dependent.

Contingency Table

A contingency table for $\mathbf{X}_1$ and $\mathbf{X}_2$ is the $m_1 \times m_2$ matrix of observed counts n_{ij} for all pairs of values $(\mathbf{e}_{1i}, \mathbf{e}_{2j})$ in the given sample of size n, defined as

$$\mathbf{N}_{12} = n \cdot \widehat{\mathbf{P}}_{12} = \begin{pmatrix} n_{11} & n_{12} & \cdots & n_{1m_2} \\ n_{21} & n_{22} & \cdots & n_{2m_2} \\ \vdots & \vdots & \ddots & \vdots \\ n_{m_1 1} & n_{m_1 2} & \cdots & n_{m_1 m_2} \end{pmatrix}$$

Table 3.5. Contingency table: `sepal length` vs. `sepal width`

`Sepal length` (X_1)	`Sepal width` (X_2)				
		`Short` a_{21}	`Medium` a_{22}	`Long` a_{23}	Row Counts
	`Very Short` (a_{11})	7	33	5	$n_1^1 = 45$
	`Short` (a_{12})	24	18	8	$n_2^1 = 50$
	`Long` (a_{13})	13	30	0	$n_3^1 = 43$
	`Very Long` (a_{14})	3	7	2	$n_4^1 = 12$
	Column Counts	$n_1^2 = 47$	$n_2^2 = 88$	$n_3^2 = 15$	$n = 150$

where $\widehat{\mathbf{P}}_{12}$ is the empirical joint PMF for $\mathbf{X}_1$ and $\mathbf{X}_2$, computed via Eq. (3.13). The contingency table is then augmented with row and column marginal counts, as follows:

$$\mathbf{N}_1 = n \cdot \hat{\mathbf{p}}_1 = \begin{pmatrix} n_1^1 \\ \vdots \\ n_{m_1}^1 \end{pmatrix} \qquad \mathbf{N}_2 = n \cdot \hat{\mathbf{p}}_2 = \begin{pmatrix} n_1^2 \\ \vdots \\ n_{m_2}^2 \end{pmatrix}$$

Note that the marginal row and column entries and the sample size satisfy the following constraints:

$$n_i^1 = \sum_{j=1}^{m_2} n_{ij} \qquad n_j^2 = \sum_{i=1}^{m_1} n_{ij} \qquad n = \sum_{i=1}^{m_1} n_i^1 = \sum_{j=1}^{m_2} n_j^2 = \sum_{i=1}^{m_1}\sum_{j=1}^{m_2} n_{ij}$$

It is worth noting that both $\mathbf{N}_1$ and $\mathbf{N}_2$ have a multinomial distribution with parameters $\mathbf{p}_1 = (p_1^1, \ldots, p_{m_1}^1)$ and $\mathbf{p}_2 = (p_1^2, \ldots, p_{m_2}^2)$, respectively. Further, $\mathbf{N}_{12}$ also has a multinomial distribution with parameters $\mathbf{P}_{12} = \{p_{ij}\}$, for $1 \le i \le m_1$ and $1 \le j \le m_2$.

Example 3.9 (Contingency Table). Table 3.4 shows the observed counts for the discretized `sepal length` (X_1) and `sepal width` (X_2) attributes. Augmenting the table with the row and column marginal counts and the sample size yields the final contingency table shown in Table 3.5.

χ^2 Statistic and Hypothesis Testing

Under the null hypothesis $\mathbf{X}_1$ and $\mathbf{X}_2$ are assumed to be independent, which means that their joint probability mass function is given as

$$\hat{p}_{ij} = \hat{p}_i^1 \cdot \hat{p}_j^2$$

Under this independence assumption, the expected frequency for each pair of values is given as

$$e_{ij} = n \cdot \hat{p}_{ij} = n \cdot \hat{p}_i^1 \cdot \hat{p}_j^2 = n \cdot \frac{n_i^1}{n} \cdot \frac{n_j^2}{n} = \frac{n_i^1 n_j^2}{n} \tag{3.14}$$

However, from the sample we already have the observed frequency of each pair of values, n_{ij}. We would like to determine whether there is a significant difference in the observed and expected frequencies for each pair of values. If there is no

significant difference, then the independence assumption is valid and we accept the null hypothesis that the attributes are independent. On the other hand, if there is a significant difference, then the null hypothesis should be rejected and we conclude that the attributes are dependent.

The χ^2 statistic quantifies the difference between observed and expected counts for each pair of values; it is defined as follows:

$$\chi^2 = \sum_{i=1}^{m_1} \sum_{j=1}^{m_2} \frac{(n_{ij} - e_{ij})^2}{e_{ij}} \tag{3.15}$$

At this point, we need to determine the probability of obtaining the computed χ^2 value. In general, this can be rather difficult if we do not know the sampling distribution of a given statistic. Fortunately, for the χ^2 statistic it is known that its sampling distribution follows the *chi-squared* density function with q degrees of freedom:

$$f(x|q) = \frac{1}{2^{q/2}\Gamma(q/2)} x^{\frac{q}{2}-1} \mathrm{e}^{-\frac{x}{2}} \tag{3.16}$$

where the gamma function Γ is defined as

$$\Gamma(k > 0) = \int_0^\infty x^{k-1} \mathrm{e}^{-x} dx \tag{3.17}$$

The degrees of freedom, q, represent the number of independent parameters. In the contingency table there are $m_1 \times m_2$ observed counts n_{ij}. However, note that each row i and each column j must sum to n_i^1 and n_j^2, respectively. Further, the sum of the row and column marginals must also add to n; thus we have to remove $(m_1 + m_2)$ parameters from the number of independent parameters. However, doing this removes one of the parameters, say $n_{m_1 m_2}$, twice, so we have to add back one to the count. The total degrees of freedom is therefore

$$\begin{aligned} q &= |dom(X_1)| \times |dom(X_2)| - (|dom(X_1)| + |dom(X_2)|) + 1 \\ &= m_1 m_2 - m_1 - m_2 + 1 \\ &= (m_1 - 1)(m_2 - 1) \end{aligned}$$

p-value

The *p-value* of a statistic θ is defined as the probability of obtaining a value at least as extreme as the observed value, say z, under the null hypothesis, defined as

$$p\text{-value}(z) = P(\theta \geq z) = 1 - F(\theta)$$

where $F(\theta)$ is the cumulative probability distribution for the statistic.

The p-value gives a measure of how surprising is the observed value of the statistic. If the observed value lies in a low-probability region, then the value is more surprising. In general, the lower the p-value, the more surprising the observed value, and the

Table 3.6. Expected counts

		X_2		
		Short (a_{21})	Medium (a_{22})	Short (a_{23})
X_1	Very Short (a_{11})	14.1	26.4	4.5
	Short (a_{12})	15.67	29.33	5.0
	Long (a_{13})	13.47	25.23	4.3
	Very Long (a_{14})	3.76	7.04	1.2

more the grounds for rejecting the null hypothesis. The null hypothesis is rejected if the p-value is below some *significance level*, α. For example, if $\alpha = 0.01$, then we reject the null hypothesis if $p\text{-value}(z) \leq \alpha$. The significance level α corresponds to the probability of rejecting the null hypothesis when it is true. For a given significance level α, the value of the test statistic, say z, with a p-value of $p\text{-value}(z) = \alpha$, is called a *critical value*. An alternative test for rejection of the null hypothesis is to check if $\chi^2 > z$, as in that case the p-value of the observed χ^2 value is bounded by α, that is, $p\text{-value}(\chi^2) \leq p\text{-value}(z) = \alpha$. The value $1 - \alpha$ is also called the *confidence level*.

Example 3.10. Consider the contingency table for `sepal length` and `sepal width` in Table 3.5. We compute the expected counts using Eq. (3.14); these counts are shown in Table 3.6. For example, we have

$$e_{11} = \frac{n_1^1 n_1^2}{n} = \frac{45 \cdot 47}{150} = \frac{2115}{150} = 14.1$$

Next we use Eq. (3.15) to compute the value of the χ^2 statistic, which is given as $\chi^2 = 21.8$.

Further, the number of degrees of freedom is given as

$$q = (m_1 - 1) \cdot (m_2 - 1) = 3 \cdot 2 = 6$$

The plot of the chi-squared density function with 6 degrees of freedom is shown in Figure 3.3. From the cumulative chi-squared distribution, we obtain

$$p\text{-value}(21.8) = 1 - F(21.8|6) = 1 - 0.9987 = 0.0013$$

At a significance level of $\alpha = 0.01$, we would certainly be justified in rejecting the null hypothesis because the large value of the χ^2 statistic is indeed surprising. Further, at the 0.01 significance level, the critical value of the statistic is

$$z = F^{-1}(1 - 0.01|6) = F^{-1}(0.99|6) = 16.81$$

This critical value is also shown in Figure 3.3, and we can clearly see that the observed value of 21.8 is in the rejection region, as $21.8 > z = 16.81$. In effect, we reject the null hypothesis that `sepal length` and `sepal width` are independent, and accept the alternative hypothesis that they are dependent.

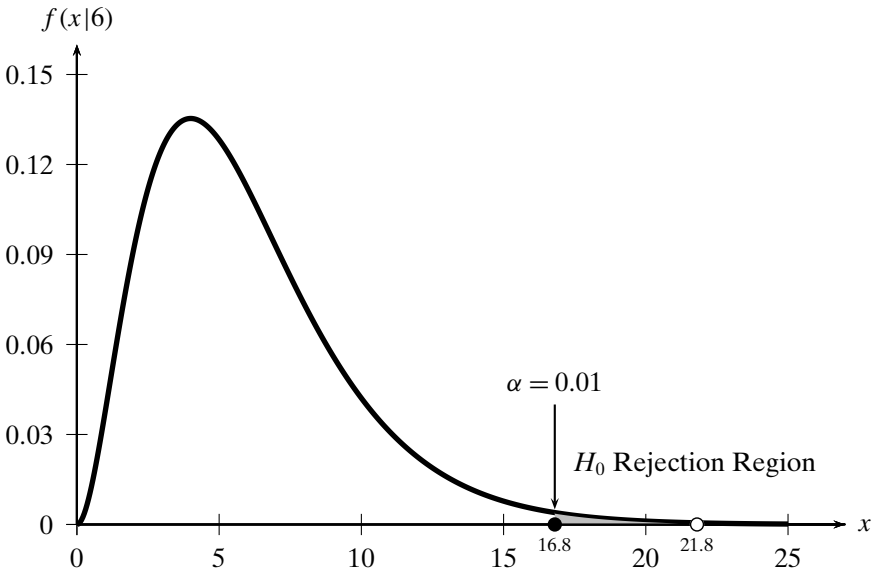


Figure 3.3. Chi-squared distribution ($q = 6$).

3.3 MULTIVARIATE ANALYSIS

Assume that the dataset comprises d categorical attributes X_j $(1 \leq j \leq d)$ with $dom(X_j) = \{a_{j1}, a_{j2}, \ldots, a_{jm_j}\}$. We are given n categorical points of the form $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{id})^T$ with $x_{ij} \in dom(X_j)$. The dataset is thus an $n \times d$ symbolic matrix

$$\mathbf{D} = \begin{pmatrix} X_1 & X_2 & \cdots & X_d \\ \hline x_{11} & x_{12} & \cdots & x_{1d} \\ x_{21} & x_{22} & \cdots & x_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nd} \end{pmatrix}$$

Each attribute X_i is modeled as an m_i-dimensional multivariate Bernoulli variable $\mathbf{X}_i$, and their joint distribution is modeled as a $d' = \sum_{j=1}^{d} m_j$ dimensional vector random variable

$$\mathbf{X} = \begin{pmatrix} \mathbf{X}_1 \\ \vdots \\ \mathbf{X}_d \end{pmatrix}$$

Each categorical data point $\mathbf{v} = (v_1, v_2, \ldots, v_d)^T$ is therefore represented as a d'-dimensional binary vector

$$\mathbf{X}(\mathbf{v}) = \begin{pmatrix} \mathbf{X}_1(v_1) \\ \vdots \\ \mathbf{X}_d(v_d) \end{pmatrix} = \begin{pmatrix} \mathbf{e}_{1k_1} \\ \vdots \\ \mathbf{e}_{dk_d} \end{pmatrix}$$

provided $v_i = a_{ik_i}$, the k_ith symbol of X_i. Here $\mathbf{e}_{ik_i}$ is the k_ith standard basis vector in $\mathbb{R}^{m_i}$.

Mean

Generalizing from the bivariate case, the mean and sample mean for $\mathbf{X}$ are given as

$$\boldsymbol{\mu} = E[\mathbf{X}] = \begin{pmatrix} \boldsymbol{\mu}_1 \\ \vdots \\ \boldsymbol{\mu}_d \end{pmatrix} = \begin{pmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_d \end{pmatrix} \qquad \hat{\boldsymbol{\mu}} = \begin{pmatrix} \hat{\boldsymbol{\mu}}_1 \\ \vdots \\ \hat{\boldsymbol{\mu}}_d \end{pmatrix} = \begin{pmatrix} \hat{\mathbf{p}}_1 \\ \vdots \\ \hat{\mathbf{p}}_d \end{pmatrix}$$

where $\mathbf{p}_i = (p_1^i, \ldots, p_{m_i}^i)^T$ is the PMF for $\mathbf{X}_i$, and $\hat{\mathbf{p}}_i = (\hat{p}_1^i, \ldots, \hat{p}_{m_i}^i)^T$ is the empirical PMF for $\mathbf{X}_i$.

Covariance Matrix

The covariance matrix for $\mathbf{X}$, and its estimate from the sample, are given as the $d' \times d'$ matrices:

$$\boldsymbol{\Sigma} = \begin{pmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} & \cdots & \boldsymbol{\Sigma}_{1d} \\ \boldsymbol{\Sigma}_{12}^T & \boldsymbol{\Sigma}_{22} & \cdots & \boldsymbol{\Sigma}_{2d} \\ \cdots & \cdots & \ddots & \cdots \\ \boldsymbol{\Sigma}_{1d}^T & \boldsymbol{\Sigma}_{2d}^T & \cdots & \boldsymbol{\Sigma}_{dd} \end{pmatrix} \qquad \widehat{\boldsymbol{\Sigma}} = \begin{pmatrix} \widehat{\boldsymbol{\Sigma}}_{11} & \widehat{\boldsymbol{\Sigma}}_{12} & \cdots & \widehat{\boldsymbol{\Sigma}}_{1d} \\ \widehat{\boldsymbol{\Sigma}}_{12}^T & \widehat{\boldsymbol{\Sigma}}_{22} & \cdots & \widehat{\boldsymbol{\Sigma}}_{2d} \\ \cdots & \cdots & \ddots & \cdots \\ \widehat{\boldsymbol{\Sigma}}_{1d}^T & \widehat{\boldsymbol{\Sigma}}_{2d}^T & \cdots & \widehat{\boldsymbol{\Sigma}}_{dd} \end{pmatrix}$$

where $d' = \sum_{i=1}^{d} m_i$, and $\boldsymbol{\Sigma}_{ij}$ (and $\widehat{\boldsymbol{\Sigma}}_{ij}$) is the $m_i \times m_j$ covariance matrix (and its estimate) for attributes $\mathbf{X}_i$ and $\mathbf{X}_j$:

$$\boldsymbol{\Sigma}_{ij} = \mathbf{P}_{ij} - \mathbf{p}_i \mathbf{p}_j^T \qquad \widehat{\boldsymbol{\Sigma}}_{ij} = \widehat{\mathbf{P}}_{ij} - \hat{\mathbf{p}}_i \hat{\mathbf{p}}_j^T \tag{3.18}$$

Here $\mathbf{P}_{ij}$ is the joint PMF and $\widehat{\mathbf{P}}_{ij}$ is the empirical joint PMF for $\mathbf{X}_i$ and $\mathbf{X}_j$, which can be computed using Eq. (3.13).

Example 3.11 (Multivariate Analysis). Let us consider the 3-dimensional subset of the Iris dataset, with the discretized attributes `sepal length` (X_1) and `sepal width` (X_2), and the categorical attribute `class` (X_3). The domains for X_1 and X_2 are given in Table 3.1 and Table 3.3, respectively, and $dom(X_3) = \{\texttt{iris-versicolor}, \texttt{iris-setosa}, \texttt{iris-virginica}\}$. Each value of X_3 occurs 50 times.

The categorical point $\mathbf{x} = (\texttt{Short}, \texttt{Medium}, \texttt{iris-versicolor})$ is modeled as the vector

$$\mathbf{X}(\mathbf{x}) = \begin{pmatrix} \mathbf{e}_{12} \\ \mathbf{e}_{22} \\ \mathbf{e}_{31} \end{pmatrix} = (0, 1, 0, 0 \mid 0, 1, 0 \mid 1, 0, 0)^T \in \mathbb{R}^{10}$$

From Example 3.8 and the fact that each value in $dom(X_3)$ occurs 50 times in a sample of $n = 150$, the sample mean is given as

$$\hat{\boldsymbol{\mu}} = \begin{pmatrix} \hat{\boldsymbol{\mu}}_1 \\ \hat{\boldsymbol{\mu}}_2 \\ \hat{\boldsymbol{\mu}}_3 \end{pmatrix} = \begin{pmatrix} \hat{\mathbf{p}}_1 \\ \hat{\mathbf{p}}_2 \\ \hat{\mathbf{p}}_3 \end{pmatrix} = (0.3, 0.333, 0.287, 0.08 \mid 0.313, 0.587, 0.1 \mid 0.33, 0.33, 0.33)^T$$

Using $\hat{\mathbf{p}}_3 = (0.33, 0.33, 0.33)^T$ we can compute the sample covariance matrix for X_3 using Eq. (3.9):

$$\widehat{\boldsymbol{\Sigma}}_{33} = \begin{pmatrix} 0.222 & -0.111 & -0.111 \\ -0.111 & 0.222 & -0.111 \\ -0.111 & -0.111 & 0.222 \end{pmatrix}$$

Using Eq. (3.18) we obtain

$$\widehat{\boldsymbol{\Sigma}}_{13} = \begin{pmatrix} -0.067 & 0.16 & -0.093 \\ 0.082 & -0.038 & -0.044 \\ 0.011 & -0.096 & 0.084 \\ -0.027 & -0.027 & 0.053 \end{pmatrix}$$

$$\widehat{\boldsymbol{\Sigma}}_{23} = \begin{pmatrix} 0.076 & -0.098 & 0.022 \\ -0.042 & 0.044 & -0.002 \\ -0.033 & 0.053 & -0.02 \end{pmatrix}$$

Combined with $\widehat{\boldsymbol{\Sigma}}_{11}$, $\widehat{\boldsymbol{\Sigma}}_{22}$ and $\widehat{\boldsymbol{\Sigma}}_{12}$ from Example 3.8, the final sample covariance matrix is the 10×10 symmetric matrix given as

$$\widehat{\boldsymbol{\Sigma}} = \begin{pmatrix} \widehat{\boldsymbol{\Sigma}}_{11} & \widehat{\boldsymbol{\Sigma}}_{12} & \widehat{\boldsymbol{\Sigma}}_{13} \\ \widehat{\boldsymbol{\Sigma}}_{12}^T & \widehat{\boldsymbol{\Sigma}}_{22} & \widehat{\boldsymbol{\Sigma}}_{23} \\ \widehat{\boldsymbol{\Sigma}}_{13}^T & \widehat{\boldsymbol{\Sigma}}_{23}^T & \widehat{\boldsymbol{\Sigma}}_{33} \end{pmatrix}$$

3.3.1 Multiway Contingency Analysis

For multiway dependence analysis, we have to first determine the empirical joint probability mass function for $\mathbf{X}$:

$$\hat{f}(\mathbf{e}_{1i_1}, \mathbf{e}_{2i_2}, \ldots, \mathbf{e}_{di_d}) = \frac{1}{n} \sum_{k=1}^{n} I_{i_1 i_2 \ldots i_d}(\mathbf{x}_k) = \frac{n_{i_1 i_2 \ldots i_d}}{n} = \hat{p}_{i_1 i_2 \ldots i_d}$$

where $I_{i_1 i_2 \ldots i_d}$ is the indicator variable

$$I_{i_1 i_2 \ldots i_d}(\mathbf{x}_k) = \begin{cases} 1 & \text{if } x_{k1} = \mathbf{e}_{1i_1}, x_{k2} = \mathbf{e}_{2i_2}, \ldots, x_{kd} = \mathbf{e}_{di_d} \\ 0 & \text{otherwise} \end{cases}$$

The sum of $I_{i_1 i_2 \ldots i_d}$ over all the n points in the sample yields the number of occurrences, $n_{i_1 i_2 \ldots i_d}$, of the symbolic vector $(a_{1i_1}, a_{2i_2}, \ldots, a_{di_d})$. Dividing the occurrences by the sample size results in the probability of observing those symbols. Using the notation $\mathbf{i} = (i_1, i_2, \ldots, i_d)$ to denote the index tuple, we can write the joint empirical PMF as the d-dimensional matrix $\widehat{\mathbf{P}}$ of size $m_1 \times m_2 \times \cdots \times m_d = \prod_{i=1}^{d} m_i$, given as

$$\widehat{\mathbf{P}}(\mathbf{i}) = \left\{ \hat{p}_{\mathbf{i}} \right\} \text{ for all index tuples } \mathbf{i}, \text{ with } 1 \le i_1 \le m_1, \ldots, 1 \le i_d \le m_d$$

where $\hat{p}_{\mathbf{i}} = \hat{p}_{i_1 i_2 \ldots i_d}$. The d-dimensional contingency table is then given as

$$\mathbf{N} = n \times \widehat{\mathbf{P}} = \left\{ n_{\mathbf{i}} \right\} \text{ for all index tuples } \mathbf{i}, \text{ with } 1 \le i_1 \le m_1, \ldots, 1 \le i_d \le m_d$$

where $n_{\mathbf{i}} = n_{i_1 i_2 \ldots i_d}$. The contingency table is augmented with the marginal count vectors $\mathbf{N}_i$ for all d attributes $\mathbf{X}_i$:

$$\mathbf{N}_i = n\hat{\mathbf{p}}_i = \begin{pmatrix} n_1^i \\ \vdots \\ n_{m_i}^i \end{pmatrix}$$

where $\hat{\mathbf{p}}_i$ is the empirical PMF for $\mathbf{X}_i$.

χ^2-Test

We can test for a d-way dependence between the d categorical attributes using the null hypothesis H_0 that they are d-way independent. The alternative hypothesis H_1 is that they are not d-way independent, that is, they are dependent in some way. Note that d-dimensional contingency analysis indicates whether all d attributes taken together are independent or not. In general we may have to conduct k-way contingency analysis to test if any subset of $k \leq d$ attributes are independent or not.

Under the null hypothesis, the expected number of occurrences of the symbol tuple $(a_{1i_1}, a_{2i_2}, \ldots, a_{di_d})$ is given as

$$e_{\mathbf{i}} = n \cdot \hat{p}_{\mathbf{i}} = n \cdot \prod_{j=1}^{d} \hat{p}_{i_j}^j = \frac{n_{i_1}^1 n_{i_2}^2 \ldots n_{i_d}^d}{n^{d-1}} \tag{3.19}$$

The chi-squared statistic measures the difference between the observed counts $n_{\mathbf{i}}$ and the expected counts $e_{\mathbf{i}}$:

$$\chi^2 = \sum_{\mathbf{i}} \frac{(n_{\mathbf{i}} - e_{\mathbf{i}})^2}{e_{\mathbf{i}}} = \sum_{i_1=1}^{m_1} \sum_{i_2=1}^{m_2} \cdots \sum_{i_d=1}^{m_d} \frac{(n_{i_1,i_2,\ldots,i_d} - e_{i_1,i_2,\ldots,i_d})^2}{e_{i_1,i_2,\ldots,i_d}} \tag{3.20}$$

The χ^2 statistic follows a chi-squared density function with q degrees of freedom. For the d-way contingency table we can compute q by noting that there are ostensibly $\prod_{i=1}^{d} |dom(X_i)|$ independent parameters (the counts). However, we have to remove $\sum_{i=1}^{d} |dom(X_i)|$ degrees of freedom because the marginal count vector along each dimension $\mathbf{X}_i$ must equal $\mathbf{N}_i$. However, doing so removes one of the parameters d times, so we need to add back $d-1$ to the free parameters count. The total number of degrees of freedom is given as

$$\begin{aligned} q &= \prod_{i=1}^{d} |dom(X_i)| - \sum_{i=1}^{d} |dom(X_i)| + (d-1) \\ &= \left(\prod_{i=1}^{d} m_i\right) - \left(\sum_{i=1}^{d} m_i\right) + d - 1 \end{aligned} \tag{3.21}$$

To reject the null hypothesis, we have to check whether the p-value of the observed χ^2 value is smaller than the desired significance level α (say $\alpha = 0.01$) using the chi-squared density with q degrees of freedom [Eq. (3.16)].

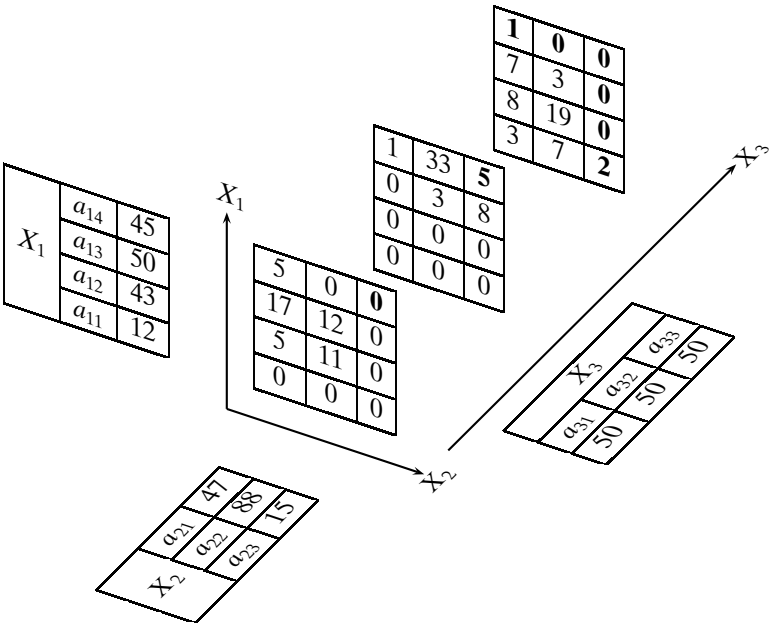


Figure 3.4. 3-Way contingency table (with marginal counts along each dimension).

Table 3.7. 3-Way expected counts

		$X_3(a_{31}/a_{32}/a_{33})$		
		X_2		
		a_{21}	a_{22}	a_{23}
X_1	a_{11}	1.25	2.35	0.40
	a_{12}	4.49	8.41	1.43
	a_{13}	5.22	9.78	1.67
	a_{14}	4.70	8.80	1.50

Example 3.12. Consider the 3-way contingency table in Figure 3.4. It shows the observed counts for each tuple of symbols (a_{1i}, a_{2j}, a_{3k}) for the three attributes `sepal length` (X_1), `sepal width` (X_2), and `class` (X_3). From the marginal counts for X_1 and X_2 in Table 3.5, and the fact that all three values of X_3 occur 50 times, we can compute the expected counts [Eq. (3.19)] for each cell. For instance,

$$e_{(4,1,1)} = \frac{n_4^1 \cdot n_1^2 \cdot n_1^3}{150^2} = \frac{45 \cdot 47 \cdot 50}{150 \cdot 150} = 4.7$$

The expected counts are the same for all three values of X_3 and are given in Table 3.7.

The value of the χ^2 statistic [Eq. (3.20)] is given as

$$\chi^2 = 231.06$$

Using Eq. (3.21), the number of degrees of freedom is given as

$$q = 4 \cdot 3 \cdot 3 - (4 + 3 + 3) + 2 = 36 - 10 + 2 = 28$$

In Figure 3.4 the counts in bold are the dependent parameters. All other counts are independent. In fact, any eight distinct cells could have been chosen as the dependent parameters.

For a significance level of $\alpha = 0.01$, the critical value of the chi-square distribution is $z = 48.28$. The observed value of $\chi^2 = 231.06$ is much greater than z, and it is thus extremely unlikely to happen under the null hypothesis. We conclude that the three attributes are not 3-way independent, but rather there is some dependence between them. However, this example also highlights one of the pitfalls of multiway contingency analysis. We can observe in Figure 3.4 that many of the observed counts are zero. This is due to the fact that the sample size is small, and we cannot reliably estimate all the multiway counts. Consequently, the dependence test may not be reliable as well.

3.4 DISTANCE AND ANGLE

With the modeling of categorical attributes as multivariate Bernoulli variables, it is possible to compute the distance or the angle between any two points $\mathbf{x}_i$ and $\mathbf{x}_j$:

$$\mathbf{x}_i = \begin{pmatrix} \mathbf{e}_{1i_1} \\ \vdots \\ \mathbf{e}_{d\,i_d} \end{pmatrix} \qquad \mathbf{x}_j = \begin{pmatrix} \mathbf{e}_{1j_1} \\ \vdots \\ \mathbf{e}_{d\,j_d} \end{pmatrix}$$

The different measures of distance and similarity rely on the number of matching and mismatching values (or symbols) across the d attributes $\mathbf{X}_k$. For instance, we can compute the number of matching values s via the dot product:

$$s = \mathbf{x}_i^T \mathbf{x}_j = \sum_{k=1}^{d} (\mathbf{e}_{ki_k})^T \mathbf{e}_{kj_k}$$

On the other hand, the number of mismatches is simply $d - s$. Also useful is the norm of each point:

$$\|\mathbf{x}_i\|^2 = \mathbf{x}_i^T \mathbf{x}_i = d$$

Euclidean Distance

The Euclidean distance between $\mathbf{x}_i$ and $\mathbf{x}_j$ is given as

$$\delta(\mathbf{x}_i, \mathbf{x}_j) = \|\mathbf{x}_i - \mathbf{x}_j\| = \sqrt{\mathbf{x}_i^T \mathbf{x}_i - 2\mathbf{x}_i \mathbf{x}_j + \mathbf{x}_j^T \mathbf{x}_j} = \sqrt{2(d - s)}$$

Thus, the maximum Euclidean distance between any two points is $\sqrt{2d}$, which happens when there are no common symbols between them, that is, when $s = 0$.

Hamming Distance

The *Hamming distance* between $\mathbf{x}_i$ and $\mathbf{x}_j$ is defined as the number of mismatched values:

$$\delta_H(\mathbf{x}_i, \mathbf{x}_j) = d - s = \frac{1}{2}\delta(\mathbf{x}_i, \mathbf{x}_j)^2$$

Hamming distance is thus equivalent to half the squared Euclidean distance.

Cosine Similarity

The cosine of the angle between $\mathbf{x}_i$ and $\mathbf{x}_j$ is given as

$$\cos\theta = \frac{\mathbf{x}_i^T \mathbf{x}_j}{\|\mathbf{x}_i\| \cdot \|\mathbf{x}_j\|} = \frac{s}{d}$$

Jaccard Coefficient

The *Jaccard Coefficient* is a commonly used similarity measure between two categorical points. It is defined as the ratio of the number of matching values to the number of distinct values that appear in both $\mathbf{x}_i$ and $\mathbf{x}_j$, across the d attributes:

$$J(\mathbf{x}_i, \mathbf{x}_j) = \frac{s}{2(d-s)+s} = \frac{s}{2d-s}$$

where we utilize the observation that when the two points do not match for dimension k, they contribute 2 to the distinct symbol count; otherwise, if they match, the number of distinct symbols increases by 1. Over the $d - s$ mismatches and s matches, the number of distinct symbols is $2(d-s)+s$.

Example 3.13. Consider the 3-dimensional categorical data from Example 3.11. The symbolic point (`Short`, `Medium`, `iris-versicolor`) is modeled as the vector

$$\mathbf{x}_1 = \begin{pmatrix} \mathbf{e}_{12} \\ \mathbf{e}_{22} \\ \mathbf{e}_{31} \end{pmatrix} = (0,1,0,0 \mid 0,1,0 \mid 1,0,0)^T \in \mathbb{R}^{10}$$

and the symbolic point (`VeryShort`, `Medium`, `iris-setosa`) is modeled as

$$\mathbf{x}_2 = \begin{pmatrix} \mathbf{e}_{11} \\ \mathbf{e}_{22} \\ \mathbf{e}_{32} \end{pmatrix} = (1,0,0,0 \mid 0,1,0 \mid 0,1,0)^T \in \mathbb{R}^{10}$$

The number of matching symbols is given as

$$\begin{aligned} s = \mathbf{x}_1^T \mathbf{x}_2 &= (\mathbf{e}_{12})^T \mathbf{e}_{11} + (\mathbf{e}_{22})^T \mathbf{e}_{22} + (\mathbf{e}_{31})^T \mathbf{e}_{32} \\ &= \begin{pmatrix} 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \\ &= 0 + 1 + 0 = 1 \end{aligned}$$

The Euclidean and Hamming distances are given as

$$\delta(\mathbf{x}_1, \mathbf{x}_2) = \sqrt{2(d-s)} = \sqrt{2 \cdot 2} = \sqrt{4} = 2$$
$$\delta_H(\mathbf{x}_1, \mathbf{x}_2) = d - s = 3 - 1 = 2$$

The cosine and Jaccard similarity are given as

$$\cos\theta = \frac{s}{d} = \frac{1}{3} = 0.333$$
$$J(\mathbf{x}_1, \mathbf{x}_2) = \frac{s}{2d-s} = \frac{1}{5} = 0.2$$

3.5 DISCRETIZATION

Discretization, also called *binning*, converts numeric attributes into categorical ones. It is usually applied for data mining methods that cannot handle numeric attributes. It can also help in reducing the number of values for an attribute, especially if there is noise in the numeric measurements; discretization allows one to ignore small and irrelevant differences in the values.

Formally, given a numeric attribute X, and a random sample $\{x_i\}_{i=1}^n$ of size n drawn from X, the discretization task is to divide the value range of X into k consecutive intervals, also called *bins*, by finding $k-1$ boundary values $v_1, v_2, \ldots, v_{k-1}$ that yield the k intervals:

$$[x_{\min}, v_1],\ (v_1, v_2],\ \ldots,\ (v_{k-1}, x_{\max}]$$

where the extremes of the range of X are given as

$$x_{\min} = \min_i\{x_i\} \qquad x_{\max} = \max_i\{x_i\}$$

The resulting k intervals or bins, which span the entire range of X, are usually mapped to symbolic values that comprise the domain for the new categorical attribute X.

Equal-Width Intervals

The simplest binning approach is to partition the range of X into k *equal-width* intervals. The interval width is simply the range of X divided by k:

$$w = \frac{x_{\max} - x_{\min}}{k}$$

Thus, the ith interval boundary is given as

$$v_i = x_{\min} + iw,\ \text{for } i = 1, \ldots, k-1$$

Equal-Frequency Intervals

In *equal-frequency* binning we divide the range of X into intervals that contain (approximately) equal number of points; equal frequency may not be possible due to repeated values. The intervals can be computed from the empirical quantile or

inverse cumulative distribution function $\hat{F}^{-1}(q)$ for X [Eq. (2.2)]. Recall that $\hat{F}^{-1}(q) = \min\{x \mid P(X \leq x) \geq q\}$, for $q \in [0, 1]$. In particular, we require that each interval contain $1/k$ of the probability mass; therefore, the interval boundaries are given as follows:

$$v_i = \hat{F}^{-1}(i/k) \text{ for } i = 1, \ldots, k-1$$

Example 3.14. Consider the `sepal length` attribute in the Iris dataset. Its minimum and maximum values are

$$x_{\min} = 4.3 \qquad x_{\max} = 7.9$$

We discretize it into $k = 4$ bins using equal-width binning. The width of an interval is given as

$$w = \frac{7.9 - 4.3}{4} = \frac{3.6}{4} = 0.9$$

and therefore the interval boundaries are

$$v_1 = 4.3 + 0.9 = 5.2 \qquad v_2 = 4.3 + 2 \cdot 0.9 = 6.1 \qquad v_3 = 4.3 + 3 \cdot 0.9 = 7.0$$

The four resulting bins for `sepal length` are shown in Table 3.1, which also shows the number of points n_i in each bin, which are not balanced among the bins.

For equal-frequency discretization, consider the empirical inverse cumulative distribution function (CDF) for `sepal length` shown in Figure 3.5. With $k = 4$ bins, the bin boundaries are the quartile values (which are shown as dashed lines):

$$v_1 = \hat{F}^{-1}(0.25) = 5.1 \qquad v_2 = \hat{F}^{-1}(0.50) = 5.8 \qquad v_3 = \hat{F}^{-1}(0.75) = 6.4$$

The resulting intervals are shown in Table 3.8. We can see that although the interval widths vary, they contain a more balanced number of points. We do not get identical

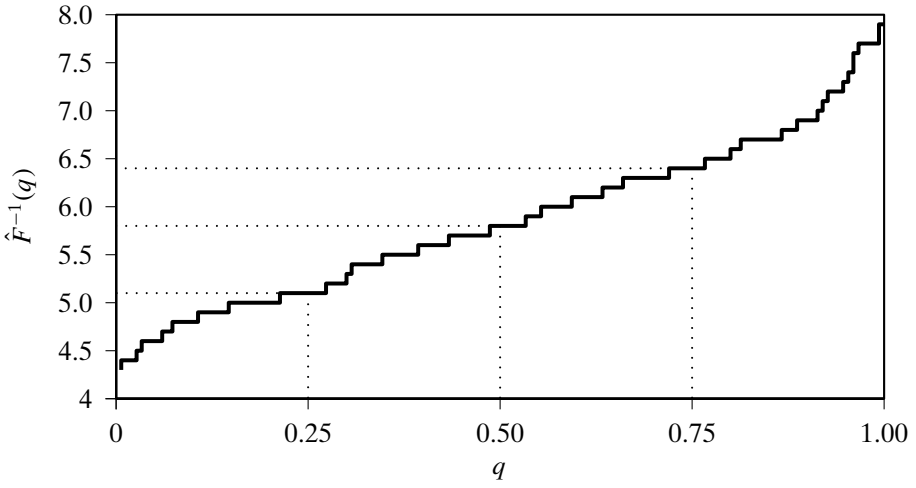


Figure 3.5. Empirical inverse CDF: `sepal length`.

Table 3.8. Equal-frequency discretization: `sepal length`

Bin	Width	Count
[4.3, 5.1]	0.8	$n_1 = 41$
(5.1, 5.8]	0.7	$n_2 = 39$
(5.8, 6.4]	0.6	$n_3 = 35$
(6.4, 7.9]	1.5	$n_4 = 35$

counts for all the bins because many values are repeated; for instance, there are nine points with value 5.1 and there are seven points with value 5.8.

3.6 FURTHER READING

For a comprehensive introduction to categorical data analysis see Agresti (2012). Some aspects also appear in Wasserman (2004). For an entropy-based supervised discretization method that takes the class attribute into account see Fayyad and Irani (1993).

Agresti, A. (2012). *Categorical Data Analysis*, 3rd ed. Hoboken, NJ: John Wiley & Sons.

Fayyad, U. M. and Irani, K. B. (1993). Multi-interval Discretization of Continuous-valued Attributes for Classification Learning. *In Proceedings of the 13th International Joint Conference on Artificial Intelligence*. Morgan-Kaufmann, pp. 1022–1027.

Wasserman, L. (2004). *All of Statistics: A Concise Course in Statistical Inference*. NewYork: Springer Science + Business Media.

3.7 EXERCISES

Q1. Show that for categorical points, the cosine similarity between any two vectors in lies in the range $\cos\theta \in [0, 1]$, and consequently $\theta \in [0^\circ, 90^\circ]$.

Q2. Prove that $E[(\mathbf{X}_1 - \boldsymbol{\mu}_1)(\mathbf{X}_2 - \boldsymbol{\mu}_2)^T] = E[\mathbf{X}_1\mathbf{X}_2^T] - E[\mathbf{X}_1]E[\mathbf{X}_2]^T$.

Table 3.9. Contingency table for Q3

	$Z = f$		$Z = g$	
	$Y = d$	$Y = e$	$Y = d$	$Y = e$
$X = a$	5	10	10	5
$X = b$	15	5	5	20
$X = c$	20	10	25	10

Table 3.10. χ^2 Critical values for different *p-values* for different degrees of freedom (q): For example, for $q = 5$ degrees of freedom, the critical value of $\chi^2 = 11.070$ has *p-value* $= 0.05$.

q	0.995	0.99	0.975	0.95	0.90	0.10	0.05	0.025	0.01	0.005
1	—	—	0.001	0.004	0.016	2.706	3.841	5.024	6.635	7.879
2	0.010	0.020	0.051	0.103	0.211	4.605	5.991	7.378	9.210	10.597
3	0.072	0.115	0.216	0.352	0.584	6.251	7.815	9.348	11.345	12.838
4	0.207	0.297	0.484	0.711	1.064	7.779	9.488	11.143	13.277	14.860
5	0.412	0.554	0.831	1.145	1.610	9.236	11.070	12.833	15.086	16.750
6	0.676	0.872	1.237	1.635	2.204	10.645	12.592	14.449	16.812	18.548

Q3. Consider the 3-way contingency table for attributes X, Y, Z shown in Table 3.9. Compute the χ^2 metric for the correlation between Y and Z. Are they dependent or independent at the 95% confidence level? See Table 3.10 for χ^2 values.

Q4. Consider the "mixed" data given in Table 3.11. Here X_1 is a numeric attribute and X_2 is a categorical one. Assume that the domain of X_2 is given as $dom(X_2) = \{a, b\}$. Answer the following questions.

(a) What is the mean vector for this dataset?

(b) What is the covariance matrix?

Q5. In Table 3.11, assuming that X_1 is discretized into three bins, as follows:

$$c_1 = (-2, -0.5]$$
$$c_2 = (-0.5, 0.5]$$
$$c_3 = (0.5, 2]$$

Answer the following questions:

(a) Construct the contingency table between the discretized X_1 and X_2 attributes. Include the marginal counts.

(b) Compute the χ^2 statistic between them.

(c) Determine whether they are dependent or not at the 5% significance level. Use the χ^2 critical values from Table 3.10.

Table 3.11. Dataset for Q4 and Q5

X_1	X_2
0.3	a
−0.3	b
0.44	a
−0.60	a
0.40	a
1.20	b
−0.12	a
−1.60	b
1.60	b
−1.32	a

CHAPTER 4 Graph Data

The traditional paradigm in data analysis typically assumes that each data instance is independent of another. However, often data instances may be connected or linked to other instances via various types of relationships. The instances themselves may be described by various attributes. What emerges is a network or graph of instances (or nodes), connected by links (or edges). Both the nodes and edges in the graph may have several attributes that may be numerical or categorical, or even more complex (e.g., time series data). Increasingly, today's massive data is in the form of such graphs or networks. Examples include the World Wide Web (with its Web pages and hyperlinks), social networks (wikis, blogs, tweets, and other social media data), semantic networks (ontologies), biological networks (protein interactions, gene regulation networks, metabolic pathways), citation networks for scientific literature, and so on. In this chapter we look at the analysis of the link structure in graphs that arise from these kinds of networks. We will study basic topological properties as well as models that give rise to such graphs.

4.1 GRAPH CONCEPTS

Graphs

Formally, a *graph* $G = (V, E)$ is a mathematical structure consisting of a finite nonempty set V of *vertices* or *nodes*, and a set $E \subseteq V \times V$ of *edges* consisting of *unordered* pairs of vertices. An edge from a node to itself, (v_i, v_i), is called a *loop*. An undirected graph without loops is called a *simple graph*. Unless mentioned explicitly, we will consider a graph to be simple. An edge $e = (v_i, v_j)$ between v_i and v_j is said to be *incident with* nodes v_i and v_j; in this case we also say that v_i and v_j are *adjacent* to one another, and that they are *neighbors*. The number of nodes in the graph G, given as $|V| = n$, is called the *order* of the graph, and the number of edges in the graph, given as $|E| = m$, is called the *size* of G.

A *directed graph* or *digraph* has an edge set E consisting of *ordered* pairs of vertices. A directed edge (v_i, v_j) is also called an *arc*, and is said to be *from* v_i *to* v_j. We also say that v_i is the *tail* and v_j the *head* of the arc.

A *weighted graph* consists of a graph together with a weight w_{ij} for each edge $(v_i, v_j) \in E$. Every graph can be considered to be a weighted graph in which the edges have weight one.

Subgraphs

A graph $H = (V_H, E_H)$ is called a *subgraph* of $G = (V, E)$ if $V_H \subseteq V$ and $E_H \subseteq E$. We also say that G is a *supergraph* of H. Given a subset of the vertices $V' \subseteq V$, the *induced subgraph* $G' = (V', E')$ consists exactly of all the edges present in G between vertices in V'. More formally, for all $v_i, v_j \in V'$, $(v_i, v_j) \in E' \iff (v_i, v_j) \in E$. In other words, two nodes are adjacent in G' if and only if they are adjacent in G. A (sub)graph is called *complete* (or a *clique*) if there exists an edge between all pairs of nodes.

Degree

The *degree* of a node $v_i \in V$ is the number of edges incident with it, and is denoted as $d(v_i)$ or just d_i. The *degree sequence* of a graph is the list of the degrees of the nodes sorted in non-increasing order.

Let N_k denote the number of vertices with degree k. The *degree frequency distribution* of a graph is given as

$$(N_0, N_1, \ldots, N_t)$$

where t is the maximum degree for a node in G. Let X be a random variable denoting the degree of a node. The *degree distribution* of a graph gives the probability mass function f for X, given as

$$\big(f(0), f(1), \ldots, f(t)\big)$$

where $f(k) = P(X = k) = \frac{N_k}{n}$ is the probability of a node with degree k, given as the number of nodes N_k with degree k, divided by the total number of nodes n. In graph analysis, we typically make the assumption that the input graph represents a population, and therefore we write f instead of $\hat{f}$ for the probability distributions.

For directed graphs, the *indegree* of node v_i, denoted as $id(v_i)$, is the number of edges with v_i as head, that is, the number of incoming edges at v_i. The *outdegree* of v_i, denoted $od(v_i)$, is the number of edges with v_i as the tail, that is, the number of outgoing edges from v_i.

Path and Distance

A *walk* in a graph G between nodes x and y is an ordered sequence of vertices, starting at x and ending at y,

$$x = v_0,\ v_1,\ \ldots,\ v_{t-1},\ v_t = y$$

such that there is an edge between every pair of consecutive vertices, that is, $(v_{i-1}, v_i) \in E$ for all $i = 1, 2, \ldots, t$. The length of the walk, t, is measured in terms of *hops* – the number of edges along the walk. In a walk, there is no restriction on the number of times a given vertex may appear in the sequence; thus both the vertices and edges may be repeated. A walk starting and ending at the same vertex (i.e., with $y = x$) is called *closed*. A *trail* is a walk with distinct edges, and a *path* is a walk with *distinct* vertices (with the exception of the start and end vertices). A closed path with length

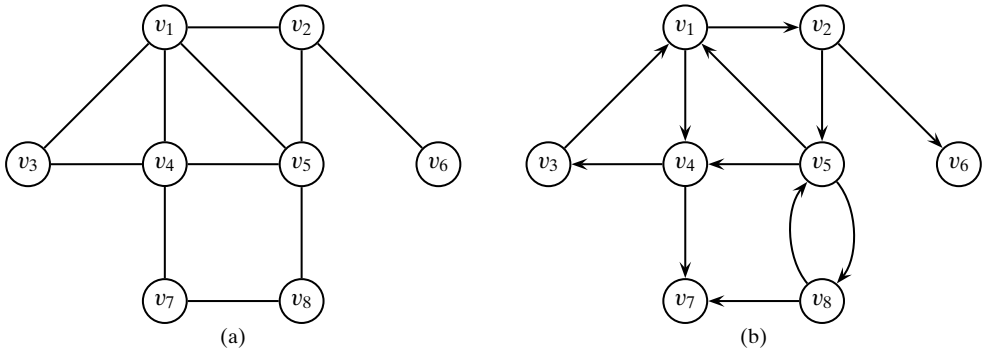


Figure 4.1. (a) A graph (undirected). (b) A directed graph.

$t \geq 3$ is called a *cycle*, that is, a cycle begins and ends at the same vertex and has distinct nodes.

A path of minimum length between nodes x and y is called a *shortest path*, and the length of the shortest path is called the *distance* between x and y, denoted as $d(x, y)$. If no path exists between the two nodes, the distance is assumed to be $d(x, y) = \infty$.

Connectedness

Two nodes v_i and v_j are said to be *connected* if there exists a path between them. A graph is *connected* if there is a path between all pairs of vertices. A *connected component*, or just *component*, of a graph is a maximal connected subgraph. If a graph has only one component it is connected; otherwise it is *disconnected*, as by definition there cannot be a path between two different components.

For a directed graph, we say that it is *strongly connected* if there is a (directed) path between all ordered pairs of vertices. We say that it is *weakly connected* if there exists a path between node pairs only by considering edges as undirected.

Example 4.1. Figure 4.1a shows a graph with $|V| = 8$ vertices and $|E| = 11$ edges. Because $(v_1, v_5) \in E$, we say that v_1 and v_5 are adjacent. The degree of v_1 is $d(v_1) = d_1 = 4$. The degree sequence of the graph is

$$(4, 4, 4, 3, 2, 2, 2, 1)$$

and therefore its degree frequency distribution is given as

$$(N_0, N_1, N_2, N_3, N_4) = (0, 1, 3, 1, 3)$$

We have $N_0 = 0$ because there are no isolated vertices, and $N_4 = 3$ because there are three nodes, v_1, v_4 and v_5, that have degree $k = 4$; the other numbers are obtained in a similar fashion. The degree distribution is given as

$$\big(f(0), f(1), f(2), f(3), f(4)\big) = (0, 0.125, 0.375, 0.125, 0.375)$$

The vertex sequence $(v_3, v_1, v_2, v_5, v_1, v_2, v_6)$ is a walk of length 6 between v_3 and v_6. We can see that vertices v_1 and v_2 have been visited more than once. In

contrast, the vertex sequence $(v_3, v_4, v_7, v_8, v_5, v_2, v_6)$ is a path of length 6 between v_3 and v_6. However, this is not the shortest path between them, which happens to be (v_3, v_1, v_2, v_6) with length 3. Thus, the distance between them is given as $d(v_3, v_6) = 3$.

Figure 4.1b shows a directed graph with 8 vertices and 12 edges. We can see that edge (v_5, v_8) is distinct from edge (v_8, v_5). The indegree of v_7 is $id(v_7) = 2$, whereas its outdegree is $od(v_7) = 0$. Thus, there is no (directed) path from v_7 to any other vertex.

Adjacency Matrix

A graph $G = (V, E)$, with $|V| = n$ vertices, can be conveniently represented in the form of an $n \times n$, symmetric binary *adjacency matrix*, $\mathbf{A}$, defined as

$$\mathbf{A}(i, j) = \begin{cases} 1 & \text{if } v_i \text{ is adjacent to } v_j \\ 0 & \text{otherwise} \end{cases}$$

If the graph is directed, then the adjacency matrix $\mathbf{A}$ is not symmetric, as $(v_i, v_j) \in E$ obviously does not imply that $(v_j, v_i) \in E$.

If the graph is weighted, then we obtain an $n \times n$ *weighted adjacency matrix*, $\mathbf{A}$, defined as

$$\mathbf{A}(i, j) = \begin{cases} w_{ij} & \text{if } v_i \text{ is adjacent to } v_j \\ 0 & \text{otherwise} \end{cases}$$

where w_{ij} is the weight on edge $(v_i, v_j) \in E$. A weighted adjacency matrix can always be converted into a binary one, if desired, by using some threshold τ on the edge weights

$$\mathbf{A}(i, j) = \begin{cases} 1 & \text{if } w_{ij} \geq \tau \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

Graphs from Data Matrix

Many datasets that are not in the form of a graph can nevertheless be converted into one. Let $\mathbf{D} = \{\mathbf{x}_i\}_{i=1}^n$ (with $\mathbf{x}_i \in \mathbb{R}^d$), be a dataset consisting of n points in a d-dimensional space. We can define a weighted graph $G = (V, E)$, where there exists a node for each point in $\mathbf{D}$, and there exists an edge between each pair of points, with weight

$$w_{ij} = sim(\mathbf{x}_i, \mathbf{x}_j)$$

where $sim(\mathbf{x}_i, \mathbf{x}_j)$ denotes the similarity between points $\mathbf{x}_i$ and $\mathbf{x}_j$. For instance, similarity can be defined as being inversely related to the Euclidean distance between the points via the transformation

$$w_{ij} = sim(\mathbf{x}_i, \mathbf{x}_j) = \exp\left\{-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right\} \tag{4.2}$$

where σ is the spread parameter (equivalent to the standard deviation in the normal density function). This transformation restricts the similarity function $sim()$ to lie in the range $[0, 1]$. One can then choose an appropriate threshold τ and convert the weighted adjacency matrix into a binary one via Eq. (4.1).

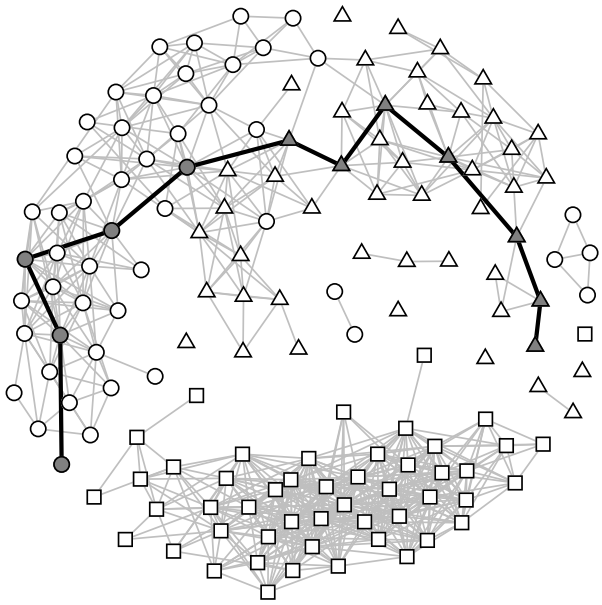


Figure 4.2. Iris similarity graph.

Example 4.2. Figure 4.2 shows the similarity graph for the Iris dataset (see Table 1.1). The pairwise similarity between distinct pairs of points was computed using Eq. (4.2), with $\sigma = 1/\sqrt{2}$ (we do not allow loops, to keep the graph simple). The mean similarity between points was 0.197, with a standard deviation of 0.290.

A binary adjacency matrix was obtained via Eq. (4.1) using a threshold of $\tau = 0.777$, which results in an edge between points having similarity higher than two standard deviations from the mean. The resulting Iris graph has 150 nodes and 753 edges.

The nodes in the Iris graph in Figure 4.2 have also been categorized according to their class. The circles correspond to class `iris-versicolor`, the triangles to `iris-virginica`, and the squares to `iris-setosa`. The graph has two big components, one of which is exclusively composed of nodes labeled as `iris-setosa`.

4.2 TOPOLOGICAL ATTRIBUTES

In this section we study some of the purely topological, that is, edge-based or structural, attributes of graphs. These attributes are *local* if they apply to only a single node (or an edge), and *global* if they refer to the entire graph.

Degree

We have already defined the degree of a node v_i as the number of its neighbors. A more general definition that holds even when the graph is weighted is as follows:

$$d_i = \sum_j \mathbf{A}(i, j)$$

The degree is clearly a local attribute of each node. One of the simplest global attribute is the *average degree*:

$$\mu_d = \frac{\sum_i d_i}{n}$$

The preceding definitions can easily be generalized for (weighted) directed graphs. For example, we can obtain the indegree and outdegree by taking the summation over the incoming and outgoing edges, as follows:

$$id(v_i) = \sum_j \mathbf{A}(j,i)$$

$$od(v_i) = \sum_j \mathbf{A}(i,j)$$

The average indegree and average outdegree can be obtained likewise.

Average Path Length

The *average path length*, also called the *characteristic path length*, of a connected graph is given as

$$\mu_L = \frac{\sum_i \sum_{j>i} d(v_i, v_j)}{\binom{n}{2}} = \frac{2}{n(n-1)} \sum_i \sum_{j>i} d(v_i, v_j)$$

where n is the number of nodes in the graph, and $d(v_i, v_j)$ is the distance between v_i and v_j. For a directed graph, the average is over all ordered pairs of vertices:

$$\mu_L = \frac{1}{n(n-1)} \sum_i \sum_j d(v_i, v_j)$$

For a disconnected graph the average is taken over only the connected pairs of vertices.

Eccentricity

The *eccentricity* of a node v_i is the maximum distance from v_i to any other node in the graph:

$$e(v_i) = \max_j \{d(v_i, v_j)\}$$

If the graph is disconnected the eccentricity is computed only over pairs of vertices with finite distance, that is, only for vertices connected by a path.

Radius and Diameter

The *radius* of a connected graph, denoted $r(G)$, is the minimum eccentricity of any node in the graph:

$$r(G) = \min_i \{e(v_i)\} = \min_i \left\{ \max_j \{d(v_i, v_j)\} \right\}$$

The *diameter*, denoted $d(G)$, is the maximum eccentricity of any vertex in the graph:

$$d(G) = \max_i \{e(v_i)\} = \max_{i,j} \{d(v_i, v_j)\}$$

For a disconnected graph, the diameter is the maximum eccentricity over all the connected components of the graph.

The diameter of a graph G is sensitive to outliers. A more robust notion is *effective diameter*, defined as the minimum number of hops for which a large fraction, typically 90%, of all connected pairs of nodes can reach each other. More formally, let $H(k)$ denote the number of pairs of nodes that can reach each other in k hops or less. The effective diameter is defined as the smallest value of k such that $H(k) \geq 0.9 \times H(d(G))$.

Example 4.3. For the graph in Figure 4.1a, the eccentricity of node v_4 is $e(v_4) = 3$ because the node farthest from it is v_6 and $d(v_4, v_6) = 3$. The radius of the graph is $r(G) = 2$; both v_1 and v_5 have the least eccentricity value of 2. The diameter of the graph is $d(G) = 4$, as the largest distance over all the pairs is $d(v_6, v_7) = 4$.

The diameter of the Iris graph is $d(G) = 11$, which corresponds to the bold path connecting the gray nodes in Figure 4.2. The degree distribution for the Iris graph is shown in Figure 4.3. The numbers at the top of each bar indicate the frequency. For example, there are exactly 13 nodes with degree 7, which corresponds to the probability $f(7) = \frac{13}{150} = 0.0867$.

The path length histogram for the Iris graph is shown in Figure 4.4. For instance, 1044 node pairs have a distance of 2 hops between them. With $n = 150$ nodes, there

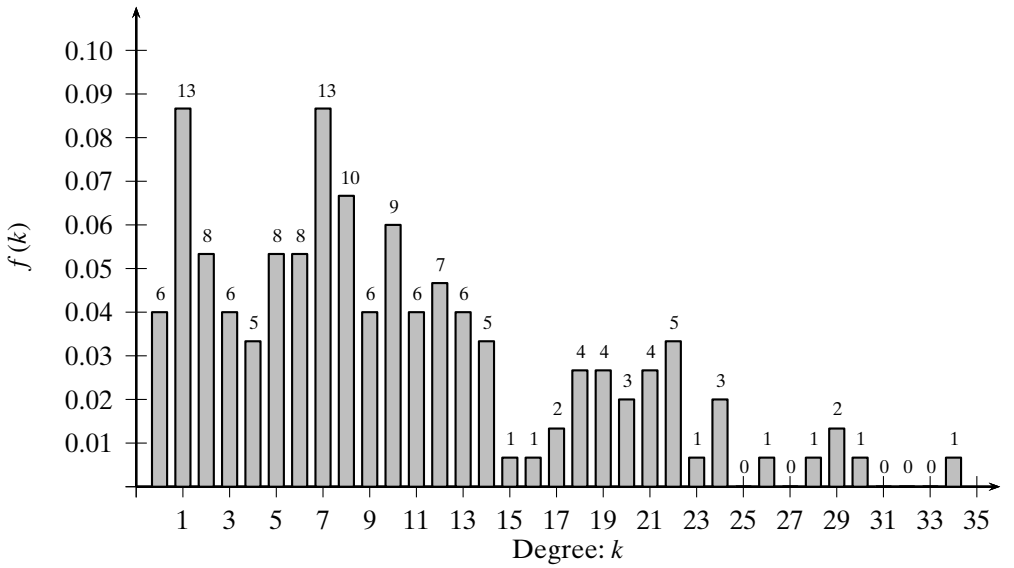


Figure 4.3. Iris graph: degree distribution.

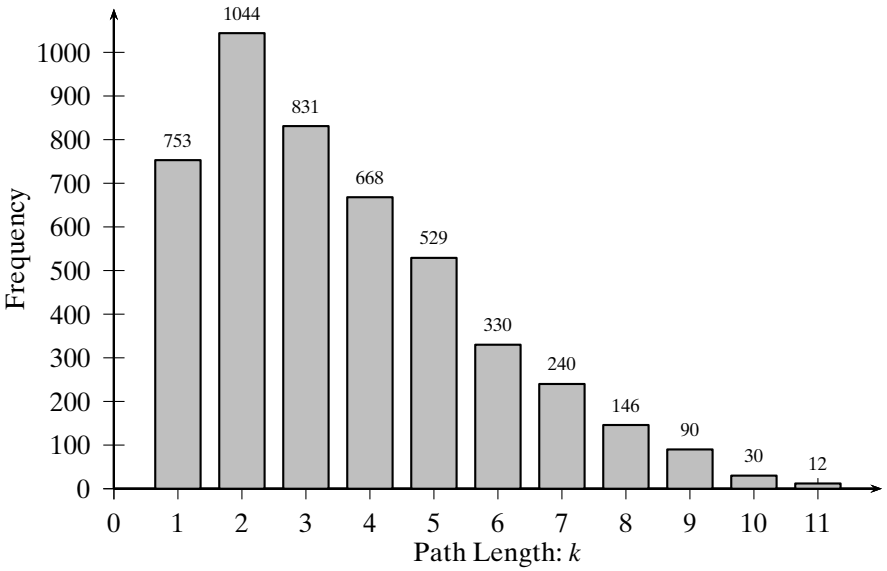


Figure 4.4. Iris graph: path length histogram.

are $\binom{n}{2} = 11,175$ pairs. Out of these 6502 pairs are unconnected, and there are a total of 4673 reachable pairs. Out of these $\frac{4175}{4673} = 0.89$ fraction are reachable in 6 hops, and $\frac{4415}{4673} = 0.94$ fraction are reachable in 7 hops. Thus, we can determine that the effective diameter is 7. The average path length is 3.58.

Clustering Coefficient

The *clustering coefficient* of a node v_i is a measure of the density of edges in the neighborhood of v_i. Let $G_i = (V_i, E_i)$ be the subgraph induced by the neighbors of vertex v_i. Note that $v_i \notin V_i$, as we assume that G is simple. Let $|V_i| = n_i$ be the number of neighbors of v_i, and $|E_i| = m_i$ be the number of edges among the neighbors of v_i. The clustering coefficient of v_i is defined as

$$C(v_i) = \frac{\text{no. of edges in } G_i}{\text{maximum number of edges in } G_i} = \frac{m_i}{\binom{n_i}{2}} = \frac{2 \cdot m_i}{n_i(n_i - 1)}$$

The clustering coefficient gives an indication about the "cliquishness" of a node's neighborhood, because the denominator corresponds to the case when G_i is a complete subgraph.

The *clustering coefficient* of a graph G is simply the average clustering coefficient over all the nodes, given as

$$C(G) = \frac{1}{n} \sum_i C(v_i)$$

Because $C(v_i)$ is well defined only for nodes with degree $d(v_i) \geq 2$, we can define $C(v_i) = 0$ for nodes with degree less than 2. Alternatively, we can take the summation only over nodes with $d(v_i) \geq 2$.

The clustering coefficient $C(v_i)$ of a node is closely related to the notion of transitive relationships in a graph or network. That is, if there exists an edge between v_i and v_j, and another between v_i and v_k, then how likely are v_j and v_k to be linked or connected to each other. Define the subgraph composed of the edges (v_i, v_j) and (v_i, v_k) to be a *connected triple* centered at v_i. A connected triple centered at v_i that includes (v_j, v_k) is called a *triangle* (a complete subgraph of size 3). The clustering coefficient of node v_i can be expressed as

$$C(v_i) = \frac{\text{no. of triangles including } v_i}{\text{no. of connected triples centered at } v_i}$$

Note that the number of connected triples centered at v_i is simply $\binom{d_i}{2} = \frac{n_i(n_i-1)}{2}$, where $d_i = n_i$ is the number of neighbors of v_i.

Generalizing the aforementioned notion to the entire graph yields the *transitivity* of the graph, defined as

$$T(G) = \frac{3 \times \text{no. of triangles in } G}{\text{no. of connected triples in } G}$$

The factor 3 in the numerator is due to the fact that each triangle contributes to three connected triples centered at each of its three vertices. Informally, transitivity measures the degree to which a friend of your friend is also your friend, say, in a social network.

Efficiency

The *efficiency* for a pair of nodes v_i and v_j is defined as $\frac{1}{d(v_i,v_j)}$. If v_i and v_j are not connected, then $d(v_i, v_j) = \infty$ and the efficiency is $1/\infty = 0$. As such, the smaller the distance between the nodes, the more "efficient" the communication between them. The *efficiency* of a graph G is the average efficiency over all pairs of nodes, whether connected or not, given as

$$\frac{2}{n(n-1)} \sum_i \sum_{j>i} \frac{1}{d(v_i, v_j)}$$

The maximum efficiency value is 1, which holds for a complete graph.

The *local efficiency* for a node v_i is defined as the efficiency of the subgraph G_i induced by the neighbors of v_i. Because $v_i \notin G_i$, the local efficiency is an indication of the local fault tolerance, that is, how efficient is the communication between neighbors of v_i when v_i is removed or deleted from the graph.

Example 4.4. For the graph in Figure 4.1a, consider node v_4. Its neighborhood graph is shown in Figure 4.5. The clustering coefficient of node v_4 is given as

$$C(v_4) = \frac{2}{\binom{4}{2}} = \frac{2}{6} = 0.33$$

The clustering coefficient for the entire graph (over all nodes) is given as

$$C(G) = \frac{1}{8}\left(\frac{1}{2} + \frac{1}{3} + 1 + \frac{1}{3} + \frac{1}{3} + 0 + 0 + 0\right) = \frac{2.5}{8} = 0.3125$$

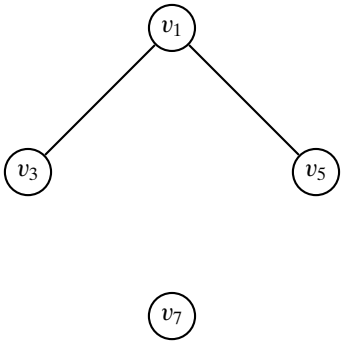


Figure 4.5. Subgraph G_4 induced by node v_4.

The local efficiency of v_4 is given as

$$\frac{2}{4 \cdot 3}\left(\frac{1}{d(v_1, v_3)} + \frac{1}{d(v_1, v_5)} + \frac{1}{d(v_1, v_7)} + \frac{1}{d(v_3, v_5)} + \frac{1}{d(v_3, v_7)} + \frac{1}{d(v_5, v_7)}\right)$$
$$= \frac{1}{6}(1 + 1 + 0 + 0.5 + 0 + 0) = \frac{2.5}{6} = 0.417$$

4.3 CENTRALITY ANALYSIS

The notion of *centrality* is used to rank the vertices of a graph in terms of how "central" or important they are. A centrality can be formally defined as a function $c: V \to \mathbb{R}$, that induces a total order on V. We say that v_i is at least as central as v_j if $c(v_i) \geq c(v_j)$.

4.3.1 Basic Centralities

Degree Centrality

The simplest notion of centrality is the degree d_i of a vertex v_i – the higher the degree, the more important or central the vertex. For directed graphs, one may further consider the indegree centrality and outdegree centrality of a vertex.

Eccentricity Centrality

According to this notion, the less eccentric a node is, the more central it is. Eccentricity centrality is thus defined as follows:

$$c(v_i) = \frac{1}{e(v_i)} = \frac{1}{\max_j \{d(v_i, v_j)\}}$$

A node v_i that has the least eccentricity, that is, for which the eccentricity equals the graph radius, $e(v_i) = r(G)$, is called a *center node*, whereas a node that has the highest eccentricity, that is, for which eccentricity equals the graph diameter, $e(v_i) = d(G)$, is called a *periphery node*.

Eccentricity centrality is related to the problem of *facility location*, that is, choosing the optimum location for a resource or facility. The central node minimizes the maximum distance to any node in the network, and thus the most central node would be an ideal location for, say, a hospital, because it is desirable to minimize the maximum distance someone has to travel to get to the hospital quickly.

Closeness Centrality

Whereas eccentricity centrality uses the maximum of the distances from a given node, closeness centrality uses the sum of all the distances to rank how central a node is

$$c(v_i) = \frac{1}{\sum_j d(v_i, v_j)}$$

A node v_i with the smallest total distance, $\sum_j d(v_i, v_j)$, is called the *median node*.

Closeness centrality optimizes a different objective function for the facility location problem. It tries to minimize the total distance over all the other nodes, and thus a median node, which has the highest closeness centrality, is the optimal one to, say, locate a facility such as a new coffee shop or a mall, as in this case it is not as important to minimize the distance for the farthest node.

Betweenness Centrality

For a given vertex v_i the betweenness centrality measures how many shortest paths between all pairs of vertices include v_i. This gives an indication as to the central "monitoring" role played by v_i for various pairs of nodes. Let η_{jk} denote the number of shortest paths between vertices v_j and v_k, and let $\eta_{jk}(v_i)$ denote the number of such paths that include or contain v_i. Then the fraction of paths through v_i is denoted as

$$\gamma_{jk}(v_i) = \frac{\eta_{jk}(v_i)}{\eta_{jk}}$$

If the two vertices v_j and v_k are not connected, we assume $\gamma_{jk} = 0$.

The betweenness centrality for a node v_i is defined as

$$c(v_i) = \sum_{j \neq i} \sum_{\substack{k \neq i \\ k > j}} \gamma_{jk} = \sum_{j \neq i} \sum_{\substack{k \neq i \\ k > j}} \frac{\eta_{jk}(v_i)}{\eta_{jk}} \tag{4.3}$$

Example 4.5. Consider Figure 4.1a. The values for the different node centrality measures are given in Table 4.1. According to degree centrality, nodes v_1, v_4, and v_5 are the most central. The eccentricity centrality is the highest for the center nodes in the graph, which are v_1 and v_5. It is the least for the periphery nodes, of which there are two, v_6 and, v_7.

Nodes v_1 and v_5 have the highest closeness centrality value. In terms of betweenness, vertex v_5 is the most central, with a value of 6.5. We can compute this value by considering only those pairs of nodes v_j and v_k that have at least one shortest

Table 4.1. Centrality values

Centrality	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Degree	4	3	2	4	4	1	2	2
Eccentricity	0.5	0.33	0.33	0.33	0.5	0.25	0.25	0.33
$e(v_i)$	2	3	3	3	2	4	4	3
Closeness	0.100	0.083	0.071	0.091	0.100	0.056	0.067	0.071
$\sum_j d(v_i, v_j)$	10	12	14	11	10	18	15	14
Betweenness	4.5	6	0	5	6.5	0	0.83	1.17

path passing through v_5, as only these node pairs have $\gamma_{jk} > 0$ in Eq. (4.3). We have

$$c(v_5) = \gamma_{18} + \gamma_{24} + \gamma_{27} + \gamma_{28} + \gamma_{38} + \gamma_{46} + \gamma_{48} + \gamma_{67} + \gamma_{68}$$
$$= 1 + \frac{1}{2} + \frac{2}{3} + 1 + \frac{2}{3} + \frac{1}{2} + \frac{1}{2} + \frac{2}{3} + 1 = 6.5$$

4.3.2 Web Centralities

We now consider directed graphs, especially in the context of the Web. For example, hypertext documents have directed links pointing from one document to another; citation networks of scientific articles have directed edges from a paper to the cited papers, and so on. We consider notions of centrality that are particularly suited to such Web-scale graphs.

Prestige

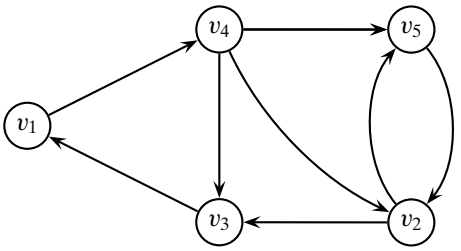
We first look at the notion of *prestige*, or the *eigenvector centrality*, of a node in a directed graph. As a centrality, prestige is supposed to be a measure of the importance or rank of a node. Intuitively the more the links that point to a given node, the higher its prestige. However, prestige does not depend simply on the indegree; it also (recursively) depends on the prestige of the nodes that point to it.

Let $G = (V, E)$ be a directed graph, with $|V| = n$. The adjacency matrix of G is an $n \times n$ asymmetric matrix $\mathbf{A}$ given as

$$\mathbf{A}(u, v) = \begin{cases} 1 & \text{if } (u, v) \in E \\ 0 & \text{if } (u, v) \notin E \end{cases}$$

Let $p(u)$ be a positive real number, called the *prestige* score for node u. Using the intuition that the prestige of a node depends on the prestige of other nodes pointing to it, we can obtain the prestige score of a given node v as follows:

$$p(v) = \sum_u \mathbf{A}(u, v) \cdot p(u)$$
$$= \sum_u \mathbf{A}^T(v, u) \cdot p(u)$$



(a)

$$\mathbf{A} = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix} \qquad \mathbf{A}^T = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

(b) (c)

Figure 4.6. Example graph (a), adjacency matrix (b), and its transpose (c).

For example, in Figure 4.6, the prestige of v_5 depends on the prestige of v_2 and v_4.

Across all the nodes, we can recursively express the prestige scores as

$$\mathbf{p}' = \mathbf{A}^T \mathbf{p} \tag{4.4}$$

where $\mathbf{p}$ is an n-dimensional column vector corresponding to the prestige scores for each vertex.

Starting from an initial prestige vector we can use Eq. (4.4) to obtain an updated prestige vector in an iterative manner. In other words, if $\mathbf{p}_{k-1}$ is the prestige vector across all the nodes at iteration $k-1$, then the updated prestige vector at iteration k is given as

$$\begin{aligned} \mathbf{p}_k &= \mathbf{A}^T \mathbf{p}_{k-1} \\ &= \mathbf{A}^T (\mathbf{A}^T \mathbf{p}_{k-2}) = \left(\mathbf{A}^T\right)^2 \mathbf{p}_{k-2} \\ &= \left(\mathbf{A}^T\right)^2 (\mathbf{A}^T \mathbf{p}_{k-3}) = \left(\mathbf{A}^T\right)^3 \mathbf{p}_{k-3} \\ &= \vdots \\ &= \left(\mathbf{A}^T\right)^k \mathbf{p}_0 \end{aligned}$$

where $\mathbf{p}_0$ is the initial prestige vector. It is well known that the vector $\mathbf{p}_k$ converges to the dominant eigenvector of $\mathbf{A}^T$ with increasing k.

The dominant eigenvector of $\mathbf{A}^T$ and the corresponding eigenvalue can be computed using the *power iteration* approach whose pseudo-code is shown in Algorithm 4.1. The method starts with the vector $\mathbf{p}_0$, which can be initialized to the vector $(1, 1, \ldots, 1)^T \in \mathbb{R}^n$. In each iteration, we multiply on the left by $\mathbf{A}^T$, and scale the intermediate $\mathbf{p}_k$ vector by dividing it by the maximum entry $\mathbf{p}_k[i]$ in $\mathbf{p}_k$ to prevent numeric overflow. The ratio of the maximum entry in iteration k to that in $k-1$, given as $\lambda = \frac{\mathbf{p}_k[i]}{\mathbf{p}_{k-1}[i]}$, yields an estimate for the eigenvalue. The iterations continue until the difference between successive eigenvector estimates falls below some threshold $\epsilon > 0$.

ALGORITHM 4.1. Power Iteration Method: Dominant Eigenvector

POWERITERATION $(\mathbf{A}, \epsilon)$:

$k \leftarrow 0$ // iteration
$\mathbf{p}_0 \leftarrow \mathbf{1} \in \mathbb{R}^n$ // initial vector
repeat
  $k \leftarrow k+1$
  $\mathbf{p}_k \leftarrow \mathbf{A}^T \mathbf{p}_{k-1}$ // eigenvector estimate
  $i \leftarrow \arg\max_j \{\mathbf{p}_k[j]\}$ // maximum value index
  $\lambda \leftarrow \mathbf{p}_k[i] / \mathbf{p}_{k-1}[i]$ // eigenvalue estimate
  $\mathbf{p}_k \leftarrow \frac{1}{\mathbf{p}_k[i]} \mathbf{p}_k$ // scale vector
until $\|\mathbf{p}_k - \mathbf{p}_{k-1}\| \leq \epsilon$
$\mathbf{p} \leftarrow \frac{1}{\|\mathbf{p}_k\|} \mathbf{p}_k$ // normalize eigenvector
return $\mathbf{p}, \lambda$

Table 4.2. Power method via scaling

$\mathbf{p}_0$	$\mathbf{p}_1$	$\mathbf{p}_2$	$\mathbf{p}_3$
$\begin{pmatrix}1\\1\\1\\1\\1\end{pmatrix}$	$\begin{pmatrix}1\\2\\2\\1\\2\end{pmatrix} \rightarrow \begin{pmatrix}0.5\\1\\1\\0.5\\1\end{pmatrix}$	$\begin{pmatrix}1\\1.5\\1.5\\0.5\\1.5\end{pmatrix} \rightarrow \begin{pmatrix}0.67\\1\\1\\0.33\\1\end{pmatrix}$	$\begin{pmatrix}1\\1.33\\1.33\\0.67\\1.33\end{pmatrix} \rightarrow \begin{pmatrix}0.75\\1\\1\\0.5\\1\end{pmatrix}$
λ	2	1.5	1.33
$\mathbf{p}_4$	$\mathbf{p}_5$	$\mathbf{p}_6$	$\mathbf{p}_7$
$\begin{pmatrix}1\\1.5\\1.5\\0.75\\1.5\end{pmatrix} \rightarrow \begin{pmatrix}0.67\\1\\1\\0.5\\1\end{pmatrix}$	$\begin{pmatrix}1\\1.5\\1.5\\0.67\\1.5\end{pmatrix} \rightarrow \begin{pmatrix}0.67\\1\\1\\0.44\\1\end{pmatrix}$	$\begin{pmatrix}1\\1.44\\1.44\\0.67\\1.44\end{pmatrix} \rightarrow \begin{pmatrix}0.69\\1\\1\\0.46\\1\end{pmatrix}$	$\begin{pmatrix}1\\1.46\\1.46\\0.69\\1.46\end{pmatrix} \rightarrow \begin{pmatrix}0.68\\1\\1\\0.47\\1\end{pmatrix}$
1.5	1.5	1.444	1.462

Example 4.6. Consider the example shown in Figure 4.6. Starting with an initial prestige vector $\mathbf{p}_0 = (1,1,1,1,1)^T$, in Table 4.2 we show several iterations of the power method for computing the dominant eigenvector of $\mathbf{A}^T$. In each iteration we obtain $\mathbf{p}_k = \mathbf{A}^T \mathbf{p}_{k-1}$. For example,

$$\mathbf{p}_1 = \mathbf{A}^T \mathbf{p}_0 = \begin{pmatrix}0&0&1&0&0\\0&0&0&1&1\\0&1&0&1&0\\1&0&0&0&0\\0&1&0&1&0\end{pmatrix}\begin{pmatrix}1\\1\\1\\1\\1\end{pmatrix} = \begin{pmatrix}1\\2\\2\\1\\2\end{pmatrix}$$

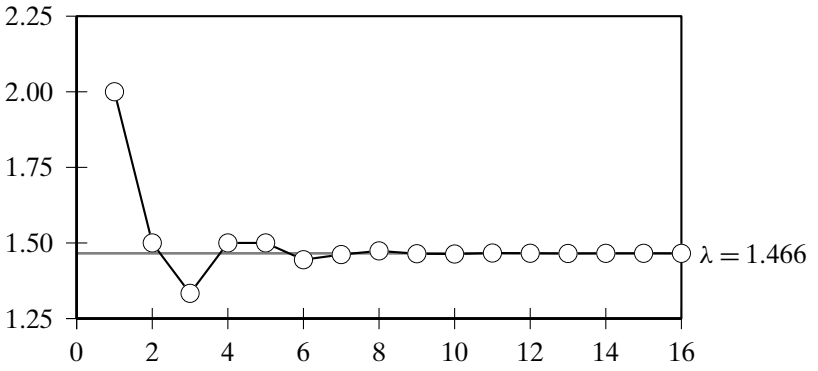


Figure 4.7. Convergence of the ratio to dominant eigenvalue.

Before the next iteration, we scale $\mathbf{p}_1$ by dividing each entry by the maximum value in the vector, which is 2 in this case, to obtain

$$\mathbf{p}_1 = \frac{1}{2}\begin{pmatrix}1\\1\\2\\1\\2\end{pmatrix} = \begin{pmatrix}0.5\\1\\1\\0.5\\1\end{pmatrix}$$

As k becomes large, we get

$$\mathbf{p}_k = \mathbf{A}^T\mathbf{p}_{k-1} \simeq \lambda\mathbf{p}_{k-1}$$

which implies that the ratio of the maximum element of $\mathbf{p}_k$ to that of $\mathbf{p}_{k-1}$ should approach λ. The table shows this ratio for successive iterations. We can see in Figure 4.7 that within 10 iterations the ratio converges to $\lambda = 1.466$. The scaled dominant eigenvector converges to

$$\mathbf{p}_k = \begin{pmatrix}1\\1.466\\1.466\\0.682\\1.466\end{pmatrix}$$

After normalizing it to be a unit vector, the dominant eigenvector is given as

$$\mathbf{p} = \begin{pmatrix}0.356\\0.521\\0.521\\0.243\\0.521\end{pmatrix}$$

Thus, in terms of prestige, v_2, v_3, and v_5 have the highest values, as all of them have indegree 2 and are pointed to by nodes with the same incoming values of prestige. On the other hand, although v_1 and v_4 have the same indegree, v_1 is ranked higher, because v_3 contributes its prestige to v_1, but v_4 gets its prestige only from v_1.

PageRank

PageRank is a method for computing the prestige or centrality of nodes in the context of Web search. The Web graph consists of pages (the nodes) connected by hyperlinks (the edges). The method uses the so-called *random surfing* assumption that a person surfing the Web randomly chooses one of the outgoing links from the current page, or with some very small probability randomly jumps to any of the other pages in the Web graph. The PageRank of a Web page is defined to be the probability of a random web surfer landing at that page. Like prestige, the PageRank of a node v recursively depends on the PageRank of other nodes that point to it.

Normalized Prestige We assume for the moment that each node u has outdegree at least 1. We discuss later how to handle the case when a node has no outgoing edges. Let $od(u) = \sum_v \mathbf{A}(u, v)$ denote the outdegree of node u. Because a random surfer can choose among any of its outgoing links, if there is a link from u to v, then the probability of visiting v from u is $\frac{1}{od(u)}$.

Starting from an initial probability or PageRank $p_0(u)$ for each node, such that

$$\sum_u p_0(u) = 1$$

we can compute an updated PageRank vector for v as follows:

$$\begin{aligned} p(v) &= \sum_u \frac{\mathbf{A}(u,v)}{od(u)} \cdot p(u) \\ &= \sum_u \mathbf{N}(u,v) \cdot p(u) \\ &= \sum_u \mathbf{N}^T(v,u) \cdot p(u) \end{aligned} \tag{4.5}$$

where $\mathbf{N}$ is the normalized adjacency matrix of the graph, given as

$$\mathbf{N}(u,v) = \begin{cases} \frac{1}{od(u)} & \text{if } (u,v) \in E \\ 0 & \text{if } (u,v) \notin E \end{cases}$$

Across all nodes, we can express the PageRank vector as follows:

$$\mathbf{p}' = \mathbf{N}^T \mathbf{p} \tag{4.6}$$

So far, the PageRank vector is essentially a normalized prestige vector.

Random Jumps In the random surfing approach, there is a small probability of jumping from one node to any of the other nodes in the graph, even if they do not have a link between them. In essence, one can think of the Web graph as a (virtual) fully connected directed graph, with an adjacency matrix given as

$$\mathbf{A}_r = \mathbf{1}_{n \times n} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{pmatrix}$$

Here $\mathbf{1}_{n\times n}$ is the $n \times n$ matrix of all ones. For the random surfer matrix, the outdegree of each node is $od(u) = n$, and the probability of jumping from u to any node v is simply $\frac{1}{od(u)} = \frac{1}{n}$. Thus, if one allows only random jumps from one node to another, the PageRank can be computed analogously to Eq. (4.5):

$$\begin{aligned} p(v) &= \sum_u \frac{\mathbf{A}_r(u,v)}{od(u)} \cdot p(u) \\ &= \sum_u \mathbf{N}_r(u,v) \cdot p(u) \\ &= \sum_u \mathbf{N}_r^T(v,u) \cdot p(u) \end{aligned}$$

where $\mathbf{N}_r$ is the normalized adjacency matrix of the fully connected Web graph, given as

$$\mathbf{N}_r = \begin{pmatrix} \frac{1}{n} & \frac{1}{n} & \cdots & \frac{1}{n} \\ \frac{1}{n} & \frac{1}{n} & \cdots & \frac{1}{n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{1}{n} & \frac{1}{n} & \cdots & \frac{1}{n} \end{pmatrix} = \frac{1}{n}\mathbf{A}_r = \frac{1}{n}\mathbf{1}_{n\times n}$$

Across all the nodes the random jump PageRank vector can be represented as

$$\mathbf{p}' = \mathbf{N}_r^T\mathbf{p}$$

PageRank The full PageRank is computed by assuming that with some small probability, α, a random Web surfer jumps from the current node u to any other random node v, and with probability $1-\alpha$ the user follows an existing link from u to v. In other words, we combine the normalized prestige vector, and the random jump vector, to obtain the final PageRank vector, as follows:

$$\begin{aligned} \mathbf{p}' &= (1-\alpha)\mathbf{N}^T\mathbf{p} + \alpha\mathbf{N}_r^T\mathbf{p} \\ &= \left((1-\alpha)\mathbf{N}^T + \alpha\mathbf{N}_r^T\right)\mathbf{p} \\ &= \mathbf{M}^T\mathbf{p} \end{aligned} \tag{4.7}$$

where $\mathbf{M} = (1-\alpha)\mathbf{N} + \alpha\mathbf{N}_r$ is the combined normalized adjacency matrix. The PageRank vector can be computed in an iterative manner, starting with an initial PageRank assignment $\mathbf{p}_0$, and updating it in each iteration using Eq. (4.7). One minor problem arises if a node u does not have any outgoing edges, that is, when $od(u) = 0$. Such a node acts like a sink for the normalized prestige score. Because there is no outgoing edge from u, the only choice u has is to simply jump to another random node. Thus, we need to make sure that if $od(u) = 0$ then for the row corresponding to u in $\mathbf{M}$, denoted as $\mathbf{M}_u$, we set $\alpha = 1$, that is,

$$\mathbf{M}_u = \begin{cases} \mathbf{M}_u & \text{if } od(u) > 0 \\ \frac{1}{n}\mathbf{1}_n^T & \text{if } od(u) = 0 \end{cases}$$

where $\mathbf{1}_n$ is the n-dimensional vector of all ones. We can use the power iteration method in Algorithm 4.1 to compute the dominant eigenvector of $\mathbf{M}^T$.

Example 4.7. Consider the graph in Figure 4.6. The normalized adjacency matrix is given as

$$\mathbf{N} = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0.5 & 0 & 0.5 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0.33 & 0.33 & 0 & 0.33 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Because there are $n = 5$ nodes in the graph, the normalized random jump adjacency matrix is given as

$$\mathbf{N}_r = \begin{pmatrix} 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \end{pmatrix}$$

Assuming that $\alpha = 0.1$, the combined normalized adjacency matrix is given as

$$\mathbf{M} = 0.9\mathbf{N} + 0.1\mathbf{N}_r = \begin{pmatrix} 0.02 & 0.02 & 0.02 & 0.92 & 0.02 \\ 0.02 & 0.02 & 0.47 & 0.02 & 0.47 \\ 0.92 & 0.02 & 0.02 & 0.02 & 0.02 \\ 0.02 & 0.32 & 0.32 & 0.02 & 0.32 \\ 0.02 & 0.92 & 0.02 & 0.02 & 0.02 \end{pmatrix}$$

Computing the dominant eigenvector and eigenvalue of $\mathbf{M}^T$ we obtain $\lambda = 1$ and

$$\mathbf{p} = \begin{pmatrix} 0.419 \\ 0.546 \\ 0.417 \\ 0.422 \\ 0.417 \end{pmatrix}$$

Node v_2 has the highest PageRank value.

Hub and Authority Scores

Note that the PageRank of a node is independent of any query that a user may pose, as it is a global value for a Web page. However, for a specific user query, a page with a high global PageRank may not be that relevant. One would like to have a query-specific notion of the PageRank or prestige of a page. The Hyperlink Induced Topic Search (HITS) method is designed to do this. In fact, it computes two values to judge the importance of a page. The *authority score* of a page is analogous to PageRank or prestige, and it depends on how many "good" pages point to it. On the other hand, the *hub score* of a page is based on how many "good" pages it points to. In other words, a page with high authority has many hub pages pointing to it, and a page with high hub score points to many pages that have high authority.

Given a user query the HITS method first uses standard search engines to retrieve the set of relevant pages. It then expands this set to include any pages that point to some page in the set, or any pages that are pointed to by some page in the set. Any pages originating from the same host are eliminated. HITS is applied only on this expanded query specific graph G.

We denote by $a(u)$ the authority score and by $h(u)$ the hub score of node u. The authority score depends on the hub score and vice versa in the following manner:

$$a(v) = \sum_u \mathbf{A}^T(v,u) \cdot h(u)$$

$$h(v) = \sum_u \mathbf{A}(v,u) \cdot a(u)$$

In matrix notation, we obtain

$$\mathbf{a}' = \mathbf{A}^T\mathbf{h}$$

$$\mathbf{h}' = \mathbf{A}\mathbf{a}$$

In fact, we can rewrite the above recursively as follows:

$$\mathbf{a}_k = \mathbf{A}^T\mathbf{h}_{k-1} = \mathbf{A}^T(\mathbf{A}\mathbf{a}_{k-1}) = (\mathbf{A}^T\mathbf{A})\mathbf{a}_{k-1}$$

$$\mathbf{h}_k = \mathbf{A}\mathbf{a}_{k-1} = \mathbf{A}(\mathbf{A}^T\mathbf{h}_{k-1}) = (\mathbf{A}\mathbf{A}^T)\mathbf{h}_{k-1}$$

In other words, as $k \to \infty$, the authority score converges to the dominant eigenvector of $\mathbf{A}^T\mathbf{A}$, whereas the hub score converges to the dominant eigenvector of $\mathbf{A}\mathbf{A}^T$. The power iteration method can be used to compute the eigenvector in both cases. Starting with an initial authority vector $\mathbf{a} = \mathbf{1}_n$, the vector of all ones, we can compute the vector $\mathbf{h} = \mathbf{A}\mathbf{a}$. To prevent numeric overflows, we scale the vector by dividing by the maximum element. Next, we can compute $\mathbf{a} = \mathbf{A}^T\mathbf{h}$, and scale it too, which completes one iteration. This process is repeated until both $\mathbf{a}$ and $\mathbf{h}$ converge.

Example 4.8. For the graph in Figure 4.6, we can iteratively compute the authority and hub score vectors, by starting with $\mathbf{a} = (1,1,1,1,1)^T$. In the first iteration, we have

$$\mathbf{h} = \mathbf{A}\mathbf{a} = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 1 \\ 3 \\ 1 \end{pmatrix}$$

After scaling by dividing by the maximum value 3, we get

$$\mathbf{h}' = \begin{pmatrix} 0.33 \\ 0.67 \\ 0.33 \\ 1 \\ 0.33 \end{pmatrix}$$

Next we update **a** as follows:

$$\mathbf{a} = \mathbf{A}^T\mathbf{h}' = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0.33 \\ 0.67 \\ 0.33 \\ 1 \\ 0.33 \end{pmatrix} = \begin{pmatrix} 0.33 \\ 1.33 \\ 1.67 \\ 0.33 \\ 1.67 \end{pmatrix}$$

After scaling by dividing by the maximum value 1.67, we get

$$\mathbf{a}' = \begin{pmatrix} 0.2 \\ 0.8 \\ 1 \\ 0.2 \\ 1 \end{pmatrix}$$

This sets the stage for the next iteration. The process continues until **a** and **h** converge to the dominant eigenvectors of $\mathbf{A}^T\mathbf{A}$ and $\mathbf{A}\mathbf{A}^T$, respectively, given as

$$\mathbf{a} = \begin{pmatrix} 0 \\ 0.46 \\ 0.63 \\ 0 \\ 0.63 \end{pmatrix} \qquad \mathbf{h} = \begin{pmatrix} 0 \\ 0.58 \\ 0 \\ 0.79 \\ 0.21 \end{pmatrix}$$

From these scores, we conclude that v_4 has the highest hub score because it points to three nodes – v_2, v_3, and v_5 – with good authority. On the other hand, both v_3 and v_5 have high authority scores, as the two nodes v_4 and v_2 with the highest hub scores point to them.

4.4 GRAPH MODELS

Surprisingly, many real-world networks exhibit certain common characteristics, even though the underlying data can come from vastly different domains, such as social networks, biological networks, telecommunication networks, and so on. A natural question is to understand the underlying processes that might give rise to such real-world networks. We consider several network measures that will allow us to compare and contrast different graph models. Real-world networks are usually *large* and *sparse*. By large we mean that the order or the number of nodes n is very large, and by sparse we mean that the graph size or number of edges $m = O(n)$. The models we study below make a similar assumption that the graphs are large and sparse.

Small-world Property

It has been observed that many real-world graphs exhibit the so-called *small-world* property that there is a short path between any pair of nodes. We say that a graph G exhibits small-world behavior if the average path length μ_L scales logarithmically with

the number of nodes in the graph, that is, if

$$\mu_L \propto \log n$$

where n is the number of nodes in the graph. A graph is said to have *ultra-small-world* property if the average path length is much smaller than $\log n$, that is, if $\mu_L \ll \log n$.

Scale-free Property

In many real-world graphs it has been observed that the empirical degree distribution $f(k)$ exhibits a *scale-free* behavior captured by a power-law relationship with k, that is, the probability that a node has degree k satisfies the condition

$$f(k) \propto k^{-\gamma} \tag{4.8}$$

Intuitively, a power law indicates that the vast majority of nodes have very small degrees, whereas there are a few "hub" nodes that have high degrees, that is, they connect to or interact with lots of nodes. A power-law relationship leads to a scale-free or scale invariant behavior because scaling the argument by some constant c does not change the proportionality. To see this, let us rewrite Eq. (4.8) as an equality by introducing a proportionality constant α that does not depend on k, that is,

$$f(k) = \alpha k^{-\gamma} \tag{4.9}$$

Then we have

$$f(ck) = \alpha(ck)^{-\gamma} = (\alpha c^{-\gamma})k^{-\gamma} \propto k^{-\gamma}$$

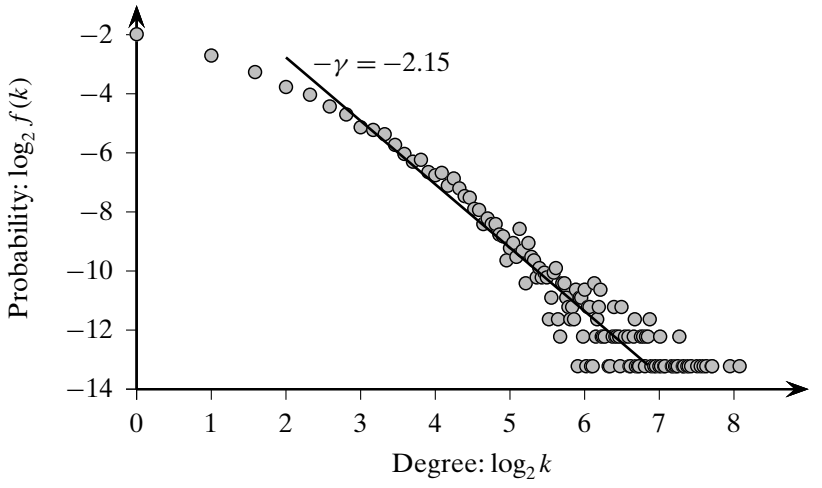
Also, taking the logarithm on both sides of Eq. (4.9) gives

$$\log f(k) = \log(\alpha k^{-\gamma})$$
$$\text{or } \log f(k) = -\gamma \log k + \log \alpha$$

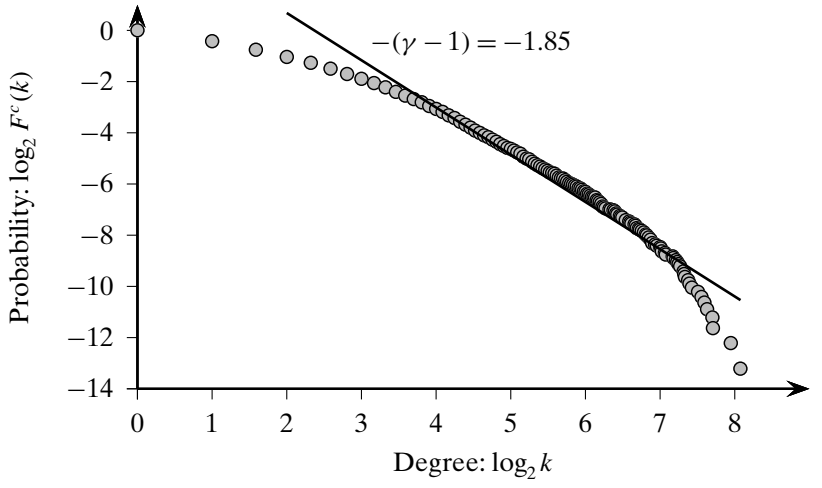
which is the equation of a straight line in the log-log plot of k versus $f(k)$, with $-\gamma$ giving the slope of the line. Thus, the usual approach to check whether a graph has scale-free behavior is to perform a least-square fit of the points $\big(\log k, \log f(k)\big)$ to a line, as illustrated in Figure 4.8a.

In practice, one of the problems with estimating the degree distribution for a graph is the high level of noise for the higher degrees, where frequency counts are the lowest. One approach to address the problem is to use the cumulative degree distribution $F(k)$, which tends to smooth out the noise. In particular, we use $F^c(k) = 1 - F(k)$, which gives the probability that a randomly chosen node has degree greater than k. If $f(k) \propto k^{-\gamma}$, and assuming that $\gamma > 1$, we have

$$\begin{aligned} F^c(k) = 1 - F(k) &= 1 - \sum_0^k f(x) = \sum_k^\infty f(x) = \sum_k^\infty x^{-\gamma} \\ &\simeq \int_k^\infty x^{-\gamma} dx = \left. \frac{x^{-\gamma+1}}{-\gamma+1} \right|_k^\infty = \frac{1}{(\gamma-1)} \cdot k^{-(\gamma-1)} \\ &\propto k^{-(\gamma-1)} \end{aligned}$$



(a) Degree distribution



(b) Cumulative degree distribution

Figure 4.8. Degree distribution and its cumulative distribution.

In other words, the log-log plot of $F^c(k)$ versus k will also be a power law with slope $-(\gamma - 1)$ as opposed to $-\gamma$. Owing to the smoothing effect, plotting $\log k$ versus $\log F^c(k)$ and observing the slope gives a better estimate of the power law, as illustrated in Figure 4.8b.

Clustering Effect

Real-world graphs often also exhibit a *clustering effect*, that is, two nodes are more likely to be connected if they share a common neighbor. The clustering effect is captured by a high clustering coefficient for the graph G. Let $C(k)$ denote the average clustering coefficient for all nodes with degree k; then the clustering effect also

manifests itself as a power-law relationship between $C(k)$ and k:

$$C(k) \propto k^{-\gamma}$$

In other words, a log-log plot of k versus $C(k)$ exhibits a straight line behavior with negative slope $-\gamma$. Intuitively, the power-law behavior indicates hierarchical clustering of the nodes. That is, nodes that are sparsely connected (i.e., have smaller degrees) are part of highly clustered areas (i.e., have higher average clustering coefficients). Further, only a few hub nodes (with high degrees) connect these clustered areas (the hub nodes have smaller clustering coefficients).

Example 4.9. Figure 4.8a plots the degree distribution for a graph of human protein interactions, where each node is a protein and each edge indicates if the two incident proteins interact experimentally. The graph has $n = 9521$ nodes and $m = 37,060$ edges. A linear relationship between $\log k$ and $\log f(k)$ is clearly visible, although very small and very large degree values do not fit the linear trend. The best fit line after ignoring the extremal degrees yields a value of $\gamma = 2.15$. The plot of $\log k$ versus $\log F^c(k)$ makes the linear fit quite prominent. The slope obtained here is $-(\gamma - 1) = 1.85$, that is, $\gamma = 2.85$. We can conclude that the graph exhibits scale-free behavior (except at the degree extremes), with γ somewhere between 2 and 3, as is typical of many real-world graphs.

The diameter of the graph is $d(G) = 14$, which is very close to $\log_2 n = \log_2(9521) = 13.22$. The network is thus small-world.

Figure 4.9 plots the average clustering coefficient as a function of degree. The log-log plot has a very weak linear trend, as observed from the line of best fit that gives a slope of $-\gamma = -0.55$. We can conclude that the graph exhibits weak hierarchical clustering behavior.

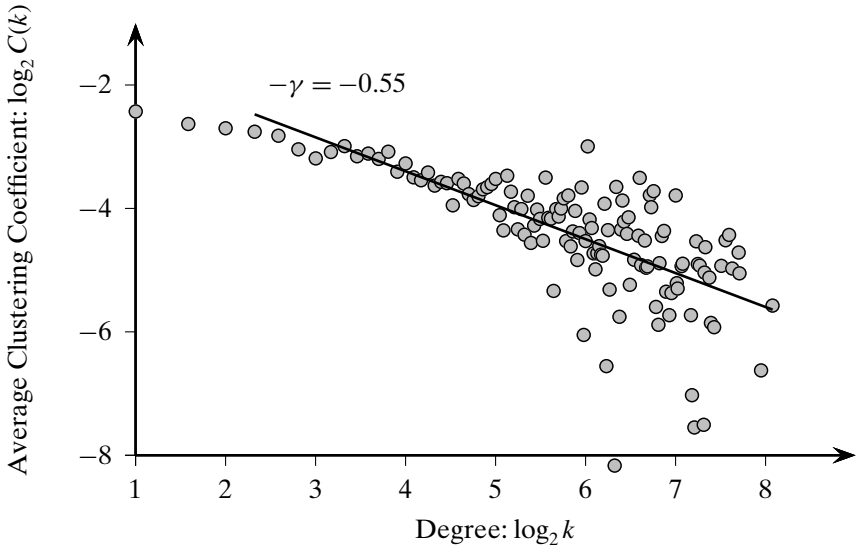


Figure 4.9. Average clustering coefficient distribution.

4.4.1 Erdös–Rényi Random Graph Model

The Erdös–Rényi (ER) model generates a random graph such that any of the possible graphs with a fixed number of nodes and edges has equal probability of being chosen.

The ER model has two parameters: the number of nodes n and the number of edges m. Let M denote the maximum number of edges possible among the n nodes, that is,

$$M = \binom{n}{2} = \frac{n(n-1)}{2}$$

The ER model specifies a collection of graphs $\mathcal{G}(n, m)$ with n nodes and m edges, such that each graph $G \in \mathcal{G}$ has equal probability of being selected:

$$P(G) = \frac{1}{\binom{M}{m}} = \binom{M}{m}^{-1}$$

where $\binom{M}{m}$ is the number of possible graphs with m edges (with n nodes) corresponding to the ways of choosing the m edges out of a total of M possible edges.

Let $V = \{v_1, v_2, \ldots, v_n\}$ denote the set of n nodes. The ER method chooses a random graph $G = (V, E) \in \mathcal{G}$ via a generative process. At each step, it randomly selects two distinct vertices $v_i, v_j \in V$, and adds an edge (v_i, v_j) to E, provided the edge is not already in the graph G. The process is repeated until exactly m edges have been added to the graph.

Let X be a random variable denoting the degree of a node for $G \in \mathcal{G}$. Let p denote the probability of an edge in G, which can be computed as

$$p = \frac{m}{M} = \frac{m}{\binom{n}{2}} = \frac{2m}{n(n-1)}$$

Average Degree

For any given node in G its degree can be at most $n - 1$ (because we do not allow loops). Because p is the probability of an edge for any node, the random variable X, corresponding to the degree of a node, follows a binomial distribution with probability of success p, given as

$$f(k) = P(X = k) = \binom{n-1}{k} p^k (1-p)^{n-1-k}$$

The average degree μ_d is then given as the expected value of X:

$$\mu_d = E[X] = (n-1)p$$

We can also compute the variance of the degrees among the nodes by computing the variance of X:

$$\sigma_d^2 = var(X) = (n-1)p(1-p)$$

Degree Distribution

To obtain the degree distribution for large and sparse random graphs, we need to derive an expression for $f(k) = P(X = k)$ as $n \to \infty$. Assuming that $m = O(n)$, we

can write $p = \frac{m}{n(n-1)/2} = \frac{O(n)}{n(n-1)/2} = \frac{1}{O(n)} \to 0$. In other words, we are interested in the asymptotic behavior of the graphs as $n \to \infty$ and $p \to 0$.

Under these two trends, notice that the expected value and variance of X can be rewritten as

$$E[X] = (n-1)p \simeq np \text{ as } n \to \infty$$
$$var(X) = (n-1)p(1-p) \simeq np \text{ as } n \to \infty \text{ and } p \to 0$$

In other words, for large and sparse random graphs the expectation and variance of X are the same:

$$E[X] = var(X) = np$$

and the binomial distribution can be approximated by a Poisson distribution with parameter λ, given as

$$f(k) = \frac{\lambda^k e^{-\lambda}}{k!}$$

where $\lambda = np$ represents both the expected value and variance of the distribution. Using Stirling's approximation of the factorial $k! \simeq k^k e^{-k}\sqrt{2\pi k}$ we obtain

$$f(k) = \frac{\lambda^k e^{-\lambda}}{k!} \simeq \frac{\lambda^k e^{-\lambda}}{k^k e^{-k}\sqrt{2\pi k}} = \frac{e^{-\lambda}}{\sqrt{2\pi}} \frac{(\lambda e)^k}{\sqrt{k} k^k}$$

In other words, we have

$$f(k) \propto \alpha^k k^{-\frac{1}{2}} k^{-k}$$

for $\alpha = \lambda e = npe$. We conclude that large and sparse random graphs follow a Poisson degree distribution, which does not exhibit a power-law relationship. Thus, in one crucial respect, the ER random graph model is not adequate to describe real-world scale-free graphs.

Clustering Coefficient

Let us consider a node v_i in G with degree k. The clustering coefficient of v_i is given as

$$C(v_i) = \frac{2m_i}{k(k-1)}$$

where $k = n_i$ also denotes the number of nodes and m_i denotes the number of edges in the subgraph induced by neighbors of v_i. However, because p is the probability of an edge, the expected number of edges m_i among the neighbors of v_i is simply

$$m_i = \frac{pk(k-1)}{2}$$

Thus, we obtain

$$C(v_i) = \frac{2m_i}{k(k-1)} = p$$

In other words, the expected clustering coefficient across all nodes of all degrees is uniform, and thus the overall clustering coefficient is also uniform:

$$C(G) = \frac{1}{n}\sum_i C(v_i) = p$$

Furthermore, for sparse graphs we have $p \to 0$, which in turn implies that $C(G) = C(v_i) \to 0$. Thus, large random graphs have no clustering effect whatsoever, which is contrary to many real-world networks.

Diameter

We saw earlier that the expected degree of a node is $\mu_d = \lambda$, which means that within one hop from a given node, we can reach λ other nodes. Because each of the neighbors of the initial node also has average degree λ, we can approximate the number of nodes that are two hops away as λ^2. In general, at a coarse level of approximation (i.e., ignoring shared neighbors), we can estimate the number of nodes at a distance of k hops away from a starting node v_i as λ^k. However, because there are a total of n distinct vertices in the graph, we have

$$\sum_{k=1}^{t} \lambda^k = n$$

where t denotes the maximum number of hops from v_i. We have

$$\sum_{k=1}^{t} \lambda^k = \frac{\lambda^{t+1} - 1}{\lambda - 1} \simeq \lambda^t$$

Plugging into the expression above, we have

$$\lambda^t \simeq n \quad \text{or}$$

$$t \log \lambda \simeq \log n \quad \text{which implies}$$

$$t \simeq \frac{\log n}{\log \lambda} \propto \log n$$

Because the path length from a node to the farthest node is bounded by t, it follows that the diameter of the graph is also bounded by that value, that is,

$$d(G) \propto \log n$$

assuming that the expected degree λ is fixed. We can thus conclude that random graphs satisfy at least one property of real-world graphs, namely that they exhibit small-world behavior.

4.4.2 Watts–Strogatz Small-world Graph Model

The random graph model fails to exhibit a high clustering coefficient, but it is small-world. The Watts–Strogatz (WS) model tries to explicitly model high local clustering by starting with a regular network in which each node is connected to its k neighbors on the right and left, assuming that the initial n vertices are arranged in a large circular backbone. Such a network will have a high clustering coefficient, but will not be small-world. Surprisingly, adding a small amount of randomness in the regular network by randomly *rewiring* some of the edges or by adding a small fraction of random edges leads to the emergence of the small-world phenomena.

The WS model starts with n nodes arranged in a circular layout, with each node connected to its immediate left and right neighbors. The edges in the initial layout are

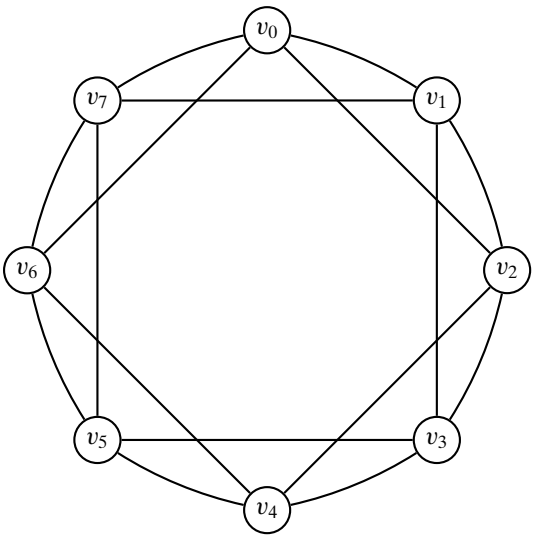


Figure 4.10. Watts–Strogatz regular graph: $n = 8, k = 2$.

called *backbone* edges. Each node has edges to an additional $k-1$ neighbors to the left and right. Thus, the WS model starts with a *regular* graph of degree $2k$, where each node is connected to its k neighbors on the right and k neighbors on the left, as illustrated in Figure 4.10.

Clustering Coefficient and Diameter of Regular Graph

Consider the subgraph G_v induced by the $2k$ neighbors of a node v. The clustering coefficient of v is given as

$$C(v) = \frac{m_v}{M_v} \tag{4.10}$$

where m_v is the actual number of edges, and M_v is the maximum possible number of edges, among the neighbors of v.

To compute m_v, consider some node r_i that is at a distance of i hops (with $1 \leq i \leq k$) from v to the right, considering only the backbone edges. The node r_i has edges to $k-i$ of its immediate right neighbors (restricted to the right neighbors of v), and to $k-1$ of its left neighbors (all k left neighbors, excluding v). Owing to the symmetry about v, a node l_i that is at a distance of i backbone hops from v to the left has the same number of edges. Thus, the degree of any node in G_v that is i backbone hops away from v is given as

$$d_i = (k-i) + (k-1) = 2k - i - 1$$

Because each edge contributes to the degree of its two incident nodes, summing the degrees of all neighbors of v, we obtain

$$2m_v = 2\left(\sum_{i=1}^{k} 2k - i - 1\right)$$

$$m_v = 2k^2 - \frac{k(k+1)}{2} - k$$
$$m_v = \frac{3}{2}k(k-1) \tag{4.11}$$

On the other hand, the number of possible edges among the $2k$ neighbors of v is given as

$$M_v = \binom{2k}{2} = \frac{2k(2k-1)}{2} = k(2k-1)$$

Plugging the expressions for m_v and M_v into Eq. (4.10), the clustering coefficient of a node v is given as

$$C(v) = \frac{m_v}{M_v} = \frac{3k-3}{4k-2}$$

As k increases, the clustering coefficient approaches $\frac{3}{4}$ because $C(G) = C(v) \to \frac{3}{4}$ as $k \to \infty$.

The WS regular graph thus has a high clustering coefficient. However, it does not satisfy the small-world property. To see this, note that along the backbone, the farthest node from v has a distance of at most $\frac{n}{2}$ hops. Further, because each node is connected to k neighbors on either side, one can reach the farthest node in at most $\frac{n/2}{k}$ hops. More precisely, the diameter of a regular WS graph is given as

$$d(G) = \begin{cases} \left\lceil \frac{n}{2k} \right\rceil & \text{if } n \text{ is even} \\ \left\lceil \frac{n-1}{2k} \right\rceil & \text{if } n \text{ is odd} \end{cases}$$

The regular graph has a diameter that scales linearly in the number of nodes, and thus it is not small-world.

Random Perturbation of Regular Graph

Edge Rewiring Starting with the regular graph of degree $2k$, the WS model perturbs the regular structure by adding some randomness to the network. One approach is to randomly rewire edges with probability r. That is, for each edge (u, v) in the graph, with probability r, replace v with another randomly chosen node avoiding loops and duplicate edges. Because the WS regular graph has $m = kn$ total edges, after rewiring, rm of the edges are random, and $(1-r)m$ are regular.

Edge Shortcuts An alternative approach is that instead of rewiring edges, we add a few *shortcut* edges between random pairs of nodes, as shown in Figure 4.11. The total number of random shortcut edges added to the network is given as $mr = knr$, so that r can be considered as the probability, per edge, of adding a shortcut edge. The total number of edges in the graph is then simply $m + mr = (1+r)m = (1+r)kn$. Because $r \in [0, 1]$, the number of edges then lies in the range $[kn, 2kn]$.

In either approach, if the probability r of rewiring or adding shortcut edges is $r = 0$, then we are left with the original regular graph, with high clustering coefficient, but with no small-world property. On the other hand, if the rewiring or shortcut probability $r = 1$, the regular structure is disrupted, and the graph approaches a random graph, with little to no clustering effect, but with small-world property. Surprisingly, introducing

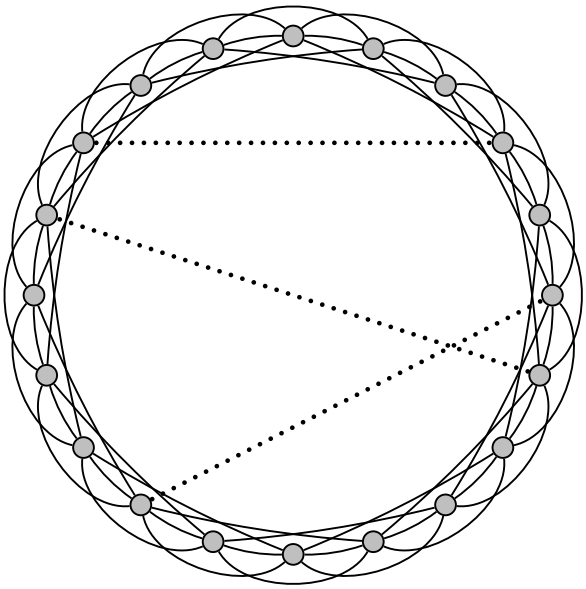


Figure 4.11. Watts–Strogatz graph ($n = 20$, $k = 3$): shortcut edges are shown dotted.

only a small amount of randomness leads to a significant change in the regular network. As one can see in Figure 4.11, the presence of a few long-range shortcuts reduces the diameter of the network significantly. That is, even for a low value of r, the WS model retains most of the regular local clustering structure, but at the same time becomes small-world.

Properties of Watts–Strogatz Graphs

Degree Distribution Let us consider the shortcut approach, which is easier to analyze. In this approach, each vertex has degree at least $2k$. In addition there are the shortcut edges, which follow a binomial distribution. Each node can have $n' = n - 2k - 1$ additional shortcut edges, so we take n' as the number of independent trials to add edges. Because a node has degree $2k$, with shortcut edge probability of r, we expect roughly $2kr$ shortcuts from that node, but the node can connect to at most $n - 2k - 1$ other nodes. Thus, we can take the probability of success as

$$p = \frac{2kr}{n - 2k - 1} = \frac{2kr}{n'} \tag{4.12}$$

Let X denote the random variable denoting the number of shortcuts for each node. Then the probability of a node with j shortcut edges is given as

$$f(j) = P(X = j) = \binom{n'}{j} p^j (1-p)^{n'-j}$$

with $E[X] = n'p = 2kr$. The expected degree of each node in the network is therefore

$$2k + E[X] = 2k + 2kr = 2k(1 + r)$$

It is clear that the degree distribution of the WS graph does not adhere to a power law. Thus, such networks are not scale-free.

Clustering Coefficient After the shortcut edges have been added, each node v has expected degree $2k(1+r)$, that is, it is on average connected to $2kr$ new neighbors, in addition to the $2k$ original ones. The number of possible edges among v's neighbors is given as

$$M_v = \frac{2k(1+r)(2k(1+r)-1)}{2} = (1+r)k(4kr+2k-1)$$

Because the regular WS graph remains intact even after adding shortcuts, the neighbors of v retain all $\frac{3k(k-1)}{2}$ initial edges, as given in Eq. (4.11). In addition, some of the shortcut edges may link pairs of nodes among v's neighbors. Let Y be the random variable that denotes the number of shortcut edges present among the $2k(1+r)$ neighbors of v; then Y follows a binomial distribution with probability of success p, as given in Eq. (4.12). Thus, the expected number of shortcut edges is given as

$$E[Y] = pM_v$$

Let m_v be the random variable corresponding to the actual number of edges present among v's neighbors, whether regular or shortcut edges. The expected number of edges among the neighbors of v is then given as

$$E[m_v] = E\left[\frac{3k(k-1)}{2} + Y\right] = \frac{3k(k-1)}{2} + pM_v$$

Because the binomial distribution is essentially concentrated around the mean, we can now approximate the clustering coefficient by using the expected number of edges, as follows:

$$\begin{aligned} C(v) \simeq \frac{E[m_v]}{M_v} &= \frac{\frac{3k(k-1)}{2} + pM_v}{M_v} = \frac{3k(k-1)}{2M_v} + p \\ &= \frac{3(k-1)}{(1+r)(4kr+2(2k-1))} + \frac{2kr}{n-2k-1} \end{aligned}$$

using the value of p given in Eq. (4.12). For large graphs we have $n \to \infty$, so we can drop the second term above, to obtain

$$C(v) \simeq \frac{3(k-1)}{(1+r)(4kr+2(2k-1))} = \frac{3k-3}{4k-2+2r(2kr+4k-1)} \tag{4.13}$$

As $r \to 0$, the above expression becomes equivalent to Eq. (4.10). Thus, for small values of r the clustering coefficient remains high.

Diameter Deriving an analytical expression for the diameter of the WS model with random edge shortcuts is not easy. Instead we resort to an empirical study of the behavior of WS graphs when a small number of random shortcuts are added. In Example 4.10 we find that small values of shortcut edge probability r are enough to reduce the diameter from $O(n)$ to $O(\log n)$. The WS model thus leads to graphs that are small-world and that also exhibit the clustering effect. However, the WS graphs do not display a scale-free degree distribution.

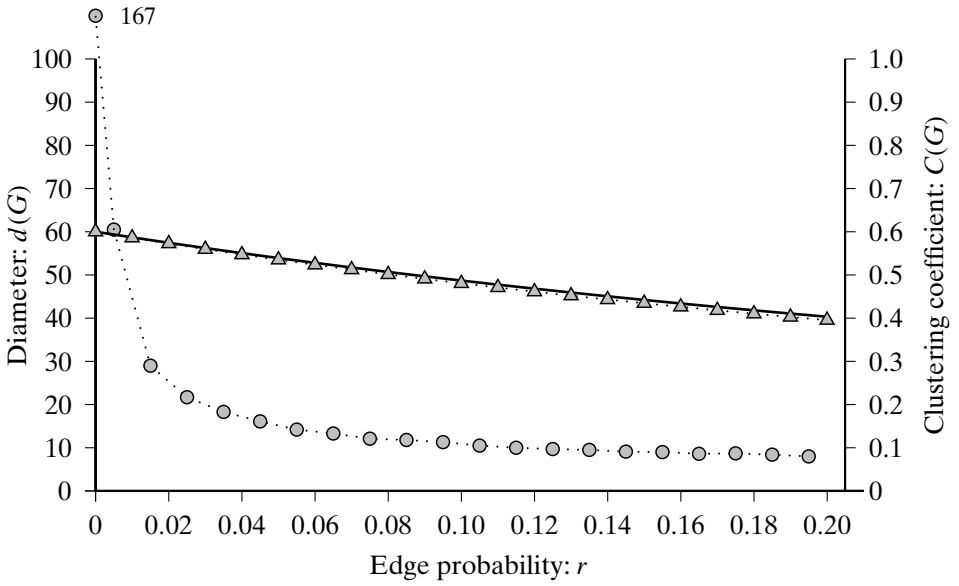


Figure 4.12. Watts-Strogatz model: diameter (circles) and clustering coefficient (triangles).

Example 4.10. Figure 4.12 shows a simulation of the WS model, for a graph with $n = 1000$ vertices and $k = 3$. The x-axis shows different values of the probability r of adding random shortcut edges. The diameter values are shown as circles using the left y-axis, whereas the clustering values are shown as triangles using the right y-axis. These values are the averages over 10 runs of the WS model. The solid line gives the clustering coefficient from the analytical formula in Eq. (4.13), which is in perfect agreement with the simulation values.

The initial regular graph has diameter

$$d(G) = \left\lceil \frac{n}{2k} \right\rceil = \left\lceil \frac{1000}{6} \right\rceil = 167$$

and its clustering coefficient is given as

$$C(G) = \frac{3(k-1)}{2(2k-1)} = \frac{6}{10} = 0.6$$

We can observe that the diameter quickly reduces, even with very small edge addition probability. For $r = 0.005$, the diameter is 61. For $r = 0.1$, the diameter shrinks to 11, which is on the same scale as $O(\log_2 n)$ because $\log_2 1000 \simeq 10$. On the other hand, we can observe that clustering coefficient remains high. For $r = 0.1$, the clustering coefficient is 0.48. Thus, the simulation study confirms that the addition of even a small number of random shortcut edges reduces the diameter of the WS regular graph from $O(n)$ (large-world) to $O(\log n)$ (small-world). At the same time the graph retains its local clustering property.

4.4.3 Barabási–Albert Scale-free Model

The Barabási–Albert (BA) model tries to capture the scale-free degree distributions of real-world graphs via a generative process that adds new nodes and edges at each time step. Further, the edge growth is based on the concept of *preferential attachment*; that is, edges from the new vertex are more likely to link to nodes with higher degrees. For this reason the model is also known as the *rich get richer* approach. The BA model mimics a dynamically growing graph by adding new vertices and edges at each time-step $t = 1, 2, \ldots$. Let G_t denote the graph at time t, and let n_t denote the number of nodes, and m_t the number of edges in G_t.

Initialization

The BA model starts at time-step $t = 0$, with an initial graph G_0 with n_0 nodes and m_0 edges. Each node in G_0 should have degree at least 1; otherwise it will never be chosen for preferential attachment. We will assume that each node has initial degree 2, being connected to its left and right neighbors in a circular layout. Thus $m_0 = n_0$.

Growth and Preferential Attachment

The BA model derives a new graph G_{t+1} from G_t by adding exactly one new node u and adding $q \leq n_0$ new edges from u to q distinct nodes $v_j \in G_t$, where node v_j is chosen with probability $\pi_t(v_j)$ proportional to its degree in G_t, given as

$$\pi_t(v_j) = \frac{d_j}{\sum_{v_i \in G_t} d_i} \tag{4.14}$$

Because only one new vertex is added at each step, the number of nodes in G_t is given as

$$n_t = n_0 + t$$

Further, because exactly q new edges are added at each time-step, the number of edges in G_t is given as

$$m_t = m_0 + qt$$

Because the sum of the degrees is two times the number of edges in the graph, we have

$$\sum_{v_i \in G_t} d(v_i) = 2m_t = 2(m_0 + qt)$$

We can thus rewrite Eq. (4.14) as

$$\pi_t(v_j) = \frac{d_j}{2(m_0 + qt)} \tag{4.15}$$

As the network grows, owing to preferential attachment, one intuitively expects high degree hubs to emerge.

Example 4.11. Figure 4.13 shows a graph generated according to the BA model, with parameters $n_0 = 3, q = 2$, and $t = 12$. Initially, at time $t = 0$, the graph has $n_0 = 3$ vertices, namely $\{v_0, v_1, v_2\}$ (shown in gray), connected by $m_0 = 3$ edges (shown in bold). At each time step $t = 1, \ldots, 12$, vertex v_{t+2} is added to the growing network

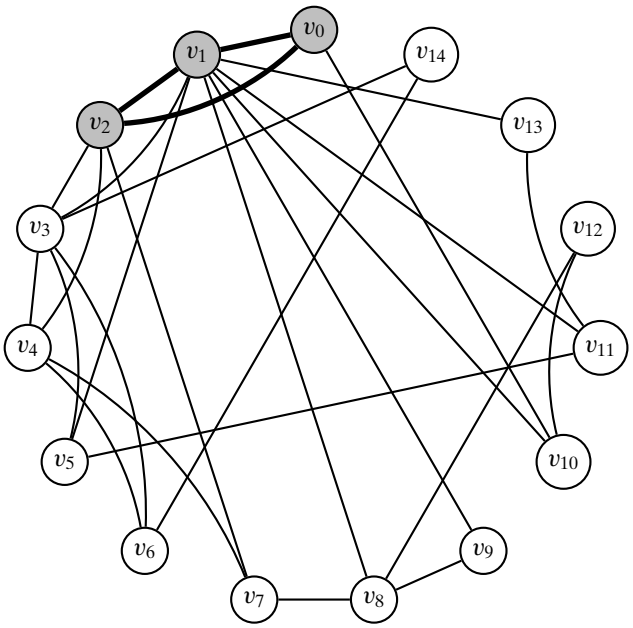


Figure 4.13. Barabási–Albert graph ($n_0 = 3$, $q = 2$, $t = 12$).

and is connected to $q = 2$ vertices chosen with a probability proportional to their degree.

For example, at $t = 1$, vertex v_3 is added, with edges to v_1 and v_2, chosen according to the distribution

$$\pi_0(v_i) = 1/3 \text{ for } i = 0, 1, 2$$

At $t = 2$, v_4 is added. Using Eq. (4.15), nodes v_2 and v_3 are preferentially chosen according to the probability distribution

$$\pi_1(v_0) = \pi_1(v_3) = \frac{2}{10} = 0.2$$

$$\pi_1(v_1) = \pi_1(v_2) = \frac{3}{10} = 0.3$$

The final graph after $t = 12$ time-steps shows the emergence of some hub nodes, such as v_1 (with degree 9) and v_3 (with degree 6).

Degree Distribution

We now study two different approaches to estimate the degree distribution for the BA model, namely the discrete approach, and the continuous approach.

Discrete Approach The discrete approach is also called the *master-equation* method. Let X_t be a random variable denoting the degree of a node in G_t, and let $f_t(k)$ denote the probability mass function for X_t. That is, $f_t(k)$ is the degree distribution for the

graph G_t at time-step t. Simply put, $f_t(k)$ is the fraction of nodes with degree k at time t. Let n_t denote the number of nodes and m_t the number of edges in G_t. Further, let $n_t(k)$ denote the number of nodes with degree k in G_t. Then we have

$$f_t(k) = \frac{n_t(k)}{n_t}$$

Because we are interested in large real-world graphs, as $t \to \infty$, the number of nodes and edges in G_t can be approximated as

$$\begin{aligned} n_t &= n_0 + t \simeq t \\ m_t &= m_0 + qt \simeq qt \end{aligned} \tag{4.16}$$

Based on Eq. (4.14), at time-step $t+1$, the probability $\pi_t(k)$ that some node with degree k in G_t is chosen for preferential attachment can be written as

$$\pi_t(k) = \frac{k \cdot n_t(k)}{\sum_i i \cdot n_t(i)}$$

Dividing the numerator and denominator by n_t, we have

$$\pi_t(k) = \frac{k \cdot \frac{n_t(k)}{n_t}}{\sum_i i \cdot \frac{n_t(i)}{n_t}} = \frac{k \cdot f_t(k)}{\sum_i i \cdot f_t(i)} \tag{4.17}$$

Note that the denominator is simply the expected value of X_t, that is, the mean degree in G_t, because

$$E[X_t] = \mu_d(G_t) = \sum_i i \cdot f_t(i) \tag{4.18}$$

Note also that in any graph the average degree is given as

$$\mu_d(G_t) = \frac{\sum_i d_i}{n_t} = \frac{2m_t}{n_t} \simeq \frac{2qt}{t} = 2q \tag{4.19}$$

where we used Eq. (4.16), that is, $m_t = qt$. Equating Eqs. (4.18) and (4.19), we can rewrite the preferential attachment probability [Eq. (4.17)] for a node of degree k as

$$\pi_t(k) = \frac{k \cdot f_t(k)}{2q} \tag{4.20}$$

We now consider the change in the number of nodes with degree k, when a new vertex u joins the growing network at time-step $t+1$. The net change in the number of nodes with degree k is given as the number of nodes with degree k at time $t+1$ minus the number of nodes with degree k at time t, given as

$$(n_t + 1) \cdot f_{t+1}(k) - n_t \cdot f_t(k)$$

Using the approximation that $n_t \simeq t$ from Eq. (4.16), the net change in degree k nodes is

$$(n_t + 1) \cdot f_{t+1}(k) - n_t \cdot f_t(k) = (t+1) \cdot f_{t+1}(k) - t \cdot f_t(k) \tag{4.21}$$

The number of nodes with degree k increases whenever u connects to a vertex v_i of degree $k-1$ in G_t, as in this case v_i will have degree k in G_{t+1}. Over the q edges added

at time $t+1$, the number of nodes with degree $k-1$ in G_t that are chosen to connect to u is given as

$$q\pi_t(k-1) = \frac{q \cdot (k-1) \cdot f_t(k-1)}{2q} = \frac{1}{2} \cdot (k-1) \cdot f_t(k-1) \tag{4.22}$$

where we use Eq. (4.20) for $\pi_t(k-1)$. Note that Eq. (4.22) holds only when $k > q$. This is because v_i must have degree at least q, as each node that is added at time $t \geq 1$ has initial degree q. Therefore, if $d_i = k-1$, then $k-1 \geq q$ implies that $k > q$ (we can also ensure that the initial n_0 edges have degree q by starting with clique of size $n_0 = q+1$).

At the same time, the number of nodes with degree k decreases whenever u connects to a vertex v_i with degree k in G_t, as in this case v_i will have a degree $k+1$ in G_{t+1}. Using Eq. (4.20), over the q edges added at time $t+1$, the number of nodes with degree k in G_t that are chosen to connect to u is given as

$$q \cdot \pi_t(k) = \frac{q \cdot k \cdot f_t(k)}{2q} = \frac{1}{2} \cdot k \cdot f_t(k) \tag{4.23}$$

Based on the preceding discussion, when $k > q$, the net change in the number of nodes with degree k is given as the difference between Eqs. (4.22) and (4.23) in G_t:

$$q \cdot \pi_t(k-1) - q \cdot \pi_t(k) = \frac{1}{2} \cdot (k-1) \cdot f_t(k-1) - \frac{1}{2} k \cdot f_t(k) \tag{4.24}$$

Equating Eqs. (4.21) and (4.24) we obtain the master equation for $k > q$:

$$(t+1) \cdot f_{t+1}(k) - t \cdot f_t(k) = \frac{1}{2} \cdot (k-1) \cdot f_t(k-1) - \frac{1}{2} \cdot k \cdot f_t(k) \tag{4.25}$$

On the other hand, when $k = q$, assuming that there are no nodes in the graph with degree less than q, then only the newly added node contributes to an increase in the number of nodes with degree $k = q$ by one. However, if u connects to an existing node v_i with degree k, then there will be a decrease in the number of degree k nodes because in this case v_i will have degree $k+1$ in G_{t+1}. The net change in the number of nodes with degree k is therefore given as

$$1 - q \cdot \pi_t(k) = 1 - \frac{1}{2} \cdot k \cdot f_t(k) \tag{4.26}$$

Equating Eqs. (4.21) and (4.26) we obtain the master equation for the boundary condition $k = q$:

$$(t+1) \cdot f_{t+1}(k) - t \cdot f_t(k) = 1 - \frac{1}{2} \cdot k \cdot f_t(k) \tag{4.27}$$

Our goal is now to obtain the stationary or time-invariant solutions for the master equations. In other words, we study the solution when

$$f_{t+1}(k) = f_t(k) = f(k) \tag{4.28}$$

The stationary solution gives the degree distribution that is independent of time.

Let us first derive the stationary solution for $k = q$. Substituting Eq. (4.28) into Eq. (4.27) and setting $k = q$, we obtain

$$\begin{aligned}(t+1)\cdot f(q) - t\cdot f(q) &= 1 - \frac{1}{2}\cdot q\cdot f(q)\\ 2f(q) &= 2 - q\cdot f(q), \text{ which implies that}\\ f(q) &= \frac{2}{q+2} \end{aligned} \tag{4.29}$$

The stationary solution for $k > q$ gives us a recursion for $f(k)$ in terms of $f(k-1)$:

$$\begin{aligned}(t+1)\cdot f(k) - t\cdot f(k) &= \frac{1}{2}\cdot(k-1)\cdot f(k-1) - \frac{1}{2}\cdot k\cdot f(k)\\ 2f(k) &= (k-1)\cdot f(k-1) - k\cdot f(k), \text{ which implies that}\\ f(k) &= \left(\frac{k-1}{k+2}\right)\cdot f(k-1) \end{aligned} \tag{4.30}$$

Expanding (4.30) until the boundary condition $k = q$ yields

$$\begin{aligned} f(k) &= \frac{(k-1)}{(k+2)}\cdot f(k-1)\\ &= \frac{(k-1)(k-2)}{(k+2)(k+1)}\cdot f(k-2)\\ &\vdots\\ &= \frac{(k-1)(k-2)(k-3)(k-4)\cdots(q+3)(q+2)(q+1)(q)}{(k+2)(k+1)(k)(k-1)\cdots(q+6)(q+5)(q+4)(q+3)}\cdot f(q)\\ &= \frac{(q+2)(q+1)q}{(k+2)(k+1)k}\cdot f(q) \end{aligned}$$

Plugging in the stationary solution for $f(q)$ from Eq. (4.29) gives the general solution

$$f(k) = \frac{(q+2)(q+1)q}{(k+2)(k+1)k}\cdot\frac{2}{(q+2)} = \frac{2q(q+1)}{k(k+1)(k+2)}$$

For constant q and large k, it is easy to see that the degree distribution scales as

$$f(k) \propto k^{-3} \tag{4.31}$$

In other words, the BA model yields a power-law degree distribution with $\gamma = 3$, especially for large degrees.

Continuous Approach The continuous approach is also called the *mean-field* method. In the BA model, the vertices that are added early on tend to have a higher degree, because they have more chances to acquire connections from the vertices that are added to the network at a later time. The time dependence of the degree of a vertex can be approximated as a continuous random variable. Let $k_i = d_t(i)$ denote the degree of vertex v_i at time t. At time t, the probability that the newly added node u links to

v_i is given as $\pi_t(i)$. Further, the change in v_i's degree per time-step is given as $q \cdot \pi_t(i)$. Using the approximation that $n_t \simeq t$ and $m_t \simeq qt$ from Eq. (4.16), the rate of change of k_i with time can be written as

$$\frac{dk_i}{dt} = q \cdot \pi_t(i) = q \cdot \frac{k_i}{2qt} = \frac{k_i}{2t}$$

Rearranging the terms in the preceding equation $\frac{dk_i}{dt} = \frac{k_i}{2t}$ and integrating on both sides, we have

$$\begin{aligned}
\int \frac{1}{k_i} dk_i &= \int \frac{1}{2t} dt \\
\ln k_i &= \frac{1}{2} \ln t + C \\
e^{\ln k_i} &= e^{\ln t^{1/2}} \cdot e^C, \text{ which implies} \\
k_i &= \alpha \cdot t^{1/2}
\end{aligned} \tag{4.32}$$

where C is the constant of integration, and thus $\alpha = e^C$ is also a constant.

Let t_i denote the time when node i was added to the network. Because the initial degree for any node is q, we obtain the boundary condition that $k_i = q$ at time $t = t_i$. Plugging these into Eq. (4.32), we get

$$\begin{aligned}
k_i &= \alpha \cdot t_i^{1/2} = q, \text{ which implies that} \\
\alpha &= \frac{q}{\sqrt{t_i}}
\end{aligned} \tag{4.33}$$

Substituting Eq. (4.33) into Eq. (4.32) leads to the particular solution

$$k_i = \alpha \cdot \sqrt{t} = q \cdot \sqrt{t/t_i} \tag{4.34}$$

Intuitively, this solution confirms the rich-gets-richer phenomenon. It suggests that if a node v_i is added early to the network (i.e., t_i is small), then as time progresses (i.e., t gets larger), the degree of v_i keeps on increasing (as a square root of the time t).

Let us now consider the probability that the degree of v_i at time t is less than some value k, i.e., $P(k_i < k)$. Note that if $k_i < k$, then by Eq. (4.34), we have

$$\begin{aligned}
k_i &< k \\
q \cdot \sqrt{\frac{t}{t_i}} &< k \\
\frac{t}{t_i} &< \frac{k^2}{q^2}, \text{ which implies that} \\
t_i &> \frac{q^2 t}{k^2}
\end{aligned}$$

Thus, we can write

$$P(k_i < k) = P\left(t_i > \frac{q^2 t}{k^2}\right) = 1 - P\left(t_i \leq \frac{q^2 t}{k^2}\right)$$

In other words, the probability that node v_i has degree less than k is the same as the probability that the time t_i at which v_i enters the graph is greater than $\frac{q^2}{k^2}t$, which in turn can be expressed as 1 minus the probability that t_i is less than or equal to $\frac{q^2}{k^2}t$.

Note that vertices are added to the graph at a uniform rate of one vertex per time-step, that is, $\frac{1}{n_t} \simeq \frac{1}{t}$. Thus, the probability that t_i is less than or equal to $\frac{q^2}{k^2}t$ is given as

$$\begin{aligned} P(k_i < k) &= 1 - P\left(t_i \leq \frac{q^2 t}{k^2}\right) \\ &= 1 - \frac{q^2 t}{k^2} \cdot \frac{1}{t} \\ &= 1 - \frac{q^2}{k^2} \end{aligned}$$

Because v_i is any generic node in the graph, $P(k_i < k)$ can be considered to be the cumulative degree distribution $F_t(k)$ at time t. We can obtain the degree distribution $f_t(k)$ by taking the derivative of $F_t(k)$ with respect to k to obtain

$$\begin{aligned} f_t(k) = \frac{d}{dk}F_t(k) = \frac{d}{dk}P(k_i < k) \\ &= \frac{d}{dk}\left(1 - \frac{q^2}{k^2}\right) \\ &= 0 - \left(\frac{k^2 \cdot 0 - q^2 \cdot 2k}{k^4}\right) \\ &= \frac{2q^2}{k^3} \\ &\propto k^{-3} \end{aligned} \tag{4.35}$$

In Eq. (4.35) we made use of the quotient rule for computing the derivative of the quotient $f(k) = \frac{g(k)}{h(k)}$, given as

$$\frac{df(k)}{dk} = \frac{h(k) \cdot \frac{dg(k)}{dk} - g(k) \cdot \frac{dh(k)}{dk}}{h(k)^2}$$

Here $g(k) = q^2$ and $h(k) = k^2$, and $\frac{dg(k)}{dk} = 0$ and $\frac{dh(k)}{dk} = 2k$.

Note that the degree distribution from the continuous approach, given in Eq. (4.35), is very close to that obtained from the discrete approach given in Eq. (4.31). Both solutions confirm that the degree distribution is proportional to k^{-3}, which gives the power-law behavior with $\gamma = 3$.

Clustering Coefficient and Diameter

Closed form solutions for the clustering coefficient and diameter for the BA model are difficult to derive. It has been shown that the diameter of BA graphs scales as

$$d(G_t) = O\left(\frac{\log n_t}{\log\log n_t}\right)$$

suggesting that they exhibit *ultra-small-world* behavior, when $q > 1$. Further, the expected clustering coefficient of the BA graphs scales as

$$E[C(G_t)] = O\left(\frac{(\log n_t)^2}{n_t}\right)$$

which is only slightly better than the clustering coefficient for random graphs, which scale as $O(n_t^{-1})$. In Example 4.12, we empirically study the clustering coefficient and diameter for random instances of the BA model with a given set of parameters.

Example 4.12. Figure 4.14 plots the empirical degree distribution obtained as the average of 10 different BA graphs generated with the parameters $n_0 = 3$, $q = 3$, and for $t = 997$ time-steps, so that the final graph has $n = 1000$ vertices. The slope of the line in the log-log scale confirms the existence of a power law, with the slope given as $-\gamma = -2.64$.

The average clustering coefficient over the 10 graphs was $C(G) = 0.019$, which is not very high, indicating that the BA model does not capture the clustering effect. On the other hand, the average diameter was $d(G) = 6$, indicating ultra-small-world behavior.

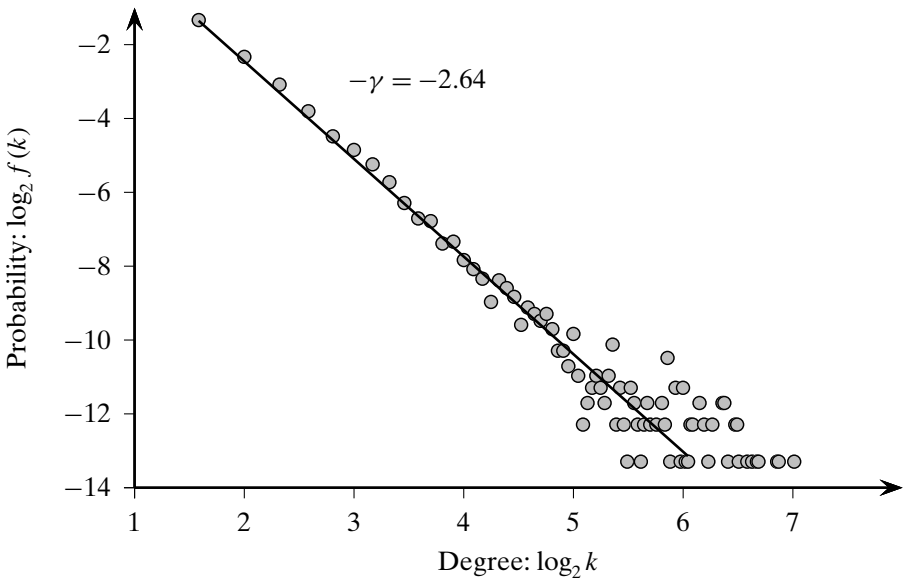


Figure 4.14. Barabási–Albert model ($n_0 = 3, t = 997, q = 3$): degree distribution.

4.5 FURTHER READING

The theory of random graphs was founded in Erdős and Rényi (1959); for a detailed treatment of the topic see Bollobás (2001). Alternative graph models for real-world networks were proposed in Watts and Strogatz (1998) and Barabási and Albert (1999). One of the first comprehensive books on graph data analysis was Wasserman and Faust (1994). More recent books on network science Lewis (2009) and Newman (2010). For PageRank see Brin and Page (1998), and for the hubs and authorities approach see Kleinberg (1999). For an up-to-date treatment of the patterns, laws, and models (including the RMat generator) for real-world networks, see Chakrabarti and Faloutsos (2012).

Barabási, A.-L. and Albert, R. (1999). "Emergence of scaling in random networks." *Science*, 286 (5439): 509–512.

Bollobás, B. (2001). *Random Graphs,* 2nd ed. Vol. 73. New York: Cambridge University Press.

Brin, S. and Page, L. (1998). "The anatomy of a large-scale hypertextual Web search engine." *Computer Networks and ISDN Systems*, 30 (1): 107–117.

Chakrabarti, D. and Faloutsos, C. (2012). "Graph Mining: Laws, Tools, and Case Studies.", *Synthesis Lectures on Data Mining and Knowledge Discovery*, 7(1): 1–207. San Rafael, CA: Morgan & Claypool Publishers.

Erdős, P. and Rényi, A. (1959). "On random graphs." *Publicationes Mathematicae Debrecen*, 6, 290–297.

Kleinberg, J. M. (1999). "Authoritative sources in a hyperlinked environment." *Journal of the ACM*, 46 (5): 604–632.

Lewis, T. G. (2009). *Network Science: Theory and Applications*. Hoboken. NJ: John Wiley & Sons.

Newman, M. (2010). *Networks: An Introduction*. Oxford: Oxford University Press.

Wasserman, S. and Faust, K. (1994). *Social Network Analysis: Methods and Applications*. Structural Analysis in the Social Sciences. New York: Cambridge University Press.

Watts, D. J. and Strogatz, S. H. (1998). "Collective dynamics of 'small-world' networks." *Nature*, 393 (6684): 440–442.

4.6 EXERCISES

Q1. Given the graph in Figure 4.15, find the fixed-point of the prestige vector.

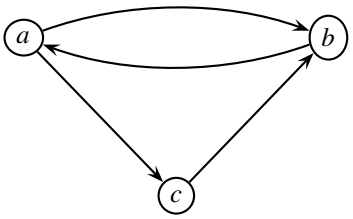


Figure 4.15. Graph for Q1

Q2. Given the graph in Figure 4.16, find the fixed-point of the authority and hub vectors.

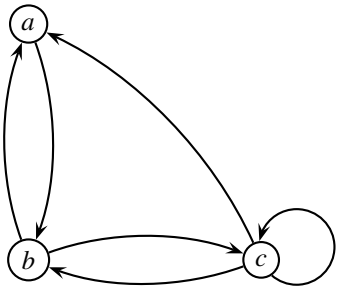


Figure 4.16. Graph for Q2.

Q3. Consider the double star graph given in Figure 4.17 with n nodes, where only nodes 1 and 2 are connected to all other vertices, and there are no other links. Answer the following questions (treating n as a variable).

(a) What is the degree distribution for this graph?
(b) What is the mean degree?
(c) What is the clustering coefficient for vertex 1 and vertex 3?
(d) What is the clustering coefficient $C(G)$ for the entire graph? What happens to the clustering coefficient as $n \to \infty$?
(e) What is the transitivity $T(G)$ for the graph? What happens to $T(G)$ and $n \to \infty$?
(f) What is the average path length for the graph?
(g) What is the betweenness value for node 1?
(h) What is the degree variance for the graph?

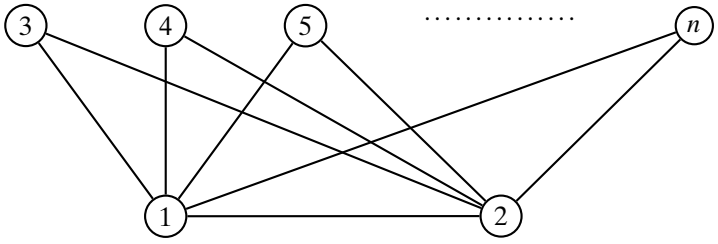


Figure 4.17. Graph for Q3.

Q4. Consider the graph in Figure 4.18. Compute the hub and authority score vectors. Which nodes are the hubs and which are the authorities?

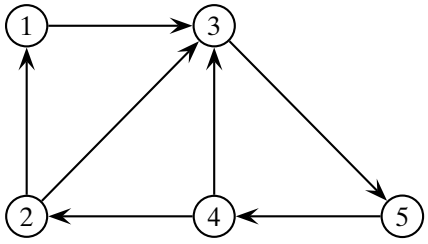


Figure 4.18. Graph for Q4.

Q5. Prove that in the BA model at time-step $t+1$, the probability $\pi_t(k)$ that some node with degree k in G_t is chosen for preferential attachment is given as

$$\pi_t(k) = \frac{k \cdot n_t(k)}{\sum_i i \cdot n_t(i)}$$

CHAPTER 5 Kernel Methods

Before we can mine data, it is important to first find a suitable data representation that facilitates data analysis. For example, for complex data such as text, sequences, images, and so on, we must typically extract or construct a set of attributes or features, so that we can represent the data instances as multivariate vectors. That is, given a data instance $\mathbf{x}$ (e.g., a sequence), we need to find a mapping ϕ, so that $\phi(\mathbf{x})$ is the vector representation of $\mathbf{x}$. Even when the input data is a numeric data matrix, if we wish to discover nonlinear relationships among the attributes, then a nonlinear mapping ϕ may be used, so that $\phi(\mathbf{x})$ represents a vector in the corresponding high-dimensional space comprising nonlinear attributes. We use the term *input space* to refer to the data space for the input data $\mathbf{x}$ and *feature space* to refer to the space of mapped vectors $\phi(\mathbf{x})$. Thus, given a set of data objects or instances $\mathbf{x}_i$, and given a mapping function ϕ, we can transform them into feature vectors $\phi(\mathbf{x}_i)$, which then allows us to analyze complex data instances via numeric analysis methods.

Example 5.1 (Sequence-based Features). Consider a dataset of DNA sequences over the alphabet $\Sigma = \{A, C, G, T\}$. One simple feature space is to represent each sequence in terms of the probability distribution over symbols in Σ. That is, given a sequence $\mathbf{x}$ with length $|\mathbf{x}| = m$, the mapping into feature space is given as

$$\phi(\mathbf{x}) = \{P(A), P(C), P(G), P(T)\}$$

where $P(s) = \frac{n_s}{m}$ is the probability of observing symbol $s \in \Sigma$, and n_s is the number of times s appears in sequence $\mathbf{x}$. Here the input space is the set of sequences Σ^*, and the feature space is $\mathbb{R}^4$. For example, if $\mathbf{x} = ACAGCAGTA$, with $m = |\mathbf{x}| = 9$, since A occurs four times, C and G occur twice, and T occurs once, we have

$$\phi(\mathbf{x}) = (4/9, 2/9, 2/9, 1/9) = (0.44, 0.22, 0.22, 0.11)$$

Likewise, for another sequence $\mathbf{y} = AGCAAGCGAG$, we have

$$\phi(\mathbf{y}) = (4/10, 2/10, 4/10, 0) = (0.4, 0.2, 0.4, 0)$$

The mapping ϕ now allows one to compute statistics over the data sample to make inferences about the population. For example, we may compute the mean

symbol composition. We can also define the distance between any two sequences, for example,

$$\delta(\mathbf{x}, \mathbf{y}) = \big\|\phi(\mathbf{x}) - \phi(\mathbf{y})\big\| = \sqrt{(0.44-0.4)^2 + (0.22-0.2)^2 + (0.22-0.4)^2 + (0.11-0)^2} = 0.22$$

We can compute larger feature spaces by considering, for example, the probability distribution over all substrings or words of size up to k over the alphabet Σ, and so on.

Example 5.2 (Nonlinear Features). As an example of a nonlinear mapping consider the mapping ϕ that takes as input a vector $\mathbf{x} = (x_1, x_2)^T \in \mathbb{R}^2$ and maps it to a "quadratic" feature space via the nonlinear mapping

$$\phi(\mathbf{x}) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)^T \in \mathbb{R}^3$$

For example, the point $\mathbf{x} = (5.9, 3)^T$ is mapped to the vector

$$\phi(\mathbf{x}) = (5.9^2, 3^2, \sqrt{2} \cdot 5.9 \cdot 3)^T = (34.81, 9, 25.03)^T$$

The main benefit of this transformation is that we may apply well-known linear analysis methods in the feature space. However, because the features are nonlinear combinations of the original attributes, this allows us to mine nonlinear patterns and relationships.

Whereas mapping into feature space allows one to analyze the data via algebraic and probabilistic modeling, the resulting feature space is usually very high-dimensional; it may even be infinite dimensional. Thus, transforming all the input points into feature space can be very expensive, or even impossible. Because the dimensionality is high, we also run into the curse of dimensionality highlighted later in Chapter 6.

Kernel methods avoid explicitly transforming each point $\mathbf{x}$ in the input space into the mapped point $\phi(\mathbf{x})$ in the feature space. Instead, the input objects are represented via their $n \times n$ pairwise similarity values. The similarity function, called a *kernel*, is chosen so that it represents a dot product in some high-dimensional feature space, yet it can be computed without directly constructing $\phi(\mathbf{x})$. Let $\mathcal{I}$ denote the input space, which can comprise any arbitrary set of objects, and let $\mathbf{D} = \{\mathbf{x}_i\}_{i=1}^n \subset \mathcal{I}$ be a dataset comprising n objects in the input space. We can represent the pairwise similarity values between points in $\mathbf{D}$ via the $n \times n$ *kernel matrix*, defined as

$$\mathbf{K} = \begin{pmatrix} K(\mathbf{x}_1, \mathbf{x}_1) & K(\mathbf{x}_1, \mathbf{x}_2) & \cdots & K(\mathbf{x}_1, \mathbf{x}_n) \\ K(\mathbf{x}_2, \mathbf{x}_1) & K(\mathbf{x}_2, \mathbf{x}_2) & \cdots & K(\mathbf{x}_2, \mathbf{x}_n) \\ \vdots & \vdots & \ddots & \vdots \\ K(\mathbf{x}_n, \mathbf{x}_1) & K(\mathbf{x}_n, \mathbf{x}_2) & \cdots & K(\mathbf{x}_n, \mathbf{x}_n) \end{pmatrix}$$

where $K: \mathcal{I} \times \mathcal{I} \to \mathbb{R}$ is a *kernel function* on any two points in input space. However, we require that K corresponds to a dot product in some feature space. That is, for any

$\mathbf{x}_i, \mathbf{x}_j \in \mathcal{I}$, the kernel function should satisfy the condition

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) \tag{5.1}$$

where $\phi: \mathcal{I} \to \mathcal{F}$ is a mapping from the input space $\mathcal{I}$ to the feature space $\mathcal{F}$. Intuitively, this means that we should be able to compute the value of the dot product using the original input representation $\mathbf{x}$, without having recourse to the mapping $\phi(\mathbf{x})$. Obviously, not just any arbitrary function can be used as a kernel; a valid kernel function must satisfy certain conditions so that Eq. (5.1) remains valid, as discussed in Section 5.1.

It is important to remark that the transpose operator for the dot product applies only when $\mathcal{F}$ is a vector space. When $\mathcal{F}$ is an abstract vector space with an inner product, the kernel is written as $K(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$. However, for convenience we use the transpose operator throughout this chapter; when $\mathcal{F}$ is an inner product space it should be understood that

$$\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) \equiv \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$$

Example 5.3 (Linear and Quadratic Kernels). Consider the identity mapping, $\phi(\mathbf{x}) \to \mathbf{x}$. This naturally leads to the *linear kernel*, which is simply the dot product between two input vectors, and thus satisfies Eq. (5.1):

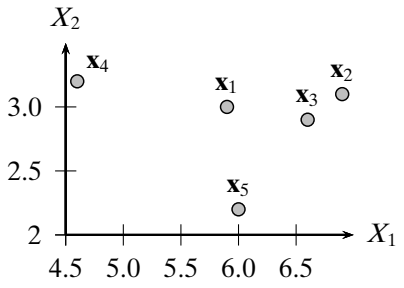
$$\phi(\mathbf{x})^T \phi(\mathbf{y}) = \mathbf{x}^T \mathbf{y} = K(\mathbf{x}, \mathbf{y})$$

For example, consider the first five points from the two-dimensional Iris dataset shown in Figure 5.1a:

$$\mathbf{x}_1 = \begin{pmatrix} 5.9 \\ 3 \end{pmatrix} \quad \mathbf{x}_2 = \begin{pmatrix} 6.9 \\ 3.1 \end{pmatrix} \quad \mathbf{x}_3 = \begin{pmatrix} 6.6 \\ 2.9 \end{pmatrix} \quad \mathbf{x}_4 = \begin{pmatrix} 4.6 \\ 3.2 \end{pmatrix} \quad \mathbf{x}_5 = \begin{pmatrix} 6 \\ 2.2 \end{pmatrix}$$

The kernel matrix for the linear kernel is shown in Figure 5.1b. For example,

$$K(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1^T \mathbf{x}_2 = 5.9 \times 6.9 + 3 \times 3.1 = 40.71 + 9.3 = 50.01$$



(a)

K	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{x}_5$
$\mathbf{x}_1$	43.81	50.01	47.64	36.74	42.00
$\mathbf{x}_2$	50.01	57.22	54.53	41.66	48.22
$\mathbf{x}_3$	47.64	54.53	51.97	39.64	45.98
$\mathbf{x}_4$	36.74	41.66	39.64	31.40	34.64
$\mathbf{x}_5$	42.00	48.22	45.98	34.64	40.84

(b)

Figure 5.1. (a) Example points. (b) Linear kernel matrix.

Consider the quadratic mapping $\phi : \mathbb{R}^2 \rightarrow \mathbb{R}^3$ from Example 5.2, that maps $\mathbf{x} = (x_1, x_2)^T$ as follows:

$$\phi(\mathbf{x}) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)^T$$

The dot product between the mapping for two input points $\mathbf{x}, \mathbf{y} \in \mathbb{R}^2$ is given as

$$\phi(\mathbf{x})^T\phi(\mathbf{y}) = x_1^2y_1^2 + x_2^2y_2^2 + 2x_1y_1x_2y_2$$

We can rearrange the preceding to obtain the (homogeneous) *quadratic kernel* function as follows:

$$\begin{aligned}\phi(\mathbf{x})^T\phi(\mathbf{y}) &= x_1^2y_1^2 + x_2^2y_2^2 + 2x_1y_1x_2y_2\\ &= (x_1y_1 + x_2y_2)^2\\ &= (\mathbf{x}^T\mathbf{y})^2\\ &= K(\mathbf{x}, \mathbf{y})\end{aligned}$$

We can thus see that the dot product in feature space can be computed by evaluating the kernel in input space, without explicitly mapping the points into feature space. For example, we have

$$\phi(\mathbf{x}_1) = (5.9^2, 3^2, \sqrt{2} \cdot 5.9 \cdot 3)^T = (34.81, 9, 25.03)^T$$

$$\phi(\mathbf{x}_2) = (6.9^2, 3.1^2, \sqrt{2} \cdot 6.9 \cdot 3.1)^T = (47.61, 9.61, 30.25)^T$$

$$\phi(\mathbf{x}_1)^T\phi(\mathbf{x}_2) = 34.81 \times 47.61 + 9 \times 9.61 + 25.03 \times 30.25 = 2501$$

We can verify that the homogeneous quadratic kernel gives the same value

$$K(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{x}_1^T\mathbf{x}_2)^2 = (50.01)^2 = 2501$$

We shall see that many data mining methods can be *kernelized*, that is, instead of mapping the input points into feature space, the data can be represented via the $n \times n$ kernel matrix $\mathbf{K}$, and all relevant analysis can be performed over $\mathbf{K}$. This is usually done via the so-called *kernel trick*, that is, show that the analysis task requires only dot products $\phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$ in feature space, which can be replaced by the corresponding kernel $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$ that can be computed efficiently in input space. Once the kernel matrix has been computed, we no longer even need the input points $\mathbf{x}_i$, as all operations involving only dot products in the feature space can be performed over the $n \times n$ kernel matrix $\mathbf{K}$. An immediate consequence is that when the input data is the typical $n \times d$ numeric matrix $\mathbf{D}$ and we employ the linear kernel, the results obtained by analyzing $\mathbf{K}$ are equivalent to those obtained by analyzing $\mathbf{D}$ (as long as only dot products are involved in the analysis). Of course, kernel methods allow much more flexibility, as we can just as easily perform non-linear analysis by employing nonlinear kernels, or we may analyze (non-numeric) complex objects without explicitly constructing the mapping $\phi(\mathbf{x})$.

Example 5.4. Consider the five points from Example 5.3 along with the linear kernel matrix shown in Figure 5.1. The mean of the five points in feature space is simply the mean in input space, as ϕ is the identity function for the linear kernel:

$$\boldsymbol{\mu}_\phi = \sum_{i=1}^{5} \phi(\mathbf{x}_i) = \sum_{i=1}^{5} \mathbf{x}_i = (6.00, 2.88)^T$$

Now consider the squared magnitude of the mean in feature space:

$$\left\|\boldsymbol{\mu}_\phi\right\|^2 = \boldsymbol{\mu}_\phi^T \boldsymbol{\mu}_\phi = (6.0^2 + 2.88^2) = 44.29$$

Because this involves only a dot product in feature space, the squared magnitude can be computed directly from $\mathbf{K}$. As we shall see later [see Eq. (5.12)] the squared norm of the mean vector in feature space is equivalent to the average value of the kernel matrix $\mathbf{K}$. For the kernel matrix in Figure 5.1b we have

$$\frac{1}{5^2} \sum_{i=1}^{5} \sum_{j=1}^{5} K(\mathbf{x}_i, \mathbf{x}_j) = \frac{1107.36}{25} = 44.29$$

which matches the $\left\|\boldsymbol{\mu}_\phi\right\|^2$ value computed earlier. This example illustrates that operations involving dot products in feature space can be cast as operations over the kernel matrix $\mathbf{K}$.

Kernel methods offer a radically different view of the data. Instead of thinking of the data as vectors in input or feature space, we consider only the kernel values between pairs of points. The kernel matrix can also be considered as a weighted adjacency matrix for the complete graph over the n input points, and consequently there is a strong connection between kernels and graph analysis, in particular algebraic graph theory.

5.1 KERNEL MATRIX

Let $\mathcal{I}$ denote the input space, which can be any arbitrary set of data objects, and let $\mathbf{D} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\} \subset \mathcal{I}$ denote a subset of n objects in the input space. Let $\phi: \mathcal{I} \to \mathcal{F}$ be a mapping from the input space into the feature space $\mathcal{F}$, which is endowed with a dot product and norm. Let $K: \mathcal{I} \times \mathcal{I} \to \mathbb{R}$ be a function that maps pairs of input objects to their dot product value in feature space, that is, $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$, and let $\mathbf{K}$ be the $n \times n$ kernel matrix corresponding to the subset $\mathbf{D}$.

The function K is called a **positive semidefinite kernel** if and only if it is symmetric:

$$K(\mathbf{x}_i, \mathbf{x}_j) = K(\mathbf{x}_j, \mathbf{x}_i)$$

and the corresponding kernel matrix $\mathbf{K}$ for any subset $\mathbf{D} \subset \mathcal{I}$ is positive semidefinite, that is,

$$\mathbf{a}^T \mathbf{K} \mathbf{a} \geq 0, \text{ for all vectors } \mathbf{a} \in \mathbb{R}^n$$

which implies that

$$\sum_{i=1}^{n}\sum_{j=1}^{n} a_i a_j K(\mathbf{x}_i, \mathbf{x}_j) \geq 0, \text{ for all } a_i \in \mathbb{R}, i \in [1, n] \tag{5.2}$$

We first verify that if $K(\mathbf{x}_i, \mathbf{x}_j)$ represents the dot product $\phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$ in some feature space, then K is a positive semidefinite kernel. Consider any dataset $\mathbf{D}$, and let $\mathbf{K} = \{K(\mathbf{x}_i, \mathbf{x}_j)\}$ be the corresponding kernel matrix. First, K is symmetric since the dot product is symmetric, which also implies that $\mathbf{K}$ is symmetric. Second, $\mathbf{K}$ is positive semidefinite because

$$\begin{aligned}
\mathbf{a}^T\mathbf{K}\mathbf{a} &= \sum_{i=1}^{n}\sum_{j=1}^{n} a_i a_j K(\mathbf{x}_i, \mathbf{x}_j) \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n} a_i a_j \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j) \\
&= \left(\sum_{i=1}^{n} a_i \phi(\mathbf{x}_i)\right)^T \left(\sum_{j=1}^{n} a_j \phi(\mathbf{x}_j)\right) \\
&= \left\|\sum_{i=1}^{n} a_i \phi(\mathbf{x}_i)\right\|^2 \geq 0
\end{aligned}$$

Thus, K is a positive semidefinite kernel.

We now show that if we are given a positive semidefinite kernel $K: \mathcal{I} \times \mathcal{I} \to \mathbb{R}$, then it corresponds to a dot product in some feature space $\mathcal{F}$.

5.1.1 Reproducing Kernel Map

For the reproducing kernel map ϕ, we map each point $\mathbf{x} \in \mathcal{I}$ into a function in a *functional space* $\{f: \mathcal{I} \to \mathbb{R}\}$ comprising functions that map points in $\mathcal{I}$ into $\mathbb{R}$. Algebraically this space of functions is an abstract vector space where each point happens to be a function. In particular, any $\mathbf{x} \in \mathcal{I}$ in the input space is mapped to the following function:

$$\phi(\mathbf{x}) = K(\mathbf{x}, \cdot)$$

where the $\cdot$ stands for any argument in $\mathcal{I}$. That is, each object $\mathbf{x}$ in the input space gets mapped to a *feature point* $\phi(\mathbf{x})$, which is in fact a function $K(\mathbf{x}, \cdot)$ that represents its similarity to all other points in the input space $\mathcal{I}$.

Let $\mathcal{F}$ be the set of all functions or points that can be obtained as a linear combination of any subset of feature points, defined as

$$\begin{aligned}
\mathcal{F} &= span\big\{K(\mathbf{x}, \cdot) \mid \mathbf{x} \in \mathcal{I}\big\} \\
&= \left\{\mathbf{f} = f(\cdot) = \sum_{i=1}^{m} \alpha_i\, K(\mathbf{x}_i, \cdot) \;\middle|\; m \in \mathbb{N}, \alpha_i \in \mathbb{R}, \{\mathbf{x}_1, \ldots, \mathbf{x}_m\} \subseteq \mathcal{I}\right\}
\end{aligned}$$

We use the dual notation $\mathbf{f}$ and $f(\cdot)$ interchangeably to emphasize the fact that each point $\mathbf{f}$ in the feature space is in fact a function $f(\cdot)$. Note that by definition the feature point $\phi(\mathbf{x}) = K(\mathbf{x}, \cdot)$ belongs to $\mathcal{F}$.

Let $\mathbf{f}, \mathbf{g} \in \mathcal{F}$ be any two points in feature space:

$$\mathbf{f} = f(\cdot) = \sum_{i=1}^{m_a} \alpha_i \, K(\mathbf{x}_i, \cdot) \qquad \mathbf{g} = g(\cdot) = \sum_{j=1}^{m_b} \beta_j \, K(\mathbf{x}_j, \cdot)$$

Define the dot product between two points as

$$\mathbf{f}^T\mathbf{g} = f(\cdot)^T g(\cdot) = \sum_{i=1}^{m_a}\sum_{j=1}^{m_b} \alpha_i \beta_j K(\mathbf{x}_i, \mathbf{x}_j) \tag{5.3}$$

We emphasize that the notation $\mathbf{f}^T\mathbf{g}$ is only a convenience; it denotes the inner product $\langle \mathbf{f}, \mathbf{g} \rangle$ because $\mathcal{F}$ is an abstract vector space, with an inner product as defined above.

We can verify that the dot product is *bilinear*, that is, linear in both arguments, because

$$\mathbf{f}^T\mathbf{g} = \sum_{i=1}^{m_a}\sum_{j=1}^{m_b} \alpha_i \, \beta_j \, K(\mathbf{x}_i, \mathbf{x}_j) = \sum_{i=1}^{m_a} \alpha_i \, g(\mathbf{x}_i) = \sum_{j=1}^{m_b} \beta_j \, f(\mathbf{x}_j)$$

The fact that K is positive semidefinite implies that

$$\|\mathbf{f}\|^2 = \mathbf{f}^T\mathbf{f} = \sum_{i=1}^{m_a}\sum_{j=1}^{m_a} \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}) \geq 0$$

Thus, the space $\mathcal{F}$ is a *pre-Hilbert space*, defined as a normed inner product space, because it is endowed with a symmetric bilinear dot product and a norm. By adding the limit points of all Cauchy sequences that are convergent, $\mathcal{F}$ can be turned into a *Hilbert space*, defined as a normed inner product space that is complete. However, showing this is beyond the scope of this chapter.

The space $\mathcal{F}$ has the so-called *reproducing property*, that is, we can evaluate a function $f(\cdot) = \mathbf{f}$ at a point $\mathbf{x} \in \mathcal{I}$ by taking the dot product of $\mathbf{f}$ with $\phi(\mathbf{x})$, that is,

$$\mathbf{f}^T\phi(\mathbf{x}) = f(\cdot)^T K(\mathbf{x}, \cdot) = \sum_{i=1}^{m_a} \alpha_i \, K(\mathbf{x}_i, \mathbf{x}) = f(\mathbf{x})$$

For this reason, the space $\mathcal{F}$ is also called a *reproducing kernel Hilbert space*.

All we have to do now is to show that $K(\mathbf{x}_i, \mathbf{x}_j)$ corresponds to a dot product in the feature space $\mathcal{F}$. This is indeed the case, because using Eq. (5.3) for any two feature points $\phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \in \mathcal{F}$ their dot product is given as

$$\phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j) = K(\mathbf{x}_i, \cdot)^T K(\mathbf{x}_j, \cdot) = K(\mathbf{x}_i, \mathbf{x}_j)$$

The reproducing kernel map shows that any positive semidefinite kernel corresponds to a dot product in some feature space. This means we can apply well known algebraic and geometric methods to understand and analyze the data in these spaces.

Empirical Kernel Map

The reproducing kernel map ϕ maps the input space into a potentially infinite dimensional feature space. However, given a dataset $\mathbf{D} = \{\mathbf{x}_i\}_{i=1}^n$, we can obtain a finite

dimensional mapping by evaluating the kernel only on points in $\mathbf{D}$. That is, define the map ϕ as follows:

$$\phi(\mathbf{x}) = \Big(K(\mathbf{x}_1, \mathbf{x}), K(\mathbf{x}_2, \mathbf{x}), \ldots, K(\mathbf{x}_n, \mathbf{x})\Big)^T \in \mathbb{R}^n$$

which maps each point $\mathbf{x} \in \mathcal{I}$ to the n-dimensional vector comprising the kernel values of $\mathbf{x}$ with each of the objects $\mathbf{x}_i \in \mathbf{D}$. We can define the dot product in feature space as

$$\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) = \sum_{k=1}^{n} K(\mathbf{x}_k, \mathbf{x}_i) K(\mathbf{x}_k, \mathbf{x}_j) = \mathbf{K}_i^T \mathbf{K}_j \tag{5.4}$$

where $\mathbf{K}_i$ denotes the ith column of $\mathbf{K}$, which is also the same as the ith row of $\mathbf{K}$ (considered as a column vector), as $\mathbf{K}$ is symmetric. However, for ϕ to be a valid map, we require that $\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) = K(\mathbf{x}_i, \mathbf{x}_j)$, which is clearly not satisfied by Eq. (5.4). One solution is to replace $\mathbf{K}_i^T \mathbf{K}_j$ in Eq. (5.4) with $\mathbf{K}_i^T \mathbf{A} \mathbf{K}_j$ for some positive semidefinite matrix $\mathbf{A}$ such that

$$\mathbf{K}_i^T \mathbf{A} \mathbf{K}_j = \mathbf{K}(\mathbf{x}_i, \mathbf{x}_j)$$

If we can find such an $\mathbf{A}$, it would imply that over all pairs of mapped points we have

$$\left\{\mathbf{K}_i^T \mathbf{A} \mathbf{K}_j\right\}_{i,j=1}^{n} = \left\{K(\mathbf{x}_i, \mathbf{x}_j)\right\}_{i,j=1}^{n}$$

which can be written compactly as

$$\mathbf{K}\mathbf{A}\mathbf{K} = \mathbf{K}$$

This immediately suggests that we take $\mathbf{A} = \mathbf{K}^{-1}$, the (pseudo) inverse of the kernel matrix $\mathbf{K}$. The modified map ϕ, called the *empirical kernel map*, is then defined as

$$\phi(\mathbf{x}) = \mathbf{K}^{-1/2} \cdot \Big(K(\mathbf{x}_1, \mathbf{x}), K(\mathbf{x}_2, \mathbf{x}), \ldots, K(\mathbf{x}_n, \mathbf{x})\Big)^T \in \mathbb{R}^n$$

so that the dot product yields

$$\begin{aligned}
\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) &= \left(\mathbf{K}^{-1/2}\, \mathbf{K}_i\right)^T \left(\mathbf{K}^{-1/2}\, \mathbf{K}_j\right) \\
&= \mathbf{K}_i^T \left(\mathbf{K}^{-1/2} \mathbf{K}^{-1/2}\right) \mathbf{K}_j \\
&= \mathbf{K}_i^T\, \mathbf{K}^{-1}\, \mathbf{K}_j
\end{aligned}$$

Over all pairs of mapped points, we have

$$\left\{\mathbf{K}_i^T \mathbf{K}^{-1}\, \mathbf{K}_j\right\}_{i,j=1}^{n} = \mathbf{K}\, \mathbf{K}^{-1}\, \mathbf{K} = \mathbf{K}$$

as desired. However, it is important to note that this empirical feature representation is valid only for the n points in $\mathbf{D}$. If points are added to or removed from $\mathbf{D}$, the kernel map will have to be updated for all points.

5.1.2 Mercer Kernel Map

In general different feature spaces can be constructed for the same kernel K. We now describe how to construct the Mercer map.

Data-specific Kernel Map

The Mercer kernel map is best understood starting from the kernel matrix for the dataset $\mathbf{D}$ in input space. Because $\mathbf{K}$ is a symmetric positive semidefinite matrix, it has real and non-negative eigenvalues, and it can be decomposed as follows:

$$\mathbf{K} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$$

where $\mathbf{U}$ is the orthonormal matrix of eigenvectors $\mathbf{u}_i = (u_{i1}, u_{i2}, \ldots, u_{in})^T \in \mathbb{R}^n$ (for $i = 1, \ldots, n$), and $\mathbf{\Lambda}$ is the diagonal matrix of eigenvalues, with both arranged in non-increasing order of the eigenvalues $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n \geq 0$:

$$\mathbf{U} = \begin{pmatrix} | & | & & | \\ \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_n \\ | & | & & | \end{pmatrix} \qquad \mathbf{\Lambda} = \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{pmatrix}$$

The kernel matrix $\mathbf{K}$ can therefore be rewritten as the spectral sum

$$\mathbf{K} = \lambda_1 \mathbf{u}_1 \mathbf{u}_1^T + \lambda_2 \mathbf{u}_2 \mathbf{u}_2^T + \cdots + \lambda_n \mathbf{u}_n \mathbf{u}_n^T$$

In particular the kernel function between $\mathbf{x}_i$ and $\mathbf{x}_j$ is given as

$$\begin{aligned} \mathbf{K}(\mathbf{x}_i, \mathbf{x}_j) &= \lambda_1 \, u_{1i} \, u_{1j} + \lambda_2 \, u_{2i} \, u_{2j} \cdots + \lambda_n \, u_{ni} \, u_{nj} \\ &= \sum_{k=1}^{n} \lambda_k \, u_{ki} \, u_{kj} \end{aligned} \tag{5.5}$$

where u_{ki} denotes the ith component of eigenvector $\mathbf{u}_k$. It follows that if we define the Mercer map ϕ as follows:

$$\phi(\mathbf{x}_i) = \left(\sqrt{\lambda_1} \, u_{1i}, \sqrt{\lambda_2} \, u_{2i}, \ldots, \sqrt{\lambda_n} \, u_{ni} \right)^T \tag{5.6}$$

then $\mathbf{K}(\mathbf{x}_i, \mathbf{x}_j)$ is a dot product in feature space between the mapped points $\phi(\mathbf{x}_i)$ and $\phi(\mathbf{x}_j)$ because

$$\begin{aligned} \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) &= \left(\sqrt{\lambda_1} \, u_{1i}, \ldots, \sqrt{\lambda_n} \, u_{ni} \right) \left(\sqrt{\lambda_1} \, u_{1j}, \ldots, \sqrt{\lambda_n} \, u_{nj} \right)^T \\ &= \lambda_1 \, u_{1i} \, u_{1j} + \cdots + \lambda_n \, u_{ni} \, u_{nj} = K(\mathbf{x}_i, \mathbf{x}_j) \end{aligned}$$

Noting that $\mathbf{U}_i = (u_{1i}, u_{2i}, \ldots, u_{ni})^T$ is the ith row of $\mathbf{U}$, we can rewrite the Mercer map ϕ as

$$\phi(\mathbf{x}_i) = \sqrt{\mathbf{\Lambda}} \mathbf{U}_i \tag{5.7}$$

Thus, the kernel value is simply the dot product between scaled rows of $\mathbf{U}$:

$$\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) = \left(\sqrt{\mathbf{\Lambda}} \mathbf{U}_i \right)^T \left(\sqrt{\mathbf{\Lambda}} \mathbf{U}_j \right) = \mathbf{U}_i^T \mathbf{\Lambda} \mathbf{U}_j$$

The Mercer map, defined equivalently in Eqs. (5.6) and (5.7), is obviously restricted to the input dataset $\mathbf{D}$, just like the empirical kernel map, and is therefore called the *data-specific Mercer kernel map*. It defines a data-specific feature space of dimensionality at most n, comprising the eigenvectors of $\mathbf{K}$.

Example 5.5. Let the input dataset comprise the five points shown in Figure 5.1a, and let the corresponding kernel matrix be as shown in Figure 5.1b. Computing the eigen-decomposition of $\mathbf{K}$, we obtain $\lambda_1 = 223.95$, $\lambda_2 = 1.29$, and $\lambda_3 = \lambda_4 = \lambda_5 = 0$. The effective dimensionality of the feature space is 2, comprising the eigenvectors $\mathbf{u}_1$ and $\mathbf{u}_2$. Thus, the matrix $\mathbf{U}$ is given as follows:

$$\mathbf{U} = \left(\begin{array}{c|rr} & \mathbf{u}_1 & \mathbf{u}_2 \\ \hline \mathbf{U}_1 & -0.442 & 0.163 \\ \mathbf{U}_2 & -0.505 & -0.134 \\ \mathbf{U}_3 & -0.482 & -0.181 \\ \mathbf{U}_4 & -0.369 & 0.813 \\ \mathbf{U}_5 & -0.425 & -0.512 \end{array}\right)$$

and we have

$$\mathbf{\Lambda} = \begin{pmatrix} 223.95 & 0 \\ 0 & 1.29 \end{pmatrix} \qquad \sqrt{\mathbf{\Lambda}} = \begin{pmatrix} \sqrt{223.95} & 0 \\ 0 & \sqrt{1.29} \end{pmatrix} = \begin{pmatrix} 14.965 & 0 \\ 0 & 1.135 \end{pmatrix}$$

The kernel map is specified via Eq. (5.7). For example, for $\mathbf{x}_1 = (5.9, 3)^T$ and $\mathbf{x}_2 = (6.9, 3.1)^T$ we have

$$\phi(\mathbf{x}_1) = \sqrt{\mathbf{\Lambda}}\mathbf{U}_1 = \begin{pmatrix} 14.965 & 0 \\ 0 & 1.135 \end{pmatrix} \begin{pmatrix} -0.442 \\ 0.163 \end{pmatrix} = \begin{pmatrix} -6.616 \\ 0.185 \end{pmatrix}$$

$$\phi(\mathbf{x}_2) = \sqrt{\mathbf{\Lambda}}\mathbf{U}_2 = \begin{pmatrix} 14.965 & 0 \\ 0 & 1.135 \end{pmatrix} \begin{pmatrix} -0.505 \\ -0.134 \end{pmatrix} = \begin{pmatrix} -7.563 \\ -0.153 \end{pmatrix}$$

Their dot product is given as

$$\begin{aligned} \phi(\mathbf{x}_1)^T \phi(\mathbf{x}_2) &= 6.616 \times 7.563 - 0.185 \times 0.153 \\ &= 50.038 - 0.028 = 50.01 \end{aligned}$$

which matches the kernel value $K(\mathbf{x}_1, \mathbf{x}_2)$ in Figure 5.1b.

Mercer Kernel Map

For compact continuous spaces, analogous to the discrete case in Eq. (5.5), the kernel value between any two points can be written as the infinite spectral decomposition

$$K(\mathbf{x}_i, \mathbf{x}_j) = \sum_{k=1}^{\infty} \lambda_k \, \mathbf{u}_k(\mathbf{x}_i) \, \mathbf{u}_k(\mathbf{x}_j)$$

where $\{\lambda_1, \lambda_2, \ldots\}$ is the infinite set of eigenvalues, and $\{\mathbf{u}_1(\cdot), \mathbf{u}_2(\cdot), \ldots\}$ is the corresponding set of orthogonal and normalized *eigenfunctions*, that is, each function $\mathbf{u}_i(\cdot)$ is a solution to the integral equation

$$\int K(\mathbf{x}, \mathbf{y}) \, \mathbf{u}_i(\mathbf{y}) \, d\mathbf{y} = \lambda_i \mathbf{u}_i(\mathbf{x})$$

and K is a continuous positive semidefinite kernel, that is, for all functions $a(\cdot)$ with a finite square integral (i.e., $\int a(\mathbf{x})^2\, d\mathbf{x} < 0$) K satisfies the condition

$$\int\int K(\mathbf{x}_1, \mathbf{x}_2)\, a(\mathbf{x}_1)\, a(\mathbf{x}_2)\, d\mathbf{x}_1\, d\mathbf{x}_2 \geq 0$$

We can see that this positive semidefinite kernel for compact continuous spaces is analogous to the the discrete kernel in Eq. (5.2). Further, similarly to the data-specific Mercer map [Eq. (5.6)], the general Mercer kernel map is given as

$$\phi(\mathbf{x}_i) = \left(\sqrt{\lambda_1}\, \mathbf{u}_1(\mathbf{x}_i), \sqrt{\lambda_2}\, \mathbf{u}_2(\mathbf{x}_i), \ldots\right)^T$$

with the kernel value being equivalent to the dot product between two mapped points:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

5.2 VECTOR KERNELS

We now consider two of the most commonly used vector kernels in practice. Kernels that map an (input) vector space into another (feature) vector space are called *vector kernels*. For multivariate input data, the input vector space will be the d-dimensional real space $\mathbb{R}^d$. Let $\mathbf{D}$ comprise n input points $\mathbf{x}_i \in \mathbb{R}^d$, for $i = 1, 2, \ldots, n$. Commonly used (nonlinear) kernel functions over vector data include the polynomial and Gaussian kernels, as described next.

Polynomial Kernel

Polynomial kernels are of two types: homogeneous or inhomogeneous. Let $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$. The *homogeneous polynomial kernel* is defined as

$$K_q(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x})^T \phi(\mathbf{y}) = (\mathbf{x}^T \mathbf{y})^q \tag{5.8}$$

where q is the degree of the polynomial. This kernel corresponds to a feature space spanned by all products of exactly q attributes.

The most typical cases are the *linear* (with $q = 1$) and *quadratic* (with $q = 2$) kernels, given as

$$K_1(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T \mathbf{y}$$
$$K_2(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T \mathbf{y})^2$$

The *inhomogeneous polynomial kernel* is defined as

$$K_q(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x})^T \phi(\mathbf{y}) = (c + \mathbf{x}^T \mathbf{y})^q \tag{5.9}$$

where q is the degree of the polynomial, and $c \geq 0$ is some constant. When $c = 0$ we obtain the homogeneous kernel. When $c > 0$, this kernel corresponds to the feature space spanned by all products of at most q attributes. This can be seen from the binomial expansion

$$K_q(\mathbf{x}, \mathbf{y}) = (c + \mathbf{x}^T \mathbf{y})^q = \sum_{k=1}^{q} \binom{q}{k} c^{q-k} \left(\mathbf{x}^T \mathbf{y}\right)^k$$

For example, for the typical value of $c = 1$, the inhomogeneous kernel is a weighted sum of the homogeneous polynomial kernels for all powers up to q, that is,

$$(1 + \mathbf{x}^T\mathbf{y})^q = 1 + q\mathbf{x}^T\mathbf{y} + \binom{q}{2}\left(\mathbf{x}^T\mathbf{y}\right)^2 + \cdots + q\left(\mathbf{x}^T\mathbf{y}\right)^{q-1} + \left(\mathbf{x}^T\mathbf{y}\right)^q$$

Example 5.6. Consider the points $\mathbf{x}_1$ and $\mathbf{x}_2$ in Figure 5.1.

$$\mathbf{x}_1 = \begin{pmatrix} 5.9 \\ 3 \end{pmatrix} \qquad \mathbf{x}_2 = \begin{pmatrix} 6.9 \\ 3.1 \end{pmatrix}$$

The homogeneous quadratic kernel is given as

$$K(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{x}_1^T\mathbf{x}_2)^2 = 50.01^2 = 2501$$

The inhomogeneous quadratic kernel is given as

$$K(\mathbf{x}_1, \mathbf{x}_2) = (1 + \mathbf{x}_1^T\mathbf{x}_2)^2 = (1 + 50.01)^2 = 51.01^2 = 2602.02$$

For the polynomial kernel it is possible to construct a mapping ϕ from the input to the feature space. Let $n_0, n_1, \ldots, n_d$ denote non-negative integers, such that $\sum_{i=0}^{d} n_i = q$. Further, let $\mathbf{n} = (n_0, n_1, \ldots, n_d)$, and let $|\mathbf{n}| = \sum_{i=0}^{d} n_i = q$. Also, let $\binom{q}{\mathbf{n}}$ denote the multinomial coefficient

$$\binom{q}{\mathbf{n}} = \binom{q}{n_0, n_1, \ldots, n_d} = \frac{q!}{n_0!n_1!\ldots n_d!}$$

The multinomial expansion of the inhomogeneous kernel is then given as

$$\begin{aligned}
K_q(\mathbf{x}, \mathbf{y}) = (c + \mathbf{x}^T\mathbf{y})^q &= \left(c + \sum_{k=1}^{d} x_k y_k\right)^q = (c + x_1y_1 + \cdots + x_dy_d)^q \\
&= \sum_{|\mathbf{n}|=q} \binom{q}{\mathbf{n}} c^{n_0} (x_1y_1)^{n_1} (x_2y_2)^{n_2} \ldots (x_dy_d)^{n_d} \\
&= \sum_{|\mathbf{n}|=q} \binom{q}{\mathbf{n}} c^{n_0} \left(x_1^{n_1}x_2^{n_2}\ldots x_d^{n_d}\right)\left(y_1^{n_1}y_2^{n_2}\ldots y_d^{n_d}\right) \\
&= \sum_{|\mathbf{n}|=q} \left(\sqrt{a_{\mathbf{n}}}\prod_{k=1}^{d} x_k^{n_k}\right)\left(\sqrt{a_{\mathbf{n}}}\prod_{k=1}^{d} y_k^{n_k}\right) \\
&= \phi(\mathbf{x})^T\phi(\mathbf{y})
\end{aligned}$$

where $a_{\mathbf{n}} = \binom{q}{\mathbf{n}} c^{n_0}$, and the summation is over all $\mathbf{n} = (n_0, n_1, \ldots, n_d)$ such that $|\mathbf{n}| = n_0 + n_1 + \cdots + n_d = q$. Using the notation $\mathbf{x}^{\mathbf{n}} = \prod_{k=1}^{d} x_k^{n_k}$, the mapping $\phi : \mathbb{R}^d \to \mathbb{R}^m$ is given as the vector

$$\phi(\mathbf{x}) = (\ldots, a_{\mathbf{n}}\mathbf{x}^{\mathbf{n}}, \ldots)^T = \left(\ldots, \sqrt{\binom{q}{\mathbf{n}} c^{n_0}} \prod_{k=1}^{d} x_k^{n_k}, \ \ldots\right)^T$$

where the variable $\mathbf{n} = (n_0, \ldots, n_d)$ ranges over all the possible assignments, such that $|\mathbf{n}| = q$. It can be shown that the dimensionality of the feature space is given as

$$m = \binom{d+q}{q}$$

Example 5.7 (Quadratic Polynomial Kernel). Let $\mathbf{x}, \mathbf{y} \in \mathbb{R}^2$ and let $c = 1$. The inhomogeneous quadratic polynomial kernel is given as

$$K(\mathbf{x}, \mathbf{y}) = (1 + \mathbf{x}^T\mathbf{y})^2 = (1 + x_1y_1 + x_2y_2)^2$$

The set of all assignments $\mathbf{n} = (n_0, n_1, n_2)$, such that $|\mathbf{n}| = q = 2$, and the corresponding terms in the multinomial expansion are shown below.

Assignments $\mathbf{n} = (n_0, n_1, n_2)$	Coefficient $a_{\mathbf{n}} = \binom{q}{\mathbf{n}} c^{n_0}$	Variables $\mathbf{x}^{\mathbf{n}}\mathbf{y}^{\mathbf{n}} = \prod_{k=1}^{d}(x_i y_i)^{n_i}$
$(1, 1, 0)$	2	x_1y_1
$(1, 0, 1)$	2	x_2y_2
$(0, 1, 1)$	2	$x_1y_1x_2y_2$
$(2, 0, 0)$	1	1
$(0, 2, 0)$	1	$(x_1y_1)^2$
$(0, 0, 2)$	1	$(x_2y_2)^2$

Thus, the kernel can be written as

$$\begin{aligned} K(\mathbf{x}, \mathbf{y}) &= 1 + 2x_1y_1 + 2x_2y_2 + 2x_1y_1x_2y_2 + x_1^2y_1^2 + x_2^2y_2^2 \\ &= \left(1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2\right)\left(1, \sqrt{2}y_1, \sqrt{2}y_2, \sqrt{2}y_1y_2, y_1^2, y_2^2\right)^T \\ &= \phi(\mathbf{x})^T\phi(\mathbf{y}) \end{aligned}$$

When the input space is $\mathbb{R}^2$, the dimensionality of the feature space is given as

$$m = \binom{d+q}{q} = \binom{2+2}{2} = \binom{4}{2} = 6$$

In this case the inhomogeneous quadratic kernel with $c = 1$ corresponds to the mapping $\phi : \mathbb{R}^2 \to \mathbb{R}^6$, given as

$$\phi(\mathbf{x}) = \left(1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2,\ x_1^2,\ x_2^2\right)^T$$

For example, for $\mathbf{x}_1 = (5.9, 3)^T$ and $\mathbf{x}_2 = (6.9, 3.1)^T$, we have

$$\begin{aligned} \phi(\mathbf{x}_1) &= \left(1, \sqrt{2}\cdot 5.9, \sqrt{2}\cdot 3, \sqrt{2}\cdot 5.9\cdot 3,\ 5.9^2,\ 3^2\right)^T \\ &= (1, 8.34, 4.24, 25.03, 34.81, 9)^T \\ \phi(\mathbf{x}_2) &= \left(1, \sqrt{2}\cdot 6.9, \sqrt{2}\cdot 3.1, \sqrt{2}\cdot 6.9\cdot 3.1,\ 6.9^2,\ 3.1^2\right)^T \\ &= (1, 9.76, 4.38, 30.25, 47.61, 9.61)^T \end{aligned}$$

Thus, the inhomogeneous kernel value is

$$\phi(\mathbf{x}_1)^T\phi(\mathbf{x}_2) = 1 + 81.40 + 18.57 + 757.16 + 1657.30 + 86.49 = 2601.92$$

On the other hand, when the input space is $\mathbb{R}^2$, the homogeneous quadratic kernel corresponds to the mapping $\phi : \mathbb{R}^2 \to \mathbb{R}^3$, defined as

$$\phi(\mathbf{x}) = \left(\sqrt{2}x_1x_2,\ x_1^2,\ x_2^2\right)^T$$

because only the degree 2 terms are considered. For example, for $\mathbf{x}_1$ and $\mathbf{x}_2$, we have

$$\phi(\mathbf{x}_1) = \left(\sqrt{2}\cdot 5.9\cdot 3,\ 5.9^2,\ 3^2\right)^T = (25.03, 34.81, 9)^T$$

$$\phi(\mathbf{x}_2) = \left(\sqrt{2}\cdot 6.9\cdot 3.1,\ 6.9^2,\ 3.1^2\right)^T = (30.25, 47.61, 9.61)^T$$

and thus

$$K(\mathbf{x}_1, \mathbf{x}_2) = \phi(\mathbf{x}_1)^T\phi(\mathbf{x}_2) = 757.16 + 1657.3 + 86.49 = 2500.95$$

These values essentially match those shown in Example 5.6 up to four significant digits.

Gaussian Kernel

The Gaussian kernel, also called the Gaussian radial basis function (RBF) kernel, is defined as

$$K(\mathbf{x}, \mathbf{y}) = \exp\left\{-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{2\sigma^2}\right\} \tag{5.10}$$

where $\sigma > 0$ is the spread parameter that plays the same role as the standard deviation in a normal density function. Note that $K(\mathbf{x}, \mathbf{x}) = 1$, and further that the kernel value is inversely related to the distance between the two points $\mathbf{x}$ and $\mathbf{y}$.

Example 5.8. Consider again the points $\mathbf{x}_1$ and $\mathbf{x}_2$ in Figure 5.1:

$$\mathbf{x}_1 = \begin{pmatrix} 5.9 \\ 3 \end{pmatrix} \qquad \mathbf{x}_2 = \begin{pmatrix} 6.9 \\ 3.1 \end{pmatrix}$$

The squared distance between them is given as

$$\|\mathbf{x}_1 - \mathbf{x}_2\|^2 = \|(-1, -0.1)^T\|^2 = 1^2 + 0.1^2 = 1.01$$

With $\sigma = 1$, the Gaussian kernel is

$$K(\mathbf{x}_1, \mathbf{x}_2) = \exp\left\{-\frac{1.01^2}{2}\right\} = \exp\{-0.51\} = 0.6$$

It is interesting to note that a feature space for the Gaussian kernel has infinite dimensionality. To see this, note that the exponential function can be written as the

infinite expansion

$$\exp\{a\} = \sum_{n=0}^{\infty} \frac{a^n}{n!} = 1 + a + \frac{1}{2!}a^2 + \frac{1}{3!}a^3 + \cdots$$

Further, using $\gamma = \frac{1}{2\sigma^2}$, and noting that $\left\|\mathbf{x}-\mathbf{y}\right\|^2 = \|\mathbf{x}\|^2 + \left\|\mathbf{y}\right\|^2 - 2\mathbf{x}^T\mathbf{y}$, we can rewrite the Gaussian kernel as follows:

$$\begin{aligned} K(\mathbf{x},\mathbf{y}) &= \exp\left\{-\gamma \left\|\mathbf{x}-\mathbf{y}\right\|^2\right\} \\ &= \exp\left\{-\gamma \|\mathbf{x}\|^2\right\} \cdot \exp\left\{-\gamma \left\|\mathbf{y}\right\|^2\right\} \cdot \exp\left\{2\gamma \mathbf{x}^T\mathbf{y}\right\} \end{aligned}$$

In particular, the last term is given as the infinite expansion

$$\exp\left\{2\gamma \mathbf{x}^T\mathbf{y}\right\} = \sum_{q=0}^{\infty} \frac{(2\gamma)^q}{q!}\left(\mathbf{x}^T\mathbf{y}\right)^q = 1 + (2\gamma)\mathbf{x}^T\mathbf{y} + \frac{(2\gamma)^2}{2!}\left(\mathbf{x}^T\mathbf{y}\right)^2 + \cdots$$

Using the multinomial expansion of $(\mathbf{x}^T\mathbf{y})^q$, we can write the Gaussian kernel as

$$\begin{aligned} K(\mathbf{x},\mathbf{y}) &= \exp\left\{-\gamma \|\mathbf{x}\|^2\right\} \exp\left\{-\gamma \left\|\mathbf{y}\right\|^2\right\} \sum_{q=0}^{\infty} \frac{(2\gamma)^q}{q!} \left(\sum_{|\mathbf{n}|=q} \binom{q}{\mathbf{n}} \prod_{k=1}^{d} (x_k y_k)^{n_k} \right) \\ &= \sum_{q=0}^{\infty} \sum_{|\mathbf{n}|=q} \left(\sqrt{a_{q,\mathbf{n}}} \exp\left\{-\gamma \|\mathbf{x}\|^2\right\} \prod_{k=1}^{d} x_k^{n_k} \right) \left(\sqrt{a_{q,\mathbf{n}}} \exp\left\{-\gamma \left\|\mathbf{y}\right\|^2\right\} \prod_{k=1}^{d} y_k^{n_k} \right) \\ &= \phi(\mathbf{x})^T \phi(\mathbf{y}) \end{aligned}$$

where $a_{q,\mathbf{n}} = \frac{(2\gamma)^q}{q!}\binom{q}{\mathbf{n}}$, and $\mathbf{n} = (n_1, n_2, \ldots, n_d)$, with $|\mathbf{n}| = n_1 + n_2 + \cdots + n_d = q$. The mapping into feature space corresponds to the function $\phi : \mathbb{R}^d \to \mathbb{R}^\infty$

$$\phi(\mathbf{x}) = \left(\ldots, \sqrt{\frac{(2\gamma)^q}{q!}\binom{q}{\mathbf{n}}} \exp\left\{-\gamma \|\mathbf{x}\|^2\right\} \prod_{k=1}^{d} x_k^{n_k}, \ \ldots \right)^T$$

with the dimensions ranging over all degrees $q = 0, \ldots, \infty$, and with the variable $\mathbf{n} = (n_1, \ldots, n_d)$ ranging over all possible assignments such that $|\mathbf{n}| = q$ for each value of q. Because ϕ maps the input space into an infinite dimensional feature space, we obviously cannot explicitly transform $\mathbf{x}$ into $\phi(\mathbf{x})$, yet computing the Gaussian kernel $K(\mathbf{x},\mathbf{y})$ is straightforward.

5.3 BASIC KERNEL OPERATIONS IN FEATURE SPACE

Let us look at some of the basic data analysis tasks that can be performed solely via kernels, without instantiating $\phi(\mathbf{x})$.

Norm of a Point

We can compute the norm of a point $\phi(\mathbf{x})$ in feature space as follows:

$$\|\phi(\mathbf{x})\|^2 = \phi(\mathbf{x})^T\phi(\mathbf{x}) = K(\mathbf{x},\mathbf{x})$$

which implies that $\|\phi(\mathbf{x})\| = \sqrt{K(\mathbf{x},\mathbf{x})}$.

Distance between Points

The distance between two points $\phi(\mathbf{x}_i)$ and $\phi(\mathbf{x}_j)$ can be computed as

$$\begin{aligned}\left\|\phi(\mathbf{x}_i)-\phi(\mathbf{x}_j)\right\|^2 &= \|\phi(\mathbf{x}_i)\|^2 + \left\|\phi(\mathbf{x}_j)\right\|^2 - 2\phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j) \\ &= K(\mathbf{x}_i,\mathbf{x}_i) + K(\mathbf{x}_j,\mathbf{x}_j) - 2K(\mathbf{x}_i,\mathbf{x}_j)\end{aligned} \tag{5.11}$$

which implies that

$$\delta\big(\phi(\mathbf{x}_i),\phi(\mathbf{x}_j)\big) = \left\|\phi(\mathbf{x}_i)-\phi(\mathbf{x}_j)\right\| = \sqrt{K(\mathbf{x}_i,\mathbf{x}_i)+K(\mathbf{x}_j,\mathbf{x}_j)-2K(\mathbf{x}_i,\mathbf{x}_j)}$$

Rearranging Eq. (5.11), we can see that the kernel value can be considered as a measure of the similarity between two points, as

$$\frac{1}{2}\left(\|\phi(\mathbf{x}_i)\|^2 + \|\phi(\mathbf{x}_j)\|^2 - \|\phi(\mathbf{x}_i)-\phi(\mathbf{x}_j)\|^2\right) = K(\mathbf{x}_i,\mathbf{x}_j) = \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$$

Thus, the more the distance $\|\phi(\mathbf{x}_i)-\phi(\mathbf{x}_j)\|$ between the two points in feature space, the less the kernel value, that is, the less the similarity.

Example 5.9. Consider the two points $\mathbf{x}_1$ and $\mathbf{x}_2$ in Figure 5.1:

$$\mathbf{x}_1 = \begin{pmatrix}5.9\\3\end{pmatrix} \qquad \mathbf{x}_2 = \begin{pmatrix}6.9\\3.1\end{pmatrix}$$

Assuming the homogeneous quadratic kernel, the norm of $\phi(\mathbf{x}_1)$ can be computed as

$$\|\phi(\mathbf{x}_1)\|^2 = K(\mathbf{x}_1,\mathbf{x}_1) = (\mathbf{x}_1^T\mathbf{x}_1)^2 = 43.81^2 = 1919.32$$

which implies that the norm of the transformed point is $\|\phi(\mathbf{x}_1)\| = \sqrt{43.81^2} = 43.81$.

The distance between $\phi(\mathbf{x}_1)$ and $\phi(\mathbf{x}_2)$ in feature space is given as

$$\begin{aligned}\delta\big(\phi(\mathbf{x}_1),\phi(\mathbf{x}_2)\big) &= \sqrt{K(\mathbf{x}_1,\mathbf{x}_1)+K(\mathbf{x}_2,\mathbf{x}_2)-2K(\mathbf{x}_1,\mathbf{x}_2)} \\ &= \sqrt{1919.32+3274.13-2\cdot 2501} = \sqrt{191.45} = 13.84\end{aligned}$$

Mean in Feature Space

The mean of the points in feature space is given as

$$\boldsymbol{\mu}_\phi = \frac{1}{n}\sum_{i=1}^{n}\phi(\mathbf{x}_i)$$

Because we do not, in general, have access to $\phi(\mathbf{x}_i)$, we cannot explicitly compute the mean point in feature space.

Nevertheless, we can compute the squared norm of the mean as follows:

$$\begin{aligned}\|\boldsymbol{\mu}_\phi\|^2 &= \boldsymbol{\mu}_\phi^T \boldsymbol{\mu}_\phi \\ &= \left(\frac{1}{n}\sum_{i=1}^{n}\phi(\mathbf{x}_i)\right)^T \left(\frac{1}{n}\sum_{j=1}^{n}\phi(\mathbf{x}_j)\right) \\ &= \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}\phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j) \\ &= \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}K(\mathbf{x}_i,\mathbf{x}_j) \end{aligned} \tag{5.12}$$

The above derivation implies that the squared norm of the mean in feature space is simply the average of the values in the kernel matrix $\mathbf{K}$.

Example 5.10. Consider the five points from Example 5.3, also shown in Figure 5.1. Example 5.4 showed the norm of the mean for the linear kernel. Let us consider the Gaussian kernel with $\sigma = 1$. The Gaussian kernel matrix is given as

$$\mathbf{K} = \begin{pmatrix} 1.00 & 0.60 & 0.78 & 0.42 & 0.72 \\ 0.60 & 1.00 & 0.94 & 0.07 & 0.44 \\ 0.78 & 0.94 & 1.00 & 0.13 & 0.65 \\ 0.42 & 0.07 & 0.13 & 1.00 & 0.23 \\ 0.72 & 0.44 & 0.65 & 0.23 & 1.00 \end{pmatrix}$$

The squared norm of the mean in feature space is therefore

$$\left\|\boldsymbol{\mu}_\phi\right\|^2 = \frac{1}{25}\sum_{i=1}^{5}\sum_{j=1}^{5}K(\mathbf{x}_i,\mathbf{x}_j) = \frac{14.98}{25} = 0.599$$

which implies that $\left\|\boldsymbol{\mu}_\phi\right\| = \sqrt{0.599} = 0.774$.

Total Variance in Feature Space

Let us first derive a formula for the squared distance of a point $\phi(\mathbf{x}_i)$ to the mean $\boldsymbol{\mu}_\phi$ in feature space:

$$\begin{aligned}\|\phi(\mathbf{x}_i) - \boldsymbol{\mu}_\phi\|^2 &= \|\phi(\mathbf{x}_i)\|^2 - 2\phi(\mathbf{x}_i)^T\boldsymbol{\mu}_\phi + \|\boldsymbol{\mu}_\phi\|^2 \\ &= K(\mathbf{x}_i,\mathbf{x}_i) - \frac{2}{n}\sum_{j=1}^{n}K(\mathbf{x}_i,\mathbf{x}_j) + \frac{1}{n^2}\sum_{a=1}^{n}\sum_{b=1}^{n}K(\mathbf{x}_a,\mathbf{x}_b)\end{aligned}$$

The total variance [Eq. (1.4)] in feature space is obtained by taking the average squared deviation of points from the mean in feature space:

$$\sigma_\phi^2 = \frac{1}{n}\sum_{i=1}^{n}\|\phi(\mathbf{x}_i) - \boldsymbol{\mu}_\phi\|^2$$

$$= \frac{1}{n}\sum_{i=1}^{n}\left(K(\mathbf{x}_i,\mathbf{x}_i) - \frac{2}{n}\sum_{j=1}^{n}K(\mathbf{x}_i,\mathbf{x}_j) + \frac{1}{n^2}\sum_{a=1}^{n}\sum_{b=1}^{n}K(\mathbf{x}_a,\mathbf{x}_b)\right)$$

$$= \frac{1}{n}\sum_{i=1}^{n}K(\mathbf{x}_i,\mathbf{x}_i) - \frac{2}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}K(\mathbf{x}_i,\mathbf{x}_j) + \frac{n}{n^3}\sum_{a=1}^{n}\sum_{b=1}^{n}K(\mathbf{x}_a,\mathbf{x}_b)$$

$$= \frac{1}{n}\sum_{i=1}^{n}K(\mathbf{x}_i,\mathbf{x}_i) - \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}K(\mathbf{x}_i,\mathbf{x}_j) \tag{5.13}$$

In other words, the total variance in feature space is given as the difference between the average of the diagonal entries and the average of the entire kernel matrix $\mathbf{K}$. Also notice that by Eq. (5.12) the second term is simply $\|\boldsymbol{\mu}_\phi\|^2$.

Example 5.11. Continuing Example 5.10, the total variance in feature space for the five points, for the Gaussian kernel, is given as

$$\sigma_\phi^2 = \left(\frac{1}{n}\sum_{i=1}^{n}K(\mathbf{x}_i,\mathbf{x}_i)\right) - \|\boldsymbol{\mu}_\phi\|^2 = \frac{1}{5}\times 5 - 0.599 = 0.401$$

The distance between $\phi(\mathbf{x}_1)$ and the mean $\boldsymbol{\mu}_\phi$ in feature space is given as

$$\begin{aligned}\|\phi(\mathbf{x}_1)-\boldsymbol{\mu}_\phi\|^2 &= K(\mathbf{x}_1,\mathbf{x}_1) - \frac{2}{5}\sum_{j=1}^{5}K(\mathbf{x}_1,\mathbf{x}_j) + \|\boldsymbol{\mu}_\phi\|^2\\ &= 1 - \frac{2}{5}(1+0.6+0.78+0.42+0.72) + 0.599\\ &= 1 - 1.410 + 0.599 = 0.189\end{aligned}$$

Centering in Feature Space

We can center each point in feature space by subtracting the mean from it, as follows:

$$\hat{\phi}(\mathbf{x}_i) = \phi(\mathbf{x}_i) - \boldsymbol{\mu}_\phi$$

Because we do not have explicit representation of $\phi(\mathbf{x}_i)$ or $\boldsymbol{\mu}_\phi$, we cannot explicitly center the points. However, we can still compute the *centered kernel matrix*, that is, the kernel matrix over centered points.

The centered kernel matrix is given as

$$\hat{\mathbf{K}} = \left\{\hat{K}(\mathbf{x}_i,\mathbf{x}_j)\right\}_{i,j=1}^{n}$$

where each cell corresponds to the kernel between centered points, that is

$$\begin{aligned}\hat{K}(\mathbf{x}_i,\mathbf{x}_j) &= \hat{\phi}(\mathbf{x}_i)^T\hat{\phi}(\mathbf{x}_j)\\ &= (\phi(\mathbf{x}_i)-\boldsymbol{\mu}_\phi)^T(\phi(\mathbf{x}_j)-\boldsymbol{\mu}_\phi)\\ &= \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j) - \phi(\mathbf{x}_i)^T\boldsymbol{\mu}_\phi - \phi(\mathbf{x}_j)^T\boldsymbol{\mu}_\phi + \boldsymbol{\mu}_\phi^T\boldsymbol{\mu}_\phi\end{aligned}$$

$$= K(\mathbf{x}_i, \mathbf{x}_j) - \frac{1}{n}\sum_{k=1}^{n}\phi(\mathbf{x}_i)^T\phi(\mathbf{x}_k) - \frac{1}{n}\sum_{k=1}^{n}\phi(\mathbf{x}_j)^T\phi(\mathbf{x}_k) + \|\boldsymbol{\mu}_\phi\|^2$$

$$= K(\mathbf{x}_i, \mathbf{x}_j) - \frac{1}{n}\sum_{k=1}^{n}K(\mathbf{x}_i, \mathbf{x}_k) - \frac{1}{n}\sum_{k=1}^{n}K(\mathbf{x}_j, \mathbf{x}_k) + \frac{1}{n^2}\sum_{a=1}^{n}\sum_{b=1}^{n}K(\mathbf{x}_a, \mathbf{x}_b)$$

In other words, we can compute the centered kernel matrix using only the kernel function. Over all the pairs of points, the centered kernel matrix can be written compactly as follows:

$$\begin{aligned}\hat{\mathbf{K}} &= \mathbf{K} - \frac{1}{n}\mathbf{1}_{n\times n}\mathbf{K} - \frac{1}{n}\mathbf{K}\mathbf{1}_{n\times n} + \frac{1}{n^2}\mathbf{1}_{n\times n}\mathbf{K}\mathbf{1}_{n\times n} \\ &= \left(\mathbf{I} - \frac{1}{n}\mathbf{1}_{n\times n}\right)\mathbf{K}\left(\mathbf{I} - \frac{1}{n}\mathbf{1}_{n\times n}\right) \end{aligned} \tag{5.14}$$

where $\mathbf{1}_{n\times n}$ is the $n \times n$ singular matrix, all of whose entries equal 1.

Example 5.12. Consider the first five points from the 2-dimensional Iris dataset shown in Figure 5.1a:

$$\mathbf{x}_1 = \begin{pmatrix}5.9\\3\end{pmatrix} \quad \mathbf{x}_2 = \begin{pmatrix}6.9\\3.1\end{pmatrix} \quad \mathbf{x}_3 = \begin{pmatrix}6.6\\2.9\end{pmatrix} \quad \mathbf{x}_4 = \begin{pmatrix}4.6\\3.2\end{pmatrix} \quad \mathbf{x}_5 = \begin{pmatrix}6\\2.2\end{pmatrix}$$

Consider the linear kernel matrix shown in Figure 5.1b. We can center it by first computing

$$\mathbf{I} - \frac{1}{5}\mathbf{1}_{5\times 5} = \begin{pmatrix} 0.8 & -0.2 & -0.2 & -0.2 & -0.2 \\ -0.2 & 0.8 & -0.2 & -0.2 & -0.2 \\ -0.2 & -0.2 & 0.8 & -0.2 & -0.2 \\ -0.2 & -0.2 & -0.2 & 0.8 & -0.2 \\ -0.2 & -0.2 & -0.2 & -0.2 & 0.8 \end{pmatrix}$$

The centered kernel matrix [Eq. (5.14)] is given as

$$\hat{\mathbf{K}} = \left(\mathbf{I} - \frac{1}{5}\mathbf{1}_{5\times 5}\right) \cdot \begin{pmatrix} 43.81 & 50.01 & 47.64 & 36.74 & 42.00 \\ 50.01 & 57.22 & 54.53 & 41.66 & 48.22 \\ 47.64 & 54.53 & 51.97 & 39.64 & 45.98 \\ 36.74 & 41.66 & 39.64 & 31.40 & 34.64 \\ 42.00 & 48.22 & 45.98 & 34.64 & 40.84 \end{pmatrix} \cdot \left(\mathbf{I} - \frac{1}{5}\mathbf{1}_{5\times 5}\right)$$

$$= \begin{pmatrix} 0.02 & -0.06 & -0.06 & 0.18 & -0.08 \\ -0.06 & 0.86 & 0.54 & -1.19 & -0.15 \\ -0.06 & 0.54 & 0.36 & -0.83 & -0.01 \\ 0.18 & -1.19 & -0.83 & 2.06 & -0.22 \\ -0.08 & -0.15 & -0.01 & -0.22 & 0.46 \end{pmatrix}$$

To verify that $\hat{\mathbf{K}}$ is the same as the kernel matrix for the centered points, let us first center the points by subtracting the mean $\boldsymbol{\mu} = (6.0, 2.88)^T$. The centered points

in feature space are given as

$$\mathbf{z}_1 = \begin{pmatrix} -0.1 \\ 0.12 \end{pmatrix} \quad \mathbf{z}_2 = \begin{pmatrix} 0.9 \\ 0.22 \end{pmatrix} \quad \mathbf{z}_3 = \begin{pmatrix} 0.6 \\ 0.02 \end{pmatrix} \quad \mathbf{z}_4 = \begin{pmatrix} -1.4 \\ 0.32 \end{pmatrix} \quad \mathbf{z}_5 = \begin{pmatrix} 0.0 \\ -0.68 \end{pmatrix}$$

For example, the kernel between $\phi(\mathbf{z}_1)$ and $\phi(\mathbf{z}_2)$ is

$$\phi(\mathbf{z}_1)^T \phi(\mathbf{z}_2) = \mathbf{z}_1^T \mathbf{z}_2 = -0.09 + 0.03 = -0.06$$

which matches $\hat{\mathbf{K}}(\mathbf{x}_1, \mathbf{x}_2)$, as expected. The other entries can be verified in a similar manner. Thus, the kernel matrix obtained by centering the data and then computing the kernel is the same as that obtained via Eq. (5.14).

Normalizing in Feature Space

A common form of normalization is to ensure that points in feature space have unit length by replacing $\phi(\mathbf{x}_i)$ with the corresponding unit vector $\phi_n(\mathbf{x}_i) = \frac{\phi(\mathbf{x}_i)}{\|\phi(\mathbf{x}_i)\|}$. The dot product in feature space then corresponds to the cosine of the angle between the two mapped points, because

$$\phi_n(\mathbf{x}_i)^T \phi_n(\mathbf{x}_j) = \frac{\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)}{\|\phi(\mathbf{x}_i)\| \cdot \|\phi(\mathbf{x}_j)\|} = \cos\theta$$

If the mapped points are both centered and normalized, then a dot product corresponds to the correlation between the two points in feature space.

The normalized kernel matrix, $\mathbf{K}_n$, can be computed using only the kernel function K, as

$$\mathbf{K}_n(\mathbf{x}_i, \mathbf{x}_j) = \frac{\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)}{\|\phi(\mathbf{x}_i)\| \cdot \|\phi(\mathbf{x}_j)\|} = \frac{K(\mathbf{x}_i, \mathbf{x}_j)}{\sqrt{K(\mathbf{x}_i, \mathbf{x}_i) \cdot K(\mathbf{x}_j, \mathbf{x}_j)}}$$

$\mathbf{K}_n$ has all diagonal elements as 1.

Let $\mathbf{W}$ denote the diagonal matrix comprising the diagonal elements of $\mathbf{K}$:

$$\mathbf{W} = \text{diag}(\mathbf{K}) = \begin{pmatrix} K(\mathbf{x}_1, \mathbf{x}_1) & 0 & \cdots & 0 \\ 0 & K(\mathbf{x}_2, \mathbf{x}_2) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & K(\mathbf{x}_n, \mathbf{x}_n) \end{pmatrix}$$

The normalized kernel matrix can then be expressed compactly as

$$\mathbf{K}_n = \mathbf{W}^{-1/2} \cdot \mathbf{K} \cdot \mathbf{W}^{-1/2}$$

where $\mathbf{W}^{-1/2}$ is the diagonal matrix, defined as $\mathbf{W}^{-1/2}(\mathbf{x}_i, \mathbf{x}_i) = \frac{1}{\sqrt{K(\mathbf{x}_i, \mathbf{x}_i)}}$, with all other elements being zero.

Example 5.13. Consider the five points and the linear kernel matrix shown in Figure 5.1. We have

$$\mathbf{W} = \begin{pmatrix} 43.81 & 0 & 0 & 0 & 0 \\ 0 & 57.22 & 0 & 0 & 0 \\ 0 & 0 & 51.97 & 0 & 0 \\ 0 & 0 & 0 & 31.40 & 0 \\ 0 & 0 & 0 & 0 & 40.84 \end{pmatrix}$$

The normalized kernel is given as

$$\mathbf{K}_n = \mathbf{W}^{-1/2} \cdot \mathbf{K} \cdot \mathbf{W}^{-1/2} = \begin{pmatrix} 1.0000 & 0.9988 & 0.9984 & 0.9906 & 0.9929 \\ 0.9988 & 1.0000 & 0.9999 & 0.9828 & 0.9975 \\ 0.9984 & 0.9999 & 1.0000 & 0.9812 & 0.9980 \\ 0.9906 & 0.9828 & 0.9812 & 1.0000 & 0.9673 \\ 0.9929 & 0.9975 & 0.9980 & 0.9673 & 1.0000 \end{pmatrix}$$

The same kernel is obtained if we first normalize the feature vectors to have unit length and then take the dot products. For example, with the linear kernel, the normalized point $\phi_n(\mathbf{x}_1)$ is given as

$$\phi_n(\mathbf{x}_1) = \frac{\phi(\mathbf{x}_1)}{\|\phi(\mathbf{x}_1)\|} = \frac{\mathbf{x}_1}{\|\mathbf{x}_1\|} = \frac{1}{\sqrt{43.81}} \begin{pmatrix} 5.9 \\ 3 \end{pmatrix} = \begin{pmatrix} 0.8914 \\ 0.4532 \end{pmatrix}$$

Likewise, we have $\phi_n(\mathbf{x}_2) = \frac{1}{\sqrt{57.22}} \begin{pmatrix} 6.9 \\ 3.1 \end{pmatrix} = \begin{pmatrix} 0.9122 \\ 0.4098 \end{pmatrix}$. Their dot product is

$$\phi_n(\mathbf{x}_1)^T \phi_n(\mathbf{x}_2) = 0.8914 \cdot 0.9122 + 0.4532 \cdot 0.4098 = 0.9988$$

which matches $\mathbf{K}_n(\mathbf{x}_1, \mathbf{x}_2)$.

If we start with the centered kernel matrix $\hat{\mathbf{K}}$ from Example 5.12, and then normalize it, we obtain the normalized and centered kernel matrix $\hat{\mathbf{K}}_n$:

$$\hat{\mathbf{K}}_n = \begin{pmatrix} 1.00 & -0.44 & -0.61 & 0.80 & -0.77 \\ -0.44 & 1.00 & 0.98 & -0.89 & -0.24 \\ -0.61 & 0.98 & 1.00 & -0.97 & -0.03 \\ 0.80 & -0.89 & -0.97 & 1.00 & -0.22 \\ -0.77 & -0.24 & -0.03 & -0.22 & 1.00 \end{pmatrix}$$

As noted earlier, the kernel value $\hat{\mathbf{K}}_n(\mathbf{x}_i, \mathbf{x}_j)$ denotes the correlation between $\mathbf{x}_i$ and $\mathbf{x}_j$ in feature space, that is, it is cosine of the angle between the centered points $\phi(\mathbf{x}_i)$ and $\phi(\mathbf{x}_j)$.

5.4 KERNELS FOR COMPLEX OBJECTS

We conclude this chapter with some examples of kernels defined for complex data such as strings and graphs. The use of kernels for dimensionality reduction is described in

Section 7.3, for clustering in Section 13.2 and Chapter 16, for discriminant analysis in Section 20.2, and for classification in Sections 21.4 and 21.5.

5.4.1 Spectrum Kernel for Strings

Consider text or sequence data defined over an alphabet Σ. The l-spectrum feature map is the mapping $\phi : \Sigma^* \to \mathbb{R}^{|\Sigma|^l}$ from the set of substrings over Σ to the $|\Sigma|^l$-dimensional space representing the number of occurrences of all possible substrings of length l, defined as

$$\phi(\mathbf{x}) = \Big(\cdots, \#(\alpha), \cdots\Big)^T_{\alpha \in \Sigma^l}$$

where $\#(\alpha)$ is the number of occurrences of the l-length string α in $\mathbf{x}$.

The (full) spectrum map is an extension of the l-spectrum map, obtained by considering all lengths from $l = 0$ to $l = \infty$, leading to an infinite dimensional feature map $\phi : \Sigma^* \to \mathbb{R}^\infty$:

$$\phi(\mathbf{x}) = \Big(\cdots, \#(\alpha), \cdots\Big)^T_{\alpha \in \Sigma^*}$$

where $\#(\alpha)$ is the number of occurrences of the string α in $\mathbf{x}$.

The (l-)spectrum kernel between two strings $\mathbf{x}_i, \mathbf{x}_j$ is simply the dot product between their (l-)spectrum maps:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

A naive computation of the l-spectrum kernel takes $O(|\Sigma|^l)$ time. However, for a given string $\mathbf{x}$ of length n, the vast majority of the l-length strings have an occurrence count of zero, which can be ignored. The l-spectrum map can be effectively computed in $O(n)$ time for a string of length n (assuming $n \gg l$) because there can be at most $n - l + 1$ substrings of length l, and the l-spectrum kernel can thus be computed in $O(n + m)$ time for any two strings of length n and m, respectively.

The feature map for the (full) spectrum kernel is infinite dimensional, but once again, for a given string $\mathbf{x}$ of length n, the vast majority of the strings will have an occurrence count of zero. A straightforward implementation of the spectrum map for a string $\mathbf{x}$ of length n can be computed in $O(n^2)$ time because $\mathbf{x}$ can have at most $\sum_{l=1}^{n} n - l + 1 = n(n+1)/2$ distinct nonempty substrings. The spectrum kernel can then be computed in $O(n^2 + m^2)$ time for any two strings of length n and m, respectively. However, a much more efficient computation is enabled via suffix trees (see Chapter 10), with a total time of $O(n + m)$.

Example 5.14. Consider sequences over the DNA alphabet $\Sigma = \{A, C, G, T\}$. Let $\mathbf{x}_1 = ACAGCAGTA$, and let $\mathbf{x}_2 = AGCAAGCGAG$. For $l = 3$, the feature space has dimensionality $|\Sigma|^l = 4^3 = 64$. Nevertheless, we do not have to map the input points into the full feature space; we can compute the reduced 3-spectrum mapping by counting the number of occurrences for only the length 3 substrings that occur in each input sequence, as follows:

$$\phi(\mathbf{x}_1) = (ACA : 1, AGC : 1, AGT : 1, CAG : 2, GCA : 1, GTA : 1)$$
$$\phi(\mathbf{x}_2) = (AAG : 1, AGC : 2, CAA : 1, CGA : 1, GAG : 1, GCA : 1, GCG : 1)$$

where the notation $\alpha : \#(\alpha)$ denotes that substring α has $\#(\alpha)$ occurrences in $\mathbf{x}_i$. We can then compute the dot product by considering only the common substrings, as follows:

$$K(\mathbf{x}_1, \mathbf{x}_2) = 1 \times 2 + 1 \times 1 = 2 + 1 = 3$$

The first term in the dot product is due to the substring AGC, and the second is due to GCA, which are the only common length 3 substrings between $\mathbf{x}_1$ and $\mathbf{x}_2$.

The full spectrum can be computed by considering the occurrences of all common substrings over all possible lengths. For $\mathbf{x}_1$ and $\mathbf{x}_2$, the common substrings and their occurrence counts are given as

α	A	C	G	AG	CA	AGC	GCA	$AGCA$
$\#(\alpha)$ in $\mathbf{x}_1$	4	2	2	2	2	1	1	1
$\#(\alpha)$ in $\mathbf{x}_2$	4	2	4	3	1	2	1	1

Thus, the full spectrum kernel value is given as

$$K(\mathbf{x}_1, \mathbf{x}_2) = 16 + 4 + 8 + 6 + 2 + 2 + 1 + 1 = 40$$

5.4.2 Diffusion Kernels on Graph Nodes

Let $\mathbf{S}$ be some symmetric similarity matrix between nodes of a graph $G = (V, E)$. For instance, $\mathbf{S}$ can be the (weighted) adjacency matrix $\mathbf{A}$ [Eq. (4.1)] or the Laplacian matrix $\mathbf{L} = \mathbf{A} - \mathbf{\Delta}$ (or its negation), where $\mathbf{\Delta}$ is the degree matrix for an undirected graph G, defined as $\mathbf{\Delta}(i, i) = d_i$ and $\mathbf{\Delta}(i, j) = 0$ for all $i \neq j$, and d_i is the degree of node i.

Consider the similarity between any two nodes obtained by summing the product of the similarities over paths of length 2:

$$S^{(2)}(\mathbf{x}_i, \mathbf{x}_j) = \sum_{a=1}^{n} S(\mathbf{x}_i, \mathbf{x}_a) S(\mathbf{x}_a, \mathbf{x}_j) = \mathbf{S}_i^T \mathbf{S}_j$$

where

$$\mathbf{S}_i = \Big(S(\mathbf{x}_i, \mathbf{x}_1), S(\mathbf{x}_i, \mathbf{x}_2), \ldots, S(\mathbf{x}_i, \mathbf{x}_n)\Big)^T$$

denotes the (column) vector representing the i-th row of $\mathbf{S}$ (and because $\mathbf{S}$ is symmetric, it also denotes the ith column of $\mathbf{S}$). Over all pairs of nodes the similarity matrix over paths of length 2, denoted $\mathbf{S}^{(2)}$, is thus given as the square of the base similarity matrix $\mathbf{S}$:

$$\mathbf{S}^{(2)} = \mathbf{S} \times \mathbf{S} = \mathbf{S}^2$$

In general, if we sum up the product of the base similarities over all l-length paths between two nodes, we obtain the l-length similarity matrix $\mathbf{S}^{(l)}$, which is simply the lth power of $\mathbf{S}$, that is,

$$\mathbf{S}^{(l)} = \mathbf{S}^l$$

Power Kernels

Even path lengths lead to positive semidefinite kernels, but odd path lengths are not guaranteed to do so, unless the base matrix $\mathbf{S}$ is itself a positive semidefinite matrix. In particular, $\mathbf{K} = \mathbf{S}^2$ is a valid kernel. To see this, assume that the ith row of $\mathbf{S}$ denotes the feature map for $\mathbf{x}_i$, that is, $\phi(\mathbf{x}_i) = \mathbf{S}_i$. The kernel value between any two points is then a dot product in feature space:

$$K(\mathbf{x}_i, \mathbf{x}_j) = S^{(2)}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{S}_i^T \mathbf{S}_j = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

For a general path length l, let $\mathbf{K} = \mathbf{S}^l$. Consider the eigen-decomposition of $\mathbf{S}$:

$$\mathbf{S} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T = \sum_{i=1}^{n} \mathbf{u}_i \lambda_i \mathbf{u}_i^T$$

where $\mathbf{U}$ is the orthogonal matrix of eigenvectors and $\boldsymbol{\Lambda}$ is the diagonal matrix of eigenvalues of $\mathbf{S}$:

$$\mathbf{U} = \begin{pmatrix} | & | & & | \\ \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_n \\ | & | & & | \end{pmatrix} \qquad \boldsymbol{\Lambda} = \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \cdots & \lambda_n \end{pmatrix}$$

The eigen-decomposition of $\mathbf{K}$ can be obtained as follows:

$$\mathbf{K} = \mathbf{S}^l = \left(\mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T\right)^l = \mathbf{U}\left(\boldsymbol{\Lambda}^l\right)\mathbf{U}^T$$

where we used the fact that eigenvectors of $\mathbf{S}$ and $\mathbf{S}^l$ are identical, and further that eigenvalues of $\mathbf{S}^l$ are given as $(\lambda_i)^l$ (for all $i = 1, \ldots, n$), where λ_i is an eigenvalue of $\mathbf{S}$. For $\mathbf{K} = \mathbf{S}^l$ to be a positive semidefinite matrix, all its eigenvalues must be non-negative, which is guaranteed for all even path lengths. Because $(\lambda_i)^l$ will be negative if l is odd and λ_i is negative, odd path lengths lead to a positive semidefinite kernel only if $\mathbf{S}$ is positive semidefinite.

Exponential Diffusion Kernel

Instead of fixing the path length *a priori*, we can obtain a new kernel between nodes of a graph by considering paths of all possible lengths, but by damping the contribution of longer paths, which leads to the *exponential diffusion kernel*, defined as

$$\begin{aligned} \mathbf{K} &= \sum_{l=0}^{\infty} \frac{1}{l!} \beta^l \mathbf{S}^l \\ &= \mathbf{I} + \beta\mathbf{S} + \frac{1}{2!}\beta^2\mathbf{S}^2 + \frac{1}{3!}\beta^3\mathbf{S}^3 + \cdots \\ &= \exp\{\beta\mathbf{S}\} \end{aligned} \tag{5.15}$$

where β is a damping factor, and $\exp\{\beta\mathbf{S}\}$ is the matrix exponential. The series on the right hand side above converges for all $\beta \geq 0$.

Substituting $\mathbf{S} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T = \sum_{i=1}^{n} \lambda_i \mathbf{u}_i \mathbf{u}_i^T$ in Eq. (5.15), and utilizing the fact that $\mathbf{U}\mathbf{U}^T = \sum_{i=1}^{n} \mathbf{u}_i \mathbf{u}_i^T = \mathbf{I}$, we have

$$\begin{aligned}
\mathbf{K} &= \mathbf{I} + \beta\mathbf{S} + \frac{1}{2!}\beta^2\mathbf{S}^2 + \cdots \\
&= \left(\sum_{i=1}^{n} \mathbf{u}_i \mathbf{u}_i^T\right) + \left(\sum_{i=1}^{n} \mathbf{u}_i \beta\lambda_i \mathbf{u}_i^T\right) + \left(\sum_{i=1}^{n} \mathbf{u}_i \frac{1}{2!}\beta^2\lambda_i^2 \mathbf{u}_i^T\right) + \cdots \\
&= \sum_{i=1}^{n} \mathbf{u}_i \left(1 + \beta\lambda_i + \frac{1}{2!}\beta^2\lambda_i^2 + \cdots\right) \mathbf{u}_i^T \\
&= \sum_{i=1}^{n} \mathbf{u}_i \exp\{\beta\lambda_i\}\, \mathbf{u}_i^T \\
&= \mathbf{U} \begin{pmatrix} \exp\{\beta\lambda_1\} & 0 & \cdots & 0 \\ 0 & \exp\{\beta\lambda_2\} & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \cdots & \exp\{\beta\lambda_n\} \end{pmatrix} \mathbf{U}^T
\end{aligned} \tag{5.16}$$

Thus, the eigenvectors of $\mathbf{K}$ are the same as those for $\mathbf{S}$, whereas its eigenvalues are given as $\exp\{\beta\lambda_i\}$, where λ_i is an eigenvalue of $\mathbf{S}$. Further, $\mathbf{K}$ is symmetric because $\mathbf{S}$ is symmetric, and its eigenvalues are real and non-negative because the exponential of a real number is non-negative. $\mathbf{K}$ is thus a positive semidefinite kernel matrix. The complexity of computing the diffusion kernel is $O(n^3)$ corresponding to the complexity of computing the eigen-decomposition.

Von Neumann Diffusion Kernel

A related kernel based on powers of $\mathbf{S}$ is the *von Neumann diffusion kernel*, defined as

$$\mathbf{K} = \sum_{l=0}^{\infty} \beta^l \mathbf{S}^l \tag{5.17}$$

where $\beta \geq 0$. Expanding Eq. (5.17), we have

$$\begin{aligned}
\mathbf{K} &= \mathbf{I} + \beta\mathbf{S} + \beta^2\mathbf{S}^2 + \beta^3\mathbf{S}^3 + \cdots \\
&= \mathbf{I} + \beta\mathbf{S}(\mathbf{I} + \beta\mathbf{S} + \beta^2\mathbf{S}^2 + \cdots) \\
&= \mathbf{I} + \beta\mathbf{S}\mathbf{K}
\end{aligned}$$

Rearranging the terms in the preceding equation, we obtain a closed form expression for the von Neumann kernel:

$$\begin{aligned}
\mathbf{K} - \beta\mathbf{S}\mathbf{K} &= \mathbf{I} \\
(\mathbf{I} - \beta\mathbf{S})\mathbf{K} &= \mathbf{I} \\
\mathbf{K} &= (\mathbf{I} - \beta\mathbf{S})^{-1}
\end{aligned} \tag{5.18}$$

Plugging in the eigen-decomposition $\mathbf{S} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T$, and rewriting $\mathbf{I} = \mathbf{U}\mathbf{U}^T$, we have

$$\begin{aligned}\mathbf{K} &= \left(\mathbf{U}\mathbf{U}^T - \mathbf{U}(\beta\boldsymbol{\Lambda})\mathbf{U}^T\right)^{-1} \\ &= \left(\mathbf{U}(\mathbf{I} - \beta\boldsymbol{\Lambda})\mathbf{U}^T\right)^{-1} \\ &= \mathbf{U}(\mathbf{I} - \beta\boldsymbol{\Lambda})^{-1}\mathbf{U}^T\end{aligned}$$

where $(\mathbf{I} - \beta\boldsymbol{\Lambda})^{-1}$ is the diagonal matrix whose ith diagonal entry is $(1 - \beta\lambda_i)^{-1}$. The eigenvectors of $\mathbf{K}$ and $\mathbf{S}$ are identical, but the eigenvalues of $\mathbf{K}$ are given as $1/(1 - \beta\lambda_i)$. For $\mathbf{K}$ to be a positive semidefinite kernel, all its eigenvalues should be non-negative, which in turn implies that

$$\begin{aligned}(1 - \beta\lambda_i)^{-1} &\geq 0 \\ 1 - \beta\lambda_i &\geq 0 \\ \beta &\leq 1/\lambda_i\end{aligned}$$

Further, the inverse matrix $(\mathbf{I} - \beta\boldsymbol{\Lambda})^{-1}$ exists only if

$$\det(\mathbf{I} - \beta\boldsymbol{\Lambda}) = \prod_{i=1}^{n}(1 - \beta\lambda_i) \neq 0$$

which implies that $\beta \neq 1/\lambda_i$ for all i. Thus, for $\mathbf{K}$ to be a valid kernel, we require that $\beta < 1/\lambda_i$ for all $i = 1, \ldots, n$. The von Neumann kernel is therefore guaranteed to be positive semidefinite if $|\beta| < 1/\rho(\mathbf{S})$, where $\rho(\mathbf{S}) = \max_i\{|\lambda_i|\}$ is called the *spectral radius* of $\mathbf{S}$, defined as the largest eigenvalue of $\mathbf{S}$ in absolute value.

Example 5.15. Consider the graph in Figure 5.2. Its adjacency and degree matrices are given as

$$\mathbf{A} = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix} \qquad \boldsymbol{\Delta} = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{pmatrix}$$

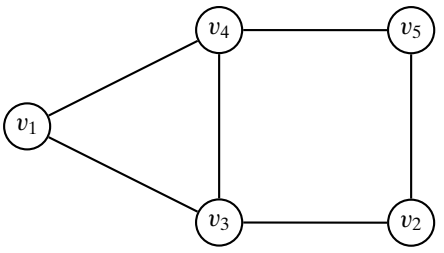


Figure 5.2. Graph diffusion kernel.

The negated Laplacian matrix for the graph is therefore

$$\mathbf{S} = -\mathbf{L} = \mathbf{A} - \mathbf{D} = \begin{pmatrix} -2 & 0 & 1 & 1 & 0 \\ 0 & -2 & 1 & 0 & 1 \\ 1 & 1 & -3 & 1 & 0 \\ 1 & 0 & 1 & -3 & 1 \\ 0 & 1 & 0 & 1 & -2 \end{pmatrix}$$

The eigenvalues of $\mathbf{S}$ are as follows:

$$\lambda_1 = 0 \qquad \lambda_2 = -1.38 \qquad \lambda_3 = -2.38 \qquad \lambda_4 = -3.62 \qquad \lambda_5 = -4.62$$

and the eigenvectors of $\mathbf{S}$ are

$$\mathbf{U} = \begin{pmatrix} \mathbf{u}_1 & \mathbf{u}_2 & \mathbf{u}_3 & \mathbf{u}_4 & \mathbf{u}_5 \\ \hline 0.45 & -0.63 & 0.00 & 0.63 & 0.00 \\ 0.45 & 0.51 & -0.60 & 0.20 & -0.37 \\ 0.45 & -0.20 & -0.37 & -0.51 & 0.60 \\ 0.45 & -0.20 & 0.37 & -0.51 & -0.60 \\ 0.45 & 0.51 & 0.60 & 0.20 & 0.37 \end{pmatrix}$$

Assuming $\beta = 0.2$, the exponential diffusion kernel matrix is given as

$$\mathbf{K} = \exp\{0.2\mathbf{S}\} = \mathbf{U} \begin{pmatrix} \exp\{0.2\lambda_1\} & 0 & \cdots & 0 \\ 0 & \exp\{0.2\lambda_2\} & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \cdots & \exp\{0.2\lambda_n\} \end{pmatrix} \mathbf{U}^T$$

$$= \begin{pmatrix} 0.70 & 0.01 & 0.14 & 0.14 & 0.01 \\ 0.01 & 0.70 & 0.13 & 0.03 & 0.14 \\ 0.14 & 0.13 & 0.59 & 0.13 & 0.03 \\ 0.14 & 0.03 & 0.13 & 0.59 & 0.13 \\ 0.01 & 0.14 & 0.03 & 0.13 & 0.70 \end{pmatrix}$$

For the von Neumann diffusion kernel, we have

$$(\mathbf{I} - 0.2\mathbf{\Lambda})^{-1} = \begin{pmatrix} 1 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0 & 0.78 & 0.00 & 0.00 & 0.00 \\ 0 & 0.00 & 0.68 & 0.00 & 0.00 \\ 0 & 0.00 & 0.00 & 0.58 & 0.00 \\ 0 & 0.00 & 0.00 & 0.00 & 0.52 \end{pmatrix}$$

For instance, because $\lambda_2 = -1.38$, we have $1 - \beta\lambda_2 = 1 + 0.2 \times 1.38 = 1.28$, and therefore the second diagonal entry is $(1-\beta\lambda_2)^{-1} = 1/1.28 = 0.78$. The von Neumann kernel is given as

$$\mathbf{K} = \mathbf{U}(\mathbf{I} - 0.2\mathbf{\Lambda})^{-1}\mathbf{U}^T = \begin{pmatrix} 0.75 & 0.02 & 0.11 & 0.11 & 0.02 \\ 0.02 & 0.74 & 0.10 & 0.03 & 0.11 \\ 0.11 & 0.10 & 0.66 & 0.10 & 0.03 \\ 0.11 & 0.03 & 0.10 & 0.66 & 0.10 \\ 0.02 & 0.11 & 0.03 & 0.10 & 0.74 \end{pmatrix}$$

5.5 FURTHER READING

Kernel methods have been extensively studied in machine learning and data mining. For an in-depth introduction and more advanced topics see Schölkopf and Smola (2002) and Shawe-Taylor and Cristianini (2004). For applications of kernel methods in bioinformatics see Schölkopf, Tsuda, and Vert (2004).

Schölkopf, B. and Smola, A. J. (2002). *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA: MIT Press.

Schölkopf, B., Tsuda, K., and Vert, J.-P. (2004). *Kernel Methods in Computational Biology*. Cambridge, MA: MIT Press.

Shawe-Taylor, J. and Cristianini, N. (2004). *Kernel Methods for Pattern Analysis*. New York: Cambridge University Press.

5.6 EXERCISES

Q1. Prove that the dimensionality of the feature space for the inhomogeneous polynomial kernel of degree q is

$$m = \binom{d+q}{q}$$

Q2. Consider the data shown in Table 5.1. Assume the following kernel function: $K(\mathbf{x}_i, \mathbf{x}_j) = \|\mathbf{x}_i - \mathbf{x}_j\|^2$. Compute the kernel matrix $\mathbf{K}$.

Table 5.1. Dataset for Q2

i	$\mathbf{x}_i$
$\mathbf{x}_1$	(4, 2.9)
$\mathbf{x}_2$	(2.5, 1)
$\mathbf{x}_3$	(3.5, 4)
$\mathbf{x}_4$	(2, 2.1)

Q3. Show that eigenvectors of $\mathbf{S}$ and $\mathbf{S}^l$ are identical, and further that eigenvalues of $\mathbf{S}^l$ are given as $(\lambda_i)^l$ (for all $i = 1, \ldots, n$), where λ_i is an eigenvalue of $\mathbf{S}$, and $\mathbf{S}$ is some $n \times n$ symmetric similarity matrix.

Q4. The von Neumann diffusion kernel is a valid positive semidefinite kernel if $|\beta| < \frac{1}{\rho(\mathbf{S})}$, where $\rho(\mathbf{S})$ is the spectral radius of $\mathbf{S}$. Can you derive better bounds for cases when $\beta > 0$ and when $\beta < 0$?

Q5. Given the three points $\mathbf{x}_1 = (2.5, 1)^T$, $\mathbf{x}_2 = (3.5, 4)^T$, and $\mathbf{x}_3 = (2, 2.1)^T$.

(a) Compute the kernel matrix for the Gaussian kernel assuming that $\sigma^2 = 5$.

(b) Compute the distance of the point $\phi(\mathbf{x}_1)$ from the mean in feature space.

(c) Compute the dominant eigenvector and eigenvalue for the kernel matrix from (a).

CHAPTER 6 High-dimensional Data

In data mining typically the data is very high dimensional, as the number of attributes can easily be in the hundreds or thousands. Understanding the nature of high-dimensional space, or *hyperspace*, is very important, especially because hyperspace does not behave like the more familiar geometry in two or three dimensions.

6.1 HIGH-DIMENSIONAL OBJECTS

Consider the $n \times d$ data matrix

$$\mathbf{D} = \begin{array}{c|cccc} & X_1 & X_2 & \cdots & X_d \\ \hline \mathbf{x}_1 & x_{11} & x_{12} & \cdots & x_{1d} \\ \mathbf{x}_2 & x_{21} & x_{22} & \cdots & x_{2d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{x}_n & x_{n1} & x_{n2} & \cdots & x_{nd} \end{array}$$

where each point $\mathbf{x}_i \in \mathbb{R}^d$ and each attribute $X_j \in \mathbb{R}^n$.

Hypercube

Let the minimum and maximum values for each attribute X_j be given as

$$\min(X_j) = \min_i \{x_{ij}\} \qquad \max(X_j) = \max_i \{x_{ij}\}$$

The data hyperspace can be considered as a d-dimensional *hyper-rectangle*, defined as

$$\begin{aligned} R_d &= \prod_{j=1}^{d} \Big[\min(X_j), \max(X_j)\Big] \\ &= \Big\{\mathbf{x} = (x_1, x_2, \ldots, x_d)^T \;\Big|\; x_j \in [\min(X_j), \max(X_j)], \text{for } j = 1, \ldots, d\Big\} \end{aligned}$$

Assume the data is centered to have mean $\boldsymbol{\mu} = \mathbf{0}$. Let m denote the largest absolute value in $\mathbf{D}$, given as

$$m = \max_{j=1}^{d} \max_{i=1}^{n} \{|x_{ij}|\}$$

The data hyperspace can be represented as a *hypercube*, centered at $\mathbf{0}$, with all sides of length $l = 2m$, given as

$$H_d(l) = \left\{ \mathbf{x} = (x_1, x_2, \ldots, x_d)^T \mid \forall i,\ x_i \in [-l/2, l/2] \right\}$$

The hypercube in one dimension, $H_1(l)$, represents an interval, which in two dimensions, $H_2(l)$, represents a square, and which in three dimensions, $H_3(l)$, represents a cube, and so on. The *unit hypercube* has all sides of length $l = 1$, and is denoted as $H_d(1)$.

Hypersphere

Assume that the data has been centered, so that $\boldsymbol{\mu} = \mathbf{0}$. Let r denote the largest magnitude among all points:

$$r = \max_{i} \{\|\mathbf{x}_i\|\}$$

The data hyperspace can also be represented as a d-dimensional *hyperball* centered at $\mathbf{0}$ with radius r, defined as

$$B_d(r) = \{\mathbf{x} \mid \|\mathbf{x}\| \leq r\}$$

$$\text{or } B_d(r) = \left\{ \mathbf{x} = (x_1, x_2, \ldots, x_d) \mid \sum_{j=1}^{d} x_j^2 \leq r^2 \right\}$$

The surface of the hyperball is called a *hypersphere*, and it consists of all the points exactly at distance r from the center of the hyperball, defined as

$$S_d(r) = \{\mathbf{x} \mid \|\mathbf{x}\| = r\}$$

$$\text{or } S_d(r) = \left\{ \mathbf{x} = (x_1, x_2, \ldots, x_d) \mid \sum_{j=1}^{d} (x_j)^2 = r^2 \right\}$$

Because the hyperball consists of all the surface and interior points, it is also called a *closed hypersphere*.

Example 6.1. Consider the 2-dimensional, centered, Iris dataset, plotted in Figure 6.1. The largest absolute value along any dimension is $m = 2.06$, and the point with the largest magnitude is $(2.06, 0.75)$, with $r = 2.19$. In two dimensions, the hypercube representing the data space is a square with sides of length $l = 2m = 4.12$. The hypersphere marking the extent of the space is a circle (shown dashed) with radius $r = 2.19$.

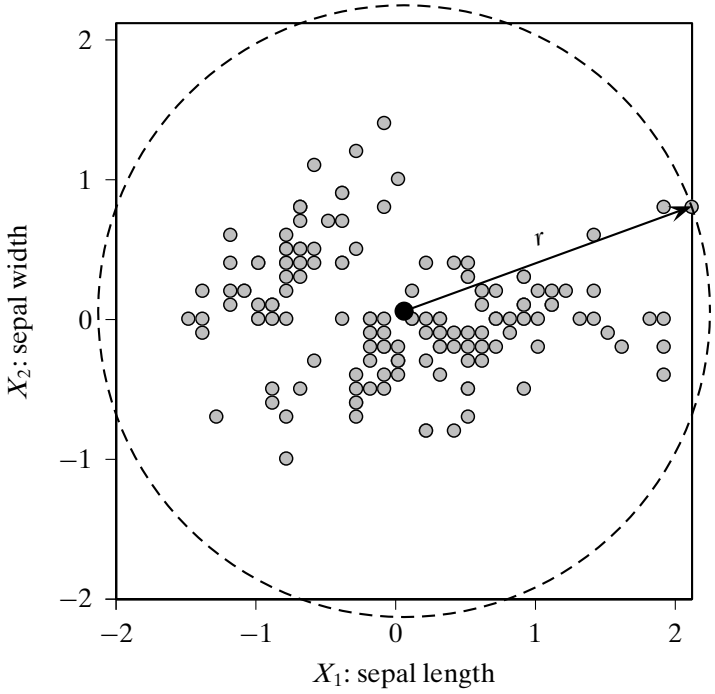


Figure 6.1. Iris data hyperspace: hypercube (solid; with $l = 4.12$) and hypersphere (dashed; with $r = 2.19$).

6.2 HIGH-DIMENSIONAL VOLUMES

Hypercube

The volume of a hypercube with edge length l is given as

$$\text{vol}(H_d(l)) = l^d$$

Hypersphere

The volume of a hyperball and its corresponding hypersphere is identical because the volume measures the total content of the object, including all internal space. Consider the well known equations for the volume of a hypersphere in lower dimensions

$$\text{vol}(S_1(r)) = 2r \tag{6.1}$$

$$\text{vol}(S_2(r)) = \pi r^2 \tag{6.2}$$

$$\text{vol}(S_3(r)) = \frac{4}{3}\pi r^3 \tag{6.3}$$

As per the derivation in Appendix 6.7, the general equation for the volume of a d-dimensional hypersphere is given as

$$\text{vol}(S_d(r)) = K_d r^d = \left(\frac{\pi^{\frac{d}{2}}}{\Gamma\left(\frac{d}{2}+1\right)}\right) r^d \tag{6.4}$$

where

$$K_d = \frac{\pi^{d/2}}{\Gamma(\frac{d}{2}+1)} \tag{6.5}$$

is a scalar that depends on the dimensionality d, and Γ is the gamma function [Eq. (3.17)], defined as (for $\alpha > 0$)

$$\Gamma(\alpha) = \int_0^\infty x^{\alpha-1} e^{-x} dx \tag{6.6}$$

By direct integration of Eq. (6.6), we have

$$\Gamma(1) = 1 \qquad \text{and} \qquad \Gamma\left(\frac{1}{2}\right) = \sqrt{\pi} \tag{6.7}$$

The gamma function also has the following property for any $\alpha > 1$:

$$\Gamma(\alpha) = (\alpha-1)\Gamma(\alpha-1) \tag{6.8}$$

For any integer $n \geq 1$, we immediately have

$$\Gamma(n) = (n-1)! \tag{6.9}$$

Turning our attention back to Eq. (6.4), when d is even, then $\frac{d}{2}+1$ is an integer, and by Eq. (6.9) we have

$$\Gamma\left(\frac{d}{2}+1\right) = \left(\frac{d}{2}\right)!$$

and when d is odd, then by Eqs. (6.8) and (6.7), we have

$$\Gamma\left(\frac{d}{2}+1\right) = \left(\frac{d}{2}\right)\left(\frac{d-2}{2}\right)\left(\frac{d-4}{2}\right)\cdots\left(\frac{d-(d-1)}{2}\right)\Gamma\left(\frac{1}{2}\right) = \left(\frac{d!!}{2^{(d+1)/2}}\right)\sqrt{\pi}$$

where $d!!$ denotes the double factorial (or multifactorial), given as

$$d!! = \begin{cases} 1 & \text{if } d = 0 \text{ or } d = 1 \\ d \cdot (d-2)!! & \text{if } d \geq 2 \end{cases}$$

Putting it all together we have

$$\Gamma\left(\frac{d}{2}+1\right) = \begin{cases} \left(\frac{d}{2}\right)! & \text{if } d \text{ is even} \\ \sqrt{\pi}\left(\frac{d!!}{2^{(d+1)/2}}\right) & \text{if } d \text{ is odd} \end{cases} \tag{6.10}$$

Plugging in values of $\Gamma(d/2+1)$ in Eq. (6.4) gives us the equations for the volume of the hypersphere in different dimensions.

Example 6.2. By Eq. (6.10), we have for $d = 1$, $d = 2$ and $d = 3$:

$$\Gamma(1/2+1) = \frac{1}{2}\sqrt{\pi}$$
$$\Gamma(2/2+1) = 1! = 1$$
$$\Gamma(3/2+1) = \frac{3}{4}\sqrt{\pi}$$

Thus, we can verify that the volume of a hypersphere in one, two, and three dimensions is given as

$$\text{vol}(S_1(r)) = \frac{\sqrt{\pi}}{\frac{1}{2}\sqrt{\pi}} r = 2r$$
$$\text{vol}(S_2(r)) = \frac{\pi}{1} r^2 = \pi r^2$$
$$\text{vol}(S_3(r)) = \frac{\pi^{3/2}}{\frac{3}{4}\sqrt{\pi}} r^3 = \frac{4}{3}\pi r^3$$

which match the expressions in Eqs. (6.1), (6.2), and (6.3), respectively.

Surface Area The *surface area* of the hypersphere can be obtained by differentiating its volume with respect to r, given as

$$\text{area}(S_d(r)) = \frac{d}{dr}\text{vol}(S_d(r)) = \left(\frac{\pi^{\frac{d}{2}}}{\Gamma\left(\frac{d}{2}+1\right)}\right) d r^{d-1} = \left(\frac{2\pi^{\frac{d}{2}}}{\Gamma\left(\frac{d}{2}\right)}\right) r^{d-1}$$

We can quickly verify that for two dimensions the surface area of a circle is given as $2\pi r$, and for three dimensions the surface area of sphere is given as $4\pi r^2$.

Asymptotic Volume An interesting observation about the hypersphere volume is that as dimensionality increases, the volume first increases up to a point, and then starts to decrease, and ultimately vanishes. In particular, for the unit hypersphere with $r = 1$,

$$\lim_{d\to\infty} \text{vol}(S_d(1)) = \lim_{d\to\infty} \frac{\pi^{\frac{d}{2}}}{\Gamma(\frac{d}{2}+1)} \to 0$$

Example 6.3. Figure 6.2 plots the volume of the unit hypersphere in Eq. (6.4) with increasing dimensionality. We see that initially the volume increases, and achieves the highest volume for $d = 5$ with $\text{vol}(S_5(1)) = 5.263$. Thereafter, the volume drops rapidly and essentially becomes zero by $d = 30$.

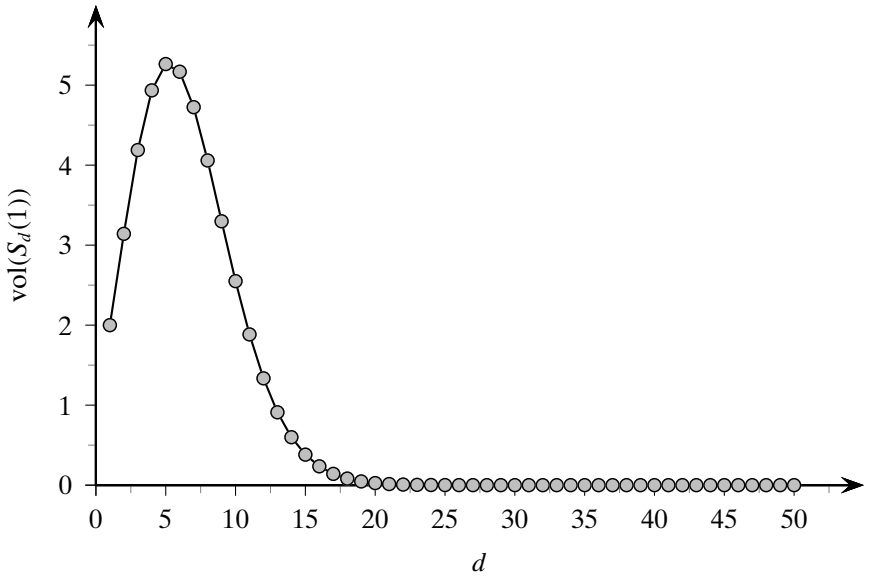


Figure 6.2. Volume of a unit hypersphere.

6.3 HYPERSPHERE INSCRIBED WITHIN HYPERCUBE

We next look at the space enclosed within the largest hypersphere that can be accommodated within a hypercube (which represents the dataspace). Consider a hypersphere of radius r inscribed in a hypercube with sides of length $2r$. When we take the ratio of the volume of the hypersphere of radius r to the hypercube with side length $l = 2r$, we observe the following trends.

In two dimensions, we have

$$\frac{\text{vol}(S_2(r))}{\text{vol}(H_2(2r))} = \frac{\pi r^2}{4r^2} = \frac{\pi}{4} = 78.5\%$$

Thus, an inscribed circle occupies $\frac{\pi}{4}$ of the volume of its enclosing square, as illustrated in Figure 6.3a.

In three dimensions, the ratio is given as

$$\frac{\text{vol}(S_3(r))}{\text{vol}(H_3(2r))} = \frac{\frac{4}{3}\pi r^3}{8r^3} = \frac{\pi}{6} = 52.4\%$$

An inscribed sphere takes up only $\frac{\pi}{6}$ of the volume of its enclosing cube, as shown in Figure 6.3b, which is quite a sharp decrease over the 2-dimensional case.

For the general case, as the dimensionality d increases asymptotically, we get

$$\lim_{d\to\infty} \frac{\text{vol}(S_d(r))}{\text{vol}(H_d(2r))} = \lim_{d\to\infty} \frac{\pi^{d/2}}{2^d \Gamma(\frac{d}{2}+1)} \to 0$$

This means that as the dimensionality increases, most of the volume of the hypercube is in the "corners," whereas the center is essentially empty. The mental picture that

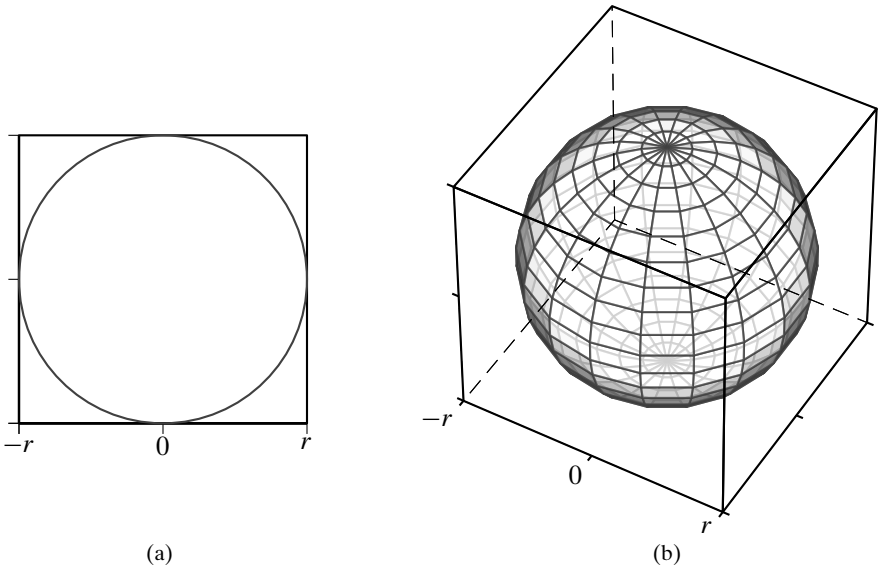


Figure 6.3. Hypersphere inscribed inside a hypercube: in (a) two and (b) three dimensions.

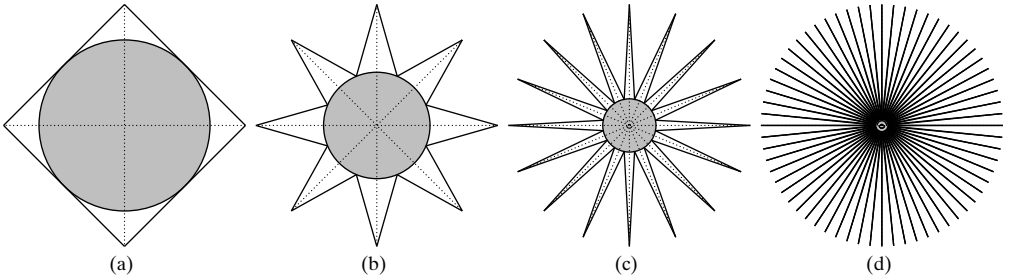


Figure 6.4. Conceptual view of high-dimensional space: (a) two, (b) three, (c) four, and (d) higher dimensions. In d dimensions there are 2^d "corners" and 2^{d-1} diagonals. The radius of the inscribed circle accurately reflects the difference between the volume of the hypercube and the inscribed hypersphere in d dimensions.

emerges is that high-dimensional space looks like a rolled-up porcupine, as illustrated in Figure 6.4.

6.4 VOLUME OF THIN HYPERSPHERE SHELL

Let us now consider the volume of a thin hypersphere shell of width ϵ bounded by an outer hypersphere of radius r, and an inner hypersphere of radius $r - \epsilon$. The volume of the thin shell is given as the difference between the volumes of the two bounding hyperspheres, as illustrated in Figure 6.5.

Let $S_d(r, \epsilon)$ denote the thin hypershell of width ϵ. Its volume is given as

$$\text{vol}(S_d(r, \epsilon)) = \text{vol}(S_d(r)) - \text{vol}(S_d(r - \epsilon)) = K_d r^d - K_d (r - \epsilon)^d.$$

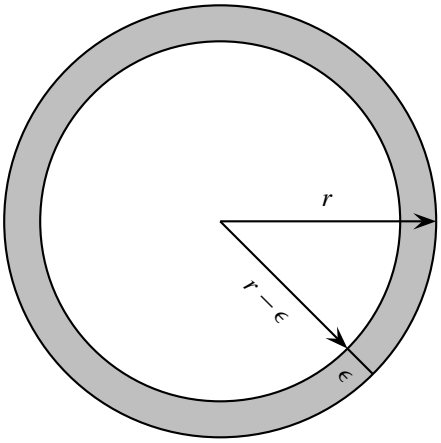


Figure 6.5. Volume of a thin shell (for $\epsilon > 0$).

Let us consider the ratio of the volume of the thin shell to the volume of the outer sphere:

$$\frac{\text{vol}(S_d(r,\epsilon))}{\text{vol}(S_d(r))} = \frac{K_d r^d - K_d(r-\epsilon)^d}{K_d r^d} = 1 - \left(1 - \frac{\epsilon}{r}\right)^d$$

Example 6.4. For example, for a circle in two dimensions, with $r = 1$ and $\epsilon = 0.01$ the volume of the thin shell is $1 - (0.99)^2 = 0.0199 \simeq 2\%$. As expected, in two-dimensions, the thin shell encloses only a small fraction of the volume of the original hypersphere. For three dimensions this fraction becomes $1 - (0.99)^3 = 0.0297 \simeq 3\%$, which is still a relatively small fraction.

Asymptotic Volume

As d increases, in the limit we obtain

$$\lim_{d\to\infty} \frac{\text{vol}(S_d(r,\epsilon))}{\text{vol}(S_d(r))} = \lim_{d\to\infty} 1 - \left(1 - \frac{\epsilon}{r}\right)^d \to 1$$

That is, almost all of the volume of the hypersphere is contained in the thin shell as $d \to \infty$. This means that in high-dimensional spaces, unlike in lower dimensions, most of the volume is concentrated around the surface (within ϵ) of the hypersphere, and the center is essentially void. In other words, if the data is distributed uniformly in the d-dimensional space, then all of the points essentially lie on the boundary of the space (which is a $d-1$ dimensional object). Combined with the fact that most of the hypercube volume is in the corners, we can observe that in high dimensions, data tends to get scattered on the boundary and corners of the space.

6.5 DIAGONALS IN HYPERSPACE

Another counterintuitive behavior of high-dimensional spaces deals with the diagonals. Let us assume that we have a d-dimensional hypercube, with origin $\mathbf{0}_d = (0_1, 0_2, \ldots, 0_d)$, and bounded in each dimension in the range $[-1, 1]$. Then each "corner" of the hyperspace is a d-dimensional vector of the form $(\pm 1_1, \pm 1_2, \ldots, \pm 1_d)^T$. Let $\mathbf{e}_i = (0_1, \ldots, 1_i, \ldots, 0_d)^T$ denote the d-dimensional canonical unit vector in dimension i, and let $\mathbf{1}$ denote the d-dimensional diagonal vector $(1_1, 1_2, \ldots, 1_d)^T$.

Consider the angle θ_d between the diagonal vector $\mathbf{1}$ and the first axis $\mathbf{e}_1$, in d dimensions:

$$\cos\theta_d = \frac{\mathbf{e}_1^T\mathbf{1}}{\|\mathbf{e}_1\|\,\|\mathbf{1}\|} = \frac{\mathbf{e}_1^T\mathbf{1}}{\sqrt{\mathbf{e}_1^T\mathbf{e}_1}\sqrt{\mathbf{1}^T\mathbf{1}}} = \frac{1}{\sqrt{1}\sqrt{d}} = \frac{1}{\sqrt{d}}$$

Example 6.5. Figure 6.6 illustrates the angle between the diagonal vector $\mathbf{1}$ and $\mathbf{e}_1$, for $d = 2$ and $d = 3$. In two dimensions, we have $\cos\theta_2 = \frac{1}{\sqrt{2}}$ whereas in three dimensions, we have $\cos\theta_3 = \frac{1}{\sqrt{3}}$.

Asymptotic Angle

As d increases, the angle between the d-dimensional diagonal vector $\mathbf{1}$ and the first axis vector $\mathbf{e}_1$ is given as

$$\lim_{d\to\infty} \cos\theta_d = \lim_{d\to\infty} \frac{1}{\sqrt{d}} \to 0$$

which implies that

$$\lim_{d\to\infty} \theta_d \to \frac{\pi}{2} = 90^\circ$$

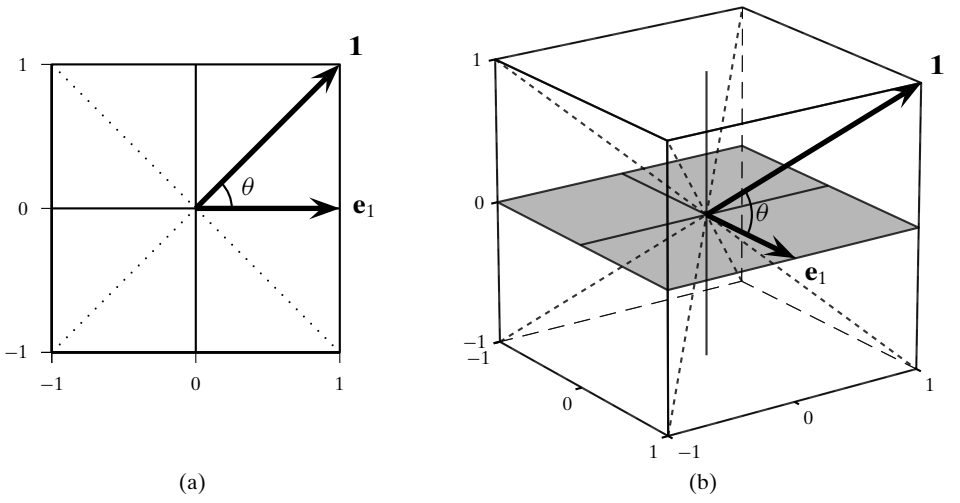


Figure 6.6. Angle between diagonal vector $\mathbf{1}$ and $\mathbf{e}_1$: in (a) two and (b) three dimensions.

This analysis holds for the angle between the diagonal vector $\mathbf{1}_d$ and any of the d principal axis vectors $\mathbf{e}_i$ (i.e., for all $i \in [1, d]$). In fact, the same result holds for any diagonal vector and any principal axis vector (in both directions). This implies that in high dimensions all of the diagonal vectors are perpendicular (or orthogonal) to all the coordinates axes! Because there are 2^d corners in a d-dimensional hyperspace, there are 2^d diagonal vectors from the origin to each of the corners. Because the diagonal vectors in opposite directions define a new axis, we obtain 2^{d-1} new axes, each of which is essentially orthogonal to all of the d principal coordinate axes! Thus, in effect, high-dimensional space has an exponential number of orthogonal "axes." A consequence of this strange property of high-dimensional space is that if there is a point or a group of points, say a cluster of interest, near a diagonal, these points will get projected into the origin and will not be visible in lower dimensional projections.

6.6 DENSITY OF THE MULTIVARIATE NORMAL

Let us consider how, for the standard multivariate normal distribution, the density of points around the mean changes in d dimensions. In particular, consider the probability of a point being within a fraction $\alpha > 0$, of the peak density at the mean.

For a multivariate normal distribution [Eq. (2.33)], with $\boldsymbol{\mu} = \mathbf{0}_d$ (the d-dimensional zero vector), and $\boldsymbol{\Sigma} = \mathbf{I}_d$ (the $d \times d$ identity matrix), we have

$$f(\mathbf{x}) = \frac{1}{(\sqrt{2\pi})^d} \exp\left\{-\frac{\mathbf{x}^T\mathbf{x}}{2}\right\} \tag{6.11}$$

At the mean $\mu = \mathbf{0}_d$, the peak density is $f(\mathbf{0}_d) = \frac{1}{(\sqrt{2\pi})^d}$. Thus, the set of points $\mathbf{x}$ with density at least α fraction of the density at the mean, with $0 < \alpha < 1$, is given as

$$\frac{f(\mathbf{x})}{f(\mathbf{0})} \geq \alpha$$

which implies that

$$\exp\left\{-\frac{\mathbf{x}^T\mathbf{x}}{2}\right\} \geq \alpha$$

$$\text{or } \mathbf{x}^T\mathbf{x} \leq -2\ln(\alpha)$$

$$\text{and thus } \sum_{i=1}^{d} (x_i)^2 \leq -2\ln(\alpha) \tag{6.12}$$

It is known that if the random variables X_1, X_2, …, X_k are independent and identically distributed, and if each variable has a standard normal distribution, then their squared sum $X^2 + X_2^2 + \cdots + X_k^2$ follows a χ^2 distribution with k degrees of freedom, denoted as χ_k^2. Because the projection of the standard multivariate normal onto any attribute X_j is a standard univariate normal, we conclude that $\mathbf{x}^T\mathbf{x} = \sum_{i=1}^{d} (x_i)^2$ has a χ^2 distribution with d degrees of freedom. The probability that a point $\mathbf{x}$ is within α times the density at the mean can be computed from the χ_d^2 density function using Eq. (6.12),

as follows:

$$P\left(\frac{f(\mathbf{x})}{f(\mathbf{0})} \geq \alpha\right) = P\left(\mathbf{x}^T\mathbf{x} \leq -2\ln(\alpha)\right)$$
$$= \int_0^{-2\ln(\alpha)} f_{\chi^2_d}(\mathbf{x}^T\mathbf{x})$$
$$= F_{\chi^2_d}(-2\ln(\alpha)) \tag{6.13}$$

where $f_{\chi^2_q}(x)$ is the chi-squared probability density function [Eq. (3.16)] with q degrees of freedom:

$$f_{\chi^2_q}(x) = \frac{1}{2^{q/2}\Gamma(q/2)} x^{\frac{q}{2}-1} e^{-\frac{x}{2}}$$

and $F_{\chi^2_q}(x)$ is its cumulative distribution function.

As dimensionality increases, this probability decreases sharply, and eventually tends to zero, that is,

$$\lim_{d\to\infty} P\left(\mathbf{x}^T\mathbf{x} \leq -2\ln(\alpha)\right) \to 0 \tag{6.14}$$

Thus, in higher dimensions the probability density around the mean decreases very rapidly as one moves away from the mean. In essence the entire probability mass migrates to the tail regions.

Example 6.6. Consider the probability of a point being within 50% of the density at the mean, that is, $\alpha = 0.5$. From Eq. (6.13) we have

$$P\left(\mathbf{x}^T\mathbf{x} \leq -2\ln(0.5)\right) = F_{\chi^2_d}(1.386)$$

We can compute the probability of a point being within 50% of the peak density by evaluating the cumulative χ^2 distribution for different degrees of freedom (the number of dimensions). For $d = 1$, we find that the probability is $F_{\chi^2_1}(1.386) = 76.1\%$. For $d = 2$ the probability decreases to $F_{\chi^2_2}(1.386) = 50\%$, and for $d = 3$ it reduces to 29.12%. Looking at Figure 6.7, we can see that only about 24% of the density is in the tail regions for one dimension, but for two dimensions more than 50% of the density is in the tail regions.

Figure 6.8 plots the χ^2_d distribution and shows the probability $P\left(\mathbf{x}^T\mathbf{x} \leq 1.386\right)$ for two and three dimensions. This probability decreases rapidly with dimensionality; by $d = 10$, it decreases to 0.075%, that is, 99.925% of the points lie in the extreme or tail regions.

Distance of Points from the Mean

Let us consider the average distance of a point $\mathbf{x}$ from the center of the standard multivariate normal. Let r^2 denote the square of the distance of a point $\mathbf{x}$ to the center $\boldsymbol{\mu} = \mathbf{0}$, given as

$$r^2 = \|\mathbf{x} - \mathbf{0}\|^2 = \mathbf{x}^T\mathbf{x} = \sum_{i=1}^{d} x_i^2$$

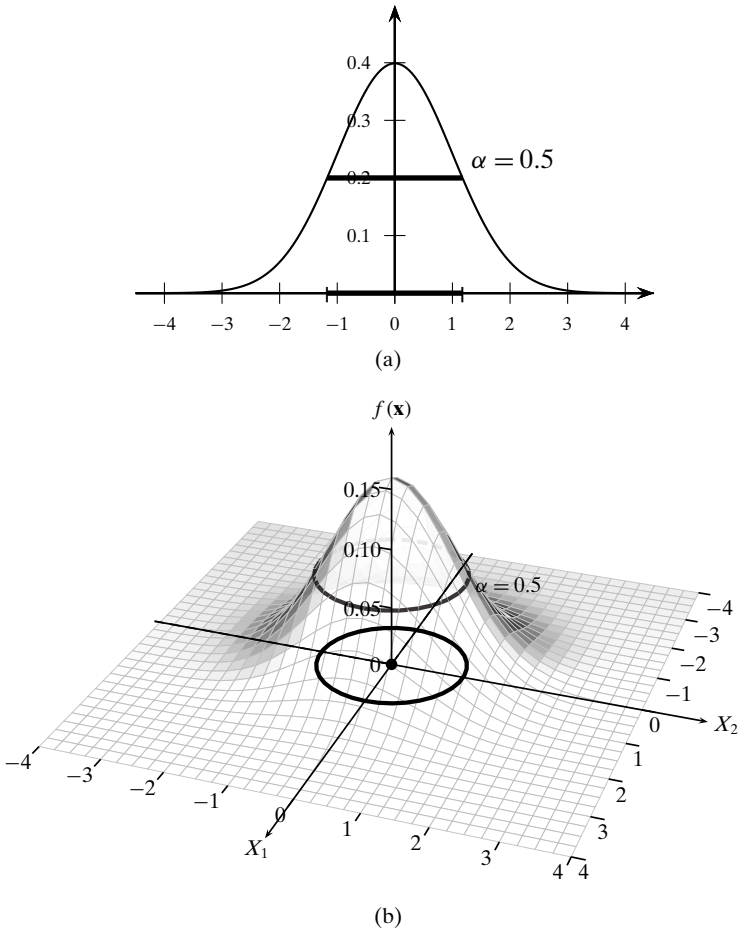


Figure 6.7. Density contour for α fraction of the density at the mean: in (a) one and (b) two dimensions.

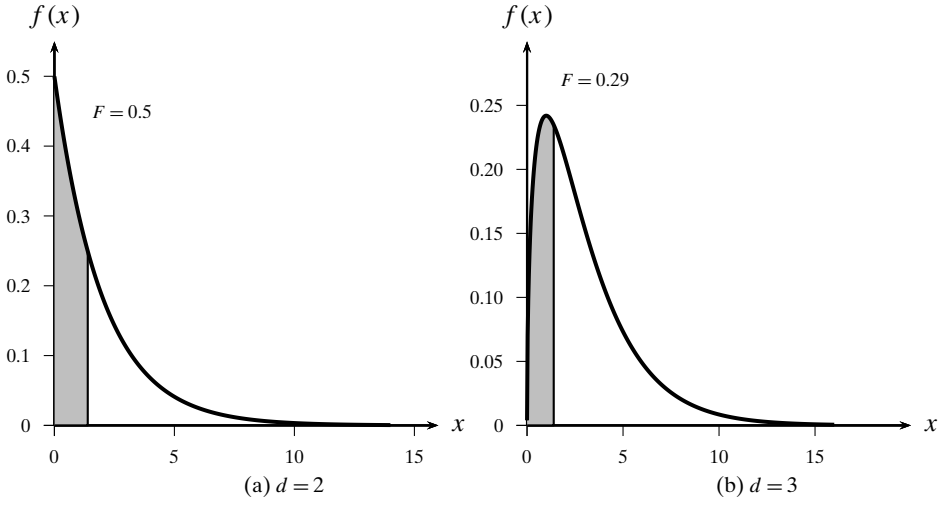


Figure 6.8. Probability $P(\mathbf{x}^T\mathbf{x} \leq -2\ln(\alpha))$, with $\alpha = 0.5$.

$\mathbf{x}^T\mathbf{x}$ follows a χ^2 distribution with d degrees of freedom, which has mean d and variance $2d$. It follows that the mean and variance of the random variable r^2 is

$$\mu_{r^2} = d \qquad \sigma^2_{r^2} = 2d$$

By the central limit theorem, as $d \to \infty$, r^2 is approximately normal with mean d and variance $2d$, which implies that r^2 is concentrated about its mean value of d. As a consequence, the distance r of a point $\mathbf{x}$ to the center of the standard multivariate normal is likewise approximately concentrated around its mean $\sqrt{d}$.

Next, to estimate the spread of the distance r around its mean value, we need to derive the standard deviation of r from that of r^2. Assuming that σ_r is much smaller compared to r, then using the fact that $\frac{d \log r}{dr} = \frac{1}{r}$, after rearranging the terms, we have

$$\begin{aligned} \frac{dr}{r} &= d \log r \\ &= \frac{1}{2} d \log r^2 \end{aligned}$$

Using the fact that $\frac{d \log r^2}{dr^2} = \frac{1}{r^2}$, and rearranging the terms, we obtain

$$\frac{dr}{r} = \frac{1}{2} \frac{dr^2}{r^2}$$

which implies that $dr = \frac{1}{2r} dr^2$. Setting the change in r^2 equal to the standard deviation of r^2, we have $dr^2 = \sigma_{r^2} = \sqrt{2d}$, and setting the mean radius $r = \sqrt{d}$, we have

$$\sigma_r = dr = \frac{1}{2\sqrt{d}} \sqrt{2d} = \frac{1}{\sqrt{2}}$$

We conclude that for large d, the radius r (or the distance of a point $\mathbf{x}$ from the origin $\mathbf{0}$) follows a normal distribution with mean $\sqrt{d}$ and standard deviation $1/\sqrt{2}$. Nevertheless, the density at the mean distance $\sqrt{d}$, is exponentially smaller than that at the peak density because

$$\frac{f(\mathbf{x})}{f(\mathbf{0})} = \exp\left\{-\mathbf{x}^T\mathbf{x}/2\right\} = \exp\{-d/2\}$$

Combined with the fact that the probability mass migrates away from the mean in high dimensions, we have another interesting observation, namely that, whereas the density of the standard multivariate normal is maximized at the center $\mathbf{0}$, most of the probability mass (the points) is concentrated in a small band around the mean distance of $\sqrt{d}$ from the center.

6.7 APPENDIX: DERIVATION OF HYPERSPHERE VOLUME

The volume of the hypersphere can be derived via integration using spherical polar coordinates. We consider the derivation in two and three dimensions, and then for a general d.

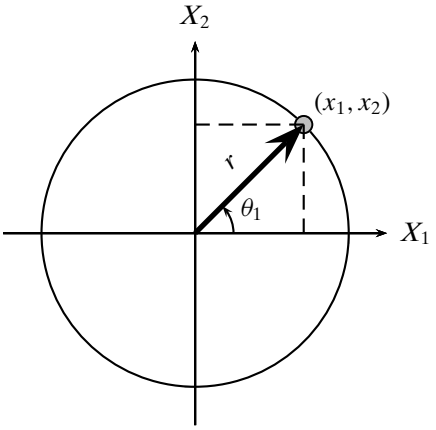


Figure 6.9. Polar coordinates in two dimensions.

Volume in Two Dimensions

As illustrated in Figure 6.9, in $d = 2$ dimensions, the point $\mathbf{x} = (x_1, x_2) \in \mathbb{R}^2$ can be expressed in polar coordinates as follows:

$$x_1 = r\cos\theta_1 = rc_1$$
$$x_2 = r\sin\theta_1 = rs_1$$

where $r = \|\mathbf{x}\|$, and we use the notation $\cos\theta_1 = c_1$ and $\sin\theta_1 = s_1$ for convenience.

The *Jacobian matrix* for this transformation is given as

$$J(\theta_1) = \begin{pmatrix} \frac{\partial x_1}{\partial r} & \frac{\partial x_1}{\partial \theta_1} \\ \frac{\partial x_2}{\partial r} & \frac{\partial x_2}{\partial \theta_1} \end{pmatrix} = \begin{pmatrix} c_1 & -rs_1 \\ s_1 & rc_1 \end{pmatrix}$$

The determinant of the Jacobian matrix is called the *Jacobian*. For $J(\theta_1)$, the Jacobian is given as

$$\det(J(\theta_1)) = rc_1^2 + rs_1^2 = r(c_1^2 + s_1^2) = r \tag{6.15}$$

Using the Jacobian in Eq. (6.15), the volume of the hypersphere in two dimensions can be obtained by integration over r and θ_1 (with $r > 0$, and $0 \le \theta_1 \le 2\pi$)

$$\begin{aligned} \text{vol}(S_2(r)) &= \int_r \int_{\theta_1} \Big|\det(J(\theta_1))\Big| \, dr \, d\theta_1 \\ &= \int_0^r \int_0^{2\pi} r \, dr \, d\theta_1 = \int_0^r r \, dr \int_0^{2\pi} d\theta_1 \\ &= \frac{r^2}{2}\Big|_0^r \cdot \theta_1 \Big|_0^{2\pi} = \pi r^2 \end{aligned}$$

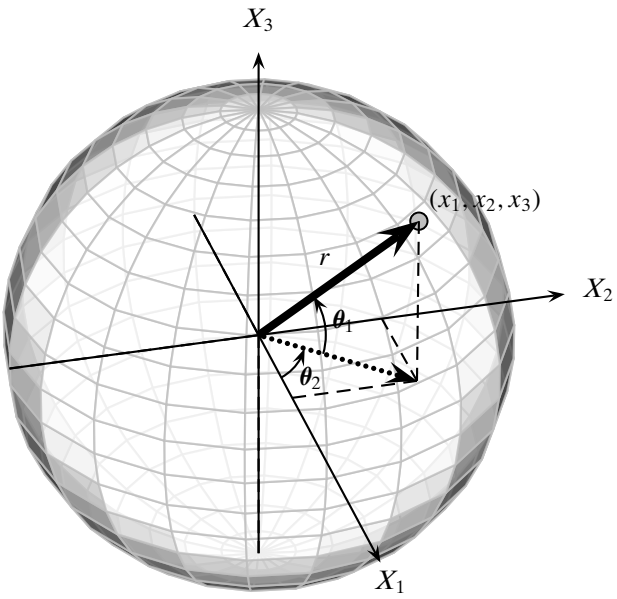


Figure 6.10. Polar coordinates in three dimensions.

Volume in Three Dimensions

As illustrated in Figure 6.10, in $d = 3$ dimensions, the point $\mathbf{x} = (x_1, x_2, x_3) \in \mathbb{R}^3$ can be expressed in polar coordinates as follows:

$$x_1 = r\cos\theta_1\cos\theta_2 = rc_1c_2$$
$$x_2 = r\cos\theta_1\sin\theta_2 = rc_1s_2$$
$$x_3 = r\sin\theta_1 = rs_1$$

where $r = \|\mathbf{x}\|$, and we used the fact that the dotted vector that lies in the X_1–X_2 plane in Figure 6.10 has magnitude $r\cos\theta_1$.

The Jacobian matrix is given as

$$J(\theta_1, \theta_2) = \begin{pmatrix} \frac{\partial x_1}{\partial r} & \frac{\partial x_1}{\partial \theta_1} & \frac{\partial x_1}{\partial \theta_2} \\ \frac{\partial x_2}{\partial r} & \frac{\partial x_2}{\partial \theta_1} & \frac{\partial x_2}{\partial \theta_2} \\ \frac{\partial x_3}{\partial r} & \frac{\partial x_3}{\partial \theta_1} & \frac{\partial x_3}{\partial \theta_2} \end{pmatrix} = \begin{pmatrix} c_1c_2 & -rs_1c_2 & -rc_1s_2 \\ c_1s_2 & -rs_1s_2 & rc_1c_2 \\ s_1 & rc_1 & 0 \end{pmatrix}$$

The Jacobian is then given as

$$\begin{aligned} \det(J(\theta_1, \theta_2)) &= s_1(-rs_1)(c_1)\det(J(\theta_2)) - rc_1c_1c_1\det(J(\theta_2)) \\ &= -r^2c_1(s_1^2 + c_2^2) = -r^2c_1 \end{aligned} \tag{6.16}$$

In computing this determinant we made use of the fact that if a column of a matrix $\mathbf{A}$ is multiplied by a scalar s, then the resulting determinant is $s\det(\mathbf{A})$. We also relied on the fact that the $(3,1)$-*minor* of $J(\theta_1, \theta_2)$, obtained by deleting row 3 and column 1 is actually $J(\theta_2)$ with the first column multiplied by $-rs_1$ and the second column

multiplied by c_1. Likewise, the $(3,2)$-minor of $J(\theta_1, \theta_2))$ is $J(\theta_2)$ with both the columns multiplied by c_1.

The volume of the hypersphere for $d = 3$ is obtained via a triple integral with $r > 0$, $-\pi/2 \leq \theta_1 \leq \pi/2$, and $0 \leq \theta_2 \leq 2\pi$

$$\begin{aligned}
\text{vol}(S_3(r)) &= \int_r \int_{\theta_1} \int_{\theta_2} \Big|\det(J(\theta_1, \theta_2))\Big| \, dr \, d\theta_1 \, d\theta_2 \\
&= \int_0^r \int_{-\pi/2}^{\pi/2} \int_0^{2\pi} r^2 \cos\theta_1 \, dr \, d\theta_1 \, d\theta_2 = \int_0^r r^2 \, dr \int_{-\pi/2}^{\pi/2} \cos\theta_1 d\theta_1 \int_0^{2\pi} d\theta_2 \\
&= \frac{r^3}{3}\Big|_0^r \cdot \sin\theta_1 \Big|_{-\pi/2}^{\pi/2} \cdot \theta_2 \Big|_0^{2\pi} = \frac{r^3}{3} \cdot 2 \cdot 2\pi = \frac{4}{3}\pi r^3 \qquad (6.17)
\end{aligned}$$

Volume in d Dimensions

Before deriving a general expression for the hypersphere volume in d dimensions, let us consider the Jacobian in four dimensions. Generalizing the polar coordinates from three dimensions in Figure 6.10 to four dimensions, we obtain

$$\begin{aligned}
x_1 &= r\cos\theta_1\cos\theta_2\cos\theta_3 = rc_2c_2c_3 \\
x_2 &= r\cos\theta_1\cos\theta_2\sin\theta_3 = rc_1c_2s_3 \\
x_3 &= r\cos\theta_1\sin\theta_2 = rc_1s_1 \\
x_4 &= r\sin\theta_1 = rs_1
\end{aligned}$$

The Jacobian matrix is given as

$$J(\theta_1, \theta_2, \theta_3) = \begin{pmatrix} \frac{\partial x_1}{\partial r} & \frac{\partial x_1}{\partial \theta_1} & \frac{\partial x_1}{\partial \theta_2} & \frac{\partial x_1}{\partial \theta_3} \\ \frac{\partial x_2}{\partial r} & \frac{\partial x_2}{\partial \theta_1} & \frac{\partial x_2}{\partial \theta_2} & \frac{\partial x_2}{\partial \theta_3} \\ \frac{\partial x_3}{\partial r} & \frac{\partial x_3}{\partial \theta_1} & \frac{\partial x_3}{\partial \theta_2} & \frac{\partial x_3}{\partial \theta_3} \\ \frac{\partial x_4}{\partial r} & \frac{\partial x_4}{\partial \theta_1} & \frac{\partial x_4}{\partial \theta_2} & \frac{\partial x_4}{\partial \theta_3} \end{pmatrix} = \begin{pmatrix} c_1c_2c_3 & -rs_1c_2c_3 & -rc_1s_2c_3 & rc_1c_2s_3 \\ c_1c_2s_3 & -rs_1c_2s_3 & -rc_1s_2s_3 & rc_1c_2c_3 \\ c_1s_2 & -rs_1s_2 & rc_1c_2 & 0 \\ s_1 & rc_1 & 0 & 0 \end{pmatrix}$$

Utilizing the Jacobian in three dimensions [Eq. (6.16)], the Jacobian in four dimensions is given as

$$\begin{aligned}
\det(J(\theta_1, \theta_2, \theta_3)) &= s_1(-rs_1)(c_1)(c_1)\det(J(\theta_2, \theta_3)) - rc_1(c_1)(c_1)(c_1)\det(J(\theta_2, \theta_3)) \\
&= r^3 s_1^2 c_1^2 c_2 + r^3 c_1^4 c_2 = r^3 c_1^2 c_2 (s_1^2 + c_1^2) = r^3 c_1^2 c_2
\end{aligned}$$

Jacobian in d Dimensions By induction, we can obtain the d-dimensional Jacobian as follows:

$$\det(J(\theta_1, \theta_2, \ldots, \theta_{d-1})) = (-1)^d r^{d-1} c_1^{d-2} c_2^{d-3} \ldots c_{d-2}$$

The volume of the hypersphere is given by the d-dimensional integral with $r > 0$, $-\pi/2 \leq \theta_i \leq \pi/2$ for all $i = 1, \ldots, d-2$, and $0 \leq \theta_{d-1} \leq 2\pi$:

$$\begin{aligned}\text{vol}(S_d(r)) &= \int_r \int_{\theta_1} \int_{\theta_2} \ldots \int_{\theta_{d-1}} \left|\det(J(\theta_1, \theta_2, \ldots, \theta_{d-1}))\right| dr\, d\theta_1\, d\theta_2 \ldots d\theta_{d-1} \\ &= \int_0^r r^{d-1} dr \int_{-\pi/2}^{\pi/2} c_1^{d-2} d\theta_1 \ldots \int_{-\pi/2}^{\pi/2} c_{d-2} d\theta_{d-2} \int_0^{2\pi} d\theta_{d-1} \end{aligned} \tag{6.18}$$

Consider one of the intermediate integrals:

$$\int_{-\pi/2}^{\pi/2} (\cos\theta)^k d\theta = 2 \int_0^{\pi/2} \cos^k \theta d\theta \tag{6.19}$$

Let us substitute $u = \cos^2\theta$, then we have $\theta = \cos^{-1}(u^{1/2})$, and the Jacobian is

$$J = \frac{\partial\theta}{\partial u} = -\frac{1}{2} u^{-1/2} (1-u)^{-1/2} \tag{6.20}$$

Substituting Eq. (6.20) in Eq. (6.19), we get the new integral:

$$\begin{aligned} 2 \int_0^{\pi/2} \cos^k \theta d\theta &= \int_0^1 u^{(k-1)/2} (1-u)^{-1/2} du \\ &= B\left(\frac{k+1}{2}, \frac{1}{2}\right) = \frac{\Gamma\left(\frac{k+1}{2}\right)\Gamma\left(\frac{1}{2}\right)}{\Gamma\left(\frac{k}{2}+1\right)} \end{aligned} \tag{6.21}$$

where $B(\alpha, \beta)$ is the *beta function*, given as

$$B(\alpha, \beta) = \int_0^1 u^{\alpha-1} (1-u)^{\beta-1} du$$

and it can be expressed in terms of the gamma function [Eq. (6.6)] via the identity

$$B(\alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$$

Using the fact that $\Gamma(1/2) = \sqrt{\pi}$, and $\Gamma(1) = 1$, plugging Eq. (6.21) into Eq. (6.18), we get

$$\begin{aligned} \text{vol}(S_d(r)) &= \frac{r^d}{d} \frac{\Gamma\left(\frac{d-1}{2}\right)\Gamma\left(\frac{1}{2}\right)}{\Gamma\left(\frac{d}{2}\right)} \frac{\Gamma\left(\frac{d-2}{2}\right)\Gamma\left(\frac{1}{2}\right)}{\Gamma\left(\frac{d-1}{2}\right)} \ldots \frac{\Gamma(1)\Gamma\left(\frac{1}{2}\right)}{\Gamma\left(\frac{3}{2}\right)} 2\pi \\ &= \frac{\pi\,\Gamma\left(\frac{1}{2}\right)^{d/2-1} r^d}{\frac{d}{2}\Gamma\left(\frac{d}{2}\right)} \\ &= \left(\frac{\pi^{d/2}}{\Gamma\left(\frac{d}{2}+1\right)}\right) r^d \end{aligned}$$

which matches the expression in Eq. (6.4).

6.8 FURTHER READING

For an introduction to the geometry of d-dimensional spaces see Kendall (1961) and also Scott (1992, Section 1.5). The derivation of the mean distance for the multivariate normal is from MacKay (2003, p. 130).

Kendall, M. G. (1961). *A Course in the Geometry of n Dimensions*. New York: Hafner.
MacKay, D. J. (2003). *Information Theory, Inference and Learning Algorithms*. New York: Cambridge University Press.
Scott, D. W. (1992). *Multivariate Density Estimation: Theory, Practice, and Visualization*. New York: John Wiley & Sons.

6.9 EXERCISES

Q1. Given the gamma function in Eq. (6.6), show the following:
- **(a)** $\Gamma(1) = 1$
- **(b)** $\Gamma\left(\frac{1}{2}\right) = \sqrt{\pi}$
- **(c)** $\Gamma(\alpha) = (\alpha - 1)\Gamma(\alpha - 1)$

Q2. Show that the asymptotic volume of the hypersphere $S_d(r)$ for any value of radius r eventually tends to zero as d increases.

Q3. The ball with center $\mathbf{c} \in \mathbb{R}^d$ and radius r is defined as

$$B_d(\mathbf{c}, r) = \left\{\mathbf{x} \in \mathbb{R}^d \mid \delta(\mathbf{x}, \mathbf{c}) \leq r\right\}$$

where $\delta(\mathbf{x}, \mathbf{c})$ is the distance between $\mathbf{x}$ and $\mathbf{c}$, which can be specified using the L_p-norm:

$$L_p(\mathbf{x}, \mathbf{c}) = \left(\sum_{i=1}^{d} |x_i - c_i|^p\right)^{\frac{1}{p}}$$

where $p \neq 0$ is any real number. The distance can also be specified using the L_∞-norm:

$$L_\infty(\mathbf{x}, \mathbf{c}) = \max_i\left\{|x_i - c_i|\right\}$$

Answer the following questions:
- **(a)** For $d = 2$, sketch the shape of the hyperball inscribed inside the unit square, using the L_p-distance with $p = 0.5$ and with center $\mathbf{c} = (0.5, 0.5)^T$.
- **(b)** With $d = 2$ and $\mathbf{c} = (0.5, 0.5)^T$, using the L_∞-norm, sketch the shape of the ball of radius $r = 0.25$ inside a unit square.
- **(c)** Compute the formula for the maximum distance between any two points in the unit hypercube in d dimensions, when using the L_p-norm. What is the maximum distance for $p = 0.5$ when $d = 2$? What is the maximum distance for the L_∞-norm?

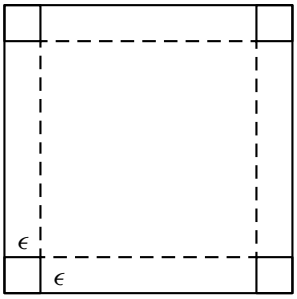


Figure 6.11. For Q4.

Q4. Consider the corner hypercubes of length $\epsilon \leq 1$ inside a unit hypercube. The 2-dimensional case is shown in Figure 6.11. Answer the following questions:

- **(a)** Let $\epsilon = 0.1$. What is the fraction of the total volume occupied by the corner cubes in two dimensions?
- **(b)** Derive an expression for the volume occupied by all of the corner hypercubes of length $\epsilon < 1$ as a function of the dimension d. What happens to the fraction of the volume in the corners as $d \to \infty$?
- **(c)** What is the fraction of volume occupied by the thin hypercube shell of width $\epsilon < 1$ as a fraction of the total volume of the outer (unit) hypercube, as $d \to \infty$? For example, in two dimensions the thin shell is the space between the outer square (solid) and inner square (dashed).

Q5. Prove Eq. (6.14), that is, $\lim_{d\to\infty} P\left(\mathbf{x}^T\mathbf{x} \leq -2\ln(\alpha)\right) \to 0$, for any $\alpha \in (0, 1)$ and $\mathbf{x} \in \mathbb{R}^d$.

Q6. Consider the conceptual view of high-dimensional space shown in Figure 6.4. Derive an expression for the radius of the inscribed circle, so that the area in the spokes accurately reflects the difference between the volume of the hypercube and the inscribed hypersphere in d dimensions. For instance, if the length of a half-diagonal is fixed at 1, then the radius of the inscribed circle is $\frac{1}{\sqrt{2}}$ in Figure 6.4a.

Q7. Consider the unit hypersphere (with radius $r = 1$). Inside the hypersphere inscribe a hypercube (i.e., the largest hypercube you can fit inside the hypersphere). An example in two dimensions is shown in Figure 6.12. Answer the following questions:

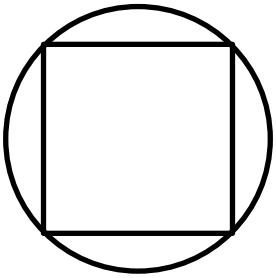


Figure 6.12. For Q7.

(a) Derive an expression for the volume of the inscribed hypercube for any given dimensionality d. Derive the expression for one, two, and three dimensions, and then generalize to higher dimensions.

(b) What happens to the ratio of the volume of the inscribed hypercube to the volume of the enclosing hypersphere as $d \to \infty$? Again, give the ratio in one, two and three dimensions, and then generalize.

Q8. Assume that a unit hypercube is given as $[0,1]^d$, that is, the range is $[0,1]$ in each dimension. The main diagonal in the hypercube is defined as the vector from $(\mathbf{0}, 0) = (\overbrace{0, \ldots, 0}^{d-1}, 0)$ to $(\mathbf{1}, 1) = (\overbrace{1, \ldots, 1}^{d-1}, 1)$. For example, when $d = 2$, the main diagonal goes from $(0,0)$ to $(1,1)$. On the other hand, the main anti-diagonal is defined as the vector from $(\mathbf{1}, 0) = (\overbrace{1, \ldots, 1}^{d-1}, 0)$ to $(\mathbf{0}, 1) = (\overbrace{0, \ldots, 0}^{d-1}, 1)$ For example, for $d = 2$, the anti-diagonal is from $(1,0)$ to $(0,1)$.

(a) Sketch the diagonal and anti-diagonal in $d = 3$ dimensions, and compute the angle between them.

(b) What happens to the angle between the main diagonal and anti-diagonal as $d \to \infty$. First compute a general expression for the d dimensions, and then take the limit as $d \to \infty$.

Q9. Draw a sketch of a hypersphere in four dimensions.

CHAPTER 7 Dimensionality Reduction

We saw in Chapter 6 that high-dimensional data has some peculiar characteristics, some of which are counterintuitive. For example, in high dimensions the center of the space is devoid of points, with most of the points being scattered along the surface of the space or in the corners. There is also an apparent proliferation of orthogonal axes. As a consequence high-dimensional data can cause problems for data mining and analysis, although in some cases high-dimensionality can help, for example, for nonlinear classification. Nevertheless, it is important to check whether the dimensionality can be reduced while preserving the essential properties of the full data matrix. This can aid data visualization as well as data mining. In this chapter we study methods that allow us to obtain optimal lower-dimensional projections of the data.

7.1 BACKGROUND

Let the data $\mathbf{D}$ consist of n points over d attributes, that is, it is an $n \times d$ matrix, given as

$$\mathbf{D} = \begin{pmatrix} & X_1 & X_2 & \cdots & X_d \\ \mathbf{x}_1 & x_{11} & x_{12} & \cdots & x_{1d} \\ \mathbf{x}_2 & x_{21} & x_{22} & \cdots & x_{2d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{x}_n & x_{n1} & x_{n2} & \cdots & x_{nd} \end{pmatrix}$$

Each point $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{id})^T$ is a vector in the ambient d-dimensional vector space spanned by the d standard basis vectors $\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_d$, where $\mathbf{e}_i$ corresponds to the ith attribute X_i. Recall that the standard basis is an orthonormal basis for the data space, that is, the basis vectors are pairwise orthogonal, $\mathbf{e}_i^T \mathbf{e}_j = 0$, and have unit length $\|\mathbf{e}_i\| = 1$.

As such, given any other set of d orthonormal vectors $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_d$, with $\mathbf{u}_i^T \mathbf{u}_j = 0$ and $\|\mathbf{u}_i\| = 1$ (or $\mathbf{u}_i^T \mathbf{u}_i = 1$), we can re-express each point $\mathbf{x}$ as the linear combination

$$\mathbf{x} = a_1 \mathbf{u}_1 + a_2 \mathbf{u}_2 + \cdots + a_d \mathbf{u}_d \tag{7.1}$$

where the vector $\mathbf{a} = (a_1, a_2, \ldots, a_d)^T$ represents the coordinates of $\mathbf{x}$ in the new basis. The above linear combination can also be expressed as a matrix multiplication:

$$\mathbf{x} = \mathbf{U}\mathbf{a} \tag{7.2}$$

where $\mathbf{U}$ is the $d \times d$ matrix, whose ith column comprises the ith basis vector $\mathbf{u}_i$:

$$\mathbf{U} = \begin{pmatrix} | & | & & | \\ \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_d \\ | & | & & | \end{pmatrix}$$

The matrix $\mathbf{U}$ is an *orthogonal* matrix, whose columns, the basis vectors, are *orthonormal*, that is, they are pairwise orthogonal and have unit length

$$\mathbf{u}_i^T \mathbf{u}_j = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

Because $\mathbf{U}$ is orthogonal, this means that its inverse equals its transpose:

$$\mathbf{U}^{-1} = \mathbf{U}^T$$

which implies that $\mathbf{U}^T\mathbf{U} = \mathbf{I}$, where $\mathbf{I}$ is the $d \times d$ identity matrix.

Multiplying Eq. (7.2) on both sides by $\mathbf{U}^T$ yields the expression for computing the coordinates of $\mathbf{x}$ in the new basis

$$\begin{aligned} \mathbf{U}^T\mathbf{x} &= \mathbf{U}^T\mathbf{U}\mathbf{a} \\ \mathbf{a} &= \mathbf{U}^T\mathbf{x} \end{aligned} \tag{7.3}$$

Example 7.1. Figure 7.1a shows the centered Iris dataset, with $n = 150$ points, in the $d = 3$ dimensional space comprising the `sepal length` (X_1), `sepal width` (X_2), and `petal length` (X_3) attributes. The space is spanned by the standard basis vectors

$$\mathbf{e}_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \qquad \mathbf{e}_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \qquad \mathbf{e}_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

Figure 7.1b shows the same points in the space comprising the new basis vectors

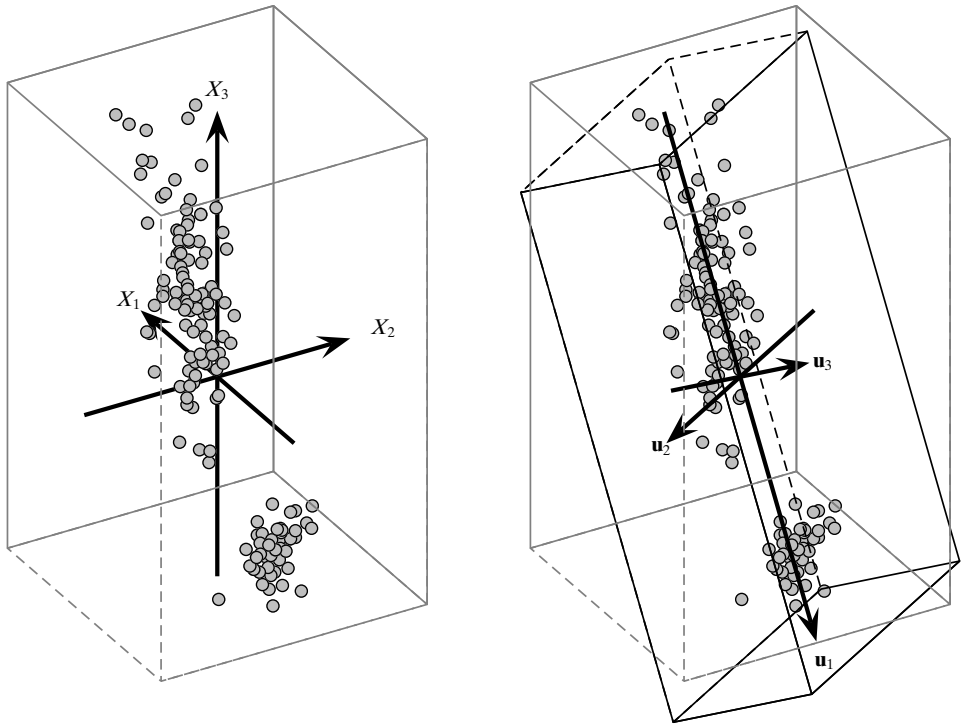
$$\mathbf{u}_1 = \begin{pmatrix} -0.390 \\ 0.089 \\ -0.916 \end{pmatrix} \qquad \mathbf{u}_2 = \begin{pmatrix} -0.639 \\ -0.742 \\ 0.200 \end{pmatrix} \qquad \mathbf{u}_3 = \begin{pmatrix} -0.663 \\ 0.664 \\ 0.346 \end{pmatrix}$$

For example, the new coordinates of the centered point $\mathbf{x} = (-0.343, -0.754, 0.241)^T$ can be computed as

$$\mathbf{a} = \mathbf{U}^T\mathbf{x} = \begin{pmatrix} -0.390 & 0.089 & -0.916 \\ -0.639 & -0.742 & 0.200 \\ -0.663 & 0.664 & 0.346 \end{pmatrix} \begin{pmatrix} -0.343 \\ -0.754 \\ 0.241 \end{pmatrix} = \begin{pmatrix} -0.154 \\ 0.828 \\ -0.190 \end{pmatrix}$$

One can verify that $\mathbf{x}$ can be written as the linear combination

$$\mathbf{x} = -0.154\mathbf{u}_1 + 0.828\mathbf{u}_2 - 0.190\mathbf{u}_3$$



(a) Original Basis (b) Optimal Basis

Figure 7.1. Iris data: optimal basis in three dimensions.

Because there are potentially infinite choices for the set of orthonormal basis vectors, one natural question is whether there exists an *optimal* basis, for a suitable notion of optimality. Further, it is often the case that the input dimensionality d is very large, which can cause various problems owing to the curse of dimensionality (see Chapter 6). It is natural to ask whether we can find a reduced dimensionality subspace that still preserves the essential characteristics of the data. That is, we are interested in finding the optimal r-dimensional representation of $\mathbf{D}$, with $r \ll d$. In other words, given a point $\mathbf{x}$, and assuming that the basis vectors have been sorted in decreasing order of importance, we can truncate its linear expansion [Eq. (7.1)] to just r terms, to obtain

$$\mathbf{x}' = a_1\mathbf{u}_1 + a_2\mathbf{u}_2 + \cdots + a_r\mathbf{u}_r = \sum_{i=1}^{r} a_i\mathbf{u}_i \tag{7.4}$$

Here $\mathbf{x}'$ is the projection of $\mathbf{x}$ onto the first r basis vectors, which can be written in matrix notation as follows:

$$\mathbf{x}' = \begin{pmatrix} | & | & & | \\ \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_r \\ | & | & & | \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_r \end{pmatrix} = \mathbf{U}_r\mathbf{a}_r \tag{7.5}$$

where $\mathbf{U}_r$ is the matrix comprising the first r basis vectors, and $\mathbf{a}_r$ is vector comprising the first r coordinates. Further, because $\mathbf{a} = \mathbf{U}^T\mathbf{x}$ from Eq. (7.3), restricting it to the first r terms, we get

$$\mathbf{a}_r = \mathbf{U}_r^T\mathbf{x} \tag{7.6}$$

Plugging this into Eq. (7.5), the projection of $\mathbf{x}$ onto the first r basis vectors can be compactly written as

$$\mathbf{x}' = \mathbf{U}_r\mathbf{U}_r^T\mathbf{x} = \mathbf{P}_r\mathbf{x} \tag{7.7}$$

where $\mathbf{P}_r = \mathbf{U}_r\mathbf{U}_r^T$ is the *orthogonal projection matrix* for the subspace spanned by the first r basis vectors. That is, $\mathbf{P}_r$ is symmetric and $\mathbf{P}_r^2 = \mathbf{P}_r$. This is easy to verify because $\mathbf{P}_r^T = (\mathbf{U}_r\mathbf{U}_r^T)^T = \mathbf{U}_r\mathbf{U}_r^T = \mathbf{P}_r$, and $\mathbf{P}_r^2 = (\mathbf{U}_r\mathbf{U}_r^T)(\mathbf{U}_r\mathbf{U}_r^T) = \mathbf{U}_r\mathbf{U}_r^T = \mathbf{P}_r$, where we use the observation that $\mathbf{U}_r^T\mathbf{U}_r = \mathbf{I}_{r\times r}$, the $r \times r$ identity matrix. The projection matrix $\mathbf{P}_r$ can also be written as the decomposition

$$\mathbf{P}_r = \mathbf{U}_r\mathbf{U}_r^T = \sum_{i=1}^{r}\mathbf{u}_i\mathbf{u}_i^T \tag{7.8}$$

From Eqs. (7.1) and (7.4), the projection of $\mathbf{x}$ onto the remaining dimensions comprises the *error vector*

$$\boldsymbol{\epsilon} = \sum_{i=r+1}^{d} a_i\mathbf{u}_i = \mathbf{x} - \mathbf{x}'$$

It is worth noting that that $\mathbf{x}'$ and $\boldsymbol{\epsilon}$ are orthogonal vectors:

$$\mathbf{x}'^T\boldsymbol{\epsilon} = \sum_{i=1}^{r}\sum_{j=r+1}^{d} a_i a_j \mathbf{u}_i^T\mathbf{u}_j = 0$$

This is a consequence of the basis being orthonormal. In fact, we can make an even stronger statement. The subspace spanned by the first r basis vectors

$$S_r = span\,(\mathbf{u}_1, \ldots, \mathbf{u}_r)$$

and the subspace spanned by the remaining basis vectors

$$S_{d-r} = span\,(\mathbf{u}_{r+1}, \ldots, \mathbf{u}_d)$$

are *orthogonal subspaces*, that is, all pairs of vectors $\mathbf{x} \in S_r$ and $\mathbf{y} \in S_{d-r}$ must be orthogonal. The subspace S_{d-r} is also called the *orthogonal complement* of S_r.

Example 7.2. Continuing Example 7.1, approximating the centered point $\mathbf{x} = (-0.343, -0.754, 0.241)^T$ by using only the first basis vector $\mathbf{u}_1 = (-0.390, 0.089, -0.916)^T$, we have

$$\mathbf{x}' = a_1\mathbf{u}_1 = -0.154\mathbf{u}_1 = \begin{pmatrix} 0.060 \\ -0.014 \\ 0.141 \end{pmatrix}$$

The projection of $\mathbf{x}$ on $\mathbf{u}_1$ could have been obtained directly from the projection matrix

$$\mathbf{P}_1 = \mathbf{u}_1\mathbf{u}_1^T = \begin{pmatrix} -0.390 \\ 0.089 \\ -0.916 \end{pmatrix} \begin{pmatrix} -0.390 & 0.089 & -0.916 \end{pmatrix}$$

$$= \begin{pmatrix} 0.152 & -0.035 & 0.357 \\ -0.035 & 0.008 & -0.082 \\ 0.357 & -0.082 & 0.839 \end{pmatrix}$$

That is

$$\mathbf{x}' = \mathbf{P}_1\mathbf{x} = \begin{pmatrix} 0.060 \\ -0.014 \\ 0.141 \end{pmatrix}$$

The error vector is given as

$$\boldsymbol{\epsilon} = a_2\mathbf{u}_2 + a_3\mathbf{u}_3 = \mathbf{x} - \mathbf{x}' = \begin{pmatrix} -0.40 \\ -0.74 \\ 0.10 \end{pmatrix}$$

One can verify that $\mathbf{x}'$ and $\boldsymbol{\epsilon}$ are orthogonal, i.e.,

$$\mathbf{x}'^T\boldsymbol{\epsilon} = \begin{pmatrix} 0.060 & -0.014 & 0.141 \end{pmatrix} \begin{pmatrix} -0.40 \\ -0.74 \\ 0.10 \end{pmatrix} = 0$$

The goal of dimensionality reduction is to seek an r-dimensional basis that gives the best possible approximation $\mathbf{x}_i'$ over all the points $\mathbf{x}_i \in \mathbf{D}$. Alternatively, we may seek to minimize the error $\boldsymbol{\epsilon}_i = \mathbf{x}_i - \mathbf{x}_i'$ over all the points.

7.2 PRINCIPAL COMPONENT ANALYSIS

Principal Component Analysis (PCA) is a technique that seeks a r-dimensional basis that best captures the variance in the data. The direction with the largest projected variance is called the first principal component. The orthogonal direction that captures the second largest projected variance is called the second principal component, and so on. As we shall see, the direction that maximizes the variance is also the one that minimizes the mean squared error.

7.2.1 Best Line Approximation

We will start with $r = 1$, that is, the one-dimensional subspace or line $\mathbf{u}$ that best approximates $\mathbf{D}$ in terms of the variance of the projected points. This will lead to the general PCA technique for the best $1 \leq r \leq d$ dimensional basis for $\mathbf{D}$.

Without loss of generality, we assume that $\mathbf{u}$ has magnitude $\|\mathbf{u}\|^2 = \mathbf{u}^T\mathbf{u} = 1$; otherwise it is possible to keep on increasing the projected variance by simply

increasing the magnitude of $\mathbf{u}$. We also assume that the data has been centered so that it has mean $\boldsymbol{\mu} = \mathbf{0}$.

The projection of $\mathbf{x}_i$ on the vector $\mathbf{u}$ is given as

$$\mathbf{x}_i' = \left(\frac{\mathbf{u}^T\mathbf{x}_i}{\mathbf{u}^T\mathbf{u}}\right)\mathbf{u} = (\mathbf{u}^T\mathbf{x}_i)\mathbf{u} = a_i\mathbf{u}$$

where the scalar

$$a_i = \mathbf{u}^T\mathbf{x}_i$$

gives the coordinate of $\mathbf{x}_i'$ along $\mathbf{u}$. Note that because the mean point is $\boldsymbol{\mu} = \mathbf{0}$, its coordinate along $\mathbf{u}$ is $\mu_{\mathbf{u}} = 0$.

We have to choose the direction $\mathbf{u}$ such that the variance of the projected points is maximized. The projected variance along $\mathbf{u}$ is given as

$$\begin{aligned}
\sigma_{\mathbf{u}}^2 &= \frac{1}{n}\sum_{i=1}^{n}(a_i - \mu_{\mathbf{u}})^2 \\
&= \frac{1}{n}\sum_{i=1}^{n}(\mathbf{u}^T\mathbf{x}_i)^2 \\
&= \frac{1}{n}\sum_{i=1}^{n}\mathbf{u}^T\left(\mathbf{x}_i\mathbf{x}_i^T\right)\mathbf{u} \\
&= \mathbf{u}^T\left(\frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i\mathbf{x}_i^T\right)\mathbf{u} \\
&= \mathbf{u}^T\boldsymbol{\Sigma}\mathbf{u}
\end{aligned} \tag{7.9}$$

where $\boldsymbol{\Sigma}$ is the covariance matrix for the centered data $\mathbf{D}$.

To maximize the projected variance, we have to solve a constrained optimization problem, namely to maximize $\sigma_{\mathbf{u}}^2$ subject to the constraint that $\mathbf{u}^T\mathbf{u} = 1$. This can be solved by introducing a Lagrangian multiplier α for the constraint, to obtain the unconstrained maximization problem

$$\max_{\mathbf{u}} J(\mathbf{u}) = \mathbf{u}^T\boldsymbol{\Sigma}\mathbf{u} - \alpha(\mathbf{u}^T\mathbf{u} - 1) \tag{7.10}$$

Setting the derivative of $J(\mathbf{u})$ with respect to $\mathbf{u}$ to the zero vector, we obtain

$$\begin{aligned}
\frac{\partial}{\partial\mathbf{u}}J(\mathbf{u}) &= \mathbf{0} \\
\frac{\partial}{\partial\mathbf{u}}\left(\mathbf{u}^T\boldsymbol{\Sigma}\mathbf{u} - \alpha(\mathbf{u}^T\mathbf{u} - 1)\right) &= \mathbf{0} \\
2\boldsymbol{\Sigma}\mathbf{u} - 2\alpha\mathbf{u} &= \mathbf{0} \\
\boldsymbol{\Sigma}\mathbf{u} &= \alpha\mathbf{u}
\end{aligned} \tag{7.11}$$

This implies that α is an eigenvalue of the covariance matrix $\boldsymbol{\Sigma}$, with the associated eigenvector $\mathbf{u}$. Further, taking the dot product with $\mathbf{u}$ on both sides of Eq. (7.11) yields

$$\mathbf{u}^T\boldsymbol{\Sigma}\mathbf{u} = \mathbf{u}^T\alpha\mathbf{u}$$

From Eq. (7.9), we then have

$$\sigma_{\mathbf{u}}^2 = \alpha \mathbf{u}^T \mathbf{u}$$
$$\text{or } \sigma_{\mathbf{u}}^2 = \alpha \tag{7.12}$$

To maximize the projected variance $\sigma_{\mathbf{u}}^2$, we should thus choose the largest eigenvalue of $\mathbf{\Sigma}$. In other words, the dominant eigenvector $\mathbf{u}_1$ specifies the direction of most variance, also called the *first principal component*, that is, $\mathbf{u} = \mathbf{u}_1$. Further, the largest eigenvalue λ_1 specifies the projected variance, that is, $\sigma_{\mathbf{u}}^2 = \alpha = \lambda_1$.

Minimum Squared Error Approach

We now show that the direction that maximizes the projected variance is also the one that minimizes the average squared error. As before, assume that the dataset $\mathbf{D}$ has been centered by subtracting the mean from each point. For a point $\mathbf{x}_i \in \mathbf{D}$, let $\mathbf{x}_i'$ denote its projection along the direction $\mathbf{u}$, and let $\boldsymbol{\epsilon}_i = \mathbf{x}_i - \mathbf{x}_i'$ denote the error vector. The mean squared error (*MSE*) optimization condition is defined as

$$MSE(\mathbf{u}) = \frac{1}{n}\sum_{i=1}^{n} \|\boldsymbol{\epsilon}_i\|^2 \tag{7.13}$$
$$= \frac{1}{n}\sum_{i=1}^{n} \|\mathbf{x}_i - \mathbf{x}_i'\|^2$$
$$= \frac{1}{n}\sum_{i=1}^{n} (\mathbf{x}_i - \mathbf{x}_i')^T(\mathbf{x}_i - \mathbf{x}_i')$$
$$= \frac{1}{n}\sum_{i=1}^{n} \left(\|\mathbf{x}_i\|^2 - 2\mathbf{x}_i^T\mathbf{x}_i' + (\mathbf{x}_i')^T\mathbf{x}_i' \right) \tag{7.14}$$

Noting that $\mathbf{x}_i' = (\mathbf{u}^T\mathbf{x}_i)\mathbf{u}$, we have

$$= \frac{1}{n}\sum_{i=1}^{n} \left(\|\mathbf{x}_i\|^2 - 2\mathbf{x}_i^T(\mathbf{u}^T\mathbf{x}_i)\mathbf{u} + \left((\mathbf{u}^T\mathbf{x}_i)\mathbf{u}\right)^T(\mathbf{u}^T\mathbf{x}_i)\mathbf{u} \right)$$
$$= \frac{1}{n}\sum_{i=1}^{n} \left(\|\mathbf{x}_i\|^2 - 2(\mathbf{u}^T\mathbf{x}_i)(\mathbf{x}_i^T\mathbf{u}) + (\mathbf{u}^T\mathbf{x}_i)(\mathbf{x}_i^T\mathbf{u})\mathbf{u}^T\mathbf{u} \right)$$
$$= \frac{1}{n}\sum_{i=1}^{n} \left(\|\mathbf{x}_i\|^2 - (\mathbf{u}^T\mathbf{x}_i)(\mathbf{x}_i^T\mathbf{u}) \right)$$
$$= \frac{1}{n}\sum_{i=1}^{n} \|\mathbf{x}_i\|^2 - \frac{1}{n}\sum_{i=1}^{n} \mathbf{u}^T(\mathbf{x}_i\mathbf{x}_i^T)\mathbf{u}$$
$$= \frac{1}{n}\sum_{i=1}^{n} \|\mathbf{x}_i\|^2 - \mathbf{u}^T\left(\frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i\mathbf{x}_i^T\right)\mathbf{u}$$
$$= \sum_{i=1}^{n} \frac{\|\mathbf{x}_i\|^2}{n} - \mathbf{u}^T\mathbf{\Sigma}\mathbf{u} \tag{7.15}$$

Note that by Eq. (1.4) the total variance of the centered data (i.e., with $\boldsymbol{\mu} = \mathbf{0}$) is given as

$$var(\mathbf{D}) = \frac{1}{n}\sum_{i=1}^{n} \|\mathbf{x}_i - \mathbf{0}\|^2 = \frac{1}{n}\sum_{i=1}^{n} \|\mathbf{x}_i\|^2$$

Further, by Eq. (2.28), we have

$$var(\mathbf{D}) = tr(\boldsymbol{\Sigma}) = \sum_{i=1}^{d} \sigma_i^2$$

Thus, we may rewrite Eq. (7.15) as

$$MSE(\mathbf{u}) = var(\mathbf{D}) - \mathbf{u}^T\boldsymbol{\Sigma}\mathbf{u} = \sum_{i=1}^{d} \sigma_i^2 - \mathbf{u}^T\boldsymbol{\Sigma}\mathbf{u}$$

Because the first term, $var(\mathbf{D})$, is a constant for a given dataset $\mathbf{D}$, the vector $\mathbf{u}$ that minimizes $MSE(\mathbf{u})$ is thus the same one that maximizes the second term, the projected variance $\mathbf{u}^T\boldsymbol{\Sigma}\mathbf{u}$. Because we know that $\mathbf{u}_1$, the dominant eigenvector of $\boldsymbol{\Sigma}$, maximizes the projected variance, we have

$$MSE(\mathbf{u}_1) = var(\mathbf{D}) - \mathbf{u}_1^T\boldsymbol{\Sigma}\mathbf{u}_1 = var(\mathbf{D}) - \mathbf{u}_1^T\lambda_1\mathbf{u}_1 = var(\mathbf{D}) - \lambda_1 \tag{7.16}$$

Thus, the principal component $\mathbf{u}_1$, which is the direction that maximizes the projected variance, is also the direction that minimizes the mean squared error.

Example 7.3. Figure 7.2 shows the first principal component, that is, the best one-dimensional approximation, for the three dimensional Iris dataset shown in Figure 7.1a. The covariance matrix for this dataset is given as

$$\boldsymbol{\Sigma} = \begin{pmatrix} 0.681 & -0.039 & 1.265 \\ -0.039 & 0.187 & -0.320 \\ 1.265 & -0.320 & 3.092 \end{pmatrix}$$

The variance values σ_i^2 for each of the original dimensions are given along the main diagonal of $\boldsymbol{\Sigma}$. For example, $\sigma_1^2 = 0.681$, $\sigma_2^2 = 0.187$, and $\sigma_3^2 = 3.092$. The largest eigenvalue of $\boldsymbol{\Sigma}$ is $\lambda_1 = 3.662$, and the corresponding dominant eigenvector is $\mathbf{u}_1 = (-0.390, 0.089, -0.916)^T$. The unit vector $\mathbf{u}_1$ thus maximizes the projected variance, which is given as $J(\mathbf{u}_1) = \alpha = \lambda_1 = 3.662$. Figure 7.2 plots the principal component $\mathbf{u}_1$. It also shows the error vectors $\boldsymbol{\epsilon}_i$, as thin gray line segments.

The total variance of the data is given as

$$var(\mathbf{D}) = \frac{1}{n}\sum_{i=1}^{n} \|\mathbf{x}\|^2 = \frac{1}{150} \cdot 594.04 = 3.96$$

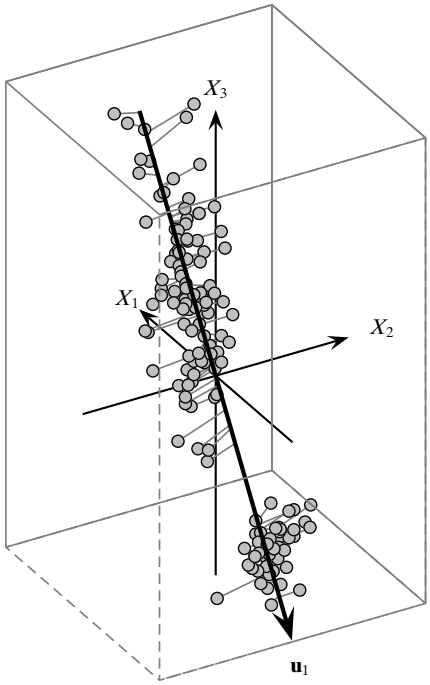


Figure 7.2. Best one-dimensional or line approximation.

We can also directly obtain the total variance as the trace of the covariance matrix:

$$var(\mathbf{D}) = tr(\mathbf{\Sigma}) = \sigma_1^2 + \sigma_2^2 + \sigma_3^2 = 0.681 + 0.187 + 3.092 = 3.96$$

Thus, using Eq. (7.16), the minimum value of the mean squared error is given as

$$MSE(\mathbf{u}_1) = var(\mathbf{D}) - \lambda_1 = 3.96 - 3.662 = 0.298$$

7.2.2 Best 2-dimensional Approximation

We are now interested in the best two-dimensional approximation to $\mathbf{D}$. As before, assume that $\mathbf{D}$ has already been centered, so that $\boldsymbol{\mu} = \mathbf{0}$. We already computed the direction with the most variance, namely $\mathbf{u}_1$, which is the eigenvector corresponding to the largest eigenvalue λ_1 of $\mathbf{\Sigma}$. We now want to find another direction $\mathbf{v}$, which also maximizes the projected variance, but is orthogonal to $\mathbf{u}_1$. According to Eq. (7.9) the projected variance along $\mathbf{v}$ is given as

$$\sigma_{\mathbf{v}}^2 = \mathbf{v}^T \mathbf{\Sigma} \mathbf{v}$$

We further require that $\mathbf{v}$ be a unit vector orthogonal to $\mathbf{u}_1$, that is,

$$\mathbf{v}^T \mathbf{u}_1 = 0$$
$$\mathbf{v}^T \mathbf{v} = 1$$

The optimization condition then becomes

$$\max_{\mathbf{v}} J(\mathbf{v}) = \mathbf{v}^T\boldsymbol{\Sigma}\mathbf{v} - \alpha(\mathbf{v}^T\mathbf{v} - 1) - \beta(\mathbf{v}^T\mathbf{u}_1 - 0) \tag{7.17}$$

Taking the derivative of $J(\mathbf{v})$ with respect to $\mathbf{v}$, and setting it to the zero vector, gives

$$2\boldsymbol{\Sigma}\mathbf{v} - 2\alpha\mathbf{v} - \beta\mathbf{u}_1 = \mathbf{0} \tag{7.18}$$

If we multiply on the left by $\mathbf{u}_1^T$ we get

$$\begin{aligned} 2\mathbf{u}_1^T\boldsymbol{\Sigma}\mathbf{v} - 2\alpha\mathbf{u}_1^T\mathbf{v} - \beta\mathbf{u}_1^T\mathbf{u}_1 &= 0 \\ 2\mathbf{v}^T\boldsymbol{\Sigma}\mathbf{u}_1 - \beta &= 0, \text{which implies that} \\ \beta &= 2\mathbf{v}^T\lambda_1\mathbf{u}_1 = 2\lambda_1\mathbf{v}^T\mathbf{u}_1 = 0 \end{aligned}$$

In the derivation above we used the fact that $\mathbf{u}_1^T\boldsymbol{\Sigma}\mathbf{v} = \mathbf{v}^T\boldsymbol{\Sigma}\mathbf{u}_1$, and that $\mathbf{v}$ is orthogonal to $\mathbf{u}_1$. Plugging $\beta = 0$ into Eq. (7.18) gives us

$$\begin{aligned} 2\boldsymbol{\Sigma}\mathbf{v} - 2\alpha\mathbf{v} &= \mathbf{0} \\ \boldsymbol{\Sigma}\mathbf{v} &= \alpha\mathbf{v} \end{aligned}$$

This means that $\mathbf{v}$ is another eigenvector of $\boldsymbol{\Sigma}$. Also, as in Eq. (7.12), we have $\sigma_{\mathbf{v}}^2 = \alpha$. To maximize the variance along $\mathbf{v}$, we should choose $\alpha = \lambda_2$, the second largest eigenvalue of $\boldsymbol{\Sigma}$, with the *second principal component* being given by the corresponding eigenvector, that is, $\mathbf{v} = \mathbf{u}_2$.

Total Projected Variance

Let $\mathbf{U}_2$ be the matrix whose columns correspond to the two principal components, given as

$$\mathbf{U}_2 = \begin{pmatrix} | & | \\ \mathbf{u}_1 & \mathbf{u}_2 \\ | & | \end{pmatrix}$$

Given the point $\mathbf{x}_i \in \mathbf{D}$ its coordinates in the two-dimensional subspace spanned by $\mathbf{u}_1$ and $\mathbf{u}_2$ can be computed via Eq. (7.6), as follows:

$$\mathbf{a}_i = \mathbf{U}_2^T\mathbf{x}_i$$

Assume that each point $\mathbf{x}_i \in \mathbb{R}^d$ in $\mathbf{D}$ has been projected to obtain its coordinates $\mathbf{a}_i \in \mathbb{R}^2$, yielding the new dataset $\mathbf{A}$. Further, because $\mathbf{D}$ is assumed to be centered, with $\boldsymbol{\mu} = \mathbf{0}$, the coordinates of the projected mean are also zero because $\mathbf{U}_2^T\boldsymbol{\mu} = \mathbf{U}_2^T\mathbf{0} = \mathbf{0}$.

The total variance for $\mathbf{A}$ is given as

$$\begin{aligned} var(\mathbf{A}) &= \frac{1}{n}\sum_{i=1}^{n} \|\mathbf{a}_i - \mathbf{0}\|^2 \\ &= \frac{1}{n}\sum_{i=1}^{n} \left(\mathbf{U}_2^T\mathbf{x}_i\right)^T \left(\mathbf{U}_2^T\mathbf{x}_i\right) \\ &= \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i^T \left(\mathbf{U}_2\mathbf{U}_2^T\right)\mathbf{x}_i \\ &= \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i^T \mathbf{P}_2 \mathbf{x}_i \end{aligned} \tag{7.19}$$

where $\mathbf{P}_2$ is the orthogonal projection matrix [Eq. (7.8)] given as

$$\mathbf{P}_2 = \mathbf{U}_2\mathbf{U}_2^T = \mathbf{u}_1\mathbf{u}_1^T + \mathbf{u}_2\mathbf{u}_2^T$$

Substituting this into Eq. (7.19), the projected total variance is given as

$$\begin{aligned} var(\mathbf{A}) &= \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i^T \mathbf{P}_2 \mathbf{x}_i \qquad (7.20) \\ &= \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i^T \left(\mathbf{u}_1\mathbf{u}_1^T + \mathbf{u}_2\mathbf{u}_2^T\right)\mathbf{x}_i \\ &= \frac{1}{n}\sum_{i=1}^{n} (\mathbf{u}_1^T\mathbf{x}_i)(\mathbf{x}_i^T\mathbf{u}_1) + \frac{1}{n}\sum_{i=1}^{n} (\mathbf{u}_2^T\mathbf{x}_i)(\mathbf{x}_i^T\mathbf{u}_2) \\ &= \mathbf{u}_1^T\boldsymbol{\Sigma}\mathbf{u}_1 + \mathbf{u}_2^T\boldsymbol{\Sigma}\mathbf{u}_2 \qquad (7.21) \end{aligned}$$

Because $\mathbf{u}_1$ and $\mathbf{u}_2$ are eigenvectors of $\boldsymbol{\Sigma}$, we have $\boldsymbol{\Sigma}\mathbf{u}_1 = \lambda_1\mathbf{u}_1$ and $\boldsymbol{\Sigma}\mathbf{u}_2 = \lambda_2\mathbf{u}_2$, so that

$$var(\mathbf{A}) = \mathbf{u}_1^T\boldsymbol{\Sigma}\mathbf{u}_1 + \mathbf{u}_2^T\boldsymbol{\Sigma}\mathbf{u}_2 = \mathbf{u}_1^T\lambda_1\mathbf{u}_1 + \mathbf{u}_2^T\lambda_2\mathbf{u}_2 = \lambda_1 + \lambda_2 \tag{7.22}$$

Thus, the sum of the eigenvalues is the total variance of the projected points, and the first two principal components maximize this variance.

Mean Squared Error

We now show that the first two principal components also minimize the mean square error objective. The mean square error objective is given as

$$\begin{aligned} MSE &= \frac{1}{n}\sum_{i=1}^{n} \left\|\mathbf{x}_i - \mathbf{x}_i'\right\|^2 \\ &= \frac{1}{n}\sum_{i=1}^{n} \left(\|\mathbf{x}_i\|^2 - 2\mathbf{x}_i^T\mathbf{x}_i' + (\mathbf{x}_i')^T\mathbf{x}_i' \right), \text{ using Eq. (7.14)} \\ &= var(\mathbf{D}) + \frac{1}{n}\sum_{i=1}^{n} \left(-2\mathbf{x}_i^T\mathbf{P}_2\mathbf{x}_i + (\mathbf{P}_2\mathbf{x}_i)^T\mathbf{P}_2\mathbf{x}_i\right), \text{ using Eq. (7.7) that } \mathbf{x}_i' = \mathbf{P}_2\mathbf{x}_i \end{aligned}$$

$$= var(\mathbf{D}) - \frac{1}{n}\sum_{i=1}^{n}\left(\mathbf{x}_i^T\mathbf{P}_2\mathbf{x}_i\right)$$

$$= var(\mathbf{D}) - var(\mathbf{A}), \text{ using Eq. (7.20)} \tag{7.23}$$

Thus, the MSE objective is minimized precisely when the total projected variance $var(\mathbf{A})$ is maximized. From Eq. (7.22), we have

$$MSE = var(\mathbf{D}) - \lambda_1 - \lambda_2$$

Example 7.4. For the Iris dataset from Example 7.1, the two largest eigenvalues are $\lambda_1 = 3.662$, and $\lambda_2 = 0.239$, with the corresponding eigenvectors:

$$\mathbf{u}_1 = \begin{pmatrix} -0.390 \\ 0.089 \\ -0.916 \end{pmatrix} \qquad \mathbf{u}_2 = \begin{pmatrix} -0.639 \\ -0.742 \\ 0.200 \end{pmatrix}$$

The projection matrix is given as

$$\mathbf{P}_2 = \mathbf{U}_2\mathbf{U}_2^T = \begin{pmatrix} | & | \\ \mathbf{u}_1 & \mathbf{u}_2 \\ | & | \end{pmatrix}\begin{pmatrix} — & \mathbf{u}_1^T & — \\ — & \mathbf{u}_2^T & — \end{pmatrix} = \mathbf{u}_1\mathbf{u}_1^T + \mathbf{u}_2\mathbf{u}_2^T$$

$$= \begin{pmatrix} 0.152 & -0.035 & 0.357 \\ -0.035 & 0.008 & -0.082 \\ 0.357 & -0.082 & 0.839 \end{pmatrix} + \begin{pmatrix} 0.408 & 0.474 & -0.128 \\ 0.474 & 0.551 & -0.148 \\ -0.128 & -0.148 & 0.04 \end{pmatrix}$$

$$= \begin{pmatrix} 0.560 & 0.439 & 0.229 \\ 0.439 & 0.558 & -0.230 \\ 0.229 & -0.230 & 0.879 \end{pmatrix}$$

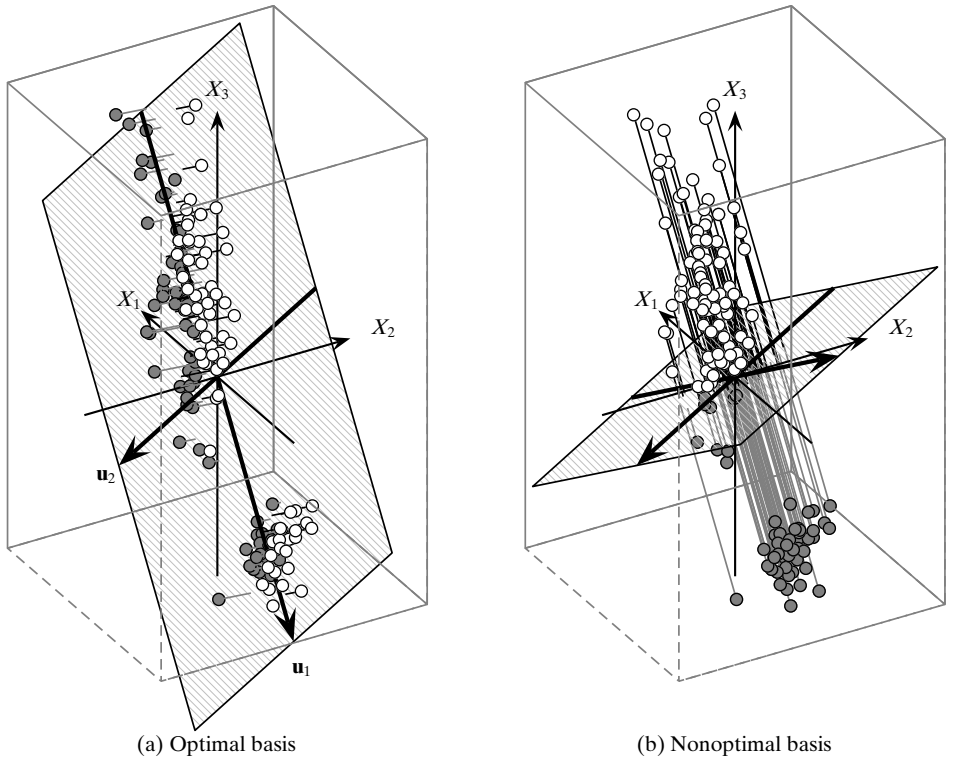
Thus, each point $\mathbf{x}_i$ can be approximated by its projection onto the first two principal components $\mathbf{x}_i' = \mathbf{P}_2\mathbf{x}_i$. Figure 7.3a plots this optimal 2-dimensional subspace spanned by $\mathbf{u}_1$ and $\mathbf{u}_2$. The error vector $\boldsymbol{\epsilon}_i$ for each point is shown as a thin line segment. The gray points are behind the 2-dimensional subspace, whereas the white points are in front of it. The total variance captured by the subspace is given as

$$\lambda_1 + \lambda_2 = 3.662 + 0.239 = 3.901$$

The mean squared error is given as

$$MSE = var(\mathbf{D}) - \lambda_1 - \lambda_2 = 3.96 - 3.662 - 0.239 = 0.059$$

Figure 7.3b plots a nonoptimal 2-dimensional subspace. As one can see the optimal subspace maximizes the variance, and minimizes the squared error, whereas the nonoptimal subspace captures less variance, and has a high mean squared error value, which can be pictorially seen from the lengths of the error vectors (line segments). In fact, this is the worst possible 2-dimensional subspace; its MSE is 3.662.



(a) Optimal basis (b) Nonoptimal basis

Figure 7.3. Best two-dimensional approximation.

7.2.3 Best *r*-dimensional Approximation

We are now interested in the best r-dimensional approximation to $\mathbf{D}$, where $2 < r \leq d$. Assume that we have already computed the first $j-1$ principal components or eigenvectors, $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_{j-1}$, corresponding to the $j-1$ largest eigenvalues of $\boldsymbol{\Sigma}$, for $1 \leq j \leq r$. To compute the jth new basis vector $\mathbf{v}$, we have to ensure that it is normalized to unit length, that is, $\mathbf{v}^T\mathbf{v} = 1$, and is orthogonal to all previous components $\mathbf{u}_i$, i.e., $\mathbf{u}_i^T\mathbf{v} = 0$, for $1 \leq i < j$. As before, the projected variance along $\mathbf{v}$ is given as

$$\sigma_{\mathbf{v}}^2 = \mathbf{v}^T\boldsymbol{\Sigma}\mathbf{v}$$

Combined with the constraints on $\mathbf{v}$, this leads to the following maximization problem with Lagrange multipliers:

$$\max_{\mathbf{v}} J(\mathbf{v}) = \mathbf{v}^T\boldsymbol{\Sigma}\mathbf{v} - \alpha(\mathbf{v}^T\mathbf{v} - 1) - \sum_{i=1}^{j-1} \beta_i(\mathbf{u}_i^T\mathbf{v} - 0)$$

Taking the derivative of $J(\mathbf{v})$ with respect to $\mathbf{v}$ and setting it to the zero vector gives

$$2\boldsymbol{\Sigma}\mathbf{v} - 2\alpha\mathbf{v} - \sum_{i=1}^{j-1} \beta_i\mathbf{u}_i = \mathbf{0} \tag{7.24}$$

If we multiply on the left by $\mathbf{u}_k^T$, for $1 \leq k < j$, we get

$$2\mathbf{u}_k^T\boldsymbol{\Sigma}\mathbf{v} - 2\alpha\mathbf{u}_k^T\mathbf{v} - \beta_k\mathbf{u}_k^T\mathbf{u}_k - \sum_{\substack{i=1 \\ i\neq k}}^{j-1}\beta_i\mathbf{u}_k^T\mathbf{u}_i = 0$$

$$2\mathbf{v}^T\boldsymbol{\Sigma}\mathbf{u}_k - \beta_k = 0$$

$$\beta_k = 2\mathbf{v}^T\lambda_k\mathbf{u}_k = 2\lambda_k\mathbf{v}^T\mathbf{u}_k = 0$$

where we used the fact that $\boldsymbol{\Sigma}\mathbf{u}_k = \lambda_k\mathbf{u}_k$, as $\mathbf{u}_k$ is the eigenvector corresponding to the kth largest eigenvalue λ_k of $\boldsymbol{\Sigma}$. Thus, we find that $\beta_i = 0$ for all $i < j$ in Eq. (7.24), which implies that

$$\boldsymbol{\Sigma}\mathbf{v} = \alpha\mathbf{v}$$

To maximize the variance along $\mathbf{v}$, we set $\alpha = \lambda_j$, the jth largest eigenvalue of $\boldsymbol{\Sigma}$, with $\mathbf{v} = \mathbf{u}_j$ giving the jth principal component.

In summary, to find the best r-dimensional approximation to $\mathbf{D}$, we compute the eigenvalues of $\boldsymbol{\Sigma}$. Because $\boldsymbol{\Sigma}$ is positive semidefinite, its eigenvalues must all be non-negative, and we can thus sort them in decreasing order as follows:

$$\lambda_1 \geq \lambda_2 \geq \cdots \lambda_r \geq \lambda_{r+1} \cdots \geq \lambda_d \geq 0$$

We then select the r largest eigenvalues, and their corresponding eigenvectors to form the best r-dimensional approximation.

Total Projected Variance

Let $\mathbf{U}_r$ be the r-dimensional basis vector matrix

$$\mathbf{U}_r = \begin{pmatrix} | & | & & | \\ \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_r \\ | & | & & | \end{pmatrix}$$

with the projection matrix given as

$$\mathbf{P}_r = \mathbf{U}_r\mathbf{U}_r^T = \sum_{i=1}^{r}\mathbf{u}_i\mathbf{u}_i^T$$

Let $\mathbf{A}$ denote the dataset formed by the coordinates of the projected points in the r-dimensional subspace, that is, $\mathbf{a}_i = \mathbf{U}_r^T\mathbf{x}_i$, and let $\mathbf{x}_i' = \mathbf{P}_r\mathbf{x}_i$ denote the projected point in the original d-dimensional space. Following the derivation for Eqs. (7.19), (7.21), and (7.22), the projected variance is given as

$$var(\mathbf{A}) = \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i^T\mathbf{P}_r\mathbf{x}_i = \sum_{i=1}^{r}\mathbf{u}_i^T\boldsymbol{\Sigma}\mathbf{u}_i = \sum_{i=1}^{r}\lambda_i$$

Thus, the total projected variance is simply the sum of the r largest eigenvalues of $\boldsymbol{\Sigma}$.

Mean Squared Error

Based on the derivation for Eq. (7.23), the mean squared error objective in r dimensions can be written as

$$\begin{aligned} MSE &= \frac{1}{n}\sum_{i=1}^{n}\left\|\mathbf{x}_i - \mathbf{x}_i'\right\|^2 \\ &= var(\mathbf{D}) - var(\mathbf{A}) \\ &= var(\mathbf{D}) - \sum_{i=1}^{r}\mathbf{u}_i^T\boldsymbol{\Sigma}\mathbf{u}_i \\ &= var(\mathbf{D}) - \sum_{i=1}^{r}\lambda_i \end{aligned}$$

The first r-principal components maximize the projected variance $var(\mathbf{A})$, and thus they also minimize the MSE.

Total Variance

Note that the total variance of $\mathbf{D}$ is invariant to a change in basis vectors. Therefore, we have the following identity:

$$var(\mathbf{D}) = \sum_{i=1}^{d}\sigma_i^2 = \sum_{i=1}^{d}\lambda_i$$

Choosing the Dimensionality

Often we may not know how many dimensions, r, to use for a good approximation. One criteria for choosing r is to compute the fraction of the total variance captured by the first r principal components, computed as

$$f(r) = \frac{\lambda_1 + \lambda_2 + \cdots + \lambda_r}{\lambda_1 + \lambda_2 + \cdots + \lambda_d} = \frac{\sum_{i=1}^{r}\lambda_i}{\sum_{i=1}^{d}\lambda_i} = \frac{\sum_{i=1}^{r}\lambda_i}{var(\mathbf{D})} \tag{7.25}$$

Given a certain desired variance threshold, say α, starting from the first principal component, we keep on adding additional components, and stop at the smallest value r, for which $f(r) \geq \alpha$. In other words, we select the fewest number of dimensions such that the subspace spanned by those r dimensions captures at least α fraction of the total variance. In practice, α is usually set to 0.9 or higher, so that the reduced dataset captures at least 90% of the total variance.

Algorithm 7.1 gives the pseudo-code for the principal component analysis algorithm. Given the input data $\mathbf{D} \in \mathbb{R}^{n\times d}$, it first centers it by subtracting the mean from each point. Next, it computes the eigenvectors and eigenvalues of the covariance matrix $\boldsymbol{\Sigma}$. Given the desired variance threshold α, it selects the smallest set of dimensions r that capture at least α fraction of the total variance. Finally, it computes the coordinates of each point in the new r-dimensional principal component subspace, to yield the new data matrix $\mathbf{A} \in \mathbb{R}^{n\times r}$.

ALGORITHM 7.1. Principal Component Analysis

PCA ($\mathbf{D}, \alpha$):

1 $\boldsymbol{\mu} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i$ // compute mean
2 $\mathbf{Z} = \mathbf{D} - \mathbf{1} \cdot \boldsymbol{\mu}^T$ // center the data
3 $\boldsymbol{\Sigma} = \frac{1}{n}\left(\mathbf{Z}^T\mathbf{Z}\right)$ // compute covariance matrix
4 $(\lambda_1, \lambda_2, \ldots, \lambda_d) = \text{eigenvalues}(\boldsymbol{\Sigma})$ // compute eigenvalues
5 $\mathbf{U} = \begin{pmatrix}\mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_d\end{pmatrix} = \text{eigenvectors}(\boldsymbol{\Sigma})$ // compute eigenvectors
6 $f(r) = \frac{\sum_{i=1}^{r}\lambda_i}{\sum_{i=1}^{d}\lambda_i}$, for all $r = 1, 2, \ldots, d$ // fraction of total variance
7 Choose smallest r so that $f(r) \geq \alpha$ // choose dimensionality
8 $\mathbf{U}_r = \begin{pmatrix}\mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_r\end{pmatrix}$ // reduced basis
9 $\mathbf{A} = \{\mathbf{a}_i \mid \mathbf{a}_i = \mathbf{U}_r^T\mathbf{x}_i, \text{for } i = 1, \ldots, n\}$ // reduced dimensionality data

Example 7.5. Given the 3-dimensional Iris dataset in Figure 7.1a, its covariance matrix is

$$\boldsymbol{\Sigma} = \begin{pmatrix} 0.681 & -0.039 & 1.265 \\ -0.039 & 0.187 & -0.320 \\ 1.265 & -0.32 & 3.092 \end{pmatrix}$$

The eigenvalues and eigenvectors of $\boldsymbol{\Sigma}$ are given as

$$\lambda_1 = 3.662 \qquad \lambda_2 = 0.239 \qquad \lambda_3 = 0.059$$

$$\mathbf{u}_1 = \begin{pmatrix} -0.390 \\ 0.089 \\ -0.916 \end{pmatrix} \qquad \mathbf{u}_2 = \begin{pmatrix} -0.639 \\ -0.742 \\ 0.200 \end{pmatrix} \qquad \mathbf{u}_3 = \begin{pmatrix} -0.663 \\ 0.664 \\ 0.346 \end{pmatrix}$$

The total variance is therefore $\lambda_1 + \lambda_2 + \lambda_3 = 3.662 + 0.239 + 0.059 = 3.96$. The optimal 3-dimensional basis is shown in Figure 7.1b.

To find a lower dimensional approximation, let $\alpha = 0.95$. The fraction of total variance for different values of r is given as

r	1	2	3
$f(r)$	0.925	0.985	1.0

For example, for $r = 1$, the fraction of total variance is given as $f(1) = \frac{3.662}{3.96} = 0.925$. Thus, we need at least $r = 2$ dimensions to capture 95% of the total variance. This optimal 2-dimensional subspace is shown as the shaded plane in Figure 7.3a. The reduced dimensionality dataset $\mathbf{A}$ is shown in Figure 7.4. It consists of the point coordinates $\mathbf{a}_i = \mathbf{U}_2^T\mathbf{x}_i$ in the new 2-dimensional principal components basis comprising $\mathbf{u}_1$ and $\mathbf{u}_2$.

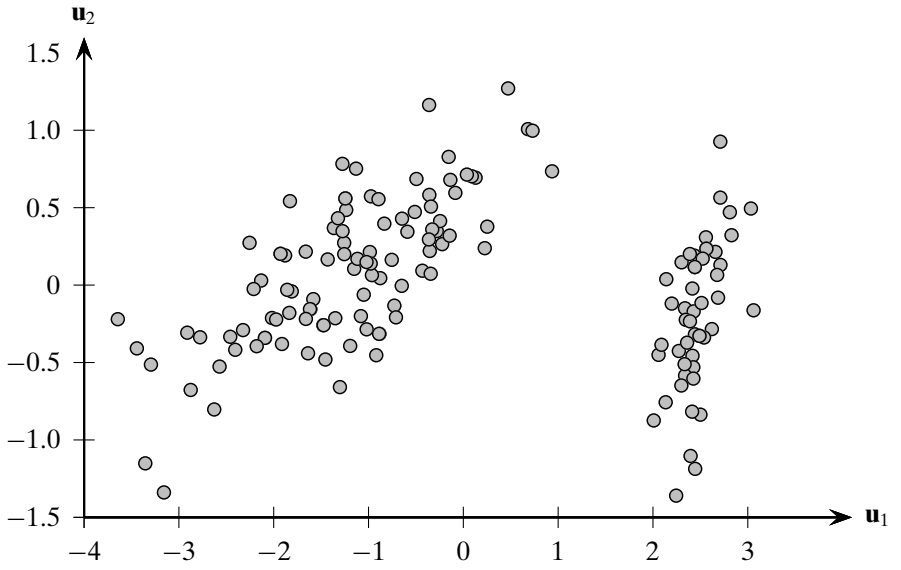


Figure 7.4. Reduced dimensionality dataset: Iris principal components.

7.2.4 Geometry of PCA

Geometrically, when $r = d$, PCA corresponds to a orthogonal change of basis, so that the total variance is captured by the sum of the variances along each of the principal directions $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_d$, and further, all covariances are zero. This can be seen by looking at the collective action of the full set of principal components, which can be arranged in the $d \times d$ orthogonal matrix

$$\mathbf{U} = \begin{pmatrix} | & | & & | \\ \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_d \\ | & | & & | \end{pmatrix}$$

with $\mathbf{U}^{-1} = \mathbf{U}^T$.

Each principal component $\mathbf{u}_i$ corresponds to an eigenvector of the covariance matrix $\mathbf{\Sigma}$, that is,

$$\mathbf{\Sigma}\mathbf{u}_i = \lambda_i \mathbf{u}_i \text{ for all } 1 \leq i \leq d$$

which can be written compactly in matrix notation as follows:

$$\mathbf{\Sigma}\begin{pmatrix} | & | & & | \\ \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_d \\ | & | & & | \end{pmatrix} = \begin{pmatrix} | & | & & | \\ \lambda_1\mathbf{u}_1 & \lambda_2\mathbf{u}_2 & \cdots & \lambda_d\mathbf{u}_d \\ | & | & & | \end{pmatrix}$$

$$\mathbf{\Sigma}\mathbf{U} = \mathbf{U}\begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_d \end{pmatrix}$$

$$\mathbf{\Sigma}\mathbf{U} = \mathbf{U}\mathbf{\Lambda} \tag{7.26}$$

If we multiply Eq. (7.26) on the left by $\mathbf{U}^{-1} = \mathbf{U}^T$ we obtain

$$\mathbf{U}^T\boldsymbol{\Sigma}\mathbf{U} = \mathbf{U}^T\mathbf{U}\boldsymbol{\Lambda} = \boldsymbol{\Lambda} = \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_d \end{pmatrix}$$

This means that if we change the basis to $\mathbf{U}$, we change the covariance matrix $\boldsymbol{\Sigma}$ to a similar matrix $\boldsymbol{\Lambda}$, which in fact is the covariance matrix in the new basis. The fact that $\boldsymbol{\Lambda}$ is diagonal confirms that after the change of basis, all of the covariances vanish, and we are left with only the variances along each of the principal components, with the variance along each new direction $\mathbf{u}_i$ being given by the corresponding eigenvalue λ_i.

It is worth noting that in the new basis, the equation

$$\mathbf{x}^T\boldsymbol{\Sigma}^{-1}\mathbf{x} = 1 \tag{7.27}$$

defines a d-dimensional ellipsoid (or hyper-ellipse). The eigenvectors $\mathbf{u}_i$ of $\boldsymbol{\Sigma}$, that is, the principal components, are the directions for the principal axes of the ellipsoid. The square roots of the eigenvalues, that is, $\sqrt{\lambda_i}$, give the lengths of the semi-axes.

Multiplying Eq. (7.26) on the right by $\mathbf{U}^{-1} = \mathbf{U}^T$, we have

$$\boldsymbol{\Sigma} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T \tag{7.28}$$

Assuming that $\boldsymbol{\Sigma}$ is invertible or nonsingular, we have

$$\boldsymbol{\Sigma}^{-1} = (\mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T)^{-1} = \left(\mathbf{U}^{-1}\right)^T\boldsymbol{\Lambda}^{-1}\mathbf{U}^{-1} = \mathbf{U}\boldsymbol{\Lambda}^{-1}\mathbf{U}^T$$

where

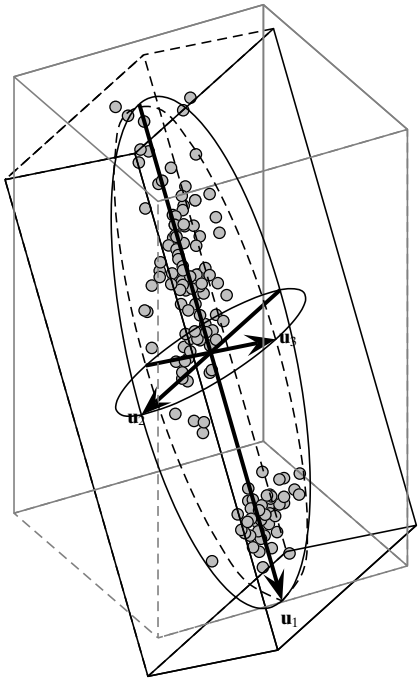
$$\boldsymbol{\Lambda}^{-1} = \begin{pmatrix} \frac{1}{\lambda_1} & 0 & \cdots & 0 \\ 0 & \frac{1}{\lambda_2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{1}{\lambda_d} \end{pmatrix}$$

Substituting $\boldsymbol{\Sigma}^{-1}$ in Eq. (7.27), and using the fact that $\mathbf{x} = \mathbf{U}\mathbf{a}$ from Eq. (7.2), where $\mathbf{a} = (a_1, a_2, \ldots, a_d)^T$ represents the coordinates of $\mathbf{x}$ in the new basis, we get

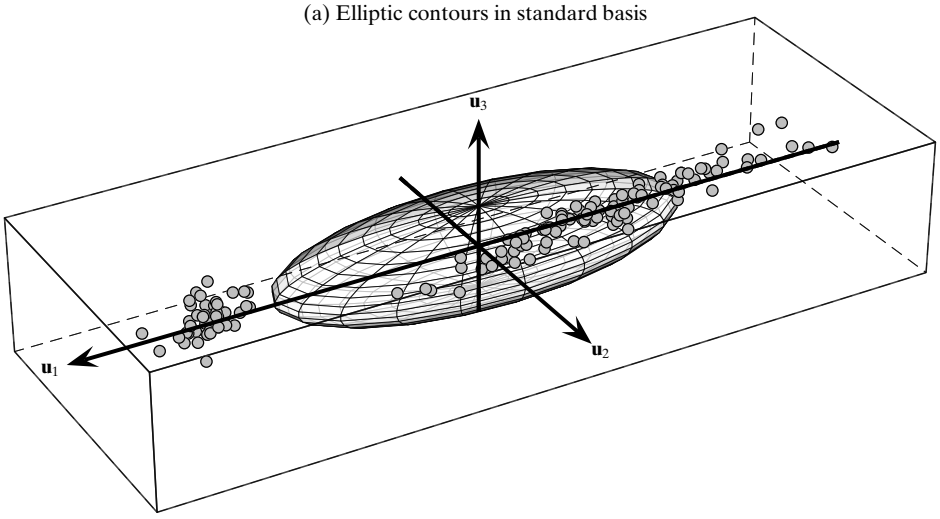
$$\begin{aligned} \mathbf{x}^T\boldsymbol{\Sigma}^{-1}\mathbf{x} &= 1 \\ \left(\mathbf{a}^T\mathbf{U}^T\right)\mathbf{U}\boldsymbol{\Lambda}^{-1}\mathbf{U}^T\left(\mathbf{U}\mathbf{a}\right) &= 1 \\ \mathbf{a}^T\boldsymbol{\Lambda}^{-1}\mathbf{a} &= 1 \\ \sum_{i=1}^{d}\frac{a_i^2}{\lambda_i} &= 1 \end{aligned}$$

which is precisely the equation for an ellipse centered at $\mathbf{0}$, with semi-axes lengths $\sqrt{\lambda_i}$. Thus $\mathbf{x}^T\boldsymbol{\Sigma}^{-1}\mathbf{x} = 1$, or equivalently $\mathbf{a}^T\boldsymbol{\Lambda}^{-1}\mathbf{a} = 1$ in the new principal components basis, defines an ellipsoid in d-dimensions, where the semi-axes lengths equal the standard deviations (squared root of the variance, $\sqrt{\lambda_i}$) along each axis. Likewise, the equation $\mathbf{x}^T\boldsymbol{\Sigma}^{-1}\mathbf{x} = s$, or equivalently $\mathbf{a}^T\boldsymbol{\Lambda}^{-1}\mathbf{a} = s$, for different values of the scalar s, represents concentric ellipsoids.

Example 7.6. Figure 7.5b shows the ellipsoid $\mathbf{x}^T\boldsymbol{\Sigma}^{-1}\mathbf{x} = \mathbf{a}^T\boldsymbol{\Lambda}^{-1}\mathbf{a} = 1$ in the new principal components basis. Each semi-axis length corresponds to the standard deviation $\sqrt{\lambda_i}$ along that axis. Because all pairwise covariances are zero in the principal components basis, the ellipsoid is axis-parallel, that is, each of its axes coincides with a basis vector.



(a) Elliptic contours in standard basis



(b) Axis parallel ellipsoid in principal components basis

Figure 7.5. Iris data: standard and principal components basis in three dimensions.

On the other hand, in the original standard d-dimensional basis for $\mathbf{D}$, the ellipsoid will not be axis-parallel, as shown by the contours of the ellipsoid in Figure 7.5a. Here the semi-axis lengths correspond to half the value range in each direction; the length was chosen so that the ellipsoid encompasses most of the points.

7.3 KERNEL PRINCIPAL COMPONENT ANALYSIS

Principal component analysis can be extended to find nonlinear "directions" in the data using kernel methods. Kernel PCA finds the directions of most variance in the feature space instead of the input space. That is, instead of trying to find linear combinations of the input dimensions, kernel PCA finds linear combinations in the high-dimensional feature space obtained as some nonlinear transformation of the input dimensions. Thus, the linear principal components in the feature space correspond to nonlinear directions in the input space. As we shall see, using the *kernel trick*, all operations can be carried out in terms of the kernel function in input space, without having to transform the data into feature space.

Example 7.7. Consider the nonlinear Iris dataset shown in Figure 7.6, obtained via a nonlinear transformation applied on the centered Iris data. In particular, the `sepal length` (A_1) and `sepal width` attributes (A_2) were transformed as follows:

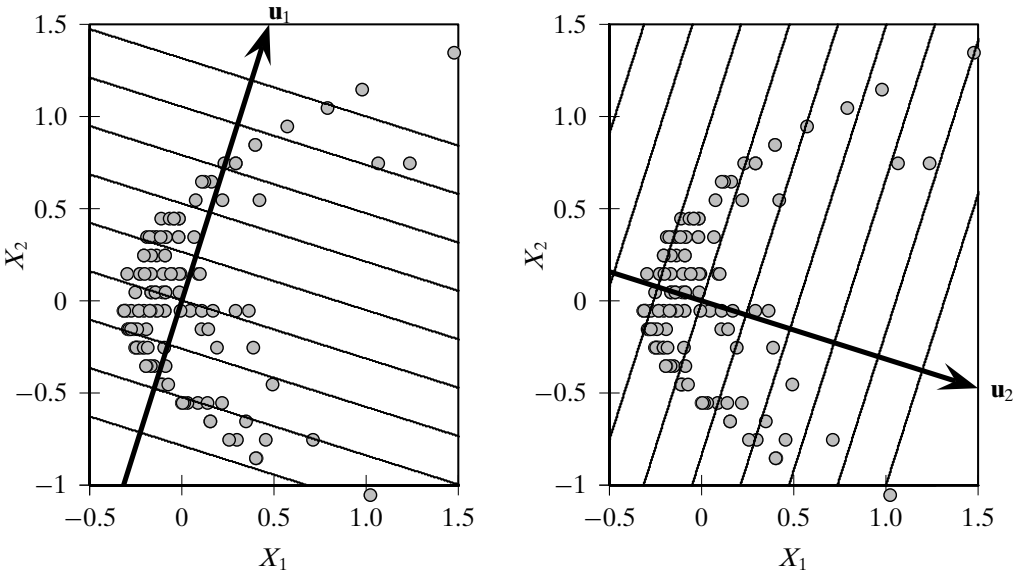
$$X_1 = 0.2A_1^2 + A_2^2 + 0.1A_1A_2$$
$$X_2 = A_2$$

The points show a clear quadratic (nonlinear) relationship between the two variables. Linear PCA yields the following two directions of most variance:

$$\lambda_1 = 0.197 \qquad \lambda_2 = 0.087$$
$$\mathbf{u}_1 = \begin{pmatrix} 0.301 \\ 0.953 \end{pmatrix} \qquad \mathbf{u}_2 = \begin{pmatrix} -0.953 \\ 0.301 \end{pmatrix}$$

These two principal components are illustrated in Figure 7.6. Also shown in the figure are lines of constant projections onto the principal components, that is, the set of all points in the input space that have the same coordinates when projected onto $\mathbf{u}_1$ and $\mathbf{u}_2$, respectively. For instance, the lines of constant projections in Figure 7.6a correspond to the solutions of $\mathbf{u}_1^T\mathbf{x} = s$ for different values of the coordinate s. Figure 7.7 shows the coordinates of each point in the principal components space comprising $\mathbf{u}_1$ and $\mathbf{u}_2$. It is clear from the figures that $\mathbf{u}_1$ and $\mathbf{u}_2$ do not fully capture the nonlinear relationship between X_1 and X_2. We shall see later in this section that kernel PCA is able to capture this dependence better.

Let ϕ correspond to a mapping from the input space to the feature space. Each point in feature space is given as the image $\phi(\mathbf{x}_i)$ of the point $\mathbf{x}_i$ in input space. In the input space, the first principal component captures the direction with the most projected variance; it is the eigenvector corresponding to the largest eigenvalue of the



(a) $\lambda_1 = 0.197$ (b) $\lambda_2 = 0.087$

Figure 7.6. Nonlinear Iris dataset: PCA in input space.

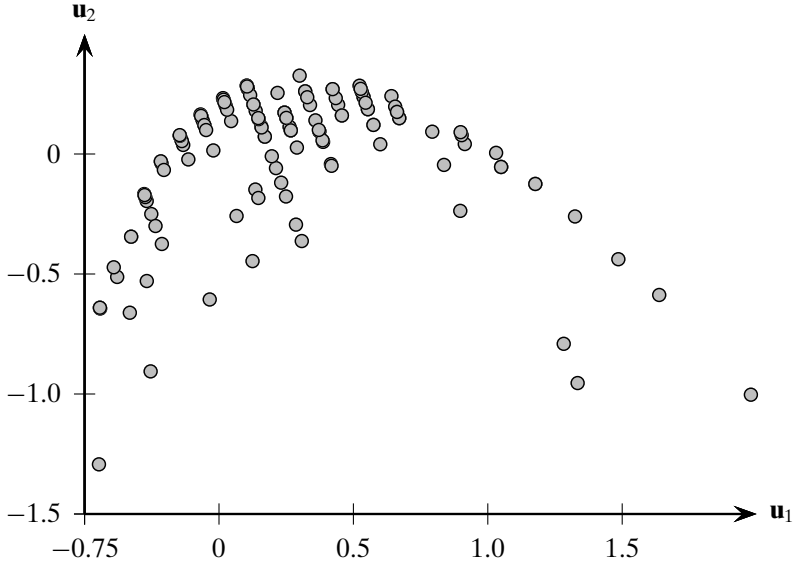


Figure 7.7. Projection onto principal components.

covariance matrix. Likewise, in feature space, we can find the first kernel principal component $\mathbf{u}_1$ (with $\mathbf{u}_1^T\mathbf{u}_1 = 1$), by solving for the eigenvector corresponding to the largest eigenvalue of the covariance matrix in feature space:

$$\boldsymbol{\Sigma}_\phi \mathbf{u}_1 = \lambda_1 \mathbf{u}_1 \tag{7.29}$$

where $\boldsymbol{\Sigma}_\phi$, the covariance matrix in feature space, is given as

$$\boldsymbol{\Sigma}_\phi = \frac{1}{n}\sum_{i=1}^{n}\phi(\mathbf{x}_i)\phi(\mathbf{x}_i)^T \tag{7.30}$$

Here we assume that the points are centered, that is, $\phi(\mathbf{x}_i) = \phi(\mathbf{x}_i) - \boldsymbol{\mu}_\phi$, where $\boldsymbol{\mu}_\phi$ is the mean in feature space.

Plugging in the expansion of $\boldsymbol{\Sigma}_\phi$ from Eq. (7.30) into Eq. (7.29), we get

$$\left(\frac{1}{n}\sum_{i=1}^{n}\phi(\mathbf{x}_i)\phi(\mathbf{x}_i)^T\right)\mathbf{u}_1 = \lambda_1\mathbf{u}_1 \tag{7.31}$$

$$\frac{1}{n}\sum_{i=1}^{n}\phi(\mathbf{x}_i)\left(\phi(\mathbf{x}_i)^T\mathbf{u}_1\right) = \lambda_1\mathbf{u}_1$$

$$\sum_{i=1}^{n}\left(\frac{\phi(\mathbf{x}_i)^T\mathbf{u}_1}{n\,\lambda_1}\right)\phi(\mathbf{x}_i) = \mathbf{u}_1$$

$$\sum_{i=1}^{n}c_i\phi(\mathbf{x}_i) = \mathbf{u}_1 \tag{7.32}$$

where $c_i = \frac{\phi(\mathbf{x}_i)^T\mathbf{u}_1}{n\lambda_1}$ is a scalar value. From Eq. (7.32) we see that the best direction in the feature space, $\mathbf{u}_1$, is just a linear combination of the transformed points, where the scalars c_i show the importance of each point toward the direction of most variance.

We can now substitute Eq. (7.32) back into Eq. (7.31) to get

$$\left(\frac{1}{n}\sum_{i=1}^{n}\phi(\mathbf{x}_i)\phi(\mathbf{x}_i)^T\right)\left(\sum_{j=1}^{n}c_j\phi(\mathbf{x}_j)\right) = \lambda_1\sum_{i=1}^{n}c_i\phi(\mathbf{x}_i)$$

$$\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{n}c_j\phi(\mathbf{x}_i)\phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j) = \lambda_1\sum_{i=1}^{n}c_i\phi(\mathbf{x}_i)$$

$$\sum_{i=1}^{n}\left(\phi(\mathbf{x}_i)\sum_{j=1}^{n}c_j\phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)\right) = n\lambda_1\sum_{i=1}^{n}c_i\phi(\mathbf{x}_i)$$

In the preceding equation, we can replace the dot product in feature space, namely $\phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$, by the corresponding kernel function in input space, namely $K(\mathbf{x}_i, \mathbf{x}_j)$, which yields

$$\sum_{i=1}^{n}\left(\phi(\mathbf{x}_i)\sum_{j=1}^{n}c_jK(\mathbf{x}_i, \mathbf{x}_j)\right) = n\lambda_1\sum_{i=1}^{n}c_i\phi(\mathbf{x}_i) \tag{7.33}$$

Note that we assume that the points in feature space are centered, that is, we assume that the kernel matrix $\mathbf{K}$ has already been centered using Eq. (5.14):

$$\mathbf{K} = \left(\mathbf{I} - \frac{1}{n}\mathbf{1}_{n\times n}\right)\mathbf{K}\left(\mathbf{I} - \frac{1}{n}\mathbf{1}_{n\times n}\right)$$

where $\mathbf{I}$ is the $n \times n$ identity matrix, and $\mathbf{1}_{n\times n}$ is the $n \times n$ matrix all of whose elements are 1.

We have so far managed to replace one of the dot products with the kernel function. To make sure that all computations in feature space are only in terms of dot products, we can take any point, say $\phi(\mathbf{x}_k)$ and multiply Eq. (7.33) by $\phi(\mathbf{x}_k)^T$ on both sides to obtain

$$\sum_{i=1}^{n}\left(\phi(\mathbf{x}_k)^T\phi(\mathbf{x}_i)\sum_{j=1}^{n}c_j K(\mathbf{x}_i,\mathbf{x}_j)\right) = n\lambda_1\sum_{i=1}^{n}c_i\phi(\mathbf{x}_k)^T\phi(\mathbf{x}_i)$$

$$\sum_{i=1}^{n}\left(K(\mathbf{x}_k,\mathbf{x}_i)\sum_{j=1}^{n}c_j K(\mathbf{x}_i,\mathbf{x}_j)\right) = n\lambda_1\sum_{i=1}^{n}c_i K(\mathbf{x}_k,\mathbf{x}_i) \tag{7.34}$$

Further, let $\mathbf{K}_i$ denote row i of the centered kernel matrix, written as the column vector

$$\mathbf{K}_i = (K(\mathbf{x}_i,\mathbf{x}_1)\ \ K(\mathbf{x}_i,\mathbf{x}_2)\ \ \cdots\ \ K(\mathbf{x}_i,\mathbf{x}_n))^T$$

Let $\mathbf{c}$ denote the column vector of weights

$$\mathbf{c} = (c_1\ \ c_2\ \ \cdots\ \ c_n)^T$$

We can plug $\mathbf{K}_i$ and $\mathbf{c}$ into Eq. (7.34), and rewrite it as

$$\sum_{i=1}^{n}K(\mathbf{x}_k,\mathbf{x}_i)\mathbf{K}_i^T\mathbf{c} = n\lambda_1\mathbf{K}_k^T\mathbf{c}$$

In fact, because we can choose any of the n points, $\phi(\mathbf{x}_k)$, in the feature space, to obtain Eq. (7.34), we have a set of n equations:

$$\sum_{i=1}^{n}K(\mathbf{x}_1,\mathbf{x}_i)\mathbf{K}_i^T\mathbf{c} = n\lambda_1\mathbf{K}_1^T\mathbf{c}$$

$$\sum_{i=1}^{n}K(\mathbf{x}_2,\mathbf{x}_i)\mathbf{K}_i^T\mathbf{c} = n\lambda_1\mathbf{K}_2^T\mathbf{c}$$

$$\vdots \qquad = \qquad \vdots$$

$$\sum_{i=1}^{n}K(\mathbf{x}_n,\mathbf{x}_i)\mathbf{K}_i^T\mathbf{c} = n\lambda_1\mathbf{K}_n^T\mathbf{c}$$

We can compactly represent all of these n equations as follows:

$$\mathbf{K}^2\mathbf{c} = n\lambda_1\mathbf{K}\mathbf{c}$$

where $\mathbf{K}$ is the centered kernel matrix. Multiplying by $\mathbf{K}^{-1}$ on both sides, we obtain

$$\mathbf{K}^{-1}\mathbf{K}^2\mathbf{c} = n\lambda_1\mathbf{K}^{-1}\mathbf{K}\mathbf{c}$$

$$\mathbf{K}\mathbf{c} = n\lambda_1\mathbf{c}$$

$$\mathbf{K}\mathbf{c} = \eta_1\mathbf{c} \tag{7.35}$$

where $\eta_1 = n\lambda_1$. Thus, the weight vector $\mathbf{c}$ is the eigenvector corresponding to the largest eigenvalue η_1 of the kernel matrix $\mathbf{K}$.

Once $\mathbf{c}$ is found, we can plug it back into Eq. (7.32) to obtain the first kernel principal component $\mathbf{u}_1$. The only constraint we impose is that $\mathbf{u}_1$ should be normalized to be a unit vector, as follows:

$$\mathbf{u}_1^T\mathbf{u}_1 = 1$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j) = 1$$

$$\mathbf{c}^T\mathbf{K}\mathbf{c} = 1$$

Noting that $\mathbf{K}\mathbf{c} = \eta_1\mathbf{c}$ from Eq. (7.35), we get

$$\mathbf{c}^T(\eta_1\mathbf{c}) = 1$$

$$\eta_1\mathbf{c}^T\mathbf{c} = 1$$

$$\|\mathbf{c}\|^2 = \frac{1}{\eta_1}$$

However, because $\mathbf{c}$ is an eigenvector of $\mathbf{K}$ it will have unit norm. Thus, to ensure that $\mathbf{u}_1$ is a unit vector, we have to scale the weight vector $\mathbf{c}$ so that its norm is $\|\mathbf{c}\| = \sqrt{\frac{1}{\eta_1}}$, which can be achieved by multiplying $\mathbf{c}$ by $\sqrt{\frac{1}{\eta_1}}$.

In general, because we do not map the input points into the feature space via ϕ, it is not possible to directly compute the principal direction, as it is specified in terms of $\phi(\mathbf{x}_i)$, as seen in Eq. (7.32). However, what matters is that we can project any point $\phi(\mathbf{x})$ onto the principal direction $\mathbf{u}_1$, as follows:

$$\mathbf{u}_1^T\phi(\mathbf{x}) = \sum_{i=1}^{n} c_i \phi(\mathbf{x}_i)^T\phi(\mathbf{x}) = \sum_{i=1}^{n} c_i K(\mathbf{x}_i, \mathbf{x})$$

which requires only kernel operations. When $\mathbf{x} = \mathbf{x}_i$ is one of the input points, the projection of $\phi(\mathbf{x}_i)$ onto the principal component $\mathbf{u}_1$ can be written as the dot product

$$\mathbf{a}_i = \mathbf{u}_1^T\phi(\mathbf{x}_i) = \mathbf{K}_i^T\mathbf{c} \tag{7.36}$$

where $\mathbf{K}_i$ is the column vector corresponding to the ith row in the kernel matrix. Thus, we have shown that all computations, either for the solution of the principal component, or for the projection of points, can be carried out using only the kernel function. Finally, we can obtain the additional principal components by solving for the other eigenvalues and eigenvectors of Eq. (7.35). In other words, if we sort the eigenvalues of $\mathbf{K}$ in decreasing order $\eta_1 \geq \eta_2 \geq \cdots \geq \eta_n \geq 0$, we can obtain the jth principal component as the corresponding eigenvector $\mathbf{c}_j$, which has to be normalized so that the norm is $\|\mathbf{c}_j\| = \sqrt{\frac{1}{\eta_j}}$, provided $\eta_j > 0$. Also, because $\eta_j = n\lambda_j$, the variance along the jth principal component is given as $\lambda_j = \frac{\eta_j}{n}$. Algorithm 7.2 gives the pseudo-code for the kernel PCA method.

ALGORITHM 7.2. Kernel Principal Component Analysis

KERNELPCA ($\mathbf{D}, K, \alpha$):

$\mathbf{K} = \left\{K(\mathbf{x}_i, \mathbf{x}_j)\right\}_{i,j=1,\ldots,n}$ // compute $n \times n$ kernel matrix

$\mathbf{K} = (\mathbf{I} - \frac{1}{n}\mathbf{1}_{n\times n})\mathbf{K}(\mathbf{I} - \frac{1}{n}\mathbf{1}_{n\times n})$ // center the kernel matrix

$(\eta_1, \eta_2, \ldots, \eta_d) = \text{eigenvalues}(\mathbf{K})$ // compute eigenvalues

$\begin{pmatrix}\mathbf{c}_1 & \mathbf{c}_2 & \cdots & \mathbf{c}_n\end{pmatrix} = \text{eigenvectors}(\mathbf{K})$ // compute eigenvectors

$\lambda_i = \frac{\eta_i}{n}$ for all $i = 1, \ldots, n$ // compute variance for each component

$\mathbf{c}_i = \sqrt{\frac{1}{\eta_i}} \cdot \mathbf{c}_i$ for all $i = 1, \ldots, n$ // ensure that $\mathbf{u}_i^T\mathbf{u}_i = 1$

$f(r) = \frac{\sum_{i=1}^{r}\lambda_i}{\sum_{i=1}^{d}\lambda_i}$, for all $r = 1, 2, \ldots, d$ // fraction of total variance

Choose smallest r so that $f(r) \geq \alpha$ // choose dimensionality

$\mathbf{C}_r = \begin{pmatrix}\mathbf{c}_1 & \mathbf{c}_2 & \cdots & \mathbf{c}_r\end{pmatrix}$ // reduced basis

$\mathbf{A} = \{\mathbf{a}_i \mid \mathbf{a}_i = \mathbf{C}_r^T\mathbf{K}_i, \text{for } i = 1, \ldots, n\}$ // reduced dimensionality data

Example 7.8. Consider the nonlinear Iris data from Example 7.7 with $n = 150$ points. Let us use the homogeneous quadratic polynomial kernel in Eq. (5.8):

$$K(\mathbf{x}_i, \mathbf{x}_j) = \left(\mathbf{x}_i^T\mathbf{x}_j\right)^2$$

The kernel matrix $\mathbf{K}$ has three nonzero eigenvalues:

$$\eta_1 = 31.0 \qquad \eta_2 = 8.94 \qquad \eta_3 = 2.76$$

$$\lambda_1 = \frac{\eta_1}{150} = 0.2067 \qquad \lambda_2 = \frac{\eta_2}{150} = 0.0596 \qquad \lambda_3 = \frac{\eta_3}{150} = 0.0184$$

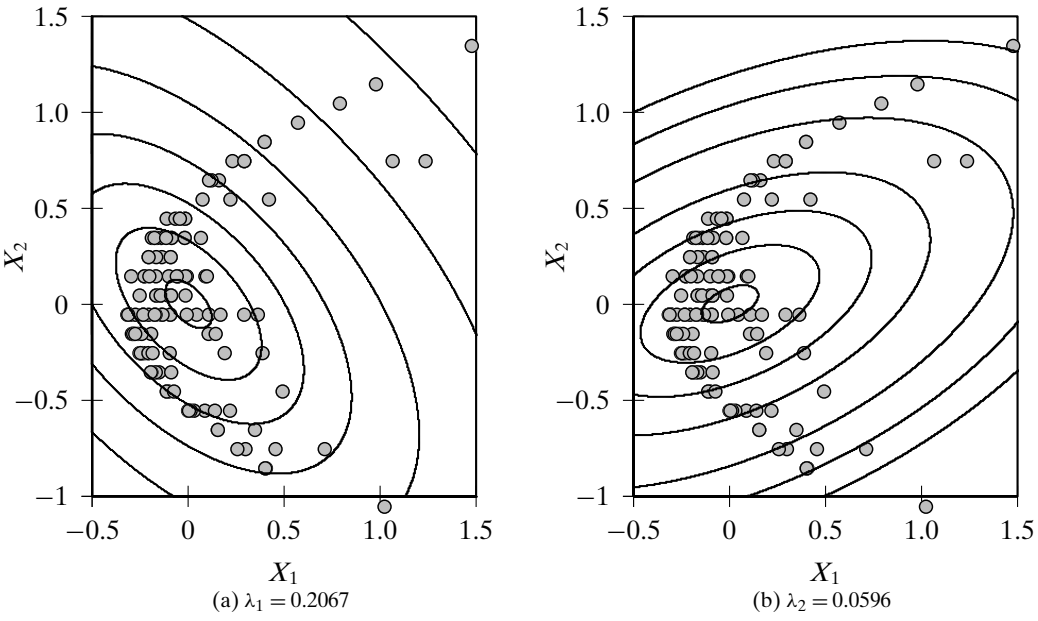
The corresponding eigenvectors $\mathbf{c}_1$, $\mathbf{c}_2$, and $\mathbf{c}_3$ are not shown because they lie in $\mathbb{R}^{150}$.

Figure 7.8 shows the contour lines of constant projection onto the first three kernel principal components. These lines are obtained by solving the equations $\mathbf{u}_i^T\mathbf{x} = \sum_{j=1}^{n} c_{ij}K(\mathbf{x}_j, \mathbf{x}) = s$ for different projection values s, for each of the eigenvectors $\mathbf{c}_i = (c_{i1}, c_{i2}, \ldots, c_{in})^T$ of the kernel matrix. For instance, for the first principal component this corresponds to the solutions $\mathbf{x} = (x_1, x_2)^T$, shown as contour lines, of the following equation:

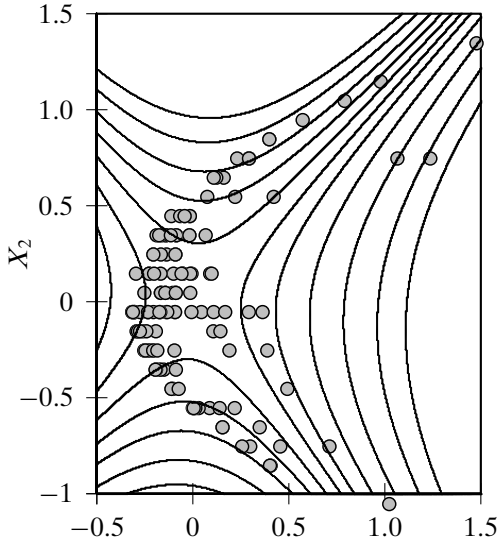
$$1.0426x_1^2 + 0.995x_2^2 + 0.914x_1x_2 = s$$

for each chosen value of s. The principal components are also not shown in the figure, as it is typically not possible or feasible to map the points into feature space, and thus one cannot derive an explicit expression for $\mathbf{u}_i$. However, because the projection onto the principal components can be carried out via kernel operations via Eq. (7.36), Figure 7.9 shows the projection of the points onto the first two kernel principal components, which capture $\frac{\lambda_1+\lambda_2}{\lambda_1+\lambda_2+\lambda_3} = \frac{0.2663}{0.2847} = 93.5\%$ of the total variance.

Incidentally, the use of a linear kernel $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T\mathbf{x}_j$ yields exactly the same principal components as shown in Figure 7.7.



(a) $\lambda_1 = 0.2067$ (b) $\lambda_2 = 0.0596$



(c) $\lambda_3 = 0.0184$

Figure 7.8. Kernel PCA: homogeneous quadratic kernel.

7.4 SINGULAR VALUE DECOMPOSITION

Principal components analysis is a special case of a more general matrix decomposition method called *Singular Value Decomposition (SVD)*. We saw in Eq. (7.28) that PCA yields the following decomposition of the covariance matrix:

$$\mathbf{\Sigma} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T \tag{7.37}$$

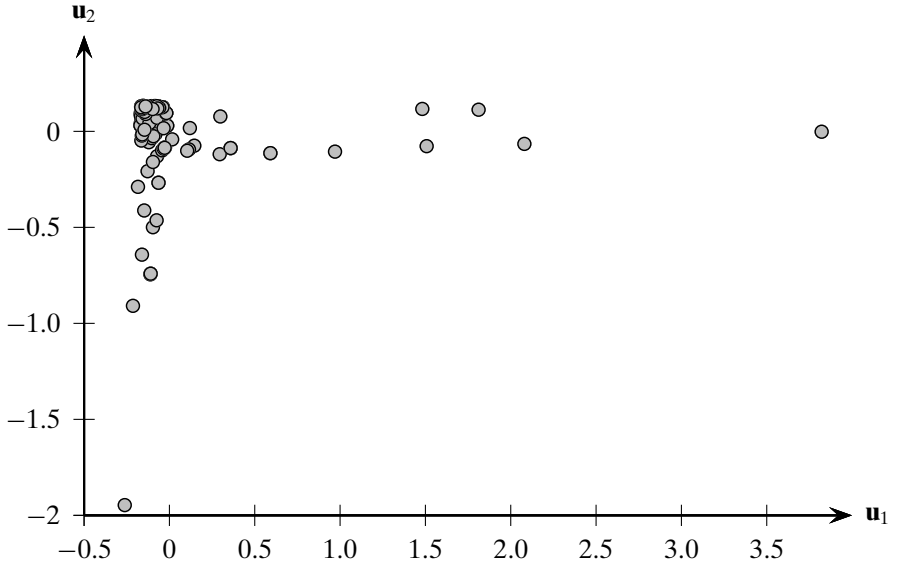


Figure 7.9. Projected point coordinates: homogeneous quadratic kernel.

where the covariance matrix has been factorized into the orthogonal matrix $\mathbf{U}$ containing its eigenvectors, and a diagonal matrix $\mathbf{\Lambda}$ containing its eigenvalues (sorted in decreasing order). SVD generalizes the above factorization for any matrix. In particular for an $n \times d$ data matrix $\mathbf{D}$ with n points and d columns, SVD factorizes $\mathbf{D}$ as follows:

$$\mathbf{D} = \mathbf{L}\mathbf{\Delta}\mathbf{R}^T \tag{7.38}$$

where $\mathbf{L}$ is a orthogonal $n \times n$ matrix, $\mathbf{R}$ is an orthogonal $d \times d$ matrix, and $\mathbf{\Delta}$ is an $n \times d$ "diagonal" matrix. The columns of $\mathbf{L}$ are called the *left singular vectors*, and the columns of $\mathbf{R}$ (or rows of $\mathbf{R}^T$) are called the *right singular vectors*. The matrix $\mathbf{\Delta}$ is defined as

$$\mathbf{\Delta}(i,j) = \begin{cases} \delta_i & \text{If } i = j \\ 0 & \text{If } i \neq j \end{cases}$$

where $i = 1, \ldots, n$ and $j = 1, \ldots, d$. The entries $\mathbf{\Delta}(i,i) = \delta_i$ along the main diagonal of $\mathbf{\Delta}$ are called the *singular values* of $\mathbf{D}$, and they are all non-negative. If the rank of $\mathbf{D}$ is $r \leq \min(n,d)$, then there will be only r nonzero singular values, which we assume are ordered as follows:

$$\delta_1 \geq \delta_2 \geq \cdots \geq \delta_r > 0$$

One can discard those left and right singular vectors that correspond to zero singular values, to obtain the *reduced SVD* as

$$\mathbf{D} = \mathbf{L}_r \mathbf{\Delta}_r \mathbf{R}_r^T \tag{7.39}$$

where $\mathbf{L}_r$ is the $n \times r$ matrix of the left singular vectors, $\mathbf{R}_r$ is the $d \times r$ matrix of the right singular vectors, and $\mathbf{\Delta}_r$ is the $r \times r$ diagonal matrix containing the positive singular vectors. The reduced SVD leads directly to the *spectral decomposition* of $\mathbf{D}$, given as

$$\begin{aligned}
\mathbf{D} &= \mathbf{L}_r \mathbf{\Delta}_r \mathbf{R}_r^T \\
&= \begin{pmatrix} | & | & & | \\ \boldsymbol{l}_1 & \boldsymbol{l}_2 & \cdots & \boldsymbol{l}_r \\ | & | & & | \end{pmatrix} \begin{pmatrix} \delta_1 & 0 & \cdots & 0 \\ 0 & \delta_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \delta_r \end{pmatrix} \begin{pmatrix} — & \mathbf{r}_1^T & — \\ — & \mathbf{r}_2^T & — \\ & \vdots & \\ — & \mathbf{r}_r^T & — \end{pmatrix} \\
&= \delta_1 \boldsymbol{l}_1 \mathbf{r}_1^T + \delta_2 \boldsymbol{l}_2 \mathbf{r}_2^T + \cdots + \delta_r \boldsymbol{l}_r \mathbf{r}_r^T \\
&= \sum_{i=1}^{r} \delta_i \boldsymbol{l}_i \mathbf{r}_i^T
\end{aligned}$$

The spectral decomposition represents $\mathbf{D}$ as a sum of rank one matrices of the form $\delta_i \boldsymbol{l}_i \mathbf{r}_i^T$. By selecting the q largest singular values $\delta_1, \delta_2, \ldots, \delta_q$ and the corresponding left and right singular vectors, we obtain the best rank q approximation to the original matrix $\mathbf{D}$. That is, if $\mathbf{D}_q$ is the matrix defined as

$$\mathbf{D}_q = \sum_{i=1}^{q} \delta_i \boldsymbol{l}_i \mathbf{r}_i^T$$

then it can be shown that $\mathbf{D}_q$ is the rank q matrix that minimizes the expression

$$\|\mathbf{D} - \mathbf{D}_q\|_F$$

where $\|\mathbf{A}\|_F$ is called the *Frobenius Norm* of the $n \times d$ matrix $\mathbf{A}$, defined as

$$\|\mathbf{A}\|_F = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{d} \mathbf{A}(i,j)^2}$$

7.4.1 Geometry of SVD

In general, any $n \times d$ matrix $\mathbf{D}$ represents a *linear transformation*, $\mathbf{D}: \mathbb{R}^d \rightarrow \mathbb{R}^n$, from the space of d-dimensional vectors to the space of n-dimensional vectors because for any $\mathbf{x} \in \mathbb{R}^d$ there exists $\mathbf{y} \in \mathbb{R}^n$ such that

$$\mathbf{D}\mathbf{x} = \mathbf{y}$$

The set of all vectors $\mathbf{y} \in \mathbb{R}^n$ such that $\mathbf{D}\mathbf{x} = \mathbf{y}$ over all possible $\mathbf{x} \in \mathbb{R}^d$ is called the *column space* of $\mathbf{D}$, and the set of all vectors $\mathbf{x} \in \mathbb{R}^d$, such that $\mathbf{D}^T\mathbf{y} = \mathbf{x}$ over all $\mathbf{y} \in \mathbb{R}^n$, is called the *row space* of $\mathbf{D}$, which is equivalent to the column space of $\mathbf{D}^T$. In other words, the column space of $\mathbf{D}$ is the set of all vectors that can be obtained as linear combinations of columns of $\mathbf{D}$, and the row space of $\mathbf{D}$ is the set of all vectors that can

be obtained as linear combinations of the rows of $\mathbf{D}$ (or columns of $\mathbf{D}^T$). Also note that the set of all vectors $\mathbf{x} \in \mathbb{R}^d$, such that $\mathbf{Dx} = \mathbf{0}$ is called the *null space* of $\mathbf{D}$, and finally, the set of all vectors $\mathbf{y} \in \mathbb{R}^n$, such that $\mathbf{D}^T\mathbf{y} = \mathbf{0}$ is called the *left null space* of $\mathbf{D}$.

One of the main properties of SVD is that it gives a basis for each of the four fundamental spaces associated with the matrix $\mathbf{D}$. If $\mathbf{D}$ has rank r, it means that it has only r independent columns, and also only r independent rows. Thus, the r left singular vectors $\boldsymbol{l}_1, \boldsymbol{l}_2, \ldots, \boldsymbol{l}_r$ corresponding to the r nonzero singular values of $\mathbf{D}$ in Eq. (7.38) represent a basis for the column space of $\mathbf{D}$. The remaining $n - r$ left singular vectors $\boldsymbol{l}_{r+1}, \ldots, \boldsymbol{l}_n$ represent a basis for the left null space of $\mathbf{D}$. For the row space, the r right singular vectors $\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_r$ corresponding to the r non-zero singular values, represent a basis for the row space of $\mathbf{D}$, and the remaining $d - r$ right singular vectors $\mathbf{r}_j$ $(j = r+1, \ldots, d)$, represent a basis for the null space of $\mathbf{D}$.

Consider the reduced SVD expression in Eq. (7.39). Right multiplying both sides of the equation by $\mathbf{R}_r$ and noting that $\mathbf{R}_r^T\mathbf{R}_r = \mathbf{I}_r$, where $\mathbf{I}_r$ is the $r \times r$ identity matrix, we have

$$\mathbf{DR}_r = \mathbf{L}_r \boldsymbol{\Delta}_r \mathbf{R}_r^T \mathbf{R}_r$$

$$\mathbf{DR}_r = \mathbf{L}_r \boldsymbol{\Delta}_r$$

$$\mathbf{DR}_r = \mathbf{L}_r \begin{pmatrix} \delta_1 & 0 & \cdots & 0 \\ 0 & \delta_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \delta_r \end{pmatrix}$$

$$\mathbf{D} \begin{pmatrix} | & | & & | \\ \mathbf{r}_1 & \mathbf{r}_2 & \cdots & \mathbf{r}_r \\ | & | & & | \end{pmatrix} = \begin{pmatrix} | & | & & | \\ \delta_1 \boldsymbol{l}_1 & \delta_2 \boldsymbol{l}_2 & \cdots & \delta_r \boldsymbol{l}_r \\ | & | & & | \end{pmatrix}$$

From the above, we conclude that

$$\mathbf{Dr}_i = \delta_i \boldsymbol{l}_i \quad \text{for all } i = 1, \ldots, r$$

In other words, SVD is a special factorization of the matrix $\mathbf{D}$, such that any basis vector $\mathbf{r}_i$ for the row space is mapped to the corresponding basis vector $\boldsymbol{l}_i$ in the column space, scaled by the singular value δ_i. As such, we can think of the SVD as a mapping from an orthonormal basis $(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_r)$ in $\mathbb{R}^d$ (the row space) to an orthonormal basis $(\boldsymbol{l}_1, \boldsymbol{l}_2, \ldots, \boldsymbol{l}_r)$ in $\mathbb{R}^n$ (the column space), with the corresponding axes scaled according to the singular values $\delta_1, \delta_2, \ldots, \delta_r$.

7.4.2 Connection between SVD and PCA

Assume that the matrix $\mathbf{D}$ has been centered, and assume that it has been factorized via SVD [Eq. (7.38)] as $\mathbf{D} = \mathbf{L}\boldsymbol{\Delta}\mathbf{R}^T$. Consider the *scatter matrix* for $\mathbf{D}$, given as $\mathbf{D}^T\mathbf{D}$. We have

$$\begin{aligned} \mathbf{D}^T\mathbf{D} &= \left(\mathbf{L}\boldsymbol{\Delta}\mathbf{R}^T\right)^T \left(\mathbf{L}\boldsymbol{\Delta}\mathbf{R}^T\right) \\ &= \mathbf{R}\boldsymbol{\Delta}^T\mathbf{L}^T\mathbf{L}\boldsymbol{\Delta}\mathbf{R}^T \end{aligned}$$

$$= \mathbf{R}(\boldsymbol{\Delta}^T\boldsymbol{\Delta})\mathbf{R}^T$$
$$= \mathbf{R}\boldsymbol{\Delta}_d^2\mathbf{R}^T \tag{7.40}$$

where $\boldsymbol{\Delta}_d^2$ is the $d \times d$ diagonal matrix defined as $\boldsymbol{\Delta}_d^2(i,i) = \delta_i^2$, for $i = 1, \ldots, d$. Only $r \leq \min(d, n)$ of these eigenvalues are positive, whereas the rest are all zeros.

Because the covariance matrix of centered $\mathbf{D}$ is given as $\boldsymbol{\Sigma} = \frac{1}{n}\mathbf{D}^T\mathbf{D}$, and because it can be decomposed as $\boldsymbol{\Sigma} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T$ via PCA [Eq. (7.37)], we have

$$\mathbf{D}^T\mathbf{D} = n\boldsymbol{\Sigma}$$
$$= n\mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T$$
$$= \mathbf{U}(n\boldsymbol{\Lambda})\mathbf{U}^T \tag{7.41}$$

Equating Eq. (7.40) and Eq. (7.41), we conclude that the right singular vectors $\mathbf{R}$ are the same as the eigenvectors of $\boldsymbol{\Sigma}$. Further, the corresponding singular values of $\mathbf{D}$ are related to the eigenvalues of $\boldsymbol{\Sigma}$ by the expression

$$n\lambda_i = \delta_i^2$$
$$\text{or, } \lambda_i = \frac{\delta_i^2}{n}, \text{ for } i = 1, \ldots, d \tag{7.42}$$

Let us now consider the matrix $\mathbf{D}\mathbf{D}^T$. We have

$$\mathbf{D}\mathbf{D}^T = (\mathbf{L}\boldsymbol{\Delta}\mathbf{R}^T)(\mathbf{L}\boldsymbol{\Delta}\mathbf{R}^T)^T$$
$$= \mathbf{L}\boldsymbol{\Delta}\mathbf{R}^T\mathbf{R}\boldsymbol{\Delta}^T\mathbf{L}^T$$
$$= \mathbf{L}(\boldsymbol{\Delta}\boldsymbol{\Delta}^T)\mathbf{L}^T$$
$$= \mathbf{L}\boldsymbol{\Delta}_n^2\mathbf{L}^T$$

where $\boldsymbol{\Delta}_n^2$ is the $n \times n$ diagonal matrix given as $\boldsymbol{\Delta}_n^2(i,i) = \delta_i^2$, for $i = 1, \ldots, n$. Only r of these singular values are positive, whereas the rest are all zeros. Thus, the left singular vectors in $\mathbf{L}$ are the eigenvectors of the matrix $n \times n$ matrix $\mathbf{D}\mathbf{D}^T$, and the corresponding eigenvalues are given as δ_i^2.

Example 7.9. Let us consider the $n \times d$ centered Iris data matrix $\mathbf{D}$ from Example 7.1, with $n = 150$ and $d = 3$. In Example 7.5 we computed the eigenvectors and eigenvalues of the covariance matrix $\boldsymbol{\Sigma}$ as follows:

$$\lambda_1 = 3.662 \qquad \lambda_2 = 0.239 \qquad \lambda_3 = 0.059$$
$$\mathbf{u}_1 = \begin{pmatrix} -0.390 \\ 0.089 \\ -0.916 \end{pmatrix} \qquad \mathbf{u}_2 = \begin{pmatrix} -0.639 \\ -0.742 \\ 0.200 \end{pmatrix} \qquad \mathbf{u}_3 = \begin{pmatrix} -0.663 \\ 0.664 \\ 0.346 \end{pmatrix}$$

Computing the SVD of $\mathbf{D}$ yields the following nonzero singular values and the corresponding right singular vectors

$$\delta_1 = 23.437 \qquad \delta_2 = 5.992 \qquad \delta_3 = 2.974$$

$$\mathbf{r}_1 = \begin{pmatrix} -0.390 \\ 0.089 \\ -0.916 \end{pmatrix} \qquad \mathbf{r}_2 = \begin{pmatrix} 0.639 \\ 0.742 \\ -0.200 \end{pmatrix} \qquad \mathbf{r}_3 = \begin{pmatrix} -0.663 \\ 0.664 \\ 0.346 \end{pmatrix}$$

We do not show the left singular vectors $\boldsymbol{l}_1, \boldsymbol{l}_2, \boldsymbol{l}_3$ because they lie in $\mathbb{R}^{150}$. Using Eq. (7.42) one can verify that $\lambda_i = \frac{\delta_i^2}{n}$. For example,

$$\lambda_1 = \frac{\delta_1^2}{n} = \frac{23.437^2}{150} = \frac{549.29}{150} = 3.662$$

Notice also that the right singular vectors are equivalent to the principal components or eigenvectors of $\boldsymbol{\Sigma}$, up to isomorphism. That is, they may potentially be reversed in direction. For the Iris dataset, we have $\mathbf{r}_1 = \mathbf{u}_1$, $\mathbf{r}_2 = -\mathbf{u}_2$, and $\mathbf{r}_3 = \mathbf{u}_3$. Here the second right singular vector is reversed in sign when compared to the second principal component.

7.5 FURTHER READING

Principal component analysis was pioneered in Pearson (1901). For a comprehensive description of PCA see Jolliffe (2002). Kernel PCA was first introduced in Schölkopf, Smola, and Müller (1998). For further exploration of non-linear dimensionality reduction methods see Lee and Verleysen (2007). The requisite linear algebra background can be found in Strang (2006).

Jolliffe, I. (2002). *Principal Component Analysis,* 2nd ed. Springer Series in Statistics. New York: Springer Science + Business Media.

Lee, J. A. and Verleysen, M. (2007). *Nonlinear Dimensionality Reduction*. New York: Springer Science + Business Media.

Pearson, K. (1901). "On lines and planes of closest fit to systems of points in space." *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2 (11): 559–572.

Schölkopf, B., Smola, A. J., and Müller, K.-R. (1998). "Nonlinear component analysis as a kernel eigenvalue problem." *Neural Computation*, 10 (5): 1299–1319.

Strang, G. (2006). *Linear Algebra and Its Applications*, 4th ed. Independence, KY: Thomson Brooks/Cole, Cengage Learning.

7.6 EXERCISES

Q1. Consider the following data matrix $\mathbf{D}$:

X_1	X_2
8	−20
0	−1
10	−19
10	−20
2	0

(a) Compute the mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$ for $\mathbf{D}$.
(b) Compute the eigenvalues of $\boldsymbol{\Sigma}$.
(c) What is the "intrinsic" dimensionality of this dataset (discounting some small amount of variance)?
(d) Compute the first principal component.
(e) If the $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ from above characterize the normal distribution from which the points were generated, sketch the orientation/extent of the 2-dimensional normal density function.

Q2. Given the covariance matrix $\boldsymbol{\Sigma} = \begin{pmatrix} 5 & 4 \\ 4 & 5 \end{pmatrix}$, answer the following questions:

(a) Compute the eigenvalues of $\boldsymbol{\Sigma}$ by solving the equation $\det(\boldsymbol{\Sigma} - \lambda \mathbf{I}) = 0$.
(b) Find the corresponding eigenvectors by solving the equation $\boldsymbol{\Sigma}\mathbf{u}_i = \lambda_i \mathbf{u}_i$.

Q3. Compute the singular values and the left and right singular vectors of the following matrix:

$$\mathbf{A} = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Q4. Consider the data in Table 7.1. Define the kernel function as follows: $K(\mathbf{x}_i, \mathbf{x}_j) = \|\mathbf{x}_i - \mathbf{x}_j\|^2$. Answer the following questions:

(a) Compute the kernel matrix $\mathbf{K}$.
(b) Find the first kernel principal component.

Table 7.1. Dataset for Q4

i	$\mathbf{x}_i$
$\mathbf{x}_1$	(4, 2.9)
$\mathbf{x}_4$	(2.5, 1)
$\mathbf{x}_7$	(3.5, 4)
$\mathbf{x}_9$	(2, 2.1)

Q5. Given the two points $\mathbf{x}_1 = (1, 2)^T$, and $\mathbf{x}_2 = (2, 1)^T$, use the kernel function

$$K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T \mathbf{x}_j)^2$$

to find the kernel principal component, by solving the equation $\mathbf{Kc} = \eta_1 \mathbf{c}$.

PART TWO FREQUENT PATTERN MINING

CHAPTER 8 Itemset Mining

In many applications one is interested in how often two or more objects of interest co-occur. For example, consider a popular website, which logs all incoming traffic to its site in the form of weblogs. Weblogs typically record the source and destination pages requested by some user, as well as the time, return code whether the request was successful or not, and so on. Given such weblogs, one might be interested in finding if there are sets of web pages that many users tend to browse whenever they visit the website. Such "frequent" sets of web pages give clues to user browsing behavior and can be used for improving the browsing experience.

The quest to mine frequent patterns appears in many other domains. The prototypical application is *market basket analysis*, that is, to mine the sets of items that are frequently bought together at a supermarket by analyzing the customer shopping carts (the so-called "market baskets"). Once we mine the frequent sets, they allow us to extract *association rules* among the item sets, where we make some statement about how likely are two sets of items to co-occur or to conditionally occur. For example, in the weblog scenario frequent sets allow us to extract rules like, "Users who visit the sets of pages `main`, `laptops` and `rebates` also visit the pages `shopping-cart` and `checkout`", indicating, perhaps, that the special rebate offer is resulting in more laptop sales. In the case of market baskets, we can find rules such as "Customers who buy milk and cereal also tend to buy bananas," which may prompt a grocery store to co-locate bananas in the cereal aisle. We begin this chapter with algorithms to mine frequent itemsets, and then show how they can be used to extract association rules.

8.1 FREQUENT ITEMSETS AND ASSOCIATION RULES

Itemsets and Tidsets

Let $\mathcal{I} = \{x_1, x_2, \ldots, x_m\}$ be a set of elements called *items*. A set $X \subseteq \mathcal{I}$ is called an *itemset*. The set of items $\mathcal{I}$ may denote, for example, the collection of all products sold at a supermarket, the set of all web pages at a website, and so on. An itemset of cardinality (or size) k is called a k-itemset. Further, we denote by $\mathcal{I}^{(k)}$ the set of all k-itemsets, that is, subsets of $\mathcal{I}$ with size k. Let $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$ be another set of elements called

transaction identifiers or *tids*. A set $T \subseteq \mathcal{T}$ is called a *tidset*. We assume that itemsets and tidsets are kept sorted in lexicographic order.

A *transaction* is a tuple of the form $\langle t, X \rangle$, where $t \in \mathcal{T}$ is a unique transaction identifier, and X is an itemset. The set of transactions $\mathcal{T}$ may denote the set of all customers at a supermarket, the set of all the visitors to a website, and so on. For convenience, we refer to a transaction $\langle t, X \rangle$ by its identifier t.

Database Representation

A binary database $\mathbf{D}$ is a binary relation on the set of tids and items, that is, $\mathbf{D} \subseteq \mathcal{T} \times \mathcal{I}$. We say that tid $t \in \mathcal{T}$ *contains* item $x \in \mathcal{I}$ iff $(t, x) \in \mathbf{D}$. In other words, $(t, x) \in \mathbf{D}$ iff $x \in X$ in the tuple $\langle t, X \rangle$. We say that tid t *contains* itemset $X = \{x_1, x_2, \ldots, x_k\}$ iff $(t, x_i) \in \mathbf{D}$ for all $i = 1, 2, \ldots, k$.

Example 8.1. Figure 8.1a shows an example binary database. Here $\mathcal{I} = \{A, B, C, D, E\}$, and $\mathcal{T} = \{1, 2, 3, 4, 5, 6\}$. In the binary database, the cell in row t and column x is 1 iff $(t, x) \in \mathbf{D}$, and 0 otherwise. We can see that transaction 1 contains item B, and it also contains the itemset BE, and so on.

For a set X, we denote by 2^X the powerset of X, that is, the set of all subsets of X. Let $\mathbf{i} : 2^{\mathcal{T}} \to 2^{\mathcal{I}}$ be a function, defined as follows:

$$\mathbf{i}(T) = \{x \mid \forall t \in T,\ t \text{ contains } x\} \tag{8.1}$$

where $T \subseteq \mathcal{T}$, and $\mathbf{i}(T)$ is the set of items that are common to *all* the transactions in the tidset T. In particular, $\mathbf{i}(t)$ is the set of items contained in tid $t \in \mathcal{T}$. Note that in this chapter we drop the set notation for convenience (e.g., we write $\mathbf{i}(t)$ instead of $\mathbf{i}(\{t\})$). It is sometimes convenient to consider the binary database $\mathbf{D}$, as a *transaction database* consisting of tuples of the form $\langle t, \mathbf{i}(t) \rangle$, with $t \in \mathcal{T}$. The transaction or itemset database can be considered as a horizontal representation of the binary database, where we omit items that are not contained in a given tid.

Let $\mathbf{t} : 2^{\mathcal{I}} \to 2^{\mathcal{T}}$ be a function, defined as follows:

$$\mathbf{t}(X) = \{t \mid t \in \mathcal{T} \text{ and } t \text{ contains } X\} \tag{8.2}$$

where $X \subseteq \mathcal{I}$, and $\mathbf{t}(X)$ is the set of tids that contain *all* the items in the itemset X. In particular, $\mathbf{t}(x)$ is the set of tids that contain the single item $x \in \mathcal{I}$. It is also sometimes convenient to think of the binary database $\mathbf{D}$, as a tidset database containing a collection of tuples of the form $\langle x, \mathbf{t}(x) \rangle$, with $x \in \mathcal{I}$. The tidset database is a vertical representation of the binary database, where we omit tids that do not contain a given item.

Example 8.2. Figure 8.1b shows the corresponding transaction database for the binary database in Figure 8.1a. For instance, the first transaction is $\langle 1, \{A, B, D, E\} \rangle$, where we omit item C since $(1, C) \notin \mathbf{D}$. Henceforth, for convenience, we drop the set notation for itemsets and tidsets if there is no confusion. Thus, we write $\langle 1, \{A, B, D, E\} \rangle$ as $\langle 1, ABDE \rangle$.

D	A	B	C	D	E
1	1	1	0	1	1
2	0	1	1	0	1
3	1	1	0	1	1
4	1	1	1	0	1
5	1	1	1	1	1
6	0	1	1	1	0

(a) Binary database

t	$\mathbf{i}(t)$
1	$ABDE$
2	BCE
3	$ABDE$
4	$ABCE$
5	$ABCDE$
6	BCD

(b) Transaction database

x	A	B	C	D	E
$\mathbf{t}(x)$	1	1	2	1	1
	3	2	4	3	2
	4	3	5	5	3
	5	4	6	6	4
		5			5
		6			

(c) Vertical database

Figure 8.1. An example database.

Figure 8.1c shows the corresponding vertical database for the binary database in Figure 8.1a. For instance, the tuple corresponding to item A, shown in the first column, is $\langle A, \{1,3,4,5\}\rangle$, which we write as $\langle A, 1345\rangle$ for convenience; we omit tids 2 and 6 because $(2, A) \notin \mathbf{D}$ and $(6, A) \notin \mathbf{D}$.

Support and Frequent Itemsets

The *support* of an itemset X in a dataset $\mathbf{D}$, denoted $sup(X, \mathbf{D})$, is the number of transactions in $\mathbf{D}$ that contain X:

$$sup(X, \mathbf{D}) = \big|\{t \mid \langle t, \mathbf{i}(t)\rangle \in \mathbf{D} \text{ and } X \subseteq \mathbf{i}(t)\}\big| = |\mathbf{t}(X)|$$

The *relative support* of X is the fraction of transactions that contain X:

$$rsup(X, \mathbf{D}) = \frac{sup(X, \mathbf{D})}{|\mathbf{D}|}$$

It is an estimate of the *joint probability* of the items comprising X.

An itemset X is said to be *frequent* in $\mathbf{D}$ if $sup(X, \mathbf{D}) \geq minsup$, where $minsup$ is a user defined *minimum support threshold*. When there is no confusion about the database $\mathbf{D}$, we write support as $sup(X)$, and relative support as $rsup(X)$. If $minsup$ is specified as a fraction, then we assume that relative support is implied. We use the set $\mathcal{F}$ to denote the set of all frequent itemsets, and $\mathcal{F}^{(k)}$ to denote the set of frequent k-itemsets.

Example 8.3. Given the example dataset in Figure 8.1, let $minsup = 3$ (in relative support terms we mean $minsup = 0.5$). Table 8.1 shows all the 19 frequent itemsets in the database, grouped by their support value. For example, the itemset BCE is contained in tids 2, 4, and 5, so $\mathbf{t}(BCE) = 245$ and $sup(BCE) = |\mathbf{t}(BCE)| = 3$. Thus, BCE is a frequent itemset. The 19 frequent itemsets shown in the table comprise the set $\mathcal{F}$. The sets of all frequent k-itemsets are

$$\mathcal{F}^{(1)} = \{A, B, C, D, E\}$$
$$\mathcal{F}^{(2)} = \{AB, AD, AE, BC, BD, BE, CE, DE\}$$
$$\mathcal{F}^{(3)} = \{ABD, ABE, ADE, BCE, BDE\}$$
$$\mathcal{F}^{(4)} = \{ABDE\}$$

Table 8.1. Frequent itemsets with $minsup = 3$

sup	itemsets
6	B
5	E, BE
4	$A, C, D, AB, AE, BC, BD, ABE$
3	$AD, CE, DE, ABD, ADE, BCE, BDE, ABDE$

Association Rules

An *association rule* is an expression $X \xrightarrow{s,c} Y$, where X and Y are itemsets and they are disjoint, that is, $X, Y \subseteq \mathcal{I}$, and $X \cap Y = \emptyset$. Let the itemset $X \cup Y$ be denoted as XY. The *support* of the rule is the number of transactions in which both X and Y co-occur as subsets:

$$s = sup(X \longrightarrow Y) = |\mathbf{t}(XY)| = sup(XY)$$

The *relative support* of the rule is defined as the fraction of transactions where X and Y co-occur, and it provides an estimate of the joint probability of X and Y:

$$rsup(X \longrightarrow Y) = \frac{sup(XY)}{|\mathbf{D}|} = P(X \wedge Y)$$

The *confidence* of a rule is the conditional probability that a transaction contains Y given that it contains X:

$$c = conf(X \longrightarrow Y) = P(Y|X) = \frac{P(X \wedge Y)}{P(X)} = \frac{sup(XY)}{sup(X)}$$

A rule is *frequent* if the itemset XY is frequent, that is, $sup(XY) \geq minsup$ and a rule is *strong* if $conf \geq minconf$, where *minconf* is a user-specified minimum confidence threshold.

Example 8.4. Consider the association rule $BC \longrightarrow E$. Using the itemset support values shown in Table 8.1, the support and confidence of the rule are as follows:

$$s = sup(BC \longrightarrow E) = sup(BCE) = 3$$

$$c = conf(BC \longrightarrow E) = \frac{sup(BCE)}{sup(BC)} = 3/4 = 0.75$$

Itemset and Rule Mining

From the definition of rule support and confidence, we can observe that to generate frequent and high confidence association rules, we need to first enumerate all the frequent itemsets along with their support values. Formally, given a binary database $\mathbf{D}$ and a user defined minimum support threshold *minsup*, the task of frequent itemset mining is to enumerate all itemsets that are frequent, i.e., those that have support at least *minsup*. Next, given the set of frequent itemsets $\mathcal{F}$ and a minimum confidence value *minconf*, the association rule mining task is to find all frequent and strong rules.

8.2 ITEMSET MINING ALGORITHMS

We begin by describing a naive or brute-force algorithm that enumerates all the possible itemsets $X \subseteq \mathcal{I}$, and for each such subset determines its support in the input dataset $\mathbf{D}$. The method comprises two main steps: (1) candidate generation and (2) support computation.

Candidate Generation

This step generates all the subsets of $\mathcal{I}$, which are called *candidates*, as each itemset is potentially a candidate frequent pattern. The candidate itemset search space is clearly exponential because there are $2^{|\mathcal{I}|}$ potentially frequent itemsets. It is also instructive to note the structure of the itemset search space; the set of all itemsets forms a lattice structure where any two itemsets X and Y are connected by a link iff X is an *immediate subset* of Y, that is, $X \subseteq Y$ and $|X| = |Y| - 1$. In terms of a practical search strategy, the itemsets in the lattice can be enumerated using either a breadth-first (BFS) or depth-first (DFS) search on the *prefix tree*, where two itemsets X, Y are connected by a link iff X is an immediate subset and prefix of Y. This allows one to enumerate itemsets starting with an empty set, and adding one more item at a time.

Support Computation

This step computes the support of each candidate pattern X and determines if it is frequent. For each transaction $\langle t, \mathbf{i}(t) \rangle$ in the database, we determine if X is a subset of $\mathbf{i}(t)$. If so, we increment the support of X.

The pseudo-code for the brute-force method is shown in Algorithm 8.1. It enumerates each itemset $X \subseteq \mathcal{I}$, and then computes its support by checking if $X \subseteq \mathbf{i}(t)$ for each $t \in \mathcal{T}$.

ALGORITHM 8.1. Algorithm BRUTEFORCE

```
BRUTEFORCE (D, I, minsup):
1  F ← ∅ // set of frequent itemsets
2  foreach X ⊆ I do
3      sup(X) ← COMPUTESUPPORT (X, D)
4      if sup(X) ≥ minsup then
5          F ← F ∪ {(X, sup(X))}
6  return F

COMPUTESUPPORT (X, D):
7  sup(X) ← 0
8  foreach ⟨t, i(t)⟩ ∈ D do
9      if X ⊆ i(t) then
10         sup(X) ← sup(X) + 1
11 return sup(X)
```

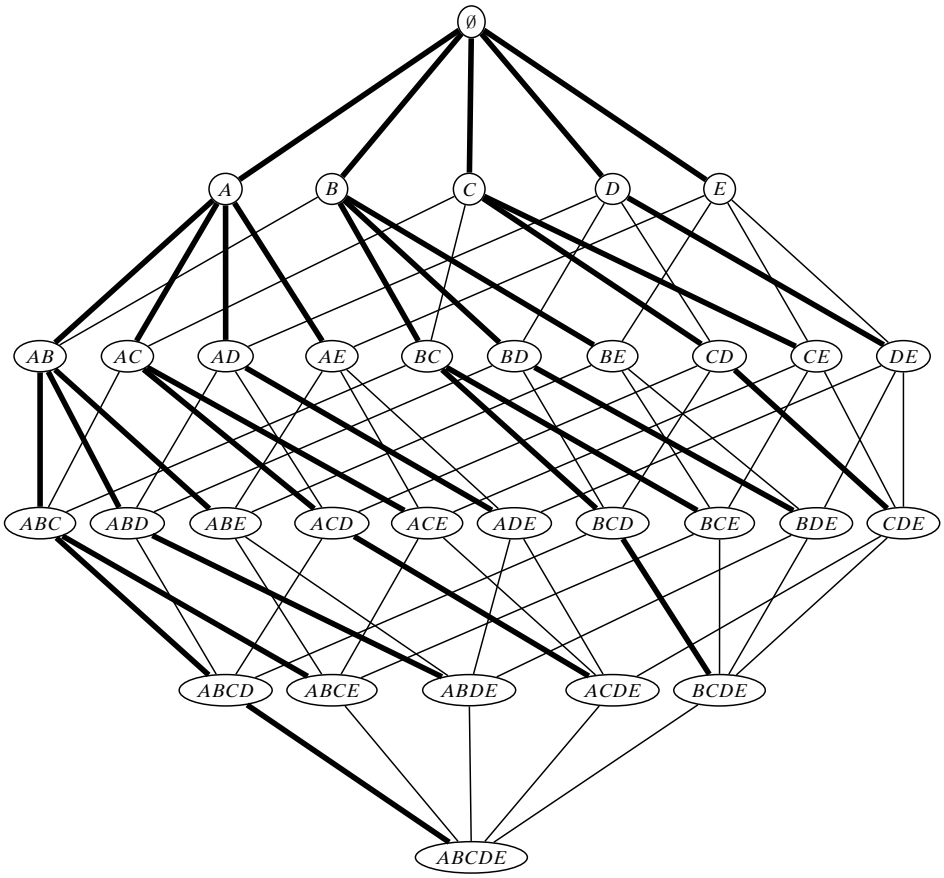


Figure 8.2. Itemset lattice and prefix-based search tree (in bold).

Example 8.5. Figure 8.2 shows the itemset lattice for the set of items $\mathcal{I} = \{A, B, C, D, E\}$. There are $2^{|\mathcal{I}|} = 2^5 = 32$ possible itemsets including the empty set. The corresponding prefix search tree is also shown (in bold). The brute-force method explores the entire itemset search space, regardless of the *minsup* threshold employed. If $minsup = 3$, then the brute-force method would output the set of frequent itemsets shown in Table 8.1.

Computational Complexity

Support computation takes time $O(|\mathcal{I}| \cdot |\mathbf{D}|)$ in the worst case, and because there are $O(2^{|\mathcal{I}|})$ possible candidates, the computational complexity of the brute-force method is $O(|\mathcal{I}| \cdot |\mathbf{D}| \cdot 2^{|\mathcal{I}|})$. Because the database $\mathbf{D}$ can be very large, it is also important to measure the input/output (I/O) complexity. Because we make one complete database scan to compute the support of each candidate, the I/O complexity of BRUTEFORCE is $O(2^{|\mathcal{I}|})$ database scans. Thus, the brute force approach is computationally infeasible for even small itemset spaces, whereas in practice $\mathcal{I}$ can be very large (e.g., a supermarket carries thousands of items). The approach is impractical from an I/O perspective as well.

We shall see next how to systematically improve on the brute force approach, by improving both the candidate generation and support counting steps.

8.2.1 Level-wise Approach: Apriori Algorithm

The brute force approach enumerates all possible itemsets in its quest to determine the frequent ones. This results in a lot of wasteful computation because many of the candidates may not be frequent. Let $X, Y \subseteq \mathcal{I}$ be any two itemsets. Note that if $X \subseteq Y$, then $sup(X) \geq sup(Y)$, which leads to the following two observations: (1) if X is frequent, then any subset $Y \subseteq X$ is also frequent, and (2) if X is not frequent, then any superset $Y \supseteq X$ cannot be frequent. The *Apriori algorithm* utilizes these two properties to significantly improve the brute-force approach. It employs a level-wise or breadth-first exploration of the itemset search space, and prunes all supersets of any infrequent candidate, as no superset of an infrequent itemset can be frequent. It also avoids generating any candidate that has an infrequent subset. In addition to improving the candidate generation step via itemset pruning, the Apriori method also significantly improves the I/O complexity. Instead of counting the support for a single itemset, it explores the prefix tree in a breadth-first manner, and computes the support of all the valid candidates of size k that comprise level k in the prefix tree.

Example 8.6. Consider the example dataset in Figure 8.1; let $minsup = 3$. Figure 8.3 shows the itemset search space for the Apriori method, organized as a prefix tree where two itemsets are connected if one is a prefix and immediate subset of the other. Each node shows an itemset along with its support, thus $AC(2)$ indicates that $sup(AC) = 2$. Apriori enumerates the candidate patterns in a level-wise manner, as shown in the figure, which also demonstrates the power of pruning the search space via the two Apriori properties. For example, once we determine that AC is infrequent, we can prune any itemset that has AC as a prefix, that is, the entire subtree under AC can be pruned. Likewise for CD. Also, the extension BCD from BC can be pruned, since it has an infrequent subset, namely CD.

Algorithm 8.2 shows the pseudo-code for the Apriori method. Let $\mathcal{C}^{(k)}$ denote the prefix tree comprising all the candidate k-itemsets. The method begins by inserting the single items into an initially empty prefix tree to populate $\mathcal{C}^{(1)}$. The while loop (lines 5–11) first computes the support for the current set of candidates at level k via the COMPUTESUPPORT procedure that generates k-subsets of each transaction in the database $\mathbf{D}$, and for each such subset it increments the support of the corresponding candidate in $\mathcal{C}^{(k)}$ if it exists. This way, the database is scanned only once per level, and the supports for all candidate k-itemsets are incremented during that scan. Next, we remove any infrequent candidate (line 9). The leaves of the prefix tree that survive comprise the set of frequent k-itemsets $\mathcal{F}^{(k)}$, which are used to generate the candidate $(k+1)$-itemsets for the next level (line 10). The EXTENDPREFIXTREE procedure employs prefix-based extension for candidate generation. Given two frequent k-itemsets X_a and X_b with a common $k-1$ length prefix, that is, given two sibling leaf nodes with a common parent, we generate the $(k+1)$-length candidate $X_{ab} = X_a \cup X_b$. This candidate is retained only if it has no infrequent subset. Finally, if a k-itemset X_a has no extension, it is pruned from the prefix tree, and we recursively

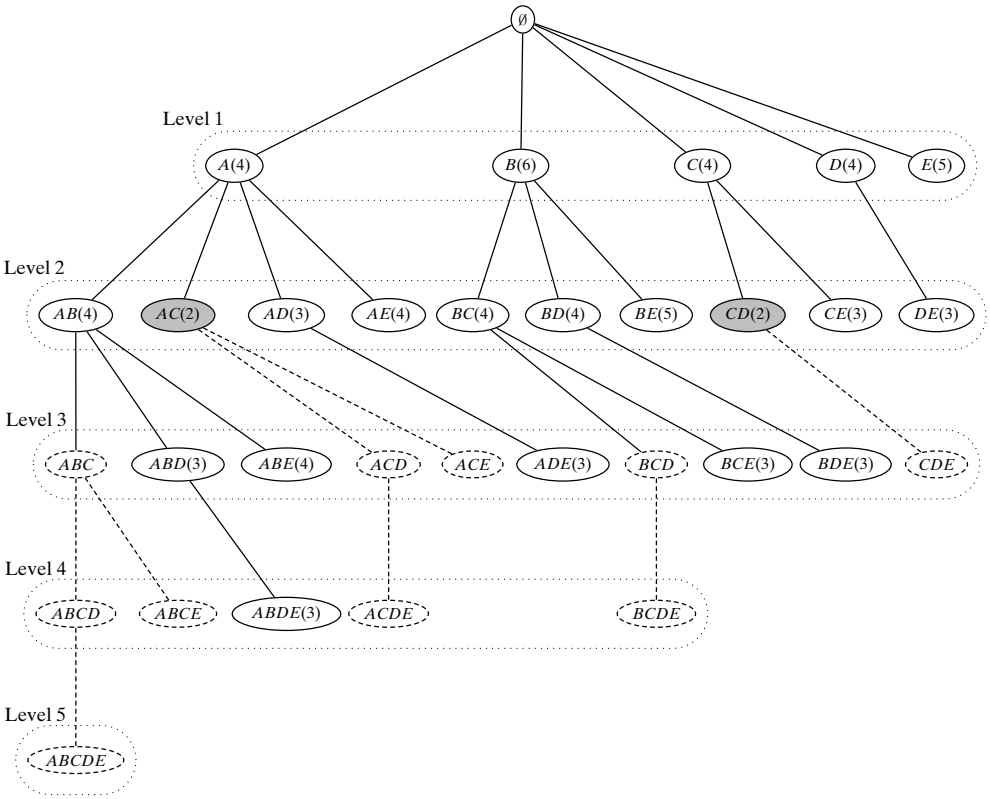


Figure 8.3. Apriori: prefix search tree and effect of pruning. Shaded nodes indicate infrequent itemsets, whereas dashed nodes and lines indicate all of the pruned nodes and branches. Solid lines indicate frequent itemsets.

prune any of its ancestors with no k-itemset extension, so that in $\mathcal{C}^{(k)}$ all leaves are at level k. If new candidates were added, the whole process is repeated for the next level. This process continues until no new candidates are added.

Example 8.7. Figure 8.4 illustrates the Apriori algorithm on the example dataset from Figure 8.1 using $minsup = 3$. All the candidates $\mathcal{C}^{(1)}$ are frequent (see Figure 8.4a). During extension all the pairwise combinations will be considered, since they all share the empty prefix $\emptyset$ as their parent. These comprise the new prefix tree $\mathcal{C}^{(2)}$ in Figure 8.4b; because E has no prefix-based extensions, it is removed from the tree. After support computation $AC(2)$ and $CD(2)$ are eliminated (shown in gray) since they are infrequent. The next level prefix tree is shown in Figure 8.4c. The candidate BCD is pruned due to the presence of the infrequent subset CD. All of the candidates at level 3 are frequent. Finally, $\mathcal{C}^{(4)}$ (shown in Figure 8.4d) has only one candidate $X_{ab} = ABDE$, which is generated from $X_a = ABD$ and $X_b = ABE$ because this is the only pair of siblings. The mining process stops after this step, since no more extensions are possible.

The worst-case computational complexity of the Apriori algorithm is still $O(|\mathcal{I}| \cdot |\mathbf{D}| \cdot 2^{|\mathcal{I}|})$, as all itemsets may be frequent. In practice, due to the pruning of the search

ALGORITHM 8.2. Algorithm APRIORI

```
APRIORI (D, I, minsup):
1  F ← ∅
2  C^(1) ← {∅} // Initial prefix tree with single items
3  foreach i ∈ I do Add i as child of ∅ in C^(1) with sup(i) ← 0
4  k ← 1 // k denotes the level
5  while C^(k) ≠ ∅ do
6      COMPUTESUPPORT (C^(k), D)
7      foreach leaf X ∈ C^(k) do
8          if sup(X) ≥ minsup then F ← F ∪ {(X, sup(X))}
9          else remove X from C^(k)
10     C^(k+1) ← EXTENDPREFIXTREE (C^(k))
11     k ← k + 1
12 return F^(k)

COMPUTESUPPORT (C^(k), D):
13 foreach ⟨t, i(t)⟩ ∈ D do
14     foreach k-subset X ⊆ i(t) do
15         if X ∈ C^(k) then sup(X) ← sup(X) + 1

EXTENDPREFIXTREE (C^(k)):
16 foreach leaf X_a ∈ C^(k) do
17     foreach leaf X_b ∈ SIBLING(X_a), such that b > a do
18         X_ab ← X_a ∪ X_b
           // prune candidate if there are any infrequent subsets
19         if X_j ∈ C^(k), for all X_j ⊂ X_ab, such that |X_j| = |X_ab| − 1 then
20             Add X_ab as child of X_a with sup(X_ab) ← 0
21     if no extensions from X_a then
22         remove X_a, and all ancestors of X_a with no extensions, from C^(k)
23 return C^(k)
```

space the cost is much lower. However, in terms of I/O cost Apriori requires $O(|\mathcal{I}|)$ database scans, as opposed to the $O(2^{|\mathcal{I}|})$ scans in the brute-force method. In practice, it requires only l database scans, where l is the length of the longest frequent itemset.

8.2.2 Tidset Intersection Approach: Eclat Algorithm

The support counting step can be improved significantly if we can index the database in such a way that it allows fast frequency computations. Notice that in the level-wise approach, to count the support, we have to generate subsets of each transaction and check whether they exist in the prefix tree. This can be expensive because we may end up generating many subsets that do not exist in the prefix tree.

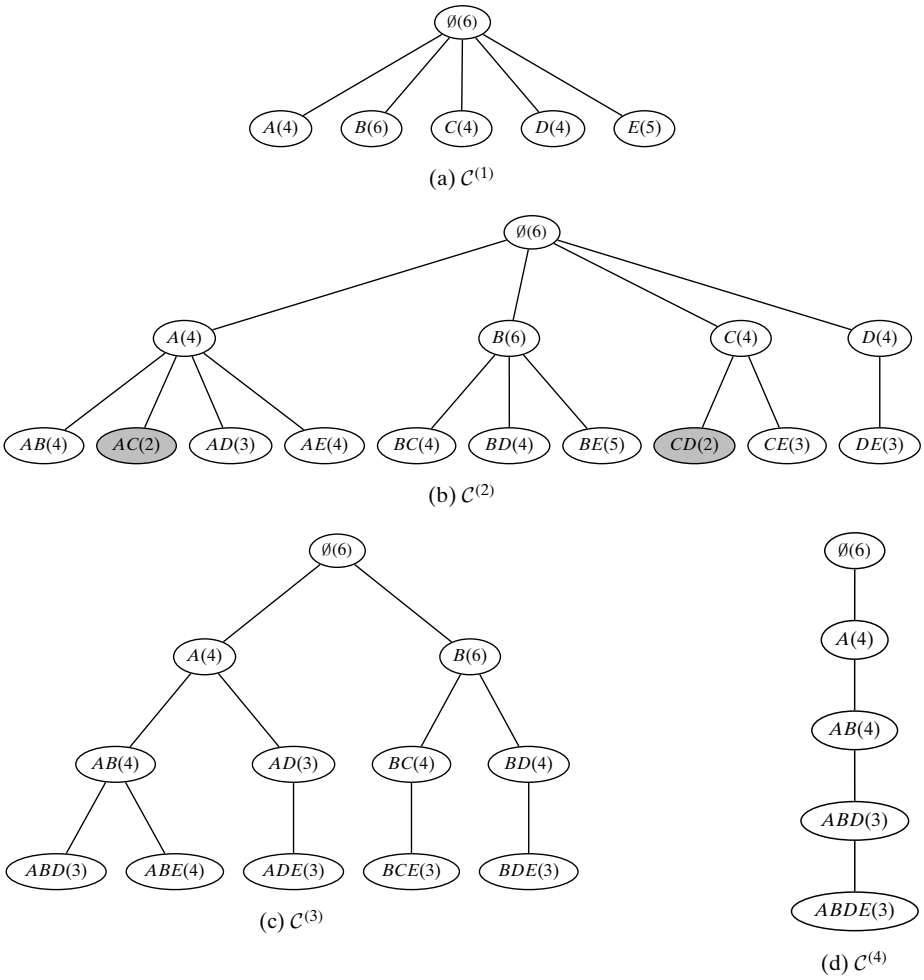


Figure 8.4. Itemset mining: Apriori algorithm. The prefix search trees $\mathcal{C}^{(k)}$ at each level are shown. Leaves (unshaded) comprise the set of frequent k-itemsets $\mathcal{F}^{(k)}$.

The Eclat algorithm leverages the tidsets directly for support computation. The basic idea is that the support of a candidate itemset can be computed by intersecting the tidsets of suitably chosen subsets. In general, given $\mathbf{t}(X)$ and $\mathbf{t}(Y)$ for any two frequent itemsets X and Y, we have

$$\mathbf{t}(XY) = \mathbf{t}(X) \cap \mathbf{t}(Y)$$

The support of candidate XY is simply the cardinality of $\mathbf{t}(XY)$, that is, $sup(XY) = |\mathbf{t}(XY)|$. Eclat intersects the tidsets only if the frequent itemsets share a common prefix, and it traverses the prefix search tree in a DFS-like manner, processing a group of itemsets that have the same prefix, also called a *prefix equivalence class*.

Example 8.8. For example, if we know that the tidsets for item A and C are $\mathbf{t}(A) = 1345$ and $\mathbf{t}(C) = 2456$, respectively, then we can determine the support of AC by intersecting the two tidsets, to obtain $\mathbf{t}(AC) = \mathbf{t}(A) \cap \mathbf{t}(C) = 1345 \cap 2456 = 45$.

ALGORITHM 8.3. Algorithm ECLAT

```
// Initial Call: F ← ∅, P ← {⟨i, t(i)⟩ | i ∈ I, |t(i)| ≥ minsup}
ECLAT (P, minsup, F):
1 foreach ⟨X_a, t(X_a)⟩ ∈ P do
2     F ← F ∪ {(X_a, sup(X_a))}
3     P_a ← ∅
4     foreach ⟨X_b, t(X_b)⟩ ∈ P, with X_b > X_a do
5         X_ab = X_a ∪ X_b
6         t(X_ab) = t(X_a) ∩ t(X_b)
7         if sup(X_ab) ≥ minsup then
8             P_a ← P_a ∪ {⟨X_ab, t(X_ab)⟩}
9     if P_a ≠ ∅ then ECLAT (P_a, minsup, F)
```

In this case, we have $sup(AC) = |45| = 2$. An example of a prefix equivalence class is the set $P_A = \{AB, AC, AD, AE\}$, as all the elements of P_A share A as the prefix.

The pseudo-code for Eclat is given in Algorithm 8.3. It employs a vertical representation of the binary database **D**. Thus, the input is the set of tuples $\langle i, \mathbf{t}(i)\rangle$ for all frequent items $i \in \mathcal{I}$, which comprise an equivalence class P (they all share the empty prefix); it is assumed that P contains only frequent itemsets. In general, given a prefix equivalence class P, for each frequent itemset $X_a \in P$, we try to intersect its tidset with the tidsets of all other itemsets $X_b \in P$. The candidate pattern is $X_{ab} = X_a \cup X_b$, and we check the cardinality of the intersection $\mathbf{t}(X_a) \cap \mathbf{t}(X_b)$ to determine whether it is frequent. If so, X_{ab} is added to the new equivalence class P_a that contains all itemsets that share X_a as a prefix. A recursive call to Eclat then finds all extensions of the X_a branch in the search tree. This process continues until no extensions are possible over all branches.

Example 8.9. Figure 8.5 illustrates the Eclat algorithm. Here $minsup = 3$, and the initial prefix equivalence class is

$$P_\emptyset = \{\langle A, 1345\rangle, \langle B, 123456\rangle, \langle C, 2456\rangle, \langle D, 1356\rangle, \langle E, 12345\rangle\}$$

Eclat intersects $\mathbf{t}(A)$ with each of $\mathbf{t}(B)$, $\mathbf{t}(C)$, $\mathbf{t}(D)$, and $\mathbf{t}(E)$ to obtain the tidsets for AB, AC, AD and AE, respectively. Out of these AC is infrequent and is pruned (marked gray). The frequent itemsets and their tidsets comprise the new prefix equivalence class

$$P_A = \{\langle AB, 1345\rangle, \langle AD, 135\rangle, \langle AE, 1345\rangle\}$$

which is recursively processed. On return, Eclat intersects $\mathbf{t}(B)$ with $\mathbf{t}(C)$, $\mathbf{t}(D)$, and $\mathbf{t}(E)$ to obtain the equivalence class

$$P_B = \{\langle BC, 2456\rangle, \langle BD, 1356\rangle, \langle BE, 12345\rangle\}$$

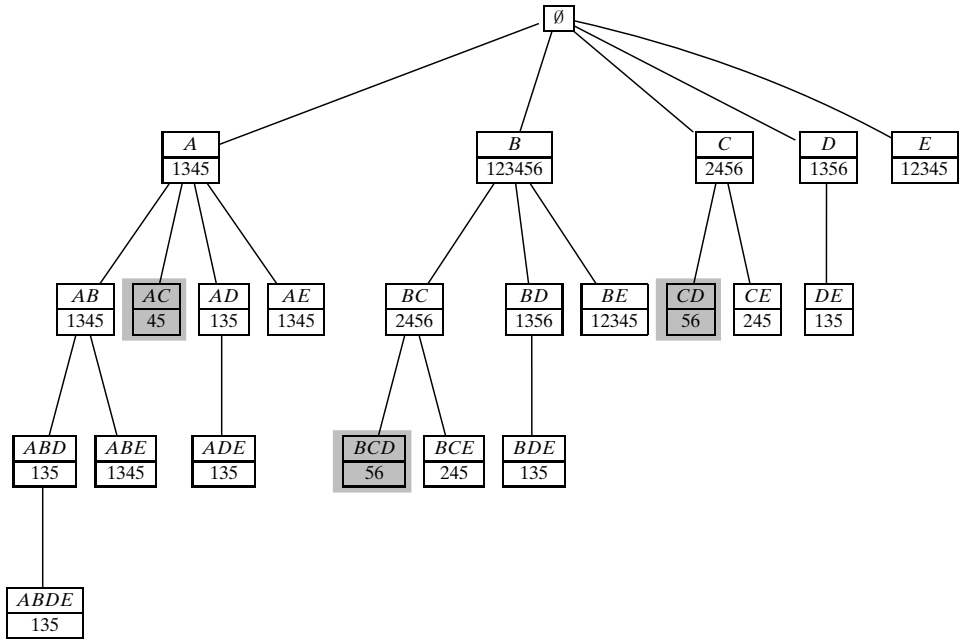


Figure 8.5. Eclat algorithm: tidlist intersections (gray boxes indicate infrequent itemsets).

Other branches are processed in a similar manner; the entire search space that Eclat explores is shown in Figure 8.5. The gray nodes indicate infrequent itemsets, whereas the rest constitute the set of frequent itemsets.

The computational complexity of Eclat is $O(|\mathbf{D}| \cdot 2^{|\mathcal{I}|})$ in the worst case, since there can be $2^{|\mathcal{I}|}$ frequent itemsets, and an intersection of two tidsets takes at most $O(|\mathbf{D}|)$ time. The I/O complexity of Eclat is harder to characterize, as it depends on the size of the intermediate tidsets. With t as the average tidset size, the initial database size is $O(t \cdot |\mathcal{I}|)$, and the total size of all the intermediate tidsets is $O(t \cdot 2^{|\mathcal{I}|})$. Thus, Eclat requires $\frac{t \cdot 2^{|\mathcal{I}|}}{t \cdot |\mathcal{I}|} = O(2^{|\mathcal{I}|}/|\mathcal{I}|)$ database scans in the worst case.

Diffsets: Difference of Tidsets

The Eclat algorithm can be significantly improved if we can shrink the size of the intermediate tidsets. This can be achieved by keeping track of the differences in the tidsets as opposed to the full tidsets. Formally, let $X_k = \{x_1, x_2, \ldots, x_{k-1}, x_k\}$ be a k-itemset. Define the *diffset* of X_k as the set of tids that contain the prefix $X_{k-1} = \{x_1, \ldots, x_{k-1}\}$ but do not contain the item x_k, given as

$$\mathbf{d}(X_k) = \mathbf{t}(X_{k-1}) \setminus \mathbf{t}(X_k)$$

Consider two k-itemsets $X_a = \{x_1, \ldots, x_{k-1}, x_a\}$ and $X_b = \{x_1, \ldots, x_{k-1}, x_b\}$ that share the common $(k-1)$-itemset $X = \{x_1, x_2, \ldots, x_{k-1}\}$ as a prefix. The diffset of $X_{ab} = X_a \cup X_b = \{x_1, \ldots, x_{k-1}, x_a, x_b\}$ is given as

$$\mathbf{d}(X_{ab}) = \mathbf{t}(X_a) \setminus \mathbf{t}(X_{ab}) = \mathbf{t}(X_a) \setminus \mathbf{t}(X_b) \tag{8.3}$$

However, note that

$$\mathbf{t}(X_a) \setminus \mathbf{t}(X_b) = \mathbf{t}(X_a) \cap \overline{\mathbf{t}(X_b)}$$

and taking the union of the above with the emptyset $\mathbf{t}(X) \cap \overline{\mathbf{t}(X)}$, we can obtain an expression for $\mathbf{d}(X_{ab})$ in terms of $\mathbf{d}(X_a)$ and $\mathbf{d}(X_b)$ as follows:

$$\begin{aligned}
\mathbf{d}(X_{ab}) &= \mathbf{t}(X_a) \setminus \mathbf{t}(X_b) \\
&= \mathbf{t}(X_a) \cap \overline{\mathbf{t}(X_b)} \\
&= \big(\mathbf{t}(X_a) \cap \overline{\mathbf{t}(X_b)}\big) \cup \big(\mathbf{t}(X) \cap \overline{\mathbf{t}(X)}\big) \\
&= \Big(\big(\mathbf{t}(X_a) \cup \mathbf{t}(X)\big) \cap \big(\overline{\mathbf{t}(X_b)} \cup \overline{\mathbf{t}(X)}\big)\Big) \cap \big(\mathbf{t}(X_a) \cup \overline{\mathbf{t}(X)}\big) \cap \big(\overline{\mathbf{t}(X_b)} \cup \mathbf{t}(X)\big) \\
&= \big(\mathbf{t}(X) \cap \overline{\mathbf{t}(X_b)}\big) \cap \overline{\big(\mathbf{t}(X) \cap \overline{\mathbf{t}(X_a)}\big)} \cap \mathcal{T} \\
&= \mathbf{d}(X_b) \setminus \mathbf{d}(X_a)
\end{aligned}$$

Thus, the diffset of X_{ab} can be obtained from the diffsets of its subsets X_a and X_b, which means that we can replace all intersection operations in Eclat with diffset operations. Using diffsets the support of a candidate itemset can be obtained by subtracting the diffset size from the support of the prefix itemset:

$$sup(X_{ab}) = sup(X_a) - |\mathbf{d}(X_{ab})|$$

which follows directly from Eq. (8.3).

The variant of Eclat that uses the diffset optimization is called dEclat, whose pseudo-code is shown in Algorithm 8.4. The input comprises all the frequent single items $i \in \mathcal{I}$ along with their diffsets, which are computed as

$$\mathbf{d}(i) = \mathbf{t}(\emptyset) \setminus \mathbf{t}(i) = \mathcal{T} \setminus \mathbf{t}(i)$$

Given an equivalence class P, for each pair of distinct itemsets X_a and X_b we generate the candidate pattern $X_{ab} = X_a \cup X_b$ and check whether it is frequent via the use of diffsets (lines 6–7). Recursive calls are made to find further extensions. It is important

ALGORITHM 8.4. Algorithm dECLAT

```
// Initial Call: F ← ∅,
    P ← {⟨i, d(i), sup(i)⟩ | i ∈ I, d(i) = T \ t(i), sup(i) ≥ minsup}
dECLAT (P, minsup, F):
1  foreach ⟨X_a, d(X_a), sup(X_a)⟩ ∈ P do
2      F ← F ∪ {(X_a, sup(X_a))}
3      P_a ← ∅
4      foreach ⟨X_b, d(X_b), sup(X_b)⟩ ∈ P, with X_b > X_a do
5          X_ab = X_a ∪ X_b
6          d(X_ab) = d(X_b) \ d(X_a)
7          sup(X_ab) = sup(X_a) − |d(X_ab)|
8          if sup(X_ab) ≥ minsup then
9              P_a ← P_a ∪ {⟨X_ab, d(X_ab), sup(X_ab)⟩}
10     if P_a ≠ ∅ then dECLAT (P_a, minsup, F)
```

to note that the switch from tidsets to diffsets can be made during any recursive call to the method. In particular, if the initial tidsets have small cardinality, then the initial call should use tidset intersections, with a switch to diffsets starting with 2-itemsets. Such optimizations are not described in the pseudo-code for clarity.

Example 8.10. Figure 8.6 illustrates the dEclat algorithm. Here $minsup = 3$, and the initial prefix equivalence class comprises all frequent items and their diffsets, computed as follows:

$$\mathbf{d}(A) = \mathcal{T} \setminus 1345 = 26$$
$$\mathbf{d}(B) = \mathcal{T} \setminus 123456 = \emptyset$$
$$\mathbf{d}(C) = \mathcal{T} \setminus 2456 = 13$$
$$\mathbf{d}(D) = \mathcal{T} \setminus 1356 = 24$$
$$\mathbf{d}(E) = \mathcal{T} \setminus 12345 = 6$$

where $\mathcal{T} = 123456$. To process candidates with A as a prefix, dEclat computes the diffsets for AB, AC, AD and AE. For instance, the diffsets of AB and AC are given as

$$\mathbf{d}(AB) = \mathbf{d}(B) \setminus \mathbf{d}(A) = \emptyset \setminus \{2, 6\} = \emptyset$$
$$\mathbf{d}(AC) = \mathbf{d}(C) \setminus \mathbf{d}(A) = \{1, 3\} \setminus \{2, 6\} = 13$$

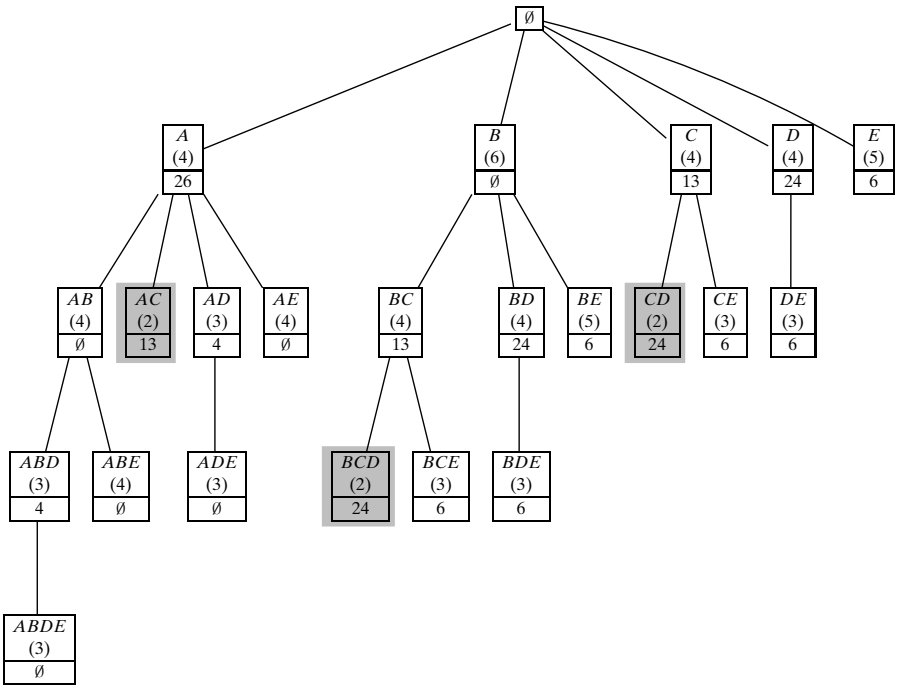


Figure 8.6. dEclat algorithm: diffsets (gray boxes indicate infrequent itemsets).

and their support values are

$$sup(AB) = sup(A) - |\mathbf{d}(AB)| = 4 - 0 = 4$$
$$sup(AC) = sup(A) - |\mathbf{d}(AC)| = 4 - 2 = 2$$

Whereas AB is frequent, we can prune AC because it is not frequent. The frequent itemsets and their diffsets and support values comprise the new prefix equivalence class:

$$P_A = \{\langle AB, \emptyset, 4\rangle, \langle AD, 4, 3\rangle, \langle AE, \emptyset, 4\rangle\}$$

which is recursively processed. Other branches are processed in a similar manner. The entire search space for dEclat is shown in Figure 8.6. The support of an itemset is shown within brackets. For example, A has support 4 and diffset $\mathbf{d}(A) = 26$.

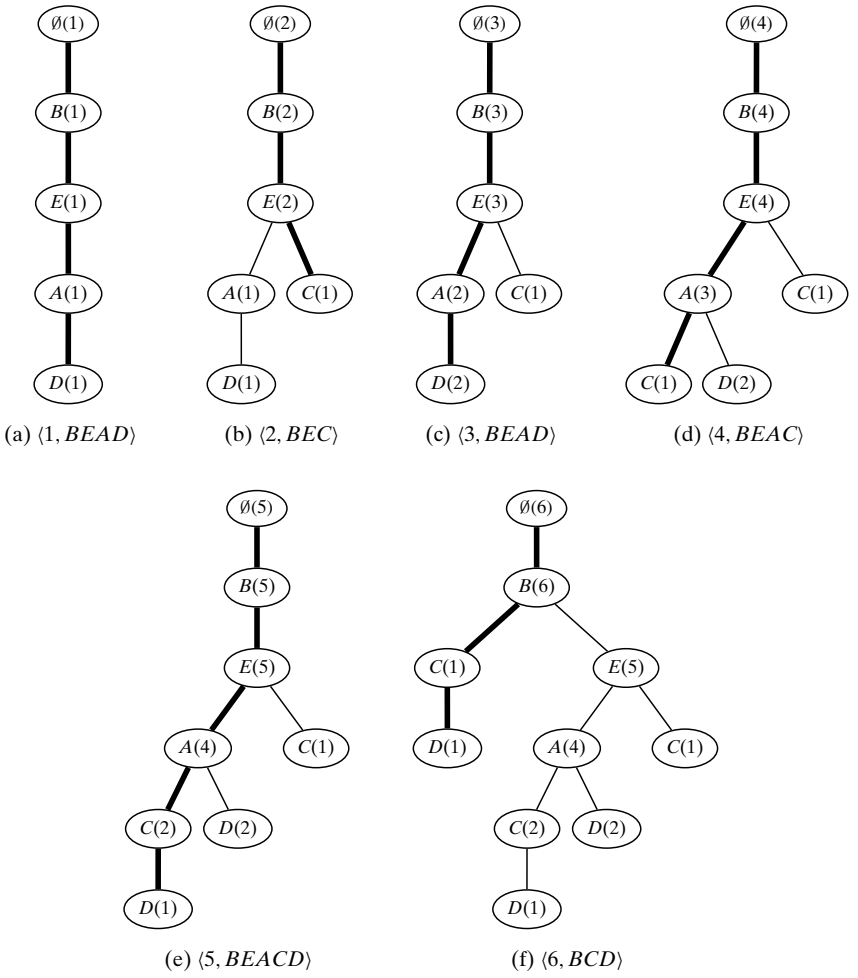
8.2.3 Frequent Pattern Tree Approach: FPGrowth Algorithm

The FPGrowth method indexes the database for fast support computation via the use of an augmented prefix tree called the *frequent pattern tree* (FP-tree). Each node in the tree is labeled with a single item, and each child node represents a different item. Each node also stores the support information for the itemset comprising the items on the path from the root to that node. The FP-tree is constructed as follows. Initially the tree contains as root the null item $\emptyset$. Next, for each tuple $\langle t, X\rangle \in \mathbf{D}$, where $X = \mathbf{i}(t)$, we insert the itemset X into the FP-tree, incrementing the count of all nodes along the path that represents X. If X shares a prefix with some previously inserted transaction, then X will follow the same path until the common prefix. For the remaining items in X, new nodes are created under the common prefix, with counts initialized to 1. The FP-tree is complete when all transactions have been inserted.

The FP-tree can be considered as a prefix compressed representation of $\mathbf{D}$. Because we want the tree to be as compact as possible, we want the most frequent items to be at the top of the tree. FPGrowth therefore reorders the items in decreasing order of support, that is, from the initial database, it first computes the support of all single items $i \in \mathcal{I}$. Next, it discards the infrequent items, and sorts the frequent items by decreasing support. Finally, each tuple $\langle t, X\rangle \in \mathbf{D}$ is inserted into the FP-tree after reordering X by decreasing item support.

Example 8.11. Consider the example database in Figure 8.1. We add each transaction one by one into the FP-tree, and keep track of the count at each node. For our example database the sorted item order is $\{B(6), E(5), A(4), C(4), D(4)\}$. Next, each transaction is reordered in this same order; for example, $\langle 1, ABDE\rangle$ becomes $\langle 1, BEAD\rangle$. Figure 8.7 illustrates step-by-step FP-tree construction as each sorted transaction is added to it. The final FP-tree for the database is shown in Figure 8.7f.

Once the FP-tree has been constructed, it serves as an index in lieu of the original database. All frequent itemsets can be mined from the tree directly via the FPGROWTH method, whose pseudo-code is shown in Algorithm 8.5. The method accepts as input a FP-tree R constructed from the input database $\mathbf{D}$, and the current itemset prefix P, which is initially empty.



(a) $\langle 1, BEAD \rangle$ (b) $\langle 2, BEC \rangle$ (c) $\langle 3, BEAD \rangle$ (d) $\langle 4, BEAC \rangle$

(e) $\langle 5, BEACD \rangle$ (f) $\langle 6, BCD \rangle$

Figure 8.7. Frequent pattern tree: bold edges indicate current transaction.

Given a FP-tree R, projected FP-trees are built for each frequent item i in R in increasing order of support. To project R on item i, we find all the occurrences of i in the tree, and for each occurrence, we determine the corresponding path from the root to i (line 13). The count of item i on a given path is recorded in $cnt(i)$ (line 14), and the path is inserted into the new projected tree R_X, where X is the itemset obtained by extending the prefix P with the item i. While inserting the path, the count of each node in R_X along the given path is incremented by the path count $cnt(i)$. We omit the item i from the path, as it is now part of the prefix. The resulting FP-tree is a projection of the itemset X that comprises the current prefix extended with item i (line 9). We then call FPGROWTH recursively with projected FP-tree R_X and the new prefix itemset X as the parameters (line 16). The base case for the recursion happens when the input FP-tree R is a single path. FP-trees that are paths are handled by enumerating all itemsets that are subsets of the path, with the support of each such itemset being given by the least frequent item in it (lines 2–6).

ALGORITHM 8.5. Algorithm FPGROWTH

```
// Initial Call: R ← FP-tree(D), P ← ∅, F ← ∅
FPGROWTH (R, P, F, minsup):
1  Remove infrequent items from R
2  if ISPATH(R) then // insert subsets of R into F
3      foreach Y ⊆ R do
4          X ← P ∪ Y
5          sup(X) ← min_{x∈Y}{cnt(x)}
6          F ← F ∪ {(X, sup(X))}
7  else  // process projected FP-trees for each frequent item i
8      foreach i ∈ R in increasing order of sup(i) do
9          X ← P ∪ {i}
10         sup(X) ← sup(i) // sum of cnt(i) for all nodes labeled i
11         F ← F ∪ {(X, sup(X))}
12         R_X ← ∅ // projected FP-tree for X
13         foreach path ∈ PATHFROMROOT(i) do
14             cnt(i) ← count of i in path
15             Insert path, excluding i, into FP-tree R_X with count cnt(i)
16         if R_X ≠ ∅ then FPGROWTH (R_X, X, F, minsup)
```

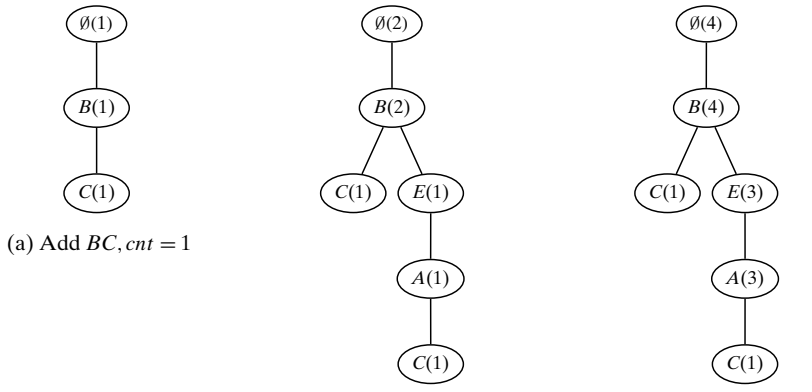
Example 8.12. We illustrate the FPGrowth method on the FP-tree R built in Example 8.11, as shown in Figure 8.7f. Let $minsup = 3$. The initial prefix is $P = \emptyset$, and the set of frequent items i in R are $B(6)$, $E(5)$, $A(4)$, $C(4)$, and $D(4)$. FPGrowth creates a projected FP-tree for each item, but in increasing order of support.

The projected FP-tree for item D is shown in Figure 8.8c. Given the initial FP-tree R shown in Figure 8.7f, there are three paths from the root to a node labeled D, namely

$$\begin{aligned} BCD, &\quad cnt(D) = 1 \\ BEACD, &\quad cnt(D) = 1 \\ BEAD, &\quad cnt(D) = 2 \end{aligned}$$

These three paths, excluding the last item $i = D$, are inserted into the new FP-tree R_D with the counts incremented by the corresponding $cnt(D)$ values, that is, we insert into R_D, the paths BC with count of 1, $BEAC$ with count of 1, and finally BEA with count of 2, as shown in Figures 8.8a–c. The projected FP-tree for D is shown in Figure 8.8c, which is processed recursively.

When we process R_D, we have the prefix itemset $P = D$, and after removing the infrequent item C (which has support 2), we find that the resulting FP-tree is a single path $B(4)$–$E(3)$–$A(3)$. Thus, we enumerate all subsets of this path and prefix them



(a) Add $BC, cnt = 1$

(b) Add $BEAC, cnt = 1$

(c) Add $BEA, cnt = 2$

Figure 8.8. Projected frequent pattern tree for D.

with D, to obtain the frequent itemsets $DB(4)$, $DE(3)$, $DA(3)$, $DBE(3)$, $DBA(3)$, $DEA(3)$, and $DBEA(3)$. At this point the call from D returns.

In a similar manner, we process the remaining items at the top level. The projected trees for C, A, and E are all single-path trees, allowing us to generate the frequent itemsets $\{CB(4), CE(3), CBE(3)\}$, $\{AE(4), AB(4), AEB(4)\}$, and $\{EB(5)\}$, respectively. This process is illustrated in Figure 8.9.

8.3 GENERATING ASSOCIATION RULES

Given a collection of frequent itemsets $\mathcal{F}$, to generate association rules we iterate over all itemsets $Z \in \mathcal{F}$, and calculate the confidence of various rules that can be derived from the itemset. Formally, given a frequent itemset $Z \in \mathcal{F}$, we look at all proper subsets $X \subset Z$ to compute rules of the form

$$X \xrightarrow{s,c} Y, \text{ where } Y = Z \setminus X$$

where $Z \setminus X = Z - X$. The rule must be frequent because

$$s = sup(XY) = sup(Z) \geq minsup$$

Thus, we have to only check whether the rule confidence satisfies the *minconf* threshold. We compute the confidence as follows:

$$c = \frac{sup(X \cup Y)}{sup(X)} = \frac{sup(Z)}{sup(X)}$$

If $c \geq minconf$, then the rule is a strong rule. On the other hand, if $conf(X \longrightarrow Y) < c$, then $conf(W \longrightarrow Z \setminus W) < c$ for all subsets $W \subset X$, as $sup(W) \geq sup(X)$. We can thus avoid checking subsets of X.

Algorithm 8.6 shows the pseudo-code for the association rule mining algorithm. For each frequent itemset $Z \in \mathcal{F}$, with size at least 2, we initialize the set of antecedents

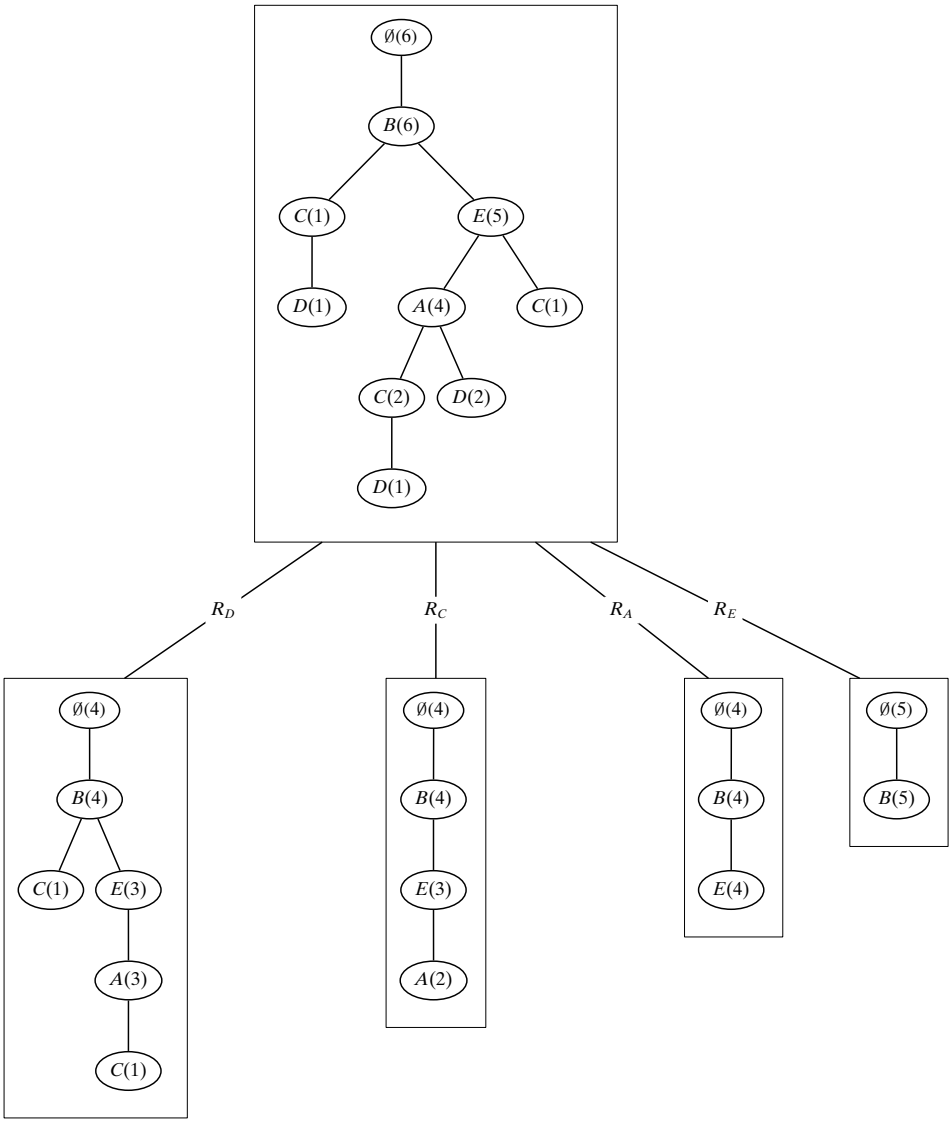


Figure 8.9. FPGrowth algorithm: frequent pattern tree projection.

$\mathcal{A}$ with all the nonempty subsets of Z (line 2). For each $X \in \mathcal{A}$ we check whether the confidence of the rule $X \longrightarrow Z \setminus X$ is at least *minconf* (line 7). If so, we output the rule. Otherwise, we remove all subsets $W \subset X$ from the set of possible antecedents (line 10).

Example 8.13. Consider the frequent itemset $ABDE(3)$ from Table 8.1, whose support is shown within the brackets. Assume that $minconf = 0.9$. To generate strong association rules we initialize the set of antecedents to

$$\mathcal{A} = \{ABD(3), ABE(4), ADE(3), BDE(3), AB(3), AD(4), AE(4),$$
$$BD(4), BE(5), DE(3), A(4), B(6), D(4), E(5)\}$$

ALGORITHM 8.6. Algorithm AssociationRules

```
AssociationRules (F, minconf):
1  foreach Z ∈ F, such that |Z| ≥ 2 do
2      A ← {X | X ⊂ Z, X ≠ ∅}
3      while A ≠ ∅ do
4          X ← maximal element in A
5          A ← A \ X// remove X from A
6          c ← sup(Z)/sup(X)
7          if c ≥ minconf then
8              print X ⟶ Y, sup(Z), c
9          else
10             A ← A \ {W | W ⊂ X} // remove all subsets of X from A
```

The first subset is $X = ABD$, and the confidence of $ABD \longrightarrow E$ is $3/3 = 1.0$, so we output it. The next subset is $X = ABE$, but the corresponding rule $ABE \longrightarrow D$ is not strong since $conf(ABE \longrightarrow D) = 3/4 = 0.75$. We can thus remove from $\mathcal{A}$ all subsets of ABE; the updated set of antecedents is therefore

$$\mathcal{A} = \{ADE(3), BDE(3), AD(4), BD(4), DE(3), D(4)\}$$

Next, we select $X = ADE$, which yields a strong rule, and so do $X = BDE$ and $X = AD$. However, when we process $X = BD$, we find that $conf(BD \longrightarrow AE) = 3/4 = 0.75$, and thus we can prune all subsets of BD from $\mathcal{A}$, to yield

$$\mathcal{A} = \{DE(3)\}$$

The last rule to be tried is $DE \longrightarrow AB$ which is also strong. The final set of strong rules that are output are as follows:

$$ABD \longrightarrow E, conf = 1.0$$
$$ADE \longrightarrow B, conf = 1.0$$
$$BDE \longrightarrow A, conf = 1.0$$
$$AD \longrightarrow BE, conf = 1.0$$
$$DE \longrightarrow AB, conf = 1.0$$

8.4 FURTHER READING

The association rule mining problem was introduced in Agrawal, Imieliński, and Swami (1993). The Apriori method was proposed in Agrawal and Srikant (1994), and a similar approach was outlined independently in Mannila, Toivonen, and Verkamo

(1994). The tidlist intersection based Eclat method is described in Zaki et al. (1997), and the dEclat approach that uses diffset appears in Zaki and Gouda (2003). Finally, the FPGrowth algorithm is described in Han, Pei, and Yin (2000). For an experimental comparison between several of the frequent itemset mining algorithms see Goethals and Zaki (2004). There is a very close connection between itemset mining and association rules, and formal concept analysis (Ganter, Wille, and Franzke, 1997). For example, association rules can be considered to be *partial implications* (Luxenburger, 1991) with frequency constraints.

Agrawal, R., Imieliński, T., and Swami, A. (May 1993). "Mining association rules between sets of items in large databases." *In Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM.

Agrawal, R. and Srikant, R. (Sept. 1994). "Fast algorithms for mining association rules." *In Proceedings of the 20th International Conference on Very Large Data Bases*, pp. 487–499.

Ganter, B., Wille, R., and Franzke, C. (1997). *Formal Concept Analysis: Mathematical Foundations.* New York: Springer-Verlag.

Goethals, B. and Zaki, M. J. (2004). "Advances in frequent itemset mining implementations: report on FIMI'03." *ACM SIGKDD Explorations*, 6 (1): 109–117.

Han, J., Pei, J., and Yin, Y. (May 2000). "Mining frequent patterns without candidate generation." *In Proceedings of the ACM SIGMOD International Conference on Management of Data,* ACM.

Luxenburger, M. (1991). "Implications partielles dans un contexte." *Mathématiques et Sciences Humaines*, 113: 35–55.

Mannila, H., Toivonen, H., and Verkamo, I. A. (1994). Efficient algorithms for discovering association rules. *In Proceedings of the AAAI Workshop on Knowledge Discovery in Databases*, AAAI Press.

Zaki, M. J. and Gouda, K. (2003). "Fast vertical mining using diffsets." *In Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, pp. 326–335.

Zaki, M. J., Parthasarathy, S., Ogihara, M., and Li, W. (1997). "New algorithms for fast discovery of association rules." *In Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, pp. 283–286.

8.5 EXERCISES

Q1. Given the database in Table 8.2.

- **(a)** Using $minsup = 3/8$, show how the Apriori algorithm enumerates all frequent patterns from this dataset.
- **(b)** With $minsup = 2/8$, show how FPGrowth enumerates the frequent itemsets.

Q2. Consider the vertical database shown in Table 8.3. Assuming that $minsup = 3$, enumerate all the frequent itemsets using the Eclat method.

Table 8.2. Transaction database for Q1

tid	itemset
t_1	$ABCD$
t_2	$ACDF$
t_3	$ACDEG$
t_4	$ABDF$
t_5	BCG
t_6	DFG
t_7	ABG
t_8	$CDFG$

Table 8.3. Dataset for Q2

A	B	C	D	E
1	2	1	1	2
3	3	2	6	3
5	4	3		4
6	5	5		5
	6	6		

Q3. Given two k-itemsets $X_a = \{x_1, \ldots, x_{k-1}, x_a\}$ and $X_b = \{x_1, \ldots, x_{k-1}, x_b\}$ that share the common $(k-1)$-itemset $X = \{x_1, x_2, \ldots, x_{k-1}\}$ as a prefix, prove that

$$sup(X_{ab}) = sup(X_a) - |\mathbf{d}(X_{ab})|$$

where $X_{ab} = X_a \cup X_b$, and $\mathbf{d}(X_{ab})$ is the diffset of X_{ab}.

Q4. Given the database in Table 8.4. Show all rules that one can generate from the set ABE.

Table 8.4. Dataset for Q4

tid	itemset
t_1	ACD
t_2	BCE
t_3	$ABCE$
t_4	BDE
t_5	$ABCE$
t_6	$ABCD$

Q5. Consider the *partition* algorithm for itemset mining. It divides the database into k partitions, not necessarily equal, such that $\mathbf{D} = \cup_{i=1}^{k} \mathbf{D}_i$, where $\mathbf{D}_i$ is partition i, and for any $i \neq j$, we have $\mathbf{D}_i \cap \mathbf{D}_j = \emptyset$. Also let $n_i = |\mathbf{D}_i|$ denote the number of transactions in partition $\mathbf{D}_i$. The algorithm first mines only locally frequent itemsets, that is, itemsets whose relative support is above the *minsup* threshold specified as a fraction. In the second step, it takes the union of all locally frequent itemsets, and computes their support in the entire database $\mathbf{D}$ to determine which of them are globally frequent. Prove that if a pattern is globally frequent in the database, then it must be locally frequent in at least one partition.

Q6. Consider Figure 8.10. It shows a simple taxonomy on some food items. Each leaf is a simple item and an internal node represents a higher-level category or item. Each item (single or high-level) has a unique integer label noted under it. Consider the database composed of the simple items shown in Table 8.5 Answer the following questions:

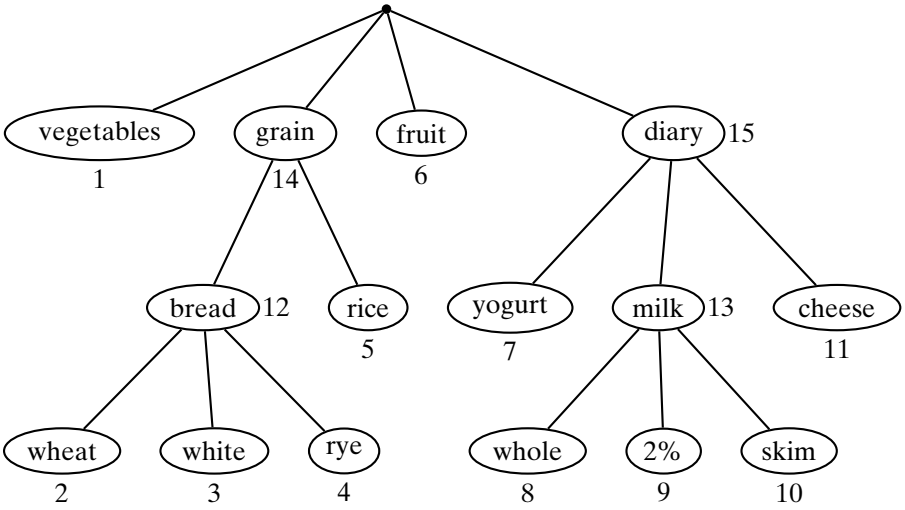


Figure 8.10. Item taxonomy for Q6.

Table 8.5. Dataset for Q6

tid	itemset
1	2 3 6 7
2	1 3 4 8 11
3	3 9 11
4	1 5 6 7
5	1 3 8 10 11
6	3 5 7 9 11
7	4 6 8 10 11
8	1 3 5 8 11

(a) What is the size of the itemset search space if one restricts oneself to only itemsets composed of simple items?

(b) Let $X = \{x_1, x_2, \ldots, x_k\}$ be a frequent itemset. Let us replace some $x_i \in X$ with its parent in the taxonomy (provided it exists) to obtain X', then the support of the new itemset X' is:

- **i.** more than support of X
- **ii.** less than support of X
- **iii.** not equal to support of X
- **iv.** more than or equal to support of X
- **v.** less than or equal to support of X

(c) Use $minsup = 7/8$. Find all frequent itemsets composed only of high-level items in the taxonomy. Keep in mind that if a simple item appears in a transaction, then its high-level ancestors are all assumed to occur in the transaction as well.

Q7. Let $\mathbf{D}$ be a database with n transactions. Consider a sampling approach for mining frequent itemsets, where we extract a random sample $\mathbf{S} \subset \mathbf{D}$, with say m transactions, and we mine all the frequent itemsets in the sample, denoted as $\mathcal{F}_S$. Next, we make one complete scan of $\mathbf{D}$, and for each $X \in \mathcal{F}_S$, we find its actual support in the whole database. Some of the itemsets in the sample may not be truly frequent in the database; these are the false positives. Also, some of the true frequent itemsets in the original database may never be present in the sample at all; these are the false negatives.

Prove that if X is a false negative, then this case can be detected by counting the support in $\mathbf{D}$ for every itemset belonging to the *negative border* of $\mathcal{F}_S$, denoted $Bd^-(\mathcal{F}_S)$, which is defined as the set of minimal infrequent itemsets in sample $\mathbf{S}$. Formally,

$$Bd^-(\mathcal{F}_S) = \inf\{Y \mid sup(Y) < minsup \text{ and } \forall Z \subset Y, sup(Z) \geq minsup\}$$

where inf returns the minimal elements of the set.

Q8. Assume that we want to mine frequent patterns from relational tables. For example consider Table 8.6, with three attributes A, B, and C, and six records. Each attribute has a domain from which it draws its values, for example, the domain of A is $dom(A) = \{a_1, a_2, a_3\}$. Note that no record can have more than one value of a given attribute.

Table 8.6. Data for Q8

tid	A	B	C
1	a_1	b_1	c_1
2	a_2	b_3	c_2
3	a_2	b_3	c_3
4	a_2	b_1	c_1
5	a_2	b_3	c_3
6	a_3	b_3	c_3

We define a *relational pattern* P over some k attributes $X_1, X_2, \ldots, X_k$ to be a subset of the Cartesian product of the domains of the attributes, i.e., $P \subseteq dom(X_1) \times dom(X_2) \times \cdots \times dom(X_k)$. That is, $P = P_1 \times P_2 \times \cdots \times P_k$, where each $P_i \subseteq dom(X_i)$. For example, $\{a_1, a_2\} \times \{c_1\}$ is a possible pattern over attributes A and C, whereas $\{a_1\} \times \{b_1\} \times \{c_1\}$ is another pattern over attributes A, B and C.

The support of relational pattern $P = P_1 \times P_2 \times \cdots \times P_k$ in dataset $\mathbf{D}$ is defined as the number of records in the dataset that belong to it; it is given as

$$sup(P) = \big|\{r = (r_1, r_2, \ldots, r_n) \in \mathbf{D} : r_i \in P_i \text{ for all } P_i \text{ in } P\}\big|$$

For example, $sup(\{a_1, a_2\} \times \{c_1\}) = 2$, as both records 1 and 4 contribute to its support. Note, however that the pattern $\{a_1\} \times \{c_1\}$ has a support of 1, since only record 1 belongs to it. Thus, relational patterns **do not** satisfy the Apriori property that we

used for frequent itemsets, that is, subsets of a frequent relational pattern can be infrequent.

We call a relational pattern $P = P_1 \times P_2 \times \cdots \times P_k$ over attributes $X_1, \ldots, X_k$ as *valid* iff for all $u \in P_i$ and all $v \in P_j$, the pair of values $(X_i = u, X_j = v)$ occurs together in some record. For example, $\{a_1, a_2\} \times \{c_1\}$ is a valid pattern since both $(A = a_1, C = c_1)$ and $(A = a_2, C = c_1)$ occur in some records (namely, records 1 and 4, respectively), whereas $\{a_1, a_2\} \times \{c_2\}$ is not a valid pattern, since there is no record that has the values $(A = a_1, C = c_2)$. Thus, for a pattern to be valid every pair of values in P from distinct attributes must belong to some record.

Given that $minsup = 2$, find all frequent, valid, relational patterns in the dataset in Table 8.6.

Q9. Given the following multiset dataset:

tid	multiset
1	$ABCA$
2	$ABABA$
3	$CABBA$

Using $minsup = 2$, answer the following:

(a) Find all frequent multisets. Recall that a multiset is still a set (i.e., order is not important), but it allows multiple occurrences of an item.

(b) Find all minimal infrequent multisets, that is, those multisets that have no infrequent sub-multisets.

CHAPTER 9 Summarizing Itemsets

The search space for frequent itemsets is usually very large and it grows exponentially with the number of items. In particular, a low minimum support value may result in an intractable number of frequent itemsets. An alternative approach, studied in this chapter, is to determine condensed representations of the frequent itemsets that summarize their essential characteristics. The use of condensed representations can not only reduce the computational and storage demands, but it can also make it easier to analyze the mined patterns. In this chapter we discuss three of these representations: closed, maximal, and nonderivable itemsets.

9.1 MAXIMAL AND CLOSED FREQUENT ITEMSETS

Given a binary database $\mathbf{D} \subseteq \mathcal{T} \times \mathcal{I}$, over the tids $\mathcal{T}$ and items $\mathcal{I}$, let $\mathcal{F}$ denote the set of all frequent itemsets, that is,

$$\mathcal{F} = \{X \mid X \subseteq \mathcal{I} \text{ and } sup(X) \geq minsup\}$$

Maximal Frequent Itemsets

A frequent itemset $X \in \mathcal{F}$ is called *maximal* if it has no frequent supersets. Let $\mathcal{M}$ be the set of all maximal frequent itemsets, given as

$$\mathcal{M} = \{X \mid X \in \mathcal{F} \text{ and } \nexists Y \supset X, \text{ such that } Y \in \mathcal{F}\}$$

The set $\mathcal{M}$ is a condensed representation of the set of all frequent itemset $\mathcal{F}$, because we can determine whether any itemset X is frequent or not using $\mathcal{M}$. If there exists a maximal itemset Z such that $X \subseteq Z$, then X must be frequent; otherwise X cannot be frequent. On the other hand, we cannot determine $sup(X)$ using $\mathcal{M}$ alone, although we can lower-bound it, that is, $sup(X) \geq sup(Z)$ if $X \subseteq Z \in \mathcal{M}$.

Example 9.1. Consider the dataset given in Figure 9.1a. Using any of the algorithms discussed in Chapter 8 and $minsup = 3$, we obtain the frequent itemsets shown in Figure 9.1b. Notice that there are 19 frequent itemsets out of the $2^5 - 1 = 31$ possible nonempty itemsets. Out of these, there are only two maximal itemsets,

Tid	Itemset
1	$ABDE$
2	BCE
3	$ABDE$
4	$ABCE$
5	$ABCDE$
6	BCD

(a) Transaction database

sup	Itemsets
6	B
5	E, BE
4	$A, C, D, AB, AE, BC, BD, ABE$
3	$AD, CE, DE, ABD, ADE, BCE, BDE, ABDE$

(b) Frequent itemsets ($minsup = 3$)

Figure 9.1. An example database.

$ABDE$ and BCE. Any other frequent itemset must be a subset of one of the maximal itemsets. For example, we can determine that ABE is frequent, since $ABE \subset ABDE$, and we can establish that $sup(ABE) \geq sup(ABDE) = 3$.

Closed Frequent Itemsets

Recall that the function $\mathbf{t}: 2^{\mathcal{I}} \rightarrow 2^{\mathcal{T}}$ [Eq. (8.2)] maps itemsets to tidsets, and the function $\mathbf{i}: 2^{\mathcal{T}} \rightarrow 2^{\mathcal{I}}$ [Eq. (8.1)] maps tidsets to itemsets. That is, given $T \subseteq \mathcal{T}$, and $X \subseteq \mathcal{I}$, we have

$$\mathbf{t}(X) = \{t \in \mathcal{T} \mid t \text{ contains } X\}$$

$$\mathbf{i}(T) = \{x \in \mathcal{I} \mid \forall t \in T,\ t \text{ contains } x\}$$

Define by $\mathbf{c}: 2^{\mathcal{I}} \rightarrow 2^{\mathcal{I}}$ the *closure operator*, given as

$$\mathbf{c}(X) = \mathbf{i} \circ \mathbf{t}(X) = \mathbf{i}(\mathbf{t}(X))$$

The closure operator $\mathbf{c}$ maps itemsets to itemsets, and it satisfies the following three properties:

- *Extensive*: $X \subseteq \mathbf{c}(X)$
- *Monotonic*: If $X_i \subseteq X_j$, then $\mathbf{c}(X_i) \subseteq \mathbf{c}(X_j)$
- *Idempotent*: $\mathbf{c}(\mathbf{c}(X)) = \mathbf{c}(X)$

An itemset X is called *closed* if $\mathbf{c}(X) = X$, that is, if X is a fixed point of the closure operator $\mathbf{c}$. On the other hand, if $X \neq \mathbf{c}(X)$, then X is not closed, but the set $\mathbf{c}(X)$ is called its closure. From the properties of the closure operator, both X and $\mathbf{c}(X)$ have the same tidset. It follows that a frequent set $X \in \mathcal{F}$ is closed if it has no frequent superset *with the same frequency* because by definition, it is the largest itemset common to all the tids in the tidset $\mathbf{t}(X)$. The set of all closed frequent itemsets is thus defined as

$$\mathcal{C} = \left\{X \mid X \in \mathcal{F} \text{ and } \nexists Y \supset X \text{ such that } sup(X) = sup(Y)\right\} \tag{9.1}$$

Put differently, X is closed if all supersets of X have strictly less support, that is, $sup(X) > sup(Y)$, for all $Y \supset X$.

The set of all closed frequent itemsets $\mathcal{C}$ is a condensed representation, as we can determine whether an itemset X is frequent, as well as the exact support of X using $\mathcal{C}$ alone. The itemset X is frequent if there exists a closed frequent itemset $Z \in \mathcal{C}$ such that $X \subseteq Z$. Further, the support of X is given as

$$sup(X) = \max\{sup(Z) | Z \in \mathcal{C}, X \subseteq Z\}$$

The following relationship holds between the set of all, closed, and maximal frequent itemsets:

$$\mathcal{M} \subseteq \mathcal{C} \subseteq \mathcal{F}$$

Minimal Generators

A frequent itemset X is a *minimal generator* if it has no subsets with the same support:

$$\mathcal{G} = \{X \mid X \in \mathcal{F} \text{ and } \nexists Y \subset X, \text{ such that } sup(X) = sup(Y)\}$$

In other words, all subsets of X have strictly higher support, that is, $sup(X) < sup(Y)$, for all $Y \subset X$. The concept of minimum generator is closely related to the notion of closed itemsets. Given an equivalence class of itemsets that have the same tidset, a closed itemset is the unique maximum element of the class, whereas the minimal generators are the minimal elements of the class.

Example 9.2. Consider the example dataset in Figure 9.1a. The frequent closed (as well as maximal) itemsets using $minsup = 3$ are shown in Figure 9.2. We can see, for instance, that the itemsets AD, DE, ABD, ADE, BDE, and $ABDE$, occur in the same three transactions, namely 135, and thus constitute an equivalence class. The largest itemset among these, namely $ABDE$, is the closed itemset. Using the closure operator yields the same result; we have $\mathbf{c}(AD) = \mathbf{i}(\mathbf{t}(AD)) = \mathbf{i}(135) = ABDE$, which indicates that the closure of AD is $ABDE$. To verify that $ABDE$ is closed note that $\mathbf{c}(ABDE) = \mathbf{i}(\mathbf{t}(ABDE)) = \mathbf{i}(135) = ABDE$. The minimal elements of the equivalence class, namely AD and DE, are the minimal generators. No subset of these itemsets shares the same tidset.

The set of all closed frequent itemsets, and the corresponding set of minimal generators, is as follows:

Tidset	$\mathcal{C}$	$\mathcal{G}$
1345	ABE	A
123456	B	B
1356	BD	D
12345	BE	E
2456	BC	C
135	$ABDE$	AD, DE
245	BCE	CE

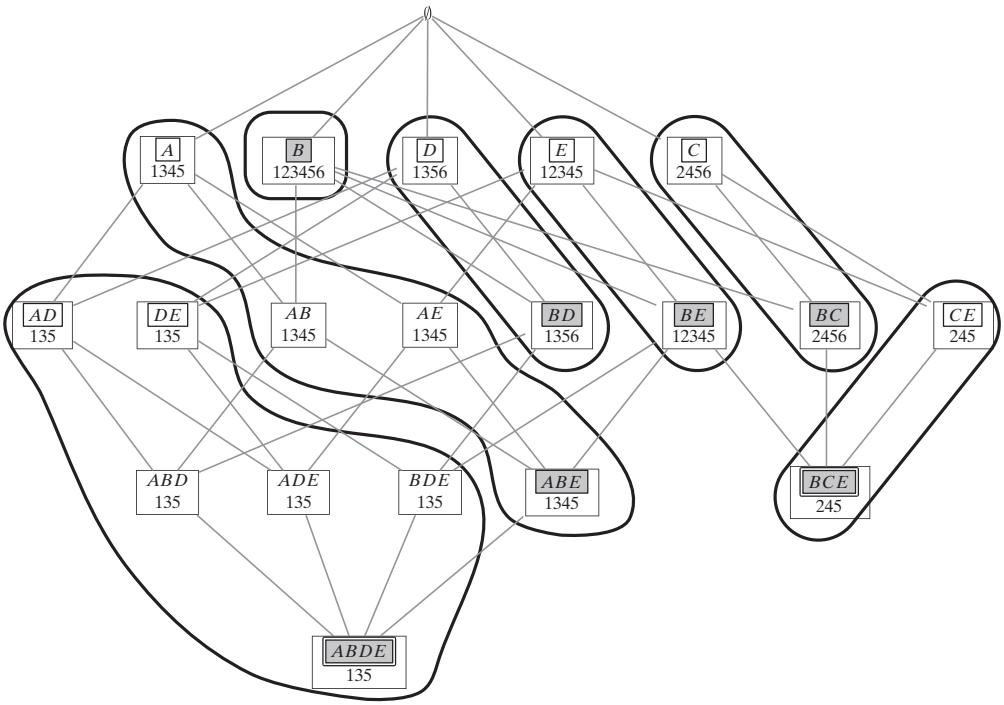


Figure 9.2. Frequent, closed, minimal generators, and maximal frequent itemsets. Itemsets that are boxed and shaded are closed, whereas those within boxes (but unshaded) are the minimal generators; maximal itemsets are shown boxed with double lines.

Out of the closed itemsets, the maximal ones are $ABDE$ and BCE. Consider itemset AB. Using $\mathcal{C}$ we can determine that

$$sup(AB) = \max\{sup(ABE), sup(ABDE)\} = \max\{4, 3\} = 4$$

9.2 MINING MAXIMAL FREQUENT ITEMSETS: GENMAX ALGORITHM

Mining maximal itemsets requires additional steps beyond simply determining the frequent itemsets. Assuming that the set of maximal frequent itemsets is initially empty, that is, $\mathcal{M} = \emptyset$, each time we generate a new frequent itemset X, we have to perform the following maximality checks

- **Subset Check:** $\nexists Y \in \mathcal{M}$, such that $X \subset Y$. If such a Y exists, then clearly X is not maximal. Otherwise, we add X to $\mathcal{M}$, as a potentially maximal itemset.
- **Superset Check:** $\nexists Y \in \mathcal{M}$, such that $Y \subset X$. If such a Y exists, then Y cannot be maximal, and we have to remove it from $\mathcal{M}$.

These two maximality checks take $O(|\mathcal{M}|)$ time, which can get expensive, especially as $\mathcal{M}$ grows; thus for efficiency reasons it is crucial to minimize the number of times these checks are performed. As such, any of the frequent itemset mining algorithms

from Chapter 8 can be extended to mine maximal frequent itemsets by adding the maximality checking steps. Here we consider the GenMax method, which is based on the tidset intersection approach of Eclat (see Section 8.2.2). We shall see that it never inserts a nonmaximal itemset into $\mathcal{M}$. It thus eliminates the superset checks and requires only subset checks to determine maximality.

Algorithm 9.1 shows the pseudo-code for GenMax. The initial call takes as input the set of frequent items along with their tidsets, $\langle i, \mathbf{t}(i)\rangle$, and the initially empty set of maximal itemsets, $\mathcal{M}$. Given a set of itemset–tidset pairs, called IT-pairs, of the form $\langle X, \mathbf{t}(X)\rangle$, the recursive GenMax method works as follows. In lines 1–3, we check if the entire current branch can be pruned by checking if the union of all the itemsets, $Y = \bigcup X_i$, is already subsumed by (or contained in) some maximal pattern $Z \in \mathcal{M}$. If so, no maximal itemset can be generated from the current branch, and it is pruned. On the other hand, if the branch is not pruned, we intersect each IT-pair $\langle X_i, \mathbf{t}(X_i)\rangle$ with all the other IT-pairs $\langle X_j, \mathbf{t}(X_j)\rangle$, with $j > i$, to generate new candidates X_{ij}, which are added to the IT-pair set P_i (lines 6–9). If P_i is not empty, a recursive call to GENMAX is made to find other potentially frequent extensions of X_i. On the other hand, if P_i is empty, it means that X_i cannot be extended, and it is potentially maximal. In this case, we add X_i to the set $\mathcal{M}$, provided that X_i is not contained in any previously added maximal set $Z \in \mathcal{M}$ (line 12). Note also that, because of this check for maximality before inserting any itemset into $\mathcal{M}$, we never have to remove any itemsets from it. In other words, all itemsets in $\mathcal{M}$ are guaranteed to be maximal. On termination of GenMax, the set $\mathcal{M}$ contains the final set of all maximal frequent itemsets. The GenMax approach also includes a number of other optimizations to reduce the maximality checks and to improve the support computations. Further, GenMax utilizes diffsets (differences of tidsets) for fast support computation, which were described in Section 8.2.2. We omit these optimizations here for clarity.

ALGORITHM 9.1. Algorithm GENMAX

```
   // Initial Call: M ← ∅, P ← {⟨i, t(i)⟩ | i ∈ I, sup(i) ≥ minsup}
   GENMAX (P, minsup, M):
1  Y ← ⋃ X_i
2  if ∃Z ∈ M, such that Y ⊆ Z then
3      return // prune entire branch
4  foreach ⟨X_i, t(X_i)⟩ ∈ P do
5      P_i ← ∅
6      foreach ⟨X_j, t(X_j)⟩ ∈ P, with j > i do
7          X_ij ← X_i ∪ X_j
8          t(X_ij) = t(X_i) ∩ t(X_j)
9          if sup(X_ij) ≥ minsup then P_i ← P_i ∪ {⟨X_ij, t(X_ij)⟩}
10     if P_i ≠ ∅ then GENMAX (P_i, minsup, M)
11     else if ∄Z ∈ M, X_i ⊆ Z then
12         M = M ∪ X_i   // add X_i to maximal set
```

Example 9.3. Figure 9.3 shows the execution of GenMax on the example database from Figure 9.1a using $minsup = 3$. Initially the set of maximal itemsets is empty. The root of the tree represents the initial call with all IT-pairs consisting of frequent single items and their tidsets. We first intersect $\mathbf{t}(A)$ with the tidsets of the other items. The set of frequent extensions from A are

$$P_A = \{\langle AB, 1345\rangle, \langle AD, 135\rangle, \langle AE, 1345\rangle\}$$

Choosing $X_i = AB$, leads to the next set of extensions, namely

$$P_{AB} = \{\langle ABD, 135\rangle, \langle ABE, 1345\rangle\}$$

Finally, we reach the left-most leaf corresponding to $P_{ABD} = \{\langle ABDE, 135\rangle\}$. At this point, we add $ABDE$ to the set of maximal frequent itemsets because it has no other extensions, so that $\mathcal{M} = \{ABDE\}$.

The search then backtracks one level, and we try to process ABE, which is also a candidate to be maximal. However, it is contained in $ABDE$, so it is pruned. Likewise, when we try to process $P_{AD} = \{\langle ADE, 135\rangle\}$ it will get pruned because it is also subsumed by $ABDE$, and similarly for AE. At this stage, all maximal itemsets starting with A have been found, and we next proceed with the B branch. The left-most B branch, namely BCE, cannot be extended further. Because BCE is not

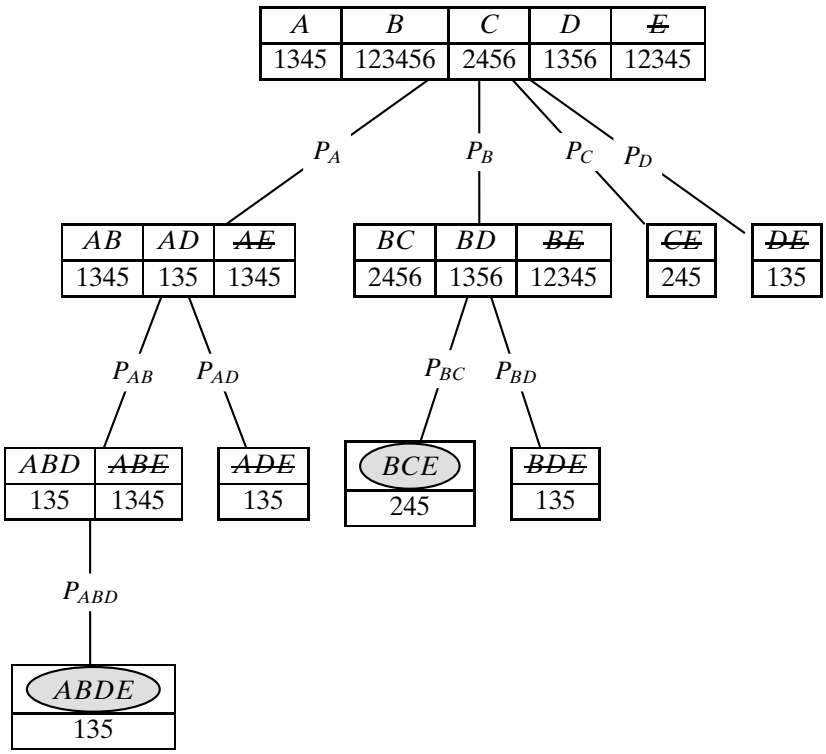


Figure 9.3. Mining maximal frequent itemsets. Maximal itemsets are shown as shaded ovals, whereas pruned branches are shown with the strike-through. Infrequent itemsets are not shown.

a subset of any maximal itemset in $\mathcal{M}$, we insert it as a maximal itemset, so that $\mathcal{M} = \{ABDE, BCE\}$. Subsequently, all remaining branches are subsumed by one of these two maximal itemsets, and are thus pruned.

9.3 MINING CLOSED FREQUENT ITEMSETS: CHARM ALGORITHM

Mining closed frequent itemsets requires that we perform closure checks, that is, whether $X = \mathbf{c}(X)$. Direct closure checking can be very expensive, as we would have to verify that X is the largest itemset common to all the tids in $\mathbf{t}(X)$, that is, $X = \bigcap_{t \in \mathbf{t}(X)} \mathbf{i}(t)$. Instead, we will describe a vertical tidset intersection based method called CHARM that performs more efficient closure checking. Given a collection of IT-pairs $\{\langle X_i, \mathbf{t}(X_i)\rangle\}$, the following three properties hold:

Property (1) If $\mathbf{t}(X_i) = \mathbf{t}(X_j)$, then $\mathbf{c}(X_i) = \mathbf{c}(X_j) = \mathbf{c}(X_i \cup X_j)$, which implies that we can replace every occurrence of X_i with $X_i \cup X_j$ and prune the branch under X_j because its closure is identical to the closure of $X_i \cup X_j$.

Property (2) If $\mathbf{t}(X_i) \subset \mathbf{t}(X_j)$, then $\mathbf{c}(X_i) \neq \mathbf{c}(X_j)$ but $\mathbf{c}(X_i) = \mathbf{c}(X_i \cup X_j)$, which means that we can replace every occurrence of X_i with $X_i \cup X_j$, but we cannot prune X_j because it generates a different closure. Note that if $\mathbf{t}(X_i) \supset \mathbf{t}(X_j)$ then we simply interchange the role of X_i and X_j.

Property (3) If $\mathbf{t}(X_i) \neq \mathbf{t}(X_j)$, then $\mathbf{c}(X_i) \neq \mathbf{c}(X_j) \neq \mathbf{c}(X_i \cup X_j)$. In this case we cannot remove either X_i or X_j, as each of them generates a different closure.

Algorithm 9.2 presents the pseudo-code for Charm, which is also based on the Eclat algorithm described in Section 8.2.2. It takes as input the set of all frequent single items along with their tidsets. Also, initially the set of all closed itemsets, $\mathcal{C}$, is empty. Given any IT-pair set $P = \{\langle X_i, \mathbf{t}(X_i)\rangle\}$, the method first sorts them in increasing order of support. For each itemset X_i we try to extend it with all other items X_j in the sorted order, and we apply the above three properties to prune branches where possible. First we make sure that $X_{ij} = X_i \cup X_j$ is frequent, by checking the cardinality of $\mathbf{t}(X_{ij})$. If yes, then we check properties 1 and 2 (lines 8 and 12). Note that whenever we replace X_i with $X_{ij} = X_i \cup X_j$, we make sure to do so in the current set P, as well as the new set P_i. Only when property 3 holds do we add the new extension X_{ij} to the set P_i (line 14). If the set P_i is not empty, then we make a recursive call to Charm. Finally, if X_i is not a subset of any closed set Z with the same support, we can safely add it to the set of closed itemsets, $\mathcal{C}$ (line 18). For fast support computation, Charm uses the diffset optimization described in Section 8.2.2; we omit it here for clarity.

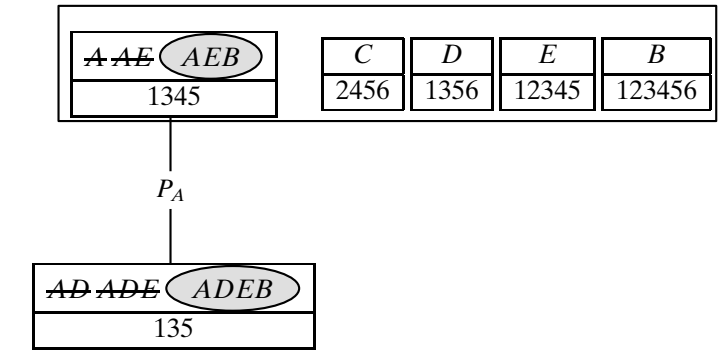
Example 9.4. We illustrate the Charm algorithm for mining frequent closed itemsets from the example database in Figure 9.1a, using *minsup* = 3. Figure 9.4 shows the sequence of steps. The initial set of IT-pairs, after support based sorting, is shown at the root of the search tree. The sorted order is A, C, D, E, and B. We first process extensions from A, as shown in Figure 9.4a. Because AC is not frequent,

ALGORITHM 9.2. Algorithm CHARM

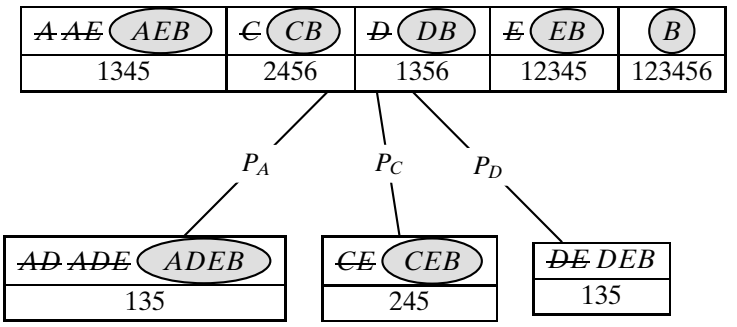
```
// Initial Call: C ← ∅, P ← {⟨i, t(i)⟩ : i ∈ I, sup(i) ≥ minsup}
CHARM (P, minsup, C):
1  Sort P in increasing order of support (i.e., by increasing |t(X_i)|)
2  foreach ⟨X_i, t(X_i)⟩ ∈ P do
3      P_i ← ∅
4      foreach ⟨X_j, t(X_j)⟩ ∈ P, with j > i do
5          X_ij = X_i ∪ X_j
6          t(X_ij) = t(X_i) ∩ t(X_j)
7          if sup(X_ij) ≥ minsup then
8              if t(X_i) = t(X_j) then // Property 1
9                  Replace X_i with X_ij in P and P_i
10                 Remove ⟨X_j, t(X_j)⟩ from P
11             else
12                 if t(X_i) ⊂ t(X_j) then // Property 2
13                     Replace X_i with X_ij in P and P_i
14                 else // Property 3
15                     P_i ← P_i ∪ {⟨X_ij, t(X_ij)⟩}
16     if P_i ≠ ∅ then CHARM (P_i, minsup, C)
17     if ∄Z ∈ C, such that X_i ⊆ Z and t(X_i) = t(Z) then
18         C = C ∪ X_i    // Add X_i to closed set
```

it is pruned. AD is frequent and because $\mathbf{t}(A) \neq \mathbf{t}(D)$, we add $\langle AD, 135\rangle$ to the set P_A (property 3). When we combine A with E, property 2 applies, and we simply replace all occurrences of A in both P and P_A with AE, which is illustrated with the strike-through. Likewise, because $\mathbf{t}(A) \subset \mathbf{t}(B)$ all current occurrences of A, actually AE, in both P and P_A are replaced by AEB. The set P_A thus contains only one itemset $\{\langle ADEB, 135\rangle\}$. When CHARM is invoked with P_A as the IT-pair, it jumps straight to line 18, and adds $ADEB$ to the set of closed itemsets $\mathcal{C}$. When the call returns, we check whether AEB can be added as a closed itemset. AEB is a subset of $ADEB$, but it does not have the same support, thus AEB is also added to $\mathcal{C}$. At this point all closed itemsets containing A have been found.

The Charm algorithm proceeds with the remaining branches as shown in Figure 9.4b. For instance, C is processed next. CD is infrequent and thus pruned. CE is frequent and it is added to P_C as a new extension (via property 3). Because $\mathbf{t}(C) \subset \mathbf{t}(B)$, all occurrences of C are replaced by CB, and $P_C = \{\langle CEB, 245\rangle\}$. CEB and CB are both found to be closed. The computation proceeds in this manner until all closed frequent itemsets are enumerated. Note that when we get to DEB and perform the closure check, we find that it is a subset of $ADEB$ and also has the same support; thus DEB is not closed.



(a) Process A



(b) Charm

Figure 9.4. Mining closed frequent itemsets. Closed itemsets are shown as shaded ovals. Strike-through represents itemsets X_i replaced by $X_i \cup X_j$ during execution of the algorithm. Infrequent itemsets are not shown.

9.4 NONDERIVABLE ITEMSETS

An itemset is called *nonderivable* if its support cannot be deduced from the supports of its subsets. The set of all frequent nonderivable itemsets is a summary or condensed representation of the set of all frequent itemsets. Further, it is lossless with respect to support, that is, the exact support of all other frequent itemsets can be deduced from it.

Generalized Itemsets

Let $\mathcal{T}$ be a set of tids, let $\mathcal{I}$ be a set of items, and let X be a k-itemset, that is, $X = \{x_1, x_2, \ldots, x_k\}$. Consider the tidsets $\mathbf{t}(x_i)$ for each item $x_i \in X$. These k-tidsets induce a partitioning of the set of all tids into 2^k regions, some of which may be empty, where each partition contains the tids for some subset of items $Y \subseteq X$, but for none of the remaining items $Z = Y \setminus X$. Each such region is therefore the tidset of a *generalized itemset* comprising items in X or their negations. As such a generalized itemset can be represented as $Y\overline{Z}$, where Y consists of regular items and Z consists of negated items. We define the support of a generalized itemset $Y\overline{Z}$ as the number of transactions that

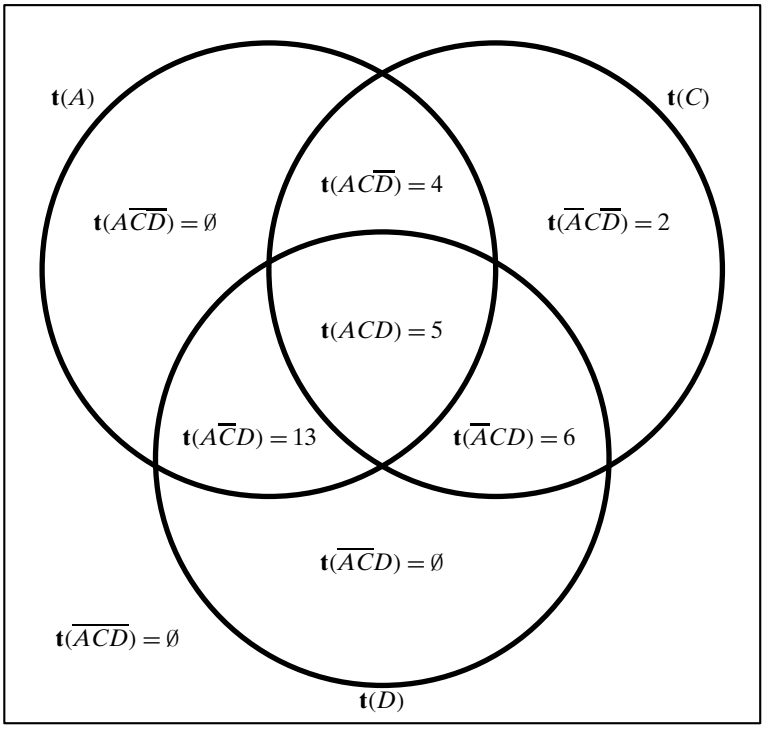


Figure 9.5. Tidset partitioning induced by $\mathbf{t}(A)$, $\mathbf{t}(C)$, and $\mathbf{t}(D)$.

contain all items in Y but no item in Z:

$$sup(Y\overline{Z}) = \left|\{t \in \mathcal{T} \mid Y \subseteq \mathbf{i}(t) \text{ and } Z \cap \mathbf{i}(t) = \emptyset\}\right|$$

Example 9.5. Consider the example dataset in Figure 9.1a. Let $X = ACD$. We have $\mathbf{t}(A) = 1345$, $\mathbf{t}(C) = 2456$, and $\mathbf{t}(D) = 1356$. These three tidsets induce a partitioning on the space of all tids, as illustrated in the Venn diagram shown in Figure 9.5. For example, the region labeled $\mathbf{t}(AC\overline{D}) = 4$ represents those tids that contain A and C but not D. Thus, the support of the generalized itemset $AC\overline{D}$ is 1. The tids that belong to all the eight regions are shown. Some regions are empty, which means that the support of the corresponding generalized itemset is 0.

Inclusion–Exclusion Principle

Let $Y\overline{Z}$ be a generalized itemset, and let $X = Y \cup Z = YZ$. The inclusion–exclusion principle allows one to directly compute the support of $Y\overline{Z}$ as a combination of the supports for all itemsets W, such that $Y \subseteq W \subseteq X$:

$$sup(Y\overline{Z}) = \sum_{Y \subseteq W \subseteq X} -1^{|W \setminus Y|} \cdot sup(W) \tag{9.2}$$

Example 9.6. Let us compute the support of the generalized itemset $\overline{A}C\overline{D} = C\overline{AD}$, where $Y = C$, $Z = AD$ and $X = YZ = ACD$. In the Venn diagram shown in Figure 9.5, we start with all the tids in $\mathbf{t}(C)$, and remove the tids contained in $\mathbf{t}(AC)$ and $\mathbf{t}(CD)$. However, we realize that in terms of support this removes $sup(ACD)$ twice, so we need to add it back. In other words, the support of $C\overline{AD}$ is given as

$$\begin{aligned} sup(C\overline{AD}) &= sup(C) - sup(AC) - sup(CD) + sup(ACD) \\ &= 4 - 2 - 2 + 1 = 1 \end{aligned}$$

But, this is precisely what the inclusion–exclusion formula gives:

$$\begin{aligned} sup(C\overline{AD}) = {} & (-1)^0\, sup(C) + && W = C, |W \setminus Y| = 0 \\ & (-1)^1\, sup(AC) + && W = AC, |W \setminus Y| = 1 \\ & (-1)^1\, sup(CD) + && W = CD, |W \setminus Y| = 1 \\ & (-1)^2\, sup(ACD) && W = ACD, |W \setminus Y| = 2 \\ = {} & sup(C) - sup(AC) - sup(CD) + sup(ACD) \end{aligned}$$

We can see that the support of $C\overline{AD}$ is a combination of the support values over all itemsets W such that $C \subseteq W \subseteq ACD$.

Support Bounds for an Itemset

Notice that the inclusion–exclusion formula in Eq. (9.2) for the support of $Y\overline{Z}$ has terms for all subsets between Y and $X = YZ$. Put differently, for a given k-itemset X, there are 2^k generalized itemsets of the form $Y\overline{Z}$, with $Y \subseteq X$ and $Z = X \setminus Y$, and each such generalized itemset has a term for $sup(X)$ in the inclusion–exclusion equation; this happens when $W = X$. Because the support of any (generalized) itemset must be non-negative, we can derive a bound on the support of X from each of the 2^k generalized itemsets by setting $sup(Y\overline{Z}) \geq 0$. However, note that whenever $|X \setminus Y|$ is even, the coefficient of $sup(X)$ is $+1$, but when $|X \setminus Y|$ is odd, the coefficient of $sup(X)$ is -1 in Eq. (9.2). Thus, from the 2^k possible subsets $Y \subseteq X$, we derive 2^{k-1} lower bounds and 2^{k-1} upper bounds for $sup(X)$, obtained after setting $sup(Y\overline{Z}) \geq 0$, and rearranging the terms in the inclusion–exclusion formula, so that $sup(X)$ is on the left hand side and the the remaining terms are on the right hand side

$$\textbf{Upper Bounds } (|X \setminus Y| \text{ is odd}): \qquad sup(X) \leq \sum_{Y \subseteq W \subset X} -1^{(|X \setminus Y|+1)} sup(W) \tag{9.3}$$

$$\textbf{Lower Bounds } (|X \setminus Y| \text{ is even}): \qquad sup(X) \geq \sum_{Y \subseteq W \subset X} -1^{(|X \setminus Y|+1)} sup(W) \tag{9.4}$$

Note that the only difference in the two equations is the inequality, which depends on the starting subset Y.

Example 9.7. Consider Figure 9.5, which shows the partitioning induced by the tidsets of A, C, and D. We wish to determine the support bounds for $X = ACD$ using each of the generalized itemsets $Y\overline{Z}$ where $Y \subseteq X$. For example, if $Y = C$, then the inclusion-exclusion principle [Eq. (9.2)] gives us

$$sup(C\overline{AD}) = sup(C) - sup(AC) - sup(CD) + sup(ACD)$$

Setting $sup(C\overline{AD}) \geq 0$, and rearranging the terms, we obtain

$$sup(ACD) \geq -sup(C) + sup(AC) + sup(CD)$$

which is precisely the expression from the lower-bound formula in Eq. (9.4) because $|X \setminus Y| = |ACD - C| = |AD| = 2$ is even.

As another example, let $Y = \emptyset$. Setting $sup(\overline{ACD}) \geq 0$, we have

$$\begin{aligned} sup(\overline{ACD}) &= sup(\emptyset) - sup(A) - sup(C) - sup(D) + \\ &\quad sup(AC) + sup(AD) + sup(CD) - sup(ACD) \geq 0 \\ \implies sup(ACD) &\leq sup(\emptyset) - sup(A) - sup(C) - sup(D) + \\ &\quad sup(AC) + sup(AD) + sup(CD) \end{aligned}$$

Notice that this rule gives an upper bound on the support of ACD, which also follows from Eq. (9.3) because $|X \setminus Y| = 3$ is odd.

In fact, from each of the regions in Figure 9.5, we get one bound, and out of the eight possible regions, exactly four give upper bounds and the other four give lower bounds for the support of ACD:

$$\begin{array}{lll} sup(ACD) & \geq 0 & \text{when } Y = ACD \\ & \leq sup(AC) & \text{when } Y = AC \\ & \leq sup(AD) & \text{when } Y = AD \\ & \leq sup(CD) & \text{when } Y = CD \\ & \geq sup(AC) + sup(AD) - sup(A) & \text{when } Y = A \\ & \geq sup(AC) + sup(CD) - sup(C) & \text{when } Y = C \\ & \geq sup(AD) + sup(CD) - sup(D) & \text{when } Y = D \\ & \leq sup(AC) + sup(AD) + sup(CD) - & \\ & \quad sup(A) - sup(C) - sup(D) + sup(\emptyset) & \text{when } Y = \emptyset \end{array}$$

This derivation of the bounds is schematically summarized in Figure 9.6. For instance, at level 2 the inequality is $\geq$, which implies that if Y is any itemset at this level, we will obtain a lower bound. The signs at different levels indicate the coefficient of the corresponding itemset in the upper or lower bound computations via Eq. (9.3) and Eq. (9.4). Finally, the subset lattice shows which intermediate terms W have to be considered in the summation. For instance, if $Y = A$, then the intermediate terms are $W \in \{AC, AD, A\}$, with the corresponding signs $\{+1, +1, -1\}$, so that we obtain the lower bound rule:

$$sup(ACD) \geq sup(AC) + sup(AD) - sup(A)$$

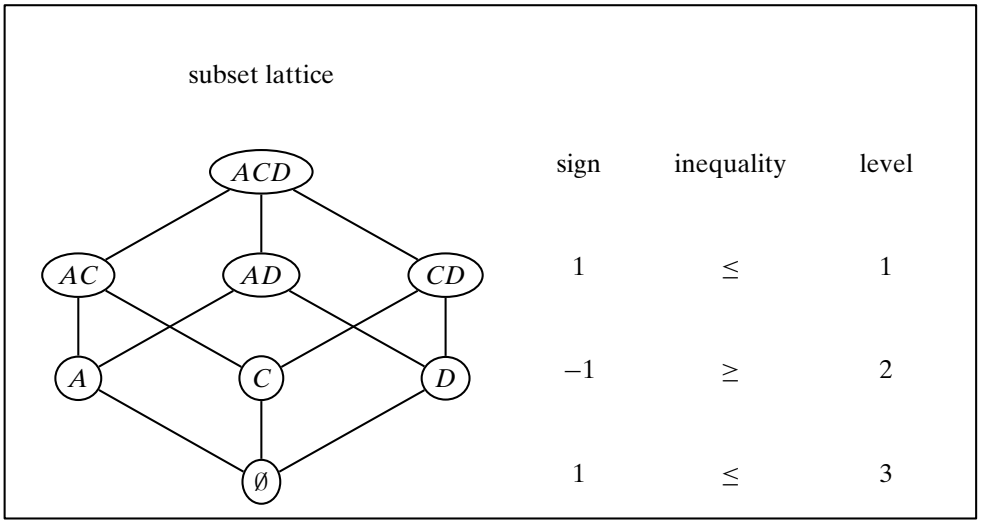


Figure 9.6. Support bounds from subsets.

Nonderivable Itemsets

Given an itemset X, and $Y \subseteq X$, let $IE(Y)$ denote the summation

$$IE(Y) = \sum_{Y \subseteq W \subset X} -1^{(|X \setminus Y|+1)} \cdot sup(W)$$

Then, the sets of all upper and lower bounds for $sup(X)$ are given as

$$UB(X) = \left\{ IE(Y) \middle| \; Y \subseteq X, \; |X \setminus Y| \text{ is odd} \right\}$$
$$LB(X) = \left\{ IE(Y) \middle| \; Y \subseteq X, \; |X \setminus Y| \text{ is even} \right\}$$

An itemset X is called *nonderivable* if $\max\{LB(X)\} \neq \min\{UB(X)\}$, which implies that the support of X cannot be derived from the support values of its subsets; we know only the range of possible values, that is,

$$sup(X) \in \Big[\max\{LB(X)\}, \min\{UB(X)\} \Big]$$

On the other hand, X is derivable if $sup(X) = \max\{LB(X)\} = \min\{UB(X)\}$ because in this case $sup(X)$ can be derived exactly using the supports of its subsets. Thus, the set of all frequent nonderivable itemsets is given as

$$\mathcal{N} = \left\{ X \in \mathcal{F} \mid \max\{LB(X)\} \neq \min\{UB(X)\} \right\}$$

where $\mathcal{F}$ is the set of all frequent itemsets.

Example 9.8. Consider the set of upper bound and lower bound formulas for $sup(ACD)$ outlined in Example 9.7. Using the tidset information in Figure 9.5, the

support lower bounds are

$$\begin{aligned} sup(ACD) &\geq 0 \\ &\geq sup(AC) + sup(AD) - sup(A) = 2 + 3 - 4 = 1 \\ &\geq sup(AC) + sup(CD) - sup(C) = 2 + 2 - 4 = 0 \\ &\geq sup(AD) + sup(CD) - sup(D) = 3 + 2 - 4 = 0 \end{aligned}$$

and the upper bounds are

$$\begin{aligned} sup(ACD) &\leq sup(AC) = 2 \\ &\leq sup(AD) = 3 \\ &\leq sup(CD) = 2 \\ &\leq sup(AC) + sup(AD) + sup(CD) - sup(A) - sup(C) - \\ &\quad sup(D) + sup(\emptyset) = 2 + 3 + 2 - 4 - 4 - 4 + 6 = 1 \end{aligned}$$

Thus, we have

$$\begin{aligned} LB(ACD) &= \{0, 1\} & \max\{LB(ACD)\} &= 1 \\ UB(ACD) &= \{1, 2, 3\} & \min\{UB(ACD)\} &= 1 \end{aligned}$$

Because $\max\{LB(ACD)\} = \min\{UB(ACD)\}$ we conclude that ACD is derivable.

Note that is it not essential to derive all the upper and lower bounds before one can conclude whether an itemset is derivable. For example, let $X = ABDE$. Considering its immediate subsets, we can obtain the following upper bound values:

$$\begin{aligned} sup(ABDE) &\leq sup(ABD) = 3 \\ &\leq sup(ABE) = 4 \\ &\leq sup(ADE) = 3 \\ &\leq sup(BDE) = 3 \end{aligned}$$

From these upper bounds, we know for sure that $sup(ABDE) \leq 3$. Now, let us consider the lower bound derived from $Y = AB$:

$$sup(ABDE) \geq sup(ABD) + sup(ABE) - sup(AB) = 3 + 4 - 4 = 3$$

At this point we know that $sup(ABDE) \geq 3$, so without processing any further bounds, we can conclude that $sup(ABDE) \in [3, 3]$, which means that $ABDE$ is derivable.

For the example database in Figure 9.1a, the set of all frequent nonderivable itemsets, along with their support bounds, is

$$\begin{aligned} \mathcal{N} = \big\{ & A[0,6], B[0,6], C[0,6], D[0,6], E[0,6], \\ & AD[2,4], AE[3,4], CE[3,4], DE[3,4] \big\} \end{aligned}$$

Notice that single items are always nonderivable by definition.

9.5 FURTHER READING

The concept of closed itemsets is based on the elegant lattice theoretic framework of formal concept analysis (Ganter, Wille, and Franzke, 1997). The Charm algorithm for mining frequent closed itemsets appears in Zaki and Hsiao (2005), and the GenMax method for mining maximal frequent itemsets is described in Gouda and Zaki (2005). For an Apriori style algorithm for maximal patterns, called MaxMiner, that uses very effective support lower bound based itemset pruning see Bayardo (1998). The notion of minimal generators was proposed in Bastide et al. (2000); they refer to them as *key patterns*. Nonderivable itemset mining task was introduced in Calders and Goethals (2007).

Bastide, Y., Taouil, R., Pasquier, N., Stumme, G., and Lakhal, L. (2000). "Mining frequent patterns with counting inference." *ACM SIGKDD Explorations*, 2 (2): 66–75.

Bayardo R. J., Jr. (1998). "Efficiently mining long patterns from databases." *In Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM, pp. 85–93.

Calders, T. and Goethals, B. (2007). "Non-derivable itemset mining." *Data Mining and Knowledge Discovery*, 14 (1): 171–206.

Ganter, B., Wille, R., and Franzke, C. (1997). *Formal Concept Analysis: Mathematical Foundations*. New York: Springer-Verlag.

Gouda, K. and Zaki, M. J. (2005). "Genmax: An efficient algorithm for mining maximal frequent itemsets." *Data Mining and Knowledge Discovery*, 11 (3): 223–242.

Zaki, M. J. and Hsiao, C.-J. (2005). "Efficient algorithms for mining closed itemsets and their lattice structure." *IEEE Transactions on Knowledge and Data Engineering*, 17 (4): 462–478.

9.6 EXERCISES

Q1. True or False:

- **(a)** Maximal frequent itemsets are sufficient to determine all frequent itemsets with their supports.
- **(b)** An itemset and its closure share the same set of transactions.
- **(c)** The set of all maximal frequent sets is a subset of the set of all closed frequent itemsets.
- **(d)** The set of all maximal frequent sets is the set of longest possible frequent itemsets.

Q2. Given the database in Table 9.1

- **(a)** Show the application of the closure operator on AE, that is, compute $\mathbf{c}(AE)$. Is AE closed?
- **(b)** Find all frequent, closed, and maximal itemsets using $minsup = 2/6$.

Q3. Given the database in Table 9.2, find all minimal generators using $minsup = 1$.

Table 9.1. Dataset for Q2

Tid	Itemset
t_1	ACD
t_2	BCE
t_3	$ABCE$
t_4	BDE
t_5	$ABCE$
t_6	$ABCD$

Table 9.2. Dataset for Q3

Tid	Itemset
1	ACD
2	BCD
3	AC
4	ABD
5	$ABCD$
6	BCD

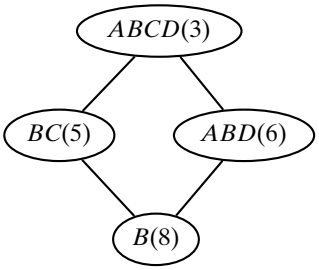


Figure 9.7. Closed itemset lattice for Q4.

Q4. Consider the frequent closed itemset lattice shown in Figure 9.7. Assume that the item space is $\mathcal{I} = \{A, B, C, D, E\}$. Answer the following questions:

(a) What is the frequency of CD?

(b) Find all frequent itemsets and their frequency, for itemsets in the subset interval $[B, ABD]$.

(c) Is ADE frequent? If yes, show its support. If not, why?

Q5. Let $\mathcal{C}$ be the set of all closed frequent itemsets and $\mathcal{M}$ the set of all maximal frequent itemsets for some database. Prove that $\mathcal{M} \subseteq \mathcal{C}$.

Q6. Prove that the closure operator $\mathbf{c} = \mathbf{i} \circ \mathbf{t}$ satisfies the following properties (X and Y are some itemsets):

(a) Extensive: $X \subseteq \mathbf{c}(X)$

(b) Monotonic: If $X \subseteq Y$ then $\mathbf{c}(X) \subseteq \mathbf{c}(Y)$

(c) Idempotent: $\mathbf{c}(X) = \mathbf{c}(\mathbf{c}(X))$

Table 9.3. Dataset for Q7

Tid	Itemset
1	ACD
2	BCD
3	ACD
4	ABD
5	$ABCD$
6	BC

Q7. Let δ be an integer. An itemset X is called a δ-*free* itemset iff for all subsets $Y \subset X$, we have $sup(Y) - sup(X) > \delta$. For any itemset X, we define the δ-*closure* of X as follows:

$$\delta\text{-closure}(X) = \{Y \mid X \subset Y, sup(X) - sup(Y) \leq \delta, \text{ and } Y \text{ is maximal}\}$$

Consider the database shown in Table 9.3. Answer the following questions:

(a) Given $\delta = 1$, compute all the δ-free itemsets.

(b) For each of the δ-free itemsets, compute its δ-closure for $\delta = 1$.

Q8. Given the lattice of frequent itemsets (along with their supports) shown in Figure 9.8, answer the following questions:

(a) List all the closed itemsets.

(b) Is BCD derivable? What about $ABCD$? What are the bounds on their supports.

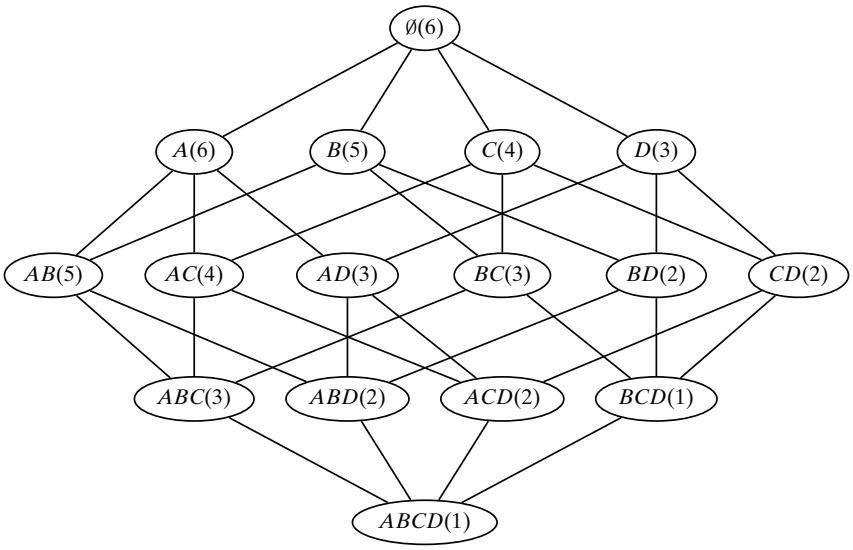


Figure 9.8. Frequent itemset lattice for Q8.

Q9. Prove that if an itemset X is derivable, then so is any superset $Y \supset X$. Using this observation describe an algorithm to mine all nonderivable itemsets.

CHAPTER 10 Sequence Mining

Many real-world applications such as bioinformatics, Web mining, and text mining have to deal with sequential and temporal data. Sequence mining helps discover patterns across time or positions in a given dataset. In this chapter we consider methods to mine frequent sequences, which allow gaps between elements, as well as methods to mine frequent substrings, which do not allow gaps between consecutive elements.

10.1 FREQUENT SEQUENCES

Let Σ denote an *alphabet*, defined as a finite set of characters or symbols, and let $|\Sigma|$ denote its cardinality. A *sequence* or a *string* is defined as an ordered list of symbols, and is written as $\mathbf{s} = s_1 s_2 \ldots s_k$, where $s_i \in \Sigma$ is a symbol at position i, also denoted as $\mathbf{s}[i]$. Here $|\mathbf{s}| = k$ denotes the *length* of the sequence. A sequence with length k is also called a *k-sequence*. We use the notation $\mathbf{s}[i:j] = s_i s_{i+1} \cdots s_{j-1} s_j$ to denote the *substring* or sequence of consecutive symbols in positions i through j, where $j > i$. Define the *prefix* of a sequence $\mathbf{s}$ as any substring of the form $\mathbf{s}[1:i] = s_1 s_2 \ldots s_i$, with $0 \leq i \leq n$. Also, define the *suffix* of $\mathbf{s}$ as any substring of the form $\mathbf{s}[i:n] = s_i s_{i+1} \ldots s_n$, with $1 \leq i \leq n+1$. Note that $\mathbf{s}[1:0]$ is the empty prefix, and $\mathbf{s}[n+1:n]$ is the empty suffix. Let $\Sigma^\star$ be the set of all possible sequences that can be constructed using the symbols in Σ, including the empty sequence $\emptyset$ (which has length zero).

Let $\mathbf{s} = s_1 s_2 \ldots s_n$ and $\mathbf{r} = r_1 r_2 \ldots r_m$ be two sequences over Σ. We say that $\mathbf{r}$ is a *subsequence* of $\mathbf{s}$ denoted $\mathbf{r} \subseteq \mathbf{s}$, if there exists a one-to-one mapping $\phi : [1, m] \rightarrow [1, n]$, such that $\mathbf{r}[i] = \mathbf{s}[\phi(i)]$ and for any two positions i, j in $\mathbf{r}$, $i < j \implies \phi(i) < \phi(j)$. In other words, each position in $\mathbf{r}$ is mapped to a different position in $\mathbf{s}$, and the order of symbols is preserved, even though there may be intervening gaps between consecutive elements of $\mathbf{r}$ in the mapping. If $\mathbf{r} \subseteq \mathbf{s}$, we also say that $\mathbf{s}$ *contains* $\mathbf{r}$. The sequence $\mathbf{r}$ is called a *consecutive subsequence* or substring of $\mathbf{s}$ provided $r_1 r_2 \ldots r_m = s_j s_{j+1} \ldots s_{j+m-1}$, i.e., $\mathbf{r}[1:m] = \mathbf{s}[j : j+m-1]$, with $1 \leq j \leq n-m+1$. For substrings we do not allow any gaps between the elements of $\mathbf{r}$ in the mapping.

Example 10.1. Let $\Sigma = \{A, C, G, T\}$, and let $\mathbf{s} = ACTGAACG$. Then $\mathbf{r}_1 = CGAAG$ is a subsequence of $\mathbf{s}$, and $\mathbf{r}_2 = CTGA$ is a substring of $\mathbf{s}$. The sequence $\mathbf{r}_3 = ACT$ is a prefix of $\mathbf{s}$, and so is $\mathbf{r}_4 = ACTGA$, whereas $\mathbf{r}_5 = GAACG$ is one of the suffixes of $\mathbf{s}$.

Given a database $\mathbf{D} = \{\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_N\}$ of N sequences, and given some sequence $\mathbf{r}$, the *support* of $\mathbf{r}$ in the database $\mathbf{D}$ is defined as the total number of sequences in $\mathbf{D}$ that contain $\mathbf{r}$

$$sup(\mathbf{r}) = \left|\left\{\mathbf{s}_i \in \mathbf{D} | \mathbf{r} \subseteq \mathbf{s}_i\right\}\right|$$

The *relative support* of $\mathbf{r}$ is the fraction of sequences that contain $\mathbf{r}$

$$rsup(\mathbf{r}) = sup(\mathbf{r})/N$$

Given a user-specified *minsup* threshold, we say that a sequence $\mathbf{r}$ is *frequent* in database $\mathbf{D}$ if $sup(\mathbf{r}) \geq minsup$. A frequent sequence is *maximal* if it is not a subsequence of any other frequent sequence, and a frequent sequence is *closed* if it is not a subsequence of any other frequent sequence with the same support.

10.2 MINING FREQUENT SEQUENCES

For sequence mining the order of the symbols matters, and thus we have to consider all possible *permutations* of the symbols as the possible frequent candidates. Contrast this with itemset mining, where we had only to consider *combinations* of the items. The sequence search space can be organized in a prefix search tree. The root of the tree, at level 0, contains the empty sequence, with each symbol $x \in \Sigma$ as one of its children. As such, a node labeled with the sequence $\mathbf{s} = s_1 s_2 \ldots s_k$ at level k has children of the form $\mathbf{s}' = s_1 s_2 \ldots s_k s_{k+1}$ at level $k+1$. In other words, $\mathbf{s}$ is a prefix of each child $\mathbf{s}'$, which is also called an *extension* of $\mathbf{s}$.

Example 10.2. Let $\Sigma = \{A, C, G, T\}$ and let the sequence database $\mathbf{D}$ consist of the three sequences shown in Table 10.1. The sequence search space organized as a prefix search tree is illustrated in Figure 10.1. The support of each sequence is shown within brackets. For example, the node labeled A has three extensions AA, AG, and AT, out of which AT is infrequent if $minsup = 3$.

The subsequence search space is conceptually infinite because it comprises all sequences in Σ^*, that is, all sequences of length zero or more that can be created using symbols in Σ. In practice, the database $\mathbf{D}$ consists of bounded length sequences. Let l denote the length of the longest sequence in the database, then, in the worst case, we will have to consider all candidate sequences of length up to l, which gives the following

Table 10.1. Example sequence database

Id	Sequence
$\mathbf{s}_1$	$CAGAAGT$
$\mathbf{s}_2$	$TGACAG$
$\mathbf{s}_3$	$GAAGT$

ALGORITHM 10.1. Algorithm GSP

```
GSP (D, Σ, minsup):
1  F ← ∅
2  C(1) ← {∅} // Initial prefix tree with single symbols
3  foreach s ∈ Σ do Add s as child of ∅ in C(1) with sup(s) ← 0
4  k ← 1 // k denotes the level
5  while C(k) ≠ ∅ do
6      ComputeSupport (C(k), D)
7      foreach leaf s ∈ C(k) do
8          if sup(r) ≥ minsup then F ← F ∪ {(r, sup(r))}
9          else remove s from C(k)
10     C(k+1) ← ExtendPrefixTree (C(k))
11     k ← k + 1
12 return F(k)

ComputeSupport (C(k), D):
13 foreach s_i ∈ D do
14     foreach r ∈ C(k) do
15         if r ⊆ s_i then sup(r) ← sup(r) + 1

ExtendPrefixTree (C(k)):
16 foreach leaf r_a ∈ C(k) do
17     foreach leaf r_b ∈ Children(Parent(r_a)) do
18         r_ab ← r_a + r_b[k] // extend r_a with last item of r_b
           // prune if there are any infrequent subsequences
19         if r_c ∈ C(k), for all r_c ⊂ r_ab, such that |r_c| = |r_ab| − 1 then
20             Add r_ab as child of r_a with sup(r_ab) ← 0
21     if no extensions from r_a then
22         remove r_a, and all ancestors of r_a with no extensions, from C(k)
23 return C(k)
```

bound on the size of the search space:

$$|\Sigma|^1 + |\Sigma|^2 + \cdots + |\Sigma|^l = O(|\Sigma|^l) \tag{10.1}$$

since at level k there are $|\Sigma|^k$ possible subsequences of length k.

10.2.1 Level-wise Mining: GSP

We can devise an effective sequence mining algorithm that searches the sequence prefix tree using a level-wise or breadth-first search. Given the set of frequent sequences at level k, we generate all possible sequence extensions or *candidates* at level $k+1$. We next compute the support of each candidate and prune those that are not frequent. The search stops when no more frequent extensions are possible.

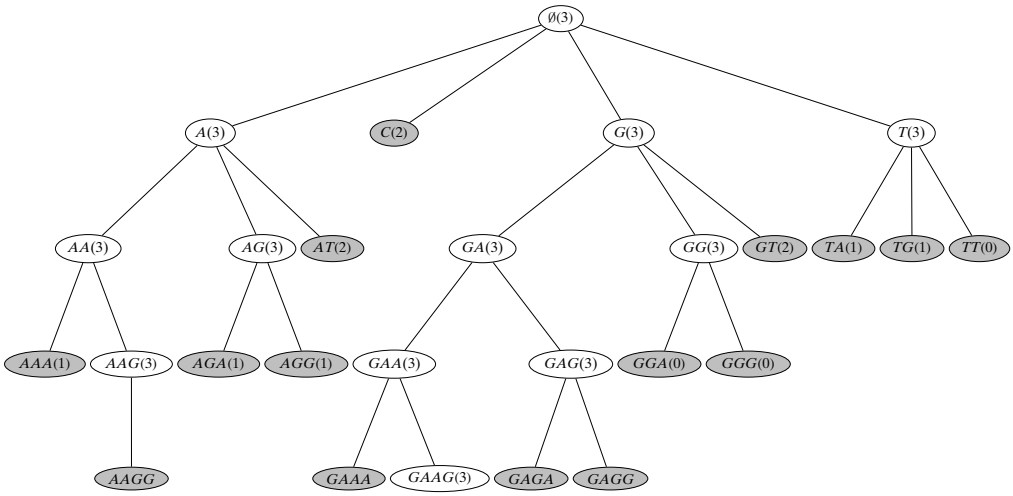


Figure 10.1. Sequence search space: shaded ovals represent candidates that are infrequent; those without support in brackets can be pruned based on an infrequent subsequence. Unshaded ovals represent frequent sequences.

The pseudo-code for the level-wise, generalized sequential pattern (GSP) mining method is shown in Algorithm 10.1. It uses the antimonotonic property of support to prune candidate patterns, that is, no supersequence of an infrequent sequence can be frequent, and all subsequences of a frequent sequence must be frequent. The prefix search tree at level k is denoted $\mathcal{C}^{(k)}$. Initially $\mathcal{C}^{(1)}$ comprises all the symbols in Σ. Given the current set of candidate k-sequences $\mathcal{C}^{(k)}$, the method first computes their support (line 6). For each database sequence $\mathbf{s}_i \in \mathbf{D}$, we check whether a candidate sequence $\mathbf{r} \in \mathcal{C}^{(k)}$ is a subsequence of $\mathbf{s}_i$. If so, we increment the support of $\mathbf{r}$. Once the frequent sequences at level k have been found, we generate the candidates for level $k+1$ (line 10). For the extension, each leaf $\mathbf{r}_a$ is extended with the last symbol of any other leaf $\mathbf{r}_b$ that shares the same prefix (i.e., has the same parent), to obtain the new candidate $(k+1)$-sequence $\mathbf{r}_{ab} = \mathbf{r}_a + \mathbf{r}_b[k]$ (line 18). If the new candidate $\mathbf{r}_{ab}$ contains any infrequent k-sequence, we prune it.

Example 10.3. For example, let us mine the database shown in Table 10.1 using $minsup = 3$. That is, we want to find only those subsequences that occur in all three database sequences. Figure 10.1 shows that we begin by extending the empty sequence $\emptyset$ at level 0, to obtain the candidates A, C, G, and T at level 1. Out of these C can be pruned because it is not frequent. Next we generate all possible candidates at level 2. Notice that using A as the prefix we generate all possible extensions AA, AG, and AT. A similar process is repeated for the other two symbols G and T. Some candidate extensions can be pruned without counting. For example, the extension $GAAA$ obtained from GAA can be pruned because it has an infrequent subsequence AAA. The figure shows all the frequent sequences (unshaded), out of which $GAAG(3)$ and $T(3)$ are the maximal ones.

The computational complexity of GSP is $O(|\Sigma|^l)$ as per Eq. (10.1), where l is the length of the longest frequent sequence. The I/O complexity is $O(l \cdot \mathbf{D})$ because we compute the support of an entire level in one scan of the database.

10.2.2 Vertical Sequence Mining: Spade

The Spade algorithm uses a vertical database representation for sequence mining. The idea is to record for each symbol the sequence identifiers and the positions where it occurs. For each symbol $s \in \Sigma$, we keep a set of tuples of the form $\langle i, pos(s)\rangle$, where $pos(s)$ is the set of positions in the database sequence $\mathbf{s}_i \in \mathbf{D}$ where symbol s appears. Let $\mathcal{L}(s)$ denote the set of such sequence-position tuples for symbol s, which we refer to as the *poslist*. The set of poslists for each symbol $s \in \Sigma$ thus constitutes a vertical representation of the input database. In general, given k-sequence $\mathbf{r}$, its poslist $\mathcal{L}(\mathbf{r})$ maintains the list of positions for the occurrences of the last symbol $\mathbf{r}[k]$ in each database sequence $\mathbf{s}_i$, provided $\mathbf{r} \subseteq \mathbf{s}_i$. The support of sequence $\mathbf{r}$ is simply the number of distinct sequences in which $\mathbf{r}$ occurs, that is, $sup(\mathbf{r}) = |\mathcal{L}(\mathbf{r})|$.

Example 10.4. In Table 10.1, the symbol A occurs in $\mathbf{s}_1$ at positions 2, 4, and 5. Thus, we add the tuple $\langle 1, \{2, 4, 5\}\rangle$ to $\mathcal{L}(A)$. Because A also occurs at positions 3 and 5 in sequence $\mathbf{s}_2$, and at positions 2 and 3 in $\mathbf{s}_3$, the complete poslist for A is $\{\langle 1, \{2, 4, 5\}\rangle, \langle 2, \{3, 5\}\rangle, \langle 1, \{2, 3\}\rangle\}$. We have $sup(A) = 3$, as its poslist contains three tuples. Figure 10.2 shows the poslist for each symbol, as well as other sequences. For example, for sequence GT, we find that it is a subsequence of $\mathbf{s}_1$ and $\mathbf{s}_3$.

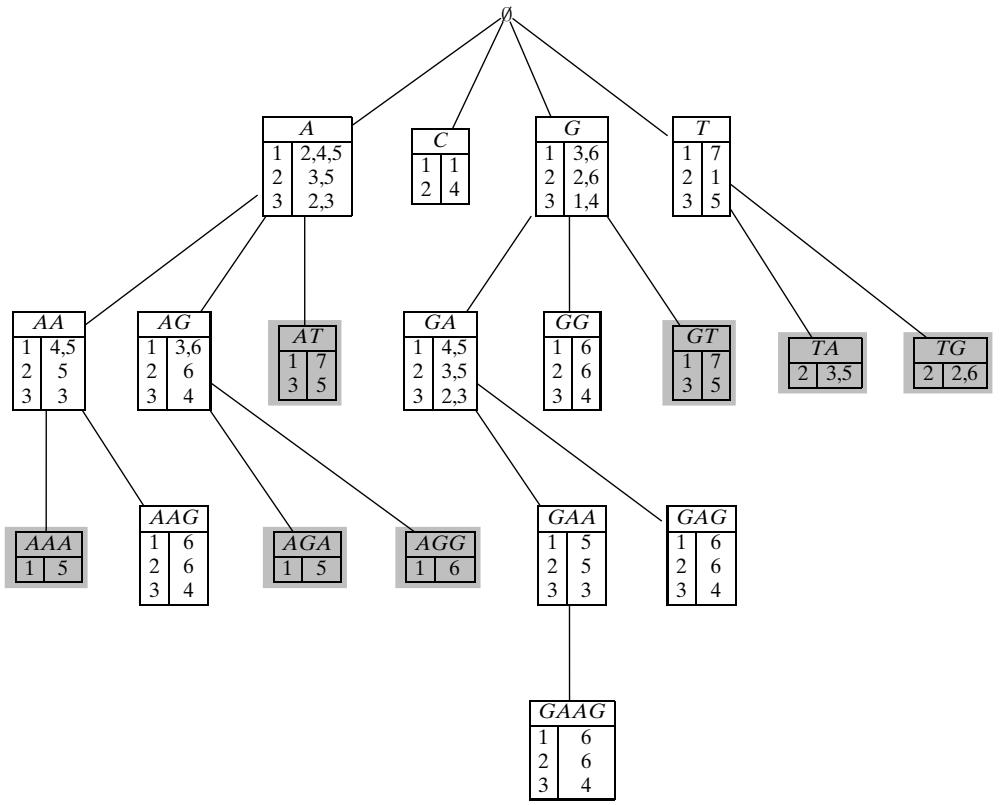


Figure 10.2. Sequence mining via Spade: infrequent sequences with at least one occurrence are shown shaded; those with zero support are not shown.

Even though there are two occurrences of GT in $\mathbf{s}_1$, the last symbol T occurs at position 7 in both occurrences, thus the poslist for GT has the tuple $\langle 1, 7\rangle$. The full poslist for GT is $\mathcal{L}(GT) = \{\langle 1, 7\rangle, \langle 3, 5\rangle\}$. The support of GT is $sup(GT) = |\mathcal{L}(GT)| = 2$.

Support computation in Spade is done via *sequential join* operations. Given the poslists for any two k-sequences $\mathbf{r}_a$ and $\mathbf{r}_b$ that share the same $(k-1)$ length prefix, the idea is to perform sequential joins on the poslists to compute the support for the new $(k+1)$ length candidate sequence $\mathbf{r}_{ab} = \mathbf{r}_a + \mathbf{r}_b[k]$. Given a tuple $\langle i, pos(\mathbf{r}_b[k])\rangle \in \mathcal{L}(\mathbf{r}_b)$, we first check if there exists a tuple $\langle i, pos(\mathbf{r}_a[k])\rangle \in \mathcal{L}(\mathbf{r}_a)$, that is, both sequences must occur in the same database sequence $\mathbf{s}_i$. Next, for each position $p \in pos(\mathbf{r}_b[k])$, we check whether there exists a position $q \in pos(\mathbf{r}_a[k])$ such that $q < p$. If yes, this means that the symbol $\mathbf{r}_b[k]$ occurs after the last position of $\mathbf{r}_a$ and thus we retain p as a valid occurrence of $\mathbf{r}_{ab}$. The poslist $\mathcal{L}(\mathbf{r}_{ab})$ comprises all such valid occurrences. Notice how we keep track of positions only for the last symbol in the candidate sequence. This is because we extend sequences from a common prefix, so there is no need to keep track of all the occurrences of the symbols in the prefix. We denote the sequential join as $\mathcal{L}(\mathbf{r}_{ab}) = \mathcal{L}(\mathbf{r}_a) \cap \mathcal{L}(\mathbf{r}_b)$.

The main advantage of the vertical approach is that it enables different search strategies over the sequence search space, including breadth or depth-first search. Algorithm 10.2 shows the pseudo-code for Spade. Given a set of sequences P that share the same prefix, along with their poslists, the method creates a new prefix equivalence class P_a for each sequence $\mathbf{r}_a \in P$ by performing sequential joins with every sequence $\mathbf{r}_b \in P$, including self-joins. After removing the infrequent extensions, the new equivalence class P_a is then processed recursively.

ALGORITHM 10.2. Algorithm SPADE

```
// Initial Call: F ← ∅, k ← 0,
    P ← {⟨s, L(s)⟩ | s ∈ Σ, sup(s) ≥ minsup}
SPADE (P, minsup, F, k):
1 foreach r_a ∈ P do
2     F ← F ∪ {(r_a, sup(r_a))}
3     P_a ← ∅
4     foreach r_b ∈ P do
5         r_ab = r_a + r_b[k]
6         L(r_ab) = L(r_a) ∩ L(r_b)
7         if sup(r_ab) ≥ minsup then
8             P_a ← P_a ∪ {⟨r_ab, L(r_ab)⟩}
9     if P_a ≠ ∅ then SPADE (P, minsup, F, k+1)
```

Example 10.5. Consider the poslists for A and G shown in Figure 10.2. To obtain $\mathcal{L}(AG)$, we perform a sequential join over the poslists $\mathcal{L}(A)$ and $\mathcal{L}(G)$. For the tuples $\langle 1, \{2, 4, 5\}\rangle \in \mathcal{L}(A)$ and $\langle 1, \{3, 6\}\rangle \in \mathcal{L}(G)$, both positions 3 and 6 for G, occur after some occurrence of A, for example, at position 2. Thus, we add the tuple $\langle 1, \{3, 6\}\rangle$ to $\mathcal{L}(AG)$. The complete poslist for AG is $\mathcal{L}(AG) = \{\langle 1, \{3, 6\}\rangle, \langle 2, 6\rangle, \langle 3, 4\rangle\}$.

Figure 10.2 illustrates the complete working of the Spade algorithm, along with all the candidates and their poslists.

10.2.3 Projection-Based Sequence Mining: PrefixSpan

Let $\mathbf{D}$ denote a database, and let $s \in \Sigma$ be any symbol. The *projected database* with respect to s, denoted $\mathbf{D}_s$, is obtained by finding the the first occurrence of s in $\mathbf{s}_i$, say at position p. Next, we retain in $\mathbf{D}_s$ only the suffix of $\mathbf{s}_i$ starting at position $p+1$. Further, any infrequent symbols are removed from the suffix. This is done for each sequence $\mathbf{s}_i \in \mathbf{D}$.

Example 10.6. Consider the three database sequences in Table 10.1. Given that the symbol G first occurs at position 3 in $\mathbf{s}_1 = CAGAAGT$, the projection of $\mathbf{s}_1$ with respect to G is the suffix $AAGT$. The projected database for G, denoted $\mathbf{D}_G$ is therefore given as: $\{\mathbf{s}_1 : AAGT,\ \mathbf{s}_2 : AAG,\ \mathbf{s}_3 : AAGT\}$.

The main idea in PrefixSpan is to compute the support for only the individual symbols in the projected database $\mathbf{D}_s$, and then to perform recursive projections on the frequent symbols in a depth-first manner. The PrefixSpan method is outlined in Algorithm 10.3. Here $\mathbf{r}$ is a frequent subsequence, and $\mathbf{D}_\mathbf{r}$ is the projected dataset for $\mathbf{r}$. Initially $\mathbf{r}$ is empty and $\mathbf{D}_\mathbf{r}$ is the entire input dataset $\mathbf{D}$. Given a database of (projected) sequences $\mathbf{D}_\mathbf{r}$, PrefixSpan first finds all the frequent symbols in the projected dataset. For each such symbol s, we extend $\mathbf{r}$ by appending s to obtain the new frequent subsequence $\mathbf{r}_s$. Next, we create the projected dataset $\mathbf{D}_s$ by projecting $\mathbf{D}_\mathbf{r}$ on symbol s. A recursive call to PrefixSpan is then made with $\mathbf{r}_s$ and $\mathbf{D}_s$.

ALGORITHM 10.3. Algorithm PREFIXSPAN

```
// Initial Call: D_r ← D, r ← ∅, F ← ∅
PREFIXSPAN (D_r, r, minsup, F):
1 foreach s ∈ Σ such that sup(s, D_r) ≥ minsup do
2     r_s = r + s // extend r by symbol s
3     F ← F ∪ {(r_s, sup(s, D_r))}
4     D_s ← ∅ // create projected data for symbol s
5     foreach s_i ∈ D_r do
6         s'_i ← projection of s_i w.r.t symbol s
7         Remove any infrequent symbols from s'_i
8         Add s'_i to D_s if s'_i ≠ ∅
9     if D_s ≠ ∅ then PREFIXSPAN (D_s, r_s, minsup, F)
```

Example 10.7. Figure 10.3 shows the projection-based PrefixSpan mining approach for the example dataset in Table 10.1 using $minsup = 3$. Initially we start with the whole database $\mathbf{D}$, which can also be denoted as $\mathbf{D}_\emptyset$. We compute the support of each symbol, and find that C is not frequent (shown crossed out). Among the frequent symbols, we first create a new projected dataset $\mathbf{D}_A$. For $\mathbf{s}_1$, we find that the first A occurs at position 2, so we retain only the suffix $GAAGT$. In $\mathbf{s}_2$, the first A occurs at position 3, so the suffix is CAG. After removing C (because it is infrequent), we are left with only AG as the projection of $\mathbf{s}_2$ on A. In a similar manner we obtain the projection for $\mathbf{s}_3$ as AGT. The left child of the root shows the final projected dataset $\mathbf{D}_A$. Now the mining proceeds recursively. Given $\mathbf{D}_A$, we count the symbol supports in $\mathbf{D}_A$, finding that only A and G are frequent, which will lead to the projection $\mathbf{D}_{AA}$ and then $\mathbf{D}_{AG}$, and so on. The complete projection-based approach is illustrated in Figure 10.3.

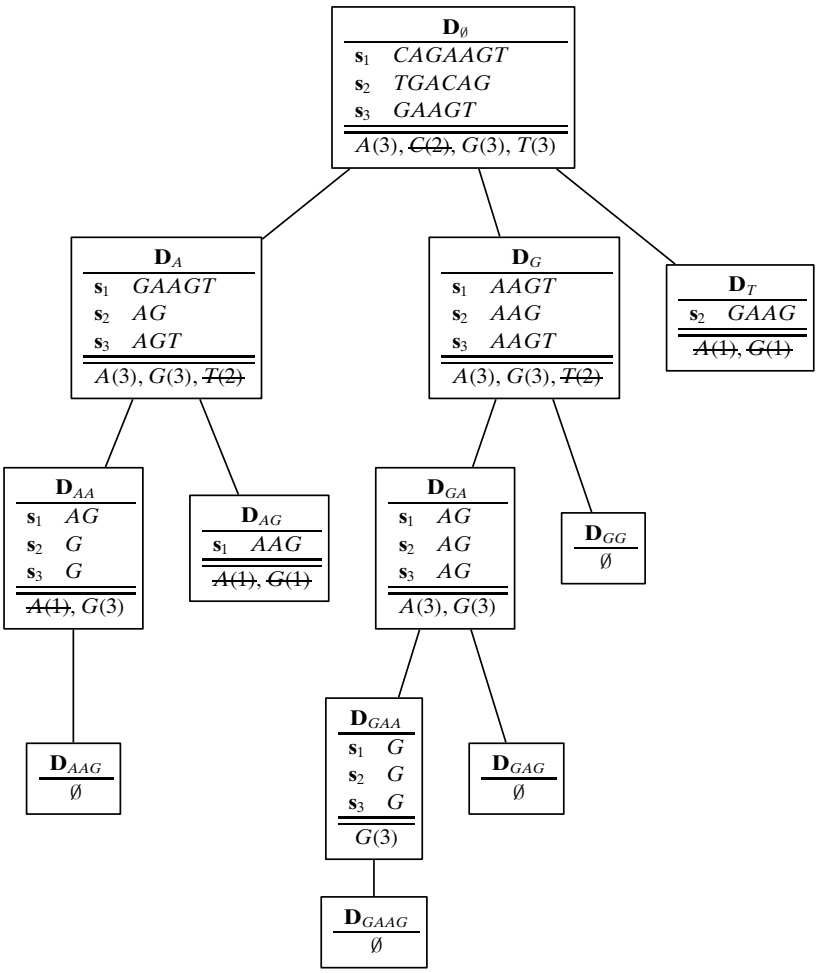


Figure 10.3. Projection-based sequence mining: PrefixSpan.

10.3 SUBSTRING MINING VIA SUFFIX TREES

We now look at efficient methods for mining frequent substrings. Let $\mathbf{s}$ be a sequence having length n, then there are at most $O(n^2)$ possible distinct substrings contained in $\mathbf{s}$. To see this consider substrings of length w, of which there are $n-w+1$ possible ones in $\mathbf{s}$. Adding over all substring lengths we get

$$\sum_{w=1}^{n}(n-w+1) = n + (n-1) + \cdots + 2 + 1 = O(n^2)$$

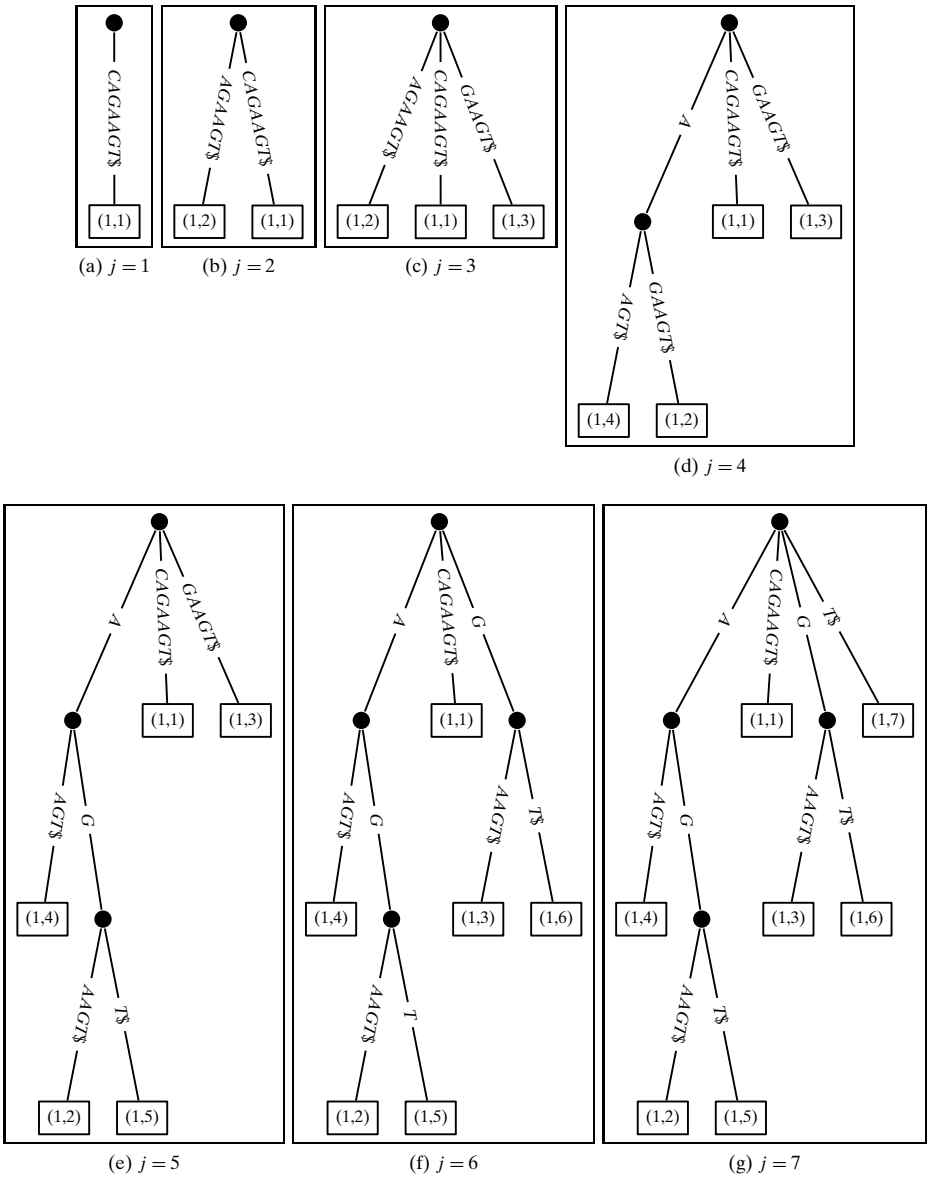
This is a much smaller search space compared to subsequences, and consequently we can design more efficient algorithms for solving the frequent substring mining task. In fact, we can mine all the frequent substrings in worst case $O(Nn^2)$ time for a dataset $\mathbf{D} = \{\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_N\}$ with N sequences.

10.3.1 Suffix Tree

Let Σ denote the alphabet, and let $\$ \notin \Sigma$ be a *terminal* character used to mark the end of a string. Given a sequence $\mathbf{s}$, we append the terminal character so that $\mathbf{s} = s_1 s_2 \ldots s_n s_{n+1}$, where $s_{n+1} = \$$, and the jth suffix of $\mathbf{s}$ is given as $\mathbf{s}[j : n+1] = s_j s_{j+1} \ldots s_{n+1}$. The *suffix tree* of the sequences in the database $\mathbf{D}$, denoted $\mathcal{T}$, stores all the suffixes for each $\mathbf{s}_i \in \mathbf{D}$ in a tree structure, where suffixes that share a common prefix lie on the same path from the root of the tree. The substring obtained by concatenating all the symbols from the root node to a node v is called the *node label* of v, and is denoted as $L(v)$. The substring that appears on an edge (v_a, v_b) is called an *edge label*, and is denoted as $L(v_a, v_b)$. A suffix tree has two kinds of nodes: internal and leaf nodes. An internal node in the suffix tree (except for the root) has at least two children, where each edge label to a child begins with a different symbol. Because the terminal character is unique, there are as many leaves in the suffix tree as there are unique suffixes over all the sequences. Each leaf node corresponds to a suffix shared by one or more sequences in $\mathbf{D}$.

It is straightforward to obtain a quadratic time and space suffix tree construction algorithm. Initially, the suffix tree $\mathcal{T}$ is empty. Next, for each sequence $\mathbf{s}_i \in \mathbf{D}$, with $|\mathbf{s}_i| = n_i$, we generate all its suffixes $\mathbf{s}_i[j : n_i + 1]$, with $1 \le j \le n_i$, and insert each of them into the tree by following the path from the root until we either reach a leaf or there is a mismatch in one of the symbols along an edge. If we reach a leaf, we insert the pair (i, j) into the leaf, noting that this is the jth suffix of sequence $\mathbf{s}_i$. If there is a mismatch in one of the symbols, say at position $p \ge j$, we add an internal vertex just before the mismatch, and create a new leaf node containing (i, j) with edge label $\mathbf{s}_i[p : n_i + 1]$.

Example 10.8. Consider the database in Table 10.1 with three sequences. In particular, let us focus on $\mathbf{s}_1 = CAGAAGT$. Figure 10.4 shows what the suffix tree $\mathcal{T}$ looks like after inserting the jth suffix of $\mathbf{s}_1$ into $\mathcal{T}$. The first suffix is the entire sequence $\mathbf{s}_1$ appended with the terminal symbol; thus the suffix tree contains a single leaf containing $(1, 1)$ under the root (Figure 10.4a). The second suffix is $AGAAGT\$$, and Figure 10.4b shows the resulting suffix tree, which now has two leaves. The third



(a) $j = 1$ (b) $j = 2$ (c) $j = 3$ (d) $j = 4$

(e) $j = 5$ (f) $j = 6$ (g) $j = 7$

Figure 10.4. Suffix tree construction: (a)–(g) show the successive changes to the tree, after we add the jth suffix of $\mathbf{s}_1 = CAGAAGT\$$ for $j = 1, \ldots, 7$.

suffix $GAAGT\$$ begins with G, which has not yet been observed, so it creates a new leaf in $\mathcal{T}$ under the root. The fourth suffix $AAGT\$$ shares the prefix A with the second suffix, so it follows the path beginning with A from the root. However, because there is a mismatch at position 2, we create an internal node right before it and insert the leaf $(1, 4)$, as shown in Figure 10.4d. The suffix tree obtained after inserting all of the suffixes of $\mathbf{s}_1$ is shown in Figure 10.4g, and the complete suffix tree for all three sequences is shown in Figure 10.5.

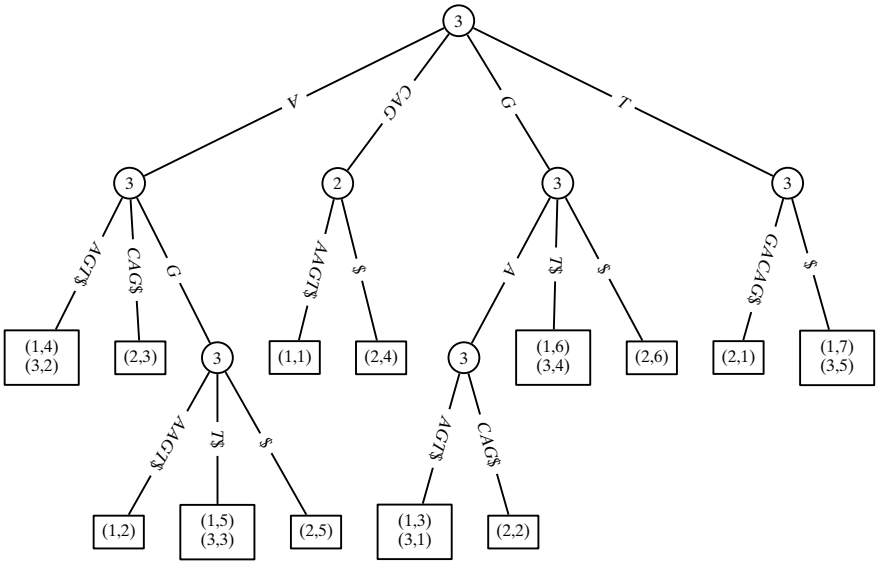


Figure 10.5. Suffix tree for all three sequences in Table 10.1. Internal nodes store support information. Leaves also record the support (not shown).

In terms of the time and space complexity, the algorithm sketched above requires $O(Nn^2)$ time and space, where N is the number of sequences in $\mathbf{D}$, and n is the longest sequence length. The time complexity follows from the fact that the method always inserts a new suffix starting from the root of the suffix tree. This means that in the worst case it compares $O(n)$ symbols per suffix insertion, giving the worst case bound of $O(n^2)$ over all n suffixes. The space complexity comes from the fact that each suffix is explicitly represented in the tree, taking $n + (n-1) + \cdots + 1 = O(n^2)$ space. Over all the N sequences in the database, we obtain $O(Nn^2)$ as the worst case time and space bounds.

Frequent Substrings

Once the suffix tree is built, we can compute all the frequent substrings by checking how many different sequences appear in a leaf node or under an internal node. The node labels for the nodes with support at least *minsup* yield the set of frequent substrings; all the prefixes of such node labels are also frequent. The suffix tree can also support ad hoc queries for finding all the occurrences in the database for any query substring $\mathbf{q}$. For each symbol in $\mathbf{q}$, we follow the path from the root until all symbols in $\mathbf{q}$ have been seen, or until there is a mismatch at any position. If $\mathbf{q}$ is found, then the set of leaves under that path is the list of occurrences of the query $\mathbf{q}$. On the other hand, if there is mismatch that means the query does not occur in the database. In terms of the query time complexity, because we have to match each character in $\mathbf{q}$, we immediately get $O(|\mathbf{q}|)$ as the time bound (assuming that $|\Sigma|$ is a constant), which is *independent* of the size of the database. Listing all the matches takes additional time, for a total time complexity of $O(|\mathbf{q}| + k)$, if there are k matches.

Example 10.9. Consider the suffix tree shown in Figure 10.5, which stores all the suffixes for the sequence database in Table 10.1. To facilitate frequent substring enumeration, we store the support for each internal as well as leaf node, that is, we store the number of distinct sequence ids that occur at or under each node. For example, the leftmost child of the root node on the path labeled A has support 3 because there are three distinct sequences under that subtree. If $minsup = 3$, then the frequent substrings are A, AG, G, GA, and T. Out of these, the maximal ones are AG, GA, and T. If $minsup = 2$, then the maximal frequent substrings are $GAAGT$ and CAG.

For ad hoc querying consider $\mathbf{q} = GAA$. Searching for symbols in $\mathbf{q}$ starting from the root leads to the leaf node containing the occurrences $(1, 3)$ and $(3, 1)$, which means that GAA appears at position 3 in $\mathbf{s}_1$ and at position 1 in $\mathbf{s}_3$. On the other hand if $\mathbf{q} = CAA$, then the search terminates with a mismatch at position 3 after following the branch labeled CAG from the root. This means that $\mathbf{q}$ does not occur in the database.

10.3.2 Ukkonen's Linear Time Algorithm

We now present a linear time and space algorithm for constructing suffix trees. We first consider how to build the suffix tree for a single sequence $\mathbf{s} = s_1 s_2 \ldots s_n s_{n+1}$, with $s_{n+1} =$ \$. The suffix tree for the entire dataset of N sequences can be obtained by inserting each sequence one by one.

Achieving Linear Space

Let us see how to reduce the space requirements of a suffix tree. If an algorithm stores all the symbols on each edge label, then the space complexity is $O(n^2)$, and we cannot achieve linear time construction either. The trick is to not explicitly store all the edge labels, but rather to use an *edge-compression* technique, where we store only the starting and ending positions of the edge label in the input string $\mathbf{s}$. That is, if an edge label is given as $\mathbf{s}[i : j]$, then we represent is as the interval $[i, j]$.

Example 10.10. Consider the suffix tree for $\mathbf{s}_1 = CAGAAGT\$$ shown in Figure 10.4g. The edge label $CAGAAGT\$$ for the suffix $(1, 1)$ can be represented via the interval $[1, 8]$ because the edge label denotes the substring $\mathbf{s}_1[1 : 8]$. Likewise, the edge label $AAGT\$$ leading to suffix $(1, 2)$ can be compressed as $[4, 8]$ because $AAGT\$ = \mathbf{s}_1[4 : 8]$. The complete suffix tree for $\mathbf{s}_1$ with compressed edge labels is shown in Figure 10.6.

In terms of space complexity, note that when we add a new suffix to the tree $\mathcal{T}$, it can create at most one new internal node. As there are n suffixes, there are n leaves in $\mathcal{T}$ and at most n internal nodes. With at most $2n$ nodes, the tree has at most $2n - 1$ edges, and thus the total space required to store an interval for each edge is $2(2n - 1) = 4n - 2 = O(n)$.

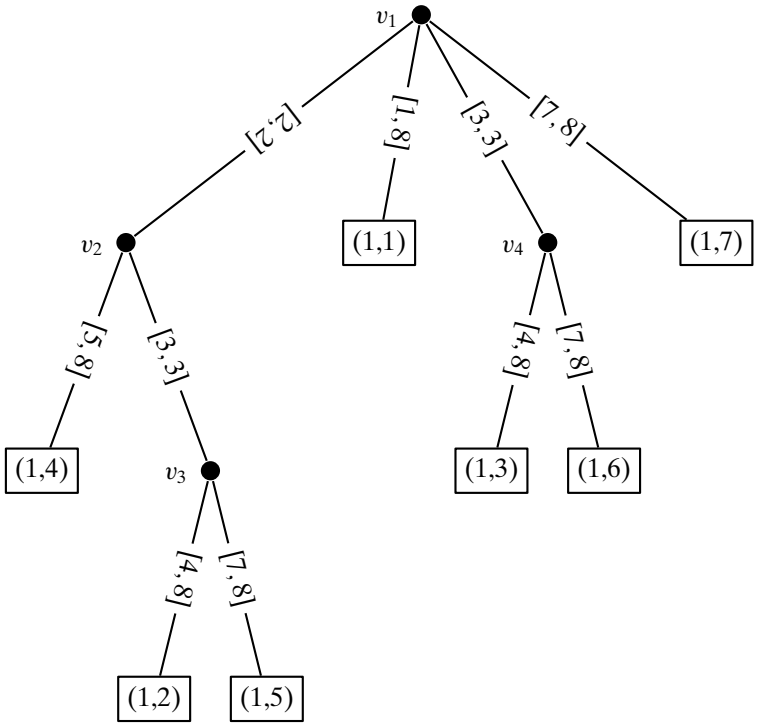


Figure 10.6. Suffix tree for $\mathbf{s}_1 = CAGAAGT\$$ using edge-compression.

Achieving Linear Time

Ukkonen's method is an *online* algorithm, that is, given a string $\mathbf{s} = s_1 s_2 \ldots s_n\$$ it constructs the full suffix tree in phases. Phase i builds the tree up to the i-th symbol in $\mathbf{s}$, that is, it updates the suffix tree from the previous phase by adding the next symbol s_i. Let $\mathcal{T}_i$ denote the suffix tree up to the ith prefix $\mathbf{s}[1:i]$, with $1 \leq i \leq n$. Ukkonen's algorithm constructs $\mathcal{T}_i$ from $\mathcal{T}_{i-1}$, by making sure that all suffixes including the *current* character s_i are in the new intermediate tree $\mathcal{T}_i$. In other words, in the ith phase, it inserts all the suffixes $\mathbf{s}[j:i]$ from $j = 1$ to $j = i$ into the tree $\mathcal{T}_i$. Each such insertion is called the jth *extension* of the ith *phase*. Once we process the terminal character at position $n+1$ we obtain the final suffix tree $\mathcal{T}$ for $\mathbf{s}$.

Algorithm 10.4 shows the code for a naive implementation of Ukkonen's approach. This method has cubic time complexity because to obtain $\mathcal{T}_i$ from $\mathcal{T}_{i-1}$ takes $O(i^2)$ time, with the last phase requiring $O(n^2)$ time. With n phases, the total time is $O(n^3)$. Our goal is to show that this time can be reduced to just $O(n)$ via the optimizations described in the following paragraghs.

Implicit Suffixes This optimization states that, in phase i, if the jth extension $\mathbf{s}[j:i]$ is found in the tree, then any subsequent extensions will also be found, and consequently there is no need to process further extensions in phase i. Thus, the suffix tree $\mathcal{T}_i$ at the end of phase i has *implicit suffixes* corresponding to extensions $j+1$ through i. It is important to note that all suffixes will become explicit the first time we encounter a new substring that does not already exist in the tree. This will surely happen in phase

ALGORITHM 10.4. Algorithm NAIVEUKKONEN

```
NAIVEUKKONEN (s):
1 n ← |s|
2 s[n+1] ← $ // append terminal character
3 T ← ∅ // add empty string as root
4 foreach i = 1,...,n+1 do // phase i - construct T_i
5     foreach j = 1,...,i do // extension j for phase i
          // Insert s[j:i] into the suffix tree
6         Find end of the path with label s[j:i-1] in T
7         Insert s_i at end of path;
8 return T
```

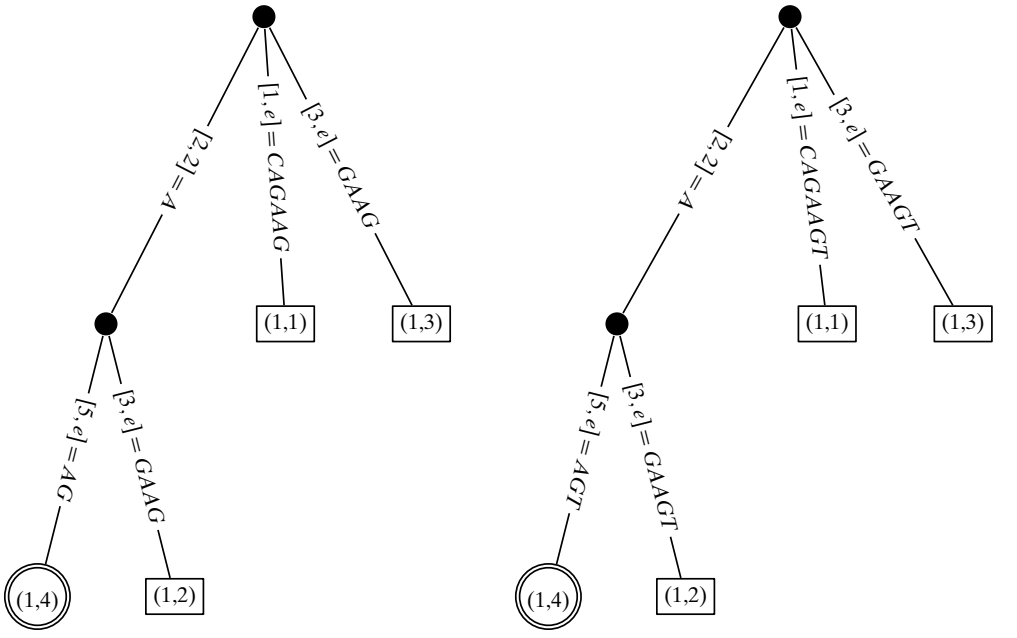
$n+1$ when we process the terminal character \$, as it cannot occur anywhere else in $\mathbf{s}$ (after all, $\$ \notin \Sigma$).

Implicit Extensions Let the current phase be i, and let $l \leq i-1$ be the last explicit suffix in the previous tree $\mathcal{T}_{i-1}$. All explicit suffixes in $\mathcal{T}_{i-1}$ have edge labels of the form $[x, i-1]$ leading to the corresponding leaf nodes, where the starting position x is node specific, but the ending position must be $i-1$ because s_{i-1} was added to the end of these paths in phase $i-1$. In the current phase i, we would have to extend these paths by adding s_i at the end. However, instead of explicitly incrementing all the ending positions, we can replace the ending position by a pointer e which keeps track of the current phase being processed. If we replace $[x, i-1]$ with $[x, e]$, then in phase i, if we set $e = i$, then immediately all the l existing suffixes get *implicitly* extended to $[x, i]$. Thus, in one operation of incrementing e we have, in effect, taken care of extensions 1 through l for phase i.

Example 10.11. Let $\mathbf{s}_1 = CAGAAGT\$$. Assume that we have already performed the first six phases, which result in the tree $\mathcal{T}_6$ shown in Figure 10.7a. The last explicit suffix in $\mathcal{T}_6$ is $l = 4$. In phase $i = 7$ we have to execute the following extensions:

$CAGAAGT$	extension 1
$AGAAGT$	extension 2
$GAAGT$	extension 3
$AAGT$	extension 4
AGT	extension 5
GT	extension 6
T	extension 7

At the start of the seventh phase, we set $e = 7$, which yields implicit extensions for all suffixes explicitly in the tree, as shown in Figure 10.7b. Notice how symbol $s_7 = T$ is now implicitly on each of the leaf edges, for example, the label $[5, e] = AG$ in $\mathcal{T}_6$ now becomes $[5, e] = AGT$ in $\mathcal{T}_7$. Thus, the first four extensions listed above are taken care of by simply incrementing e. To complete phase 7 we have to process the remaining extensions.



(a) $\mathcal{T}_6$ (b) $\mathcal{T}_7$, extensions $j = 1, \ldots, 4$

Figure 10.7. Implicit extensions in phase $i = 7$. Last explicit suffix in $\mathcal{T}_6$ is $l = 4$ (shown double-circled). Edge labels shown for convenience; only the intervals are stored.

Skip/Count Trick For the jth extension of phase i, we have to search for the substring $\mathbf{s}[j : i-1]$ so that we can add s_i at the end. However, note that this string must exist in $\mathcal{T}_{i-1}$ because we have already processed symbol s_{i-1} in the previous phase. Thus, instead of searching for each character in $\mathbf{s}[j : i-1]$ starting from the root, we first *count* the number of symbols on the edge beginning with character s_j; let this length be m. If m is longer than the length of the substring (i.e., if $m > i - j$), then the substring must end on this edge, so we simply jump to position $i - j$ and insert s_i. On the other hand, if $m \leq i - j$, then we can *skip* directly to the child node, say v_c, and search for the remaining string $\mathbf{s}[j + m : i - 1]$ from v_c using the same skip/count technique. With this optimization, the cost of an extension becomes proportional to the number of nodes on the path, as opposed to the number of characters in $\mathbf{s}[j : i-1]$.

Suffix Links We saw that with the skip/count optimization we can search for the substring $\mathbf{s}[j : i-1]$ by following nodes from parent to child. However, we still have to start from the root node each time. We can avoid searching from the root via the use of *suffix links*. For each internal node v_a we maintain a link to the internal node v_b, where $L(v_b)$ is the immediate suffix of $L(v_a)$. In extension $j-1$, let v_p denote the internal node under which we find $\mathbf{s}[j-1 : i]$, and let m be the length of the node label of v_p. To insert the jth extension $\mathbf{s}[j : i]$, we follow the suffix link from v_p to another node, say v_s, and search for the remaining substring $\mathbf{s}[j + m - 1 : i - 1]$ from v_s. The use of suffix links allows us to jump internally within the tree for different extensions, as opposed to searching from the root each time. As a final observation, if extension j

ALGORITHM 10.5. Algorithm UKKONEN

```
UKKONEN (s):
1  n ← |s|
2  s[n+1] ← $ // append terminal character
3  T ← ∅ // add empty string as root
4  l ← 0 // last explicit suffix
5  foreach i = 1, ..., n+1 do // phase i - construct T_i
6      e ← i // implicit extensions
7      foreach j = l+1, ..., i do // extension j for phase i
           // Insert s[j : i] into the suffix tree
8          Find end of s[j : i−1] in T via skip/count and suffix links
9          if s_i ∈ T then // implicit suffixes
10             break
11         else
12             Insert s_i at end of path
13             Set last explicit suffix l if needed
14 return T
```

creates a new internal node, then its suffix link will point to the new internal node that will be created during extension $j+1$.

The pseudo-code for the optimized Ukkonen's algorithm is shown in Algorithm 10.5. It is important to note that it achieves linear time and space only with all of the optimizations in conjunction, namely implicit extensions (line 6), implicit suffixes (line 9), and skip/count and suffix links for inserting extensions in $\mathcal{T}$ (line 8).

Example 10.12. Let us look at the execution of Ukkonen's algorithm on the sequence $\mathbf{s}_1 = CAGAAGT\$$, as shown in Figure 10.8. In phase 1, we process character $s_1 = C$ and insert the suffix $(1,1)$ into the tree with edge label $[1,e]$ (see Figure 10.8a). In phases 2 and 3, new suffixes $(1,2)$ and $(1,3)$ are added (see Figures 10.8b–10.8c). For phase 4, when we want to process $s_4 = A$, we note that all suffixes up to $l = 3$ are already explicit. Setting $e = 4$ implicitly extends all of them, so we have only to make sure that the last extension ($j = 4$) consisting of the single character A is in the tree. Searching from the root, we find A in the tree implicitly, and we thus proceed to the next phase. In the next phase, we set $e = 5$, and the suffix $(1,4)$ becomes explicit when we try to add the extension AA, which is not in the tree. For $e = 6$, we find the extension AG already in the tree and we skip ahead to the next phase. At this point the last explicit suffix is still $(1,4)$. For $e = 7$, T is a previously unseen symbol, and so all suffixes will become explicit, as shown in Figure 10.8g.

It is instructive to see the extensions in the last phase ($i = 7$). As described in Example 10.11, the first four extensions will be done implicitly. Figure 10.9a shows the suffix tree after these four extensions. For extension 5, we begin at the last explicit

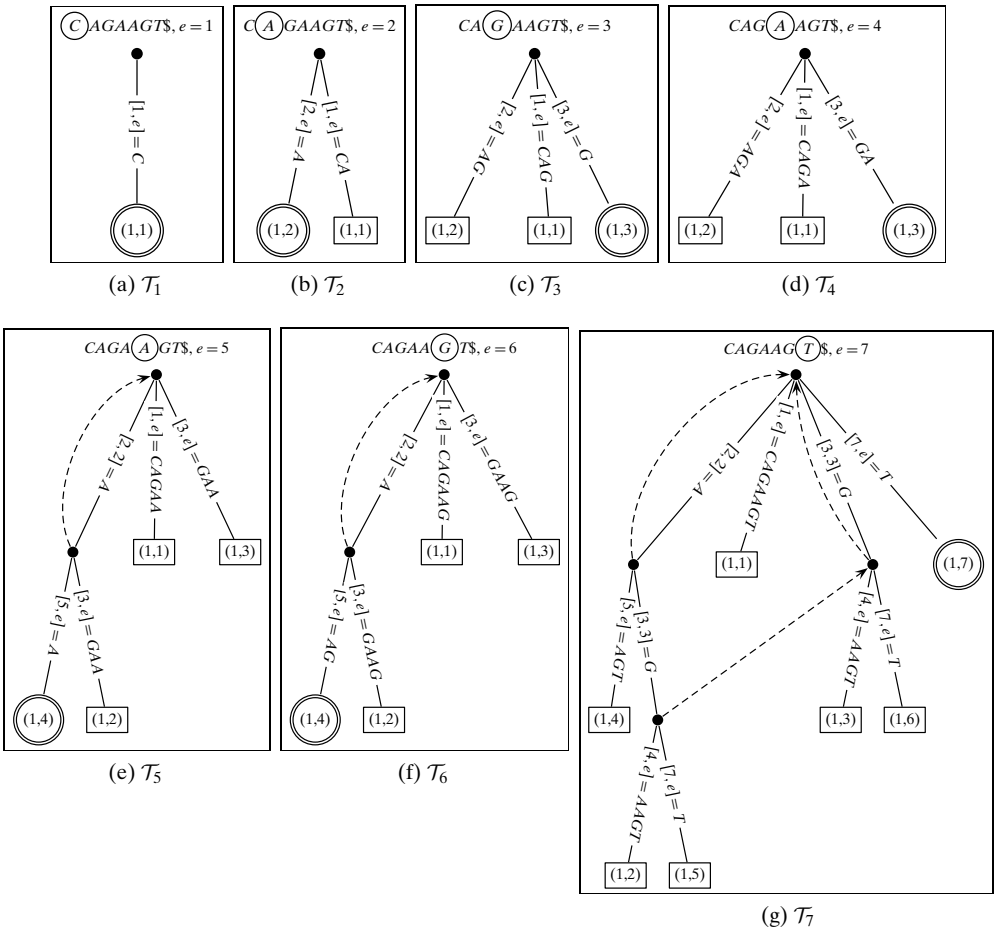


Figure 10.8. Ukkonen's linear time algorithm for suffix tree construction. Steps (a)–(g) show the successive changes to the tree after the ith phase. The suffix links are shown with dashed lines. The double-circled leaf denotes the last explicit suffix in the tree. The last step is not shown because when $e = 8$, the terminal character \$ will not alter the tree. All the edge labels are shown for ease of understanding, although the actual suffix tree keeps only the intervals for each edge.

leaf, follow its parent's suffix link, and begin searching for the remaining characters from that point. In our example, the suffix link points to the root, so we search for $\mathbf{s}[5:7] = AGT$ from the root. We skip to node v_A, and look for the remaining string GT, which has a mismatch inside the edge $[3, e]$. We thus create a new internal node after G, and insert the explicit suffix $(1, 5)$, as shown in Figure 10.9b. The next extension $\mathbf{s}[6:7] = GT$ begins at the newly created leaf node $(1, 5)$. Following the closest suffix link leads back to the root, and a search for GT gets a mismatch on the edge out of the root to leaf $(1, 3)$. We then create a new internal node v_G at that point, add a suffix link from the previous internal node v_{AG} to v_G, and add a new explicit leaf $(1, 6)$, as shown in Figure 10.9c. The last extension, namely $j = 7$, corresponding

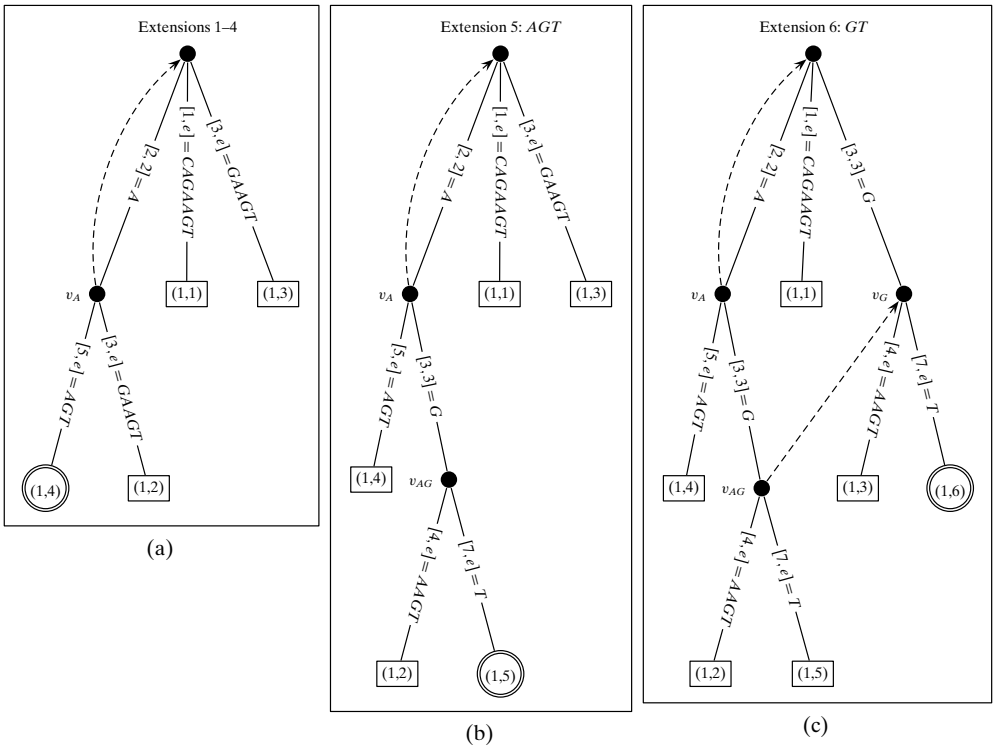


Figure 10.9. Extensions in phase $i = 7$. Initially the last explicit suffix is $l = 4$ and is shown double-circled. All the edge labels are shown for convenience; the actual suffix tree keeps only the intervals for each edge.

to $\mathbf{s}[7:7] = T$, results in making all the suffixes explicit because the symbol T has been seen for the first time. The resulting tree is shown in Figure 10.8g.

Once $\mathbf{s}_1$ has been processed, we can then insert the remaining sequences in the database $\mathbf{D}$ into the existing suffix tree. The final suffix tree for all three sequences is shown in Figure 10.5, with additional suffix links (not shown) from all the internal nodes.

Ukkonen's algorithm has time complexity of $O(n)$ for a sequence of length n because it does only a constant amount of work (amortized) to make each suffix explicit. Note that, for each phase, a certain number of extensions are done implicitly just by incrementing e. Out of the i extensions from $j = 1$ to $j = i$, let us say that l are done implicitly. For the remaining extensions, we stop the first time some suffix is implicitly in the tree; let that extension be k. Thus, phase i needs to add explicit suffixes only for suffixes $l + 1$ through $k - 1$. For creating each explicit suffix, we perform a constant number of operations, which include following the closest suffix link, skip/counting to look for the first mismatch, and inserting if needed a new suffix leaf node. Because each leaf becomes explicit only once, and the number of skip/count steps are bounded by $O(n)$ over the whole tree, we get a worst-case $O(n)$

time algorithm. The total time over the entire database of N sequences is thus $O(Nn)$, if n is the longest sequence length.

10.4 FURTHER READING

The level-wise GSP method for mining sequential patterns was proposed in Srikant and Agrawal (March 1996). Spade is described in Zaki (2001), and the PrefixSpan algorithm in Pei et al. (2004). Ukkonen's linear time suffix tree construction method appears in Ukkonen (1995). For an excellent introduction to suffix trees and their numerous applications see Gusfield (1997); the suffix tree description in this chapter has been heavily influenced by it.

Gusfield, D. (1997). *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. New York: Cambridge University Press.

Pei, J., Han, J., Mortazavi-Asl, B., Wang, J., Pinto, H., Chen, Q., Dayal, U., and Hsu, M.-C. (2004). "Mining sequential patterns by pattern-growth: The PrefixSpan approach." *IEEE Transactions on Knowledge and Data Engineering*, 16 (11): 1424–1440.

Srikant, R. and Agrawal, R. (March 1996). "Mining sequential patterns: Generalizations and performance improvements." *In Proceedings of the 5th International Conference on Extending Database Technology*. New York: Springer-Verlag.

Ukkonen, E. (1995). "On-line construction of suffix trees." *Algorithmica*, 14 (3): 249–260.

Zaki, M. J. (2001). "SPADE: An efficient algorithm for mining frequent sequences." *Machine Learning*, 42 (1–2): 31–60.

10.5 EXERCISES

Q1. Consider the database shown in Table 10.2. Answer the following questions:

(a) Let $minsup = 4$. Find all frequent sequences.

(b) Given that the alphabet is $\Sigma = \{A, C, G, T\}$. How many possible sequences of length k can there be?

Table 10.2. Sequence database for Q1

Id	Sequence
$\mathbf{s}_1$	*AATACAAGAAC*
$\mathbf{s}_2$	*GTATGGTGAT*
$\mathbf{s}_3$	*AACATGGCCAA*
$\mathbf{s}_4$	*AAGCGTGGTCAA*

Q2. Given the DNA sequence database in Table 10.3, answer the following questions using $minsup = 4$

(a) Find the maximal frequent sequences.

(b) Find all the closed frequent sequences.

(c) Find the maximal frequent substrings.
(d) Show how Spade would work on this dataset.
(e) Show the steps of the PrefixSpan algorithm.

Table 10.3. Sequence database for Q2

Id	Sequence
$\mathbf{s}_1$	$ACGTCACG$
$\mathbf{s}_2$	$TCGA$
$\mathbf{s}_3$	$GACTGCA$
$\mathbf{s}_4$	$CAGTC$
$\mathbf{s}_5$	$AGCT$
$\mathbf{s}_6$	$TGCAGCTC$
$\mathbf{s}_7$	$AGTCAG$

Q3. Given $\mathbf{s} = AABBACBBAA$, and $\Sigma = \{A, B, C\}$. Define support as the number of occurrence of a subsequence in $\mathbf{s}$. Using $minsup = 2$, answer the following questions:
(a) Show how the vertical Spade method can be extended to mine all frequent substrings (consecutive subsequences) in $\mathbf{s}$.
(b) Construct the suffix tree for $\mathbf{s}$ using Ukkonen's method. Show all intermediate steps, including all suffix links.
(c) Using the suffix tree from the previous step, find all the occurrences of the query $\mathbf{q} = ABBA$ allowing for at most two mismatches.
(d) Show the suffix tree when we add another character A just before the \$. That is, you must undo the effect of adding the \$, add the new symbol A, and then add \$ back again.
(e) Describe an algorithm to extract all the maximal frequent substrings from a suffix tree. Show all maximal frequent substrings in $\mathbf{s}$.

Q4. Consider a bitvector based approach for mining frequent subsequences. For instance, in Table 10.2, for $\mathbf{s}_1$, the symbol C occurs at positions 5 and 11. Thus, the bitvector for C in $\mathbf{s}_1$ is given as 00001000001. Because C does not appear in $\mathbf{s}_2$ its bitvector can be omitted for $\mathbf{s}_2$. The complete set of bitvectors for symbol C is

$$(\mathbf{s}_1, 00001000001)$$
$$(\mathbf{s}_3, 00100001100)$$
$$(\mathbf{s}_4, 000100000100)$$

Given the set of bitvectors for each symbol show how we can mine all frequent subsequences by using bit operations on the bitvectors. Show the frequent subsequences and their bitvectors using $minsup = 4$.

Q5. Consider the database shown in Table 10.4. Each sequence comprises itemset events that happen at the same time. For example, sequence $\mathbf{s}_1$ can be considered to be a sequence of itemsets $(AB)_{10}(B)_{20}(AB)_{30}(AC)_{40}$, where symbols within brackets are considered to co-occur at the same time, which is given in the subscripts. Describe an algorithm that can mine all the frequent subsequences over itemset events. The

Table 10.4. Sequences for Q5

Id	Time	Items
$\mathbf{s}_1$	10	A, B
	20	B
	30	A, B
	40	A, C
$\mathbf{s}_2$	20	A, C
	30	A, B, C
	50	B
$\mathbf{s}_3$	10	A
	30	B
	40	A
	50	C
	60	B
$\mathbf{s}_4$	30	A, B
	40	A
	50	B
	60	C

itemsets can be of any length as long as they are frequent. Find all frequent itemset sequences with $minsup = 3$.

Q6. The suffix tree shown in Figure 10.5 contains all suffixes for the three sequences $\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3$ in Table 10.1. Note that a pair (i, j) in a leaf denotes the jth suffix of sequence $\mathbf{s}_i$.

(a) Add a new sequence $\mathbf{s}_4 = GAAGCAGAA$ to the existing suffix tree, using the Ukkonen algorithm. Show the last character position (e), along with the suffixes (l) as they become explicit in the tree for $\mathbf{s}_4$. Show the final suffix tree after all suffixes of $\mathbf{s}_4$ have become explicit.

(b) Find all closed frequent substrings with $minsup = 2$ using the final suffix tree.

Q7. Given the following three sequences:

$$\mathbf{s}_1 : GAAGT$$
$$\mathbf{s}_2 : CAGAT$$
$$\mathbf{s}_3 : ACGT$$

Find all the frequent subsequences with $minsup = 2$, but allowing at most a gap of 1 position between successive sequence elements.

CHAPTER 11 Graph Pattern Mining

Graph data is becoming increasingly more ubiquitous in today's networked world. Examples include social networks as well as cell phone networks and blogs. The Internet is another example of graph data, as is the hyperlinked structure of the World Wide Web (WWW). Bioinformatics, especially systems biology, deals with understanding interaction networks between various types of biomolecules, such as protein–protein interactions, metabolic networks, gene networks, and so on. Another prominent source of graph data is the Semantic Web, and linked open data, with graphs represented using the Resource Description Framework (RDF) data model.

The goal of graph mining is to extract interesting subgraphs from a single large graph (e.g., a social network), or from a database of many graphs. In different applications we may be interested in different kinds of subgraph patterns, such as subtrees, complete graphs or cliques, bipartite cliques, dense subgraphs, and so on. These may represent, for example, communities in a social network, hub and authority pages on the WWW, cluster of proteins involved in similar biochemical functions, and so on. In this chapter we outline methods to mine all the frequent subgraphs that appear in a database of graphs.

11.1 ISOMORPHISM AND SUPPORT

A graph is a pair $G = (V, E)$ where V is a set of vertices, and $E \subseteq V \times V$ is a set of edges. We assume that edges are unordered, so that the graph is undirected. If (u, v) is an edge, we say that u and v are *adjacent* and that v is a *neighbor* of u, and vice versa. The set of all neighbors of u in G is given as $N(u) = \{v \in V \mid (u, v) \in E\}$. A *labeled graph* has labels associated with its vertices as well as edges. We use $L(u)$ to denote the label of the vertex u, and $L(u, v)$ to denote the label of the edge (u, v), with the set of vertex labels denoted as Σ_V and the set of edge labels as Σ_E. Given an edge $(u, v) \in G$, the tuple $\langle u, v, L(u), L(v), L(u, v)\rangle$ that augments the edge with the node and edge labels is called an *extended edge*.

Example 11.1. Figure 11.1a shows an example of an unlabeled graph, whereas Figure 11.1b shows the same graph, with labels on the vertices, taken from the vertex

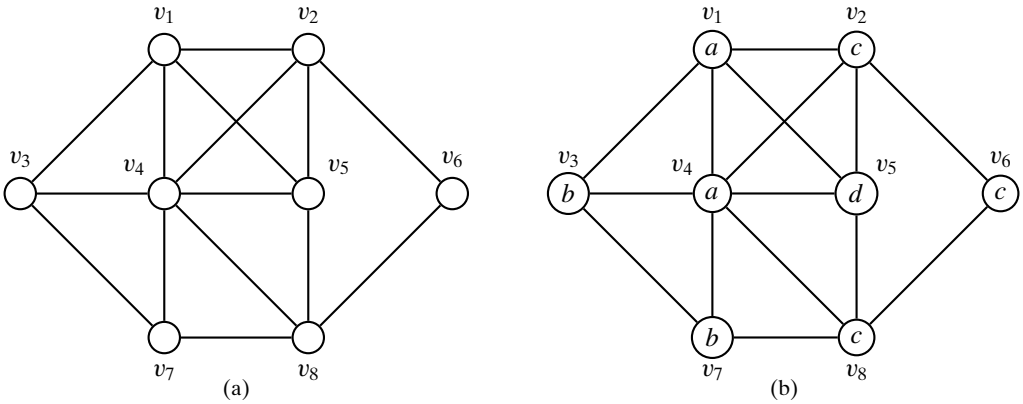


Figure 11.1. An unlabeled (a) and labeled (b) graph with eight vertices.

label set $\Sigma_V = \{a, b, c, d\}$. In this example, edges are all assumed to be unlabeled, and are therefore edge labels are not shown. Considering Figure 11.1b, the label of vertex v_4 is $L(v_4) = a$, and its neighbors are $N(v_4) = \{v_1, v_2, v_3, v_5, v_7, v_8\}$. The edge (v_4, v_1) leads to the extended edge $\langle v_4, v_1, a, a\rangle$, where we omit the edge label $L(v_4, v_1)$ because it is empty.

Subgraphs

A graph $G' = (V', E')$ is said to be a *subgraph* of G if $V' \subseteq V$ and $E' \subseteq E$. Note that this definition allows for disconnected subgraphs. However, typically data mining applications call for *connected subgraphs*, defined as a subgraph G' such that $V' \subseteq V$, $E' \subseteq E$, and for any two nodes $u, v \in V'$, there exists a *path* from u to v in G'.

Example 11.2. The graph defined by the bold edges in Figure 11.2a is a subgraph of the larger graph; it has vertex set $V' = \{v_1, v_2, v_4, v_5, v_6, v_8\}$. However, it is a disconnected subgraph. Figure 11.2b shows an example of a connected subgraph on the same vertex set V'.

Graph and Subgraph Isomorphism

A graph $G' = (V', E')$ is said to be *isomorphic* to another graph $G = (V, E)$ if there exists a bijective function $\phi : V' \to V$, i.e., both injective (into) and surjective (onto), such that

1. $(u, v) \in E' \iff (\phi(u), \phi(v)) \in E$
2. $\forall u \in V',\ L(u) = L(\phi(u))$
3. $\forall (u, v) \in E',\ L(u, v) = L(\phi(u), \phi(v))$

In other words, the *isomorphism* ϕ preserves the edge adjacencies as well as the vertex and edge labels. Put differently, the extended tuple $\langle u, v, L(u), L(v), L(u, v)\rangle \in G'$ if and only if $\langle \phi(u), \phi(v), L(\phi(u)), L(\phi(v)), L(\phi(u), \phi(v))\rangle \in G$.

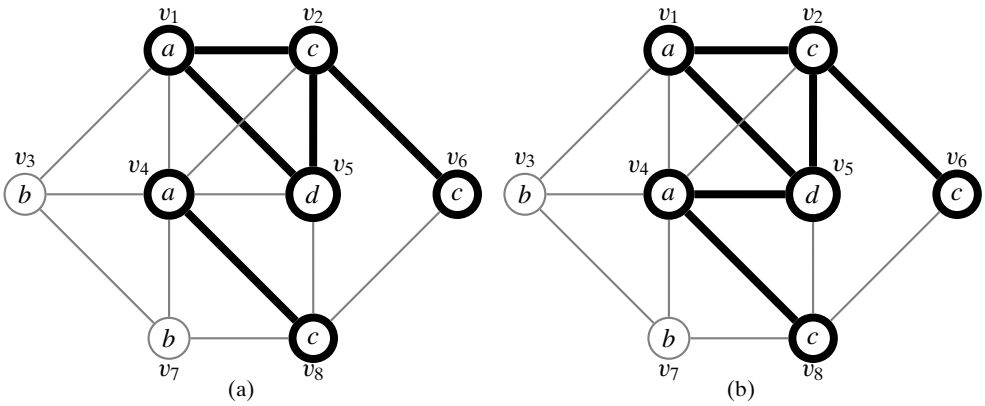


Figure 11.2. A subgraph (a) and connected subgraph (b).

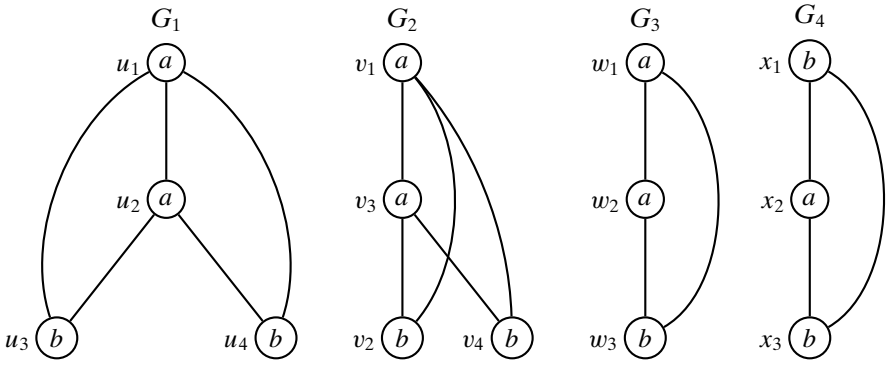


Figure 11.3. Graph and subgraph isomorphism.

If the function ϕ is only injective but not surjective, we say that the mapping ϕ is a *subgraph isomorphism* from G' to G. In this case, we say that G' is isomorphic to a subgraph of G, that is, G' is *subgraph isomorphic* to G, denoted $G' \subseteq G$; we also say that G *contains* G'.

Example 11.3. In Figure 11.3, $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic graphs. There are several possible isomorphisms between G_1 and G_2. An example of an isomorphism $\phi : V_2 \to V_1$ is

$$\phi(v_1) = u_1 \qquad \phi(v_2) = u_3 \qquad \phi(v_3) = u_2 \qquad \phi(v_4) = u_4$$

The inverse mapping ϕ^{-1} specifies the isomorphism from G_1 to G_2. For example, $\phi^{-1}(u_1) = v_1$, $\phi^{-1}(u_2) = v_3$, and so on. The set of all possible isomorphisms from G_2 to G_1 are as follows:

	v_1	v_2	v_3	v_4
ϕ_1	u_1	u_3	u_2	u_4
ϕ_2	u_1	u_4	u_2	u_3
ϕ_3	u_2	u_3	u_1	u_4
ϕ_4	u_2	u_4	u_1	u_3

The graph G_3 is subgraph isomorphic to both G_1 and G_2. The set of all possible subgraph isomorphisms from G_3 to G_1 are as follows:

	w_1	w_2	w_3
ϕ_1	u_1	u_2	u_3
ϕ_2	u_1	u_2	u_4
ϕ_3	u_2	u_1	u_3
ϕ_4	u_2	u_1	u_4

The graph G_4 is not subgraph isomorphic to either G_1 or G_2, and it is also not isomorphic to G_3 because the extended edge $\langle x_1, x_3, b, b\rangle$ has no possible mappings in G_1, G_2 or G_3.

Subgraph Support

Given a database of graphs, $\mathbf{D} = \{G_1, G_2, \ldots, G_n\}$, and given some graph G, the support of G in $\mathbf{D}$ is defined as follows:

$$sup(G) = \Big|\big\{G_i \in \mathbf{D} \mid G \subseteq G_i\big\}\Big|$$

The support is simply the number of graphs in the database that contain G. Given a *minsup* threshold, the goal of graph mining is to mine all frequent connected subgraphs with $sup(G) \geq minsup$.

To mine all the frequent subgraphs, one has to search over the space of all possible graph patterns, which is exponential in size. If we consider subgraphs with m vertices, then there are $\binom{m}{2} = O(m^2)$ possible edges. The number of possible subgraphs with m nodes is then $O(2^{m^2})$ because we may decide either to include or exclude each of the edges. Many of these subgraphs will not be connected, but $O(2^{m^2})$ is a convenient upper bound. When we add labels to the vertices and edges, the number of labeled graphs will be even more. Assume that $|\Sigma_V| = |\Sigma_E| = s$, then there are s^m possible ways to label the vertices and there are s^{m^2} ways to label the edges. Thus, the number of possible labeled subgraphs with m vertices is $2^{m^2} s^m s^{m^2} = O\big((2s)^{m^2}\big)$. This is the worst case bound, as many of these subgraphs will be isomorphic to each other, with the number of distinct subgraphs being much less. Nevertheless, the search space is still enormous because we typically have to search for all subgraphs ranging from a single vertex to some maximum number of vertices given by the largest frequent subgraph.

There are two main challenges in frequent subgraph mining. The first is to systematically generate candidate subgraphs. We use *edge-growth* as the basic mechanism for extending the candidates. The mining process proceeds in a breadth-first (level-wise) or a depth-first manner, starting with an empty subgraph (i.e., with no edge), and adding a new edge each time. Such an edge may either connect two existing vertices in the graph or it may introduce a new vertex as one end of a new edge. The key is to perform nonredundant subgraph enumeration, such that we do not generate the same graph candidate more than once. This means that we have to perform graph isomorphism checking to make sure that duplicate graphs are removed. The second challenge is to count the support of a graph in the database. This involves subgraph isomorphism checking, as we have to find the set of graphs that contain a given candidate.

11.2 CANDIDATE GENERATION

An effective strategy to enumerate subgraph patterns is the so-called *rightmost path extension*. Given a graph G, we perform a depth-first search (DFS) over its vertices, and create a DFS spanning tree, that is, one that covers or spans all the vertices. Edges that are included in the DFS tree are called *forward* edges, and all other edges are called *backward* edges. Backward edges create cycles in the graph. Once we have a DFS tree, define the *rightmost* path as the path from the root to the rightmost leaf, that is, to the leaf with the highest index in the DFS order.

Example 11.4. Consider the graph shown in Figure 11.4a. One of the possible DFS spanning trees is shown in Figure 11.4b (illustrated via bold edges), obtained by starting at v_1 and then choosing the vertex with the smallest index at each step. Figure 11.5 shows the same graph (ignoring the dashed edges), rearranged to emphasize the DFS tree structure. For instance, the edges (v_1, v_2) and (v_2, v_3) are examples of forward edges, whereas (v_3, v_1), (v_4, v_1), and (v_6, v_1) are all backward edges. The bold edges (v_1, v_5), (v_5, v_7) and (v_7, v_8) comprise the rightmost path.

For generating new candidates from a given graph G, we extend it by adding a new edge to vertices only on the rightmost path. We can either extend G by adding backward edges from the *rightmost vertex* to some other vertex on the rightmost path (disallowing self-loops or multi-edges), or we can extend G by adding forward edges from any of the vertices on the rightmost path. A backward extension does not add a new vertex, whereas a forward extension adds a new vertex.

For systematic candidate generation we impose a total order on the extensions, as follows: First, we try all backward extensions from the rightmost vertex, and then we try forward extensions from vertices on the rightmost path. Among the backward edge extensions, if u_r is the rightmost vertex, the extension (u_r, v_i) is tried before (u_r, v_j) if $i < j$. In other words, backward extensions closer to the root are considered before those farther away from the root along the rightmost path. Among the forward edge extensions, if v_x is the new vertex to be added, the extension (v_i, v_x) is tried before

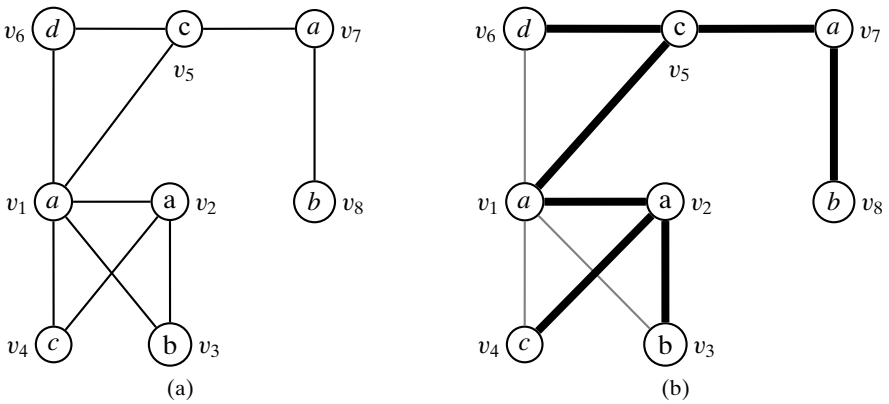


Figure 11.4. A graph (a) and a possible depth-first spanning tree (b).

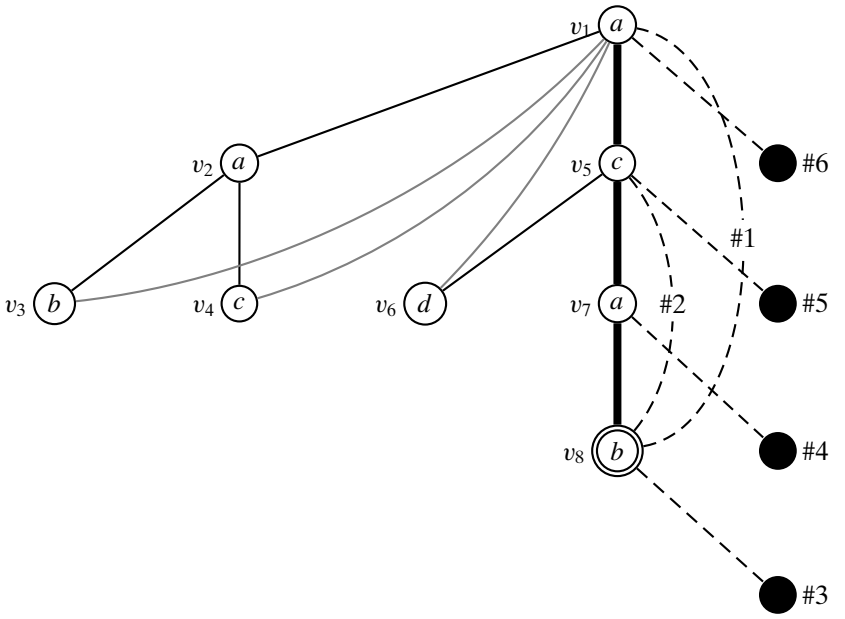


Figure 11.5. Rightmost path extensions. The bold path is the rightmost path in the DFS tree. The *rightmost vertex* is v_8, shown double circled. Solid black lines (thin and bold) indicate the *forward* edges, which are part of the DFS tree. The *backward* edges, which by definition are not part of the DFS tree, are shown in gray. The set of possible extensions on the rightmost path are shown with dashed lines. The precedence ordering of the extensions is also shown.

(v_j, v_x) if $i > j$. In other words, the vertices farther from the root (those at greater depth) are extended before those closer to the root. Also note that the new vertex will be numbered $x = r + 1$, as it will become the new rightmost vertex after the extension.

Example 11.5. Consider the order of extensions shown in Figure 11.5. Node v_8 is the rightmost vertex; thus we try backward extensions only from v_8. The first extension, denoted #1 in Figure 11.5, is the backward edge (v_8, v_1) connecting v_8 to the root, and the next extension is (v_8, v_5), denoted #2, which is also backward. No other backward extensions are possible without introducing multiple edges between the same pair of vertices. The forward extensions are tried in reverse order, starting from the rightmost vertex v_8 (extension denoted as #3) and ending at the root (extension denoted as #6). Thus, the forward extension (v_8, v_x), denoted #3, comes before the forward extension (v_7, v_x), denoted #4, and so on.

11.2.1 Canonical Code

When generating candidates using rightmost path extensions, it is possible that duplicate, that is, isomorphic, graphs are generated via different extensions. Among the isomorphic candidates, we need to keep only one for further extension, whereas the others can be pruned to avoid redundant computation. The main idea is that if we can somehow sort or rank the isomorphic graphs, we can pick the *canonical representative*, say the one with the least rank, and extend only that graph.

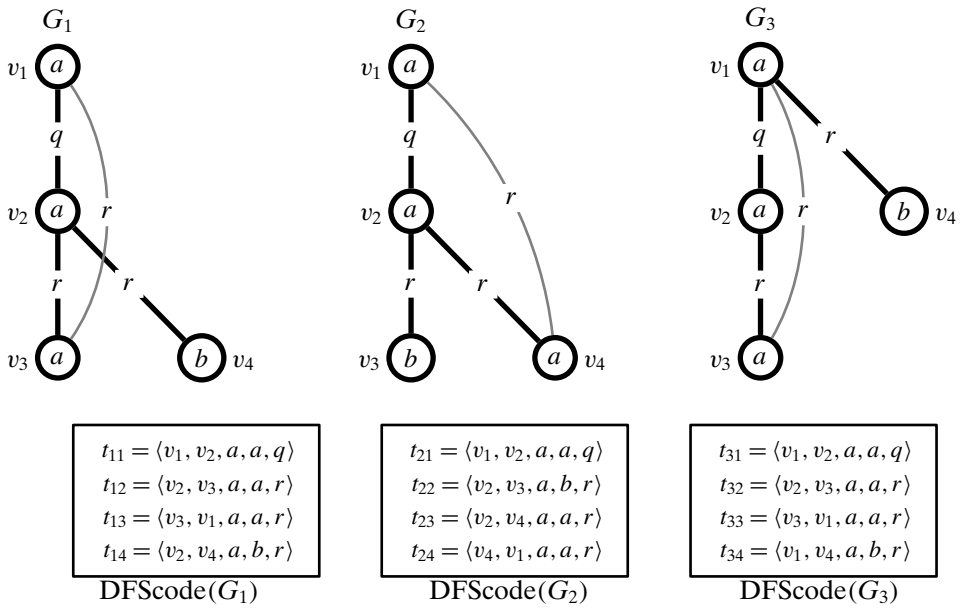


Figure 11.6. Canonical DFS code. G_1 is canonical, whereas G_2 and G_3 are noncanonical. Vertex label set $\Sigma_V = \{a, b\}$, and edge label set $\Sigma_E = \{q, r\}$. The vertices are numbered in DFS order.

Let G be a graph and let T_G be a DFS spanning tree for G. The DFS tree T_G defines an ordering of both the nodes and edges in G. The DFS node ordering is obtained by numbering the nodes consecutively in the order they are visited in the DFS walk. We assume henceforth that for a pattern graph G the nodes are numbered according to their position in the DFS ordering, so that $i < j$ implies that v_i comes before v_j in the DFS walk. The DFS edge ordering is obtained by following the edges between consecutive nodes in DFS order, with the condition that all the backward edges incident with vertex v_i are listed before any of the forward edges incident with it. The *DFS code* for a graph G, for a given DFS tree T_G, denoted DFScode(G), is defined as the sequence of extended edge tuples of the form $\langle v_i, v_j, L(v_i), L(v_j), L(v_i, v_j) \rangle$ listed in the DFS edge order.

Example 11.6. Figure 11.6 shows the DFS codes for three graphs, which are all isomorphic to each other. The graphs have node and edge labels drawn from the label sets $\Sigma_V = \{a, b\}$ and $\Sigma_E = \{q, r\}$. The edge labels are shown centered on the edges. The bold edges comprise the DFS tree for each graph. For G_1, the DFS node ordering is v_1, v_2, v_3, v_4, whereas the DFS edge ordering is (v_1, v_2), (v_2, v_3), (v_3, v_1), and (v_2, v_4). Based on the DFS edge ordering, the first tuple in the DFS code for G_1 is therefore $\langle v_1, v_2, a, a, q \rangle$. The next tuple is $\langle v_2, v_3, a, a, r \rangle$ and so on. The DFS code for each graph is shown in the corresponding box below the graph.

Canonical DFS Code

A subgraph is *canonical* if it has the smallest DFS code among all possible isomorphic graphs, with the ordering between codes defined as follows. Let t_1 and t_2 be any two

DFS code tuples:

$$t_1 = \langle v_i, v_j, L(v_i), L(v_j), L(v_i, v_j) \rangle$$
$$t_2 = \langle v_x, v_y, L(v_x), L(v_y), L(v_x, v_y) \rangle$$

We say that t_1 is smaller than t_2, written $t_1 < t_2$, iff

$$\begin{aligned} &\text{i) } (v_i, v_j) <_e (v_x, v_y), \text{ or} \\ &\text{ii) } (v_i, v_j) = (v_x, v_y) \text{ and} \\ &\qquad \langle L(v_i), L(v_j), L(v_i, v_j) \rangle <_l \langle L(v_x), L(v_y), L(v_x, v_y) \rangle \end{aligned} \tag{11.1}$$

where $<_e$ is an ordering on the edges and $<_l$ is an ordering on the vertex and edge labels. The *label order* $<_l$ is the standard lexicographic order on the vertex and edge labels. The *edge order* $<_e$ is derived from the rules for rightmost path extension, namely that all of a node's backward extensions must be considered before any forward edge from that node, and deep DFS trees are preferred over bushy DFS trees. Formally, Let $e_{ij} = (v_i, v_j)$ and $e_{xy} = (v_x, v_y)$ be any two edges. We say that $e_{ij} <_e e_{xy}$ iff

Condition (1) If e_{ij} and e_{xy} are both forward edges, then (a) $j < y$, or (b) $j = y$ and $i > x$. That is, (a) a forward extension to a node earlier in the DFS node order is smaller, or (b) if both the forward edges point to a node with the same DFS node order, then the forward extension from a node deeper in the tree is smaller.

Condition (2) If e_{ij} and e_{xy} are both backward edges, then (a) $i < x$, or (b) $i = x$ and $j < y$. That is, (a) a backward edge from a node earlier in the DFS node order is smaller, or (b) if both the backward edges originate from a node with the same DFS node order, then the backward edge to a node earlier in DFS node order (i.e., closer to the root along the rightmost path) is smaller.

Condition (3) If e_{ij} is a forward and e_{xy} is a backward edge, then $j \leq x$. That is, a forward edge to a node earlier in the DFS node order is smaller than a backward edge from that node or any node that comes after it in DFS node order.

Condition (4) If e_{ij} is a backward and e_{xy} is a forward edge, then $i < y$. That is, a backward edge from a node earlier in DFS node order is smaller than a forward edge to any later node.

Given any two DFS codes, we can compare them tuple by tuple to check which is smaller. In particular, the *canonical DFS code* for a graph G is defined as follows:

$$\mathcal{C} = \min_{G'} \Big\{ \text{DFScode}(G') \mid G' \text{ is isomorphic to } G \Big\}$$

Given a candidate subgraph G, we can first determine whether its DFS code is canonical or not. Only canonical graphs need to be retained for extension, whereas noncanonical candidates can be removed from further consideration.

Example 11.7. Consider the DFS codes for the three graphs shown in Figure 11.6. Comparing G_1 and G_2, we find that $t_{11} = t_{21}$, but $t_{12} < t_{22}$ because $\langle a, a, r\rangle <_l \langle a, b, r\rangle$. Comparing the codes for G_1 and G_3, we find that the first three tuples are equal for both the graphs, but $t_{14} < t_{34}$ because

$$(v_i, v_j) = (v_2, v_4) <_e (v_1, v_4) = (v_x, v_y)$$

due to condition (1) above. That is, both are forward edges, and we have $v_j = v_4 = v_y$ with $v_i = v_2 > v_1 = v_x$. In fact, it can be shown that the code for G_1 is the canonical DFS code for all graphs isomorphic to G_1. Thus, G_1 is the canonical candidate.

11.3 THE GSPAN ALGORITHM

We describe the gSpan algorithm to mine all frequent subgraphs from a database of graphs. Given a database $\mathbf{D} = \{G_1, G_2, \ldots, G_n\}$ comprising n graphs, and given a minimum support threshold *minsup*, the goal is to enumerate all (connected) subgraphs G that are frequent, that is, $sup(G) \geq minsup$. In gSpan, each graph is represented by its canonical DFS code, so that the task of enumerating frequent subgraphs is equivalent to the task of generating all canonical DFS codes for frequent subgraphs. Algorithm 11.1 shows the pseudo-code for gSpan.

gSpan enumerates patterns in a depth-first manner, starting with the empty code. Given a canonical and frequent code C, gSpan first determines the set of possible edge extensions along the rightmost path (line 1). The function RIGHTMOSTPATH-EXTENSIONS returns the set of edge extensions along with their support values, $\mathcal{E}$. Each extended edge t in $\mathcal{E}$ leads to a new candidate DFS code $C' = C \cup \{t\}$, with support $sup(C) = sup(t)$ (lines 3–4). For each new candidate code, gSpan checks whether it is frequent and canonical, and if so gSpan recursively extends C' (lines 5–6). The algorithm stops when there are no more frequent and canonical extensions possible.

ALGORITHM 11.1. Algorithm GSPAN

```
   // Initial Call: C ← ∅
   GSPAN (C, D, minsup):
1  E ← RIGHTMOSTPATH-EXTENSIONS(C, D) // extensions and
       supports
2  foreach (t, sup(t)) ∈ E do
3      C' ← C ∪ t // extend the code with extended edge tuple t
4      sup(C') ← sup(t) // record the support of new extension
       // recursively call GSPAN if code is frequent and
          canonical
5      if sup(C') ≥ minsup and ISCANONICAL (C') then
6          GSPAN (C', D, minsup)
```

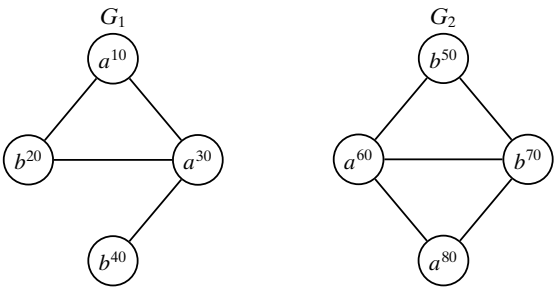


Figure 11.7. Example graph database.

Example 11.8. Consider the example graph database comprising G_1 and G_2 shown in Figure 11.7. Let $minsup = 2$, that is, assume that we are interested in mining subgraphs that appear in both the graphs in the database. For each graph the node labels and node numbers are both shown, for example, the node a^{10} in G_1 means that node 10 has label a.

Figure 11.8 shows the candidate patterns enumerated by gSpan. For each candidate the nodes are numbered in the DFS tree order. The solid boxes show frequent subgraphs, whereas the dotted boxes show the infrequent ones. The dashed boxes represent noncanonical codes. Subgraphs that do not occur even once are not shown. The figure also shows the DFS codes and their corresponding graphs.

The mining process begins with the empty DFS code C_0 corresponding to the empty subgraph. The set of possible 1-edge extensions comprises the new set of candidates. Among these, C_3 is pruned because it is not canonical (it is isomorphic to C_2), whereas C_4 is pruned because it is not frequent. The remaining two candidates, C_1 and C_2, are both frequent and canonical, and are thus considered for further extension. The depth-first search considers C_1 before C_2, with the rightmost path extensions of C_1 being C_5 and C_6. However, C_6 is not canonical; it is isomorphic to C_5, which has the canonical DFS code. Further extensions of C_5 are processed recursively. Once the recursion from C_1 completes, gSpan moves on to C_2, which will be recursively extended via rightmost edge extensions as illustrated by the subtree under C_2. After processing C_2, gSpan terminates because no other frequent and canonical extensions are found. In this example, C_{12} is a maximal frequent subgraph, that is, no supergraph of C_{12} is frequent.

This example also shows the importance of duplicate elimination via canonical checking. The groups of isomorphic subgraphs encountered during the execution of gSpan are as follows: $\{C_2, C_3\}$, $\{C_5, C_6, C_{17}\}$, $\{C_7, C_{19}\}$, $\{C_9, C_{25}\}$, $\{C_{20}, C_{21}, C_{22}, C_{24}\}$, and $\{C_{12}, C_{13}, C_{14}\}$. Within each group the first graph is canonical and thus the remaining codes are pruned.

For a complete description of gSpan we have to specify the algorithm for enumerating the rightmost path extensions and their support, so that infrequent patterns can be eliminated, and the procedure for checking whether a given DFS code is canonical, so that duplicate patterns can be pruned. These are detailed next.

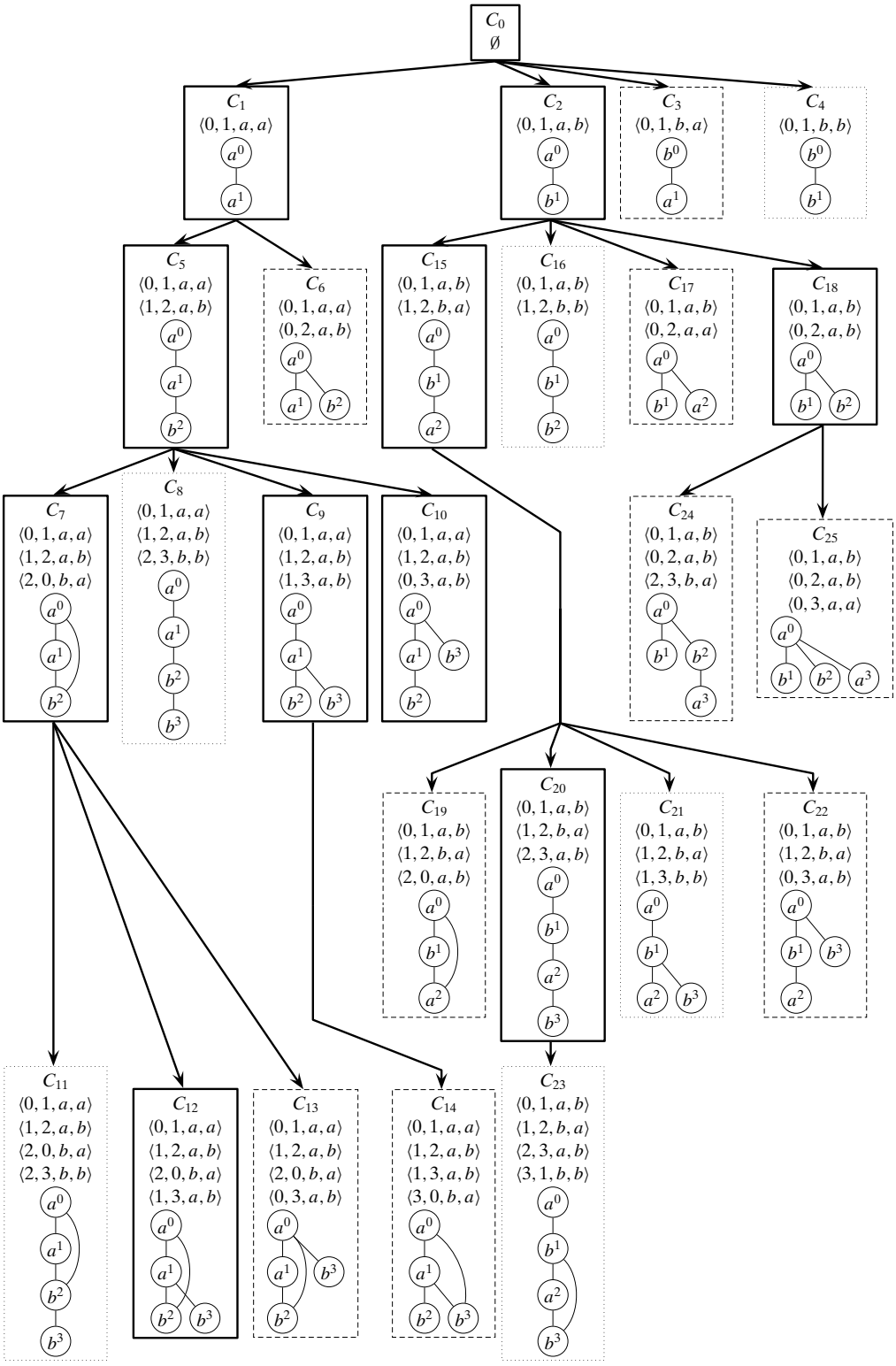


Figure 11.8. Frequent graph mining: $minsup = 2$. Solid boxes indicate the frequent subgraphs, dotted the infrequent, and dashed the noncanonical subgraphs.

11.3.1 Extension and Support Computation

The support computation task is to find the number of graphs in the database $\mathbf{D}$ that contain a candidate subgraph, which is very expensive because it involves subgraph isomorphism checks. gSpan combines the tasks of enumerating candidate extensions and support computation.

Assume that $\mathbf{D} = \{G_1, G_2, \ldots, G_n\}$ comprises n graphs. Let $C = \{t_1, t_2, \ldots, t_k\}$ denote a frequent canonical DFS code comprising k edges, and let $G(C)$ denote the graph corresponding to code C. The task is to compute the set of possible rightmost path extensions from C, along with their support values, which is accomplished via the pseudo-code in Algorithm 11.2.

Given code C, gSpan first records the nodes on the rightmost path (R), and the rightmost child (u_r). Next, gSpan considers each graph $G_i \in \mathbf{D}$. If $C = \emptyset$, then each distinct label tuple of the form $\langle L(x), L(y), L(x,y)\rangle$ for adjacent nodes x and y in G_i contributes a forward extension $\langle 0, 1, L(x), L(y), L(x,y)\rangle$ (lines 6-8). On the other hand, if C is not empty, then gSpan enumerates all possible subgraph isomorphisms Φ_i between the code C and graph G_i via the function SUBGRAPHISOMORPHISMS (line 10). Given subgraph isomorphism $\phi \in \Phi_i$, gSpan finds all possible forward and backward edge extensions, and stores them in the extension set $\mathcal{E}$.

Backward extensions (lines 12–15) are allowed only from the rightmost child u_r in C to some other node on the rightmost path R. The method considers each neighbor x of $\phi(u_r)$ in G_i and checks whether it is a mapping for some vertex $v = \phi^{-1}(x)$ along the rightmost path R in C. If the edge (u_r, v) does not already exist in C, it is a new extension, and the extended tuple $b = \langle u_r, v, L(u_r), L(v), L(u_r, v)\rangle$ is added to the set of extensions $\mathcal{E}$, along with the graph id i that contributed to that extension.

Forward extensions (lines 16–19) are allowed only from nodes on the rightmost path R to new nodes. For each node u in R, the algorithm finds a neighbor x in G_i that is not in a mapping from some node in C. For each such node x, the forward extension $f = \langle u, u_r + 1, L(\phi(u)), L(x), L(\phi(u), x)\rangle$ is added to $\mathcal{E}$, along with the graph id i. Because a forward extension adds a new vertex to the graph $G(C)$, the id of the new node in C must be $u_r + 1$, that is, one more than the highest numbered node in C, which by definition is the rightmost child u_r.

Once all the backward and forward extensions have been cataloged over all graphs G_i in the database $\mathbf{D}$, we compute their support by counting the number of distinct graph ids that contribute to each extension. Finally, the method returns the set of all extensions and their supports in sorted order (increasing) based on the tuple comparison operator in Eq. (11.1).

Example 11.9. Consider the canonical code C and the corresponding graph $G(C)$ shown in Figure 11.9a. For this code all the vertices are on the rightmost path, that is, $R = \{0, 1, 2\}$, and the rightmost child is $u_r = 2$.

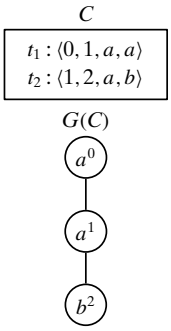
The sets of all possible isomorphisms from C to graphs G_1 and G_2 in the database (shown in Figure 11.7) are listed in Figure 11.9b as Φ_1 and Φ_2. For example, the first isomorphism $\phi_1 : G(C) \rightarrow G_1$ is defined as

$$\phi_1(0) = 10 \qquad \phi_1(1) = 30 \qquad \phi_1(2) = 20$$

ALGORITHM 11.2. Rightmost Path Extensions and Their Support

```
RIGHTMOSTPATH-EXTENSIONS (C, D):
1  R ← nodes on the rightmost path in C
2  u_r ← rightmost child in C // dfs number
3  E ← ∅ // set of extensions from C
4  foreach G_i ∈ D, i = 1, ..., n do
5      if C = ∅ then
           // add distinct label tuples in G_i as forward
              extensions
6          foreach distinct ⟨L(x), L(y), L(x, y)⟩ ∈ G_i do
7              f = ⟨0, 1, L(x), L(y), L(x, y)⟩
8              Add tuple f to E along with graph id i
9      else
10         Φ_i = SUBGRAPHISOMORPHISMS(C, G_i)
11         foreach isomorphism φ ∈ Φ_i do
               // backward extensions from rightmost child
12             foreach x ∈ N_{G_i}(φ(u_r)) such that ∃v ← φ^{-1}(x) do
13                 if v ∈ R and (u_r, v) ∉ G(C) then
14                     b = ⟨u_r, v, L(u_r), L(v), L(u_r, v)⟩
15                     Add tuple b to E along with graph id i
               // forward extensions from nodes on rightmost path
16             foreach u ∈ R do
17                 foreach x ∈ N_{G_i}(φ(u)) and ∄φ^{-1}(x) do
18                     f = ⟨u, u_r + 1, L(φ(u)), L(x), L(φ(u), x)⟩
19                     Add tuple f to E along with graph id i
   // Compute the support of each extension
20 foreach distinct extension s ∈ E do
21     sup(s) = number of distinct graph ids that support tuple s
22 return set of pairs ⟨s, sup(s)⟩ for extensions s ∈ E, in tuple sorted order
```

The list of possible backward and forward extensions for each isomorphism is shown in Figure 11.9c. For example, there are two possible edge extensions from the isomorphism ϕ_1. The first is a backward edge extension $\langle 2, 0, b, a\rangle$, as $(20, 10)$ is a valid backward edge in G_1. That is, the node $x = 10$ is a neighbor of $\phi(2) = 20$ in G_1, $\phi^{-1}(10) = 0 = v$ is on the rightmost path, and the edge $(2, 0)$ is not already in $G(C)$, which satisfy the backward extension steps in lines 12–15 in Algorithm 11.2. The second extension is a forward one $\langle 1, 3, a, b\rangle$, as $\langle 30, 40, a, b\rangle$ is a valid extended edge in G_1. That is, $x = 40$ is a neighbor of $\phi(1) = 30$ in G_1, and node 40 has not already been mapped to any node in $G(C)$, that is, $\phi_1^{-1}(40)$ does not exist. These conditions satisfy the forward extension steps in lines 16–19 in Algorithm 11.2.



(a) Code C and graph $G(C)$

Φ	ϕ	0	1	2
Φ_1	ϕ_1	10	30	20
	ϕ_2	10	30	40
	ϕ_3	30	10	20
Φ_2	ϕ_4	60	80	70
	ϕ_5	80	60	50
	ϕ_6	80	60	70

(b) Subgraph isomorphisms

Id	ϕ	Extensions
G_1	ϕ_1	$\{\langle 2, 0, b, a \rangle, \langle 1, 3, a, b \rangle\}$
	ϕ_2	$\{\langle 1, 3, a, b \rangle, \langle 0, 3, a, b \rangle\}$
	ϕ_3	$\{\langle 2, 0, b, a \rangle, \langle 0, 3, a, b \rangle\}$
G_2	ϕ_4	$\{\langle 2, 0, b, a \rangle, \langle 2, 3, b, b \rangle, \langle 0, 3, a, b \rangle\}$
	ϕ_5	$\{\langle 2, 3, b, b \rangle, \langle 1, 3, a, b \rangle\}$
	ϕ_6	$\{\langle 2, 0, b, a \rangle, \langle 2, 3, b, b \rangle, \langle 1, 3, a, b \rangle\}$

(c) Edge extensions

Extension	Support
$\langle 2, 0, b, a \rangle$	2
$\langle 2, 3, b, b \rangle$	1
$\langle 1, 3, a, b \rangle$	2
$\langle 0, 3, a, b \rangle$	2

(d) Extensions (sorted) and supports

Figure 11.9. Rightmost path extensions.

Given the set of all the edge extensions, and the graph ids that contribute to them, we obtain support for each extension by counting how many graphs contribute to it. The final set of extensions, in sorted order, along with their support values is shown in Figure 11.9d. With $minsup = 2$, the only infrequent extension is $\langle 2, 3, b, b \rangle$.

Subgraph Isomorphisms

The key step in listing the edge extensions for a given code C is to enumerate all the possible isomorphisms from C to each graph $G_i \in \mathbf{D}$. The function SUBGRAPHISOMORPHISMS, shown in Algorithm 11.3, accepts a code C and a graph G, and returns the set of all isomorphisms between C and G. The set of isomorphisms Φ is initialized by mapping vertex 0 in C to each vertex x in G that shares the same label as 0, that is, if $L(x) = L(0)$ (line 1). The method considers each tuple t_i in C and extends the current set of partial isomorphisms. Let $t_i = \langle u, v, L(u), L(v), L(u, v) \rangle$. We have to check if each isomorphism $\phi \in \Phi$ can be extended in G using the information from t_i (lines 5–12). If t_i is a forward edge, then we seek a neighbor x of $\phi(u)$ in G such that x has not already been mapped to some vertex in C, that is, $\phi^{-1}(x)$ should not exist, and the node and edge labels should match, that is, $L(x) = L(v)$, and $L(\phi(u), x) = L(u, v)$. If so, ϕ can be extended with the mapping $\phi(v) \rightarrow x$. The new extended isomorphism, denoted ϕ', is added to the initially empty set of isomorphisms Φ'. If t_i is a backward edge, we have to check if $\phi(v)$ is a neighbor of $\phi(u)$ in G. If so, we add the current isomorphism ϕ to Φ'. Thus,

ALGORITHM 11.3. Enumerate Subgraph Isomorphisms

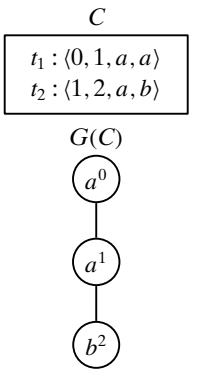
```
SUBGRAPHISOMORPHISMS (C = {t_1, t_2, ..., t_k}, G):
1  Φ ← {φ(0) → x | x ∈ G and L(x) = L(0)}
2  foreach t_i ∈ C, i = 1, ..., k do
3      ⟨u, v, L(u), L(v), L(u, v)⟩ ← t_i // expand extended edge t_i
4      Φ' ← ∅ // partial isomorphisms including t_i
5      foreach partial isomorphism φ ∈ Φ do
6          if v > u then
               // forward edge
7              foreach x ∈ N_G(φ(u)) do
8                  if ∄φ^{-1}(x) and L(x) = L(v) and L(φ(u), x) = L(u, v) then
9                      φ' ← φ ∪ {φ(v) → x}
10                     Add φ' to Φ'
11         else
               // backward edge
12             if φ(v) ∈ N_{G_j}(φ(u)) then Add φ to Φ' // valid isomorphism
13     Φ ← Φ' // update partial isomorphisms
14 return Φ
```

only those isomorphisms that can be extended in the forward case, or those that satisfy the backward edge, are retained for further checking. Once all the extended edges in C have been processed, the set Φ contains all the valid isomorphisms from C to G.

Example 11.10. Figure 11.10 illustrates the subgraph isomorphism enumeration algorithm from the code C to each of the graphs G_1 and G_2 in the database shown in Figure 11.7.

For G_1, the set of isomorphisms Φ is initialized by mapping the first node of C to all nodes labeled a in G_1 because $L(0) = a$. Thus, $\Phi = \{\phi_1(0) \rightarrow 10, \phi_2(0) \rightarrow 30\}$. We next consider each tuple in C, and see which isomorphisms can be extended. The first tuple $t_1 = \langle 0, 1, a, a \rangle$ is a forward edge, thus for ϕ_1, we consider neighbors x of 10 that are labeled a and not included in the isomorphism yet. The only other vertex that satisfies this condition is 30; thus the isomorphism is extended by mapping $\phi_1(1) \rightarrow 30$. In a similar manner the second isomorphism ϕ_2 is extended by adding $\phi_2(1) \rightarrow 10$, as shown in Figure 11.10. For the second tuple $t_2 = \langle 1, 2, a, b \rangle$, the isomorphism ϕ_1 has two possible extensions, as 30 has two neighbors labeled b, namely 20 and 40. The extended mappings are denoted ϕ_1' and ϕ_1''. For ϕ_2 there is only one extension.

The isomorphisms of C in G_2 can be found in a similar manner. The complete sets of isomorphisms in each database graph are shown in Figure 11.10.



Initial Φ

id	ϕ	0
G_1	ϕ_1	10
	ϕ_2	30
G_2	ϕ_3	60
	ϕ_4	80

Add t_1

id	ϕ	0, 1
G_1	ϕ_1	10, 30
	ϕ_2	30, 10
G_2	ϕ_3	60, 80
	ϕ_4	80, 60

Add t_2

id	ϕ	0, 1, 2
G_1	ϕ_1'	10, 30, 20
	ϕ_1''	10, 30, 40
	ϕ_2	30, 10, 20
G_2	ϕ_3	60, 80, 70
	ϕ_4'	80, 60, 50
	ϕ_4''	80, 60, 70

Figure 11.10. Subgraph isomorphisms.

11.3.2 Canonicality Checking

Given a DFS code $C = \{t_1, t_2, \ldots, t_k\}$ comprising k extended edge tuples and the corresponding graph $G(C)$, the task is to check whether the code C is canonical. This can be accomplished by trying to reconstruct the canonical code C^* for $G(C)$ in an iterative manner starting from the empty code and selecting the least rightmost path extension at each step, where the least edge extension is based on the extended tuple comparison operator in Eq. (11.1). If at any step the current (partial) canonical DFS code C^* is smaller than C, then we know that C cannot be canonical and can thus be pruned. On the other hand, if no smaller code is found after k extensions then C must be canonical. The pseudo-code for canonicality checking is given in Algorithm 11.4. The method can be considered as a restricted version of gSpan in that the graph $G(C)$ plays the role of a graph in the database, and C^* plays the role of a candidate extension. The key difference is that we consider only the smallest rightmost path edge extension among all the possible candidate extensions.

ALGORITHM 11.4. Canonicality Checking: Algorithm IsCanonical

```
IsCanonical (C):
1  D_C ← {G(C)} // graph corresponding to code C
2  C* ← ∅ // initialize canonical DFScode
3  for i = 1 ··· k do
4      E = RightMostPath-Extensions(C*, D_C) // extensions of C*
5      (s_i, sup(s_i)) ← min{E} // least rightmost edge extension of C*
6      if s_i < t_i then
7          return false // C* is smaller, thus C is not canonical
8      C* ← C* ∪ s_i
9  return true // no smaller code exists; C is canonical
```

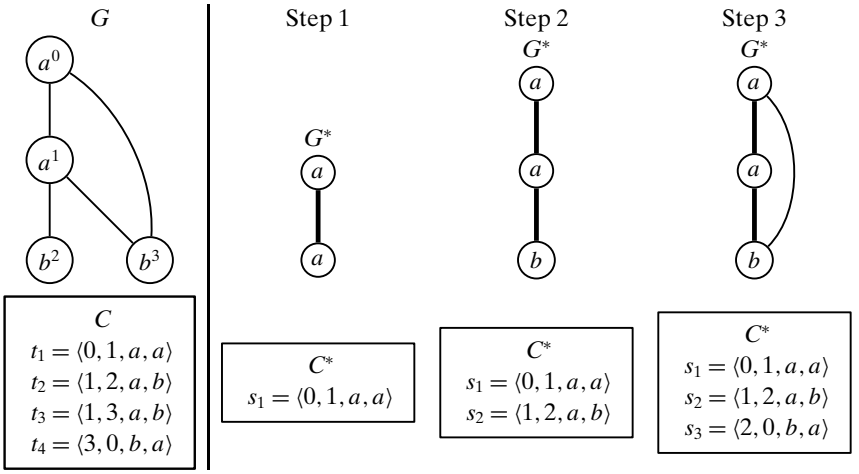


Figure 11.11. Canonicality checking.

Example 11.11. Consider the subgraph candidate C_{14} from Figure 11.8, which is replicated as graph G in Figure 11.11, along with its DFS code C. From an initial canonical code $C^* = \emptyset$, the smallest rightmost edge extension s_1 is added in Step 1. Because $s_1 = t_1$, we proceed to the next step, which finds the smallest edge extension s_2. Once again $s_2 = t_2$, so we proceed to the third step. The least possible edge extension for G^* is the extended edge s_3. However, we find that $s_3 < t_3$, which means that C cannot be canonical, and there is no need to try further edge extensions.

11.4 FURTHER READING

The gSpan algorithm was described in Yan and Han (2002), along with the notion of canonical DFS code. A different notion of canonical graphs using canonical adjacency matrices was described in Huan, Wang, and Prins (2003). Level-wise algorithms to mine frequent subgraphs appear in Kuramochi and Karypis (2001) and Inokuchi, Washio, and Motoda (2000). Markov chain Monte Carlo methods to sample a set of representative graph patterns were proposed in Al Hasan and Zaki (2009). For an efficient algorithm to mine frequent tree patterns see Zaki (2002).

Al Hasan, M. and Zaki, M. J. (2009). "Output space sampling for graph patterns." *Proceedings of the VLDB Endowment*, 2 (1): 730–741.

Huan, J., Wang, W., and Prins, J. (2003). "Efficient mining of frequent subgraphs in the presence of isomorphism." *In Proceedings of the IEEE International Conference on Data Mining*. IEEE, pp. 549–552.

Inokuchi, A., Washio, T., and Motoda, H. (2000). "An apriori-based algorithm for mining frequent substructures from graph data." *In Proceedings of the European Conference on Principles of Data Mining and Knowledge Discovery*. Springer, pp. 13–23.

Kuramochi, M. and Karypis, G. (2001). "Frequent subgraph discovery." *In Proceedings of the IEEE International Conference on Data Mining*. IEEE, pp. 313–320.

Yan, X. and Han, J. (2002). "gSpan: Graph-based substructure pattern mining." *In Proceedings of the IEEE International Conference on Data Mining*. IEEE, pp. 721–724.

Zaki, M. J. (2002). "Efficiently mining frequent trees in a forest." *In Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, pp. 71–80.

11.5 EXERCISES

Q1. Find the canonical DFS code for the graph in Figure 11.12. Try to eliminate some codes without generating the complete search tree. For example, you can eliminate a code if you can show that it will have a larger code than some other code.

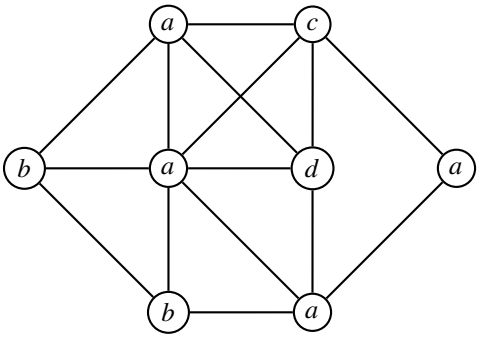


Figure 11.12. Graph for Q1.

Q2. Given the graph in Figure 11.13. Mine all the frequent subgraphs with $minsup = 1$. For each frequent subgraph, also show its canonical code.

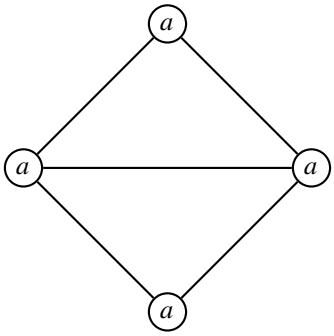


Figure 11.13. Graph for Q2.

Q3. Consider the graph shown in Figure 11.14. Show all its isomorphic graphs and their DFS codes, and find the canonical representative (you may omit isomorphic graphs that can definitely not have canonical codes).

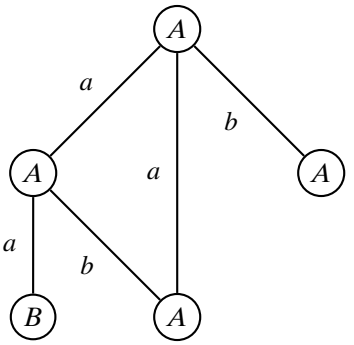


Figure 11.14. Graph for Q3.

Q4. Given the graphs in Figure 11.15, separate them into isomorphic groups.

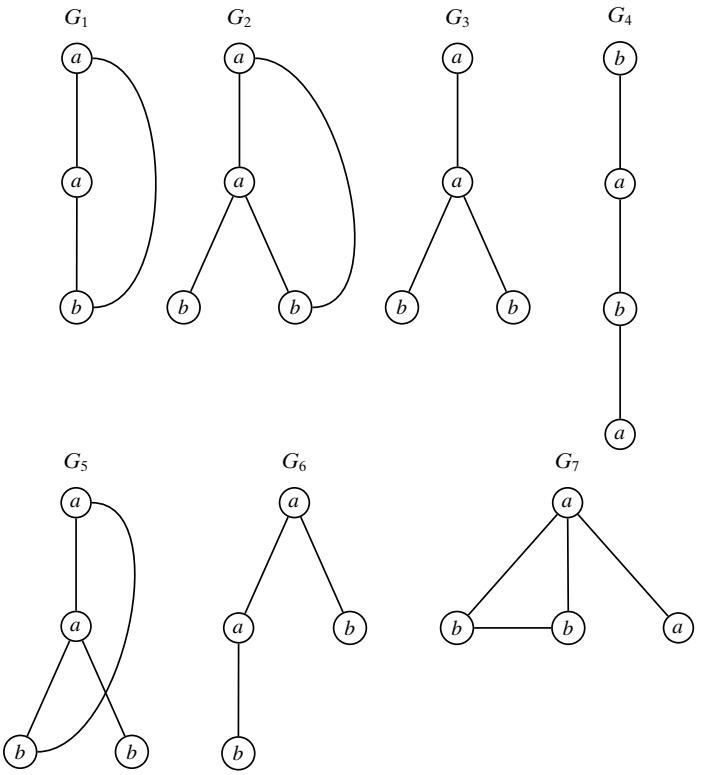


Figure 11.15. Data for Q4.

Q5. Given the graph in Figure 11.16. Find the *maximum* DFS code for the graph, subject to the constraint that all extensions (whether forward or backward) are done only from the right most path.

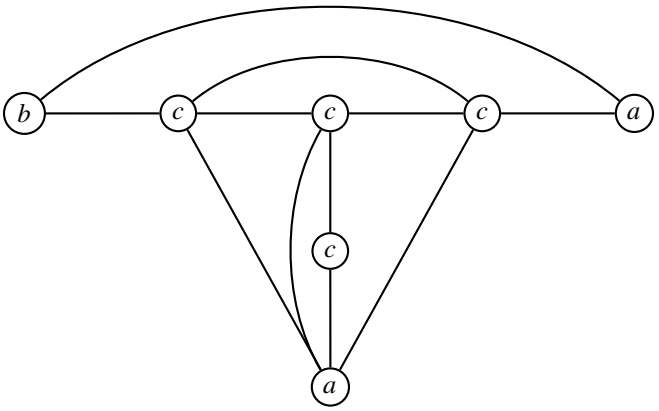


Figure 11.16. Graph for Q5.

Q6. For an edge labeled undirected graph $G = (V, E)$, define its labeled adjacency matrix $\mathbf{A}$ as follows:

$$\mathbf{A}(i,j) = \begin{cases} L(v_i) & \text{if } i = j \\ L(v_i, v_j) & \text{if } (v_i, v_j) \in E \\ 0 & \text{Otherwise} \end{cases}$$

where $L(v_i)$ is the label for vertex v_i and $L(v_i, v_j)$ is the label for edge (v_i, v_j). In other words, the labeled adjacency matrix has the node labels on the main diagonal, and it has the label of the edge (v_i, v_j) in cell $\mathbf{A}(i, j)$. Finally, a 0 in cell $\mathbf{A}(i, j)$ means that there is no edge between v_i and v_j.

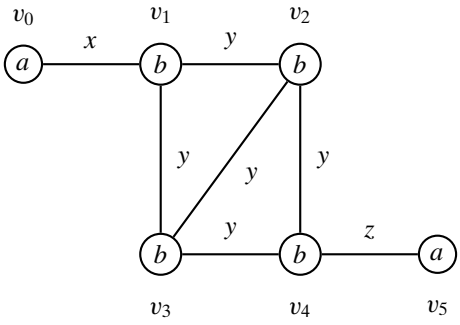


Figure 11.17. Graph for Q6.

Given a particular permutation of the vertices, a *matrix code* for the graph is obtained by concatenating the lower triangular submatrix of $\mathbf{A}$ row-by-row. For

example, one possible matrix corresponding to the default vertex permutation $v_0v_1v_2v_3v_4v_5$ for the graph in Figure 11.17 is given as

a					
x	b				
0	y	b			
0	y	y	b		
0	0	y	y	b	
0	0	0	0	z	a

The code for the matrix above is $axb0yb0yyb00yyb0000za$. Given the total ordering on the labels

$$0 < a < b < x < y < z$$

find the maximum matrix code for the graph in Figure 11.17. That is, among all possible vertex permutations and the corresponding matrix codes, you have to choose the lexicographically largest code.

CHAPTER 12 Pattern and Rule Assessment

In this chapter we discuss how to assess the significance of the mined frequent patterns, as well as the association rules derived from them. Ideally, the mined patterns and rules should satisfy desirable properties such as conciseness, novelty, utility, and so on. We outline several rule and pattern assessment measures that aim to quantify different properties of the mined results. Typically, the question of whether a pattern or rule is interesting is to a large extent a subjective one. However, we can certainly try to eliminate rules and patterns that are not statistically significant. Methods to test for the statistical significance and to obtain confidence bounds on the test statistic value are also considered in this chapter.

12.1 RULE AND PATTERN ASSESSMENT MEASURES

Let $\mathcal{I}$ be a set of items and $\mathcal{T}$ a set of tids, and let $\mathbf{D} \subseteq \mathcal{T} \times \mathcal{I}$ be a binary database. Recall that an *association rule* is an expression $X \longrightarrow Y$, where X and Y are itemsets, i.e., $X, Y \subseteq \mathcal{I}$, and $X \cap Y = \emptyset$. We call X the antecedent of the rule and Y the consequent.

The tidset for an itemset X is the set of all tids that contain X, given as

$$\mathbf{t}(X) = \left\{ t \in \mathcal{T} \mid X \text{ is contained in } t \right\}$$

The support of X is thus $sup(X) = |\mathbf{t}(X)|$. In the discussion that follows we use the short form XY to denote the union, $X \cup Y$, of the itemsets X and Y.

Given a frequent itemset $Z \in \mathcal{F}$, where $\mathcal{F}$ is the set of all frequent itemsets, we can derive different association rules by considering each proper subset of Z as the antecedent and the remaining items as the consequent, that is, for each $Z \in \mathcal{F}$, we can derive a set of rules of the form $X \longrightarrow Y$, where $X \subset Z$ and $Y = Z \setminus X$.

12.1.1 Rule Assessment Measures

Different rule interestingness measures try to quantify the dependence between the consequent and antecedent. Below we review some of the common rule assessment measures, starting with support and confidence.

Table 12.1. Example Dataset

Tid	Items
1	$ABDE$
2	BCE
3	$ABDE$
4	$ABCE$
5	$ABCDE$
6	BCD

Table 12.2. Frequent itemsets with $minsup = 3$ (relative minimum support 50%)

sup	*rsup*	Itemsets
3	0.5	$ABD, ABDE, AD, ADE, BCE, BDE, CE, DE$
4	0.67	$A, C, D, AB, ABE, AE, BC, BD$
5	0.83	E, BE
6	1.0	B

Support

The *support* of the rule is defined as the number of transactions that contain both X and Y, that is,

$$sup(X \longrightarrow Y) = sup(XY) = |\mathbf{t}(XY)| \tag{12.1}$$

The *relative support* is the fraction of transactions that contain both X and Y, that is, the empirical joint probability of the items comprising the rule

$$rsup(X \longrightarrow Y) = P(XY) = rsup(XY) = \frac{sup(XY)}{|\mathbf{D}|}$$

Typically we are interested in frequent rules, with $sup(X \longrightarrow Y) \geq minsup$, where *minsup* is a user-specified minimum support threshold. When minimum support is specified as a fraction then relative support is implied. Notice that (relative) support is a symmetric measure because $sup(X \longrightarrow Y) = sup(Y \longrightarrow X)$.

Example 12.1. We illustrate the rule assessment measures using the example binary dataset **D** in Table 12.1, shown in transactional form. It has six transactions over a set of five items $\mathcal{I} = \{A, B, C, D, E\}$. The set of all frequent itemsets with $minsup = 3$ is listed in Table 12.2. The table shows the support and relative support for each frequent itemset. The association rule $AB \longrightarrow DE$ derived from the itemset $ABDE$ has support $sup(AB \longrightarrow DE) = sup(ABDE) = 3$, and its relative support is $rsup(AB \longrightarrow DE) = sup(ABDE)/|\mathbf{D}| = 3/6 = 0.5$.

Confidence

The *confidence* of a rule is the conditional probability that a transaction contains the consequent Y given that it contains the antecedent X:

$$conf(X \longrightarrow Y) = P(Y|X) = \frac{P(XY)}{P(X)} = \frac{rsup(XY)}{rsup(X)} = \frac{sup(XY)}{sup(X)}$$

Table 12.3. Rule confidence

Rule			*conf*
A	$\longrightarrow$	E	1.00
E	$\longrightarrow$	A	0.80
B	$\longrightarrow$	E	0.83
E	$\longrightarrow$	B	1.00
E	$\longrightarrow$	BC	0.60
BC	$\longrightarrow$	E	0.75

Typically we are interested in high confidence rules, with $conf(X \longrightarrow Y) \geq minconf$, where *minconf* is a user-specified minimum confidence value. Confidence is not a symmetric measure because by definition it is conditional on the antecedent.

Example 12.2. Table 12.3 shows some example association rules along with their confidence generated from the example dataset in Table 12.1. For instance, the rule $A \longrightarrow E$ has confidence $sup(AE)/sup(A) = 4/4 = 1.0$. To see the asymmetry of confidence, observe that the rule $E \longrightarrow A$ has confidence $sup(AE)/sup(E) = 4/5 = 0.8$.

Care must be exercised in interpreting the goodness of a rule. For instance, the rule $E \longrightarrow BC$ has confidence $P(BC|E) = 0.60$, that is, given E we have a probability of 60% of finding BC. However, the unconditional probability of BC is $P(BC) = 4/6 = 0.67$, which means that E, in fact, has a deleterious effect on BC.

Lift

Lift is defined as the ratio of the observed joint probability of X and Y to the expected joint probability if they were statistically independent, that is,

$$lift(X \longrightarrow Y) = \frac{P(XY)}{P(X) \cdot P(Y)} = \frac{rsup(XY)}{rsup(X) \cdot rsup(Y)} = \frac{conf(X \longrightarrow Y)}{rsup(Y)}$$

One common use of lift is to measure the surprise of a rule. A lift value close to 1 means that the support of a rule is expected considering the supports of its components. We usually look for values that are much larger (i.e., above expectation) or smaller than 1 (i.e., below expectation).

Notice that lift is a symmetric measure, and it is always larger than or equal to the confidence because it is the confidence divided by the consequent's probability. Lift is also not downward closed, that is, assuming that $X' \subset X$ and $Y' \subset Y$, it can happen that $lift(X' \longrightarrow Y')$ may be higher than $lift(X \longrightarrow Y)$. Lift can be susceptible to noise in small datasets, as rare or infrequent itemsets that occur only a few times can have very high lift values.

Example 12.3. Table 12.4 shows three rules and their lift values, derived from the itemset $ABCE$, which has support $sup(ABCE) = 2$ in our example database in Table 12.1.

Table 12.4. Rule lift

Rule			*lift*
AE	$\longrightarrow$	BC	0.75
CE	$\longrightarrow$	AB	1.00
BE	$\longrightarrow$	AC	1.20

The lift for the rule $AE \longrightarrow BC$ is given as

$$lift(AE \longrightarrow BC) = \frac{rsup(ABCE)}{rsup(AE) \cdot rsup(BC)} = \frac{2/6}{4/6 \times 4/6} = 6/8 = 0.75$$

Since the lift value is less than 1, the observed rule support is less than the expected support. On the other hand, the rule $BE \longrightarrow AC$ has lift

$$lift(BE \longrightarrow AC) = \frac{2/6}{2/6 \times 5/6} = 6/5 = 1.2$$

indicating that it occurs more than expected. Finally, the rule $CE \longrightarrow AB$ has lift equal to 1.0, which means that the observed support and the expected support match.

Example 12.4. It is interesting to compare confidence and lift. Consider the three rules shown in Table 12.5 as well as their relative support, confidence, and lift values. Comparing the first two rules, we can see that despite having lift greater than 1, they provide different information. Whereas $E \longrightarrow AC$ is a weak rule ($conf = 0.4$), $E \longrightarrow AB$ is not only stronger in terms of confidence, but it also has more support. Comparing the second and third rules, we can see that although $B \longrightarrow E$ has lift equal to 1.0, meaning that B and E are independent events, its confidence is higher and so is its support. This example underscores the point that whenever we analyze association rules, we should evaluate them using multiple interestingness measures.

Leverage

Leverage measures the difference between the observed and expected joint probability of XY assuming that X and Y are independent

$$leverage(X \longrightarrow Y) = P(XY) - P(X) \cdot P(Y) = rsup(XY) - rsup(X) \cdot rsup(Y)$$

Leverage gives an "absolute" measure of how surprising a rule is and it should be used together with lift. Like lift it is symmetric.

Example 12.5. Consider the rules shown in Table 12.6, which are based on the example dataset in Table 12.1. The leverage of the rule $ACD \longrightarrow E$ is

$$leverage(ACD \longrightarrow E) = P(ACDE) - P(ACD) \cdot P(E) = 1/6 - 1/6 \times 5/6 = 0.03$$

Similarly, we can calculate the leverage for other rules. The first two rules have the same lift; however, the leverage of the first rule is half that of the second rule, mainly due to the higher support of ACE. Thus, considering lift in isolation may be

Table 12.5. Comparing support, confidence, and lift

Rule			*rsup*	*conf*	*lift*
E	$\longrightarrow$	AC	0.33	0.40	1.20
E	$\longrightarrow$	AB	0.67	0.80	1.20
B	$\longrightarrow$	E	0.83	0.83	1.00

Table 12.6. Rule leverage

Rule			*rsup*	*lift*	*leverage*
ACD	$\longrightarrow$	E	0.17	1.20	0.03
AC	$\longrightarrow$	E	0.33	1.20	0.06
AB	$\longrightarrow$	D	0.50	1.12	0.06
A	$\longrightarrow$	E	0.67	1.20	0.11

misleading because rules with different support may have the same lift. On the other hand, the second and third rules have different lift but the same leverage. Finally, we emphasize the need to consider leverage together with other metrics by comparing the first, second, and fourth rules, which, despite having the same lift, have different leverage values. In fact, the fourth rule $A \longrightarrow E$ may be preferable over the first two because it is simpler and has higher leverage.

Jaccard

The Jaccard coefficient measures the similarity between two sets. When applied as a rule assessment measure it computes the similarity between the tidsets of X and Y:

$$\begin{aligned} jaccard(X \longrightarrow Y) &= \frac{|\mathbf{t}(X) \cap \mathbf{t}(Y)|}{|\mathbf{t}(X) \cup \mathbf{t}(Y)|} \\ &= \frac{sup(XY)}{sup(X) + sup(Y) - sup(XY)} \\ &= \frac{P(XY)}{P(X) + P(Y) - P(XY)} \end{aligned}$$

Jaccard is a symmetric measure.

Example 12.6. Consider the three rules and their Jaccard values shown in Table 12.7. For example, we have

$$jaccard(A \longrightarrow C) = \frac{sup(AC)}{sup(A) + sup(C) - sup(AC)} = \frac{2}{4+4-2} = 2/6 = 0.33$$

Conviction

All of the rule assessment measures we considered above use only the joint probability of X and Y. Define $\neg X$ to be the event that X is not contained in a transaction,

Table 12.7. Jaccard coefficient

Rule			*rsup*	*lift*	*jaccard*
A	$\longrightarrow$	C	0.33	0.75	0.33
A	$\longrightarrow$	E	0.67	1.20	0.80
A	$\longrightarrow$	B	0.67	1.00	0.67

that is, $X \not\subseteq t \in \mathcal{T}$, and likewise for $\neg Y$. There are, in general, four possible events depending on the occurrence or non-occurrence of the itemsets X and Y as depicted in the contingency table shown in Table 12.8.

Conviction measures the expected error of the rule, that is, how often X occurs in a transaction where Y does not. It is thus a measure of the strength of a rule with respect to the complement of the consequent, defined as

$$conv(X \longrightarrow Y) = \frac{P(X) \cdot P(\neg Y)}{P(X\neg Y)} = \frac{1}{lift(X \longrightarrow \neg Y)}$$

If the joint probability of $X\neg Y$ is less than that expected under independence of X and $\neg Y$, then conviction is high, and vice versa. It is an asymmetric measure.

From Table 12.8 we observe that $P(X) = P(XY) + P(X\neg Y)$, which implies that $P(X\neg Y) = P(X) - P(XY)$. Further, $P(\neg Y) = 1 - P(Y)$. We thus have

$$conv(X \longrightarrow Y) = \frac{P(X) \cdot P(\neg Y)}{P(X) - P(XY)} = \frac{P(\neg Y)}{1 - P(XY)/P(X)} = \frac{1 - rsup(Y)}{1 - conf(X \longrightarrow Y)}$$

We conclude that conviction is infinite if confidence is one. If X and Y are independent, then conviction is 1.

Example 12.7. For the rule $A \longrightarrow DE$, we have

$$conv(A \longrightarrow DE) = \frac{1 - rsup(DE)}{1 - conf(A)} = 2.0$$

Table 12.9 shows this and some other rules, along with their conviction, support, confidence, and lift values.

Odds Ratio

The odds ratio utilizes all four entries from the contingency table shown in Table 12.8. Let us divide the dataset into two groups of transactions – those that contain X and those that do not contain X. Define the odds of Y in these two groups as follows:

$$odds(Y|X) = \frac{P(XY)/P(X)}{P(X\neg Y)/P(X)} = \frac{P(XY)}{P(X\neg Y)}$$

$$odds(Y|\neg X) = \frac{P(\neg XY)/P(\neg X)}{P(\neg X\neg Y)/P(\neg X)} = \frac{P(\neg XY)}{P(\neg X\neg Y)}$$

Table 12.8. Contingency table for X and Y

	Y	$\neg Y$	
X	$sup(XY)$	$sup(X\neg Y)$	$sup(X)$
$\neg X$	$sup(\neg XY)$	$sup(\neg X\neg Y)$	$sup(\neg X)$
	$sup(Y)$	$sup(\neg Y)$	$\|\mathbf{D}\|$

Table 12.9. Rule conviction

Rule			*rsup*	*conf*	*lift*	*conv*
A	$\longrightarrow$	DE	0.50	0.75	1.50	2.00
DE	$\longrightarrow$	A	0.50	1.00	1.50	∞
E	$\longrightarrow$	C	0.50	0.60	0.90	0.83
C	$\longrightarrow$	E	0.50	0.75	0.90	0.68

The odds ratio is then defined as the ratio of these two odds:

$$oddsratio(X \longrightarrow Y) = \frac{odds(Y|X)}{odds(Y|\neg X)} = \frac{P(XY) \cdot P(\neg X \neg Y)}{P(X\neg Y) \cdot P(\neg XY)}$$
$$= \frac{sup(XY) \cdot sup(\neg X \neg Y)}{sup(X\neg Y) \cdot sup(\neg XY)}$$

The odds ratio is a symmetric measure, and if X and Y are independent, then it has value 1. Thus, values close to 1 may indicate that there is little dependence between X and Y. Odds ratios greater than 1 imply higher odds of Y occurring in the presence of X as opposed to its complement $\neg X$, whereas odds smaller than one imply higher odds of Y occurring with $\neg X$.

Example 12.8. Let us compare the odds ratio for two rules, $C \longrightarrow A$ and $D \longrightarrow A$, using the example data in Table 12.1. The contingency tables for A and C, and for A and D, are given below:

	C	$\neg C$
A	2	2
$\neg A$	2	0

	D	$\neg D$
A	3	1
$\neg A$	1	1

The odds ratio values for the two rules are given as

$$oddsratio(C \longrightarrow A) = \frac{sup(AC) \cdot sup(\neg A \neg C)}{sup(A\neg C) \cdot sup(\neg AC)} = \frac{2 \times 0}{2 \times 2} = 0$$
$$oddsratio(D \longrightarrow A) = \frac{sup(AD) \cdot sup(\neg A \neg D)}{sup(A\neg D) \cdot sup(\neg AD)} = \frac{3 \times 1}{1 \times 1} = 3$$

Thus, $D \longrightarrow A$ is a stronger rule than $C \longrightarrow A$, which is also indicated by looking at other measures like lift and confidence:

$$conf(C \longrightarrow A) = 2/4 = 0.5 \qquad conf(D \longrightarrow A) = 3/4 = 0.75$$

$$lift(C \longrightarrow A) = \frac{2/6}{4/6 \times 4/6} = 0.75 \qquad lift(D \longrightarrow A) = \frac{3/6}{4/6 \times 4/6} = 1.125$$

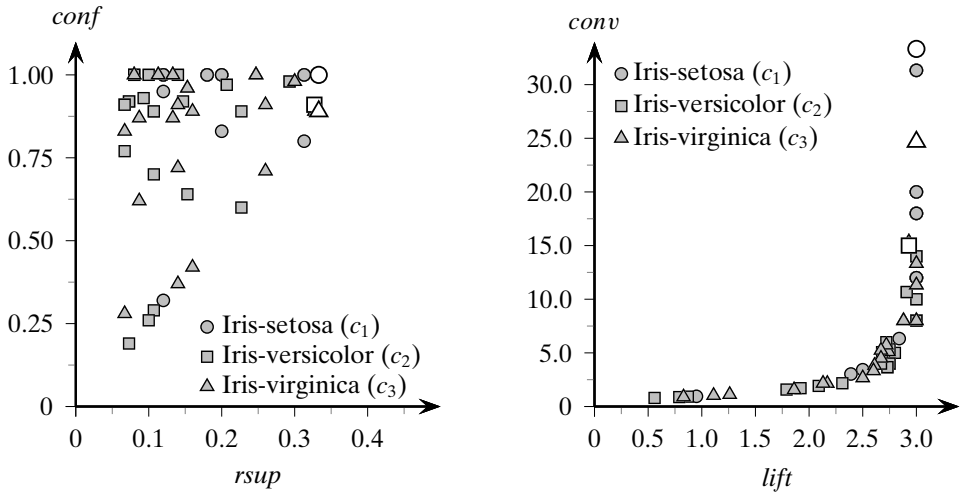
$C \longrightarrow A$ has less confidence and lift than $D \longrightarrow A$.

Example 12.9. We apply the different rule assessment measures on the Iris dataset, which has $n = 150$ examples, over one categorical attribute (`class`), and four numeric attributes (`sepal length`, `sepal width`, `petal length`, and `petal width`). To generate association rules we first discretize the numeric attributes as shown in Table 12.10. In particular, we want to determine representative class-specific rules that characterize each of the three Iris classes: `iris setosa`, `iris virginica` and `iris versicolor`, that is, we generate rules of the form $X \longrightarrow y$, where X is an itemset over the discretized numeric attributes, and y is a single item representing one of the Iris classes.

We start by generating all class-specific association rules using $minsup = 10$ and a minimum lift value of 0.1, which results in a total of 79 rules. Figure 12.1a plots the relative support and confidence of these 79 rules, with the three classes represented by different symbols. To look for the most surprising rules, we also plot in Figure 12.1b the lift and conviction value for the same 79 rules. For each class we select the most specific (i.e., with maximal antecedent) rule with the highest relative support and then confidence, and also those with the highest conviction and then lift. The selected rules are listed in Table 12.11 and Table 12.12, respectively. They are also highlighted in Figure 12.1 (as larger white symbols). Compared to the top rules for support and confidence, we observe that the best rule for c_1 is the same, but the rules for c_2 and c_3 are not the same, suggesting a trade-off between support and novelty among these rules.

Table 12.10. Iris dataset discretization and labels employed

Attribute	Range or value	Label
Sepal length	4.30–5.55	sl_1
	5.55–6.15	sl_2
	6.15–7.90	sl_3
Sepal width	2.00–2.95	sw_1
	2.95–3.35	sw_2
	3.35–4.40	sw_3
Petal length	1.00–2.45	pl_1
	2.45–4.75	pl_2
	4.75–6.90	pl_3
Petal width	0.10–0.80	pw_1
	0.80–1.75	pw_2
	1.75–2.50	pw_3
Class	Iris-setosa	c_1
	Iris-versicolor	c_2
	Iris-virginica	c_3



(a) Support vs. confidence (b) Lift vs. conviction

Figure 12.1. Iris: support vs. confidence, and conviction vs. lift for class-specific rules. The best rule for each class is shown in white.

Table 12.11. Iris: best class-specific rules according to support and confidence

Rule	*rsup*	*conf*	*lift*	*conv*
$\{pl_1, pw_1\} \longrightarrow c_1$	0.333	1.00	3.00	33.33
$pw_2 \longrightarrow c_2$	0.327	0.91	2.72	6.00
$pl_3 \longrightarrow c_3$	0.327	0.89	2.67	5.24

Table 12.12. Iris: best class-specific rules according to lift and conviction

Rule	*rsup*	*conf*	*lift*	*conv*
$\{pl_1, pw_1\} \longrightarrow c_1$	0.33	1.00	3.00	33.33
$\{pl_2, pw_2\} \longrightarrow c_2$	0.29	0.98	2.93	15.00
$\{sl_3, pl_3, pw_3\} \longrightarrow c_3$	0.25	1.00	3.00	24.67

12.1.2 Pattern Assessment Measures

We now turn our focus on measures for pattern assessment.

Support

The most basic measures are support and relative support, giving the number and fraction of transactions in $\mathbf{D}$ that contain the itemset X:

$$sup(X) = |\mathbf{t}(X)| \qquad rsup(X) = \frac{sup(X)}{|\mathbf{D}|}$$

Lift

The *lift* of a k-itemset $X = \{x_1, x_2, \ldots, x_k\}$ in dataset $\mathbf{D}$ is defined as

$$lift(X, \mathbf{D}) = \frac{P(X)}{\prod_{i=1}^{k} P(x_i)} = \frac{rsup(X)}{\prod_{i=1}^{k} rsup(x_i)} \tag{12.2}$$

that is, the ratio of the observed joint probability of items in X to the expected joint probability if all the items $x_i \in X$ were independent.

We may further generalize the notion of lift of an itemset X by considering all the different ways of partitioning it into nonempty and disjoint subsets. For instance, assume that the set $\{X_1, X_2, \ldots, X_q\}$ is a q-partition of X, i.e., a partitioning of X into q nonempty and disjoint itemsets X_i, such that $X_i \cap X_j = \emptyset$ and $\cup_i X_i = X$. Define the generalized lift of X over partitions of size q as follows:

$$lift_q(X) = \min_{X_1, \ldots, X_q} \left\{ \frac{P(X)}{\prod_{i=1}^{q} P(X_i)} \right\}$$

This is, the least value of lift over all q-partitions X. Viewed in this light, $lift(X) = lift_k(X)$, that is, lift is the value obtained from the unique k-partition of X.

Rule-based Measures

Given an itemset X, we can evaluate it using rule assessment measures by considering all possible rules that can be generated from X. Let Θ be some rule assessment measure. We generate all possible rules from X of the form $X_1 \longrightarrow X_2$ and $X_2 \longrightarrow X_1$, where the set $\{X_1, X_2\}$ is a 2-partition, or a bipartition, of X. We then compute the measure Θ for each such rule, and use summary statistics such as the mean, maximum, and minimum to characterize X. If Θ is a symmetric measure, then $\Theta(X_1 \longrightarrow X_2) = \Theta(X_2 \longrightarrow X_1)$, and we have to consider only half of the rules. For example, if Θ is rule lift, then we can define the average, maximum, and minimum lift values for X as follows:

$$AvgLift(X) = \operatorname*{avg}_{X_1, X_2} \Big\{ lift(X_1 \longrightarrow X_2) \Big\}$$

$$MaxLift(X) = \max_{X_1, X_2} \Big\{ lift(X_1 \longrightarrow X_2) \Big\}$$

$$MinLift(X) = \min_{X_1, X_2} \Big\{ lift(X_1 \longrightarrow X_2) \Big\}$$

We can also do the same for other rule measures such as leverage, confidence, and so on. In particular, when we use rule lift, then $MinLift(X)$ is identical to the generalized lift $lift_2(X)$ over all 2-partitions of X.

Example 12.10. Consider the itemset $X = \{pl_2, pw_2, c_2\}$, whose support in the discretized Iris dataset is shown in Table 12.13, along with the supports for all of its subsets. Note that the size of the database is $|\mathbf{D}| = n = 150$.

Using Eq. (12.2), the lift of X is given as

$$lift(X) = \frac{rsup(X)}{rsup(pl_2) \cdot rsup(pw_2) \cdot rsup(c_2)} = \frac{0.293}{0.3 \cdot 0.36 \cdot 0.333} = 8.16$$

Table 12.13. Support values for $\{pl_2, pw_2, c_2\}$ and its subsets

Itemset	*sup*	*rsup*
$\{pl_2, pw_2, c_2\}$	44	0.293
$\{pl_2, pw_2\}$	45	0.300
$\{pl_2, c_2\}$	44	0.293
$\{pw_2, c_2\}$	49	0.327
$\{pl_2\}$	45	0.300
$\{pw_2\}$	54	0.360
$\{c_2\}$	50	0.333

Table 12.14. Rules generated from itemset $\{pl_2, pw_2, c_2\}$

Bipartition	Rule	*lift*	*leverage*	*conf*
$\{\{pl_2\}, \{pw_2, c_2\}\}$	$pl_2 \longrightarrow \{pw_2, c_2\}$	2.993	0.195	0.978
	$\{pw_2, c_2\} \longrightarrow pl_2$	2.993	0.195	0.898
$\{\{pw_2\}, \{pl_2, c_2\}\}$	$pw_2 \longrightarrow \{pl_2, c_2\}$	2.778	0.188	0.815
	$\{pl_2, c_2\} \longrightarrow pw_2$	2.778	0.188	1.000
$\{\{c_2\}, \{pl_2, pw_2\}\}$	$c_2 \longrightarrow \{pl_2, pw_2\}$	2.933	0.193	0.880
	$\{pl_2, pw_2\} \longrightarrow c_2$	2.933	0.193	0.978

Table 12.14 shows all the possible rules that can be generated from X, along with the rule lift and leverage values. Note that because both of these measures are symmetric, we need to consider only the distinct bipartitions of which there are three, as shown in the table. The maximum, minimum, and average lift values are as follows:

$$MaxLift(X) = \max\{2.993, 2.778, 2.933\} = 2.998$$
$$MinLift(X) = \min\{2.993, 2.778, 2.933\} = 2.778$$
$$AvgLift(X) = \text{avg}\{2.993, 2.778, 2.933\} = 2.901$$

We may use other measures too. For example, the average leverage of X is given as

$$AvgLeverage(X) = \text{avg}\{0.195, 0.188, 0.193\} = 0.192$$

However, because confidence is not a symmetric measure, we have to consider all the six rules and their confidence values, as shown in Table 12.14. The average confidence for X is

$$AvgConf(X) = \text{avg}\{0.978, 0.898, 0.815, 1.0, 0.88, 0.978\} = 5.549/6 = 0.925$$

Example 12.11. Consider all frequent itemsets in the discretized Iris dataset from Example 12.9, using $minsup = 1$. We analyze the set of all possible rules that can be generated from these frequent itemsets. Figure 12.2 plots the relative support and average lift values for all the 306 frequent patterns with size at least 2 (since nontrivial

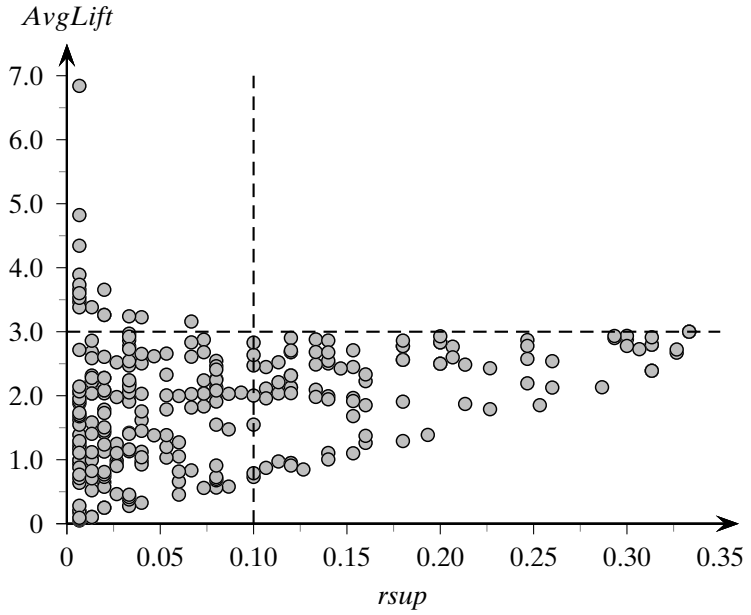


Figure 12.2. Iris: support and average lift of patterns assessed.

rules can only be generated from itemsets of size 2 or more). We can see that with the exception of low support itemsets, the average lift value is bounded above by 3.0. From among these we may select those patterns with the highest support for further analysis. For instance, the itemset $X = \{pl_1, pw_1, c_1\}$ is a maximal itemset with support $rsup(X) = 0.33$, all of whose subsets also have support $rsup = 0.33$. Thus, all of the rules that can be derived from it have a lift of 3.0, and the minimum lift of X is 3.0.

12.1.3 Comparing Multiple Rules and Patterns

We now turn our attention to comparing different rules and patterns. In general, the number of frequent itemsets and association rules can be very large and many of them may not be very relevant. We highlight cases when certain patterns and rules can be pruned, as the information contained in them may be subsumed by other more relevant ones.

Comparing Itemsets

When comparing multiple itemsets we may choose to focus on the maximal itemsets that satisfy some property, or we may consider closed itemsets that capture all of the support information. We consider these and other measures in the following paragraphs.

Maximal Itemsets An frequent itemset X is *maximal* if all of its supersets are not frequent, that is, X is maximal iff

$$sup(X) \geq minsup, \text{ and for all } Y \supset X, sup(Y) < minsup$$

Table 12.15. Iris: maximal patterns according to average lift

Pattern	Avg. lift
$\{sl_1, sw_2, pl_1, pw_1, c_1\}$	2.90
$\{sl_1, sw_3, pl_1, pw_1, c_1\}$	2.86
$\{sl_2, sw_1, pl_2, pw_2, c_2\}$	2.83
$\{sl_3, sw_2, pl_3, pw_3, c_3\}$	2.88
$\{sw_1, pl_3, pw_3, c_3\}$	2.52

Given a collection of frequent itemsets, we may choose to retain only the maximal ones, especially among those that already satisfy some other constraints on pattern assessment measures like lift or leverage.

Example 12.12. Consider the discretized Iris dataset from Example 12.9. To gain insights into the maximal itemsets that pertain to each of the Iris classes, we focus our attention on the class-specific itemsets, that is, those itemsets X that contain a class as one of the items. From the itemsets plotted in Figure 12.2, using $minsup(X) \geq 15$ (which corresponds to a relative support of 10%) and retaining only those itemsets with an average lift value of at least 2.5, we retain 37 class-specific itemsets. Among these, the maximal class-specific itemsets are shown in Table 12.15, which highlight the features that characterize each of the three classes. For instance, for class c_1 (`Iris-setosa`), the essential items are sl_1, pl_1, pw_1 and either sw_2 or sw_3. Looking at the range values in Table 12.10, we conclude that `Iris-setosa` class is characterized by `sepal-length` in the range $sl_1 = [4.30, 5.55]$, `petal-length` in the range $pl_1 = [1, 2.45]$, and so on. A similar interpretation can be carried out for the other two Iris classes.

Closed Itemsets and Minimal Generators An itemset X is *closed* if all of its supersets have strictly less support, that is,

$$sup(X) > sup(Y), \text{ for all } Y \supset X$$

An itemset X is a *minimal generator* if all its subsets have strictly higher support, that is,

$$sup(X) < sup(Y), \text{ for all } Y \subset X$$

If an itemset X is not a minimal generator, then it implies that it has some redundant items, that is, we can find some subset $Y \subset X$, which can be replaced with an even smaller subset $W \subset Y$ without changing the support of X, that is, there exists a $W \subset Y$, such that

$$sup(X) = sup(Y \cup (X \setminus Y)) = sup(W \cup (X \setminus Y))$$

One can show that all subsets of a minimal generator must themselves be minimal generators.

Table 12.16. Closed itemsets and minimal generators

sup	Closed Itemset	Minimal Generators
3	$ABDE$	AD, DE
3	BCE	CE
4	ABE	A
4	BC	C
4	BD	D
5	BE	E
6	B	B

Example 12.13. Consider the dataset in Table 12.1 and the set of frequent itemsets with $minsup = 3$ as shown in Table 12.2. There are only two maximal frequent itemsets, namely $ABDE$ and BCE, which capture essential information about whether another itemset is frequent or not: an itemset is frequent only if it is a subset of one of these two.

Table 12.16 shows the seven closed itemsets and the corresponding minimal generators. Both of these sets allow one to infer the exact support of any other frequent itemset. The support of an itemset X is the maximum support among all closed itemsets that contain it. Alternatively, the support of X is the minimum support among all minimal generators that are subsets of X. For example, the itemset AE is a subset of the closed sets ABE and $ABDE$, and it is a superset of the minimal generators A, and E; we can observe that

$$sup(AE) = \max\{sup(ABE), sup(ABDE)\} = 4$$

$$sup(AE) = \min\{sup(A), sup(E)\} = 4$$

Productive Itemsets An itemset X is *productive* if its relative support is higher than the expected relative support over all of its bipartitions, assuming they are independent. More formally, let $|X| \geq 2$, and let $\{X_1, X_2\}$ be a bipartition of X. We say that X is productive provided

$$rsup(X) > rsup(X_1) \times rsup(X_2), \text{ for all bipartitions } \{X_1, X_2\} \text{ of } X \tag{12.3}$$

This immediately implies that X is productive if its minimum lift is greater than one, as

$$MinLift(X) = \min_{X_1, X_2} \left\{ \frac{rsup(X)}{rsup(X_1) \cdot rsup(X_2)} \right\} > 1$$

In terms of leverage, X is productive if its minimum leverage is above zero because

$$MinLeverage(X) = \min_{X_1, X_2} \Big\{ rsup(X) - rsup(X_1) \times rsup(X_2) \Big\} > 0$$

Example 12.14. Considering the frequent itemsets in Table 12.2, the set $ABDE$ is not productive because there exists a bipartition with lift value of 1. For instance, for its bipartition $\{B, ADE\}$ we have

$$lift(B \longrightarrow ADE) = \frac{rsup(ABDE)}{rsup(B) \cdot rsup(ADE)} = \frac{3/6}{6/6 \cdot 3/6} = 1$$

On the other hand, ADE is productive because it has three distinct bipartitions and all of them have lift above 1:

$$lift(A \longrightarrow DE) = \frac{rsup(ADE)}{rsup(A) \cdot rsup(DE)} = \frac{3/6}{4/6 \cdot 3/6} = 1.5$$

$$lift(D \longrightarrow AE) = \frac{rsup(ADE)}{rsup(D) \cdot rsup(AE)} = \frac{3/6}{4/6 \cdot 4/6} = 1.125$$

$$lift(E \longrightarrow AD) = \frac{rsup(ADE)}{rsup(E) \cdot rsup(AD)} = \frac{3/6}{5/6 \cdot 3/6} = 1.2$$

Comparing Rules

Given two rules $R : X \longrightarrow Y$ and $R' : W \longrightarrow Y$ that have the same consequent, we say that R is *more specific* than R', or equivalently, that R' is *more general* than R provided $W \subset X$.

Nonredundant Rules We say that a rule $R : X \longrightarrow Y$ is *redundant* provided there exists a more general rule $R' : W \longrightarrow Y$ that has the same support, that is, $W \subset X$ and $sup(R) = sup(R')$. On the other hand, if $sup(R) < sup(R')$ over all its generalizations R', then R is *nonredundant*.

Improvement and Productive Rules Define the *improvement* of a rule $X \longrightarrow Y$ as follows:

$$imp(X \longrightarrow Y) = conf(X \longrightarrow Y) - \max_{W \subset X} \Big\{ conf(W \longrightarrow Y) \Big\}$$

Improvement quantifies the minimum difference between the confidence of a rule and any of its generalizations. A rule $R : X \longrightarrow Y$ is *productive* if its improvement is greater than zero, which implies that for all more general rules $R' : W \longrightarrow Y$ we have $conf(R) > conf(R')$. On the other hand, if there exists a more general rule R' with $conf(R') \geq conf(R)$, then R is *unproductive*. If a rule is redundant, it is also unproductive because its improvement is zero.

The smaller the improvement of a rule $R : X \longrightarrow Y$, the more likely it is to be unproductive. We can generalize this notion to consider rules that have at least some minimum level of improvement, that is, we may require that $imp(X \longrightarrow Y) \geq t$, where t is a user-specified minimum improvement threshold.

Example 12.15. Consider the example dataset in Table 12.1, and the set of frequent itemsets in Table 12.2. Consider rule $R : BE \longrightarrow C$, which has support 3, and confidence $3/5 = 0.60$. It has two generalizations, namely

$$R'_1 : E \longrightarrow C, \quad sup = 3, conf = 3/5 = 0.6$$
$$R'_2 : B \longrightarrow C, \quad sup = 4, conf = 4/6 = 0.67$$

Thus, $BE \longrightarrow C$ is redundant w.r.t. $E \longrightarrow C$ because they have the same support, that is, $sup(BCE) = sup(BC)$. Further, $BE \longrightarrow C$ is also unproductive, since $imp(BE \longrightarrow C) = 0.6 - \max\{0.6, 0.67\} = -0.07$; it has a more general rule, namely R'_2, with higher confidence.

12.2 SIGNIFICANCE TESTING AND CONFIDENCE INTERVALS

We now consider how to assess the statistical significance of patterns and rules, and how to derive confidence intervals for a given assessment measure.

12.2.1 Fisher Exact Test for Productive Rules

We begin by discussing the Fisher exact test for rule improvement. That is, we directly test whether the rule $R : X \longrightarrow Y$ is productive by comparing its confidence with that of each of its generalizations $R' : W \longrightarrow Y$, including the default or trivial rule $\emptyset \longrightarrow Y$.

Let $R : X \longrightarrow Y$ be an association rule. Consider its generalization $R' : W \longrightarrow Y$, where $W = X \setminus Z$ is the new antecedent formed by removing from X the subset $Z \subseteq X$. Given an input dataset $\mathbf{D}$, conditional on the fact that W occurs, we can create a 2×2 contingency table between Z and the consequent Y as shown in Table 12.17. The different cell values are as follows:

$$a = sup(WZY) = sup(XY) \qquad b = sup(WZ\neg Y) = sup(X\neg Y)$$
$$c = sup(W\neg ZY) \qquad d = sup(W\neg Z\neg Y)$$

Here, a denotes the number of transactions that contain both X and Y, b denotes the number of transactions that contain X but not Y, c denotes the number of transactions that contain W and Y but not Z, and finally d denotes the number of transactions that contain W but neither Z nor Y. The marginal counts are given as

$$\text{row marginals: } a + b = sup(WZ) = sup(X), \quad c + d = sup(W\neg Z)$$
$$\text{column marginals: } a + c = sup(WY), \quad b + d = sup(W\neg Y)$$

where the row marginals give the occurrence frequency of W with and without Z, and the column marginals specify the occurrence counts of W with and without Y. Finally, we can observe that the sum of all the cells is simply $n = a + b + c + d = sup(W)$. Notice that when $Z = X$, we have $W = \emptyset$, and the contingency table defaults to the one shown in Table 12.8.

Given a contingency table conditional on W, we are interested in the odds ratio obtained by comparing the presence and absence of Z, that is,

$$oddsratio = \frac{a/(a+b)}{b/(a+b)} \Big/ \frac{c/(c+d)}{d/(c+d)} = \frac{ad}{bc} \tag{12.4}$$

Table 12.17. Contingency table for Z and Y, conditional on $W = X \setminus Z$

W	Y	$\neg Y$	
Z	a	b	$a+b$
$\neg Z$	c	d	$c+d$
	$a+c$	$b+d$	$n = sup(W)$

Recall that the odds ratio measures the odds of X, that is, W and Z, occurring with Y versus the odds of its subset W, but not Z, occurring with Y. Under the null hypothesis H_0 that Z and Y are independent given W the odds ratio is 1. To see this, note that under the independence assumption the count in a cell of the contingency table is equal to the product of the corresponding row and column marginal counts divided by n, that is, under H_0:

$$a = (a+b)(a+c)/n \qquad b = (a+b)(b+d)/n$$
$$c = (c+d)(a+c)/n \qquad d = (c+d)(b+d)/n$$

Plugging these values in Eq. (12.4), we obtain

$$oddsratio = \frac{ad}{bc} = \frac{(a+b)(c+d)(b+d)(a+c)}{(a+c)(b+d)(a+b)(c+d)} = 1$$

The null hypothesis therefore corresponds to $H_0 : oddsratio = 1$, and the alternative hypothesis is $H_a : oddsratio > 1$. Under the null hypothesis, if we further assume that the row and column marginals are fixed, then a uniquely determines the other three values b, c, and d, and the probability mass function of observing the value a in the contingency table is given by the hypergeometric distribution. Recall that the hypergeometric distribution gives the probability of choosing s successes in t trails if we sample *without replacement* from a finite population of size T that has S successes in total, given as

$$P(s \mid t, S, T) = \binom{S}{s} \cdot \binom{T-S}{t-s} \Big/ \binom{T}{t}$$

In our context, we take the occurrence of Z as a success. The population size is $T = sup(W) = n$ because we assume that W always occurs, and the total number of successes is the support of Z given W, that is, $S = a+b$. In $t = a+c$ trials, the hypergeometric distribution gives the probability of $s = a$ successes:

$$\begin{aligned} P\Big(a \,\Big|\, (a+c), (a+b), n\Big) &= \frac{\binom{a+b}{a} \cdot \binom{n-(a+b)}{(a+c)-a}}{\binom{n}{a+c}} = \frac{\binom{a+b}{a} \cdot \binom{c+d}{c}}{\binom{n}{a+c}} \\ &= \frac{(a+b)!\,(c+d)!}{a!\,b!\,c!\,d!} \Big/ \frac{n!}{(a+c)!\,(n-(a+c))!} \\ &= \frac{(a+b)!\,(c+d)!\,(a+c)!\,(b+d)!}{n!\,a!\,b!\,c!\,d!} \end{aligned} \tag{12.5}$$

Table 12.18. Contingency table: increase a by i

W	Y	$\neg Y$	
Z	$a+i$	$b-i$	$a+b$
$\neg Z$	$c-i$	$d+i$	$c+d$
	$a+c$	$b+d$	$n = sup(W)$

Our aim is to contrast the null hypothesis H_0 that $oddsratio = 1$ with the alternative hypothesis H_a that $oddsratio > 1$. Because a determines the rest of the cells under fixed row and column marginals, we can see from Eq. (12.4) that the larger the a the larger the odds ratio, and consequently the greater the evidence for H_a. We can obtain the *p-value* for a contingency table as extreme as that in Table 12.17 by summing Eq. (12.5) over all possible values a or larger:

$$p\text{-}value(a) = \sum_{i=0}^{\min(b,c)} P(a+i \mid (a+c), (a+b), n)$$

$$= \sum_{i=0}^{\min(b,c)} \frac{(a+b)!\,(c+d)!\,(a+c)!\,(b+d)!}{n!\,(a+i)!\,(b-i)!\,(c-i)!\,(d+i)!}$$

which follows from the fact that when we increase the count of a by i, then because the row and column marginals are fixed, b and c must decrease by i, and d must increase by i, as shown in Table 12.18. The lower the *p-value* the stronger the evidence that the odds ratio is greater than one, and thus, we may reject the null hypothesis H_0 if $p\text{-}value \leq \alpha$, where α is the significance threshold (e.g., $\alpha = 0.01$). This test is known as the *Fisher Exact Test*.

In summary, to check whether a rule $R: X \longrightarrow Y$ is productive, we must compute $p\text{-}value(a) = p\text{-}value(sup(XY))$ of the contingency tables obtained from each of its generalizations $R': W \longrightarrow Y$, where $W = X \setminus Z$, for $Z \subseteq X$. If $p\text{-}value(sup(XY)) > \alpha$ for any of these comparisons, then we can reject the rule $R: X \longrightarrow Y$ as nonproductive. On the other hand, if $p\text{-}value(sup(XY)) \leq \alpha$ for all the generalizations, then R is productive. However, note that if $|X| = k$, then there are $2^k - 1$ possible generalizations; to avoid this exponential complexity for large antecedents, we typically restrict our attention to only the immediate generalizations of the form $R': X \setminus z \longrightarrow Y$, where $z \in X$ is one of the attribute values in the antecedent. However, we do include the trivial rule $\emptyset \longrightarrow Y$ because the conditional probability $P(Y|X) = conf(X \longrightarrow Y)$ should also be higher than the prior probability $P(Y) = conf(\emptyset \longrightarrow Y)$.

Example 12.16. Consider the rule $R: pw_2 \longrightarrow c_2$ obtained from the discretized Iris dataset. To test if it is productive, because there is only a single item in the antecedent, we compare it only with the default rule $\emptyset \longrightarrow c_2$. Using Table 12.17, the various cell values are

$$a = sup(pw_2, c_2) = 49 \qquad b = sup(pw_2, \neg c_2) = 5$$

$$c = sup(\neg pw_2, c_2) = 1 \qquad d = sup(\neg pw_2, \neg c_2) = 95$$

with the contingency table given as

	c_2	$\neg c_2$	
pw_2	49	5	54
$\neg pw_2$	1	95	96
	50	100	150

Thus the *p-value* is given as

$$\begin{aligned} p\text{-}value &= \sum_{i=0}^{\min(b,c)} P(a+i \mid (a+c),(a+b),n) \\ &= P(49 \mid 50,54,150) + P(50 \mid 50,54,150) \\ &= \binom{54}{49}\cdot\binom{96}{95}\Big/\binom{150}{50} + \binom{54}{50}\cdot\binom{96}{96}\Big/\binom{150}{50} \\ &= 1.51\times 10^{-32} + 1.57\times 10^{-35} = 1.51\times 10^{-32} \end{aligned}$$

Since the *p-value* is extremely small, we can safely reject the null hypothesis that the odds ratio is 1. Instead, there is a strong relationship between $X = pw_2$ and $Y = c_2$, and we conclude that $R : pw_2 \longrightarrow c_2$ is a productive rule.

Example 12.17. Consider another rule $\{sw_1, pw_2\} \longrightarrow c_2$, with $X = \{sw_1, pw_2\}$ and $Y = c_2$. Consider its three generalizations, and the corresponding contingency tables and *p-values*:

$R'_1 : pw_2 \longrightarrow c_2$

$Z = \{sw_1\}$
$W = X \setminus Z = \{pw_2\}$
$p\text{-}value = 0.84$

$W = pw_2$	c_2	$\neg c_2$	
sw_1	34	4	38
$\neg sw_1$	15	1	16
	49	5	54

$R'_2 : sw_1 \longrightarrow c_2$

$Z = \{pw_2\}$
$W = X \setminus Z = \{sw_1\}$
$p\text{-}value = 1.39\times 10^{-11}$

$W = sw_1$	c_2	$\neg c_2$	
pw_2	34	4	38
$\neg pw_2$	0	19	19
	34	23	57

$R'_3 : \emptyset \longrightarrow c_2$

$Z = \{sw_1, pw_2\}$
$W = X \setminus Z = \emptyset$
$p\text{-}value = 3.55\times 10^{-17}$

$W = \emptyset$	c_2	$\neg c_2$	
$\{sw_1, pw_2\}$	34	4	38
$\neg\{sw_1, pw_2\}$	16	96	112
	50	100	150

We can see that whereas the *p-value* with respect to R'_2 and R'_3 is small, for R'_1 we have $p\text{-}value = 0.84$, which is too high and thus we cannot reject the null hypothesis. We conclude that $R : \{sw_1, pw_2\} \longrightarrow c_2$ is not productive. In fact, its generalization R'_1 is the one that is productive, as shown in Example 12.16.

Multiple Hypothesis Testing

Given an input dataset **D**, there can be an exponentially large number of rules that need to be tested to check whether they are productive or not. We thus run into the multiple hypothesis testing problem, that is, just by the sheer number of hypothesis tests some unproductive rules will pass the $p\text{-}value \leq \alpha$ threshold by random chance. A strategy for overcoming this problem is to use the *Bonferroni correction* of the significance level that explicitly takes into account the number of experiments performed during the hypothesis testing process. Instead of using the given α threshold, we should use an adjusted threshold $\alpha' = \frac{\alpha}{\#r}$, where $\#r$ is the number of rules to be tested or its estimate. This correction ensures that the rule false discovery rate is bounded by α, where a false discovery is to claim that a rule is productive when it is not.

Example 12.18. Consider the discretized Iris dataset, using the discretization shown in Table 12.10. Let us focus only on class-specific rules, that is, rules of the form $X \rightarrow c_i$. Since each example can take on only one value at a time for a given attribute, the maximum antecedent length is four, and the maximum number of class-specific rules that can be generated from the Iris dataset is given as

$$\#r = c \times \left(\sum_{i=1}^{4} \binom{4}{i} b^i \right)$$

where c is the number of Iris classes, and b is the maximum number of bins for any other attribute. The summation is over the antecedent size i, that is, the number of attributes to be used in the antecedent. Finally, there are b^i possible combinations for the chosen set of i attributes. Because there are three Iris classes, and because each attribute has three bins, we have $c = 3$ and $b = 3$, and the number of possible rules is

$$\#r = 3 \times \left(\sum_{i=1}^{4} \binom{4}{i} 3^i \right) = 3(12 + 54 + 108 + 81) = 3 \cdot 255 = 765$$

Thus, if the input significance level is $\alpha = 0.01$, then the adjusted significance level using the Bonferroni correction is $\alpha' = \alpha / \#r = 0.01/765 = 1.31 \times 10^{-5}$. The rule $pw_2 \longrightarrow c_2$ in Example 12.16 has $p\text{-}value = 1.51 \times 10^{-32}$, and thus it remains productive even when we use α'.

12.2.2 Permutation Test for Significance

A *permutation* or *randomization* test determines the distribution of a given test statistic Θ by randomly modifying the observed data several times to obtain a random sample

of datasets, which can in turn be used for significance testing. In the context of pattern assessment, given an input dataset $\mathbf{D}$, we first generate k randomly permuted datasets $\mathbf{D}_1, \mathbf{D}_2, \ldots, \mathbf{D}_k$. We can then perform different types of significance tests. For instance, given a pattern or rule we can check whether it is statistically significant by first computing the empirical probability mass function (EPMF) for the test statistic Θ by computing its value θ_i in the ith randomized dataset $\mathbf{D}_i$ for all $i \in [1, k]$. From these values we can generate the empirical cumulative distribution function

$$\hat{F}(x) = \hat{P}(\Theta \leq x) = \frac{1}{k} \sum_{i=1}^{k} I(\theta_i \leq x)$$

where I is an indicator variable that takes on the value 1 when its argument is true, and is 0 otherwise. Let θ be the value of the test statistic in the input dataset $\mathbf{D}$, then $p\text{-}value(\theta)$, that is, the probability of obtaining a value as high as θ by random chance can be computed as

$$p\text{-}value(\theta) = 1 - F(\theta)$$

Given a significance level α, if $p\text{-}value(\theta) > \alpha$, then we accept the null hypothesis that the pattern/rule is not statistically significant. On the other hand, if $p\text{-}value(\theta) \leq \alpha$, then we can reject the null hypothesis and conclude that the pattern is significant because a value as high as θ is highly improbable. The permutation test approach can also be used to assess an entire set of rules or patterns. For instance, we may test a collection of frequent itemsets by comparing the number of frequent itemsets in $\mathbf{D}$ with the distribution of the number of frequent itemsets empirically derived from the permuted datasets $\mathbf{D}_i$. We may also do this analysis as a function of *minsup*, and so on.

Swap Randomization

A key question in generating the permuted datasets $\mathbf{D}_i$ is which characteristics of the input dataset $\mathbf{D}$ we should preserve. The *swap randomization* approach maintains as invariant the column and row margins for a given dataset, that is, the permuted datasets preserve the support of each item (the column margin) as well as the number of items in each transaction (the row margin). Given a dataset $\mathbf{D}$, we randomly create k datasets that have the same row and column margins. We then mine frequent patterns in $\mathbf{D}$ and check whether the pattern statistics are different from those obtained using the randomized datasets. If the differences are not significant, we may conclude that the patterns arise solely from the row and column margins, and not from any interesting properties of the data.

Given a binary matrix $\mathbf{D} \subseteq \mathcal{T} \times \mathcal{I}$, the swap randomization method exchanges two nonzero cells of the matrix via a *swap* that leaves the row and column margins unchanged. To illustrate how swap works, consider any two transactions $t_a, t_b \in \mathcal{T}$ and any two items $i_a, i_b \in \mathcal{I}$ such that $(t_a, i_a), (t_b, i_b) \in \mathbf{D}$ and $(t_a, i_b), (t_b, i_a) \notin \mathbf{D}$, which corresponds to the 2×2 submatrix in $\mathbf{D}$, given as

$$\mathbf{D}(t_a, i_a; t_b, i_b) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

ALGORITHM 12.1. Generate Swap Randomized Dataset

```
SWAPRANDOMIZATION(t, D ⊆ T × I):
1 while t > 0 do
2     Select pairs (t_a, i_a), (t_b, i_b) ∈ D randomly
3     if (t_a, i_b) ∉ D and (t_b, i_a) ∉ D then
4         D ← D \ {(t_a, i_a), (t_b, i_b)} ∪ {(t_a, i_b), (t_b, i_a)}
5     t = t − 1
6 return D
```

After a swap operation we obtain the new submatrix

$$\mathbf{D}(t_a, i_b; t_b, i_a) = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

where we exchange the elements in $\mathbf{D}$ so that $(t_a, i_b), (t_b, i_a) \in \mathbf{D}$, and $(t_a, i_a), (t_b, i_b) \notin \mathbf{D}$. We denote this operation as $Swap(t_a, i_a; t_b, i_b)$. Notice that a swap does not affect the row and column margins, and we can thus generate a permuted dataset with the same row and column sums as $\mathbf{D}$ through a sequence of swaps. Algorithm 12.1 shows the pseudo-code for generating a swap randomized dataset. The algorithm performs t swap trials by selecting two pairs $(t_a, i_a),\ (t_b, i_b) \in \mathbf{D}$ at random; a swap is successful only if both $(t_a, i_b),\ (t_b, i_a) \notin \mathbf{D}$.

Example 12.19. Consider the input binary dataset $\mathbf{D}$ shown in Table 12.19a, whose row and column sums are also shown. Table 12.19b shows the resulting dataset after a single swap operation $Swap(1, D; 4, C)$, highlighted by the gray cells. When we apply another swap, namely $Swap(2, C; 4, A)$, we obtain the data in Table 12.19c. We can observe that the marginal counts remain invariant.

From the input dataset $\mathbf{D}$ in Table 12.19a we generated $k = 100$ swap randomized datasets, each of which is obtained by performing 150 swaps (the product of all possible transaction pairs and item pairs, that is, $\binom{6}{2} \cdot \binom{5}{2} = 150$). Let the test statistic be the total number of frequent itemsets using $minsup = 3$. Mining $\mathbf{D}$ results in $|\mathcal{F}| = 19$ frequent itemsets. Likewise, mining each of the $k = 100$ permuted datasets results in the following empirical PMF for $|\mathcal{F}|$:

$$P\big(|\mathcal{F}| = 19\big) = 0.67 \qquad\qquad P\big(|\mathcal{F}| = 17\big) = 0.33$$

Because $p\text{-}value(19) = 0.67$, we may conclude that the set of frequent itemsets is essentially determined by the row and column marginals.

Focusing on a specific itemset, consider $ABDE$, which is one of the maximal frequent itemsets in $\mathbf{D}$, with $sup(ABDE) = 3$. The probability that $ABDE$ is frequent is $17/100 = 0.17$ because it is frequent in 17 of the 100 swapped datasets. As this probability is not very low, we may conclude that $ABDE$ is not a statistically significant pattern; it has a relatively high chance of being frequent in random datasets. Consider another itemset BCD that is not frequent in $\mathbf{D}$ because

$sup(BCD) = 2$. The empirical PMF for the support of BCD is given as

$$P(sup = 2) = 0.54 \qquad P(sup = 3) = 0.44 \qquad P(sup = 4) = 0.02$$

In a majority of the datasets BCD is infrequent, and if $minsup = 4$, then $p\text{-}value(sup = 4) = 0.02$ implies that BCD is highly unlikely to be a frequent pattern.

Table 12.19. Input data **D** and swap randomization

Tid	Items					Sum
	A	*B*	*C*	*D*	*E*	
1	1	1	0	1	1	4
2	0	1	1	0	1	3
3	1	1	0	1	1	4
4	1	1	1	0	1	4
5	1	1	1	1	1	5
6	0	1	1	1	0	3
Sum	4	6	4	4	5	

(a) Input binary data **D**

Tid	Items					Sum
	A	*B*	*C*	*D*	*E*	
1	1	1	1	0	1	4
2	0	1	1	0	1	3
3	1	1	0	1	1	4
4	1	1	0	1	1	4
5	1	1	1	1	1	5
6	0	1	1	1	0	3
Sum	4	6	4	4	5	

(b) $Swap(1, D; 4, C)$

Tid	Items					Sum
	A	*B*	*C*	*D*	*E*	
1	1	1	1	0	1	4
2	1	1	0	0	1	3
3	1	1	0	1	1	4
4	0	1	1	1	1	4
5	1	1	1	1	1	5
6	0	1	1	1	0	3
Sum	4	6	4	4	5	

(c) $Swap(2, C; 4, A)$

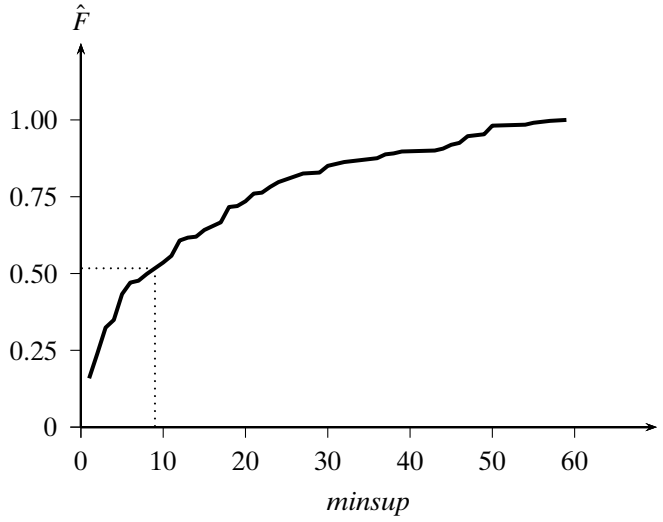


Figure 12.3. Cumulative distribution of the number of frequent itemsets as a function of minimum support.

Example 12.20. We apply the swap randomization approach to the discretized Iris dataset. Figure 12.3 shows the cumulative distribution of the number of frequent itemsets in **D** at various minimum support levels. We choose $minsup = 10$, for which

we have $\hat{F}(10) = P(sup < 10) = 0.517$. Put differently, $P(sup \geq 10) = 1 - 0.517 = 0.483$, that is, 48.3% of the itemsets that occur at least once are frequent using $minsup = 10$.

Define the test statistic to be the *relative lift*, defined as the relative change in the lift value of itemset X when comparing the input dataset $\mathbf{D}$ and a randomized dataset $\mathbf{D}_i$, that is,

$$rlift(X, \mathbf{D}, \mathbf{D}_i) = \frac{lift(X, \mathbf{D}) - lift(X, \mathbf{D}_i)}{lift(X, \mathbf{D})}$$

For an m-itemset $X = \{x_1, \ldots, x_m\}$, by Eq. (12.2) note that

$$lift(X, \mathbf{D}) = rsup(X, \mathbf{D}) \Big/ \prod_{j=1}^{m} rsup(x_j, \mathbf{D})$$

Because the swap randomization process leaves item supports (the column margins) intact, and does not change the number of transactions, we have $rsup(x_j, \mathbf{D}) = rsup(x_j, \mathbf{D}_i)$, and $|\mathbf{D}| = |\mathbf{D}_i|$. We can thus rewrite the relative lift statistic as

$$rlift(X, \mathbf{D}, \mathbf{D}_i) = \frac{sup(X, \mathbf{D}) - sup(X, \mathbf{D}_i)}{sup(X, \mathbf{D})} = 1 - \frac{sup(X, \mathbf{D}_i)}{sup(X, \mathbf{D})}$$

We generate $k = 100$ randomized datasets and compute the average relative lift for each of the 140 frequent itemsets of size two or more in the input dataset, as lift values are not defined for single items. Figure 12.4 shows the cumulative distribution for average relative lift, which ranges from -0.55 to 0.998. An average relative lift close to 1 means that the corresponding frequent pattern hardly ever occurs in any of the randomized datasets. On the other hand, a larger negative average relative lift value means that the support in randomized datasets is higher than in the input dataset. Finally, a value close to zero means that the support of the itemset is the same in both the original and randomized datasets; it is mainly a consequence of the marginal counts, and thus of little interest.

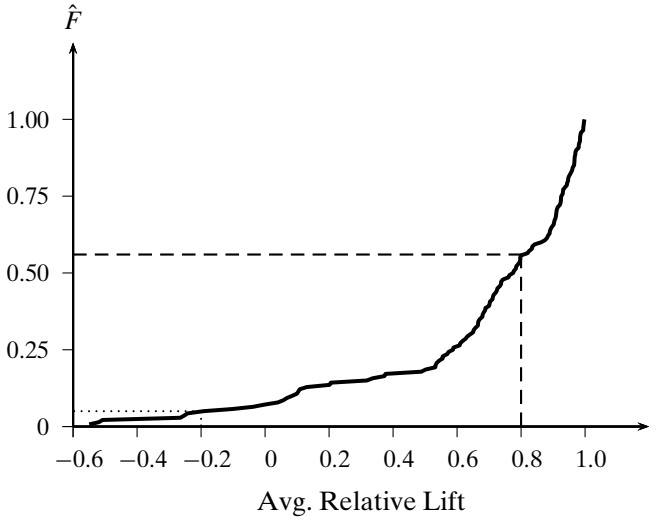


Figure 12.4. Cumulative distribution for average relative lift.

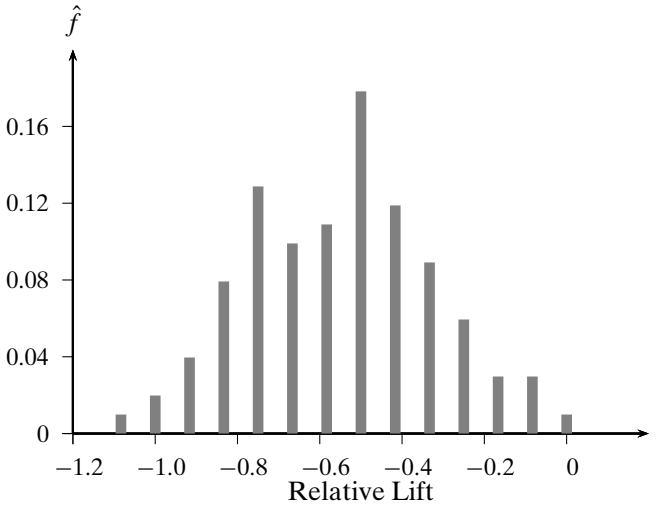


Figure 12.5. PMF for relative lift for $\{sl_1, pw_2\}$.

Figure 12.4 indicates that 44% of the frequent itemsets have average relative lift values above 0.8. These patterns are likely to be of interest. The pattern with the highest lift value of 0.998 is $\{sl_1, sw_3, pl_1, pw_1, c_1\}$. The itemset that has more or less the same support in the input and randomized datasets is $\{sl_2, c_3\}$; its average relative lift is -0.002. On the other hand, 5% of the frequent itemsets have average relative lift below -0.2. These are also of interest because they indicate more of a dis-association among the items, that is, the itemsets are more frequent by random chance. An example of such a pattern is $\{sl_1, pw_2\}$. Figure 12.5 shows the empirical probability mass function for its relative lift values across the 100 swap randomized datasets. Its average relative lift value is -0.55, and $p\text{-}value(-0.2) = 0.069$, which indicates a high probability that the itemset is disassociative.

12.2.3 Bootstrap Sampling for Confidence Interval

Typically the input transaction database $\mathbf{D}$ is just a sample from some population, and it is not enough to claim that a pattern X is frequent in $\mathbf{D}$ with support $sup(X)$. What can we say about the range of possible support values for X? Likewise, for a rule R with a given lift value in $\mathbf{D}$, what can we say about the range of lift values in different samples? In general, given a test assessment statistic Θ, bootstrap sampling allows one to infer the confidence interval for the possible values of Θ at a desired confidence level α.

The main idea is to generate k bootstrap samples from $\mathbf{D}$ using sampling *with replacement*, that is, assuming $|\mathbf{D}| = n$, each sample $\mathbf{D}_i$ is obtained by selecting at random n transactions from $\mathbf{D}$ with replacement. Given pattern X or rule $R: X \longrightarrow Y$, we can obtain the value of the test statistic in each of the bootstrap samples; let θ_i denote the value in sample $\mathbf{D}_i$. From these values we can generate the empirical

cumulative distribution function for the statistic

$$\hat{F}(x) = \hat{P}(\Theta \leq x) = \frac{1}{k}\sum_{i=1}^{k} I(\theta_i \leq x)$$

where I is an indicator variable that takes on the value 1 when its argument is true, and 0 otherwise. Given a desired confidence level α (e.g., $\alpha = 0.95$) we can compute the interval for the test statistic by discarding values from the tail ends of $\hat{F}$ on both sides that encompass $(1-\alpha)/2$ of the probability mass. Formally, let v_t denote the critical value such that $\hat{F}(v_t) = t$, which can be obtained from quantile function as $v_t = \hat{F}^{-1}(t)$. We then have

$$\begin{aligned} P\Big(\Theta \in [v_{(1-\alpha)/2}, v_{(1+\alpha)/2}]\Big) &= \hat{F}\Big((1+\alpha)/2\Big) - \hat{F}\Big((1-\alpha)/2\Big) \\ &= (1+\alpha)/2 - (1-\alpha)/2 = \alpha \end{aligned}$$

Thus, the α% confidence interval for the chosen test statistic Θ is

$$[v_{(1-\alpha)/2}, v_{(1+\alpha)/2}]$$

The pseudo-code for bootstrap sampling for estimating the confidence interval is shown in Algorithm 12.2.

ALGORITHM 12.2. Bootstrap Resampling Method

```
BOOTSTRAP-CONFIDENCEINTERVAL(X, α, k, D):
1 for i ∈ [1, k] do
2     D_i ← sample of size n with replacement from D
3     θ_i ← compute test statistic for X on D_i
4 F̂(x) = P(Θ ≤ x) = (1/k) Σ_{i=1}^{k} I(θ_i ≤ x)
5 v_{(1−α)/2} = F̂^{−1}((1 − α)/2)
6 v_{(1+α)/2} = F̂^{−1}((1 + α)/2)
7 return [v_{(1−α)/2}, v_{(1+α)/2}]
```

Example 12.21. Let the relative support *rsup* be the test statistic. Consider the itemset $X = \{sw_1, pl_3, pw_3, cl_3\}$, which has relative support $rsup(X, \mathbf{D}) = 0.113$ (or $sup(X, \mathbf{D}) = 17$) in the Iris dataset.

Using $k = 100$ bootstrap samples, we first compute the relative support of X in each of the samples ($rsup(X, \mathbf{D}_i)$). The empirical probability mass function for the relative support of X is shown in Figure 12.6 and the corresponding empirical cumulative distribution is shown in Figure 12.7. Let the confidence level be $\alpha = 0.9$. To obtain the confidence interval we have to discard the values that account for 0.05 of the probability mass at both ends of the relative support values. The critical values

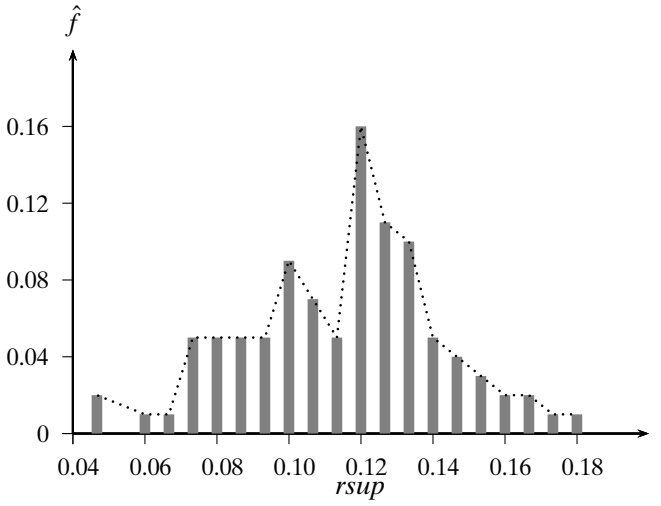


Figure 12.6. Empirical PMF for relative support.

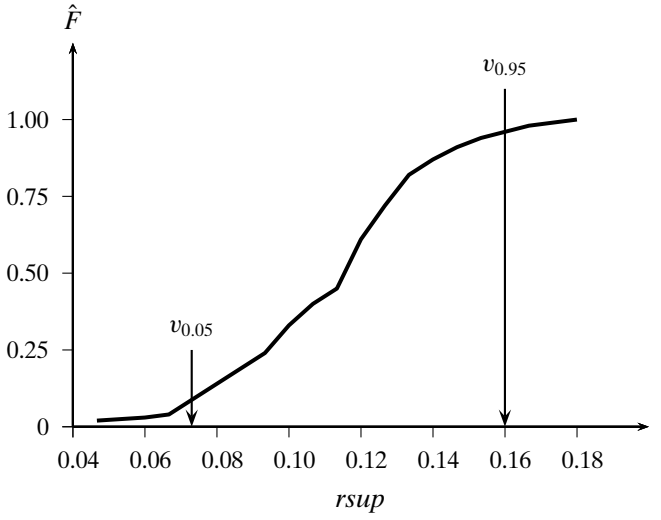


Figure 12.7. Empirical cumulative distribution for relative support.

at the left and right ends are as follows:

$$v_{(1-\alpha)/2} = v_{0.05} = 0.073$$
$$v_{(1+\alpha)/2} = v_{0.95} = 0.16$$

Thus, the 90% confidence interval for the relative support of X is $[0.073, 0.16]$, which corresponds to the interval $[11, 24]$ for its absolute support. Note that the relative support of X in the input dataset is 0.113, which has $p\text{-}value(0.113) = 0.45$, and the expected relative support value of X is $\mu_{rsup} = 0.115$.

12.3 FURTHER READING

Reviews of various measures for rule and pattern interestingness appear in Tan, Kumar, and Srivastava (2002); Geng and Hamilton (2006) and Lallich, Teytaud, and Prudhomme (2007). Randomization and resampling methods for significance testing and confidence intervals are described in Megiddo and Srikant (1998) and Gionis et al. (2007). Statistical testing and validation approaches also appear in Webb (2006) and Lallich, Teytaud, and Prudhomme (2007).

Geng, L. and Hamilton, H. J. (2006). "Interestingness measures for data mining: A survey." *ACM Computing Surveys*, 38 (3): 9.

Gionis, A., Mannila, H., Mielikäinen, T., and Tsaparas, P. (2007). "Assessing data mining results via swap randomization." *ACM Transactions on Knowledge Discovery from Data*, 1 (3): 14.

Lallich, S., Teytaud, O., and Prudhomme, E. (2007). "Association rule interestingness: measure and statistical validation." In *Quality Measures in Data Mining,* (pp. 251–275). New York: Springer Science + Business Media.

Megiddo, N. and Srikant, R. (1998). "Discovering predictive association rules." *In Proceedings of the 4th International Conference on Knowledge Discovery in Databases and Data Mining*, pp. 274–278.

Tan, P.-N., Kumar, V., and Srivastava, J. (2002). "Selecting the right interestingness measure for association patterns." *In Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,* ACM, pp. 32–41.

Webb, G. I. (2006). "Discovering significant rules." *In Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,* ACM, pp. 434–443.

12.4 EXERCISES

Q1. Show that if X and Y are independent, then $conv(X \longrightarrow Y) = 1$.

Q2. Show that if X and Y are independent then $oddsratio(X \longrightarrow Y) = 1$.

Q3. Show that for a frequent itemset X, the value of the relative lift statistic defined in Example 12.20 lies in the range

$$\left[1 - |\mathbf{D}|/minsup,\ 1\right]$$

Q4. Prove that all subsets of a minimal generator must themselves be minimal generators.

Q5. Let $\mathbf{D}$ be a binary database spanning one trillion (10^9) transactions. Because it is too time consuming to mine it directly, we use Monte Carlo sampling to find the bounds on the frequency of a given itemset X. We run 200 sampling trials $\mathbf{D}_i$ ($i = 1 \ldots 200$), with each sample of size 100, 000, and we obtain the support values for X in the various samples, as shown in Table 12.20. The table shows the number of samples where the support of the itemset was a given value. For instance, in 5 samples its support was 10,000. Answer the following questions:

Table 12.20. Data for Q5

Support	No. of samples
10,000	5
15,000	20
20,000	40
25,000	50
30,000	20
35,000	50
40,000	5
45,000	10

(a) Draw a histogram for the table, and calculate the mean and variance of the support across the different samples.

(b) Find the lower and upper bound on the support of X at the 95% confidence level. The support values given should be for the entire database **D**.

(c) Assume that $minsup = 0.25$, and let the observed support of X in a sample be $sup(X) = 32500$. Set up a hypothesis testing framework to check if the support of X is significantly higher than the *minsup* value. What is the $p\text{-}value$?

Q6. Let A and B be two binary attributes. While mining association rules at 30% minimum support and 60% minimum confidence, the following rule was mined: $A \longrightarrow B$, with $sup = 0.4$, and $conf = 0.66$. Assume that there are a total of 10,000 customers, and that 4000 of them buy both A and B; 2000 buy A but not B, 3500 buy B but not A, and 500 buy neither A nor B.

Compute the dependence between A and B via the χ^2-statistic from the corresponding contingency table. Do you think the discovered association is truly a strong rule, that is, does A predict B strongly? Set up a hypothesis testing framework, writing down the null and alternate hypotheses, to answer the above question, at the 95% confidence level. Here are some values of chi-squared statistic for the 95% confidence level for various degrees of freedom (df):

df	χ^2
1	3.84
2	5.99
3	7.82
4	9.49
5	11.07
6	12.59

PART THREE CLUSTERING

CHAPTER 13 Representative-based Clustering

Given a dataset with n points in a d-dimensional space, $\mathbf{D} = \{\mathbf{x}_i\}_{i=1}^n$, and given the number of desired clusters k, the goal of representative-based clustering is to partition the dataset into k groups or clusters, which is called a *clustering* and is denoted as $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$. Further, for each cluster C_i there exists a representative point that summarizes the cluster, a common choice being the mean (also called the *centroid*) $\boldsymbol{\mu}_i$ of all points in the cluster, that is,

$$\boldsymbol{\mu}_i = \frac{1}{n_i} \sum_{x_j \in C_i} \mathbf{x}_j$$

where $n_i = |C_i|$ is the number of points in cluster C_i.

A brute-force or exhaustive algorithm for finding a good clustering is simply to generate all possible partitions of n points into k clusters, evaluate some optimization score for each of them, and retain the clustering that yields the best score. The *exact* number of ways of partitioning n points into k nonempty and disjoint parts is given by the *Stirling numbers of the second kind*, given as

$$S(n,k) = \frac{1}{k!} \sum_{t=0}^{k} (-1)^t \binom{k}{t} (k-t)^n$$

Informally, each point can be assigned to any one of the k clusters, so there are at most k^n possible clusterings. However, any permutation of the k clusters within a given clustering yields an equivalent clustering; therefore, there are $O(k^n/k!)$ clusterings of n points into k groups. It is clear that exhaustive enumeration and scoring of all possible clusterings is not practically feasible. In this chapter we describe two approaches for representative-based clustering, namely the K-means and expectation-maximization algorithms.

13.1 K-MEANS ALGORITHM

Given a clustering $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$ we need some scoring function that evaluates its quality or goodness. This *sum of squared errors* scoring function is defined as

$$SSE(\mathcal{C}) = \sum_{i=1}^{k} \sum_{\mathbf{x}_j \in C_i} \left\| \mathbf{x}_j - \boldsymbol{\mu}_i \right\|^2 \tag{13.1}$$

The goal is to find the clustering that minimizes the SSE score:

$$\mathcal{C}^* = \arg\min_{\mathcal{C}} \{SSE(\mathcal{C})\}$$

K-means employs a greedy iterative approach to find a clustering that minimizes the SSE objective [Eq. (13.1)]. As such it can converge to a local optima instead of a globally optimal clustering.

K-means initializes the cluster means by randomly generating k points in the data space. This is typically done by generating a value uniformly at random within the range for each dimension. Each iteration of K-means consists of two steps: (1) cluster assignment, and (2) centroid update. Given the k cluster means, in the cluster assignment step, each point $\mathbf{x}_j \in \mathbf{D}$ is assigned to the closest mean, which induces a clustering, with each cluster C_i comprising points that are closer to $\boldsymbol{\mu}_i$ than any other cluster mean. That is, each point $\mathbf{x}_j$ is assigned to cluster C_{j^*}, where

$$j^* = \arg\min_{i=1}^{k} \left\{ \left\| \mathbf{x}_j - \boldsymbol{\mu}_i \right\|^2 \right\} \tag{13.2}$$

Given a set of clusters C_i, $i = 1, \ldots, k$, in the centroid update step, new mean values are computed for each cluster from the points in C_i. The cluster assignment and centroid update steps are carried out iteratively until we reach a fixed point or local minima. Practically speaking, one can assume that K-means has converged if the centroids do not change from one iteration to the next. For instance, we can stop if $\sum_{i=1}^{k} \left\| \boldsymbol{\mu}_i^t - \boldsymbol{\mu}_i^{t-1} \right\|^2 \leq \epsilon$, where $\epsilon > 0$ is the convergence threshold, t denotes the current iteration, and $\boldsymbol{\mu}_i^t$ denotes the mean for cluster C_i in iteration t.

The pseudo-code for K-means is given in Algorithm 13.1. Because the method starts with a random guess for the initial centroids, K-means is typically run several times, and the run with the lowest SSE value is chosen to report the final clustering. It is also worth noting that K-means generates convex-shaped clusters because the region in the data space corresponding to each cluster can be obtained as the intersection of half-spaces resulting from hyperplanes that bisect and are normal to the line segments that join pairs of cluster centroids.

In terms of the computational complexity of K-means, we can see that the cluster assignment step take $O(nkd)$ time because for each of the n points we have to compute its distance to each of the k clusters, which takes d operations in d dimensions. The centroid re-computation step takes $O(nd)$ time because we have to add at total of n d-dimensional points. Assuming that there are t iterations, the total time for K-means is given as $O(tnkd)$. In terms of the I/O cost it requires $O(t)$ full database scans, because we have to read the entire database in each iteration.

Example 13.1. Consider the one-dimensional data shown in Figure 13.1a. Assume that we want to cluster the data into $k = 2$ groups. Let the initial centroids be $\mu_1 = 2$ and $\mu_2 = 4$. In the first iteration, we first compute the clusters, assigning each point

ALGORITHM 13.1. K-means Algorithm

K-MEANS ($\mathbf{D}, k, \epsilon$):

```
1  t = 0
2  Randomly initialize k centroids: μ_1^t, μ_2^t, ..., μ_k^t ∈ R^d
3  repeat
4      t ← t + 1
5      C_j ← ∅ for all j = 1, ..., k
       // Cluster Assignment Step
6      foreach x_j ∈ D do
7          j* ← arg min_i { ||x_j − μ_i^t||^2 } // Assign x_j to closest centroid
8          C_j* ← C_j* ∪ {x_j}
       // Centroid Update Step
9      foreach i = 1 to k do
10         μ_i^t ← (1/|C_i|) Σ_{x_j ∈ C_i} x_j
11 until Σ_{i=1}^{k} ||μ_i^t − μ_i^{t−1}||^2 ≤ ε
```

to the closest mean, to obtain

$$C_1 = \{2, 3\} \qquad C_2 = \{4, 10, 11, 12, 20, 25, 30\}$$

We next update the means as follows:

$$\mu_1 = \frac{2+3}{2} = \frac{5}{2} = 2.5$$

$$\mu_2 = \frac{4+10+11+12+20+25+30}{7} = \frac{112}{7} = 16$$

The new centroids and clusters after the first iteration are shown in Figure 13.1b. For the second step, we repeat the cluster assignment and centroid update steps, as shown in Figure 13.1c, to obtain the new clusters:

$$C_1 = \{2, 3, 4\} \qquad C_2 = \{10, 11, 12, 20, 25, 30\}$$

and the new means:

$$\mu_1 = \frac{2+3+4}{4} = \frac{9}{3} = 3$$

$$\mu_2 = \frac{10+11+12+20+25+30}{6} = \frac{108}{6} = 18$$

The complete process until convergence is illustrated in Figure 13.1. The final clusters are given as

$$C_1 = \{2, 3, 4, 10, 11, 12\} \qquad C_2 = \{20, 25, 30\}$$

with representatives $\mu_1 = 7$ and $\mu_2 = 25$.

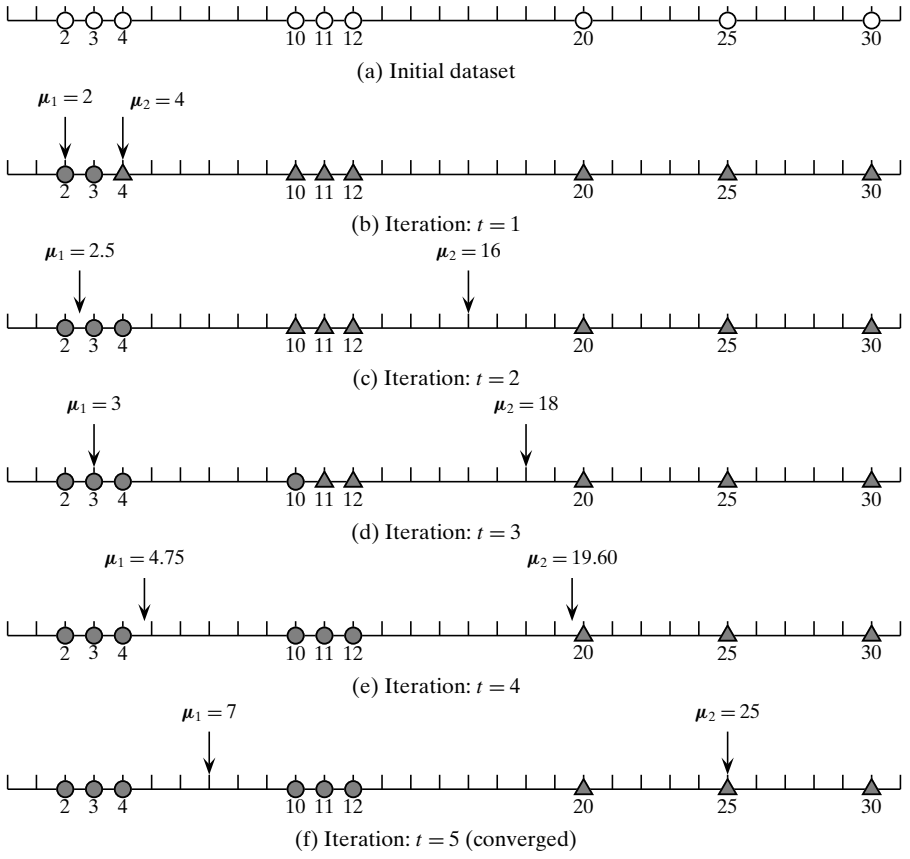


Figure 13.1. K-means in one dimension.

Example 13.2 (K-means in Two Dimensions). In Figure 13.2 we illustrate the K-means algorithm on the Iris dataset, using the first two principal components as the two dimensions. Iris has $n = 150$ points, and we want to find $k = 3$ clusters, corresponding to the three types of Irises. A random initialization of the cluster means yields

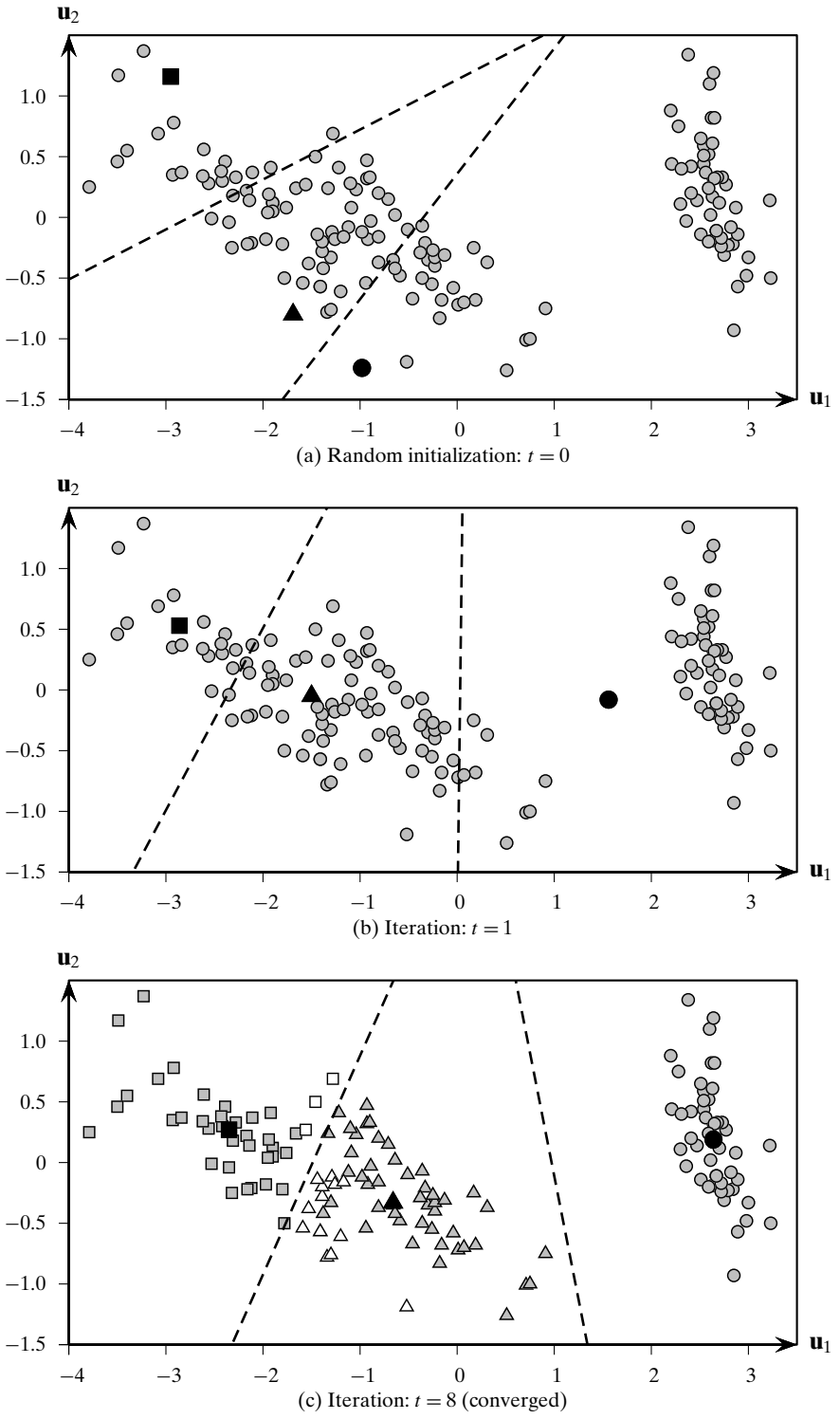
$$\boldsymbol{\mu}_1 = (-0.98, -1.24)^T \qquad \boldsymbol{\mu}_2 = (-2.96, 1.16)^T \qquad \boldsymbol{\mu}_3 = (-1.69, -0.80)^T$$

as shown in Figure 13.2a. With these initial clusters, K-means takes eight iterations to converge. Figure 13.2b shows the clusters and their means after one iteration:

$$\boldsymbol{\mu}_1 = (1.56, -0.08)^T \qquad \boldsymbol{\mu}_2 = (-2.86, 0.53)^T \qquad \boldsymbol{\mu}_3 = (-1.50, -0.05)^T$$

Finally, Figure 13.2c shows the clusters on convergence. The final means are as follows:

$$\boldsymbol{\mu}_1 = (2.64, 0.19)^T \qquad \boldsymbol{\mu}_2 = (-2.35, 0.27)^T \qquad \boldsymbol{\mu}_3 = (-0.66, -0.33)^T$$



(a) Random initialization: $t = 0$

(b) Iteration: $t = 1$

(c) Iteration: $t = 8$ (converged)

Figure 13.2. K-means in two dimensions: Iris principal components dataset.

Figure 13.2 shows the cluster means as black points, and shows the convex regions of data space that correspond to each of the three clusters. The dashed lines (hyperplanes) are the perpendicular bisectors of the line segments joining two cluster centers. The resulting convex partition of the points comprises the clustering.

Figure 13.2c shows the final three clusters: C_1 as circles, C_2 as squares, and C_3 as triangles. White points indicate a wrong grouping when compared to the known Iris types. Thus, we can see that C_1 perfectly corresponds to `iris-setosa`, and the majority of the points in C_2 correspond to `iris-virginica`, and in C_3 to `iris-versicolor`. For example, three points (white squares) of type `iris-versicolor` are wrongly clustered in C_2, and 14 points from `iris-virginica` are wrongly clustered in C_3 (white triangles). Of course, because the Iris class label is not used in clustering, it is reasonable to expect that we will not obtain a perfect clustering.

13.2 KERNEL K-MEANS

In K-means, the separating boundary between clusters is linear. Kernel K-means allows one to extract nonlinear boundaries between clusters via the use of the kernel trick outlined in Chapter 5. This way the method can be used to detect nonconvex clusters.

In kernel K-means, the main idea is to conceptually map a data point $\mathbf{x}_i$ in input space to a point $\phi(\mathbf{x}_i)$ in some high-dimensional feature space, via an appropriate nonlinear mapping ϕ. However, the kernel trick allows us to carry out the clustering in feature space purely in terms of the kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$, which can be computed in input space, but corresponds to a dot (or inner) product $\phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$ in feature space.

Assume for the moment that all points $\mathbf{x}_i \in \mathbf{D}$ have been mapped to their corresponding images $\phi(\mathbf{x}_i)$ in feature space. Let $\mathbf{K} = \{K(\mathbf{x}_i, \mathbf{x}_j)\}_{i,j=1,\ldots,n}$ denote the $n \times n$ symmetric kernel matrix, where $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$. Let $\{C_1, \ldots, C_k\}$ specify the partitioning of the n points into k clusters, and let the corresponding cluster means in feature space be given as $\{\boldsymbol{\mu}_1^\phi, \ldots, \boldsymbol{\mu}_k^\phi\}$, where

$$\boldsymbol{\mu}_i^\phi = \frac{1}{n_i} \sum_{\mathbf{x}_j \in C_i} \phi(\mathbf{x}_j)$$

is the mean of cluster C_i in feature space, with $n_i = |C_i|$.

In feature space, the kernel K-means sum of squared errors objective can be written as

$$\min_{\mathcal{C}} SSE(\mathcal{C}) = \sum_{i=1}^{k} \sum_{\mathbf{x}_j \in C_i} \left\| \phi(\mathbf{x}_j) - \boldsymbol{\mu}_i^\phi \right\|^2$$

Expanding the kernel SSE objective in terms of the kernel function, we get

$$\begin{aligned} SSE(\mathcal{C}) &= \sum_{i=1}^{k} \sum_{\mathbf{x}_j \in C_i} \left\| \phi(\mathbf{x}_j) - \boldsymbol{\mu}_i^\phi \right\|^2 \\ &= \sum_{i=1}^{k} \sum_{\mathbf{x}_j \in C_i} \left\| \phi(\mathbf{x}_j) \right\|^2 - 2\phi(\mathbf{x}_j)^T \boldsymbol{\mu}_i^\phi + \left\| \boldsymbol{\mu}_i^\phi \right\|^2 \end{aligned}$$

$$
\begin{aligned}
&= \sum_{i=1}^{k}\Big(\Big(\sum_{\mathbf{x}_j \in C_i} \big\|\phi(\mathbf{x}_j)\big\|^2\Big) - 2n_i\Big(\frac{1}{n_i}\sum_{\mathbf{x}_j \in C_i}\phi(\mathbf{x}_j)\Big)^T \boldsymbol{\mu}_i^{\phi} + n_i \left\|\boldsymbol{\mu}_i^{\phi}\right\|^2\Big) \\
&= \Big(\sum_{i=1}^{k}\sum_{\mathbf{x}_j \in C_i}\phi(\mathbf{x}_j)^T\phi(\mathbf{x}_j)\Big) - \Big(\sum_{i=1}^{k} n_i \left\|\boldsymbol{\mu}_i^{\phi}\right\|^2\Big) \\
&= \sum_{i=1}^{k}\sum_{\mathbf{x}_j \in C_i} K(\mathbf{x}_j, \mathbf{x}_j) - \sum_{i=1}^{k}\frac{1}{n_i}\sum_{\mathbf{x}_a \in C_i}\sum_{\mathbf{x}_b \in C_i} K(\mathbf{x}_a, \mathbf{x}_b) \\
&= \sum_{j=1}^{n} K(\mathbf{x}_j, \mathbf{x}_j) - \sum_{i=1}^{k}\frac{1}{n_i}\sum_{\mathbf{x}_a \in C_i}\sum_{\mathbf{x}_b \in C_i} K(\mathbf{x}_a, \mathbf{x}_b)
\end{aligned} \tag{13.3}
$$

Thus, the kernel K-means SSE objective function can be expressed purely in terms of the kernel function. Like K-means, to minimize the SSE objective we adopt a greedy iterative approach. The basic idea is to assign each point to the closest mean in feature space, resulting in a new clustering, which in turn can be used obtain new estimates for the cluster means. However, the main difficulty is that we cannot explicitly compute the mean of each cluster in feature space. Fortunately, explicitly obtaining the cluster means is not required; all operations can be carried out in terms of the kernel function $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$.

Consider the distance of a point $\phi(\mathbf{x}_j)$ to the mean $\boldsymbol{\mu}_i^{\phi}$ in feature space, which can be computed as

$$
\begin{aligned}
\left\|\phi(\mathbf{x}_j) - \boldsymbol{\mu}_i^{\phi}\right\|^2 &= \big\|\phi(\mathbf{x}_j)\big\|^2 - 2\phi(\mathbf{x}_j)^T\boldsymbol{\mu}_i^{\phi} + \left\|\boldsymbol{\mu}_i^{\phi}\right\|^2 \\
&= \phi(\mathbf{x}_j)^T\phi(\mathbf{x}_j) - \frac{2}{n_i}\sum_{\mathbf{x}_a \in C_i}\phi(\mathbf{x}_j)^T\phi(\mathbf{x}_a) + \frac{1}{n_i^2}\sum_{\mathbf{x}_a \in C_i}\sum_{\mathbf{x}_b \in C_i}\phi(\mathbf{x}_a)^T\phi(\mathbf{x}_b) \\
&= K(\mathbf{x}_j, \mathbf{x}_j) - \frac{2}{n_i}\sum_{\mathbf{x}_a \in C_i} K(\mathbf{x}_a, \mathbf{x}_j) + \frac{1}{n_i^2}\sum_{\mathbf{x}_a \in C_i}\sum_{\mathbf{x}_b \in C_i} K(\mathbf{x}_a, \mathbf{x}_b)
\end{aligned} \tag{13.4}
$$

Thus, the distance of a point to a cluster mean in feature space can be computed using only kernel operations. In the cluster assignment step of kernel K-means, we assign a point to the closest cluster mean as follows:

$$
\begin{aligned}
C^*(\mathbf{x}_j) &= \arg\min_i \left\{\left\|\phi(\mathbf{x}_j) - \boldsymbol{\mu}_i^{\phi}\right\|^2\right\} \\
&= \arg\min_i \left\{K(\mathbf{x}_j, \mathbf{x}_j) - \frac{2}{n_i}\sum_{\mathbf{x}_a \in C_i} K(\mathbf{x}_a, \mathbf{x}_j) + \frac{1}{n_i^2}\sum_{\mathbf{x}_a \in C_i}\sum_{\mathbf{x}_b \in C_i} K(\mathbf{x}_a, \mathbf{x}_b)\right\} \\
&= \arg\min_i \left\{\frac{1}{n_i^2}\sum_{\mathbf{x}_a \in C_i}\sum_{\mathbf{x}_b \in C_i} K(\mathbf{x}_a, \mathbf{x}_b) - \frac{2}{n_i}\sum_{\mathbf{x}_a \in C_i} K(\mathbf{x}_a, \mathbf{x}_j)\right\}
\end{aligned} \tag{13.5}
$$

where we drop the $K(\mathbf{x}_j, \mathbf{x}_j)$ term because it remains the same for all k clusters and does not impact the cluster assignment decision. Also note that the first term is simply the average pairwise kernel value for cluster C_i and is independent of the point $\mathbf{x}_j$. It is in fact the squared norm of the cluster mean in feature space. The second term is twice the average kernel value for points in C_i with respect to $\mathbf{x}_j$.

Algorithm 13.2 shows the pseudo-code for the kernel K-means method. It starts from an initial random partitioning of the points into k clusters. It then iteratively updates the cluster assignments by reassigning each point to the closest mean in feature space via Eq. (13.5). To facilitate the distance computation, it first computes the average kernel value, that is, the squared norm of the cluster mean, for each cluster (for loop in line 5). Next, it computes the average kernel value for each point $\mathbf{x}_j$ with points in cluster C_i (for loop in line 7). The main cluster assignment step uses these values to compute the distance of $\mathbf{x}_j$ from each of the clusters C_i and assigns $\mathbf{x}_j$ to the closest mean. This reassignment information is used to re-partition the points into a new set of clusters. That is, all points $\mathbf{x}_j$ that are closer to the mean for C_i make up the new cluster for the next iteration. This iterative process is repeated until convergence.

For convergence testing, we check if there is any change in the cluster assignments of the points. The number of points that do not change clusters is given as the sum $\sum_{i=1}^{k} |C_i^t \cap C_i^{t-1}|$, where t specifies the current iteration. The fraction of points

ALGORITHM 13.2. Kernel K-means Algorithm

KERNEL-KMEANS($\mathbf{K}, k, \epsilon$):

1 $t \leftarrow 0$
2 $\mathcal{C}^t \leftarrow \{C_1^t, \ldots, C_k^t\}$ // Randomly partition points into k clusters
3 **repeat**
4 $\quad t \leftarrow t + 1$
5 $\quad$ **foreach** $C_i \in \mathcal{C}^{t-1}$ **do** // Compute squared norm of cluster means
6 $\quad\quad \text{sqnorm}_i \leftarrow \frac{1}{n_i^2} \sum_{\mathbf{x}_a \in C_i} \sum_{\mathbf{x}_b \in C_i} K(\mathbf{x}_a, \mathbf{x}_b)$
7 $\quad$ **foreach** $\mathbf{x}_j \in \mathbf{D}$ **do** // Average kernel value for $\mathbf{x}_j$ and C_i
8 $\quad\quad$ **foreach** $C_i \in \mathcal{C}^{t-1}$ **do**
9 $\quad\quad\quad \text{avg}_{ji} \leftarrow \frac{1}{n_i} \sum_{\mathbf{x}_a \in C_i} K(\mathbf{x}_a, \mathbf{x}_j)$
$\quad$ // Find closest cluster for each point
10 $\quad$ **foreach** $\mathbf{x}_j \in \mathbf{D}$ **do**
11 $\quad\quad$ **foreach** $C_i \in \mathcal{C}^{t-1}$ **do**
12 $\quad\quad\quad d(\mathbf{x}_j, C_i) \leftarrow \text{sqnorm}_i - 2 \cdot \text{avg}_{ji}$
13 $\quad\quad j^* \leftarrow \arg\min_i \{d(\mathbf{x}_j, C_i)\}$
14 $\quad\quad C_{j^*}^t \leftarrow C_{j^*}^t \cup \{\mathbf{x}_j\}$ // Cluster reassignment
15 $\quad \mathcal{C}^t \leftarrow \{C_1^t, \ldots, C_k^t\}$
16 **until** $1 - \frac{1}{n} \sum_{i=1}^{k} |C_i^t \cap C_i^{t-1}| \leq \epsilon$

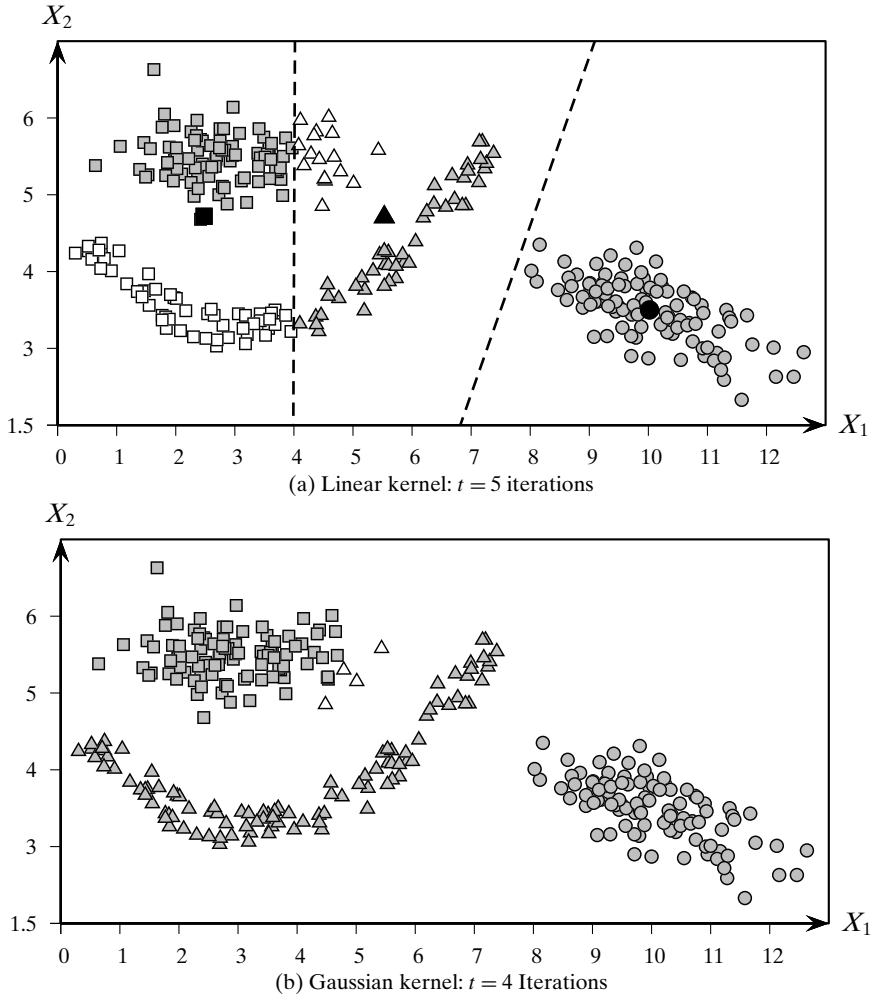


Figure 13.3. Kernel K-means: linear versus Gaussian kernel.

reassigned to a different cluster in the current iteration is given as

$$\frac{n - \sum_{i=1}^{k} |C_i^t \cap C_i^{t-1}|}{n} = 1 - \frac{1}{n} \sum_{i=1}^{k} |C_i^t \cap C_i^{t-1}|$$

Kernel K-means stops when the fraction of points with new cluster assignments falls below some threshold $\epsilon \geq 0$. For example, one can iterate until no points change clusters.

Computational Complexity

Computing the average kernel value for each cluster C_i takes time $O(n^2)$ across all clusters. Computing the average kernel value of each point with respect to each of the k clusters also takes $O(n^2)$ time. Finally, computing the closest mean for each point and cluster reassignment takes $O(kn)$ time. The total computational complexity of kernel

K-means is thus $O(tn^2)$, where t is the number of iterations until convergence. The I/O complexity is $O(t)$ scans of the kernel matrix $\mathbf{K}$.

Example 13.3. Figure 13.3 shows an application of the kernel K-means approach on a synthetic dataset with three embedded clusters. Each cluster has 100 points, for a total of $n = 300$ points in the dataset.

Using the linear kernel $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$ is equivalent to the K-means algorithm because in this case Eq. (13.5) is the same as Eq. (13.2). Figure 13.3a shows the resulting clusters; points in C_1 are shown as squares, in C_2 as triangles, and in C_3 as circles. We can see that K-means is not able to separate the three clusters due to the presence of the parabolic shaped cluster. The white points are those that are wrongly clustered, comparing with the ground truth in terms of the generated cluster labels.

Using the Gaussian kernel $K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left\{-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right\}$ from Eq. (5.10), with $\sigma = 1.5$, results in a near-perfect clustering, as shown in Figure 13.3b. Only four points (white triangles) are grouped incorrectly with cluster C_2, whereas they should belong to cluster C_1. We can see from this example that kernel K-means is able to handle nonlinear cluster boundaries. One caveat is that the value of the spread parameter σ has to be set by trial and error.

13.3 EXPECTATION-MAXIMIZATION CLUSTERING

The K-means approach is an example of a *hard assignment* clustering, where each point can belong to only one cluster. We now generalize the approach to consider *soft assignment* of points to clusters, so that each point has a probability of belonging to each cluster.

Let $\mathbf{D}$ consist of n points $\mathbf{x}_j$ in d-dimensional space $\mathbb{R}^d$. Let X_a denote the random variable corresponding to the ath attribute. We also use X_a to denote the ath column vector, corresponding to the n data samples from X_a. Let $\mathbf{X} = (X_1, X_2, \ldots, X_d)$ denote the vector random variable across the d-attributes, with $\mathbf{x}_j$ being a data sample from $\mathbf{X}$.

Gaussian Mixture Model

We assume that each cluster C_i is characterized by a multivariate normal distribution, that is,

$$f_i(\mathbf{x}) = f(\mathbf{x}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) = \frac{1}{(2\pi)^{\frac{d}{2}} |\boldsymbol{\Sigma}_i|^{\frac{1}{2}}} \exp\left\{-\frac{(\mathbf{x} - \boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}_i^{-1} (\mathbf{x} - \boldsymbol{\mu}_i)}{2}\right\} \tag{13.6}$$

where the cluster mean $\boldsymbol{\mu}_i \in \mathbb{R}^d$ and covariance matrix $\boldsymbol{\Sigma}_i \in \mathbb{R}^{d \times d}$ are both unknown parameters. $f_i(\mathbf{x})$ is the probability density at $\mathbf{x}$ attributable to cluster C_i. We assume that the probability density function of $\mathbf{X}$ is given as a *Gaussian mixture model* over all

the k cluster normals, defined as

$$f(\mathbf{x}) = \sum_{i=1}^{k} f_i(\mathbf{x}) P(C_i) = \sum_{i=1}^{k} f(\mathbf{x}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) P(C_i) \tag{13.7}$$

where the prior probabilities $P(C_i)$ are called the *mixture parameters*, which must satisfy the condition

$$\sum_{i=1}^{k} P(C_i) = 1$$

The Gaussian mixture model is thus characterized by the mean $\boldsymbol{\mu}_i$, the covariance matrix $\boldsymbol{\Sigma}_i$, and the mixture probability $P(C_i)$ for each of the k normal distributions. We write the set of all the model parameters compactly as

$$\boldsymbol{\theta} = \{\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1, P(C_i) \ldots, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k, P(C_k)\}$$

Maximum Likelihood Estimation

Given the dataset $\mathbf{D}$, we define the *likelihood* of $\boldsymbol{\theta}$ as the conditional probability of the data $\mathbf{D}$ given the model parameters $\boldsymbol{\theta}$, denoted as $P(\mathbf{D}|\boldsymbol{\theta})$. Because each of the n points $\mathbf{x}_j$ is considered to be a random sample from $\mathbf{X}$ (i.e., independent and identically distributed as $\mathbf{X}$), the likelihood of $\boldsymbol{\theta}$ is given as

$$P(\mathbf{D}|\boldsymbol{\theta}) = \prod_{j=1}^{n} f(\mathbf{x}_j)$$

The goal of maximum likelihood estimation (MLE) is to choose the parameters $\boldsymbol{\theta}$ that maximize the likelihood, that is,

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \{P(\mathbf{D}|\boldsymbol{\theta})\}$$

It is typical to maximize the log of the likelihood function because it turns the product over the points into a summation and the maximum value of the likelihood and log-likelihood coincide. That is, MLE maximizes

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \{\ln P(\mathbf{D}|\boldsymbol{\theta})\}$$

where the *log-likelihood* function is given as

$$\ln P(\mathbf{D}|\boldsymbol{\theta}) = \sum_{j=1}^{n} \ln f(\mathbf{x}_j) = \sum_{j=1}^{n} \ln\left(\sum_{i=1}^{k} f(\mathbf{x}_j|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) P(C_i)\right) \tag{13.8}$$

Directly maximizing the log-likelihood over $\boldsymbol{\theta}$ is hard. Instead, we can use the expectation-maximization (EM) approach for finding the maximum likelihood estimates for the parameters $\boldsymbol{\theta}$. EM is a two-step iterative approach that starts from an initial guess for the parameters $\boldsymbol{\theta}$. Given the current estimates for $\boldsymbol{\theta}$, in the *expectation step* EM computes the cluster posterior probabilities $P(C_i|\mathbf{x}_j)$ via the Bayes theorem:

$$P(C_i|\mathbf{x}_j) = \frac{P(C_i \text{ and } \mathbf{x}_j)}{P(\mathbf{x}_j)} = \frac{P(\mathbf{x}_j|C_i)P(C_i)}{\sum_{a=1}^{k} P(\mathbf{x}_j|C_a)P(C_a)}$$

Because each cluster is modeled as a multivariate normal distribution [Eq. (13.6)], the probability of $\mathbf{x}_j$ given cluster C_i can be obtained by considering a small interval $\epsilon > 0$ centered at $\mathbf{x}_j$, as follows:

$$P(\mathbf{x}_j|C_i) \simeq 2\epsilon \cdot f(\mathbf{x}_j|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) = 2\epsilon \cdot f_i(\mathbf{x}_j)$$

The posterior probability of C_i given $\mathbf{x}_j$ is thus given as

$$P(C_i|\mathbf{x}_j) = \frac{f_i(\mathbf{x}_j) \cdot P(C_i)}{\sum_{a=1}^{k} f_a(\mathbf{x}_j) \cdot P(C_a)} \tag{13.9}$$

and $P(C_i|\mathbf{x}_j)$ can be considered as the weight or contribution of the point $\mathbf{x}_j$ to cluster C_i. Next, in the *maximization step*, using the weights $P(C_i|\mathbf{x}_j)$ EM re-estimates $\boldsymbol{\theta}$, that is, it re-estimates the parameters $\boldsymbol{\mu}_i$, $\boldsymbol{\Sigma}_i$, and $P(C_i)$ for each cluster C_i. The re-estimated mean is given as the weighted average of all the points, the re-estimated covariance matrix is given as the weighted covariance over all pairs of dimensions, and the re-estimated prior probability for each cluster is given as the fraction of weights that contribute to that cluster. In Section 13.3.3 we formally derive the expressions for the MLE estimates for the cluster parameters, and in Section 13.3.4 we describe the generic EM approach in more detail. We begin with the application of the EM clustering algorithm for the one-dimensional and general d-dimensional cases.

13.3.1 EM in One Dimension

Consider a dataset $\mathbf{D}$ consisting of a single attribute X, where each point $x_j \in \mathbb{R}$ ($j = 1, \ldots, n$) is a random sample from X. For the mixture model [Eq. (13.7)], we use univariate normals for each cluster:

$$f_i(x) = f(x|\mu_i, \sigma_i^2) = \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left\{-\frac{(x-\mu_i)^2}{2\sigma_i^2}\right\}$$

with the cluster parameters μ_i, σ_i^2, and $P(C_i)$. The EM approach consists of three steps: initialization, expectation step, and maximization step.

Initialization

For each cluster C_i, with $i = 1, 2, \ldots, k$, we can randomly initialize the cluster parameters μ_i, σ_i^2, and $P(C_i)$. The mean μ_i is selected uniformly at random from the range of possible values for X. It is typical to assume that the initial variance is given as $\sigma_i^2 = 1$. Finally, the cluster prior probabilities are initialized to $P(C_i) = \frac{1}{k}$, so that each cluster has an equal probability.

Expectation Step

Assume that for each of the k clusters, we have an estimate for the parameters, namely the mean μ_i, variance σ_i^2, and prior probability $P(C_i)$. Given these values the clusters posterior probabilities are computed using Eq. (13.9):

$$P(C_i|x_j) = \frac{f(x_j|\mu_i, \sigma_i^2) \cdot P(C_i)}{\sum_{a=1}^{k} f(x_j|\mu_a, \sigma_a^2) \cdot P(C_a)}$$

For convenience, we use the notation $w_{ij} = P(C_i|x_j)$, treating the posterior probability as the weight or contribution of the point x_j to cluster C_i. Further, let

$$\mathbf{w}_i = (w_{i1}, \ldots, w_{in})^T$$

denote the weight vector for cluster C_i across all the n points.

Maximization Step

Assuming that all the posterior probability values or weights $w_{ij} = P(C_i|x_j)$ are known, the maximization step, as the name implies, computes the maximum likelihood estimates of the cluster parameters by re-estimating μ_i, σ_i^2, and $P(C_i)$.

The re-estimated value for the cluster mean, μ_i, is computed as the weighted mean of all the points:

$$\mu_i = \frac{\sum_{j=1}^{n} w_{ij} \cdot x_j}{\sum_{j=1}^{n} w_{ij}}$$

In terms of the weight vector $\mathbf{w}_i$ and the attribute vector $X = (x_1, x_2, \ldots, x_n)^T$, we can rewrite the above as

$$\mu_i = \frac{\mathbf{w}_i^T X}{\mathbf{w}_i^T \mathbf{1}}$$

The re-estimated value of the cluster variance is computed as the weighted variance across all the points:

$$\sigma_i^2 = \frac{\sum_{j=1}^{n} w_{ij}(x_j - \mu_i)^2}{\sum_{j=1}^{n} w_{ij}}$$

Let $Z_i = X - \mu_i \mathbf{1} = (x_1 - \mu_i, x_2 - \mu_i, \ldots, x_n - \mu_i)^T = (z_{i1}, z_{i2}, \ldots, z_{in})^T$ be the centered attribute vector for cluster C_i, and let Z_i^s be the squared vector given as $Z_i^s = (z_{i1}^2, \ldots, z_{in}^2)^T$. The variance can be expressed compactly in terms of the dot product between the weight vector and the squared centered vector:

$$\sigma_i^2 = \frac{\mathbf{w}_i^T Z_i^s}{\mathbf{w}_i^T \mathbf{1}}$$

Finally, the prior probability of cluster C_i is re-estimated as the fraction of the total weight belonging to C_i, computed as

$$P(C_i) = \frac{\sum_{j=1}^{n} w_{ij}}{\sum_{a=1}^{k} \sum_{j=1}^{n} w_{aj}} = \frac{\sum_{j=1}^{n} w_{ij}}{\sum_{j=1}^{n} 1} = \frac{\sum_{j=1}^{n} w_{ij}}{n} \tag{13.10}$$

where we made use of the fact that

$$\sum_{i=1}^{k} w_{ij} = \sum_{i=1}^{k} P(C_i|x_j) = 1$$

In vector notation the prior probability can be written as

$$P(C_i) = \frac{\mathbf{w}_i^T \mathbf{1}}{n}$$

Iteration

Starting from an initial set of values for the cluster parameters μ_i, σ_i^2 and $P(C_i)$ for all $i = 1, \ldots, k$, the EM algorithm applies the expectation step to compute the weights $w_{ij} = P(C_i|x_j)$. These values are then used in the maximization step to compute the updated cluster parameters μ_i, σ_i^2 and $P(C_i)$. Both the expectation and maximization steps are iteratively applied until convergence, for example, until the means change very little from one iteration to the next.

Example 13.4 (EM in 1D). Figure 13.4 illustrates the EM algorithm on the one-dimensional dataset:

$$x_1 = 1.0 \quad x_2 = 1.3 \quad x_3 = 2.2 \quad x_4 = 2.6 \quad x_5 = 2.8$$
$$x_6 = 5.0 \quad x_7 = 7.3 \quad x_8 = 7.4 \quad x_9 = 7.5 \quad x_{10} = 7.7 \quad x_{11} = 7.9$$

We assume that $k = 2$. The initial random means are shown in Figure 13.4a, with the initial parameters given as

$$\mu_1 = 6.63 \qquad \sigma_1^2 = 1 \qquad P(C_2) = 0.5$$
$$\mu_2 = 7.57 \qquad \sigma_2^2 = 1 \qquad P(C_2) = 0.5$$

After repeated expectation and maximization steps, the EM method converges after five iterations. After $t = 1$ (see Figure 13.4b) we have

$$\mu_1 = 3.72 \qquad \sigma_1^2 = 6.13 \qquad P(C_1) = 0.71$$
$$\mu_2 = 7.4 \qquad \sigma_2^2 = 0.69 \qquad P(C_2) = 0.29$$

After the final iteration ($t = 5$), as shown in Figure 13.4c, we have

$$\mu_1 = 2.48 \qquad \sigma_1^2 = 1.69 \qquad P(C_1) = 0.55$$
$$\mu_2 = 7.56 \qquad \sigma_2^2 = 0.05 \qquad P(C_2) = 0.45$$

One of the main advantages of the EM algorithm over K-means is that it returns the probability $P(C_i|\mathbf{x}_j)$ of each cluster C_i for each point $\mathbf{x}_j$. However, in this 1-dimensional example, these values are essentially binary; assigning each point to the cluster with the highest posterior probability, we obtain the hard clustering

$$C_1 = \{x_1, x_2, x_3, x_4, x_5, x_6\} \text{ (white points)}$$
$$C_2 = \{x_7, x_8, x_9, x_{10}, x_{11}\} \text{ (gray points)}$$

as illustrated in Figure 13.4c.

13.3.2 EM in d Dimensions

We now consider the EM method in d dimensions, where each cluster is characterized by a multivariate normal distribution [Eq. (13.6)], with parameters $\boldsymbol{\mu}_i$, $\boldsymbol{\Sigma}_i$, and $P(C_i)$. For each cluster C_i, we thus need to estimate the d-dimensional mean vector:

$$\boldsymbol{\mu}_i = (\mu_{i1}, \mu_{i2}, \ldots, \mu_{id})^T$$

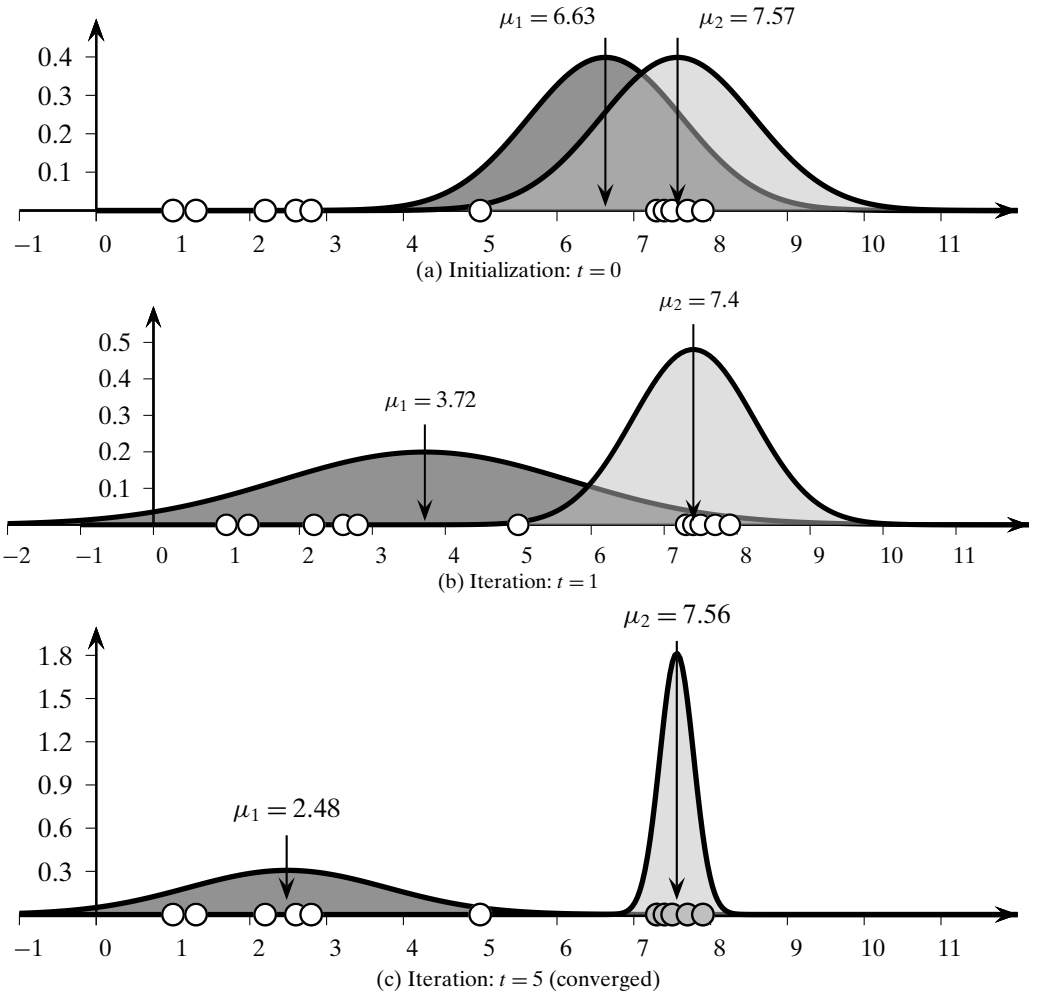


Figure 13.4. EM in one dimension.

and the $d \times d$ covariance matrix:

$$\boldsymbol{\Sigma}_i = \begin{pmatrix} (\sigma_1^i)^2 & \sigma_{12}^i & \ldots & \sigma_{1d}^i \\ \sigma_{21}^i & (\sigma_2^i)^2 & \ldots & \sigma_{2d}^i \\ \vdots & \vdots & \ddots & \\ \sigma_{d1}^i & \sigma_{d2}^i & \ldots & (\sigma_d^i)^2 \end{pmatrix}$$

Because the covariance matrix is symmetric, we have to estimate $\binom{d}{2} = \frac{d(d-1)}{2}$ pairwise covariances and d variances, for a total of $\frac{d(d+1)}{2}$ parameters for $\boldsymbol{\Sigma}_i$. This may be too many parameters for practical purposes because we may not have enough data to estimate all of them reliably. For example, if $d = 100$, then we have to estimate $100 \cdot 101/2 = 5050$ parameters! One simplification is to assume that all dimensions are

independent, which leads to a diagonal covariance matrix:

$$\boldsymbol{\Sigma}_i = \begin{pmatrix} (\sigma_1^i)^2 & 0 & \ldots & 0 \\ 0 & (\sigma_2^i)^2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \\ 0 & 0 & \ldots & (\sigma_d^i)^2 \end{pmatrix}$$

Under the independence assumption we have only d parameters to estimate for the diagonal covariance matrix.

Initialization

For each cluster C_i, with $i = 1, 2, \ldots, k$, we randomly initialize the mean $\boldsymbol{\mu}_i$ by selecting a value μ_{ia} for each dimension X_a uniformly at random from the range of X_a. The covariance matrix is initialized as the $d \times d$ identity matrix, $\boldsymbol{\Sigma}_i = \mathbf{I}$. Finally, the cluster prior probabilities are initialized to $P(C_i) = \frac{1}{k}$, so that each cluster has an equal probability.

Expectation Step

In the expectation step, we compute the posterior probability of cluster C_i given point $\mathbf{x}_j$ using Eq. (13.9), with $i = 1, \ldots, k$ and $j = 1, \ldots, n$. As before, we use the shorthand notation $w_{ij} = P(C_i|\mathbf{x}_j)$ to denote the fact that $P(C_i|\mathbf{x}_j)$ can be considered as the weight or contribution of point $\mathbf{x}_j$ to cluster C_i, and we use the notation $\mathbf{w}_i = (w_{i1}, w_{i2}, \ldots, w_{in})^T$ to denote the weight vector for cluster C_i, across all the n points.

Maximization Step

Given the weights w_{ij}, in the maximization step, we re-estimate $\boldsymbol{\Sigma}_i$, $\boldsymbol{\mu}_i$ and $P(C_i)$. The mean $\boldsymbol{\mu}_i$ for cluster C_i can be estimated as

$$\boldsymbol{\mu}_i = \frac{\sum_{j=1}^n w_{ij} \cdot \mathbf{x}_j}{\sum_{j=1}^n w_{ij}} \tag{13.11}$$

which can be expressed compactly in matrix form as

$$\boldsymbol{\mu}_i = \frac{\mathbf{D}^T \mathbf{w}_i}{\mathbf{w}_i^T \mathbf{1}}$$

Let $\mathbf{Z}_i = \mathbf{D} - \mathbf{1} \cdot \boldsymbol{\mu}_i^T$ be the centered data matrix for cluster C_i. Let $\mathbf{z}_{ji} = \mathbf{x}_j - \boldsymbol{\mu}_i \in \mathbb{R}^d$ denote the jth centered point in $\mathbf{Z}_i$. We can express $\boldsymbol{\Sigma}_i$ compactly using the outer-product form

$$\boldsymbol{\Sigma}_i = \frac{\sum_{j=1}^n w_{ij} \mathbf{z}_{ji} \mathbf{z}_{ji}^T}{\mathbf{w}_i^T \mathbf{1}} \tag{13.12}$$

Considering the pairwise attribute view, the covariance between dimensions X_a and X_b is estimated as

$$\sigma_{ab}^i = \frac{\sum_{j=1}^n w_{ij}(x_{ja} - \mu_{ia})(x_{jb} - \mu_{ib})}{\sum_{j=1}^n w_{ij}}$$

where x_{ja} and μ_{ia} denote the values of the ath dimension for $\mathbf{x}_j$ and $\boldsymbol{\mu}_i$, respectively.

Finally, the prior probability $P(C_i)$ for each cluster is the same as in the one-dimensional case [Eq. (13.10)], given as

$$P(C_i) = \frac{\sum_{j=1}^{n} w_{ij}}{n} = \frac{\mathbf{w}_i^T \mathbf{1}}{n} \tag{13.13}$$

A formal derivation of these re-estimates for $\boldsymbol{\mu}_i$ [Eq. (13.11)], $\boldsymbol{\Sigma}_i$ [Eq. (13.12)], and $P(C_i)$ [Eq. (13.13)] is given in Section 13.3.3.

EM Clustering Algorithm

The pseudo-code for the multivariate EM clustering algorithm is given in Algorithm 13.3. After initialization of $\boldsymbol{\mu}_i$, $\boldsymbol{\Sigma}_i$, and $P(C_i)$ for all $i = 1, \ldots, k$, the expectation and maximization steps are repeated until convergence. For the convergence test, we check whether $\sum_i \left\| \boldsymbol{\mu}_i^t - \boldsymbol{\mu}_i^{t-1} \right\|^2 \leq \epsilon$, where $\epsilon > 0$ is the convergence threshold, and t denotes the iteration. In words, the iterative process continues until the change in the cluster means becomes very small.

ALGORITHM 13.3. Expectation-Maximization (EM) Algorithm

EXPECTATION-MAXIMIZATION ($\mathbf{D}, k, \epsilon$):

1 $t \leftarrow 0$
// Initialization
2 Randomly initialize $\boldsymbol{\mu}_1^t, \ldots, \boldsymbol{\mu}_k^t$
3 $\boldsymbol{\Sigma}_i^t \leftarrow \mathbf{I}$, $\forall i = 1, \ldots, k$
4 $P^t(C_i) \leftarrow \frac{1}{k}$, $\forall i = 1, \ldots, k$
5 **repeat**
6 $t \leftarrow t + 1$
// Expectation Step
7 **for** $i = 1, \ldots, k$ *and* $j = 1, \ldots, n$ **do**
8 $w_{ij} \leftarrow \frac{f(\mathbf{x}_j | \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \cdot P(C_i)}{\sum_{a=1}^{k} f(\mathbf{x}_j | \boldsymbol{\mu}_a, \boldsymbol{\Sigma}_a) \cdot P(C_a)}$ // posterior probability $P^t(C_i | \mathbf{x}_j)$
// Maximization Step
9 **for** $i = 1, \ldots, k$ **do**
10 $\boldsymbol{\mu}_i^t \leftarrow \frac{\sum_{j=1}^{n} w_{ij} \cdot \mathbf{x}_j}{\sum_{j=1}^{n} w_{ij}}$ // re-estimate mean
11 $\boldsymbol{\Sigma}_i^t \leftarrow \frac{\sum_{j=1}^{n} w_{ij} (\mathbf{x}_j - \boldsymbol{\mu}_i)(\mathbf{x}_j - \boldsymbol{\mu}_i)^T}{\sum_{j=1}^{n} w_{ij}}$ // re-estimate covariance matrix
12 $P^t(C_i) \leftarrow \frac{\sum_{j=1}^{n} w_{ij}}{n}$ // re-estimate priors
13 **until** $\sum_{i=1}^{k} \left\| \boldsymbol{\mu}_i^t - \boldsymbol{\mu}_i^{t-1} \right\|^2 \leq \epsilon$

Example 13.5 (EM in 2D). Figure 13.5 illustrates the EM algorithm for the two-dimensional Iris dataset, where the two attributes are its first two principal components. The dataset consists of $n = 150$ points, and EM was run using $k = 3$, with full covariance matrix for each cluster. The initial cluster parameters are $\mathbf{\Sigma}_i = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ and $P(C_i) = 1/3$, with the means chosen as

$$\boldsymbol{\mu}_1 = (-3.59, 0.25)^T \qquad \boldsymbol{\mu}_2 = (-1.09, -0.46)^T \qquad \boldsymbol{\mu}_3 = (0.75, 1.07)^T$$

The cluster means (shown in black) and the joint probability density function are shown in Figure 13.5a.

The EM algorithm took 36 iterations to converge (using $\epsilon = 0.001$). An intermediate stage of the clustering is shown in Figure 13.5b, for $t = 1$. Finally at iteration $t = 36$, shown in Figure 13.5c, the three clusters have been correctly identified, with the following parameters:

$$\boldsymbol{\mu}_1 = (-2.02, 0.017)^T \qquad \boldsymbol{\mu}_2 = (-0.51, -0.23)^T \qquad \boldsymbol{\mu}_3 = (2.64, 0.19)^T$$

$$\mathbf{\Sigma}_1 = \begin{pmatrix} 0.56 & -0.29 \\ -0.29 & 0.23 \end{pmatrix} \qquad \mathbf{\Sigma}_2 = \begin{pmatrix} 0.36 & -0.22 \\ -0.22 & 0.19 \end{pmatrix} \qquad \mathbf{\Sigma}_3 = \begin{pmatrix} 0.05 & -0.06 \\ -0.06 & 0.21 \end{pmatrix}$$

$$P(C_1) = 0.36 \qquad P(C_2) = 0.31 \qquad P(C_3) = 0.33$$

To see the effect of a full versus diagonal covariance matrix, we ran the EM algorithm on the Iris principal components dataset under the independence assumption, which took $t = 29$ iterations to converge. The final cluster parameters were

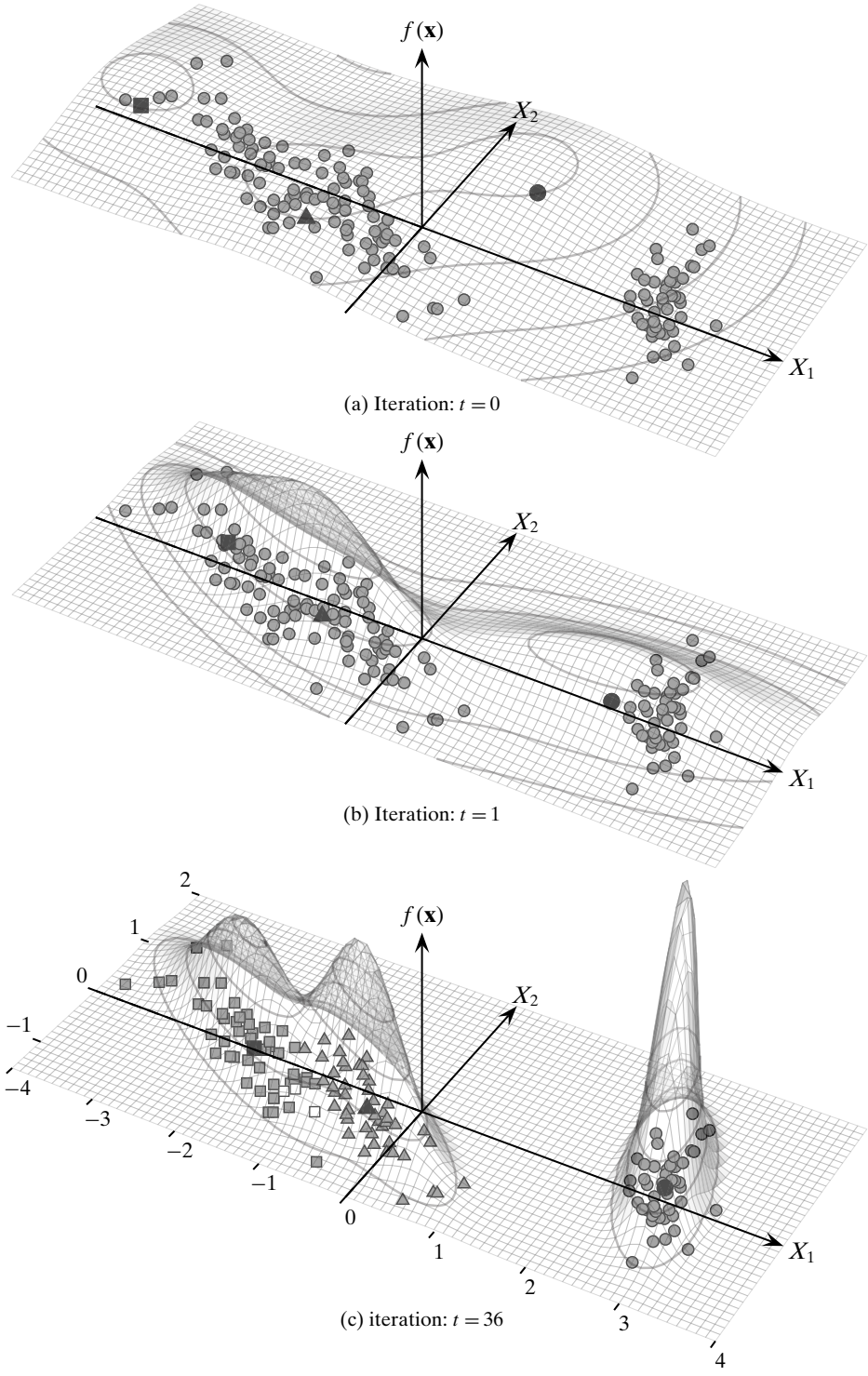
$$\boldsymbol{\mu}_1 = (-2.1, 0.28)^T \qquad \boldsymbol{\mu}_2 = (-0.67, -0.40)^T \qquad \boldsymbol{\mu}_3 = (2.64, 0.19)^T$$

$$\mathbf{\Sigma}_1 = \begin{pmatrix} 0.59 & 0 \\ 0 & 0.11 \end{pmatrix} \qquad \mathbf{\Sigma}_2 = \begin{pmatrix} 0.49 & 0 \\ 0 & 0.11 \end{pmatrix} \qquad \mathbf{\Sigma}_3 = \begin{pmatrix} 0.05 & 0 \\ 0 & 0.21 \end{pmatrix}$$

$$P(C_1) = 0.30 \qquad P(C_2) = 0.37 \qquad P(C_3) = 0.33$$

Figure 13.6b shows the clustering results. Also shown are the contours of the normal density function for each cluster (plotted so that the contours do not intersect). The results for the full covariance matrix are shown in Figure 13.6a, which is a projection of Figure 13.5c onto the 2D plane. Points in C_1 are shown as squares, in C_2 as triangles, and in C_3 as circles.

One can observe that the diagonal assumption leads to axis parallel contours for the normal density, contrasted with the rotated contours for the full covariance matrix. The full matrix yields much better clustering, which can be observed by considering the number of points grouped with the wrong Iris type (the white points). For the full covariance matrix only three points are in the wrong group, whereas for the diagonal covariance matrix 25 points are in the wrong cluster, 15 from `iris-virginica` (white triangles) and 10 from `iris-versicolor` (white squares). The points corresponding to `iris-setosa` are correctly clustered as C_3 in both approaches.

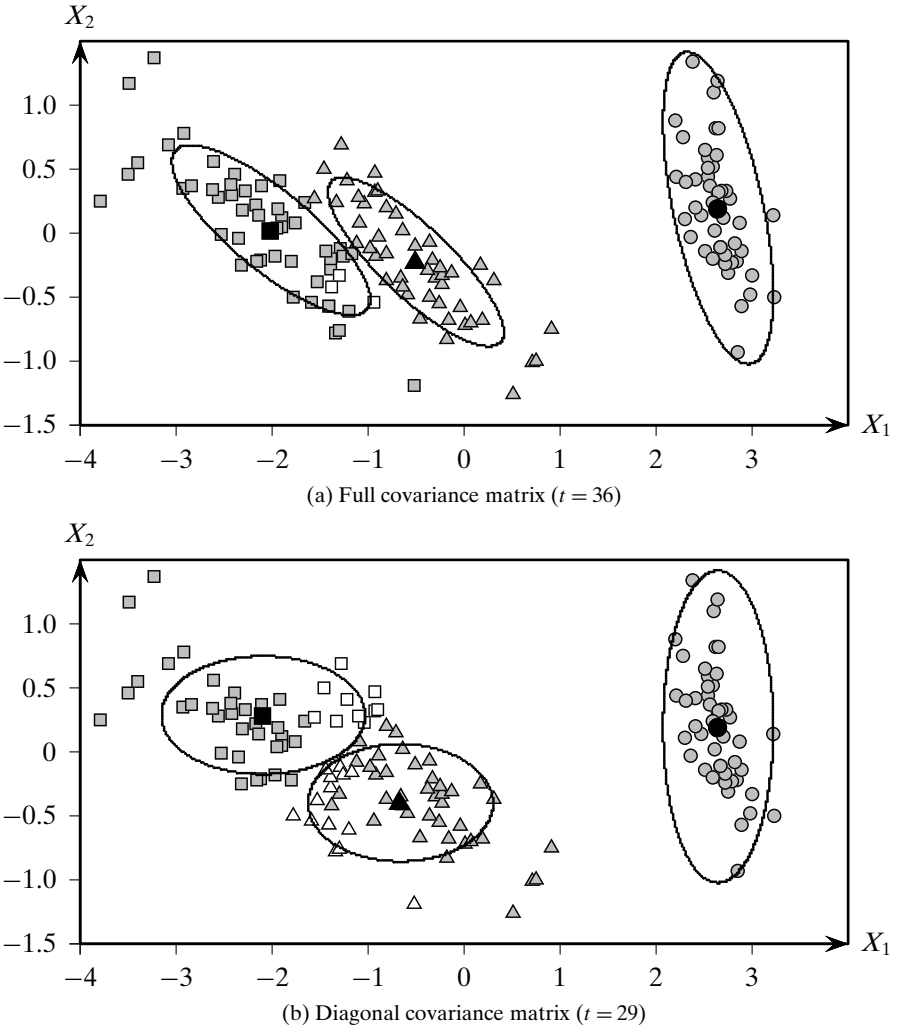


(a) Iteration: $t = 0$

(b) Iteration: $t = 1$

(c) iteration: $t = 36$

Figure 13.5. EM algorithm in two dimensions: mixture of $k = 3$ Gaussians.



(a) Full covariance matrix ($t = 36$)

(b) Diagonal covariance matrix ($t = 29$)

Figure 13.6. Iris principal components dataset: full versus diagonal covariance matrix.

Computational Complexity

For the expectation step, to compute the cluster posterior probabilities, we need to invert $\mathbf{\Sigma}_i$ and compute its determinant $|\mathbf{\Sigma}_i|$, which takes $O(d^3)$ time. Across the k clusters the time is $O(kd^3)$. For the expectation step, evaluating the density $f(\mathbf{x}_j|\boldsymbol{\mu}_i, \mathbf{\Sigma}_i)$ takes $O(d^2)$ time, for a total time of $O(knd^2)$ over the n points and k clusters. For the maximization step, the time is dominated by the update for $\mathbf{\Sigma}_i$, which takes $O(knd^2)$ time over all k clusters. The computational complexity of the EM method is thus $O(t(kd^3 + nkd^2))$, where t is the number of iterations. If we use a diagonal covariance matrix, then inverse and determinant of $\mathbf{\Sigma}_i$ can be computed in $O(d)$ time. Density computation per point takes $O(d)$ time, so that the time for the expectation step is $O(knd)$. The maximization step also takes $O(knd)$ time to re-estimate $\mathbf{\Sigma}_i$. The total time for a diagonal covariance matrix is therefore $O(tnkd)$. The I/O complexity for the

EM algorithm is $O(t)$ complete database scans because we read the entire set of points in each iteration.

K-means as Specialization of EM

Although we assumed a normal mixture model for the clusters, the EM approach can be applied with other models for the cluster density distribution $P(\mathbf{x}_j|C_i)$. For instance, K-means can be considered as a special case of the EM algorithm, obtained as follows:

$$P(\mathbf{x}_j|C_i) = \begin{cases} 1 & \text{if } C_i = \arg\min_{C_a} \left\{ \left\| \mathbf{x}_j - \boldsymbol{\mu}_a \right\|^2 \right\} \\ 0 & \text{otherwise} \end{cases}$$

Using Eq. (13.9), the posterior probability $P(C_i|\mathbf{x}_j)$ is given as

$$P(C_i|\mathbf{x}_j) = \frac{P(\mathbf{x}_j|C_i)P(C_i)}{\sum_{a=1}^{k} P(\mathbf{x}_j|C_a)P(C_a)}$$

One can see that if $P(\mathbf{x}_j|C_i) = 0$, then $P(C_i|\mathbf{x}_j) = 0$. Otherwise, if $P(\mathbf{x}_j|C_i) = 1$, then $P(\mathbf{x}_j|C_a) = 0$ for all $a \neq i$, and thus $P(C_i|\mathbf{x}_j) = \frac{1 \cdot P(C_i)}{1 \cdot P(C_i)} = 1$. Putting it all together, the posterior probability is given as

$$P(C_i|\mathbf{x}_j) = \begin{cases} 1 & \text{if } \mathbf{x}_j \in C_i, \text{i.e., if } C_i = \arg\min_{C_a} \left\{ \left\| \mathbf{x}_j - \boldsymbol{\mu}_a \right\|^2 \right\} \\ 0 & \text{otherwise} \end{cases}$$

It is clear that for K-means the cluster parameters are $\boldsymbol{\mu}_i$ and $P(C_i)$; we can ignore the covariance matrix.

13.3.3 Maximum Likelihood Estimation

In this section, we derive the maximum likelihood estimates for the cluster parameters $\boldsymbol{\mu}_i$, $\boldsymbol{\Sigma}_i$ and $P(C_i)$. We do this by taking the derivative of the log-likelihood function with respect to each of these parameters and setting the derivative to zero.

The partial derivative of the log-likelihood function [Eq. (13.8)] with respect to some parameter $\boldsymbol{\theta}_i$ for cluster C_i is given as

$$\begin{aligned} \frac{\partial}{\partial \boldsymbol{\theta}_i} \ln\Big(P(\mathbf{D}|\boldsymbol{\theta})\Big) &= \frac{\partial}{\partial \boldsymbol{\theta}_i}\left(\sum_{j=1}^{n} \ln f(\mathbf{x}_j)\right) \\ &= \sum_{j=1}^{n}\left(\frac{1}{f(\mathbf{x}_j)} \cdot \frac{\partial f(\mathbf{x}_j)}{\partial \boldsymbol{\theta}_i}\right) \\ &= \sum_{j=1}^{n}\left(\frac{1}{f(\mathbf{x}_j)} \sum_{a=1}^{k} \frac{\partial}{\partial \boldsymbol{\theta}_i}\Big(f(\mathbf{x}_j|\boldsymbol{\mu}_a, \boldsymbol{\Sigma}_a)P(C_a)\Big)\right) \\ &= \sum_{j=1}^{n}\left(\frac{1}{f(\mathbf{x}_j)} \cdot \frac{\partial}{\partial \boldsymbol{\theta}_i}\Big(f(\mathbf{x}_j|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)P(C_i)\Big)\right) \end{aligned}$$

The last step follows from the fact that because $\boldsymbol{\theta}_i$ is a parameter for the ith cluster the mixture components for the other clusters are constants with respect to $\boldsymbol{\theta}_i$. Using the

fact that $|\boldsymbol{\Sigma}_i| = \frac{1}{|\boldsymbol{\Sigma}_i^{-1}|}$ the multivariate normal density in Eq. (13.6) can be written as

$$f(\mathbf{x}_j|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) = (2\pi)^{-\frac{d}{2}} \, |\boldsymbol{\Sigma}_i^{-1}|^{\frac{1}{2}} \, \exp\big\{g(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)\big\} \tag{13.14}$$

where

$$g(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) = -\frac{1}{2}(\mathbf{x}_j - \boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}_i^{-1} (\mathbf{x}_j - \boldsymbol{\mu}_i) \tag{13.15}$$

Thus, the derivative of the log-likelihood function can be written as

$$\frac{\partial}{\partial \boldsymbol{\theta}_i} \ln\Big(P(\mathbf{D}|\boldsymbol{\theta})\Big) = \sum_{j=1}^{n} \left(\frac{1}{f(\mathbf{x}_j)} \cdot \frac{\partial}{\partial \boldsymbol{\theta}_i} \Big((2\pi)^{-\frac{d}{2}} \, |\boldsymbol{\Sigma}_i^{-1}|^{\frac{1}{2}} \exp\big\{g(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)\big\} P(C_i) \Big) \right) \tag{13.16}$$

Below, we make use of the fact that

$$\frac{\partial}{\partial \boldsymbol{\theta}_i} \exp\big\{g(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)\big\} = \exp\big\{g(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)\big\} \cdot \frac{\partial}{\partial \boldsymbol{\theta}_i} g(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \tag{13.17}$$

Estimation of Mean

To derive the maximum likelihood estimate for the mean $\boldsymbol{\mu}_i$, we have to take the derivative of the log-likelihood with respect to $\boldsymbol{\theta}_i = \boldsymbol{\mu}_i$. As per Eq. (13.16), the only term involving $\boldsymbol{\mu}_i$ is $\exp\big\{g(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)\big\}$. Using the fact that

$$\frac{\partial}{\partial \boldsymbol{\mu}_i} g(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) = \boldsymbol{\Sigma}_i^{-1}(\mathbf{x}_j - \boldsymbol{\mu}_i) \tag{13.18}$$

and making use of Eq. (13.17), the partial derivative of the log-likelihood [Eq. (13.16)] with respect to $\boldsymbol{\mu}_i$ is

$$\begin{aligned} \frac{\partial}{\partial \boldsymbol{\mu}_i} \ln(P(\mathbf{D}|\boldsymbol{\theta})) &= \sum_{j=1}^{n} \left(\frac{1}{f(\mathbf{x}_j)} (2\pi)^{-\frac{d}{2}} \, |\boldsymbol{\Sigma}_i^{-1}|^{\frac{1}{2}} \exp\big\{g(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)\big\} P(C_i) \; \boldsymbol{\Sigma}_i^{-1}(\mathbf{x}_j - \boldsymbol{\mu}_i) \right) \\ &= \sum_{j=1}^{n} \left(\frac{f(\mathbf{x}_j|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) P(C_i)}{f(\mathbf{x}_j)} \cdot \boldsymbol{\Sigma}_i^{-1}(\mathbf{x}_j - \boldsymbol{\mu}_i) \right) \\ &= \sum_{j=1}^{n} w_{ij} \boldsymbol{\Sigma}_i^{-1}(\mathbf{x}_j - \boldsymbol{\mu}_i) \end{aligned}$$

where we made use of Eqs. (13.14) and (13.9), and the fact that

$$w_{ij} = P(C_i|\mathbf{x}_j) = \frac{f(\mathbf{x}_j|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) P(C_i)}{f(\mathbf{x}_j)}$$

Setting the partial derivative of the log-likelihood to the zero vector, and multiplying both sides by $\boldsymbol{\Sigma}_i$, we get

$$\sum_{j=1}^{n} w_{ij}(\mathbf{x}_j - \boldsymbol{\mu}_i) = \mathbf{0}, \text{ which implies that}$$

$$\sum_{j=1}^{n} w_{ij}\mathbf{x}_j = \boldsymbol{\mu}_i \sum_{j=1} w_{ij}, \text{ and therefore}$$

$$\boldsymbol{\mu}_i = \frac{\sum_{j=1}^{n} w_{ij}\mathbf{x}_j}{\sum_{j=1}^{n} w_{ij}} \tag{13.19}$$

which is precisely the re-estimation formula we used in Eq. (13.11).

Estimation of Covariance Matrix

To re-estimate the covariance matrix $\boldsymbol{\Sigma}_i$, we take the partial derivative of Eq. (13.16) with respect to $\boldsymbol{\Sigma}_i^{-1}$ using the product rule for the differentiation of the term $|\boldsymbol{\Sigma}_i^{-1}|^{\frac{1}{2}} \exp\big\{g(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)\big\}$.

Using the fact that for any square matrix $\mathbf{A}$, we have $\frac{\partial |\mathbf{A}|}{\partial \mathbf{A}} = |\mathbf{A}| \cdot (\mathbf{A}^{-1})^T$ the derivative of $|\boldsymbol{\Sigma}_i^{-1}|^{\frac{1}{2}}$ with respect to $\boldsymbol{\Sigma}_i^{-1}$ is

$$\frac{\partial |\boldsymbol{\Sigma}_i^{-1}|^{\frac{1}{2}}}{\partial \boldsymbol{\Sigma}_i^{-1}} = \frac{1}{2} \cdot |\boldsymbol{\Sigma}_i^{-1}|^{-\frac{1}{2}} \cdot |\boldsymbol{\Sigma}_i^{-1}| \cdot \boldsymbol{\Sigma}_i = \frac{1}{2} \cdot |\boldsymbol{\Sigma}_i^{-1}|^{\frac{1}{2}} \cdot \boldsymbol{\Sigma}_i \tag{13.20}$$

Next, using the fact that for the square matrix $\mathbf{A} \in \mathbb{R}^{d \times d}$ and vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^d$, we have $\frac{\partial}{\partial \mathbf{A}} \mathbf{a}^T \mathbf{A} \mathbf{b} = \mathbf{a}\mathbf{b}^T$ the derivative of $\exp\big\{g(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)\big\}$ with respect to $\boldsymbol{\Sigma}_i^{-1}$ is obtained from Eq. (13.17) as follows:

$$\frac{\partial}{\partial \boldsymbol{\Sigma}_i^{-1}} \exp\big\{g(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)\big\} = -\frac{1}{2} \exp\big\{g(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)\big\}(\mathbf{x}_j - \boldsymbol{\mu}_i)(\mathbf{x}_j - \boldsymbol{\mu}_i)^T \tag{13.21}$$

Using the product rule on Eqs. (13.20) and (13.21), we get

$$\begin{aligned}
&\frac{\partial}{\partial \boldsymbol{\Sigma}_i^{-1}} |\boldsymbol{\Sigma}_i^{-1}|^{\frac{1}{2}} \exp\big\{g(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)\big\} \\
&\quad = \frac{1}{2} |\boldsymbol{\Sigma}_i^{-1}|^{\frac{1}{2}} \boldsymbol{\Sigma}_i \exp\big\{g(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)\big\} - \frac{1}{2} |\boldsymbol{\Sigma}_i^{-1}|^{\frac{1}{2}} \exp\big\{g(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)\big\}(\mathbf{x}_j - \boldsymbol{\mu}_i)(\mathbf{x}_j - \boldsymbol{\mu}_i)^T \\
&\quad = \frac{1}{2} \cdot |\boldsymbol{\Sigma}_i^{-1}|^{\frac{1}{2}} \cdot \exp\big\{g(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)\big\}\Big(\boldsymbol{\Sigma}_i - (\mathbf{x}_j - \boldsymbol{\mu}_i)(\mathbf{x}_j - \boldsymbol{\mu}_i)^T\Big)
\end{aligned} \tag{13.22}$$

Plugging Eq. (13.22) into Eq. (13.16) the derivative of the log-likelihood function with respect to $\boldsymbol{\Sigma}_i^{-1}$ is given as

$$\begin{aligned}
&\frac{\partial}{\partial \boldsymbol{\Sigma}_i^{-1}} \ln(P(\mathbf{D}|\boldsymbol{\theta})) \\
&\quad = \frac{1}{2} \sum_{j=1}^{n} \frac{(2\pi)^{-\frac{d}{2}} |\boldsymbol{\Sigma}_i^{-1}|^{\frac{1}{2}} \exp\big\{g(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)\big\} P(C_i)}{f(\mathbf{x}_j)} \Big(\boldsymbol{\Sigma}_i - (\mathbf{x}_j - \boldsymbol{\mu}_i)(\mathbf{x}_j - \boldsymbol{\mu}_i)^T\Big)
\end{aligned}$$

$$= \frac{1}{2}\sum_{j=1}^{n} \frac{f(\mathbf{x}_j|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)P(C_i)}{f(\mathbf{x}_j)} \cdot \Big(\boldsymbol{\Sigma}_i - (\mathbf{x}_j - \boldsymbol{\mu}_i)(\mathbf{x}_j - \boldsymbol{\mu}_i)^T\Big)$$

$$= \frac{1}{2}\sum_{j=1}^{n} w_{ij}\Big(\boldsymbol{\Sigma}_i - (\mathbf{x}_j - \boldsymbol{\mu}_i)(\mathbf{x}_j - \boldsymbol{\mu}_i)^T\Big)$$

Setting the derivative to the $d \times d$ zero matrix $\mathbf{0}_{d\times d}$, we can solve for $\boldsymbol{\Sigma}_i$:

$$\sum_{j=1}^{n} w_{ij}\Big(\boldsymbol{\Sigma}_i - (\mathbf{x}_j - \boldsymbol{\mu}_i)(\mathbf{x}_j - \boldsymbol{\mu}_i)^T\Big) = \mathbf{0}_{d\times d}, \text{which implies that}$$

$$\boldsymbol{\Sigma}_i = \frac{\sum_{j=1}^{n} w_{ij}(\mathbf{x}_j - \boldsymbol{\mu}_i)(\mathbf{x}_j - \boldsymbol{\mu}_i)^T}{\sum_{j=1}^{n} w_{ij}} \tag{13.23}$$

Thus, we can see that the maximum likelihood estimate for the covariance matrix is given as the weighted outer-product form in Eq. (13.12).

Estimating the Prior Probability: Mixture Parameters

To obtain a maximum likelihood estimate for the mixture parameters or the prior probabilities $P(C_i)$, we have to take the partial derivative of the log-likelihood [Eq. (13.16)] with respect to $P(C_i)$. However, we have to introduce a Lagrange multiplier α for the constraint that $\sum_{a=1}^{k} P(C_a) = 1$. We thus take the following derivative:

$$\frac{\partial}{\partial P(C_i)}\left(\ln(P(\mathbf{D}|\boldsymbol{\theta})) + \alpha\Big(\sum_{a=1}^{k} P(C_a) - 1\Big)\right) \tag{13.24}$$

The partial derivative of the log-likelihood in Eq. (13.16) with respect to $P(C_i)$ gives

$$\frac{\partial}{\partial P(C_i)} \ln(P(\mathbf{D}|\boldsymbol{\theta})) = \sum_{j=1}^{n} \frac{f(\mathbf{x}_j|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)}{f(\mathbf{x}_j)}$$

The derivative in Eq. (13.24) thus evaluates to

$$\left(\sum_{j=1}^{n} \frac{f(\mathbf{x}_j|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)}{f(\mathbf{x}_j)}\right) + \alpha$$

Setting the derivative to zero, and multiplying on both sides by $P(C_i)$, we get

$$\sum_{j=1}^{n} \frac{f(\mathbf{x}_j|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)P(C_i)}{f(\mathbf{x}_j)} = -\alpha P(C_i)$$

$$\sum_{j=1}^{n} w_{ij} = -\alpha P(C_i) \tag{13.25}$$

Taking the summation of Eq. (13.25) over all clusters yields

$$\sum_{i=1}^{k}\sum_{j=1}^{n} w_{ij} = -\alpha \sum_{i=1}^{k} P(C_i)$$

$$\text{or } n = -\alpha \tag{13.26}$$

The last step follows from the fact that $\sum_{i=1}^{k} w_{ij} = 1$. Plugging Eq. (13.26) into Eq. (13.25), gives us the maximum likelihood estimate for $P(C_i)$ as follows:

$$P(C_i) = \frac{\sum_{j=1}^{n} w_{ij}}{n} \tag{13.27}$$

which matches the formula in Eq. (13.13).

We can see that all three parameters $\boldsymbol{\mu}_i$, $\boldsymbol{\Sigma}_i$, and $P(C_i)$ for cluster C_i depend on the weights w_{ij}, which correspond to the cluster posterior probabilities $P(C_i|\mathbf{x}_j)$. Equations (13.19), (13.23), and (13.27) thus do not represent a closed-form solution for maximizing the log-likelihood function. Instead, we use the iterative EM approach to compute the w_{ij} in the expectation step, and we then re-estimate $\boldsymbol{\mu}_i$, $\boldsymbol{\Sigma}_i$ and $P(C_i)$ in the maximization step. Next, we describe the EM framework in some more detail.

13.3.4 EM Approach

Maximizing the log-likelihood function [Eq. (13.8)] directly is hard because the mixture term appears inside the logarithm. The problem is that for any point $\mathbf{x}_j$ we do not know which normal, or mixture component, it comes from. Suppose that we knew this information, that is, suppose each point $\mathbf{x}_j$ had an associated value indicating the cluster that generated the point. As we shall see, it is much easier to maximize the log-likelihood given this information.

The categorical attribute corresponding to the cluster label can be modeled as a vector random variable $\mathbf{C} = (C_1, C_2, \ldots, C_k)$, where C_i is a Bernoulli random variable (see Section 3.1.2 for details on how to model a categorical variable). If a given point is generated from cluster C_i, then $C_i = 1$, otherwise $C_i = 0$. The parameter $P(C_i)$ gives the probability $P(C_i = 1)$. Because each point can be generated from only one cluster, if $C_a = 1$ for a given point, then $C_i = 0$ for all $i \neq a$. It follows that $\sum_{i=1}^{k} P(C_i) = 1$.

For each point $\mathbf{x}_j$, let its cluster vector be $\mathbf{c}_j = (c_{j1}, \ldots, c_{jk})^T$. Only one component of $\mathbf{c}_j$ has value 1. If $c_{ji} = 1$, it means that $C_i = 1$, that is, the cluster C_i generates the point $\mathbf{x}_j$. The probability mass function of $\mathbf{C}$ is given as

$$P(\mathbf{C} = \mathbf{c}_j) = \prod_{i=1}^{k} P(C_i)^{c_{ji}}$$

Given the cluster information $\mathbf{c}_j$ for each point $\mathbf{x}_j$, the conditional probability density function for $\mathbf{X}$ is given as

$$f(\mathbf{x}_j|\mathbf{c}_j) = \prod_{i=1}^{k} f(\mathbf{x}_j|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)^{c_{ji}}$$

Only one cluster can generate $\mathbf{x}_j$, say C_a, in which case $c_{ja} = 1$, and the above expression would simplify to $f(\mathbf{x}_j|\mathbf{c}_j) = f(\mathbf{x}_j|\boldsymbol{\mu}_a, \boldsymbol{\Sigma}_a)$.

The pair $(\mathbf{x}_j, \mathbf{c}_j)$ is a random sample drawn from the joint distribution of vector random variables $\mathbf{X} = (X_1, \ldots, X_d)$ and $\mathbf{C} = (C_1, \ldots, C_k)$, corresponding to the d data attributes and k cluster attributes. The joint density function of $\mathbf{X}$ and $\mathbf{C}$ is given as

$$f(\mathbf{x}_j \text{ and } \mathbf{c}_j) = f(\mathbf{x}_j|\mathbf{c}_j)P(\mathbf{c}_j) = \prod_{i=1}^{k}\Big(f(\mathbf{x}_j|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)P(C_i)\Big)^{c_{ji}}$$

The log-likelihood for the data given the cluster information is as follows:

$$\begin{aligned}
\ln P(\mathbf{D}|\boldsymbol{\theta}) &= \ln \prod_{j=1}^{n} f(\mathbf{x}_j \text{ and } \mathbf{c}_j|\boldsymbol{\theta}) \\
&= \sum_{j=1}^{n} \ln f(\mathbf{x}_j \text{ and } \mathbf{c}_j|\boldsymbol{\theta}) \\
&= \sum_{j=1}^{n} \ln\Big(\prod_{i=1}^{k}\Big(f(\mathbf{x}_j|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)P(C_i)\Big)^{c_{ji}}\Big) \\
&= \sum_{j=1}^{n}\sum_{i=1}^{k} c_{ji}\Big(\ln f(\mathbf{x}_j|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) + \ln P(C_i)\Big) \qquad (13.28)
\end{aligned}$$

Expectation Step

In the expectation step, we compute the expected value of the log-likelihood for the labeled data given in Eq. (13.28). The expectation is over the missing cluster information $\mathbf{c}_j$ treating $\boldsymbol{\mu}_i$, $\boldsymbol{\Sigma}_i$, $P(C_i)$, and $\mathbf{x}_j$ as fixed. Owing to the linearity of expectation, the expected value of the log-likelihood is given as

$$E[\ln P(\mathbf{D}|\boldsymbol{\theta})] = \sum_{j=1}^{n}\sum_{i=1}^{k} E[c_{ji}]\Big(\ln f(\mathbf{x}_j|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) + \ln P(C_i)\Big)$$

The expected value $E[c_{ji}]$ can be computed as

$$\begin{aligned}
E[c_{ji}] &= 1 \cdot P(c_{ji} = 1|\mathbf{x}_j) + 0 \cdot P(c_{ji} = 0|\mathbf{x}_j) = P(c_{ji} = 1|\mathbf{x}_j) = P(C_i|\mathbf{x}_j) \\
&= \frac{P(\mathbf{x}_j|C_i)P(C_i)}{P(\mathbf{x}_j)} = \frac{f(\mathbf{x}_j|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)P(C_i)}{f(\mathbf{x}_j)} \\
&= w_{ij} \qquad (13.29)
\end{aligned}$$

Thus, in the expectation step we use the values of $\boldsymbol{\theta} = \{\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i, P(C_i)\}_{i=1}^{k}$ to estimate the posterior probabilities or weights w_{ij} for each point for each cluster. Using $E[c_{ji}] = w_{ij}$, the expected value of the log-likelihood function can be rewritten as

$$E[\ln P(\mathbf{D}|\boldsymbol{\theta})] = \sum_{j=1}^{n}\sum_{i=1}^{k} w_{ij}\Big(\ln f(\mathbf{x}_j|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) + \ln P(C_i)\Big) \qquad (13.30)$$

Maximization Step

In the maximization step, we maximize the expected value of the log-likelihood [Eq. (13.30)]. Taking the derivative with respect to $\boldsymbol{\mu}_i$, $\boldsymbol{\Sigma}_i$ or $P(C_i)$ we can ignore the terms for all the other clusters.

The derivative of Eq. (13.30) with respect to $\boldsymbol{\mu}_i$ is given as

$$\begin{aligned}
\frac{\partial}{\partial \boldsymbol{\mu}_i} \ln E[P(\mathbf{D}|\boldsymbol{\theta})] &= \frac{\partial}{\partial \boldsymbol{\mu}_i} \sum_{j=1}^{n} w_{ij} \ln f(\mathbf{x}_j|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \\
&= \sum_{j=1}^{n} w_{ij} \cdot \frac{1}{f(\mathbf{x}_j|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)} \frac{\partial}{\partial \boldsymbol{\mu}_i} f(\mathbf{x}_j|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \\
&= \sum_{j=1}^{n} w_{ij} \cdot \frac{1}{f(\mathbf{x}_j|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)} \cdot f(\mathbf{x}_j|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)\ \boldsymbol{\Sigma}_i^{-1}\ (\mathbf{x}_j - \boldsymbol{\mu}_i) \\
&= \sum_{j=1}^{n} w_{ij}\ \boldsymbol{\Sigma}_i^{-1} (\mathbf{x}_j - \boldsymbol{\mu}_i)
\end{aligned}$$

where we used the observation that

$$\frac{\partial}{\partial \boldsymbol{\mu}_i} f(\mathbf{x}_j|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) = f(\mathbf{x}_j|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)\ \boldsymbol{\Sigma}_i^{-1}\ (\mathbf{x}_j - \boldsymbol{\mu}_i)$$

which follows from Eqs. (13.14), (13.17), and (13.18). Setting the derivative of the expected value of the log-likelihood to the zero vector, and multiplying on both sides by $\boldsymbol{\Sigma}_i$, we get

$$\boldsymbol{\mu}_i = \frac{\sum_{j=1}^{n} w_{ij} \mathbf{x}_j}{\sum_{j=1}^{n} w_{ij}}$$

matching the formula in Eq. (13.11).

Making use of Eqs. (13.22) and (13.14), we obtain the derivative of Eq. (13.30) with respect to $\boldsymbol{\Sigma}_i^{-1}$ as follows:

$$\begin{aligned}
&\frac{\partial}{\partial \boldsymbol{\Sigma}_i^{-1}} \ln E[P(\mathbf{D}|\boldsymbol{\theta})] \\
&\quad = \sum_{j=1}^{n} w_{ij} \cdot \frac{1}{f(\mathbf{x}_j|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)} \cdot \frac{1}{2}\ f(\mathbf{x}_j|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)\big(\boldsymbol{\Sigma}_i - (\mathbf{x}_j - \boldsymbol{\mu}_i)(\mathbf{x}_j - \boldsymbol{\mu}_i)^T\big) \\
&\quad = \frac{1}{2} \sum_{j=1}^{n} w_{ij} \cdot \big(\boldsymbol{\Sigma}_i - (\mathbf{x}_j - \boldsymbol{\mu}_i)(\mathbf{x}_j - \boldsymbol{\mu}_i)^T\big)
\end{aligned}$$

Setting the derivative to the $d \times d$ zero matrix and solving for $\boldsymbol{\Sigma}_i$ yields

$$\boldsymbol{\Sigma}_i = \frac{\sum_{j=1}^{n} w_{ij} (\mathbf{x}_j - \boldsymbol{\mu}_i)(\mathbf{x}_j - \boldsymbol{\mu}_i)^T}{\sum_{j=1}^{n} w_{ij}}$$

which is the same as that in Eq. (13.12).

Using the Lagrange multiplier α for the constraint $\sum_{i=1}^{k} P(C_i) = 1$, and noting that in the log-likelihood function [Eq. (13.30)], the term $\ln f(\mathbf{x}_j|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$ is a constant with respect to $P(C_i)$, we obtain the following:

$$\frac{\partial}{\partial P(C_i)}\left(\ln E[P(\mathbf{D}|\boldsymbol{\theta})] + \alpha\left(\sum_{i=1}^{k} P(C_i) - 1\right)\right) = \frac{\partial}{\partial P(C_i)}\Big(w_{ij} \ln P(C_i) + \alpha P(C_i)\Big)$$

$$= \left(\sum_{j=1}^{n} w_{ij} \cdot \frac{1}{P(C_i)}\right) + \alpha$$

Setting the derivative to zero, we get

$$\sum_{j=1}^{n} w_{ij} = -\alpha \cdot P(C_i)$$

Using the same derivation as in Eq. (13.26) we obtain

$$P(C_i) = \frac{\sum_{j=1}^{n} w_{ij}}{n}$$

which is identical to the re-estimation formula in Eq. (13.13).

13.4 FURTHER READING

The K-means algorithm was proposed in several contexts during the 1950s and 1960s; among the first works to develop the method include MacQueen (1967); Lloyd (1982) and Hartigan (1975). Kernel k-means was first proposed in Schölkopf, Smola, and Müller (1996). The EM algorithm was proposed in Dempster, Laird, and Rubin (1977). A good review on EM method can be found in McLachlan and Krishnan (2008). For a scalable and incremental representative-based clustering method that can also generate hierarchical clusterings see Zhang, Ramakrishnan, and Livny (1996).

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). "Maximum likelihood from incomplete data via the EM algorithm." *Journal of the Royal Statistical Society, Series B*, 39 (1): 1–38.

Hartigan, J. A. (1975). *Clustering Algorithms*. New York: John Wiley & Sons.

Lloyd, S. (1982). "Least squares quantization in PCM." *IEEE Transactions on Information Theory*, 28 (2): 129–137.

MacQueen, J. (1967). "Some methods for classification and analysis of multivariate observations." *In Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1, pp. 281–297, University of California Press, Berkeley.

McLachlan, G. and Krishnan, T. (2008). *The EM Algorithm and Extensions,* 2nd ed. Hoboken, NJ: John Wiley & Sons.

Schölkopf, B., Smola, A., and Müller, K.-R. (1996). *Nonlinear component analysis as a kernel eigenvalue problem*. Technical Report No. 44. Tübingen, Germany: Max-Planck-Institut für biologische Kybernetik.

Zhang, T., Ramakrishnan, R., and Livny, M. (1996). "BIRCH: an efficient data clustering method for very large databases." *ACM SIGMOD Record*, 25 (2): 103–114.

13.5 EXERCISES

Q1. Given the following points: $2, 4, 10, 12, 3, 20, 30, 11, 25$. Assume $k = 3$, and that we randomly pick the initial means $\mu_1 = 2$, $\mu_2 = 4$ and $\mu_3 = 6$. Show the clusters obtained using K-means algorithm after one iteration, and show the new means for the next iteration.

Table 13.1. Dataset for Q2

x	$P(C_1\|x)$	$P(C_2\|x)$
2	0.9	0.1
3	0.8	0.1
7	0.3	0.7
9	0.1	0.9
2	0.9	0.1
1	0.8	0.2

Q2. Given the data points in Table 13.1, and their probability of belonging to two clusters. Assume that these points were produced by a mixture of two univariate normal distributions. Answer the following questions:

(a) Find the maximum likelihood estimate of the means μ_1 and μ_2.

(b) Assume that $\mu_1 = 2$, $\mu_2 = 7$, and $\sigma_1 = \sigma_2 = 1$. Find the probability that the point $x = 5$ belongs to cluster C_1 and to cluster C_2. You may assume that the prior probability of each cluster is equal (i.e., $P(C_1) = P(C_2) = 0.5$), and the prior probability $P(x = 5) = 0.029$.

Table 13.2. Dataset for Q3

	X_1	X_2
$\mathbf{x}_1$	0	2
$\mathbf{x}_2$	0	0
$\mathbf{x}_3$	1.5	0
$\mathbf{x}_4$	5	0
$\mathbf{x}_5$	5	2

Q3. Given the two-dimensional points in Table 13.2, assume that $k = 2$, and that initially the points are assigned to clusters as follows: $C_1 = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_4\}$ and $C_2 = \{\mathbf{x}_3, \mathbf{x}_5\}$. Answer the following questions:

(a) Apply the K-means algorithm until convergence, that is, the clusters do not change, assuming (1) the usual Euclidean distance or the L_2-*norm* as the distance

between points, defined as $\|\mathbf{x}_i - \mathbf{x}_j\|_2 = \left(\sum_{a=1}^{d}(x_{ia} - x_{ja})^2\right)^{1/2}$, and (2) the Manhattan distance or the L_1-*norm* defined as $\|\mathbf{x}_i - \mathbf{x}_j\|_1 = \sum_{a=1}^{d}|x_{ia} - x_{ja}|$.

(b) Apply the EM algorithm with $k = 2$ assuming that the dimensions are independent. Show one complete execution of the expectation and the maximization steps. Start with the assumption that $P(C_i|x_{ja}) = 0.5$ for $a = 1, 2$ and $j = 1, \ldots, 5$.

Q4. Given the categorical database in Table 13.3. Find $k = 2$ clusters in this data using the EM method. Assume that each attribute is independent, and that the domain of each attribute is $\{A, C, T\}$. Initially assume that the points are partitioned as follows: $C_1 = \{\mathbf{x}_1, \mathbf{x}_4\}$, and $C_2 = \{\mathbf{x}_2, \mathbf{x}_3\}$. Assume that $P(C_1) = P(C_2) = 0.5$.

Table 13.3. Dataset for Q4

	X_1	X_2
$\mathbf{x}_1$	A	T
$\mathbf{x}_2$	A	A
$\mathbf{x}_3$	C	C
$\mathbf{x}_4$	A	C

The probability of an attribute value given a cluster is given as

$$P(x_{ja}|C_i) = \frac{\text{No. of times the symbol } x_{ja} \text{ occurs in cluster } C_i}{\text{No. of objects in cluster } C_i}$$

for $a = 1, 2$. The probability of a point given a cluster is then given as

$$P(\mathbf{x}_j|C_i) = \prod_{a=1}^{2} P(x_{ja}|C_i)$$

Instead of computing the mean for each cluster, generate a partition of the objects by doing a hard assignment. That is, in the expectation step compute $P(C_i|\mathbf{x}_j)$, and in the maximization step assign the point $\mathbf{x}_j$ to the cluster with the largest $P(C_i|\mathbf{x}_j)$ value, which gives a new partitioning of the points. Show one full iteration of the EM algorithm and show the resulting clusters.

Table 13.4. Dataset for Q5

	X_1	X_2	X_3
$\mathbf{x}_1$	0.5	4.5	2.5
$\mathbf{x}_2$	2.2	1.5	0.1
$\mathbf{x}_3$	3.9	3.5	1.1
$\mathbf{x}_4$	2.1	1.9	4.9
$\mathbf{x}_5$	0.5	3.2	1.2
$\mathbf{x}_6$	0.8	4.3	2.6
$\mathbf{x}_7$	2.7	1.1	3.1
$\mathbf{x}_8$	2.5	3.5	2.8
$\mathbf{x}_9$	2.8	3.9	1.5
$\mathbf{x}_{10}$	0.1	4.1	2.9

Q5. Given the points in Table 13.4, assume that there are two clusters: C_1 and C_2, with $\boldsymbol{\mu}_1 = (0.5, 4.5, 2.5)^T$ and $\boldsymbol{\mu}_2 = (2.5, 2, 1.5)^T$. Initially assign each point to the closest mean, and compute the covariance matrices $\boldsymbol{\Sigma}_i$ and the prior probabilities $P(C_i)$ for $i = 1, 2$. Next, answer which cluster is more likely to have produced $\mathbf{x}_8$?

Q6. Consider the data in Table 13.5. Answer the following questions:

(a) Compute the kernel matrix $\mathbf{K}$ between the points assuming the following kernel:

$$K(\mathbf{x}_i, \mathbf{x}_j) = 1 + \mathbf{x}_i^T \mathbf{x}_j$$

(b) Assume initial cluster assignments of $C_1 = \{\mathbf{x}_1, \mathbf{x}_2\}$ and $C_2 = \{\mathbf{x}_3, \mathbf{x}_4\}$. Using kernel K-means, which cluster should $\mathbf{x}_1$ belong to in the next step?

Table 13.5. Data for Q6

	X_1	X_2	X_3
$\mathbf{x}_1$	0.4	0.9	0.6
$\mathbf{x}_2$	0.5	0.1	0.6
$\mathbf{x}_3$	0.6	0.3	0.6
$\mathbf{x}_4$	0.4	0.8	0.5

Q7. Prove the following equivalence for the multivariate normal density function:

$$\frac{\partial}{\partial \boldsymbol{\mu}_i} f(\mathbf{x}_j | \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) = f(\mathbf{x}_j | \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)\, \boldsymbol{\Sigma}_i^{-1} (\mathbf{x}_j - \boldsymbol{\mu}_i)$$

CHAPTER 14 Hierarchical Clustering

Given n points in a d-dimensional space, the goal of hierarchical clustering is to create a sequence of nested partitions, which can be conveniently visualized via a tree or hierarchy of clusters, also called the cluster *dendrogram*. The clusters in the hierarchy range from the fine-grained to the coarse-grained – the lowest level of the tree (the leaves) consists of each point in its own cluster, whereas the highest level (the root) consists of all points in one cluster. Both of these may be considered to be *trivial* clusterings. At some intermediate level, we may find meaningful clusters. If the user supplies k, the desired number of clusters, we can choose the level at which there are k clusters.

There are two main algorithmic approaches to mine hierarchical clusters: agglomerative and divisive. Agglomerative strategies work in a bottom-up manner. That is, starting with each of the n points in a separate cluster, they repeatedly merge the most similar pair of clusters until all points are members of the same cluster. Divisive strategies do just the opposite, working in a top-down manner. Starting with all the points in the same cluster, they recursively split the clusters until all points are in separate clusters. In this chapter we focus on agglomerative strategies. We discuss some divisive strategies in Chapter 16, in the context of graph partitioning.

14.1 PRELIMINARIES

Given a dataset $\mathbf{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, where $\mathbf{x}_i \in \mathbb{R}^d$, a clustering $\mathcal{C} = \{C_1, \ldots, C_k\}$ is a partition of $\mathbf{D}$, that is, each cluster is a set of points $C_i \subseteq \mathbf{D}$, such that the clusters are pairwise disjoint $C_i \cap C_j = \emptyset$ (for all $i \neq j$), and $\cup_{i=1}^{k} C_i = \mathbf{D}$. A clustering $\mathcal{A} = \{A_1, \ldots, A_r\}$ is said to be nested in another clustering $\mathcal{B} = \{B_1, \ldots, B_s\}$ if and only if $r > s$, and for each cluster $A_i \in \mathcal{A}$, there exists a cluster $B_j \in \mathcal{B}$, such that $A_i \subseteq B_j$. Hierarchical clustering yields a sequence of n nested partitions $\mathcal{C}_1, \ldots, \mathcal{C}_n$, ranging from the trivial clustering $\mathcal{C}_1 = \big\{\{\mathbf{x}_1\}, \ldots, \{\mathbf{x}_n\}\big\}$ where each point is in a separate cluster, to the other trivial clustering $\mathcal{C}_n = \big\{\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}\big\}$, where all points are in one cluster. In general, the clustering $\mathcal{C}_{t-1}$ is nested in the clustering $\mathcal{C}_t$. The cluster dendrogram is a rooted binary tree that captures this nesting structure, with edges between cluster $C_i \in \mathcal{C}_{t-1}$ and cluster $C_j \in \mathcal{C}_t$ if C_i is nested in C_j, that is, if $C_i \subset C_j$. In this way the dendrogram captures the entire sequence of nested clusterings.

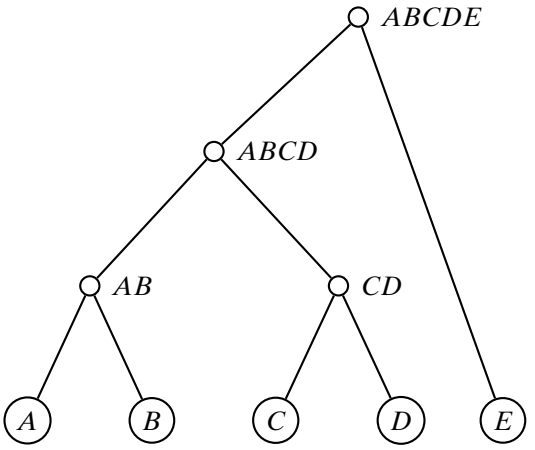


Figure 14.1. Hierarchical clustering dendrogram.

Example 14.1. Figure 14.1 shows an example of hierarchical clustering of five labeled points: A, B, C, D, and E. The dendrogram represents the following sequence of nested partitions:

Clustering	Clusters
$\mathcal{C}_1$	$\{A\}, \{B\}, \{C\}, \{D\}, \{E\}$
$\mathcal{C}_2$	$\{AB\}, \{C\}, \{D\}, \{E\}$
$\mathcal{C}_3$	$\{AB\}, \{CD\}, \{E\}$
$\mathcal{C}_4$	$\{ABCD\}, \{E\}$
$\mathcal{C}_5$	$\{ABCDE\}$

with $\mathcal{C}_{t-1} \subset \mathcal{C}_t$ for $t = 2, \ldots, 5$. We assume that A and B are merged before C and D.

Number of Hierarchical Clusterings

The number of different nested or hierarchical clusterings corresponds to the number of different binary rooted trees or dendrograms with n leaves with distinct labels. Any tree with t nodes has $t-1$ edges. Also, any rooted binary tree with m leaves has $m-1$ internal nodes. Thus, a dendrogram with m leaf nodes has a total of $t = m + m - 1 = 2m - 1$ nodes, and consequently $t - 1 = 2m - 2$ edges. To count the number of different dendrogram topologies, let us consider how we can extend a dendrogram with m leaves by adding an extra leaf, to yield a dendrogram with $m+1$ leaves. Note that we can add the extra leaf by splitting (i.e., branching from) any of the $2m-2$ edges. Further, we can also add the new leaf as a child of a new root, giving $2m - 2 + 1 = 2m - 1$ new dendrograms with $m+1$ leaves. The total number of different dendrograms with n leaves is thus obtained by the following product:

$$\prod_{m=1}^{n-1}(2m-1) = 1 \times 3 \times 5 \times 7 \times \cdots \times (2n-3) = (2n-3)!! \tag{14.1}$$

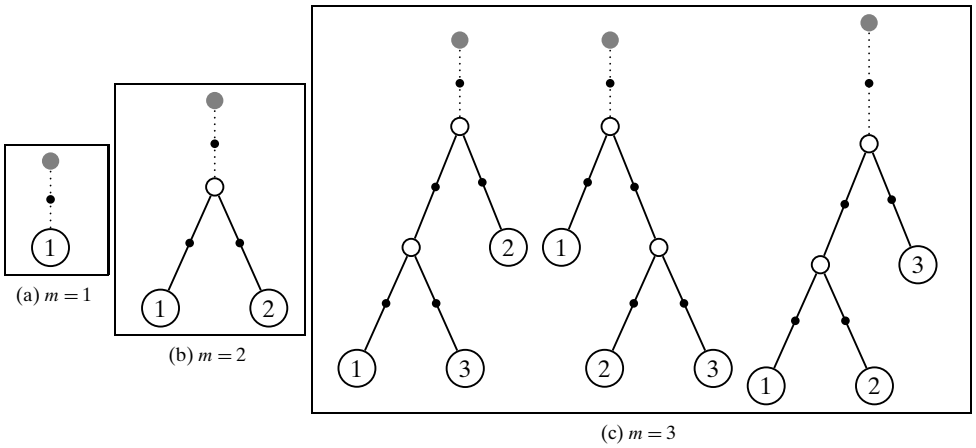


Figure 14.2. Number of hierarchical clusterings.

The index m in Eq. (14.1) goes up to $n-1$ because the last term in the product denotes the number of dendrograms one obtains when we extend a dendrogram with $n-1$ leaves by adding one more leaf, to yield dendrograms with n leaves.

The number of possible hierarchical clusterings is thus given as $(2n-3)!!$, which grows extremely rapidly. It is obvious that a naive approach of enumerating all possible hierarchical clusterings is simply infeasible.

Example 14.2. Figure 14.2 shows the number of trees with one, two, and three leaves. The gray nodes are the virtual roots, and the black dots indicate locations where a new leaf can be added. There is only one tree possible with a single leaf, as shown in Figure 14.2a. It can be extended in only one way to yield the unique tree with two leaves in Figure 14.2b. However, this tree has three possible locations where the third leaf can be added. Each of these cases is shown in Figure 14.2c. We can further see that each of the trees with $m = 3$ leaves has five locations where the fourth leaf can be added, and so on, which confirms the equation for the number of hierarchical clusterings in Eq. (14.1).

14.2 AGGLOMERATIVE HIERARCHICAL CLUSTERING

In agglomerative hierarchical clustering, we begin with each of the n points in a separate cluster. We repeatedly merge the two closest clusters until all points are members of the same cluster, as shown in the pseudo-code given in Algorithm 14.1. Formally, given a set of clusters $\mathcal{C} = \{C_1, C_2, .., C_m\}$, we find the *closest* pair of clusters C_i and C_j and merge them into a new cluster $C_{ij} = C_i \cup C_j$. Next, we update the set of clusters by removing C_i and C_j and adding C_{ij}, as follows $\mathcal{C} = \big(\mathcal{C} \setminus \{C_i, C_j\}\big) \cup \{C_{ij}\}$. We repeat the process until $\mathcal{C}$ contains only one cluster. Because the number of clusters decreases by one in each step, this process results in a sequence of n nested clusterings. If specified, we can stop the merging process when there are exactly k clusters remaining.

ALGORITHM 14.1. Agglomerative Hierarchical Clustering Algorithm

```
AGGLOMERATIVECLUSTERING(D, k):
C ← {C_i = {x_i} | x_i ∈ D} // Each point in separate cluster
Δ ← {δ(x_i, x_j): x_i, x_j ∈ D} // Compute distance matrix
repeat
    Find the closest pair of clusters C_i, C_j ∈ C
    C_ij ← C_i ∪ C_j // Merge the clusters
    C ← (C \ {C_i, C_j}) ∪ {C_ij} // Update the clustering
    Update distance matrix Δ to reflect new clustering
until |C| = k
```

14.2.1 Distance between Clusters

The main step in the algorithm is to determine the closest pair of clusters. Several distance measures, such as single link, complete link, group average, and others discussed in the following paragraphs, can be used to compute the distance between any two clusters. The between-cluster distances are ultimately based on the distance between two points, which is typically computed using the Euclidean distance or L_2-*norm*, defined as

$$\delta(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_2 = \Big(\sum_{i=1}^{d} (x_i - y_i)^2\Big)^{1/2}$$

However, one may use other distance metrics, or if available one may a user-specified distance matrix.

Single Link

Given two clusters C_i and C_j, the distance between them, denoted $\delta(C_i, C_j)$, is defined as the minimum distance between a point in C_i and a point in C_j

$$\delta(C_i, C_j) = \min\{\delta(\mathbf{x}, \mathbf{y}) \mid \mathbf{x} \in C_i, \mathbf{y} \in C_j\}$$

The name *single link* comes from the observation that if we choose the minimum distance between points in the two clusters and connect those points, then (typically) only a single link would exist between those clusters because all other pairs of points would be farther away.

Complete Link

The distance between two clusters is defined as the maximum distance between a point in C_i and a point in C_j:

$$\delta(C_i, C_j) = \max\{\delta(\mathbf{x}, \mathbf{y}) \mid \mathbf{x} \in C_i, \mathbf{y} \in C_j\}$$

The name *complete link* conveys the fact that if we connect all pairs of points from the two clusters with distance at most $\delta(C_i, C_j)$, then all possible pairs would be connected, that is, we get a complete linkage.

Group Average

The distance between two clusters is defined as the average pairwise distance between points in C_i and C_j:

$$\delta(C_i, C_j) = \frac{\sum_{\mathbf{x}\in C_i}\sum_{\mathbf{y}\in C_j}\delta(\mathbf{x},\mathbf{y})}{n_i \cdot n_j}$$

where $n_i = |C_i|$ denotes the number of points in cluster C_i.

Mean Distance

The distance between two clusters is defined as the distance between the means or centroids of the two clusters:

$$\delta(C_i, C_j) = \delta(\boldsymbol{\mu}_i, \boldsymbol{\mu}_j) \tag{14.2}$$

where $\boldsymbol{\mu}_i = \frac{1}{n_i}\sum_{\mathbf{x}\in C_i}\mathbf{x}$.

Minimum Variance: Ward's Method

The distance between two clusters is defined as the increase in the sum of squared errors (SSE) when the two clusters are merged. The SSE for a given cluster C_i is given as

$$SSE_i = \sum_{\mathbf{x}\in C_i}\|\mathbf{x}-\boldsymbol{\mu}_i\|^2$$

which can also be written as

$$\begin{aligned} SSE_i &= \sum_{\mathbf{x}\in C_i}\|\mathbf{x}-\boldsymbol{\mu}_i\|^2 \\ &= \sum_{\mathbf{x}\in C_i}\mathbf{x}^T\mathbf{x} - 2\sum_{\mathbf{x}\in C_i}\mathbf{x}^T\boldsymbol{\mu}_i + \sum_{\mathbf{x}\in C_i}\boldsymbol{\mu}_i^T\boldsymbol{\mu}_i \\ &= \left(\sum_{\mathbf{x}\in C_i}\mathbf{x}^T\mathbf{x}\right) - n_i\boldsymbol{\mu}_i^T\boldsymbol{\mu}_i \end{aligned} \tag{14.3}$$

The SSE for a clustering $\mathcal{C} = \{C_1, \ldots, C_m\}$ is given as

$$SSE = \sum_{i=1}^{m} SSE_i = \sum_{i=1}^{m}\sum_{\mathbf{x}\in C_i}\|\mathbf{x}-\boldsymbol{\mu}_i\|^2$$

Ward's measure defines the distance between two clusters C_i and C_j as the net change in the SSE value when we merge C_i and C_j into C_{ij}, given as

$$\delta(C_i, C_j) = \Delta SSE_{ij} = SSE_{ij} - SSE_i - SSE_j \tag{14.4}$$

We can obtain a simpler expression for the Ward's measure by plugging Eq. (14.3) into Eq. (14.4), and noting that because $C_{ij} = C_i \cup C_j$ and $C_i \cap C_j = \emptyset$, we

have $|C_{ij}| = n_{ij} = n_i + n_j$, and therefore

$$\begin{aligned}
\delta(C_i, C_j) &= \Delta SSE_{ij} \\
&= \sum_{\mathbf{z} \in C_{ij}} \left\| \mathbf{z} - \boldsymbol{\mu}_{ij} \right\|^2 - \sum_{\mathbf{x} \in C_i} \left\| \mathbf{x} - \boldsymbol{\mu}_i \right\|^2 - \sum_{\mathbf{y} \in C_j} \left\| \mathbf{y} - \boldsymbol{\mu}_j \right\|^2 \\
&= \sum_{\mathbf{z} \in C_{ij}} \mathbf{z}^T \mathbf{z} - n_{ij} \boldsymbol{\mu}_{ij}^T \boldsymbol{\mu}_{ij} - \sum_{\mathbf{x} \in C_i} \mathbf{x}^T \mathbf{x} + n_i \boldsymbol{\mu}_i^T \boldsymbol{\mu}_i - \sum_{\mathbf{y} \in C_j} \mathbf{y}^T \mathbf{y} + n_j \boldsymbol{\mu}_j^T \boldsymbol{\mu}_j \\
&= n_i \boldsymbol{\mu}_i^T \boldsymbol{\mu}_i + n_j \boldsymbol{\mu}_j^T \boldsymbol{\mu}_j - (n_i + n_j) \boldsymbol{\mu}_{ij}^T \boldsymbol{\mu}_{ij} \qquad (14.5)
\end{aligned}$$

The last step follows from the fact that $\sum_{\mathbf{z} \in C_{ij}} \mathbf{z}^T \mathbf{z} = \sum_{\mathbf{x} \in C_i} \mathbf{x}^T \mathbf{x} + \sum_{\mathbf{y} \in C_j} \mathbf{y}^T \mathbf{y}$. Noting that

$$\boldsymbol{\mu}_{ij} = \frac{n_i \boldsymbol{\mu}_i + n_j \boldsymbol{\mu}_j}{n_i + n_j}$$

we obtain

$$\boldsymbol{\mu}_{ij}^T \boldsymbol{\mu}_{ij} = \frac{1}{(n_i + n_j)^2} \left(n_i^2 \boldsymbol{\mu}_i^T \boldsymbol{\mu}_i + 2 n_i n_j \boldsymbol{\mu}_i^T \boldsymbol{\mu}_j + n_j^2 \boldsymbol{\mu}_j^T \boldsymbol{\mu}_j \right)$$

Plugging the above into Eq. (14.5), we finally obtain

$$\begin{aligned}
\delta(C_i, C_j) &= \Delta SSE_{ij} \\
&= n_i \boldsymbol{\mu}_i^T \boldsymbol{\mu}_i + n_j \boldsymbol{\mu}_j^T \boldsymbol{\mu}_j - \frac{1}{(n_i + n_j)} \left(n_i^2 \boldsymbol{\mu}_i^T \boldsymbol{\mu}_i + 2 n_i n_j \boldsymbol{\mu}_i^T \boldsymbol{\mu}_j + n_j^2 \boldsymbol{\mu}_j^T \boldsymbol{\mu}_j \right) \\
&= \frac{n_i (n_i + n_j) \boldsymbol{\mu}_i^T \boldsymbol{\mu}_i + n_j (n_i + n_j) \boldsymbol{\mu}_j^T \boldsymbol{\mu}_j - n_i^2 \boldsymbol{\mu}_i^T \boldsymbol{\mu}_i - 2 n_i n_j \boldsymbol{\mu}_i^T \boldsymbol{\mu}_j - n_j^2 \boldsymbol{\mu}_j^T \boldsymbol{\mu}_j}{n_i + n_j} \\
&= \frac{n_i n_j \left(\boldsymbol{\mu}_i^T \boldsymbol{\mu}_i - 2 \boldsymbol{\mu}_i^T \boldsymbol{\mu}_j + \boldsymbol{\mu}_j^T \boldsymbol{\mu}_j \right)}{n_i + n_j} \\
&= \left(\frac{n_i n_j}{n_i + n_j} \right) \left\| \boldsymbol{\mu}_i - \boldsymbol{\mu}_j \right\|^2
\end{aligned}$$

Ward's measure is therefore a weighted version of the mean distance measure because if we use Euclidean distance, the mean distance in Eq. (14.2) can be rewritten as

$$\delta(\boldsymbol{\mu}_i, \boldsymbol{\mu}_j) = \left\| \boldsymbol{\mu}_i - \boldsymbol{\mu}_j \right\|^2 \qquad (14.6)$$

We can see that the only difference is that Ward's measure weights the distance between the means by half of the harmonic mean of the cluster sizes, where the harmonic mean of two numbers n_1 and n_2 is given as $\frac{2}{\frac{1}{n_1} + \frac{1}{n_2}} = \frac{2 n_1 n_2}{n_1 + n_2}$.

Example 14.3 (Single Link). Consider the single link clustering shown in Figure 14.3 on a dataset of five points, whose pairwise distances are also shown on the bottom left. Initially, all points are in their own cluster. The closest pair of points are (A, B) and (C, D), both with $\delta = 1$. We choose to first merge A and B, and derive a new distance matrix for the merged cluster. Essentially, we have to

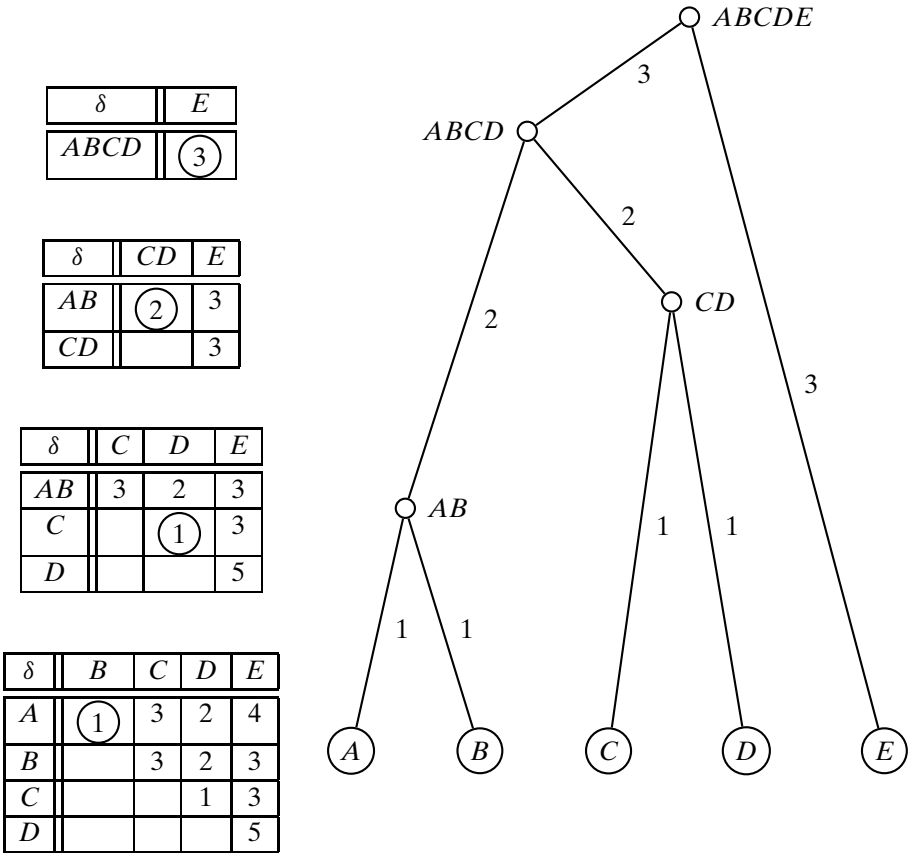


Figure 14.3. Single link agglomerative clustering.

compute the distances of the new cluster AB to all other clusters. For example, $\delta(AB, E) = 3$ because $\delta(AB, E) = \min\{\delta(A, E), \delta(B, E)\} = \min\{4, 3\} = 3$. In the next step we merge C and D because they are the closest clusters, and we obtain a new distance matrix for the resulting set of clusters. After this, AB and CD are merged, and finally, E is merged with $ABCD$. In the distance matrices, we have shown (circled) the minimum distance used at each iteration that results in a merging of the two closest pairs of clusters.

14.2.2 Updating Distance Matrix

Whenever two clusters C_i and C_j are merged into C_{ij}, we need to update the distance matrix by recomputing the distances from the newly created cluster C_{ij} to all other clusters C_r ($r \neq i$ and $r \neq j$). The Lance–Williams formula provides a general equation to recompute the distances for all of the cluster proximity measures we considered earlier; it is given as

$$\delta(C_{ij}, C_r) = \alpha_i \cdot \delta(C_i, C_r) + \alpha_j \cdot \delta(C_j, C_r) + \beta \cdot \delta(C_i, C_j) + \gamma \cdot \left|\delta(C_i, C_r) - \delta(C_j, C_r)\right| \tag{14.7}$$

Table 14.1. Lance–Williams formula for cluster proximity

Measure	α_i	α_j	β	γ
Single link	$\frac{1}{2}$	$\frac{1}{2}$	0	$-\frac{1}{2}$
Complete link	$\frac{1}{2}$	$\frac{1}{2}$	0	$\frac{1}{2}$
Group average	$\frac{n_i}{n_i+n_j}$	$\frac{n_j}{n_i+n_j}$	0	0
Mean distance	$\frac{n_i}{n_i+n_j}$	$\frac{n_j}{n_i+n_j}$	$\frac{-n_i \cdot n_j}{(n_i+n_j)^2}$	0
Ward's measure	$\frac{n_i+n_r}{n_i+n_j+n_r}$	$\frac{n_j+n_r}{n_i+n_j+n_r}$	$\frac{-n_r}{n_i+n_j+n_r}$	0

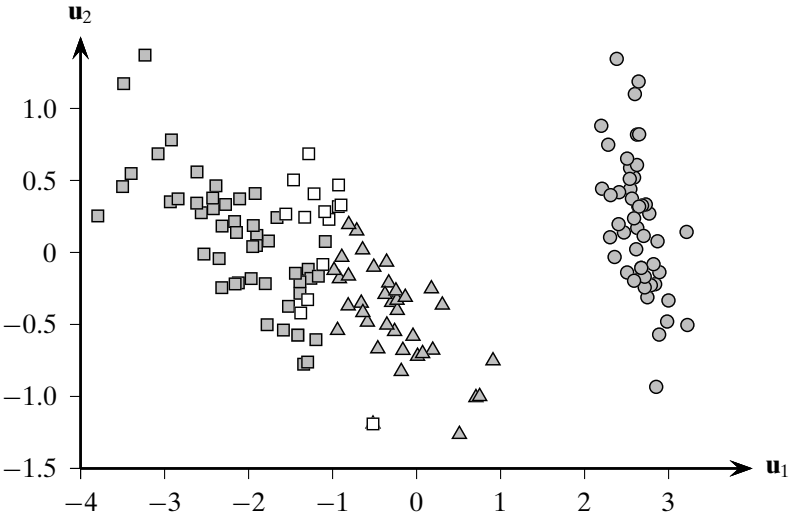


Figure 14.4. Iris dataset: complete link.

The coefficients $\alpha_i, \alpha_j, \beta$, and γ differ from one measure to another. Let $n_i = |C_i|$ denote the cardinality of cluster C_i; then the coefficients for the different distance measures are as shown in Table 14.1.

Example 14.4. Consider the two-dimensional Iris principal components dataset shown in Figure 14.4, which also illustrates the results of hierarchical clustering using the complete-link method, with $k = 3$ clusters. Table 14.2 shows the contingency table comparing the clustering results with the ground-truth Iris types (which are not used in clustering). We can observe that 15 points are misclustered in total; these points are shown in white in Figure 14.4. Whereas `iris-setosa` is well separated, the other two Iris types are harder to separate.

14.2.3 Computational Complexity

In agglomerative clustering, we need to compute the distance of each cluster to all other clusters, and at each step the number of clusters decreases by 1. Initially it takes

Table 14.2. Contingency table: clusters versus Iris types

	`iris-setosa`	`iris-virginica`	`iris-versicolor`
C_1 (circle)	50	0	0
C_2 (triangle)	0	1	36
C_3 (square)	0	49	14

$O(n^2)$ time to create the pairwise distance matrix, unless it is specified as an input to the algorithm.

At each merge step, the distances from the merged cluster to the other clusters have to be recomputed, whereas the distances between the other clusters remain the same. This means that in step t, we compute $O(n - t)$ distances. The other main operation is to find the closest pair in the distance matrix. For this we can keep the n^2 distances in a heap data structure, which allows us to find the minimum distance in $O(1)$ time; creating the heap takes $O(n^2)$ time. Deleting/updating distances from the heap takes $O(\log n)$ time for each operation, for a total time across all merge steps of $O(n^2 \log n)$. Thus, the computational complexity of hierarchical clustering is $O(n^2 \log n)$.

14.3 FURTHER READING

Hierarchical clustering has a long history, especially in taxonomy or classificatory systems, and phylogenetics; see, for example, Sokal and Sneath (1963). The generic Lance–Williams formula for distance updates appears in Lance and Williams (1967). Ward's measure is from Ward (1963). Efficient methods for single-link and complete-link measures with $O(n^2)$ complexity are given in Sibson (1973) and Defays (1977), respectively. For a good discussion of hierarchical clustering, and clustering in general, see Jain and Dubes (1988).

Defays, D. (Nov. 1977). "An efficient algorithm for a complete link method." *Computer Journal*, 20 (4): 364–366.

Jain, A. K. and Dubes, R. C. (1988). *Algorithms for Clustering Data*. Upper Saddle River, NJ: Prentice-Hall.

Lance, G. N. and Williams, W. T. (1967). "A general theory of classificatory sorting strategies 1. Hierarchical systems." *The Computer Journal*, 9 (4): 373–380.

Sibson, R. (1973). "SLINK: An optimally efficient algorithm for the single-link cluster method." *Computer Journal*, 16 (1): 30–34.

Sokal, R. R. and Sneath, P. H. (1963). *Principles of Numerical Taxonomy*. San Francisco: W.H. Freeman.

Ward, J. H. (1963). "Hierarchical grouping to optimize an objective function." *Journal of the American Statistical Association*, 58 (301): 236–244.

14.4 EXERCISES AND PROJECTS

Q1. Consider the 5-dimensional categorical data shown in Table 14.3.

Table 14.3. Data for Q1

Point	X_1	X_2	X_3	X_4	X_5
$\mathbf{x}_1$	1	0	1	1	0
$\mathbf{x}_2$	1	1	0	1	0
$\mathbf{x}_3$	0	0	1	1	0
$\mathbf{x}_4$	0	1	0	1	0
$\mathbf{x}_5$	1	0	1	0	1
$\mathbf{x}_6$	0	1	1	0	0

The similarity between categorical data points can be computed in terms of the number of matches and mismatches for the different attributes. Let n_{11} be the number of attributes on which two points $\mathbf{x}_i$ and $\mathbf{x}_j$ assume the value 1, and let n_{10} denote the number of attributes where $\mathbf{x}_i$ takes value 1, but $\mathbf{x}_j$ takes on the value of 0. Define n_{01} and n_{00} in a similar manner. The contingency table for measuring the similarity is then given as

		$\mathbf{x}_j$	
		1	0
$\mathbf{x}_i$	1	n_{11}	n_{10}
	0	n_{01}	n_{00}

Define the following similarity measures:

- Simple matching coefficient: $SMC(X_i, X_j) = \frac{n_{11}+n_{00}}{n_{11}+n_{10}+n_{01}+n_{00}}$
- Jaccard coefficient: $JC(X_i, X_j) = \frac{n_{11}}{n_{11}+n_{10}+n_{01}}$
- Rao's coefficient: $RC(X_i, X_j) = \frac{n_{11}}{n_{11}+n_{10}+n_{01}+n_{00}}$

Find the cluster dendrograms produced by the hierarchical clustering algorithm under the following scenarios:

(a) We use single link with *RC*.

(b) We use complete link with *SMC*.

(c) We use group average with *JC*.

Q2. Given the dataset in Figure 14.5, show the dendrogram resulting from the single-link hierarchical agglomerative clustering approach using the L_1-*norm* as the distance between points

$$\delta(\mathbf{x}, \mathbf{y}) = \sum_{a=1}^{2} |x_{ia} - y_{ia}|$$

Whenever there is a choice, merge the cluster that has the lexicographically smallest labeled point. Show the cluster merge order in the tree, stopping when you have $k = 4$ clusters. Show the full distance matrix at each step.

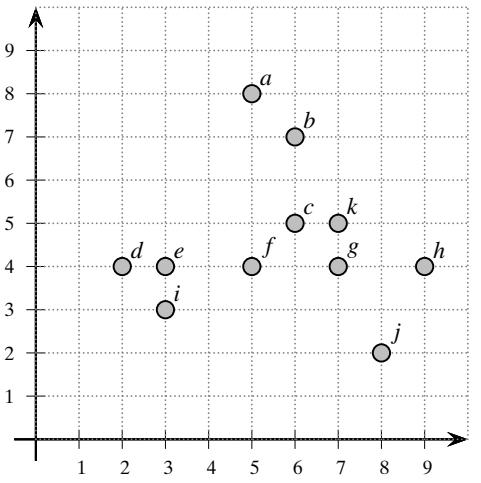


Figure 14.5. Dataset for Q2.

Table 14.4. Dataset for Q3

	A	*B*	*C*	*D*	*E*
A	0	1	3	2	4
B		0	3	2	3
C			0	1	3
D				0	5
E					0

Q3. Using the distance matrix from Table 14.4, use the average link method to generate hierarchical clusters. Show the merging distance thresholds.

Q4. Prove that in the Lance–Williams formula [Eq. (14.7)]

(a) If $\alpha_i = \frac{n_i}{n_i+n_j}$, $\alpha_j = \frac{n_j}{n_i+n_j}$, $\beta = 0$ and $\gamma = 0$, then we obtain the group average measure.

(b) If $\alpha_i = \frac{n_i+n_r}{n_i+n_j+n_r}$, $\alpha_j = \frac{n_j+n_r}{n_i+n_j+n_r}$, $\beta = \frac{-n_r}{n_i+n_j+n_r}$ and $\gamma = 0$, then we obtain Ward's measure.

Q5. If we treat each point as a vertex, and add edges between two nodes with distance less than some threshold value, then the single-link method corresponds to a well known graph algorithm. Describe this graph-based algorithm to hierarchically cluster the nodes via single-link measure, using successively higher distance thresholds.

CHAPTER 15 Density-based Clustering

The representative-based clustering methods like K-means and expectation-maximization are suitable for finding ellipsoid-shaped clusters, or at best convex clusters. However, for nonconvex clusters, such as those shown in Figure 15.1, these methods have trouble finding the true clusters, as two points from different clusters may be closer than two points in the same cluster. The density-based methods we consider in this chapter are able to mine such nonconvex clusters.

15.1 THE DBSCAN ALGORITHM

Density-based clustering uses the local density of points to determine the clusters, rather than using only the distance between points. We define a ball of radius ϵ around a point $\mathbf{x} \in \mathbb{R}^d$, called the *$\epsilon$-neighborhood* of $\mathbf{x}$, as follows:

$$N_\epsilon(\mathbf{x}) = B_d(\mathbf{x}, \epsilon) = \{\mathbf{y} \mid \delta(\mathbf{x}, \mathbf{y}) \leq \epsilon\}$$

Here $\delta(\mathbf{x}, \mathbf{y})$ represents the distance between points $\mathbf{x}$ and $\mathbf{y}$, which is usually assumed to be the Euclidean distance, that is, $\delta(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_2$. However, other distance metrics can also be used.

For any point $\mathbf{x} \in \mathbf{D}$, we say that $\mathbf{x}$ is a *core point* if there are at least $minpts$ points in its ϵ-neighborhood. In other words, $\mathbf{x}$ is a core point if $|N_\epsilon(\mathbf{x})| \geq minpts$, where $minpts$ is a user-defined local density or frequency threshold. A *border point* is defined as a point that does not meet the $minpts$ threshold, that is, it has $|N_\epsilon(\mathbf{x})| < minpts$, but it belongs to the ϵ-neighborhood of some core point $\mathbf{z}$, that is, $\mathbf{x} \in N_\epsilon(\mathbf{z})$. Finally, if a point is neither a core nor a border point, then it is called a *noise point* or an outlier.

Example 15.1. Figure 15.2a shows the ϵ-neighborhood of the point $\mathbf{x}$, using the Euclidean distance metric. Figure 15.2b shows the three different types of points, using $minpts = 6$. Here $\mathbf{x}$ is a core point because $|N_\epsilon(\mathbf{x})| = 6$, $\mathbf{y}$ is a border point because $|N_\epsilon(\mathbf{y})| < minpts$, but it belongs to the ϵ-neighborhood of the core point $\mathbf{x}$, i.e., $\mathbf{y} \in N_\epsilon(\mathbf{x})$. Finally, $\mathbf{z}$ is a noise point.

We say that a point $\mathbf{x}$ is *directly density reachable* from another point $\mathbf{y}$ if $\mathbf{x} \in N_\epsilon(\mathbf{y})$ and $\mathbf{y}$ is a core point. We say that $\mathbf{x}$ is *density reachable* from $\mathbf{y}$ if there exists a chain

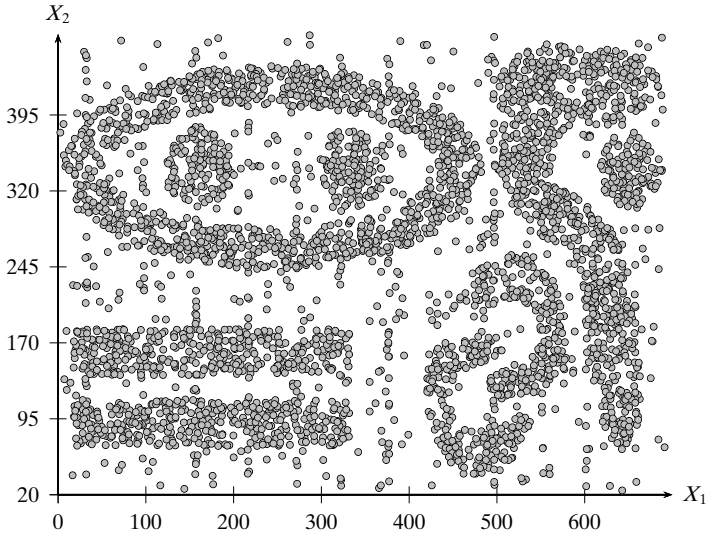


Figure 15.1. Density-based dataset.

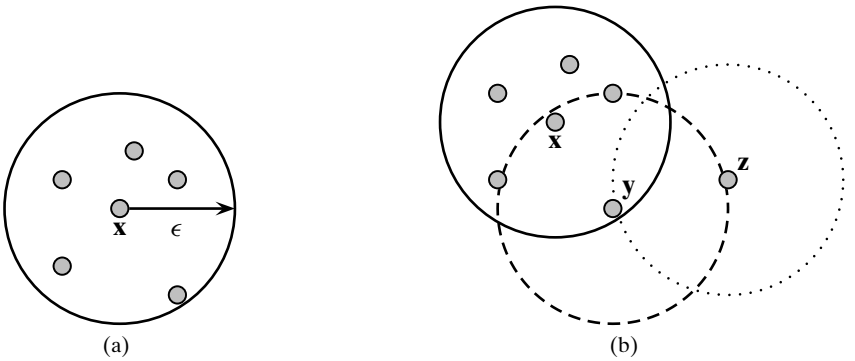


Figure 15.2. (a) Neighborhood of a point. (b) Core, border, and noise points.

of points, $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_l$, such that $\mathbf{x} = \mathbf{x}_0$ and $\mathbf{y} = \mathbf{x}_l$, and $\mathbf{x}_i$ is directly density reachable from $\mathbf{x}_{i-1}$ for all $i = 1, \ldots, l$. In other words, there is set of core points leading from $\mathbf{y}$ to $\mathbf{x}$. Note that density reachability is an asymmetric or directed relationship. Define any two points $\mathbf{x}$ and $\mathbf{y}$ to be *density connected* if there exists a core point $\mathbf{z}$, such that both $\mathbf{x}$ and $\mathbf{y}$ are density reachable from $\mathbf{z}$. A *density-based cluster* is defined as a maximal set of density connected points.

The pseudo-code for the DBSCAN density-based clustering method is shown in Algorithm 15.1. First, DBSCAN computes the ϵ-neighborhood $N_\epsilon(\mathbf{x}_i)$ for each point $\mathbf{x}_i$ in the dataset $\mathbf{D}$, and checks if it is a core point (lines 2–5). It also sets the cluster id $id(\mathbf{x}_i) = \emptyset$ for all points, indicating that they are not assigned to any cluster. Next, starting from each unassigned core point, the method recursively finds all its density connected points, which are assigned to the same cluster (line 10). Some border point

ALGORITHM 15.1. Density-based Clustering Algorithm

DBSCAN (D, ϵ, $minpts$):
1 $Core \leftarrow \emptyset$
2 **foreach** $\mathbf{x}_i \in \mathbf{D}$ **do** // Find the core points
3 Compute $N_\epsilon(\mathbf{x}_i)$
4 $id(\mathbf{x}_i) \leftarrow \emptyset$ // cluster id for $\mathbf{x}_i$
5 **if** $N_\epsilon(\mathbf{x}_i) \geq minpts$ **then** $Core \leftarrow Core \cup \{\mathbf{x}_i\}$
6 $k \leftarrow 0$ // cluster id
7 **foreach** $\mathbf{x}_i \in Core$, *such that* $id(\mathbf{x}_i) = \emptyset$ **do**
8 $k \leftarrow k + 1$
9 $id(\mathbf{x}_i) \leftarrow k$ // assign $\mathbf{x}_i$ to cluster id k
10 DENSITYCONNECTED $(\mathbf{x}_i, k)$
11 $\mathcal{C} \leftarrow \{C_i\}_{i=1}^{k}$, where $C_i \leftarrow \{\mathbf{x} \in \mathbf{D} \mid id(\mathbf{x}) = i\}$
12 $Noise \leftarrow \{\mathbf{x} \in \mathbf{D} \mid id(\mathbf{x}) = \emptyset\}$
13 $Border \leftarrow \mathbf{D} \setminus \{Core \cup Noise\}$
14 **return** $\mathcal{C}, Core, Border, Noise$

DENSITYCONNECTED (x, k):
15 **foreach** $\mathbf{y} \in N_\epsilon(\mathbf{x})$ **do**
16 $id(\mathbf{y}) \leftarrow k$ // assign $\mathbf{y}$ to cluster id k
17 **if** $\mathbf{y} \in Core$ **then** DENSITYCONNECTED $(\mathbf{y}, k)$

may be reachable from core points in more than one cluster; they may either be arbitrarily assigned to one of the clusters or to all of them (if overlapping clusters are allowed). Those points that do not belong to any cluster are treated as outliers or noise.

DBSCAN can also be considered as a search for the connected components in a graph where the vertices correspond to the core points in the dataset, and there exists an (undirected) edge between two vertices (core points) if the distance between them is less than ϵ, that is, each of them is in the ϵ-neighborhood of the other point. The connected components of this graph correspond to the core points of each cluster. Next, each core point incorporates into its cluster any border points in its neighborhood.

One limitation of DBSCAN is that it is sensitive to the choice of ϵ, in particular if clusters have different densities. If ϵ is too small, sparser clusters will be categorized as noise. If ϵ is too large, denser clusters may be merged together. In other words, if there are clusters with different local densities, then a single ϵ value may not suffice.

Example 15.2. Figure 15.3 shows the clusters discovered by DBSCAN on the density-based dataset in Figure 15.1. For the parameter values $\epsilon = 15$ and $minpts = 10$, found after parameter tuning, DBSCAN yields a near-perfect clustering comprising all nine clusters. Cluster are shown using different symbols and shading; noise points are shown as plus symbols.

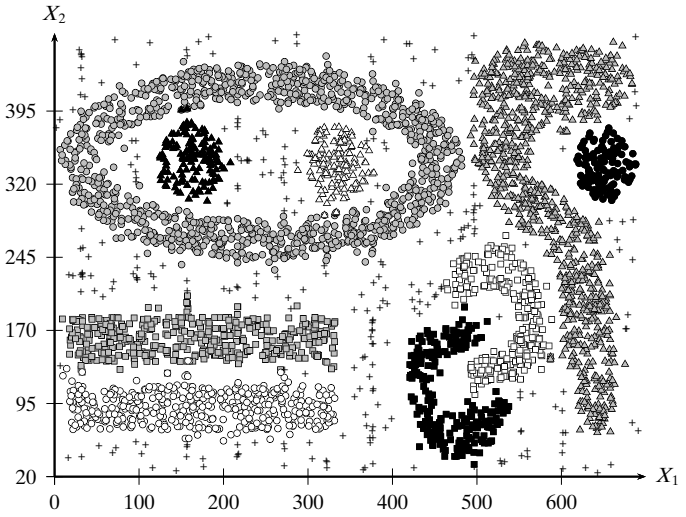


Figure 15.3. Density-based clusters.

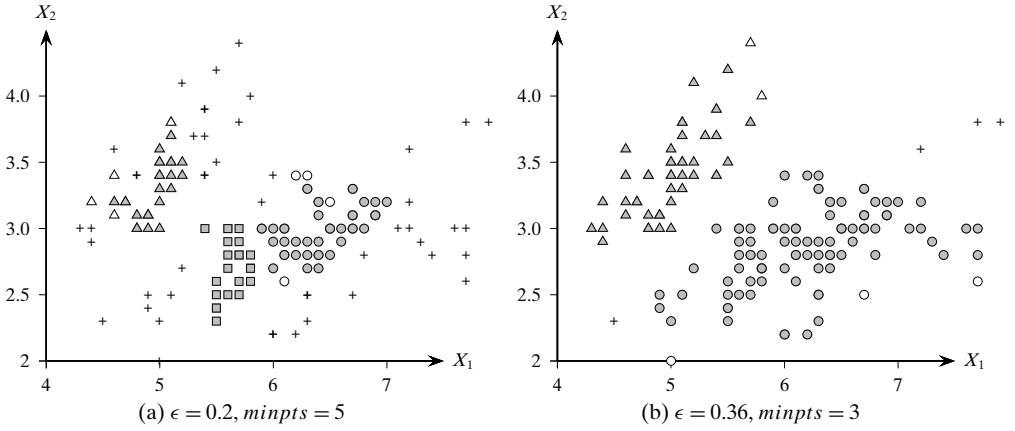


Figure 15.4. DBSCAN clustering: Iris dataset.

Example 15.3. Figure 15.4 shows the clusterings obtained via DBSCAN on the two-dimensional Iris dataset (over `sepal length` and `sepal width` attributes) for two different parameter settings. Figure 15.4a shows the clusters obtained with radius $\epsilon = 0.2$ and core threshold $minpts = 5$. The three clusters are plotted using different shaped points, namely circles, squares, and triangles. Shaded points are core points, whereas the border points for each cluster are showed unshaded (white). Noise points are shown as plus symbols. Figure 15.4b shows the clusters obtained with a larger value of radius $\epsilon = 0.36$, with $minpts = 3$. Two clusters are found, corresponding to the two dense regions of points.

For this dataset tuning the parameters is not that easy, and DBSCAN is not very effective in discovering the three Iris classes. For instance it identifies too many points (47 of them) as noise in Figure 15.4a. However, DBSCAN is able to find the two main dense sets of points, distinguishing `iris-setosa` (in triangles) from the other types of Irises, in Figure 15.4b. Increasing the radius more than $\epsilon = 0.36$ collapses all points into a single large cluster.

Computational Complexity

The main cost in DBSCAN is for computing the ϵ-neighborhood for each point. If the dimensionality is not too high this can be done efficiently using a spatial index structure in $O(n \log n)$ time. When dimensionality is high, it takes $O(n^2)$ to compute the neighborhood for each point. Once $N_\epsilon(\mathbf{x})$ has been computed the algorithm needs only a single pass over all the points to find the density connected clusters. Thus, the overall complexity of DBSCAN is $O(n^2)$ in the worst-case.

15.2 KERNEL DENSITY ESTIMATION

There is a close connection between density-based clustering and density estimation. The goal of density estimation is to determine the unknown probability density function by finding the dense regions of points, which can in turn be used for clustering. Kernel density estimation is a nonparametric technique that does not assume any fixed probability model of the clusters, as in the case of K-means or the mixture model assumed in the EM algorithm. Instead, it tries to directly infer the underlying probability density at each point in the dataset.

15.2.1 Univariate Density Estimation

Assume that X is a continuous random variable, and let $x_1, x_2, \ldots, x_n$ be a random sample drawn from the underlying probability density function $f(x)$, which is assumed to be unknown. We can directly estimate the cumulative distribution function from the data by counting how many points are less than or equal to x:

$$\hat{F}(x) = \frac{1}{n} \sum_{i=1}^{n} I(x_i \leq x)$$

where I is an indicator function that has value 1 only when its argument is true, and 0 otherwise. We can estimate the density function by taking the derivative of $\hat{F}(x)$, by considering a window of small width h centered at x, that is,

$$\hat{f}(x) = \frac{\hat{F}\left(x + \frac{h}{2}\right) - \hat{F}\left(x - \frac{h}{2}\right)}{h} = \frac{k/n}{h} = \frac{k}{nh} \tag{15.1}$$

where k is the number of points that lie in the window of width h centered at x, that is, within the closed interval $[x - \frac{h}{2},\ x + \frac{h}{2}]$. Thus, the density estimate is the ratio of the fraction of the points in the window (k/n) to the volume of the window (h). Here h plays the role of "influence." That is, a large h estimates the probability density over a large window by considering many points, which has the effect of smoothing the estimate. On the other hand, if h is small, then only the points in close proximity to x are considered. In general we want a small value of h, but not too small, as in that case no points will fall in the window and we will not be able to get an accurate estimate of the probability density.

Kernel Estimator

Kernel density estimation relies on a *kernel function* K that is non-negative, symmetric, and integrates to 1, that is, $K(x) \geq 0$, $K(-x) = K(x)$ for all values x, and $\int K(x)dx = 1$. Thus, K is essentially a probability density function. Note that K should not be confused with the positive semidefinite kernel mentioned in Chapter 5.

Discrete Kernel The density estimate $\hat{f}(x)$ from Eq. (15.1) can also be rewritten in terms of the kernel function as follows:

$$\hat{f}(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)$$

where the **discrete kernel** function K computes the number of points in a window of width h, and is defined as

$$K(z) = \begin{cases} 1 & \text{If } |z| \leq \frac{1}{2} \\ 0 & \text{Otherwise} \end{cases} \tag{15.2}$$

We can see that if $|z| = |\frac{x - x_i}{h}| \leq \frac{1}{2}$, then the point x_i is within a window of width h centered at x, as

$$\left|\frac{x - x_i}{h}\right| \leq \frac{1}{2} \text{ implies that } -\frac{1}{2} \leq \frac{x_i - x}{h} \leq \frac{1}{2}, \text{ or}$$

$$-\frac{h}{2} \leq x_i - x \leq \frac{h}{2}, \text{ and finally}$$

$$x - \frac{h}{2} \leq x_i \leq x + \frac{h}{2}$$

Example 15.4. Figure 15.5 shows the kernel density estimates using the discrete kernel for different values of the influence parameter h, for the one-dimensional Iris dataset comprising the `sepal length` attribute. The x-axis plots the $n = 150$ data points. Because several points have the same value, they are shown stacked, where the stack height corresponds to the frequency of that value.

When h is small, as shown in Figure 15.5a, the density function has many local maxima or modes. However, as we increase h from 0.25 to 2, the number of modes decreases, until h becomes large enough to yield a unimodal distribution, as shown in Figure 15.5d. We can observe that the discrete kernel yields a non-smooth (or jagged) density function.

Gaussian Kernel The width h is a parameter that denotes the spread or smoothness of the density estimate. If the spread is too large we get a more averaged value. If it is too small we do not have enough points in the window. Further, the kernel function in Eq. (15.2) has an abrupt influence. For points within the window ($|z| \leq \frac{1}{2}$) there is a net contribution of $\frac{1}{hn}$ to the probability estimate $\hat{f}(x)$. On the other hand, points outside the window ($|z| > \frac{1}{2}$) contribute 0.

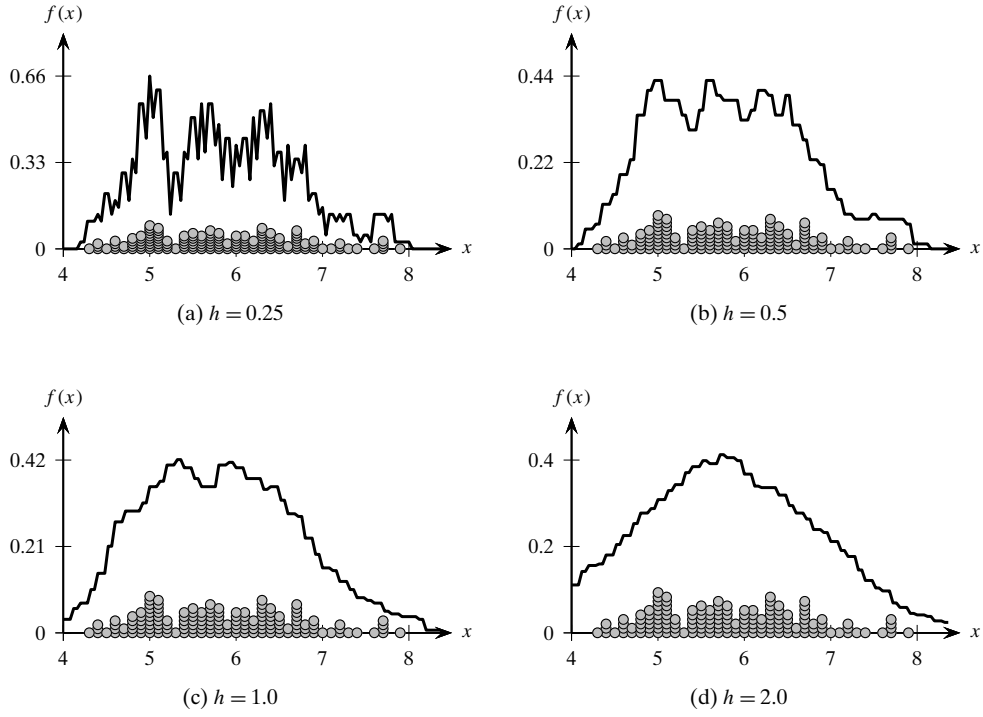


Figure 15.5. Kernel density estimation: discrete kernel (varying h).

Instead of the discrete kernel, we can define a more smooth transition of influence via a Gaussian kernel:

$$K(z) = \frac{1}{\sqrt{2\pi}} \exp\left\{-\frac{z^2}{2}\right\}$$

Thus, we have

$$K\left(\frac{x - x_i}{h}\right) = \frac{1}{\sqrt{2\pi}} \exp\left\{-\frac{(x - x_i)^2}{2h^2}\right\}$$

Here x, which is at the center of the window, plays the role of the mean, and h acts as the standard deviation.

Example 15.5. Figure 15.6 shows the univariate density function for the 1-dimensional Iris dataset (over `sepal length`) using the Gaussian kernel. Plots are shown for increasing values of the spread parameter h. The data points are shown stacked along the x-axis, with the heights corresponding to the value frequencies.

As h varies from 0.1 to 0.5, we can see the smoothing effect of increasing h on the density function. For instance, for $h = 0.1$ there are many local maxima, whereas for $h = 0.5$ there is only one density peak. Compared to the discrete kernel case shown in Figure 15.5, we can clearly see that the Gaussian kernel yields much smoother estimates, without discontinuities.

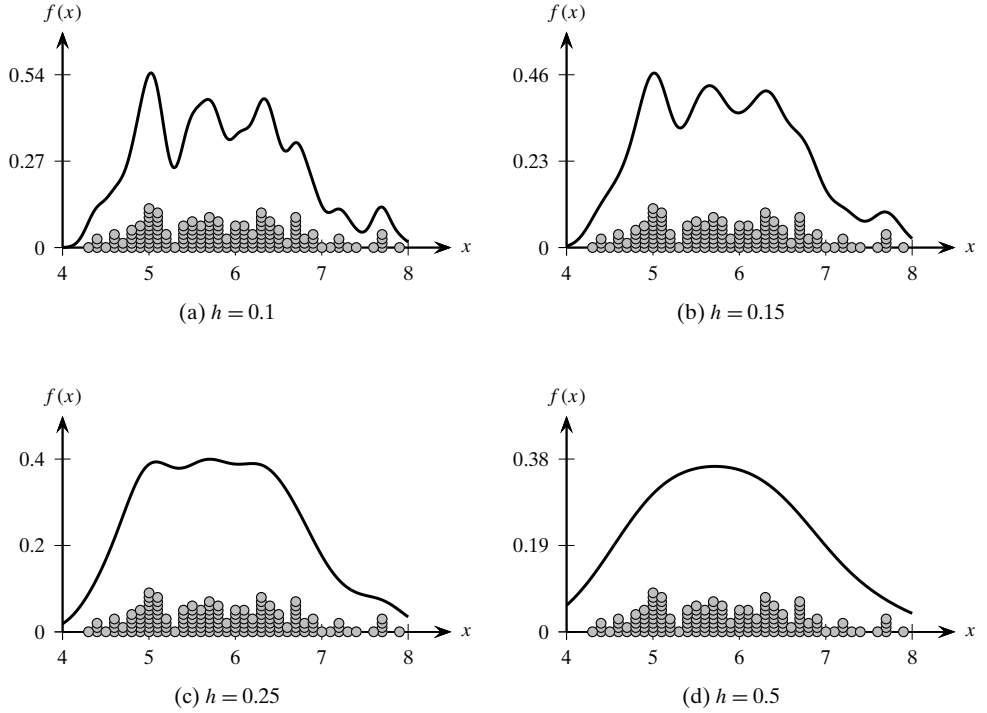


Figure 15.6. Kernel density estimation: Gaussian kernel (varying h).

15.2.2 Multivariate Density Estimation

To estimate the probability density at a d-dimensional point $\mathbf{x} = (x_1, x_2, \ldots, x_d)^T$, we define the d-dimensional "window" as a hypercube in d dimensions, that is, a hypercube centered at $\mathbf{x}$ with edge length h. The volume of such a d-dimensional hypercube is given as

$$\text{vol}(H_d(h)) = h^d$$

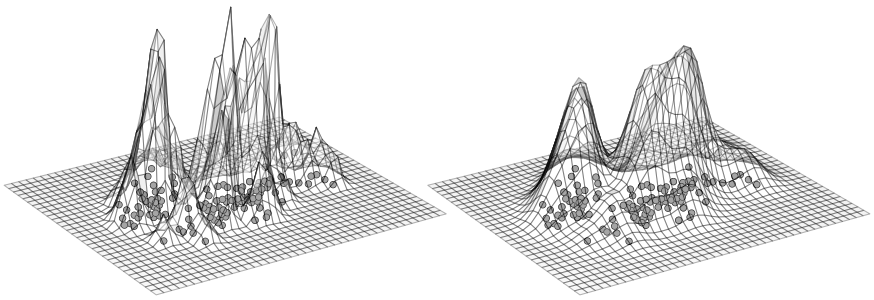
The density is then estimated as the fraction of the point weight lying within the d-dimensional window centered at $\mathbf{x}$, divided by the volume of the hypercube:

$$\hat{f}(\mathbf{x}) = \frac{1}{nh^d} \sum_{i=1}^{n} K\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right) \tag{15.3}$$

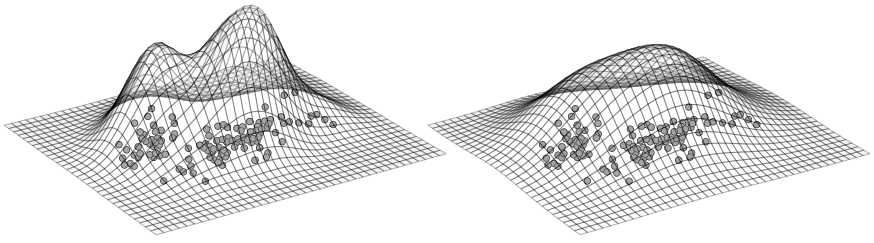
where the multivariate kernel function K satisfies the condition $\int K(\mathbf{z}) d\mathbf{z} = 1$.

Discrete Kernel For any d-dimensional vector $\mathbf{z} = (z_1, z_2, \ldots, z_d)^T$, the discrete kernel function in d-dimensions is given as

$$K(\mathbf{z}) = \begin{cases} 1 & \text{If } |z_j| \leq \frac{1}{2}, \text{ for all dimensions } j = 1, \ldots, d \\ 0 & \text{Otherwise} \end{cases}$$



(a) $h = 0.1$ (b) $h = 0.2$



(c) $h = 0.35$ (d) $h = 0.6$

Figure 15.7. Density estimation: 2D Iris dataset (varying h).

For $\mathbf{z} = \frac{\mathbf{x}-\mathbf{x}_i}{h}$, we see that the kernel computes the number of points within the hypercube centered at $\mathbf{x}$ because $K(\frac{\mathbf{x}-\mathbf{x}_i}{h}) = 1$ if and only if $|\frac{x_j - x_{ij}}{h}| \leq \frac{1}{2}$ for all dimensions j. Each point within the hypercube thus contributes a weight of $\frac{1}{n}$ to the density estimate.

Gaussian Kernel The d-dimensional Gaussian kernel is given as

$$K(\mathbf{z}) = \frac{1}{(2\pi)^{d/2}} \exp\left\{-\frac{\mathbf{z}^T\mathbf{z}}{2}\right\} \tag{15.4}$$

where we assume that the covariance matrix is the $d \times d$ identity matrix, that is, $\mathbf{\Sigma} = \mathbf{I}_d$. Plugging $\mathbf{z} = \frac{\mathbf{x}-\mathbf{x}_i}{h}$ in Eq. (15.4), we have

$$K\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right) = \frac{1}{(2\pi)^{d/2}} \exp\left\{-\frac{(\mathbf{x}-\mathbf{x}_i)^T(\mathbf{x}-\mathbf{x}_i)}{2h^2}\right\}$$

Each point contributes a weight to the density estimate inversely proportional to its distance from $\mathbf{x}$ tempered by the width parameter h.

Example 15.6. Figure 15.7 shows the probability density function for the 2D Iris dataset comprising the `sepal length` and `sepal width` attributes, using the Gaussian kernel. As expected, for small values of h the density function has several local maxima, whereas for larger values the number of maxima reduce, and ultimately for a large enough value we obtain a unimodal distribution.

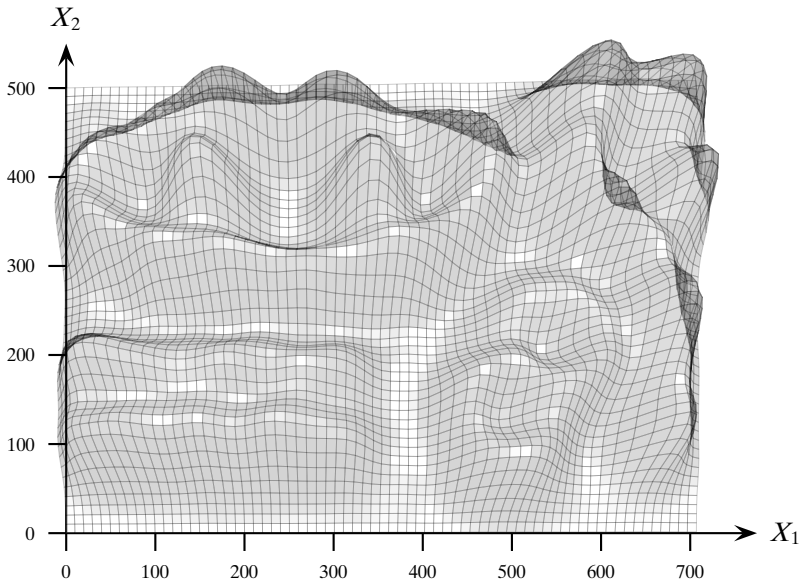


Figure 15.8. Density estimation: density-based dataset.

Example 15.7. Figure 15.8 shows the kernel density estimate for the density-based dataset in Figure 15.1, using a Gaussian kernel with $h = 20$. One can clearly discern that the density peaks closely correspond to regions with higher density of points.

15.2.3 Nearest Neighbor Density Estimation

In the preceding density estimation formulation we implicitly fixed the volume by fixing the width h, and we used the kernel function to find out the number or weight of points that lie inside the fixed volume region. An alternative approach to density estimation is to fix k, the number of points required to estimate the density, and allow the volume of the enclosing region to vary to accommodate those k points. This approach is called the k nearest neighbors (KNN) approach to density estimation. Like kernel density estimation, KNN density estimation is also a nonparametric approach.

Given k, the number of neighbors, we estimate the density at $\mathbf{x}$ as follows:

$$\hat{f}(\mathbf{x}) = \frac{k}{n \, \text{vol}(S_d(h_\mathbf{x}))}$$

where $h_\mathbf{x}$ is the distance from $\mathbf{x}$ to its kth nearest neighbor, and $\text{vol}(S_d(h_\mathbf{x}))$ is the volume of the d-dimensional hypersphere $S_d(h_\mathbf{x})$ centered at $\mathbf{x}$, with radius $h_\mathbf{x}$ [Eq. (6.4)]. In other words, the width (or radius) $h_\mathbf{x}$ is now a variable, which depends on $\mathbf{x}$ and the chosen value k.

15.3 DENSITY-BASED CLUSTERING: DENCLUE

Having laid the foundations of kernel density estimation, we can develop a general formulation of density-based clustering. The basic approach is to find the peaks in the density landscape via gradient-based optimization, and find the regions with density above a given threshold.

Density Attractors and Gradient

A point $\mathbf{x}^*$ is called a *density attractor* if it is a local maxima of the probability density function f. A density attractor can be found via a gradient ascent approach starting at some point $\mathbf{x}$. The idea is to compute the density gradient, the direction of the largest increase in the density, and to move in the direction of the gradient in small steps, until we reach a local maxima.

The gradient at a point $\mathbf{x}$ can be computed as the multivariate derivative of the probability density estimate in Eq. (15.3), given as

$$\nabla \hat{f}(\mathbf{x}) = \frac{\partial}{\partial \mathbf{x}} \hat{f}(\mathbf{x}) = \frac{1}{nh^d} \sum_{i=1}^{n} \frac{\partial}{\partial \mathbf{x}} K\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right) \tag{15.5}$$

For the Gaussian kernel [Eq. (15.4)], we have

$$\begin{aligned} \frac{\partial}{\partial \mathbf{x}} K(\mathbf{z}) &= \left(\frac{1}{(2\pi)^{d/2}} \exp\left\{-\frac{\mathbf{z}^T\mathbf{z}}{2}\right\}\right) \cdot -\mathbf{z} \cdot \frac{\partial \mathbf{z}}{\partial \mathbf{x}} \\ &= K(\mathbf{z}) \cdot -\mathbf{z} \cdot \frac{\partial \mathbf{z}}{\partial \mathbf{x}} \end{aligned}$$

Setting $\mathbf{z} = \frac{\mathbf{x}-\mathbf{x}_i}{h}$ above, we get

$$\frac{\partial}{\partial \mathbf{x}} K\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right) = K\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right) \cdot \left(\frac{\mathbf{x}_i-\mathbf{x}}{h}\right) \cdot \left(\frac{1}{h}\right)$$

which follows from the fact that $\frac{\partial}{\partial \mathbf{x}}\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right) = \frac{1}{h}$. Substituting the above in Eq. (15.5), the gradient at a point $\mathbf{x}$ is given as

$$\nabla \hat{f}(\mathbf{x}) = \frac{1}{nh^{d+2}} \sum_{i=1}^{n} K\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right) \cdot (\mathbf{x}_i - \mathbf{x}) \tag{15.6}$$

This equation can be thought of as having two parts. A vector $(\mathbf{x}_i - \mathbf{x})$ and a scalar *influence* value $K(\frac{\mathbf{x}-\mathbf{x}_i}{h})$. For each point $\mathbf{x}_i$, we first compute the direction away from $\mathbf{x}$, that is, the vector $(\mathbf{x}_i - \mathbf{x})$. Next, we scale it using the Gaussian kernel value as the weight $K\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right)$. Finally, the vector $\nabla \hat{f}(\mathbf{x})$ is the net influence at $\mathbf{x}$, as illustrated in Figure 15.9, that is, the weighted sum of the difference vectors.

We say that $\mathbf{x}^*$ is a *density attractor* for $\mathbf{x}$, or alternatively that $\mathbf{x}$ is *density attracted* to $\mathbf{x}^*$, if a hill climbing process started at $\mathbf{x}$ converges to $\mathbf{x}^*$. That is, there exists a sequence of points $\mathbf{x} = \mathbf{x}_0 \rightarrow \mathbf{x}_1 \rightarrow \ldots \rightarrow \mathbf{x}_m$, starting from $\mathbf{x}$ and ending at $\mathbf{x}_m$, such that $\|\mathbf{x}_m - \mathbf{x}^*\| \leq \epsilon$, that is, $\mathbf{x}_m$ converges to the attractor $\mathbf{x}^*$.

The typical approach is to use the gradient-ascent method to compute $\mathbf{x}^*$, that is, starting from $\mathbf{x}$, we iteratively update it at each step t via the update rule:

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \delta \cdot \nabla \hat{f}(\mathbf{x}_t)$$

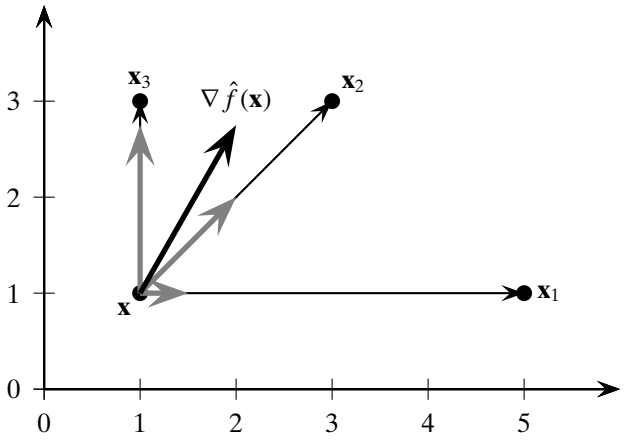


Figure 15.9. The gradient vector $\nabla \hat{f}(\mathbf{x})$ (shown in thick black) obtained as the sum of difference vectors $\mathbf{x}_i - \mathbf{x}$ (shown in gray).

where $\delta > 0$ is the step size. That is, each intermediate point is obtained after a small move in the direction of the gradient vector. However, the gradient-ascent approach can be slow to converge. Instead, one can directly optimize the move direction by setting the gradient [Eq. (15.6)] to the zero vector:

$$\nabla \hat{f}(\mathbf{x}) = \mathbf{0}$$

$$\frac{1}{nh^{d+2}} \sum_{i=1}^{n} K\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right) \cdot (\mathbf{x}_i - \mathbf{x}) = \mathbf{0}$$

$$\mathbf{x} \cdot \sum_{i=1}^{n} K\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right) = \sum_{i=1}^{n} K\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right) \mathbf{x}_i$$

$$\mathbf{x} = \frac{\sum_{i=1}^{n} K\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right) \mathbf{x}_i}{\sum_{i=1}^{n} K\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right)}$$

The point $\mathbf{x}$ is involved on both the left- and right-hand sides above; however, it can be used to obtain the following iterative update rule:

$$\mathbf{x}_{t+1} = \frac{\sum_{i=1}^{n} K\left(\frac{\mathbf{x}_t-\mathbf{x}_i}{h}\right) \mathbf{x}_i}{\sum_{i=1}^{n} K\left(\frac{\mathbf{x}_t-\mathbf{x}_i}{h}\right)} \tag{15.7}$$

where t denotes the current iteration and $\mathbf{x}_{t+1}$ is the updated value for the current vector $\mathbf{x}_t$. This direct update rule is essentially a weighted average of the influence (computed via the kernel function K) of each point $\mathbf{x}_i \in \mathbf{D}$ on the current point $\mathbf{x}_t$. The direct update rule results in much faster convergence of the hill-climbing process.

Center-defined Cluster

A cluster $C \subseteq \mathbf{D}$, is called a *center-defined cluster* if all the points $\mathbf{x} \in C$ are density attracted to a unique density attractor $\mathbf{x}^*$, such that $\hat{f}(\mathbf{x}^*) \geq \xi$, where ξ is a user-defined

minimum density threshold. In other words,

$$\hat{f}(\mathbf{x}^*) = \frac{1}{nh^d} \sum_{i=1}^{n} K\left(\frac{\mathbf{x}^* - \mathbf{x}_i}{h}\right) \geq \xi$$

Density-based Cluster

An arbitrary-shaped cluster $C \subseteq \mathbf{D}$ is called a *density-based cluster* if there exists a set of density attractors $\mathbf{x}_1^*, \mathbf{x}_2^*, \ldots, \mathbf{x}_m^*$, such that

1. Each point $\mathbf{x} \in C$ is attracted to some attractor $\mathbf{x}_i^*$.
2. Each density attractor has density above ξ. That is, $\hat{f}(\mathbf{x}_i^*) \geq \xi$.
3. Any two density attractors $\mathbf{x}_i^*$ and $\mathbf{x}_j^*$ are *density reachable*, that is, there exists a path from $\mathbf{x}_i^*$ to $\mathbf{x}_j^*$, such that for all points $\mathbf{y}$ on the path, $\hat{f}(\mathbf{y}) \geq \xi$.

DENCLUE Algorithm

The pseudo-code for DENCLUE is shown in Algorithm 15.2. The first step is to compute the density attractor $\mathbf{x}^*$ for each point $\mathbf{x}$ in the dataset (line 4). If the density at $\mathbf{x}^*$ is above the minimum density threshold ξ, the attractor is added to the set of attractors $\mathcal{A}$. The data point $\mathbf{x}$ is also added to the set of points $R(\mathbf{x}^*)$ attracted to $\mathbf{x}^*$

ALGORITHM 15.2. DENCLUE Algorithm

```
   DENCLUE (D, h, ξ, ε):
1  A ← ∅
2  foreach x ∈ D do // find density attractors
4      x* ← FINDATTRACTOR(x, D, h, ε)
5      if f̂(x*) ≥ ξ then
7          A ← A ∪ {x*}
9          R(x*) ← R(x*) ∪ {x}
11 C ← {maximal C ⊆ A | ∀x_i*, x_j* ∈ C, x_i* and x_j* are density reachable}
12 foreach C ∈ C do // density-based clusters
13     foreach x* ∈ C do C ← C ∪ R(x*)
14 return C

   FINDATTRACTOR (x, D, h, ε):
16 t ← 0
17 x_t ← x
18 repeat
20     x_{t+1} ← (Σ_{i=1}^{n} K((x_t − x_i)/h) · x_t) / (Σ_{i=1}^{n} K((x_t − x_i)/h))
21     t ← t + 1
22 until ‖x_t − x_{t−1}‖ ≤ ε
24 return x_t
```

(line 9). In the second step, DENCLUE finds all the maximal subsets of attractors $C \subseteq \mathcal{A}$, such that any pair of attractors in C is density-reachable from each other (line 11). These maximal subsets of mutually reachable attractors form the seed for each density-based cluster. Finally, for each attractor $\mathbf{x}^* \in C$, we add to the cluster all of the points $R(\mathbf{x}^*)$ that are attracted to $\mathbf{x}^*$, which results in the final set of clusters $\mathcal{C}$.

The FINDATTRACTOR method implements the hill-climbing process using the direct update rule [Eq. (15.7)], which results in fast convergence. To further speed up the influence computation, it is possible to compute the kernel values for only the nearest neighbors of $\mathbf{x}_t$. That is, we can index the points in the dataset $\mathbf{D}$ using a spatial index structure, so that we can quickly compute all the nearest neighbors of $\mathbf{x}_t$ within some radius r. For the Gaussian kernel, we can set $r = h \cdot z$, where h is the influence parameter that plays the role of standard deviation, and z specifies the number of standard deviations. Let $B_d(\mathbf{x}_t, r)$ denote the set of all points in $\mathbf{D}$ that lie within a d-dimensional ball of radius r centered at $\mathbf{x}_t$. The nearest neighbor based update rule can then be expressed as

$$\mathbf{x}_{t+1} = \frac{\sum_{\mathbf{x}_i \in B_d(\mathbf{x}_t, r)} K\left(\frac{\mathbf{x}_t - \mathbf{x}_i}{h}\right) \mathbf{x}_i}{\sum_{\mathbf{x}_i \in B_d(\mathbf{x}_t, r)} K\left(\frac{\mathbf{x}_t - \mathbf{x}_i}{h}\right)}$$

which can be used in line 20 in Algorithm 15.2. When the data dimensionality is not high, this can result in a significant speedup. However, the effectiveness deteriorates rapidly with increasing number of dimensions. This is due to two effects. The first is that finding $B_d(\mathbf{x}_t, r)$ reduces to a linear-scan of the data taking $O(n)$ time for each query. Second, due to the *curse of dimensionality* (see Chapter 6), nearly all points appear to be equally close to $\mathbf{x}_t$, thereby nullifying any benefits of computing the nearest neighbors.

Example 15.8. Figure 15.10 shows the DENCLUE clustering for the 2-dimensional Iris dataset comprising the `sepal length` and `sepal width` attributes. The results were obtained with $h = 0.2$ and $\xi = 0.08$, using a Gaussian kernel. The clustering is obtained by thresholding the probability density function in Figure 15.7b at $\xi = 0.08$. The two peaks correspond to the two final clusters. Whereas `iris setosa` is well separated, it is hard to separate the other two types of Irises.

Example 15.9. Figure 15.11 shows the clusters obtained by DENCLUE on the density-based dataset from Figure 15.1. Using the parameters $h = 10$ and $\xi = 9.5 \times 10^{-5}$, with a Gaussian kernel, we obtain eight clusters. The figure is obtained by slicing the density function at the density value ξ; only the regions above that value are plotted. All the clusters are correctly identified, with the exception of the two semicircular clusters on the lower right that appear merged into one cluster.

DENCLUE: Special Cases

It can be shown that DBSCAN is a special case of the general kernel density estimate based clustering approach, DENCLUE. If we let $h = \epsilon$ and $\xi = minpts$, then using a

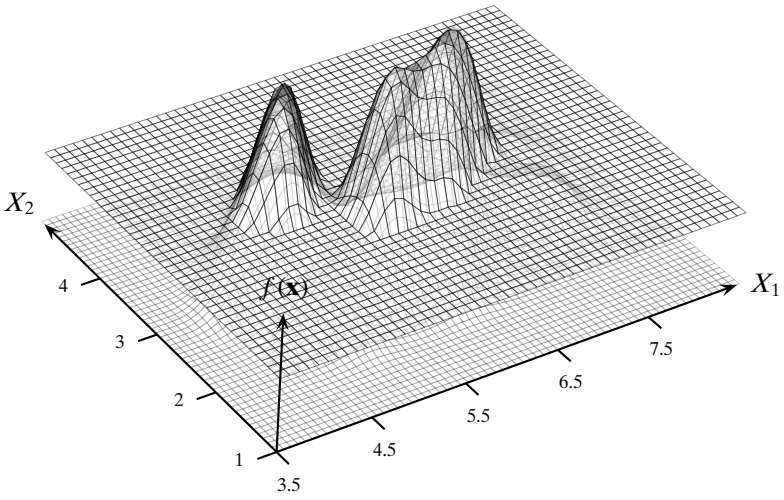


Figure 15.10. DENCLUE: Iris 2D dataset.

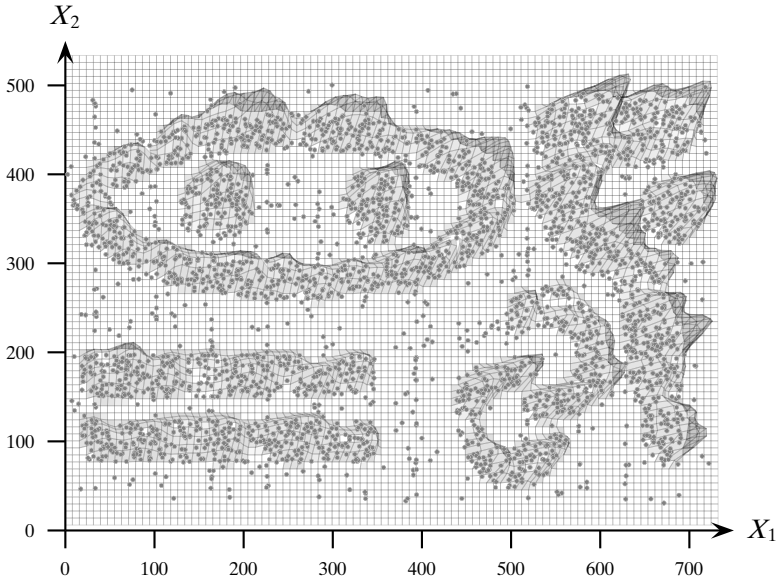


Figure 15.11. DENCLUE: density-based dataset.

discrete kernel DENCLUE yields exactly the same clusters as DBSCAN. Each density attractor corresponds to a core point, and the set of connected core points define the attractors of a density-based cluster. It can also be shown that K-means is a special case of density-based clustering for appropriates value of h and ξ, with the density attractors corresponding to the cluster centroids. Further, it is worth noting that the density-based approach can produce hierarchical clusters, by varying the ξ threshold.

For example, decreasing ξ can result in the merging of several clusters found at higher thresholds values. At the same time it can also lead to new clusters if the peak density satisfies the lower ξ value.

Computational Complexity

The time for DENCLUE is dominated by the cost of the hill-climbing process. For each point $\mathbf{x} \in \mathbf{D}$, finding the density attractor takes $O(nt)$ time, where t is the maximum number of hill-climbing iterations. This is because each iteration takes $O(n)$ time for computing the sum of the influence function over all the points $\mathbf{x}_i \in \mathbf{D}$. The total cost to compute density attractors is therefore $O(n^2 t)$. We assume that for reasonable values of h and ξ, there are only a few density attractors, that is, $|\mathcal{A}| = m \ll n$. The cost of finding the maximal reachable subsets of attractors is $O(m^2)$, and the final clusters can be obtained in $O(n)$ time.

15.4 FURTHER READING

Kernel density estimation was developed independently in Rosenblatt (1956) and Parzen (1962). For an excellent description of density estimation techniques see Silverman (1986). The density-based DBSCAN algorithm was introduced in Ester et al. (1996). The DENCLUE method was proposed in Hinneburg and Keim (1998), with the faster direct update rule appearing in Hinneburg and Gabriel (2007). However, the direct update rule is essentially the *mean-shift* algorithm first proposed in Fukunaga and Hostetler (1975). See Cheng (1995) for convergence properties and generalizations of the mean-shift method.

Cheng, Y. (1995). "Mean shift, mode seeking, and clustering." *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 17 (8): 790–799.

Ester, M., Kriegel, H.-P., Sander, J., and Xu, X. (1996). "A density-based algorithm for discovering clusters in large spatial databases with noise." *In Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining* (pp. 226–231), edited by E. Simoudis, J. Han, and U. M. Fayyad. Palo Ato, CA: AAAI Press.

Fukunaga, K. and Hostetler, L. (1975). "The estimation of the gradient of a density function, with applications in pattern recognition." *IEEE Transactions on Information Theory*, 21 (1): 32–40.

Hinneburg, A. and Gabriel, H.-H. (2007). "Denclue 2.0: Fast clustering based on kernel density estimation." *In Proceedings of the 7th International Symposium on Intelligent Data Analysis* (pp. 70–80). New York: Springer Science+Business Media.

Hinneburg, A. and Keim, D. A. (1998). *"An efficient approach to clustering in large multimedia databases with noise." In Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining* (pp. 58–65), edited by R. Agrawal and P. E. Stolorz. Palo Alto, CA: AAAI Press.

Parzen, E. (1962). On estimation of a probability density function and mode. *The Annals of Mathematical Statistics*, 33 (3): 1065–1076.

Rosenblatt, M. (1956). "Remarks on some nonparametric estimates of a density function." *The Annals of Mathematical Statistics*, 27 (3): 832–837.

Silverman, B. (1986). *Density Estimation for Statistics and Data Analysis*. Monographs on Statistics and Applied Probability. Boca Raton, FL: Chapman and Hall/CRC.

15.5 EXERCISES

Q1. Consider Figure 15.12 and answer the following questions, assuming that we use the Euclidean distance between points, and that $\epsilon = 2$ and $minpts = 3$

- **(a)** List all the core points.
- **(b)** Is a directly density reachable from d?
- **(c)** Is o density reachable from i? Show the intermediate points on the chain or the point where the chain breaks.
- **(d)** Is density reachable a symmetric relationship, that is, if x is density reachable from y, does it imply that y is density reachable from x? Why or why not?
- **(e)** Is l density connected to x? Show the intermediate points that make them density connected or violate the property, respectively.
- **(f)** Is density connected a symmetric relationship?
- **(g)** Show the density-based clusters and the noise points.

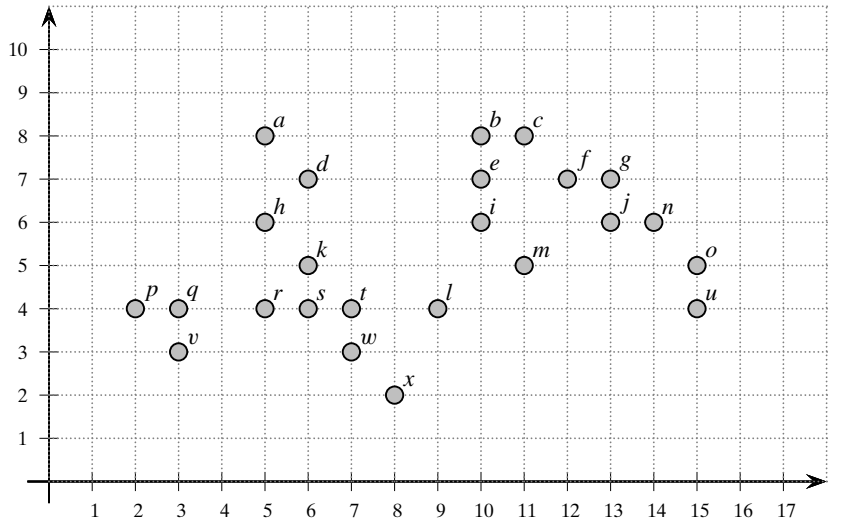


Figure 15.12. Dataset for Q1.

Q2. Consider the points in Figure 15.13. Define the following distance measures:

$$L_\infty(\mathbf{x}, \mathbf{y}) = \max_{i=1}^{d}\{|x_i - y_i|\}$$

$$L_{\frac{1}{2}}(\mathbf{x}, \mathbf{y}) = \Big(\sum_{i=1}^{d} |x_i - y_i|^{\frac{1}{2}}\Big)^2$$

$$L_{\min}(\mathbf{x}, \mathbf{y}) = \min_{i=1}^{d} \{|x_i - y_i|\}$$

$$L_{pow}(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{d} 2^{i-1}(x_i - y_i)^2\right)^{1/2}$$

(a) Using $\epsilon = 2$, $minpts = 5$, and L_∞ distance, find all core, border, and noise points.
(b) Show the shape of the ball of radius $\epsilon = 4$ using the $L_{\frac{1}{2}}$ distance. Using $minpts = 3$ show all the clusters found by DBSCAN.
(c) Using $\epsilon = 1$, $minpts = 6$, and $L_{\min}$, list all core, border, and noise points.
(d) Using $\epsilon = 4$, $minpts = 3$, and L_{pow}, show all clusters found by DBSCAN.

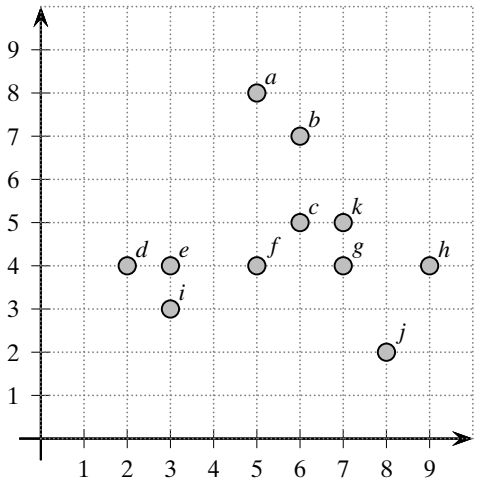


Figure 15.13. Dataset for Q2 and Q3.

Q3. Consider the points shown in Figure 15.13. Define the following two kernels:

$$K_1(\mathbf{z}) = \begin{cases} 1 & \text{If } L_\infty(\mathbf{z}, \mathbf{0}) \leq 1 \\ 0 & \text{Otherwise} \end{cases}$$

$$K_2(\mathbf{z}) = \begin{cases} 1 & \text{If } \sum_{j=1}^{d} |z_j| \leq 1 \\ 0 & \text{Otherwise} \end{cases}$$

Using each of the two kernels K_1 and K_2, answer the following questions assuming that $h = 2$:

(a) What is the probability density at e?
(b) What is the gradient at e?
(c) List all the density attractors for this dataset.

Q4. The Hessian matrix is defined as the set of partial derivatives of the gradient vector with respect to $\mathbf{x}$. What is the Hessian matrix for the Gaussian kernel? Use the gradient in Eq. (15.6).

Q5. Let us compute the probability density at a point x using the k-nearest neighbor approach, given as

$$\hat{f}(x) = \frac{k}{nV_x}$$

where k is the number of nearest neighbors, n is the total number of points, and V_x is the volume of the region encompassing the k nearest neighbors of x. In other words, we fix k and allow the volume to vary based on those k nearest neighbors of x. Given the following points

$$2, 2.5, 3, 4, 4.5, 5, 6.1$$

Find the peak density in this dataset, assuming $k = 4$. Keep in mind that this may happen at a point other than those given above. Also, a point is its own nearest neighbor.

CHAPTER 16 Spectral and Graph Clustering

In this chapter we consider clustering over graph data, that is, given a graph, the goal is to cluster the nodes by using the edges and their weights, which represent the similarity between the incident nodes. Graph clustering is related to divisive hierarchical clustering, as many methods partition the set of nodes to obtain the final clusters using the pairwise similarity matrix between nodes. As we shall see, graph clustering also has a very strong connection to spectral decomposition of graph-based matrices. Finally, if the similarity matrix is positive semidefinite, it can be considered as a kernel matrix, and graph clustering is therefore also related to kernel-based clustering.

16.1 GRAPHS AND MATRICES

Given a dataset $\mathbf{D} = \{\mathbf{x}_i\}_{i=1}^n$ consisting of n points in $\mathbb{R}^d$, let $\mathbf{A}$ denote the $n \times n$ symmetric *similarity matrix* between the points, given as

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \cdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \tag{16.1}$$

where $\mathbf{A}(i, j) = a_{ij}$ denotes the similarity or affinity between points $\mathbf{x}_i$ and $\mathbf{x}_j$. We require the similarity to be symmetric and non-negative, that is, $a_{ij} = a_{ji}$ and $a_{ij} \geq 0$, respectively. The matrix $\mathbf{A}$ may be considered to be a *weighted adjacency matrix* of the weighted (undirected) graph $G = (V, E)$, where each vertex is a point and each edge joins a pair of points, that is,

$$V = \{\mathbf{x}_i \mid i = 1, \ldots, n\}$$
$$E = \big\{(\mathbf{x}_i, \mathbf{x}_j) \mid 1 \leq i, j \leq n\big\}$$

Further, the similarity matrix $\mathbf{A}$ gives the weight on each edge, that is, a_{ij} denotes the weight of the edge $(\mathbf{x}_i, \mathbf{x}_j)$. If all affinities are 0 or 1, then $\mathbf{A}$ represents the regular adjacency relationship between the vertices.

For a vertex $\mathbf{x}_i$, let d_i denote the *degree* of the vertex, defined as

$$d_i = \sum_{j=1}^{n} a_{ij}$$

We define the *degree matrix* $\mathbf{\Delta}$ of graph G as the $n \times n$ diagonal matrix:

$$\mathbf{\Delta} = \begin{pmatrix} d_1 & 0 & \cdots & 0 \\ 0 & d_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & d_n \end{pmatrix} = \begin{pmatrix} \sum_{j=1}^{n} a_{1j} & 0 & \cdots & 0 \\ 0 & \sum_{j=1}^{n} a_{2j} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sum_{j=1}^{n} a_{nj} \end{pmatrix}$$

$\mathbf{\Delta}$ can be compactly written as $\mathbf{\Delta}(i,i) = d_i$ for all $1 \le i \le n$.

Example 16.1. Figure 16.1 shows the similarity graph for the Iris dataset, obtained as follows. Each of the $n = 150$ points $\mathbf{x}_i \in \mathbb{R}^4$ in the Iris dataset is represented by a node in G. To create the edges, we first compute the pairwise similarity between the points using the Gaussian kernel [Eq. (5.10)]:

$$a_{ij} = \exp\left\{-\frac{\left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2}{2\sigma^2}\right\}$$

using $\sigma = 1$. Each edge $(\mathbf{x}_i, \mathbf{x}_j)$ has the weight a_{ij}. Next, for each node $\mathbf{x}_i$ we compute the top q nearest neighbors in terms of the similarity value, given as

$$N_q(\mathbf{x}_i) = \left\{\mathbf{x}_j \in V : a_{ij} \le a_{iq}\right\}$$

where a_{iq} represents the similarity value between $\mathbf{x}_i$ and its qth nearest neighbor. We used a value of $q = 16$, as in this case each node records at least 15 nearest neighbors (not including the node itself), which corresponds to 10% of the nodes. An edge is added between nodes $\mathbf{x}_i$ and $\mathbf{x}_j$ if and only if both nodes are *mutual nearest neighbors*, that is, if $\mathbf{x}_j \in N_q(\mathbf{x}_i)$ and $\mathbf{x}_i \in N_q(\mathbf{x}_j)$. Finally, if the resulting graph is disconnected, we add the top q most similar (i.e., highest weighted) edges between any two connected components.

The resulting Iris similarity graph is shown in Figure 16.1. It has $|V| = n = 150$ nodes and $|E| = m = 1730$ edges. Edges with similarity $a_{ij} \ge 0.9$ are shown in black, and the remaining edges are shown in gray. Although $a_{ii} = 1.0$ for all nodes, we do not show the self-edges or loops.

Normalized Adjacency Matrix

The normalized adjacency matrix is obtained by dividing each row of the adjacency matrix by the degree of the corresponding node. Given the weighted adjacency matrix

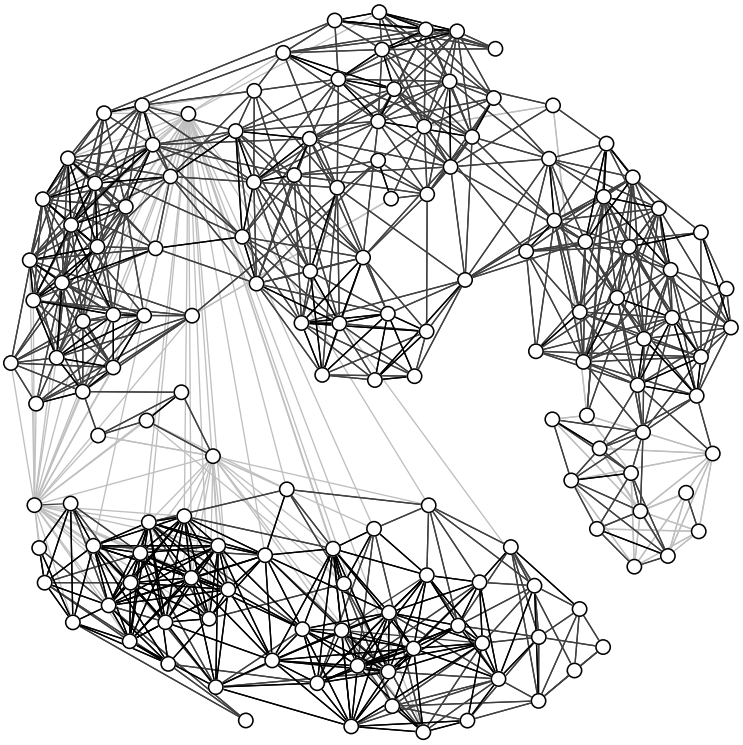


Figure 16.1. Iris similarity graph.

$\mathbf{A}$ for a graph G, its normalized adjacency matrix is defined as

$$\mathbf{M} = \mathbf{\Delta}^{-1}\mathbf{A} = \begin{pmatrix} \frac{a_{11}}{d_1} & \frac{a_{12}}{d_1} & \cdots & \frac{a_{1n}}{d_1} \\ \frac{a_{21}}{d_2} & \frac{a_{22}}{d_2} & \cdots & \frac{a_{2n}}{d_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{a_{n1}}{d_n} & \frac{a_{n2}}{d_n} & \cdots & \frac{a_{nn}}{d_n} \end{pmatrix} \tag{16.2}$$

Because $\mathbf{A}$ is assumed to have non-negative elements, this implies that each element of $\mathbf{M}$, namely m_{ij} is also non-negative, as $m_{ij} = \frac{a_{ij}}{d_i} \geq 0$. Consider the sum of the ith row in $\mathbf{M}$; we have

$$\sum_{j=1}^{n} m_{ij} = \sum_{j=1}^{n} \frac{a_{ij}}{d_i} = \frac{d_i}{d_i} = 1 \tag{16.3}$$

Thus, each row in $\mathbf{M}$ sums to 1. This implies that 1 is an eigenvalue of $\mathbf{M}$. In fact, $\lambda_1 = 1$ is the largest eigenvalue of $\mathbf{M}$, and the other eigenvalues satisfy the property that $|\lambda_i| \leq 1$. Also, if G is connected then the eigenvector corresponding to λ_1 is $\mathbf{u}_1 = \frac{1}{\sqrt{n}}(1, 1, \ldots, 1)^T = \frac{1}{\sqrt{n}}\mathbf{1}$. Because $\mathbf{M}$ is not symmetric, its eigenvectors are not necessarily orthogonal.

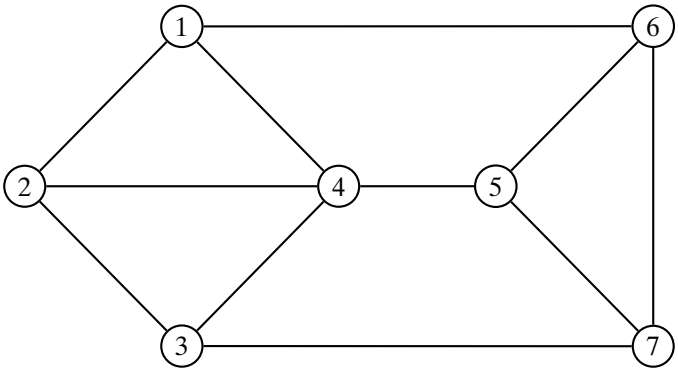


Figure 16.2. Example graph.

Example 16.2. Consider the graph in Figure 16.2. Its adjacency and degree matrices are given as

$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{pmatrix} \qquad \mathbf{\Delta} = \begin{pmatrix} 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 3 \end{pmatrix}$$

The normalized adjacency matrix is as follows:

$$\mathbf{M} = \mathbf{\Delta}^{-1}\mathbf{A} = \begin{pmatrix} 0 & 0.33 & 0 & 0.33 & 0 & 0.33 & 0 \\ 0.33 & 0 & 0.33 & 0.33 & 0 & 0 & 0 \\ 0 & 0.33 & 0 & 0.33 & 0 & 0 & 0.33 \\ 0.25 & 0.25 & 0.25 & 0 & 0.25 & 0 & 0 \\ 0 & 0 & 0 & 0.33 & 0 & 0.33 & 0.33 \\ 0.33 & 0 & 0 & 0 & 0.33 & 0 & 0.33 \\ 0 & 0 & 0.33 & 0 & 0.33 & 0.33 & 0 \end{pmatrix}$$

The eigenvalues of $\mathbf{M}$ sorted in decreasing order are as follows:

$$\lambda_1 = 1 \qquad \lambda_2 = 0.483 \qquad \lambda_3 = 0.206 \qquad \lambda_4 = -0.045$$
$$\lambda_5 = -0.405 \qquad \lambda_6 = -0.539 \qquad \lambda_7 = -0.7$$

The eigenvector corresponding to $\lambda_1 = 1$ is

$$\mathbf{u}_1 = \frac{1}{\sqrt{7}}(1,1,1,1,1,1,1)^T = (0.38, 0.38, 0.38, 0.38, 0.38, 0.38, 0.38)^T$$

Graph Laplacian Matrices

The *Laplacian matrix* of a graph is defined as

$$\begin{aligned}\mathbf{L} &= \mathbf{\Delta} - \mathbf{A} \\ &= \begin{pmatrix} \sum_{j=1}^{n} a_{1j} & 0 & \cdots & 0 \\ 0 & \sum_{j=1}^{n} a_{2j} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sum_{j=1}^{n} a_{nj} \end{pmatrix} - \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \cdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \\ &= \begin{pmatrix} \sum_{j\neq 1} a_{1j} & -a_{12} & \cdots & -a_{1n} \\ -a_{21} & \sum_{j\neq 2} a_{2j} & \cdots & -a_{2n} \\ \vdots & \vdots & \cdots & \vdots \\ -a_{n1} & -a_{n2} & \cdots & \sum_{j\neq n} a_{nj} \end{pmatrix} \end{aligned} \tag{16.4}$$

It is interesting to note that $\mathbf{L}$ is a symmetric, positive semidefinite matrix, as for any $\mathbf{c} \in \mathbb{R}^n$, we have

$$\begin{aligned}\mathbf{c}^T\mathbf{L}\mathbf{c} &= \mathbf{c}^T(\mathbf{\Delta} - \mathbf{A})\mathbf{c} = \mathbf{c}^T\mathbf{\Delta}\mathbf{c} - \mathbf{c}^T\mathbf{A}\mathbf{c} \\ &= \sum_{i=1}^{n} d_i c_i^2 - \sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j a_{ij} \\ &= \frac{1}{2}\left(\sum_{i=1}^{n} d_i c_i^2 - 2\sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j a_{ij} + \sum_{j=1}^{n} d_j c_j^2\right) \\ &= \frac{1}{2}\left(\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij} c_i^2 - 2\sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j a_{ij} + \sum_{i=j}^{n}\sum_{i=1}^{n} a_{ij} c_j^2\right) \\ &= \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}(c_i - c_j)^2 \\ &\geq 0 \qquad \text{because } a_{ij} \geq 0 \text{ and } (c_i - c_j)^2 \geq 0 \end{aligned} \tag{16.5}$$

This means that $\mathbf{L}$ has n real, non-negative eigenvalues, which can be arranged in decreasing order as follows: $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \geq 0$. Because $\mathbf{L}$ is symmetric, its eigenvectors are orthonormal. Further, from Eq. (16.4) we can see that the first column (and the first row) is a linear combination of the remaining columns (rows). That is, if L_i denotes the ith column of $\mathbf{L}$, then we can observe that $L_1 + L_2 + L_3 + \cdots + L_n = \mathbf{0}$. This implies that the rank of $\mathbf{L}$ is at most $n-1$, and the smallest eigenvalue is $\lambda_n = 0$, with the corresponding eigenvector given as $\mathbf{u}_n = \frac{1}{\sqrt{n}}(1, 1, \ldots, 1)^T = \frac{1}{\sqrt{n}}\mathbf{1}$, provided the graph is connected. If the graph is disconnected, then the number of eigenvalues equal to zero specifies the number of connected components in the graph.

Example 16.3. Consider the graph in Figure 16.2, whose adjacency and degree matrices are shown in Example 16.2. The graph Laplacian is given as

$$\mathbf{L} = \mathbf{\Delta} - \mathbf{A} = \begin{pmatrix} 3 & -1 & 0 & -1 & 0 & -1 & 0 \\ -1 & 3 & -1 & -1 & 0 & 0 & 0 \\ 0 & -1 & 3 & -1 & 0 & 0 & -1 \\ -1 & -1 & -1 & 4 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 3 & -1 & -1 \\ -1 & 0 & 0 & 0 & -1 & 3 & -1 \\ 0 & 0 & -1 & 0 & -1 & -1 & 3 \end{pmatrix}$$

The eigenvalues of $\mathbf{L}$ are as follows:

$$\lambda_1 = 5.618 \qquad \lambda_2 = 4.618 \qquad \lambda_3 = 4.414 \qquad \lambda_4 = 3.382$$
$$\lambda_5 = 2.382 \qquad \lambda_6 = 1.586 \qquad \lambda_7 = 0$$

The eigenvector corresponding to $\lambda_7 = 0$ is

$$\mathbf{u}_7 = \frac{1}{\sqrt{7}}(1,1,1,1,1,1,1)^T = (0.38, 0.38, 0.38, 0.38, 0.38, 0.38, 0.38)^T$$

The *normalized symmetric Laplacian matrix* of a graph is defined as

$$\begin{aligned} \mathbf{L}^s &= \mathbf{\Delta}^{-1/2}\mathbf{L}\mathbf{\Delta}^{-1/2} \\ &= \mathbf{\Delta}^{-1/2}(\mathbf{\Delta} - \mathbf{A})\mathbf{\Delta}^{-1/2} = \mathbf{\Delta}^{-1/2}\mathbf{\Delta}\mathbf{\Delta}^{-1/2} - \mathbf{\Delta}^{-1/2}\mathbf{A}\mathbf{\Delta}^{-1/2} \\ &= \mathbf{I} - \mathbf{\Delta}^{-1/2}\mathbf{A}\mathbf{\Delta}^{-1/2} \end{aligned} \tag{16.6}$$

where $\mathbf{\Delta}^{1/2}$ is the diagonal matrix given as $\mathbf{\Delta}^{1/2}(i,i) = \sqrt{d_i}$, and $\mathbf{\Delta}^{-1/2}$ is the diagonal matrix given as $\mathbf{\Delta}^{-1/2}(i,i) = \frac{1}{\sqrt{d_i}}$ (assuming that $d_i \neq 0$), for $1 \leq i \leq n$. In other words, the normalized Laplacian is given as

$$\begin{aligned} \mathbf{L}^s &= \mathbf{\Delta}^{-1/2}\mathbf{L}\mathbf{\Delta}^{-1/2} \\ &= \begin{pmatrix} \frac{\sum_{j\neq 1} a_{1j}}{\sqrt{d_1 d_1}} & -\frac{a_{12}}{\sqrt{d_1 d_2}} & \cdots & -\frac{a_{1n}}{\sqrt{d_1 d_n}} \\ -\frac{a_{21}}{\sqrt{d_2 d_1}} & \frac{\sum_{j\neq 2} a_{2j}}{\sqrt{d_2 d_2}} & \cdots & -\frac{a_{2n}}{\sqrt{d_2 d_n}} \\ \vdots & \vdots & \ddots & \vdots \\ -\frac{a_{n1}}{\sqrt{d_n d_1}} & -\frac{a_{n2}}{\sqrt{d_n d_2}} & \cdots & \frac{\sum_{j\neq n} a_{nj}}{\sqrt{d_n d_n}} \end{pmatrix} \end{aligned} \tag{16.7}$$

Like the derivation in Eq. (16.5), we can show that $\mathbf{L}^s$ is also positive semidefinite because for any $\mathbf{c} \in \mathbb{R}^d$, we get

$$\mathbf{c}^T\mathbf{L}^s\mathbf{c} = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}\left(\frac{c_i}{\sqrt{d_i}} - \frac{c_j}{\sqrt{d_j}}\right)^2 \geq 0 \tag{16.8}$$

Further, if L_i^s denotes the ith column of $\mathbf{L}^s$, then from Eq. (16.7) we can see that

$$\sqrt{d_1}L_1^s + \sqrt{d_2}L_2^s + \sqrt{d_3}L_3^s + \cdots + \sqrt{d_n}L_n^s = \mathbf{0}$$

That is, the first column is a linear combination of the other columns, which means that $\mathbf{L}^s$ has rank at most $n-1$, with the smallest eigenvalue $\lambda_n = 0$, and the corresponding eigenvector $\frac{1}{\sqrt{\sum_i d_i}}(\sqrt{d_1}, \sqrt{d_2}, \ldots, \sqrt{d_n})^T = \frac{1}{\sqrt{\sum_i d_i}}\boldsymbol{\Delta}^{1/2}\mathbf{1}$. Combined with the fact that $\mathbf{L}^s$ is positive semidefinite, we conclude that $\mathbf{L}^s$ has n (not necessarily distinct) real, positive eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n = 0$.

Example 16.4. We continue with Example 16.3. For the graph in Figure 16.2, its normalized symmetric Laplacian is given as

$$\mathbf{L}^s = \begin{pmatrix} 1 & -0.33 & 0 & -0.29 & 0 & -0.33 & 0 \\ -0.33 & 1 & -0.33 & -0.29 & 0 & 0 & 0 \\ 0 & -0.33 & 1 & -0.29 & 0 & 0 & -0.33 \\ -0.29 & -0.29 & -0.29 & 1 & -0.29 & 0 & 0 \\ 0 & 0 & 0 & -0.29 & 1 & -0.33 & -0.33 \\ -0.33 & 0 & 0 & 0 & -0.33 & 1 & -0.33 \\ 0 & 0 & -0.33 & 0 & -0.33 & -0.33 & 1 \end{pmatrix}$$

The eigenvalues of $\mathbf{L}^s$ are as follows:

$$\lambda_1 = 1.7 \qquad \lambda_2 = 1.539 \qquad \lambda_3 = 1.405 \qquad \lambda_4 = 1.045$$
$$\lambda_5 = 0.794 \qquad \lambda_6 = 0.517 \qquad \lambda_7 = 0$$

The eigenvector corresponding to $\lambda_7 = 0$ is

$$\begin{aligned} \mathbf{u}_7 &= \frac{1}{\sqrt{22}}(\sqrt{3}, \sqrt{3}, \sqrt{3}, \sqrt{4}, \sqrt{3}, \sqrt{3}, \sqrt{3})^T \\ &= (0.37, 0.37, 0.37, 0.43, 0.37, 0.37, 0.37)^T \end{aligned}$$

The *normalized asymmetric Laplacian* matrix is defined as

$$\begin{aligned} \mathbf{L}^a &= \boldsymbol{\Delta}^{-1}\mathbf{L} \\ &= \boldsymbol{\Delta}^{-1}(\boldsymbol{\Delta} - \mathbf{A}) = \mathbf{I} - \boldsymbol{\Delta}^{-1}\mathbf{A} \\ &= \begin{pmatrix} \frac{\sum_{j\neq 1} a_{1j}}{d_1} & -\frac{a_{12}}{d_1} & \cdots & -\frac{a_{1n}}{d_1} \\ -\frac{a_{21}}{d_2} & \frac{\sum_{j\neq 2} a_{2j}}{d_2} & \cdots & -\frac{a_{2n}}{d_2} \\ \vdots & \vdots & \ddots & \vdots \\ -\frac{a_{n1}}{d_n} & -\frac{a_{n2}}{d_n} & \cdots & \frac{\sum_{j\neq n} a_{nj}}{d_n} \end{pmatrix} \end{aligned} \tag{16.9}$$

Consider the eigenvalue equation for the symmetric Laplacian $\mathbf{L}^s$:

$$\mathbf{L}^s\mathbf{u} = \lambda\mathbf{u}$$

Left multiplying by $\mathbf{\Delta}^{-1/2}$ on both sides, we get

$$\begin{aligned}
\mathbf{\Delta}^{-1/2}\mathbf{L}^s\mathbf{u} &= \lambda\mathbf{\Delta}^{-1/2}\mathbf{u} \\
\mathbf{\Delta}^{-1/2}\left(\mathbf{\Delta}^{-1/2}\mathbf{L}\mathbf{\Delta}^{-1/2}\right)\mathbf{u} &= \lambda\mathbf{\Delta}^{-1/2}\mathbf{u} \\
\mathbf{\Delta}^{-1}\mathbf{L}\left(\mathbf{\Delta}^{-1/2}\mathbf{u}\right) &= \lambda\left(\mathbf{\Delta}^{-1/2}\mathbf{u}\right) \\
\mathbf{L}^a\mathbf{v} &= \lambda\mathbf{v}
\end{aligned}$$

where $\mathbf{v} = \mathbf{\Delta}^{-1/2}\mathbf{u}$ is an eigenvector of $\mathbf{L}^a$, and $\mathbf{u}$ is an eigenvector of $\mathbf{L}^s$. Further, $\mathbf{L}^a$ has the same set of eigenvalues as $\mathbf{L}^s$, which means that $\mathbf{L}^a$ is a positive semi-definite matrix with n real eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n = 0$. From Eq. (16.9) we can see that if L_i^a denotes the ith column of $\mathbf{L}^a$, then $L_1^a + L_2^a + \cdots + L_n^a = 0$, which implies that $\mathbf{v}_n = \frac{1}{\sqrt{n}}\mathbf{1}$ is the eigenvector corresponding to the smallest eigenvalue $\lambda_n = 0$.

Example 16.5. For the graph in Figure 16.2, its normalized asymmetric Laplacian matrix is given as

$$\mathbf{L}^a = \mathbf{\Delta}^{-1}\mathbf{L} = \begin{pmatrix}
1 & -0.33 & 0 & -0.33 & 0 & -0.33 & 0 \\
-0.33 & 1 & -0.33 & -0.33 & 0 & 0 & 0 \\
0 & -0.33 & 1 & -0.33 & 0 & 0 & -0.33 \\
-0.25 & -0.25 & -0.25 & 1 & -0.25 & 0 & 0 \\
0 & 0 & 0 & -0.33 & 1 & -0.33 & -0.33 \\
-0.33 & 0 & 0 & 0 & -0.33 & 1 & -0.33 \\
0 & 0 & -0.33 & 0 & -0.33 & -0.33 & 1
\end{pmatrix}$$

The eigenvalues of $\mathbf{L}^a$ are identical to those for $\mathbf{L}^s$, namely

$$\lambda_1 = 1.7 \qquad \lambda_2 = 1.539 \qquad \lambda_3 = 1.405 \qquad \lambda_4 = 1.045$$
$$\lambda_5 = 0.794 \qquad \lambda_6 = 0.517 \qquad \lambda_7 = 0$$

The eigenvector corresponding to $\lambda_7 = 0$ is

$$\mathbf{u}_7 = \frac{1}{\sqrt{7}}(1, 1, 1, 1, 1, 1, 1)^T = (0.38, 0.38, 0.38, 0.38, 0.38, 0.38, 0.38)^T$$

16.2 CLUSTERING AS GRAPH CUTS

A *k-way cut* in a graph is a partitioning or clustering of the vertex set, given as $\mathcal{C} = \{C_1, \ldots, C_k\}$, such that $C_i \neq \emptyset$ for all i, $C_i \cap C_j = \emptyset$ for all i, j, and $V = \bigcup_i C_i$. We require $\mathcal{C}$ to optimize some objective function that captures the intuition that nodes within a cluster should have high similarity, and nodes from different clusters should have low similarity.

Given a weighted graph G defined by its similarity matrix [Eq. (16.1)], let $S, T \subseteq V$ be any two subsets of the vertices. We denote by $W(S, T)$ the sum of the weights on all

edges with one vertex in S and the other in T, given as

$$W(S,T) = \sum_{v_i \in S} \sum_{v_j \in T} a_{ij}$$

Given $S \subseteq V$, we denote by $\overline{S}$ the complementary set of vertices, that is, $\overline{S} = V - S$. A *(vertex) cut* in a graph is defined as a partitioning of V into $S \subset V$ and $\overline{S}$. The *weight of the cut* or *cut weight* is defined as the sum of all the weights on edges between vertices in S and $\overline{S}$, given as $W(S, \overline{S})$.

Given a clustering $\mathcal{C} = \{C_1, \ldots, C_k\}$ comprising k clusters, the *size* of a cluster C_i is the number of nodes in the cluster, given as $|C_i|$. The *volume* of a cluster C_i is defined as the sum of all the weights on edges with one end in cluster C_i:

$$vol(C_i) = \sum_{v_j \in C_i} d_j = \sum_{v_j \in C_i} \sum_{v_r \in V} a_{jr} = W(C_i, V)$$

Let $\mathbf{c}_i \in \{0,1\}^n$ be the *cluster indicator vector* that records the cluster membership for cluster C_i, defined as

$$c_{ij} = \begin{cases} 1 & \text{if } v_j \in C_i \\ 0 & \text{if } v_j \notin C_i \end{cases}$$

Because a clustering creates pairwise disjoint clusters, we immediately have

$$\mathbf{c}_i^T \mathbf{c}_j = 0$$

Further, the cluster size can be written as

$$|C_i| = \mathbf{c}_i^T \mathbf{c}_i = \|\mathbf{c}_i\|^2$$

The following identities allow us to express the weight of a cut in terms of matrix operations. Let us derive an expression for the sum of the weights for all edges with one end in C_i. These edges include internal cluster edges (with both ends in C_i), as well as external cluster edges (with the other end in another cluster $C_{j \neq i}$).

$$\begin{aligned} vol(C_i) = W(C_i, V) &= \sum_{v_r \in C_i} d_r = \sum_{v_r \in C_i} c_{ir} d_r c_{ir} \\ &= \sum_{r=1}^{n} \sum_{s=1}^{n} c_{ir} \mathbf{\Delta}_{rs} c_{is} = \mathbf{c}_i^T \mathbf{\Delta} \mathbf{c}_i \end{aligned} \tag{16.10}$$

Consider the sum of weights of all internal edges:

$$\begin{aligned} W(C_i, C_i) &= \sum_{v_r \in C_i} \sum_{v_s \in C_i} a_{rs} \\ &= \sum_{r=1}^{n} \sum_{s=1}^{n} c_{ir} a_{rs} c_{is} = \mathbf{c}_i^T \mathbf{A} \mathbf{c}_i \end{aligned} \tag{16.11}$$

We can get the sum of weights for all the external edges, or the cut weight by subtracting Eq. (16.11) from Eq. (16.10), as follows:

$$W(C_i, \overline{C_i}) = \sum_{v_r \in C_i} \sum_{v_s \in V - C_i} a_{rs} = W(C_i, V) - W(C_i, C_i)$$
$$= \mathbf{c}_i(\mathbf{\Delta} - \mathbf{A})\mathbf{c}_i = \mathbf{c}_i^T \mathbf{L} \mathbf{c}_i \tag{16.12}$$

Example 16.6. Consider the graph in Figure 16.2. Assume that $C_1 = \{1, 2, 3, 4\}$ and $C_2 = \{5, 6, 7\}$ are two clusters. Their cluster indicator vectors are given as

$$\mathbf{c}_1 = (1, 1, 1, 1, 0, 0, 0)^T \qquad \mathbf{c}_2 = (0, 0, 0, 0, 1, 1, 1)^T$$

As required, we have $\mathbf{c}_1^T \mathbf{c}_2 = 0$, and $\mathbf{c}_1^T \mathbf{c}_1 = \|\mathbf{c}_1\|^2 = 4$ and $\mathbf{c}_2^T \mathbf{c}_2 = 3$ give the cluster sizes. Consider the cut weight between C_1 and C_2. Because there are three edges between the two clusters, we have $W(C_1, \overline{C_1}) = W(C_1, C_2) = 3$. Using the Laplacian matrix from Example 16.3, by Eq. (16.12) we have

$$W(C_1, \overline{C_1}) = \mathbf{c}_1^T \mathbf{L} \mathbf{c}_1$$
$$= (1, 1, 1, 1, 0, 0, 0) \begin{pmatrix} 3 & -1 & 0 & -1 & 0 & -1 & 0 \\ -1 & 3 & -1 & -1 & 0 & 0 & 0 \\ 0 & -1 & 3 & -1 & 0 & 0 & -1 \\ -1 & -1 & -1 & 4 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 3 & -1 & -1 \\ -1 & 0 & 0 & 0 & -1 & 3 & -1 \\ 0 & 0 & -1 & 0 & -1 & -1 & 3 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$
$$= (1, 0, 1, 1, -1, -1, -1)(1, 1, 1, 1, 0, 0, 0)^T = 3$$

16.2.1 Clustering Objective Functions: Ratio and Normalized Cut

The clustering objective function can be formulated as an optimization problem over the k-way cut $\mathcal{C} = \{C_1, \ldots, C_k\}$. We consider two common minimization objectives, namely ratio and normalized cut. We consider maximization objectives in Section 16.2.3, after describing the spectral clustering algorithm.

Ratio Cut

The *ratio cut* objective is defined over a k-way cut as follows:

$$\min_{\mathcal{C}} J_{rc}(\mathcal{C}) = \sum_{i=1}^{k} \frac{W(C_i, \overline{C_i})}{|C_i|} = \sum_{i=1}^{k} \frac{\mathbf{c}_i^T \mathbf{L} \mathbf{c}_i}{\mathbf{c}_i^T \mathbf{c}_i} = \sum_{i=1}^{k} \frac{\mathbf{c}_i^T \mathbf{L} \mathbf{c}_i}{\|\mathbf{c}_i\|^2} \tag{16.13}$$

where we make use of Eq. (16.12), that is, $W(C_i, \overline{C_i}) = \mathbf{c}_i^T \mathbf{L} \mathbf{c}_i$.

Ratio cut tries to minimize the sum of the similarities from a cluster C_i to other points not in the cluster $\overline{C_i}$, taking into account the size of each cluster. One can observe that the objective function has a lower value when the cut weight is minimized and when the cluster size is large.

Unfortunately, for binary cluster indicator vectors $\mathbf{c}_i$, the ratio cut objective is NP-hard. An obvious relaxation is to allow $\mathbf{c}_i$ to take on any real value. In this case, we can rewrite the objective as

$$\min_{\mathcal{C}} J_{rc}(\mathcal{C}) = \sum_{i=1}^{k} \frac{\mathbf{c}_i^T \mathbf{L} \mathbf{c}_i}{\|\mathbf{c}_i\|^2} = \sum_{i=1}^{k} \left(\frac{\mathbf{c}_i}{\|\mathbf{c}_i\|}\right)^T \mathbf{L} \left(\frac{\mathbf{c}_i}{\|\mathbf{c}_i\|}\right) = \sum_{i=1}^{k} \mathbf{u}_i^T \mathbf{L} \mathbf{u}_i \tag{16.14}$$

where $\mathbf{u}_i = \frac{\mathbf{c}_i}{\|\mathbf{c}_i\|}$ is the unit vector in the direction of $\mathbf{c}_i \in \mathbb{R}^n$, that is, $\mathbf{c}_i$ is assumed to be an arbitrary real vector.

To minimize J_{rc} we take its derivative with respect to $\mathbf{u}_i$ and set it to the zero vector. To incorporate the constraint that $\mathbf{u}_i^T \mathbf{u}_i = 1$, we introduce the Lagrange multiplier λ_i for each cluster C_i. We have

$$\frac{\partial}{\partial \mathbf{u}_i} \left(\sum_{i=1}^{k} \mathbf{u}_i^T \mathbf{L} \mathbf{u}_i + \sum_{i=1}^{n} \lambda_i (1 - \mathbf{u}_i^T \mathbf{u}_i) \right) = \mathbf{0}, \text{ which implies that}$$

$$2\mathbf{L}\mathbf{u}_i - 2\lambda_i \mathbf{u}_i = \mathbf{0}, \text{ and thus}$$

$$\mathbf{L}\mathbf{u}_i = \lambda_i \mathbf{u}_i \tag{16.15}$$

This implies that $\mathbf{u}_i$ is one of the eigenvectors of the Laplacian matrix $\mathbf{L}$, corresponding to the eigenvalue λ_i. Using Eq. (16.15), we can see that

$$\mathbf{u}_i^T \mathbf{L} \mathbf{u}_i = \mathbf{u}_i^T \lambda_i \mathbf{u}_i = \lambda_i$$

which in turn implies that to minimize the ratio cut objective [Eq. (16.14)], we should choose the k smallest eigenvalues, and the corresponding eigenvectors, so that

$$\begin{aligned} \min_{\mathcal{C}} J_{rc}(\mathcal{C}) &= \mathbf{u}_n^T \mathbf{L} \mathbf{u}_n + \cdots + \mathbf{u}_{n-k+1}^T \mathbf{L} \mathbf{u}_{n-k+1} \\ &= \lambda_n + \cdots + \lambda_{n-k+1} \end{aligned} \tag{16.16}$$

where we assume that the eigenvalues have been sorted so that $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$. Noting that the smallest eigenvalue of $\mathbf{L}$ is $\lambda_n = 0$, the k smallest eigenvalues are as follows: $0 = \lambda_n \leq \lambda_{n-1} \leq \lambda_{n-k+1}$. The corresponding eigenvectors $\mathbf{u}_n, \mathbf{u}_{n-1}, \ldots, \mathbf{u}_{n-k+1}$ represent the relaxed cluster indicator vectors. However, because $\mathbf{u}_n = \frac{1}{\sqrt{n}}\mathbf{1}$, it does not provide any guidance on how to separate the graph nodes if the graph is connected.

Normalized Cut

Normalized cut is similar to ratio cut, except that it divides the cut weight of each cluster by the volume of a cluster instead of its size. The objective function is given as

$$\min_{\mathcal{C}} J_{nc}(\mathcal{C}) = \sum_{i=1}^{k} \frac{W(C_i, \overline{C_i})}{vol(C_i)} = \sum_{i=1}^{k} \frac{\mathbf{c}_i^T \mathbf{L} \mathbf{c}_i}{\mathbf{c}_i^T \mathbf{\Delta} \mathbf{c}_i} \tag{16.17}$$

where we use Eqs. (16.12) and (16.10), that is, $W(C_i, \overline{C_i}) = \mathbf{c}_i^T \mathbf{L} \mathbf{c}_i$ and $vol(C_i) = \mathbf{c}_i^T \mathbf{\Delta} \mathbf{c}_i$, respectively. The J_{nc} objective function has lower values when the cut weight is low and when the cluster volume is high, as desired.

As in the case of ratio cut, we can obtain an optimal solution to the normalized cut objective if we relax the condition that $\mathbf{c}_i$ be a binary cluster indicator vector. Instead we assume $\mathbf{c}_i$ to be an arbitrary real vector. Using the observation that the diagonal degree matrix $\mathbf{\Delta}$ can be written as $\mathbf{\Delta} = \mathbf{\Delta}^{1/2}\mathbf{\Delta}^{1/2}$, and using the fact that $\mathbf{I} = \mathbf{\Delta}^{1/2}\mathbf{\Delta}^{-1/2}$ and $\mathbf{\Delta}^T = \mathbf{\Delta}$ (because $\mathbf{\Delta}$ is diagonal), we can rewrite the normalized cut objective in terms of the normalized symmetric Laplacian, as follows:

$$\begin{aligned}
\min_{\mathcal{C}} J_{nc}(\mathcal{C}) &= \sum_{i=1}^{k} \frac{\mathbf{c}_i^T \mathbf{L} \mathbf{c}_i}{\mathbf{c}_i^T \mathbf{\Delta} \mathbf{c}_i} \\
&= \sum_{i=1}^{k} \frac{\mathbf{c}_i^T \big(\mathbf{\Delta}^{1/2}\mathbf{\Delta}^{-1/2}\big)\mathbf{L}\big(\mathbf{\Delta}^{-1/2}\mathbf{\Delta}^{1/2}\big)\mathbf{c}_i}{\mathbf{c}_i^T \big(\mathbf{\Delta}^{1/2}\mathbf{\Delta}^{1/2}\big)\mathbf{c}_i} \\
&= \sum_{i=1}^{k} \frac{(\mathbf{\Delta}^{1/2}\mathbf{c}_i)^T(\mathbf{\Delta}^{-1/2}\mathbf{L}\mathbf{\Delta}^{-1/2})(\mathbf{\Delta}^{1/2}\mathbf{c}_i)}{(\mathbf{\Delta}^{1/2}\mathbf{c}_i)^T(\mathbf{\Delta}^{1/2}\mathbf{c}_i)} \\
&= \sum_{i=1}^{k} \left(\frac{\mathbf{\Delta}^{1/2}\mathbf{c}_i}{\big\|\mathbf{\Delta}^{1/2}\mathbf{c}_i\big\|}\right)^T \mathbf{L}^s \left(\frac{\mathbf{\Delta}^{1/2}\mathbf{c}_i}{\big\|\mathbf{\Delta}^{1/2}\mathbf{c}_i\big\|}\right) \\
&= \sum_{i=1}^{k} \mathbf{u}_i^T \mathbf{L}^s \mathbf{u}_i
\end{aligned}$$

where $\mathbf{u}_i = \frac{\mathbf{\Delta}^{1/2}\mathbf{c}_i}{\|\mathbf{\Delta}^{1/2}\mathbf{c}_i\|}$ is the unit vector in the direction of $\mathbf{\Delta}^{1/2}\mathbf{c}_i$. Following the same approach as in Eq. (16.15), we conclude that the normalized cut objective is optimized by selecting the k smallest eigenvalues of the normalized Laplacian matrix $\mathbf{L}^s$, namely $0 = \lambda_n \leq \cdots \leq \lambda_{n-k+1}$.

The normalized cut objective [Eq. (16.17)], can also be expressed in terms of the normalized asymmetric Laplacian, by differentiating Eq. (16.17) with respect to $\mathbf{c}_i$ and setting the result to the zero vector. Noting that all terms other than that for $\mathbf{c}_i$ are constant with respect to $\mathbf{c}_i$, we have:

$$\begin{aligned}
\frac{\partial}{\partial \mathbf{c}_i}\left(\sum_{j=1}^{k} \frac{\mathbf{c}_j^T \mathbf{L} \mathbf{c}_j}{\mathbf{c}_j^T \mathbf{\Delta} \mathbf{c}_j}\right) = \frac{\partial}{\partial \mathbf{c}_i}\left(\frac{\mathbf{c}_i^T \mathbf{L} \mathbf{c}_i}{\mathbf{c}_i^T \mathbf{\Delta} \mathbf{c}_i}\right) &= \mathbf{0} \\
\frac{\mathbf{L}\mathbf{c}_i(\mathbf{c}_i^T \mathbf{\Delta} \mathbf{c}_i) - \mathbf{\Delta}\mathbf{c}_i(\mathbf{c}_i^T \mathbf{L} \mathbf{c}_i)}{(\mathbf{c}_i^T \mathbf{\Delta} \mathbf{c}_i)^2} &= \mathbf{0} \\
\mathbf{L}\mathbf{c}_i = \left(\frac{\mathbf{c}_i^T \mathbf{L} \mathbf{c}_i}{\mathbf{c}_i^T \mathbf{\Delta} \mathbf{c}_i}\right)\mathbf{\Delta}\mathbf{c}_i & \\
\mathbf{\Delta}^{-1}\mathbf{L}\mathbf{c}_i = \lambda_i \mathbf{c}_i & \\
\mathbf{L}^a \mathbf{c}_i = \lambda_i \mathbf{c}_i &
\end{aligned}$$

where $\lambda_i = \frac{\mathbf{c}_i^T \mathbf{L} \mathbf{c}_i}{\mathbf{c}_i^T \mathbf{\Delta} \mathbf{c}_i}$ is the eigenvalue corresponding to the ith eigenvector $\mathbf{c}_i$ of the asymmetric Laplacian matrix $\mathbf{L}^a$. To minimize the normalized cut objective we therefore choose the k smallest eigenvalues of $\mathbf{L}^a$, namely, $0 = \lambda_n \leq \cdots \leq \lambda_{n-k+1}$.

To derive the clustering, for $\mathbf{L}^a$, we can use the corresponding eigenvectors $\mathbf{u}_n, \ldots, \mathbf{u}_{n-k+1}$, with $\mathbf{c}_i = \mathbf{u}_i$ representing the real-valued cluster indicator vectors.

However, note that for $\mathbf{L}^a$, we have $\mathbf{c}_n = \mathbf{u}_n = \frac{1}{\sqrt{n}}\mathbf{1}$. Further, for the normalized symmetric Laplacian $\mathbf{L}^s$, the real-valued cluster indicator vectors are given as $\mathbf{c}_i = \mathbf{\Delta}^{-1/2}\mathbf{u}_i$, which again implies that $\mathbf{c}_n = \frac{1}{\sqrt{n}}\mathbf{1}$. This means that the eigenvector $\mathbf{u}_n$ corresponding to the smallest eigenvalue $\lambda_n = 0$ does not by itself contain any useful information for clustering if the graph is connected.

16.2.2 Spectral Clustering Algorithm

Algorithm 16.1 gives the pseudo-code for the spectral clustering approach. We assume that the underlying graph is connected. The method takes a dataset $\mathbf{D}$ as input and computes the similarity matrix $\mathbf{A}$. Alternatively, the matrix $\mathbf{A}$ may be directly input as well. Depending on the objective function, we choose the corresponding matrix $\mathbf{B}$. For instance, for normalized cut $\mathbf{B}$ is chosen to be either $\mathbf{L}^s$ or $\mathbf{L}^a$, whereas for ratio cut we choose $\mathbf{B} = \mathbf{L}$. Next, we compute the k smallest eigenvalues and eigenvectors of $\mathbf{B}$. However, the main problem we face is that the eigenvectors $\mathbf{u}_i$ are not binary, and thus it is not immediately clear how we can assign points to clusters. One solution to this problem is to treat the $n \times k$ matrix of eigenvectors as a new data matrix:

$$\mathbf{U} = \begin{pmatrix} | & | & & | \\ \mathbf{u}_n & \mathbf{u}_{n-1} & \cdots & \mathbf{u}_{n-k+1} \\ | & | & & | \end{pmatrix} = \begin{pmatrix} u_{n,1} & u_{n-1,1} & \cdots & u_{n-k+1,1} \\ u_{n2} & u_{n-1,2} & \cdots & u_{n-k+1,2} \\ | & | & \cdots & | \\ u_{n,n} & u_{n-1,n} & \cdots & u_{n-k+1,n} \end{pmatrix} \tag{16.18}$$

Next, we normalize each row of $\mathbf{U}$ to obtain the unit vector:

$$\mathbf{y}_i = \frac{1}{\sqrt{\sum_{j=1}^{k} u_{n-j+1,i}^2}} (u_{n,i},\ u_{n-1,i},\ \ldots,\ u_{n-k+1,i})^T \tag{16.19}$$

which yields the new normalized data matrix $\mathbf{Y} \in \mathbb{R}^{n \times k}$ comprising n points in a reduced k dimensional space:

$$\mathbf{Y} = \begin{pmatrix} — & \mathbf{y}_1^T & — \\ — & \mathbf{y}_2^T & — \\ & \vdots & \\ — & \mathbf{y}_n^T & — \end{pmatrix}$$

ALGORITHM 16.1. Spectral Clustering Algorithm

SPECTRAL CLUSTERING ($\mathbf{D}, k$):

1 Compute the similarity matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$
2 **if** *ratio cut* **then** $\mathbf{B} \leftarrow \mathbf{L}$
3 **else if** *normalized cut* **then** $\mathbf{B} \leftarrow \mathbf{L}^s$ or $\mathbf{L}^a$
4 Solve $\mathbf{B}\mathbf{u}_i = \lambda_i \mathbf{u}_i$ for $i = n, \ldots, n-k+1$, where $\lambda_n \leq \lambda_{n-1} \leq \cdots \leq \lambda_{n-k+1}$
5 $\mathbf{U} \leftarrow \begin{pmatrix} \mathbf{u}_n & \mathbf{u}_{n-1} & \cdots & \mathbf{u}_{n-k+1} \end{pmatrix}$
6 $\mathbf{Y} \leftarrow$ normalize rows of $\mathbf{U}$ using Eq. (16.19)
7 $\mathcal{C} \leftarrow \{C_1, \ldots, C_k\}$ via K-means on $\mathbf{Y}$

We can now cluster the new points in $\mathbf{Y}$ into k clusters via the K-means algorithm or any other fast clustering method, as it is expected that the clusters are well-separated in the k-dimensional eigen-space. Note that for $\mathbf{L}$, $\mathbf{L}^s$, and $\mathbf{L}^a$, the cluster indicator vector corresponding to the smallest eigenvalue $\lambda_n = 0$ is a vector of all 1's, which does not provide any information about how to separate the nodes. The real information for clustering is contained in eigenvectors starting from the second smallest eigenvalue. However, if the graph is disconnected, then even the eigenvector corresponding to λ_n can contain information valuable for clustering. Thus, we retain all k eigenvectors in $\mathbf{U}$ in Eq. (16.18).

Strictly speaking, the normalization step [Eq. (16.19)] is recommended only for the normalized symmetric Laplacian $\mathbf{L}^s$. This is because the eigenvectors of $\mathbf{L}^s$ and the cluster indicator vectors are related as $\mathbf{\Delta}^{1/2}\mathbf{c}_i = \mathbf{u}_i$. The jth entry of $\mathbf{u}_i$, corresponding to vertex v_j, is given as

$$u_{ij} = \frac{\sqrt{d_j}c_{ij}}{\sqrt{\sum_{r=1}^{n} d_r c_{ir}^2}}$$

If vertex degrees vary a lot, vertices with small degrees would have very small values u_{ij}. This can cause problems for K-means for correctly clustering these vertices. The normalization step helps alleviate this problem for $\mathbf{L}^s$, though it can also help other objectives.

Computational Complexity

The computational complexity of the spectral clustering algorithm is $O(n^3)$, because computing the eigenvectors takes that much time. However, if the graph is sparse, the complexity to compute the eigenvectors is $O(mn)$ where m is the number of edges in the graph. In particular, if $m = O(n)$, then the complexity reduces to $O(n^2)$. Running the K-means method on $\mathbf{Y}$ takes $O(tnk^2)$ time, where t is the number of iterations K-means takes to converge.

Example 16.7. Consider the normalized cut approach applied to the graph in Figure 16.2. Assume that we want to find $k = 2$ clusters. For the normalized asymmetric Laplacian matrix from Example 16.5, we compute the eigenvectors, $\mathbf{v}_7$ and $\mathbf{v}_6$, corresponding to the two smallest eigenvalues, $\lambda_7 = 0$ and $\lambda_6 = 0.517$. The matrix composed of both the eigenvectors is given as

$$\mathbf{U} = \begin{pmatrix} \mathbf{u}_1 & \mathbf{u}_2 \\ \hline -0.378 & -0.226 \\ -0.378 & -0.499 \\ -0.378 & -0.226 \\ -0.378 & -0.272 \\ -0.378 & 0.425 \\ -0.378 & 0.444 \\ -0.378 & 0.444 \end{pmatrix}$$

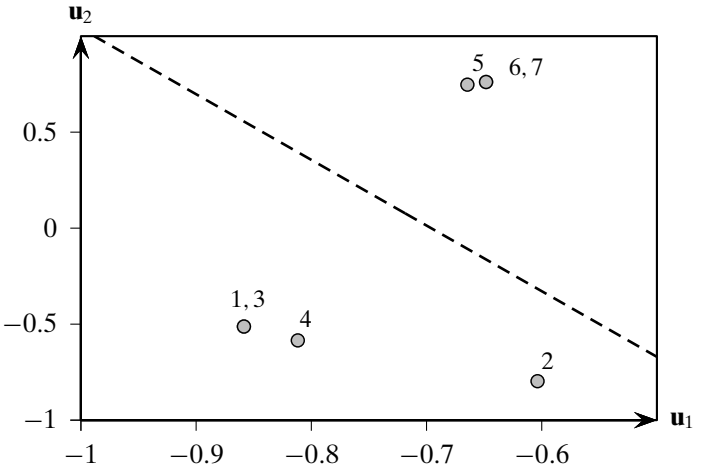


Figure 16.3. K-means on spectral dataset $\mathbf{Y}$.

We treat the ith component of $\mathbf{u}_1$ and $\mathbf{u}_2$ as the ith point $(u_{1i}, u_{2i}) \in \mathbb{R}^2$, and after normalizing all points to have unit length we obtain the new dataset:

$$\mathbf{Y} = \begin{pmatrix} -0.859 & -0.513 \\ -0.604 & -0.797 \\ -0.859 & -0.513 \\ -0.812 & -0.584 \\ -0.664 & 0.747 \\ -0.648 & 0.761 \\ -0.648 & 0.761 \end{pmatrix}$$

For instance the first point is computed as

$$\mathbf{y}_1 = \frac{1}{\sqrt{(-0.378)^2 + (-0.226^2)}}(-0.378, -0.226)^T = (-0.859, -0.513)^T$$

Figure 16.3 plots the new dataset $\mathbf{Y}$. Clustering the points into $k = 2$ groups using K-means yields the two clusters $C_1 = \{1, 2, 3, 4\}$ and $C_2 = \{5, 6, 7\}$.

Example 16.8. We apply spectral clustering on the Iris graph in Figure 16.1 using the normalized cut objective with the asymmetric Laplacian matrix $\mathbf{L}^a$. Figure 16.4 shows the $k = 3$ clusters. Comparing them with the true Iris classes (not used in the clustering), we obtain the contingency table shown in Table 16.1, indicating the number of points clustered correctly (on the main diagonal) and incorrectly (off-diagonal). We can see that cluster C_1 corresponds mainly to `iris-setosa`, C_2 to `iris-virginica`, and C_3 to `iris-versicolor`. The latter two are more difficult to separate. In total there are 18 points that are misclustered when compared to the true Iris types.

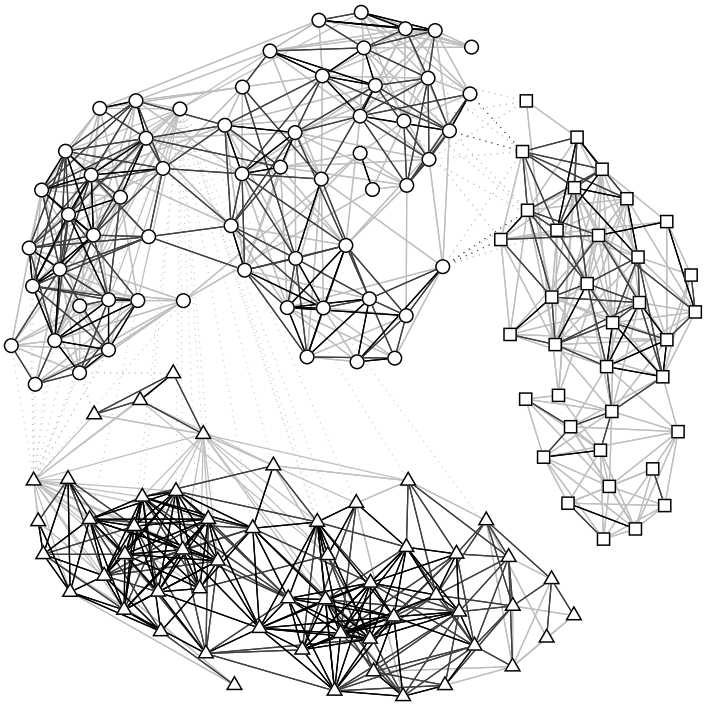


Figure 16.4. Normalized cut on Iris graph.

Table 16.1. Contingency table: clusters versus Iris types

	iris-setosa	iris-virginica	iris-versicolor
C_1 (triangle)	50	0	4
C_2 (square)	0	36	0
C_3 (circle)	0	14	46

16.2.3 Maximization Objectives: Average Cut and Modularity

We now discuss two clustering objective functions that can be formulated as maximization problems over the k-way cut $\mathcal{C} = \{C_1, \ldots, C_k\}$. These include average weight and modularity. We also explore their connections with normalized cut and kernel K-means.

Average Weight

The *average weight* objective is defined as

$$\max_{\mathcal{C}} J_{aw}(\mathcal{C}) = \sum_{i=1}^{k} \frac{W(C_i, C_i)}{|C_i|} = \sum_{i=1}^{k} \frac{\mathbf{c}_i^T \mathbf{A} \mathbf{c}_i}{\mathbf{c}_i^T \mathbf{c}_i} \tag{16.20}$$

where we used the equivalence $W(C_i, C_i) = \mathbf{c}_i^T \mathbf{A} \mathbf{c}_i$ established in Eq. (16.11). Instead of trying to minimize the weights on edges between clusters as in ratio cut, average weight tries to maximize the within cluster weights. The problem of maximizing J_{aw} for binary cluster indicator vectors is also NP-hard; we can obtain a solution by relaxing

the constraint on $\mathbf{c}_i$, by assuming that it can take on any real values for its elements. This leads to the relaxed objective

$$\max_{\mathcal{C}} J_{aw}(\mathcal{C}) = \sum_{i=1}^{k} \mathbf{u}_i^T \mathbf{A} \mathbf{u}_i \tag{16.21}$$

where $\mathbf{u}_i = \frac{\mathbf{c}_i}{\|\mathbf{c}_i\|}$. Following the same approach as in Eq. (16.15), we can maximize the objective by selecting the k largest eigenvalues of $\mathbf{A}$, and the corresponding eigenvectors

$$\begin{aligned}\max_{\mathcal{C}} J_{aw}(\mathcal{C}) &= \mathbf{u}_1^T \mathbf{A} \mathbf{u}_1 + \cdots + \mathbf{u}_k^T \mathbf{A} \mathbf{u}_k \\ &= \lambda_1 + \cdots + \lambda_k\end{aligned}$$

where $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$.

If we assume that $\mathbf{A}$ is the weighted adjacency matrix obtained from a symmetric and positive semidefinite kernel, that is, with $a_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$, then $\mathbf{A}$ will be positive semidefinite and will have non-negative real eigenvalues. In general, if we threshold $\mathbf{A}$ or if $\mathbf{A}$ is the unweighted adjacency matrix for an undirected graph, then even though $\mathbf{A}$ is symmetric, it may not be positive semidefinite. This means that in general $\mathbf{A}$ can have negative eigenvalues, though they are all real. Because J_{aw} is a maximization problem, this means that we must consider only the positive eigenvalues and the corresponding eigenvectors.

Example 16.9. For the graph in Figure 16.2, with the adjacency matrix shown in Example 16.3, its eigenvalues are as follows:

$$\lambda_1 = 3.18 \qquad \lambda_2 = 1.49 \qquad \lambda_3 = 0.62 \qquad \lambda_4 = -0.15$$
$$\lambda_5 = -1.27 \qquad \lambda_6 = -1.62 \qquad \lambda_7 = -2.25$$

We can see that the eigenvalues can be negative, as $\mathbf{A}$ is the adjacency graph and is not positive semidefinite.

Average Weight and Kernel K-means The average weight objective leads to an interesting connection between kernel K-means and graph cuts. If the weighted adjacency matrix $\mathbf{A}$ represents the kernel value between a pair of points, so that $a_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$, then we may use the sum of squared errors objective [Eq. (13.3)] of kernel K-means for graph clustering. The SSE objective is given as

$$\begin{aligned}\min_{\mathcal{C}} J_{sse}(\mathcal{C}) &= \sum_{j=1}^{n} K(\mathbf{x}_j, \mathbf{x}_j) - \sum_{i=1}^{k} \frac{1}{|C_i|} \sum_{\mathbf{x}_r \in C_i} \sum_{\mathbf{x}_s \in C_i} K(\mathbf{x}_r, \mathbf{x}_s) \\ &= \sum_{j=1}^{n} a_{jj} - \sum_{i=1}^{k} \frac{1}{|C_i|} \sum_{v_r \in C_i} \sum_{v_s \in C_i} a_{rs}\end{aligned}$$

$$= \sum_{j=1}^{n} a_{jj} - \sum_{i=1}^{k} \frac{\mathbf{c}_i^T \mathbf{A} \mathbf{c}_i}{\mathbf{c}_i^T \mathbf{c}_i}$$

$$= \sum_{j=1}^{n} a_{jj} - J_{aw}(\mathcal{C}) \tag{16.22}$$

We can observe that because $\sum_{j=1}^{n} a_{jj}$ is independent of the clustering, minimizing the SSE objective is the same as maximizing the average weight objective. In particular, if a_{ij} represents the linear kernel $\mathbf{x}_i^T \mathbf{x}_j$ between the nodes, then maximizing the average weight objective [Eq. (16.20)] is equivalent to minimizing the regular K-means SSE objective [Eq. (13.1)]. Thus, spectral clustering using J_{aw} and kernel K-means represent two different approaches to solve the same problem. Kernel K-means tries to solve the NP-hard problem by using a greedy iterative approach to directly optimize the SSE objective, whereas the graph cut formulation tries to solve the same NP-hard problem by optimally solving a relaxed problem.

Modularity

Informally, modularity is defined as the difference between the observed and expected fraction of edges within a cluster. It measures the extent to which nodes of the same type (in our case, the same cluster) are linked to each other.

Unweighted Graphs Let us assume for the moment that the graph G is unweighted, and that $\mathbf{A}$ is its binary adjacency matrix. The number of edges within a cluster C_i is given as

$$\frac{1}{2} \sum_{v_r \in C_i} \sum_{v_s \in C_i} a_{rs}$$

where we divide by $\frac{1}{2}$ because each edge is counted twice in the summation. Over all the clusters, the observed number of edges within the same cluster is given as

$$\frac{1}{2} \sum_{i=1}^{k} \sum_{v_r \in C_i} \sum_{v_s \in C_i} a_{rs} \tag{16.23}$$

Let us compute the expected number of edges between any two vertices v_r and v_s, assuming that edges are placed at random, and allowing multiple edges between the same pair of vertices. Let $|E| = m$ be the total number of edges in the graph. The probability that one end of an edge is v_r is given as $\frac{d_r}{2m}$, where d_r is the degree of v_r. The probability that one end is v_r and the other v_s is then given as

$$\frac{1}{2} p_{rs} = \frac{d_r}{2m} \cdot \frac{d_s}{2m} = \frac{d_r d_s}{4m^2}$$

The number of edges between v_r and v_s follows a binomial distribution with success probability p_{rs} over $2m$ trials (because we are selecting the two ends of m edges). The expected number of edges between v_r and v_s is given as

$$2m \cdot p_{rs} = \frac{d_r d_s}{2m}$$

The expected number of edges within a cluster C_i is then

$$\frac{1}{2}\sum_{v_r \in C_i}\sum_{v_s \in C_i}\frac{d_r d_s}{2m}$$

and the expected number of edges within the same cluster, summed over all k clusters, is given as

$$\frac{1}{2}\sum_{i=1}^{k}\sum_{v_r \in C_i}\sum_{v_s \in C_i}\frac{d_r d_s}{2m} \tag{16.24}$$

where we divide by 2 because each edge is counted twice. The *modularity* of the clustering $\mathcal{C}$ is defined as the difference between the observed and expected fraction of edges within the same cluster, obtained by subtracting Eq. (16.24) from Eq. (16.23), and dividing by the number of edges:

$$Q = \frac{1}{2m}\sum_{i=1}^{k}\sum_{v_r \in C_i}\sum_{v_s \in C_i}\left(a_{rs} - \frac{d_r d_s}{2m}\right)$$

Because $2m = \sum_{i=1}^{n} d_i$, we can rewrite modularity as follows:

$$Q = \sum_{i=1}^{k}\sum_{v_r \in C_i}\sum_{v_s \in C_i}\left(\frac{a_{rs}}{\sum_{j=1}^{n} d_j} - \frac{d_r d_s}{\left(\sum_{j=1}^{n} d_j\right)^2}\right) \tag{16.25}$$

Weighted Graphs One advantage of the modularity formulation in Eq. (16.25) is that it directly generalizes to weighted graphs. Assume that $\mathbf{A}$ is the weighted adjacency matrix; we interpret the modularity of a clustering as the difference between the observed and expected fraction of weights on edges within the clusters.

From Eq. (16.11) we have

$$\sum_{v_r \in C_i}\sum_{v_s \in C_i} a_{rs} = W(C_i, C_i)$$

and from Eq. (16.10) we have

$$\sum_{v_r \in C_i}\sum_{v_s \in C_i} d_r d_s = \left(\sum_{v_r \in C_i} d_r\right)\left(\sum_{v_s \in C_i} d_s\right) = W(C_i, V)^2$$

Further, note that

$$\sum_{j=1}^{n} d_j = W(V, V)$$

Using the above equivalences, can write the modularity objective [Eq. (16.25)] in terms of the weight function W as follows:

$$\max_{\mathcal{C}} J_Q(\mathcal{C}) = \sum_{i=1}^{k}\left(\frac{W(C_i, C_i)}{W(V, V)} - \left(\frac{W(C_i, V)}{W(V, V)}\right)^2\right) \tag{16.26}$$

We now express the modularity objective [Eq. (16.26)] in matrix terms. From Eq. (16.11), we have

$$W(C_i, C_i) = \mathbf{c}_i^T \mathbf{A} \mathbf{c}_i$$

Also note that

$$W(C_i, V) = \sum_{v_r \in C_i} d_r = \sum_{v_r \in C_i} d_r c_{ir} = \sum_{j=1}^{n} d_j c_{ij} = \sum_{j=1}^{n} \mathbf{d}^T \mathbf{c}_i$$

where $\mathbf{d} = (d_1, d_2, \ldots, d_n)^T$ is the vector of vertex degrees. Further, we have

$$W(V, V) = \sum_{j=1}^{n} d_j = tr(\mathbf{\Delta})$$

where $tr(\mathbf{\Delta})$ is the trace of $\mathbf{\Delta}$, that is, sum of the diagonal entries of $\mathbf{\Delta}$.

The clustering objective based on modularity can then be written as

$$\begin{aligned}
\max_{\mathcal{C}} J_Q(\mathcal{C}) &= \sum_{i=1}^{k} \left(\frac{\mathbf{c}_i^T \mathbf{A} \mathbf{c}_i}{tr(\mathbf{\Delta})} - \frac{(\mathbf{d}_i^T \mathbf{c}_i)^2}{tr(\mathbf{\Delta})^2} \right) \\
&= \sum_{i=1}^{k} \left(\mathbf{c}_i^T \left(\frac{\mathbf{A}}{tr(\mathbf{\Delta})} \right) \mathbf{c}_i - \mathbf{c}_i^T \left(\frac{\mathbf{d} \cdot \mathbf{d}^T}{tr(\mathbf{\Delta})^2} \right) \mathbf{c}_i \right) \\
&= \sum_{i=1}^{k} \mathbf{c}_i^T \mathbf{Q} \mathbf{c}_i
\end{aligned} \tag{16.27}$$

where $\mathbf{Q}$ is the *modularity matrix*:

$$\mathbf{Q} = \frac{1}{tr(\mathbf{\Delta})} \left(\mathbf{A} - \frac{\mathbf{d} \cdot \mathbf{d}^T}{tr(\mathbf{\Delta})} \right)$$

Directly maximizing objective Eq. (16.27) for binary cluster vectors $\mathbf{c}_i$ is hard. We resort to the approximation that elements of $\mathbf{c}_i$ can take on real values. Further, we require that $\mathbf{c}_i^T \mathbf{c}_i = \|\mathbf{c}_i\|^2 = 1$ to ensure that J_Q does not increase without bound. Following the approach in Eq. (16.15), we conclude that $\mathbf{c}_i$ is an eigenvector of $\mathbf{Q}$. However, because this a maximization problem, instead of selecting the k smallest eigenvalues, we select the k largest eigenvalues and the corresponding eigenvectors to obtain

$$\begin{aligned}
\max_{\mathcal{C}} J_Q(\mathcal{C}) &= \mathbf{u}_1^T \mathbf{Q} \mathbf{u}_1 + \cdots + \mathbf{u}_k^T \mathbf{Q} \mathbf{u}_k \\
&= \lambda_1 + \cdots + \lambda_k
\end{aligned}$$

where $\mathbf{u}_i$ is the eigenvector corresponding to λ_i, and the eigenvalues are sorted so that $\lambda_1 \geq \cdots \geq \lambda_n$. The relaxed cluster indicator vectors are given as $\mathbf{c}_i = \mathbf{u}_i$. Note that the modularity matrix $\mathbf{Q}$ is symmetric, but it is not positive semidefinite. This means that although it has real eigenvalues, they may be negative too. Also note that if Q_i denotes the ith column of $\mathbf{Q}$, then we have $Q_1 + Q_2 + \cdots + Q_n = \mathbf{0}$, which implies that 0 is an eigenvalue of $\mathbf{Q}$ with the corresponding eigenvector $\frac{1}{\sqrt{n}}\mathbf{1}$. Thus, for maximizing the modularity one should use only the positive eigenvalues.

Example 16.10. Consider the graph in Figure 16.2. The degree vector is $\mathbf{d} = (3,3,3,4,3,3,3)^T$, and the sum of degrees is $tr(\mathbf{\Delta}) = 22$. The modularity matrix is given as

$$\begin{aligned}
\mathbf{Q} &= \frac{1}{tr(\mathbf{\Delta})}\mathbf{A} - \frac{1}{tr(\mathbf{\Delta})^2}\mathbf{d}\cdot\mathbf{d}^T \\
&= \frac{1}{22}\begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{pmatrix} - \frac{1}{484}\begin{pmatrix} 9 & 9 & 9 & 12 & 9 & 9 & 9 \\ 9 & 9 & 9 & 12 & 9 & 9 & 9 \\ 9 & 9 & 9 & 12 & 9 & 9 & 9 \\ 12 & 12 & 12 & 16 & 12 & 12 & 12 \\ 9 & 9 & 9 & 12 & 9 & 9 & 9 \\ 9 & 9 & 9 & 12 & 9 & 9 & 9 \\ 9 & 9 & 9 & 12 & 9 & 9 & 9 \end{pmatrix} \\
&= \begin{pmatrix} -0.019 & 0.027 & -0.019 & 0.021 & -0.019 & 0.027 & -0.019 \\ 0.027 & -0.019 & 0.027 & 0.021 & -0.019 & -0.019 & -0.019 \\ -0.019 & 0.027 & -0.019 & 0.021 & -0.019 & -0.019 & 0.027 \\ 0.021 & 0.021 & 0.021 & -0.033 & 0.021 & -0.025 & -0.025 \\ -0.019 & -0.019 & -0.019 & 0.021 & -0.019 & 0.027 & 0.027 \\ 0.027 & -0.019 & -0.019 & -0.025 & 0.027 & -0.019 & 0.027 \\ -0.019 & -0.019 & 0.027 & -0.025 & 0.027 & 0.027 & -0.019 \end{pmatrix}
\end{aligned}$$

The eigenvalues of $\mathbf{Q}$ are as follows:

$$\lambda_1 = 0.0678 \qquad \lambda_2 = 0.0281 \qquad \lambda_3 = 0 \qquad \lambda_4 = -0.0068$$
$$\lambda_5 = -0.0579 \qquad \lambda_6 = -0.0736 \qquad \lambda_7 = -0.1024$$

The eigenvector corresponding to $\lambda_3 = 0$ is

$$\mathbf{u}_3 = \frac{1}{\sqrt{7}}(1,1,1,1,1,1,1)^T = (0.38, 0.38, 0.38, 0.38, 0.38, 0.38, 0.38)^T$$

Modularity as Average Weight Consider what happens to the modularity matrix $\mathbf{Q}$ if we use the normalized adjacency matrix $\mathbf{M} = \mathbf{\Delta}^{-1}\mathbf{A}$ in place of the standard adjacency matrix $\mathbf{A}$ in Eq. (16.27). In this case, we know by Eq. (16.3) that each row of $\mathbf{M}$ sums to 1, that is,

$$\sum_{j=1}^{n} m_{ij} = d_i = 1, \text{ for all } i = 1, \ldots, n$$

We thus have $tr(\mathbf{\Delta}) = \sum_{i=1}^{n} d_i = n$, and further $\mathbf{d}\cdot\mathbf{d}^T = \mathbf{1}_{n\times n}$, where $\mathbf{1}_{n\times n}$ is the $n \times n$ matrix of all 1's. The modularity matrix can then be written as

$$\mathbf{Q} = \frac{1}{n}\mathbf{M} - \frac{1}{n^2}\mathbf{1}_{n\times n}$$

For large graphs with many nodes, n is large and the second term practically vanishes, as $\frac{1}{n^2}$ will be very small. Thus, the modularity matrix can be reasonably

approximated as

$$\mathbf{Q} \simeq \frac{1}{n}\mathbf{M} \tag{16.28}$$

Substituting the above in the modularity objective [Eq. (16.27)], we get

$$\max_{\mathcal{C}} J_Q(\mathcal{C}) = \sum_{i=1}^{k} \mathbf{c}_i^T \mathbf{Q} \mathbf{c}_i = \sum_{i=1}^{k} \mathbf{c}_i^T \mathbf{M} \mathbf{c}_i \tag{16.29}$$

where we dropped the $\frac{1}{n}$ factor because it is a constant for a given graph; it only scales the eigenvalues without effecting the eigenvectors.

In conclusion, if we use the normalized adjacency matrix, maximizing the modularity is equivalent to selecting the k largest eigenvalues and the corresponding eigenvectors of the normalized adjacency matrix $\mathbf{M}$. Note that in this case modularity is also equivalent to the average weight objective and kernel K-means as established in Eq. (16.22).

Normalized Modularity as Normalized Cut Define the *normalized modularity* objective as follows:

$$\max_{\mathcal{C}} J_{nQ}(\mathcal{C}) = \sum_{i=1}^{k} \frac{1}{W(C_i, V)} \left(\frac{W(C_i, C_i)}{W(V, V)} - \left(\frac{W(C_i, V)}{W(V, V)} \right)^2 \right) \tag{16.30}$$

We can observe that the main difference from the modularity objective [Eq. (16.26)] is that we divide by $vol(C_i) = W(C, V_i)$ for each cluster. Simplifying the above, we obtain

$$\begin{aligned} J_{nQ}(\mathcal{C}) &= \frac{1}{W(V, V)} \sum_{i=1}^{k} \left(\frac{W(C_i, C_i)}{W(C_i, V)} - \frac{W(C_i, V)}{W(V, V)} \right) \\ &= \frac{1}{W(V, V)} \left(\sum_{i=1}^{k} \left(\frac{W(C_i, C_i)}{W(C_i, V)} \right) - \sum_{i=1}^{k} \left(\frac{W(C_i, V)}{W(V, V)} \right) \right) \\ &= \frac{1}{W(V, V)} \left(\sum_{i=1}^{k} \left(\frac{W(C_i, C_i)}{W(C_i, V)} \right) - 1 \right) \end{aligned}$$

Now consider the expression $(k-1) - W(V, V) \cdot J_{nQ}(\mathcal{C})$, we have

$$\begin{aligned} (k-1) - W(V, V) J_{nQ}(\mathcal{C}) &= (k-1) - \left(\sum_{i=1}^{k} \left(\frac{W(C_i, C_i)}{W(C_i, V)} \right) - 1 \right) \\ &= k - \sum_{i=1}^{k} \frac{W(C_i, C_i)}{W(C_i, V)} \\ &= \sum_{i=1}^{k} 1 - \frac{W(C_i, C_i)}{W(C_i, V)} \\ &= \sum_{i=1}^{k} \frac{W(C_i, V) - W(C_i, C_i)}{W(C_i, V)} \end{aligned}$$

$$= \sum_{i=1}^{k} \frac{W(C_i, \overline{C_i})}{W(C_i, V)}$$

$$= \sum_{i=1}^{k} \frac{W(C_i, \overline{C_i})}{vol(C_i)}$$

$$= J_{nc}(\mathcal{C})$$

In other words the normalized cut objective [Eq. (16.17)] is related to the normalized modularity objective [Eq. (16.30)] by the following equation:

$$J_{nc}(\mathcal{C}) = (k-1) - W(V,V) \cdot J_{nQ}(\mathcal{C})$$

Since $W(V, V)$ is a constant for a given graph, we observe that minimizing normalized cut is equivalent to maximizing normalized modularity.

Spectral Clustering Algorithm

Both average weight and modularity are maximization objectives; therefore we have to slightly modify Algorithm 16.1 for spectral clustering to use these objectives. The matrix $\mathbf{B}$ is chosen to be $\mathbf{A}$ if we are maximizing average weight or $\mathbf{Q}$ for the modularity objective. Next, instead of computing the k smallest eigenvalues we have to select the k largest eigenvalues and their corresponding eigenvectors. Because both $\mathbf{A}$ and $\mathbf{Q}$ can have negative eigenvalues, we must select only the positive eigenvalues. The rest of the algorithm remains the same.

16.3 MARKOV CLUSTERING

We now consider a graph clustering method based on simulating a random walk on a weighted graph. The basic intuition is that if node transitions reflect the weights on the edges, then transitions from one node to another within a cluster are much more likely than transitions between nodes from different clusters. This is because nodes within a cluster have higher similarities or weights, and nodes across clusters have lower similarities.

Given the weighted adjacency matrix $\mathbf{A}$ for a graph G, the normalized adjacency matrix [Eq. (16.2)] is given as $\mathbf{M} = \mathbf{\Delta}^{-1}\mathbf{A}$. The matrix $\mathbf{M}$ can be interpreted as the $n \times n$ *transition matrix* where the entry $m_{ij} = \frac{a_{ij}}{d_i}$ can be interpreted as the probability of transitioning or jumping from node i to node j in the graph G. This is because $\mathbf{M}$ is a *row stochastic* or *Markov* matrix, which satisfies the following conditions: (1) elements of the matrix are non-negative, that is, $m_{ij} \geq 0$, which follows from the fact that $\mathbf{A}$ is non-negative, and (2) rows of $\mathbf{M}$ are probability vectors, that is, row elements add to 1, because

$$\sum_{j=1}^{n} m_{ij} = \sum_{j=1}^{n} \frac{a_{ij}}{d_i} = 1$$

The matrix $\mathbf{M}$ is thus the transition matrix for a *Markov chain* or a Markov random walk on graph G. A Markov chain is a discrete-time stochastic process over a set of

states, in our case the set of vertices V. The Markov chain makes a transition from one node to another at discrete timesteps $t = 1, 2, \ldots$, with the probability of making a transition from node i to node j given as m_{ij}. Let the random variable X_t denote the state at time t. The Markov property means that the probability distribution of X_t over the states at time t depends only on the probability distribution of X_{t-1}, that is,

$$P(X_t = i | X_0, X_1, \ldots, X_{t-1}) = P(X_t = i | X_{t-1})$$

Further, we assume that the Markov chain is *homogeneous*, that is, the transition probability

$$P(X_t = j | X_{t-1} = i) = m_{ij}$$

is independent of the time step t.

Given node i the transition matrix $\mathbf{M}$ specifies the probabilities of reaching any other node j in one time step. Starting from node i at $t = 0$, let us consider the probability of being at node j at $t = 2$, that is, after two steps. We denote by $m_{ij}(2)$ the probability of reaching j from i in two time steps. We can compute this as follows:

$$\begin{aligned} m_{ij}(2) &= P(X_2 = j | X_0 = i) = \sum_{a=1}^{n} P(X_1 = a | X_0 = i) P(X_2 = j | X_1 = a) \\ &= \sum_{a=1}^{n} m_{ia} m_{aj} = \mathbf{m}_i^T M_j \end{aligned} \tag{16.31}$$

where $\mathbf{m}_i = (m_{i1}, m_{i2}, \ldots, m_{in})^T$ denotes the vector corresponding to the ith row of $\mathbf{M}$ and $M_j = (m_{1j}, m_{2j}, \ldots, m_{nj})^T$ denotes the vector corresponding to the jth column of $\mathbf{M}$.

Consider the product of $\mathbf{M}$ with itself:

$$\begin{aligned} \mathbf{M}^2 = \mathbf{M} \cdot \mathbf{M} &= \begin{pmatrix} — \mathbf{m}_1^T — \\ — \mathbf{m}_2^T — \\ \vdots \\ — \mathbf{m}_n^T — \end{pmatrix} \begin{pmatrix} | & | & & | \\ M_1 & M_2 & \cdots & M_n \\ | & | & & | \end{pmatrix} \\ &= \left\{ \mathbf{m}_i^T M_j \right\}_{i,j=1}^{n} = \left\{ m_{ij}(2) \right\}_{i,j=1}^{n} \end{aligned} \tag{16.32}$$

Equations (16.31) and (16.32) imply that $\mathbf{M}^2$ is precisely the transition probability matrix for the Markov chain over two time-steps. Likewise, the three-step transition matrix is $\mathbf{M}^2 \cdot \mathbf{M} = \mathbf{M}^3$. In general, the transition probability matrix for t time steps is given as

$$\mathbf{M}^{t-1} \cdot \mathbf{M} = \mathbf{M}^t \tag{16.33}$$

A random walk on G thus corresponds to taking successive powers of the transition matrix $\mathbf{M}$. Let $\boldsymbol{\pi}_0$ specify the initial state probability vector at time $t = 0$, that is, $\boldsymbol{\pi}_{0i} = P(X_0 = i)$ is the probability of starting at node i, for all $i = 1, \ldots, n$. Starting

from $\boldsymbol{\pi}_0$, we can obtain the state probability vector for X_t, that is, the probability of being at node i at time-step t, as follows

$$\begin{aligned}\boldsymbol{\pi}_t^T &= \boldsymbol{\pi}_{t-1}^T \mathbf{M} \\ &= \left(\boldsymbol{\pi}_{t-2}^T \mathbf{M}\right) \cdot \mathbf{M} = \boldsymbol{\pi}_{t-2}^T \mathbf{M}^2 \\ &= \left(\boldsymbol{\pi}_{t-3}^T \mathbf{M}^2\right) \cdot \mathbf{M} = \boldsymbol{\pi}_{t-3}^T \mathbf{M}^3 \\ &= \vdots \\ &= \boldsymbol{\pi}_0^T \mathbf{M}^t\end{aligned}$$

Equivalently, taking transpose on both sides, we get

$$\boldsymbol{\pi}_t = (M^t)^T \boldsymbol{\pi}_0 = (\mathbf{M}^T)^t \boldsymbol{\pi}_0$$

The state probability vector thus converges to the dominant eigenvector of $\mathbf{M}^T$, reflecting the steady-state probability of reaching any node in the graph, regardless of the starting node. Note that if the graph is directed, then the steady-state vector is equivalent to the normalized prestige vector [Eq. (4.6)].

Transition Probability Inflation

We now consider a variation of the random walk, where the probability of transitioning from node i to j is inflated by taking each element m_{ij} to the power $r \geq 1$. Given a transition matrix $\mathbf{M}$, define the inflation operator Υ as follows:

$$\Upsilon(\mathbf{M}, r) = \left\{ \frac{(m_{ij})^r}{\sum_{a=1}^{n} (m_{ia})^r} \right\}_{i,j=1}^{n} \tag{16.34}$$

The inflation operation results in a transformed or inflated transition probability matrix because the elements remain non-negative, and each row is normalized to sum to 1. The net effect of the inflation operator is to increase the higher probability transitions and decrease the lower probability transitions.

16.3.1 Markov Clustering Algorithm

The Markov clustering algorithm (MCL) is an iterative method that interleaves matrix expansion and inflation steps. Matrix expansion corresponds to taking successive powers of the transition matrix, leading to random walks of longer lengths. On the other hand, matrix inflation makes the higher probability transitions even more likely and reduces the lower probability transitions. Because nodes in the same cluster are expected to have higher weights, and consequently higher transition probabilities between them, the inflation operator makes it more likely to stay within the cluster. It thus limits the extent of the random walk.

The pseudo-code for MCL is given in Algorithm 16.2. The method works on the weighted adjacency matrix for a graph. Instead of relying on a user-specified value for k, the number of output clusters, MCL takes as input the inflation parameter $r \geq 1$. Higher values lead to more, smaller clusters, whereas smaller values lead to fewer, but larger clusters. However, the exact number of clusters cannot be pre-determined. Given the adjacency matrix $\mathbf{A}$, MCL first adds *loops* or self-edges to $\mathbf{A}$ if they do

ALGORITHM 16.2. Markov Clustering Algorithm (MCL)

```
MARKOV CLUSTERING (A, r, ε):
t ← 0
Add self-edges to A if they do not exist
M_t ← Δ^{-1}A
repeat
    t ← t + 1
    M_t ← M_{t-1} · M_{t-1}
    M_t ← Υ(M_t, r)
until ‖M_t − M_{t-1}‖_F ≤ ε
G_t ← directed graph induced by M_t
C ← {weakly connected components in G_t}
```

not exist. If $\mathbf{A}$ is a similarity matrix, then this is not required, as a node is most similar to itself, and thus $\mathbf{A}$ should have high values on the diagonals. For simple, undirected graphs, if $\mathbf{A}$ is the adjacency matrix, then adding self-edges associates return probabilities with each node.

The iterative MCL expansion and inflation process stops when the transition matrix converges, that is, when the difference between the transition matrix from two successive iterations falls below some threshold $\epsilon \geq 0$. The matrix difference is given in terms of the *Frobenius norm*:

$$\|\mathbf{M}_t - \mathbf{M}_{t-1}\|_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n}\Big(\mathbf{M}_t(i,j) - \mathbf{M}_{t-1}(i,j)\Big)^2}$$

The MCL process stops when $\|\mathbf{M}_t - \mathbf{M}_{t-1}\|_F \leq \epsilon$.

MCL Graph

The final clusters are found by enumerating the weakly connected components in the directed graph induced by the converged transition matrix $\mathbf{M}_t$. The directed graph induced by $\mathbf{M}_t$ is denoted as $G_t = (V_t, E_t)$. The vertex set is the same as the set of nodes in the original graph, that is, $V_t = V$, and the edge set is given as

$$E_t = \big\{(i,j) \mid \mathbf{M}_t(i,j) > 0\big\}$$

In other words, a directed edge (i,j) exists only if node i can transition to node j within t steps of the expansion and inflation process. A node j is called an *attractor* if $\mathbf{M}_t(j,j) > 0$, and we say that node i is attracted to attractor j if $\mathbf{M}_t(i,j) > 0$. The MCL process yields a set of attractor nodes, $V_a \subseteq V$, such that other nodes are attracted to at least one attractor in V_a. That is, for all nodes i there exists a node $j \in V_a$, such that $(i,j) \in E_t$. A strongly connected component in a directed graph

is defined a maximal subgraph such that there exists a directed path between all pairs of vertices in the subgraph. To extract the clusters from G_t, MCL first finds the strongly connected components $S_1, S_2, \ldots, S_q$ over the set of attractors V_a. Next, for each strongly connected set of attractors S_j, MCL finds the weakly connected components consisting of all nodes $i \in V_t - V_a$ attracted to an attractor in S_j. If a node i is attracted to multiple strongly connected components, it is added to each such cluster, resulting in possibly overlapping clusters.

Example 16.11. We apply the MCL method to find $k = 2$ clusters for the graph shown in Figure 16.2. We add the self-loops to the graph to obtain the adjacency matrix:

$$\mathbf{A} = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}$$

The corresponding Markov matrix is given as

$$\mathbf{M}_0 = \boldsymbol{\Delta}^{-1}\mathbf{A} = \begin{pmatrix} 0.25 & 0.25 & 0 & 0.25 & 0 & 0.25 & 0 \\ 0.25 & 0.25 & 0.25 & 0.25 & 0 & 0 & 0 \\ 0 & 0.25 & 0.25 & 0.25 & 0 & 0 & 0.25 \\ 0.20 & 0.20 & 0.20 & 0.20 & 0.20 & 0 & 0 \\ 0 & 0 & 0 & 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0 & 0 & 0 & 0.25 & 0.25 & 0.25 \\ 0 & 0 & 0.25 & 0 & 0.25 & 0.25 & 0.25 \end{pmatrix}$$

In the first iteration, we apply expansion and then inflation (with $r = 2.5$) to obtain

$$\mathbf{M}_1 = \mathbf{M}_0 \cdot \mathbf{M}_0 = \begin{pmatrix} 0.237 & 0.175 & 0.113 & 0.175 & 0.113 & 0.125 & 0.062 \\ 0.175 & 0.237 & 0.175 & 0.237 & 0.050 & 0.062 & 0.062 \\ 0.113 & 0.175 & 0.237 & 0.175 & 0.113 & 0.062 & 0.125 \\ 0.140 & 0.190 & 0.140 & 0.240 & 0.090 & 0.100 & 0.100 \\ 0.113 & 0.050 & 0.113 & 0.113 & 0.237 & 0.188 & 0.188 \\ 0.125 & 0.062 & 0.062 & 0.125 & 0.188 & 0.250 & 0.188 \\ 0.062 & 0.062 & 0.125 & 0.125 & 0.188 & 0.188 & 0.250 \end{pmatrix}$$

$$\mathbf{M}_1 = \Upsilon(\mathbf{M}_1, 2.5) = \begin{pmatrix} 0.404 & 0.188 & 0.062 & 0.188 & 0.062 & 0.081 & 0.014 \\ 0.154 & 0.331 & 0.154 & 0.331 & 0.007 & 0.012 & 0.012 \\ 0.062 & 0.188 & 0.404 & 0.188 & 0.062 & 0.014 & 0.081 \\ 0.109 & 0.234 & 0.109 & 0.419 & 0.036 & 0.047 & 0.047 \\ 0.060 & 0.008 & 0.060 & 0.060 & 0.386 & 0.214 & 0.214 \\ 0.074 & 0.013 & 0.013 & 0.074 & 0.204 & 0.418 & 0.204 \\ 0.013 & 0.013 & 0.074 & 0.074 & 0.204 & 0.204 & 0.418 \end{pmatrix}$$

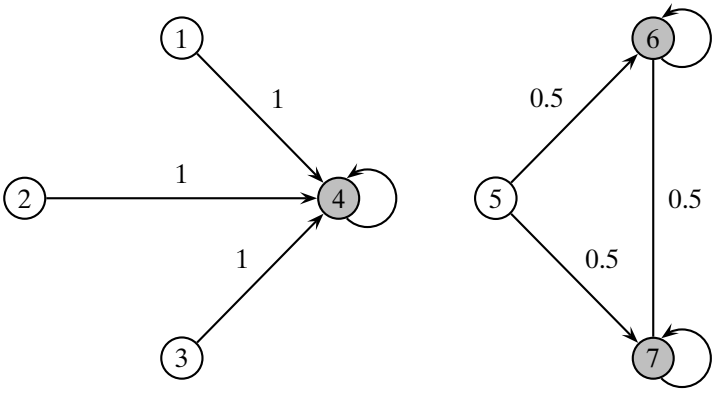


Figure 16.5. MCL attractors and clusters.

MCL converges in 10 iterations (using $\epsilon = 0.001$), with the final transition matrix

$$\mathbf{M} = \begin{array}{c|ccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0.5 \\ 6 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0.5 \\ 7 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0.5 \end{array}$$

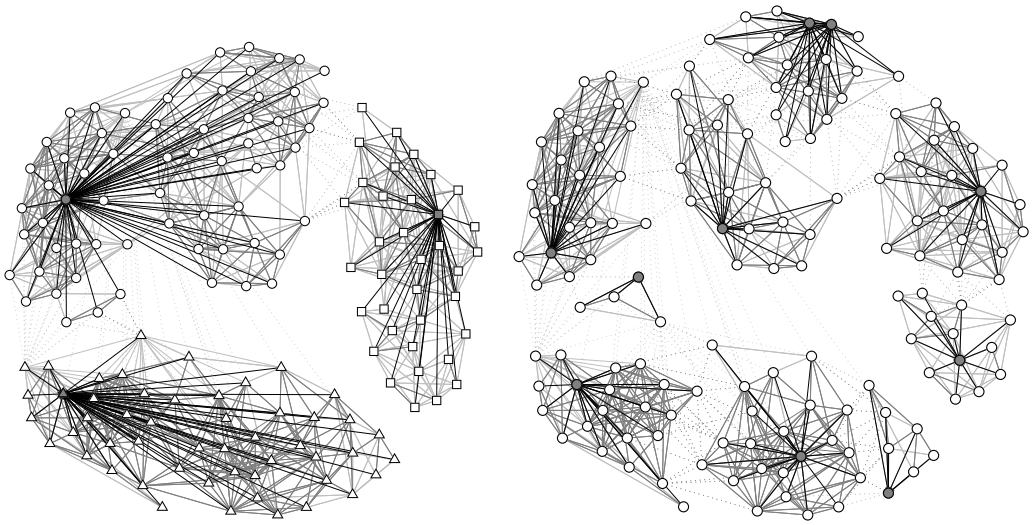
Figure 16.5 shows the directed graph induced by the converged $\mathbf{M}$ matrix, where an edge (i, j) exists if and only if $\mathbf{M}(i, j) > 0$. The nonzero diagonal elements of $\mathbf{M}$ are the attractors (nodes with self-loops, shown in gray). We can observe that $\mathbf{M}(4, 4)$, $\mathbf{M}(6, 6)$, and $\mathbf{M}(7, 7)$ are all greater than zero, making nodes 4, 6, and 7 the three attractors. Because both 6 and 7 can reach each other, the equivalence classes of attractors are $\{4\}$ and $\{6, 7\}$. Nodes $1, 2$, and 3 are attracted to 4, and node 5 is attracted to both 6 and 7. Thus, the two weakly connected components that make up the two clusters are $C_1 = \{1, 2, 3, 4\}$ and $C_2 = \{5, 6, 7\}$.

Example 16.12. Figure 16.6a shows the clusters obtained via the MCL algorithm on the Iris graph from Figure 16.1, using $r = 1.3$ in the inflation step. MCL yields three attractors (shown as gray nodes; self-loops omitted), which separate the graph into three clusters. The contingency table for the discovered clusters versus the true Iris types is given in Table 16.2. One point with class `iris-versicolor` is (wrongly) grouped with `iris-setosa` in C_1, but 14 points from `iris-virginica` are misclustered.

Notice that the only parameter for MCL is r, the exponent for the inflation step. The number of clusters is not explicitly specified, but higher values of r result in more clusters. The value of $r = 1.3$ was used above because it resulted in three clusters. Figure 16.6b shows the results for $r = 2$. MCL yields nine clusters, where one of the clusters (top-most) has two attractors.

Table 16.2. Contingency table: MCL clusters versus Iris types

	iris-setosa	iris-virginica	iris-versicolor
C_1 (triangle)	50	0	1
C_2 (square)	0	36	0
C_3 (circle)	0	14	49



(a) $r = 1.3$ (b) $r = 2$

Figure 16.6. MCL on Iris graph.

Computational Complexity

The computational complexity of the MCL algorithm is $O(tn^3)$, where t is the number of iterations until convergence. This follows from the fact that whereas the inflation operation takes $O(n^2)$ time, the expansion operation requires matrix multiplication, which takes $O(n^3)$ time. However, the matrices become sparse very quickly, and it is possible to use sparse matrix multiplication to obtain $O(n^2)$ complexity for expansion in later iterations. On convergence, the weakly connected components in G_t can be found in $O(n + m)$ time, where m is the number of edges. Because G_t is very sparse, with $m = O(n)$, the final clustering step takes $O(n)$ time.

16.4 FURTHER READING

Spectral partitioning of graphs was first proposed in Donath and Hoffman (1973). Properties of the second smallest eigenvalue of the Laplacian matrix, also called *algebraic connectivity*, were studied in Fiedler (1973). A recursive bipartitioning approach to find k clusters using the normalized cut objective was given in Shi and Malik (2000). The direct k-way partitioning approach for normalized cut, using the normalized symmetric Laplacian matrix, was proposed in Ng, Jordan, and Weiss (2001). The connection between spectral clustering objective and kernel K-means was established

in Dhillon, Guan, and Kulis (2007). The modularity objective was introduced in Newman (2003), where it was called *assortativity coefficient*. The spectral algorithm using the modularity matrix was first proposed in Smyth and White (2005). The relationship between modularity and normalized cut was shown in Yu and Ding (2010). For an excellent tutorial on spectral clustering techniques see Luxburg (2007). The Markov clustering algorithm was originally proposed in van Dongen (2000). For an extensive review of graph clustering methods see Fortunato (2010).

Dhillon, I. S., Guan, Y., and Kulis, B. (2007). "Weighted graph cuts without eigenvectors: A multilevel approach." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29 (11): 1944–1957.

Donath, W. E. and Hoffman, A. J. (September 1973)."Lower bounds for the partitioning of graphs." *IBM Journal of Research and Development*, 17 (5): 420–425.

Fiedler, M. (1973). "Algebraic connectivity of graphs." *Czechoslovak Mathematical Journal*, 23 (2): 298–305.

Fortunato, S. (2010). "Community detection in graphs." *Physics Reports*, 486 (3): 75–174.

Luxburg, U. (December 2007). "A tutorial on spectral clustering." *Statistics and Computing*, 17 (4): 395–416.

Newman, M. E. (2003). "Mixing patterns in networks." *Physical Review E*, 67 (2): 026126.

Ng, A. Y., Jordan, M. I., and Weiss, Y. (2001). "On spectral clustering: Analysis and an algorithm." *Advances in Neural Information Processing Systems 14* (pp. 849–856). Cambridge, MA: MIT Press.

Shi, J. and Malik, J. (August 2000). "Normalized cuts and image segmentation." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22 (8): 888–905.

Smyth, S. and White, S. (2005). "A spectral clustering approach to finding communities in graphs." *In Proceedings of the 5th SIAM International Conference on Data Mining,* vol. 119, p. 274.

van Dongen, S. M. (2000). "Graph clustering by flow simulation." PhD thesis. The University of Utrecht, The Netherlands.

Yu, L. and Ding, C. (2010). "Network community discovery: solving modularity clustering via normalized cut." *In Proceedings of the 8th Workshop on Mining and Learning with Graphs*. ACM pp. 34–36.

16.5 EXERCISES

Q1. Show that if Q_i denotes the ith column of the modularity matrix $\mathbf{Q}$, then $\sum_{i=1}^{n} Q_i = \mathbf{0}$.

Q2. Prove that both the normalized symmetric and asymmetric Laplacian matrices $\mathbf{L}^s$ [Eq. (16.6)] and $\mathbf{L}^a$ [Eq. (16.9)] are positive semidefinite. Also show that the smallest eigenvalue is $\lambda_n = 0$ for both.

Q3. Prove that the largest eigenvalue of the normalized adjacency matrix $\mathbf{M}$ [Eq. (16.2)] is 1, and further that all eigenvalues satisfy the condition that $|\lambda_i| \leq 1$.

Q4. Show that $\sum_{v_r \in C_i} c_{ir} d_r c_{ir} = \sum_{r=1}^{n} \sum_{s=1}^{n} c_{ir} \mathbf{\Delta}_{rs} c_{is}$, where $\mathbf{c}_i$ is the cluster indicator vector for cluster C_i and $\mathbf{\Delta}$ is the degree matrix for the graph.

Q5. For the normalized symmetric Laplacian $\mathbf{L}^s$, show that for the normalized cut objective the real-valued cluster indicator vector corresponding to the smallest eigenvalue $\lambda_n = 0$ is given as $\mathbf{c}_n = \frac{1}{\sqrt{n}}\mathbf{1}$.

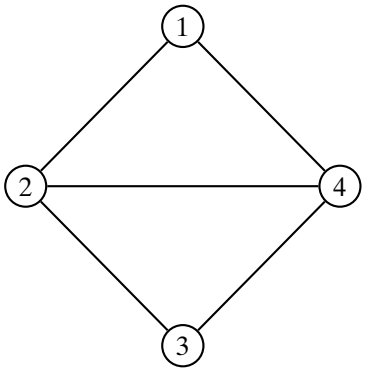


Figure 16.7. Graph for Q6.

Q6. Given the graph in Figure 16.7, answer the following questions:

(a) Cluster the graph into two clusters using ratio cut and normalized cut.

(b) Use the normalized adjacency matrix $\mathbf{M}$ for the graph and cluster it into two clusters using average weight and kernel K-means, using $\mathbf{K} = \mathbf{M}$.

(c) Cluster the graph using the MCL algorithm with inflation parameters $r = 2$ and $r = 2.5$.

Table 16.3. Data for Q7

	X_1	X_2	X_3
$\mathbf{x}_1$	0.4	0.9	0.6
$\mathbf{x}_2$	0.5	0.1	0.6
$\mathbf{x}_3$	0.6	0.3	0.6
$\mathbf{x}_4$	0.4	0.8	0.5

Q7. Consider Table 16.3. Assuming these are nodes in a graph, define the weighted adjacency matrix $\mathbf{A}$ using the linear kernel

$$\mathbf{A}(i, j) = 1 + \mathbf{x}_i^T \mathbf{x}_j$$

Cluster the data into two groups using the modularity objective.

CHAPTER 17 Clustering Validation

There exist many different clustering methods, depending on the type of clusters sought and on the inherent data characteristics. Given the diversity of clustering algorithms and their parameters it is important to develop objective approaches to assess clustering results. Cluster validation and assessment encompasses three main tasks: *clustering evaluation* seeks to assess the goodness or quality of the clustering, *clustering stability* seeks to understand the sensitivity of the clustering result to various algorithmic parameters, for example, the number of clusters, and *clustering tendency* assesses the suitability of applying clustering in the first place, that is, whether the data has any inherent grouping structure. There are a number of validity measures and statistics that have been proposed for each of the aforementioned tasks, which can be divided into three main types:

External: External validation measures employ criteria that are not inherent to the dataset. This can be in form of prior or expert-specified knowledge about the clusters, for example, class labels for each point.

Internal: Internal validation measures employ criteria that are derived from the data itself. For instance, we can use intracluster and intercluster distances to obtain measures of cluster compactness (e.g., how similar are the points in the same cluster?) and separation (e.g., how far apart are the points in different clusters?).

Relative: Relative validation measures aim to directly compare different clusterings, usually those obtained via different parameter settings for the same algorithm.

In this chapter we study some of the main techniques for clustering validation and assessment spanning all three types of measures.

17.1 EXTERNAL MEASURES

As the name implies, external measures assume that the correct or ground-truth clustering is known *a priori*. The true cluster labels play the role of external information

that is used to evaluate a given clustering. In general, we would not know the correct clustering; however, external measures can serve as way to test and validate different methods. For instance, classification datasets that specify the class for each point can be used to evaluate the quality of a clustering. Likewise, synthetic datasets with known cluster structure can be created to evaluate various clustering algorithms by quantifying the extent to which they can recover the known groupings.

Let $\mathbf{D} = \{\mathbf{x}_i\}_{i=1}^n$ be a dataset consisting of n points in a d-dimensional space, partitioned into k clusters. Let $y_i \in \{1, 2, \ldots, k\}$ denote the ground-truth cluster membership or label information for each point. The ground-truth clustering is given as $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$, where the cluster T_j consists of all the points with label j, i.e., $T_j = \{\mathbf{x}_i \in \mathbf{D} | y_i = j\}$. Also, let $\mathcal{C} = \{C_1, \ldots, C_r\}$ denote a clustering of the same dataset into r clusters, obtained via some clustering algorithm, and let $\hat{y}_i \in \{1, 2, \ldots, r\}$ denote the cluster label for $\mathbf{x}_i$. For clarity, henceforth, we will refer to $\mathcal{T}$ as the ground-truth *partitioning*, and to each T_i as a *partition*. We will call $\mathcal{C}$ a clustering, with each C_i referred to as a cluster. Because the ground truth is assumed to be known, typically clustering methods will be run with the correct number of clusters, that is, with $r = k$. However, to keep the discussion more general, we allow r to be different from k.

External evaluation measures try capture the extent to which points from the same partition appear in the same cluster, and the extent to which points from different partitions are grouped in different clusters. There is usually a trade-off between these two goals, which is either explicitly captured by a measure or is implicit in its computation. All of the external measures rely on the $r \times k$ *contingency table* $\mathbf{N}$ that is induced by a clustering $\mathcal{C}$ and the ground-truth partitioning $\mathcal{T}$, defined as follows

$$\mathbf{N}(i, j) = n_{ij} = \left| C_i \cap T_j \right|$$

In other words, the count n_{ij} denotes the number of points that are common to cluster C_i and ground-truth partition T_j. Further, for clarity, let $n_i = |C_i|$ denote the number of points in cluster C_i, and let $m_j = |T_j|$ denote the number of points in partition T_j. The contingency table can be computed from $\mathcal{T}$ and $\mathcal{C}$ in $O(n)$ time by examining the partition and cluster labels, y_i and $\hat{y}_i$, for each point $\mathbf{x}_i \in \mathbf{D}$ and incrementing the corresponding count $n_{y_i \hat{y}_i}$.

17.1.1 Matching Based Measures

Purity

Purity quantifies the extent to which a cluster C_i contains entities from only one partition. In other words, it measures how "pure" each cluster is. The purity of cluster C_i is defined as

$$purity_i = \frac{1}{n_i} \max_{j=1}^{k} \{n_{ij}\}$$

The purity of clustering $\mathcal{C}$ is defined as the weighted sum of the clusterwise purity values:

$$purity = \sum_{i=1}^{r} \frac{n_i}{n} purity_i = \frac{1}{n} \sum_{i=1}^{r} \max_{j=1}^{k} \{n_{ij}\}$$

where the ratio $\frac{n_i}{n}$ denotes the fraction of points in cluster C_i. The larger the purity of $\mathcal{C}$, the better the agreement with the groundtruth. The maximum value of purity is 1, when each cluster comprises points from only one partition. When $r = k$, a purity value of 1 indicates a perfect clustering, with a one-to-one correspondence between the clusters and partitions. However, purity can be 1 even for $r > k$, when each of the clusters is a subset of a ground-truth partition. When $r < k$, purity can never be 1, because at least one cluster must contain points from more than one partition.

Maximum Matching

The maximum matching measure selects the mapping between clusters and partitions, such that the sum of the number of common points (n_{ij}) is maximized, provided that only one cluster can match with a given partition. This is unlike purity, where two different clusters may share the same majority partition.

Formally, we treat the contingency table as a complete weighted bipartite graph $G = (V, E)$, where each partition and cluster is a node, that is, $V = \mathcal{C} \cup \mathcal{T}$, and there exists an edge $(C_i, T_j) \in E$, with weight $w(C_i, T_j) = n_{ij}$, for all $C_i \in \mathcal{C}$ and $T_j \in \mathcal{T}$. A *matching* M in G is a subset of E, such that the edges in M are pairwise nonadjacent, that is, they do not have a common vertex. The maximum matching measure is defined as the *maximum weight matching* in G:

$$match = \arg\max_{M} \left\{ \frac{w(M)}{n} \right\}$$

where the weight of a matching M is simply the sum of all the edge weights in M, given as $w(M) = \sum_{e \in M} w(e)$. The maximum matching can be computed in time $O(|V|^2 \cdot |E|) = O((r+k)^2 rk)$, which is equivalent to $O(k^4)$ if $r = O(k)$.

F-Measure

Given cluster C_i, let j_i denote the partition that contains the maximum number of points from C_i, that is, $j_i = \max_{j=1}^{k} \{n_{ij}\}$. The *precision* of a cluster C_i is the same as its purity:

$$prec_i = \frac{1}{n_i} \max_{j=1}^{k} \{n_{ij}\} = \frac{n_{ij_i}}{n_i}$$

It measures the fraction of points in C_i from the majority partition T_{j_i}.

The *recall* of cluster C_i is defined as

$$recall_i = \frac{n_{ij_i}}{|T_{j_i}|} = \frac{n_{ij_i}}{m_{j_i}}$$

where $m_{j_i} = |T_{j_i}|$. It measures the fraction of point in partition T_{j_i} shared in common with cluster C_i.

The F-measure is the harmonic mean of the precision and recall values for each cluster. The F-measure for cluster C_i is therefore given as

$$F_i = \frac{2}{\frac{1}{prec_i} + \frac{1}{recall_i}} = \frac{2 \cdot prec_i \cdot recall_i}{prec_i + recall_i} = \frac{2\, n_{ij_i}}{n_i + m_{j_i}} \tag{17.1}$$

The F-measure for the clustering $\mathcal{C}$ is the mean of clusterwise F-measure values:

$$F = \frac{1}{r}\sum_{i=1}^{r} F_i$$

F-measure thus tries to balance the precision and recall values across all the clusters. For a perfect clustering, when $r = k$, the maximum value of the F-measure is 1.

Example 17.1. Figure 17.1 shows two different clusterings obtained via the K-means algorithm on the Iris dataset, using the first two principal components as the two dimensions. Here $n = 150$, and $k = 3$. Visual inspection confirms that Figure 17.1a is a better clustering than that in Figure 17.1b. We now examine how the different contingency table based measures can be used to evaluate these two clusterings.

Consider the clustering in Figure 17.1a. The three clusters are illustrated with different symbols; the gray points are in the correct partition, whereas the white ones are wrongly clustered compared to the ground-truth Iris types. For instance, C_3 mainly corresponds to partition T_3 (`Iris-virginica`), but it has three points (the white triangles) from T_2. The complete contingency table is as follows:

	`iris-setosa` T_1	`iris-versicolor` T_2	`iris-virginica` T_3	n_i
C_1(squares)	0	47	14	61
C_2(circles)	50	0	0	50
C_3(triangles)	0	3	36	39
m_j	50	50	50	$n = 100$

To compute purity, we first note for each cluster the partition with the maximum overlap. We have the correspondence (C_1, T_2), (C_2, T_1), and (C_3, T_3). Thus, purity is given as

$$purity = \frac{1}{150}(47 + 50 + 36) = \frac{133}{150} = 0.887$$

For this contingency table, the maximum matching measure gives the same result, as the correspondence above is in fact a maximum weight matching. Thus, $match = 0.887$.

The cluster C_1 contains $n_1 = 47 + 14 = 61$ points, whereas its corresponding partition T_2 contains $m_2 = 47 + 3 = 50$ points. Thus, the precision and recall for C_1 are given as

$$prec_1 = \tfrac{47}{61} = 0.77$$
$$recall_1 = \tfrac{47}{50} = 0.94$$

The F-measure for C_1 is therefore

$$F_1 = \frac{2 \cdot 0.77 \cdot 0.94}{0.77 + 0.94} = \tfrac{1.45}{1.71} = 0.85$$

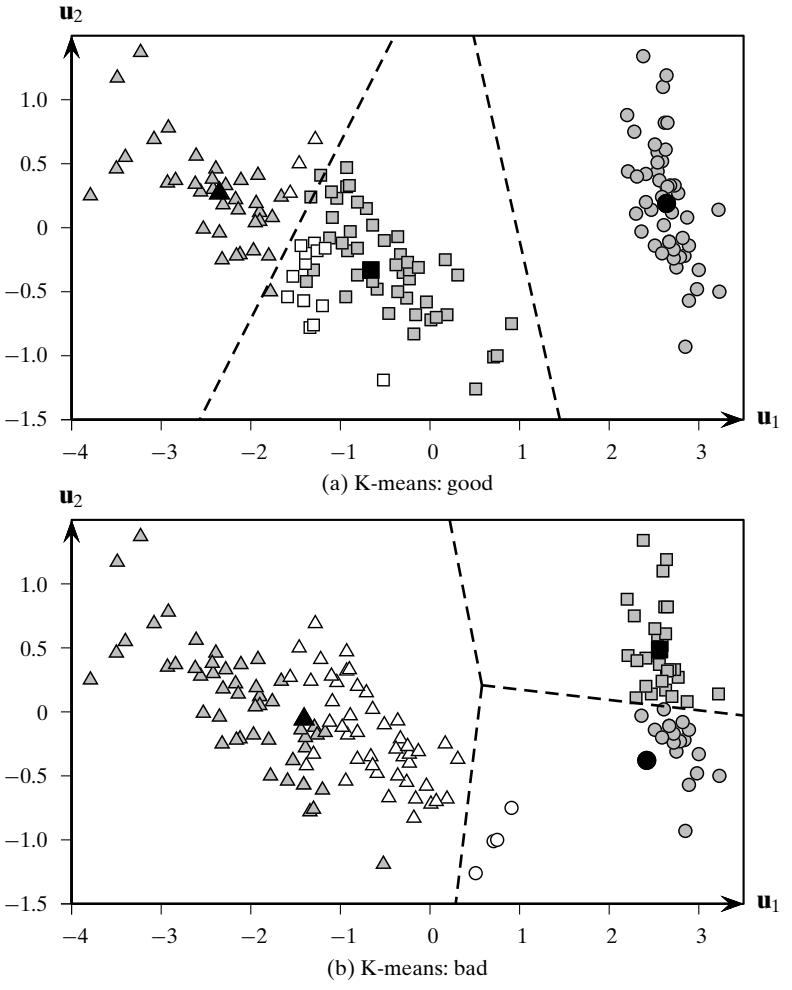


Figure 17.1. K-means: Iris principal components dataset.

We can also directly compute F_1 using Eq. (17.1)

$$F_1 = \frac{2 \cdot n_{12}}{n_1 + m_2} = \frac{2 \cdot 47}{61 + 50} = \frac{94}{111} = 0.85$$

Likewise, we obtain $F_2 = 1.0$ and $F_3 = 0.81$. Thus, the F-measure value for the clustering is given as

$$F = \frac{1}{3}(F_1 + F_2 + F_3) = \frac{2.66}{3} = 0.88$$

For the clustering in Figure 17.1b, we have the following contingency table:

	`iris-setosa`	`iris-versicolor`	`iris-virginica`	
	T_1	T_2	T_3	n_i
C_1	30	0	0	30
C_2	20	4	0	24
C_3	0	46	50	96
m_j	50	50	50	$n = 150$

For the purity measure, the partition with which each cluster shares the most points is given as (C_1, T_1), (C_2, T_1), and (C_3, T_3). Thus, the purity value for this clustering is

$$purity = \frac{1}{150}(30 + 20 + 50) = \frac{100}{150} = 0.67$$

We can see that both C_1 and C_2 choose partition T_1 as the maximum overlapping partition. However, the maximum weight matching is different; it yields the correspondence (C_1, T_1), (C_2, T_2), and (C_3, T_3), and thus

$$match = \frac{1}{150}(30 + 4 + 50) = \frac{84}{150} = 0.56$$

The table below compares the different contingency based measures for the two clusterings shown in Figure 17.1.

	purity	*match*	*F*
(a) Good	0.887	0.887	0.885
(b) Bad	0.667	0.560	0.658

As expected, the good clustering in Figure 17.1a has higher scores for the purity, maximum matching, and F-measure.

17.1.2 Entropy-based Measures

Conditional Entropy

The entropy of a clustering $\mathcal{C}$ is defined as

$$H(\mathcal{C}) = -\sum_{i=1}^{r} p_{C_i} \log p_{C_i}$$

where $p_{C_i} = \frac{n_i}{n}$ is the probability of cluster C_i. Likewise, the entropy of the partitioning $\mathcal{T}$ is defined as

$$H(\mathcal{T}) = -\sum_{j=1}^{k} p_{T_j} \log p_{T_j}$$

where $p_{T_j} = \frac{m_j}{n}$ is the probability of partition T_j.

The cluster-specific entropy of $\mathcal{T}$, that is, the conditional entropy of $\mathcal{T}$ with respect to cluster C_i is defined as

$$H(\mathcal{T}|C_i) = -\sum_{j=1}^{k} \left(\frac{n_{ij}}{n_i}\right) \log \left(\frac{n_{ij}}{n_i}\right)$$

The conditional entropy of $\mathcal{T}$ given clustering $\mathcal{C}$ is then defined as the weighted sum:

$$H(\mathcal{T}|\mathcal{C}) = \sum_{i=1}^{r} \frac{n_i}{n} H(\mathcal{T}|C_i) = -\sum_{i=1}^{r}\sum_{j=1}^{k} \frac{n_{ij}}{n} \log \left(\frac{n_{ij}}{n_i}\right)$$

$$= -\sum_{i=1}^{r}\sum_{j=1}^{k} p_{ij} \log \left(\frac{p_{ij}}{p_{C_i}}\right) \tag{17.2}$$

where $p_{ij} = \frac{n_{ij}}{n}$ is the probability that a point in cluster i also belongs to partition j. The more a cluster's members are split into different partitions, the higher the conditional entropy. For a perfect clustering, the conditional entropy value is zero, whereas the worst possible conditional entropy value is $\log k$. Further, expanding Eq. (17.2), we can see that

$$\begin{aligned} H(\mathcal{T}|\mathcal{C}) &= -\sum_{i=1}^{r}\sum_{j=1}^{k} p_{ij}\left(\log p_{ij} - \log p_{C_i}\right) \\ &= -\left(\sum_{i=1}^{r}\sum_{j=1}^{k} p_{ij}\log p_{ij}\right) + \sum_{i=1}^{r}\left(\log p_{C_i}\sum_{j=1}^{k} p_{ij}\right) \\ &= -\sum_{i=1}^{r}\sum_{j=1}^{k} p_{ij}\log p_{ij} + \sum_{i=1}^{r} p_{C_i}\log p_{C_i} \\ &= H(\mathcal{C},\mathcal{T}) - H(\mathcal{C}) \end{aligned} \tag{17.3}$$

where $H(\mathcal{C},\mathcal{T}) = -\sum_{i=1}^{r}\sum_{j=1}^{k} p_{ij}\log p_{ij}$ is the joint entropy of $\mathcal{C}$ and $\mathcal{T}$. The conditional entropy $H(\mathcal{T}|\mathcal{C})$ thus measures the remaining entropy of $\mathcal{T}$ given the clustering $\mathcal{C}$. In particular, $H(\mathcal{T}|\mathcal{C}) = 0$ if and only if $\mathcal{T}$ is completely determined by $\mathcal{C}$, corresponding to the ideal clustering. On the other hand, if $\mathcal{C}$ and $\mathcal{T}$ are independent of each other, then $H(\mathcal{T}|\mathcal{C}) = H(\mathcal{T})$, which means that $\mathcal{C}$ provides no information about $\mathcal{T}$.

Normalized Mutual Information

The *mutual information* tries to quantify the amount of shared information between the clustering $\mathcal{C}$ and partitioning $\mathcal{T}$, and it is defined as

$$I(\mathcal{C},\mathcal{T}) = \sum_{i=1}^{r}\sum_{j=1}^{k} p_{ij}\log\left(\frac{p_{ij}}{p_{C_i}\cdot p_{T_j}}\right) \tag{17.4}$$

It measures the dependence between the observed joint probability p_{ij} of $\mathcal{C}$ and $\mathcal{T}$, and the expected joint probability $p_{C_i}\cdot p_{T_j}$ under the independence assumption. When $\mathcal{C}$ and $\mathcal{T}$ are independent then $p_{ij} = p_{C_i}\cdot p_{T_j}$, and thus $I(\mathcal{C},\mathcal{T}) = 0$. However, there is no upper bound on the mutual information.

Expanding Eq. (17.4) we observe that $I(\mathcal{C},\mathcal{T}) = H(\mathcal{C}) + H(\mathcal{T}) - H(\mathcal{C},\mathcal{T})$. Using Eq. (17.3), we obtain the two equivalent expressions:

$$\begin{aligned} I(\mathcal{C},\mathcal{T}) &= H(\mathcal{T}) - H(\mathcal{T}|\mathcal{C}) \\ I(\mathcal{C},\mathcal{T}) &= H(\mathcal{C}) - H(\mathcal{C}|\mathcal{T}) \end{aligned}$$

Finally, because $H(\mathcal{C}|\mathcal{T}) \geq 0$ and $H(\mathcal{T}|\mathcal{C}) \geq 0$, we have the inequalities $I(\mathcal{C},\mathcal{T}) \leq H(\mathcal{C})$ and $I(\mathcal{C},\mathcal{T}) \leq H(\mathcal{T})$. We can obtain a normalized version of mutual information by considering the ratios $I(\mathcal{C},\mathcal{T})/H(\mathcal{C})$ and $I(\mathcal{C},\mathcal{T})/H(\mathcal{T})$, both of which can be at

most one. The *normalized mutual information* (NMI) is defined as the geometric mean of these two ratios:

$$NMI(\mathcal{C},\mathcal{T}) = \sqrt{\frac{I(\mathcal{C},\mathcal{T})}{H(\mathcal{C})} \cdot \frac{I(\mathcal{C},\mathcal{T})}{H(\mathcal{T})}} = \frac{I(\mathcal{C},\mathcal{T})}{\sqrt{H(\mathcal{C}) \cdot H(\mathcal{T})}}$$

The NMI value lies in the range $[0,1]$. Values close to 1 indicate a good clustering.

Variation of Information

This criterion is based on the mutual information between the clustering $\mathcal{C}$ and the ground-truth partitioning $\mathcal{T}$, and their entropy; it is defined as

$$\begin{aligned} VI(\mathcal{C},\mathcal{T}) &= (H(\mathcal{T}) - I(\mathcal{C},\mathcal{T})) + (H(\mathcal{C}) - I(\mathcal{C},\mathcal{T})) \\ &= H(\mathcal{T}) + H(\mathcal{C}) - 2I(\mathcal{C},\mathcal{T}) \end{aligned} \tag{17.5}$$

Variation of information (VI) is zero only when $\mathcal{C}$ and $\mathcal{T}$ are identical. Thus, the lower the VI value the better the clustering $\mathcal{C}$.

Using the equivalence $I(\mathcal{C},\mathcal{T}) = H(\mathcal{T}) - H(\mathcal{T}|\mathcal{C}) = H(\mathcal{C}) - H(\mathcal{C}|\mathcal{T})$, we can also express Eq. (17.5) as

$$VI(\mathcal{C},\mathcal{T}) = H(\mathcal{T}|\mathcal{C}) + H(\mathcal{C}|\mathcal{T})$$

Finally, noting that $H(\mathcal{T}|\mathcal{C}) = H(\mathcal{T},\mathcal{C}) - H(\mathcal{C})$, another expression for VI is given as

$$VI(\mathcal{C},\mathcal{T}) = 2H(\mathcal{T},\mathcal{C}) - H(\mathcal{T}) - H(\mathcal{C})$$

Example 17.2. We continue with Example 1, which compares the two clusterings shown in Figure 17.1. For the entropy-based measures, we use base 2 for the logarithms; the formulas are valid for any base as such.

For the clustering in Figure 17.1a, we have the following contingency table:

	`iris-setosa`	`iris-versicolor`	`iris-virginica`	
	T_1	T_2	T_3	n_i
C_1	0	47	14	61
C_2	50	0	0	50
C_3	0	3	36	39
m_j	50	50	50	$n = 100$

Consider the conditional entropy for cluster C_1:

$$\begin{aligned} H(\mathcal{T}|C_1) &= -\frac{0}{61}\log_2\left(\frac{0}{61}\right) - \frac{47}{61}\log_2\left(\frac{47}{61}\right) - \frac{14}{61}\log_2\left(\frac{14}{61}\right) \\ &= -0 - 0.77\log_2(0.77) - 0.23\log_2(0.23) = 0.29 + 0.49 = 0.78 \end{aligned}$$

In a similar manner, we obtain $H(\mathcal{T}|C_2) = 0$ and $H(\mathcal{T}|C_3) = 0.39$. The conditional entropy for the clustering $\mathcal{C}$ is then given as

$$H(\mathcal{T}|\mathcal{C}) = \frac{61}{150} \cdot 0.78 + \frac{50}{150} \cdot 0 + \frac{39}{150} \cdot 0.39 = 0.32 + 0 + 0.10 = 0.42$$

To compute the normalized mutual information, note that

$$H(\mathcal{T}) = -3\left(\frac{50}{150}\log_2\left(\frac{50}{150}\right)\right) = 1.585$$

$$H(\mathcal{C}) = -\left(\frac{61}{150}\log_2\left(\frac{61}{150}\right) + \frac{50}{150}\log_2\left(\frac{50}{150}\right) + \frac{39}{150}\log_2\left(\frac{39}{150}\right)\right)$$
$$= 0.528 + 0.528 + 0.505 = 1.561$$

$$I(\mathcal{C},\mathcal{T}) = \frac{47}{150}\log_2\left(\frac{47\cdot 150}{61\cdot 50}\right) + \frac{14}{150}\log_2\left(\frac{14\cdot 150}{61\cdot 50}\right) + \frac{50}{150}\log_2\left(\frac{50\cdot 150}{50\cdot 50}\right)$$
$$+ \frac{3}{150}\left(\log_2\frac{3\cdot 150}{39\cdot 50}\right) + \frac{36}{150}\log_2\left(\frac{36\cdot 150}{39\cdot 50}\right)$$
$$= 0.379 - 0.05 + 0.528 - 0.042 + 0.353 = 1.167$$

Thus, the NMI and VI values are

$$NMI(\mathcal{C},\mathcal{T}) = \frac{I(\mathcal{C},\mathcal{T})}{\sqrt{H(\mathcal{T})\cdot H(\mathcal{C})}} = \frac{1.167}{\sqrt{1.585\times 1.561}} = 0.742$$

$$VI(\mathcal{C},\mathcal{T}) = H(\mathcal{T}) + H(\mathcal{C}) - 2I(\mathcal{C},\mathcal{T}) = 1.585 + 1.561 - 2\cdot 1.167 = 0.812$$

We can likewise compute these measures for the other clustering in Figure 17.1b, whose contingency table is shown in Example 1.

The table below compares the entropy based measures for the two clusterings shown in Figure 17.1.

	$H(\mathcal{T}\|\mathcal{C})$	NMI	VI
(a) Good	0.418	0.742	0.812
(b) Bad	0.743	0.587	1.200

As expected, the good clustering in Figure 17.1a has a higher score for normalized mutual information, and lower scores for conditional entropy and variation of information.

17.1.3 Pairwise Measures

Given clustering $\mathcal{C}$ and ground-truth partitioning $\mathcal{T}$, the pairwise measures utilize the partition and cluster label information over all pairs of points. Let $\mathbf{x}_i, \mathbf{x}_j \in \mathbf{D}$ be any two points, with $i \neq j$. Let y_i denote the true partition label and let $\hat{y}_i$ denote the cluster label for point $\mathbf{x}_i$. If both $\mathbf{x}_i$ and $\mathbf{x}_j$ belong to the same cluster, that is, $\hat{y}_i = \hat{y}_j$, we call it a *positive* event, and if they do not belong to the same cluster, that is, $\hat{y}_i \neq \hat{y}_j$, we call that a *negative* event. Depending on whether there is agreement between the cluster labels and partition labels, there are four possibilities to consider:

- *True Positives:* $\mathbf{x}_i$ and $\mathbf{x}_j$ belong to the same partition in $\mathcal{T}$, and they are also in the same cluster in $\mathcal{C}$. This is a true positive pair because the positive event, $\hat{y}_i = \hat{y}_j$, corresponds to the ground truth, $y_i = y_j$. The number of true positive pairs is given as

$$TP = \left|\{(\mathbf{x}_i, \mathbf{x}_j) : \; y_i = y_j \text{ and } \hat{y}_i = \hat{y}_j\}\right|$$

- *False Negatives:* $\mathbf{x}_i$ and $\mathbf{x}_j$ belong to the same partition in $\mathcal{T}$, but they do not belong to the same cluster in $\mathcal{C}$. That is, the negative event, $\hat{y}_i \neq \hat{y}_j$, does not correspond to the truth, $y_i = y_j$. This pair is thus a false negative, and the number of all false negative pairs is given as

$$FN = \left|\{(\mathbf{x}_i, \mathbf{x}_j) : \; y_i = y_j \text{ and } \hat{y}_i \neq \hat{y}_j\}\right|$$

- *False Positives:* $\mathbf{x}_i$ and $\mathbf{x}_j$ do not belong to the same partition in $\mathcal{T}$, but they do belong to the same cluster in $\mathcal{C}$. This pair is a false positive because the positive event, $\hat{y}_i = \hat{y}_j$, is actually false, that is, it does not agree with the ground-truth partitioning, which indicates that $y_i \neq y_j$. The number of false positive pairs is given as

$$FP = \left|\{(\mathbf{x}_i, \mathbf{x}_j) : \; y_i \neq y_j \text{ and } \hat{y}_i = \hat{y}_j\}\right|$$

- *True Negatives:* $\mathbf{x}_i$ and $\mathbf{x}_j$ neither belong to the same partition in $\mathcal{T}$, nor do they belong to the same cluster in $\mathcal{C}$. This pair is thus a true negative, that is, $\hat{y}_i \neq \hat{y}_j$ and $y_i \neq y_j$. The number of such true negative pairs is given as

$$TN = \left|\{(\mathbf{x}_i, \mathbf{x}_j) : \; y_i \neq y_j \text{ and } \hat{y}_i \neq \hat{y}_j\}\right|$$

Because there are $N = \binom{n}{2} = \frac{n(n-1)}{2}$ pairs of points, we have the following identity:

$$N = TP + FN + FP + TN \tag{17.6}$$

A naive computation of the preceding four cases requires $O(n^2)$ time. However, they can be computed more efficiently using the contingency table $\mathbf{N} = \{n_{ij}\}$, with $1 \leq i \leq r$ and $1 \leq j \leq k$. The number of true positives is given as

$$\begin{aligned} TP &= \sum_{i=1}^{r}\sum_{j=1}^{k}\binom{n_{ij}}{2} = \sum_{i=1}^{r}\sum_{j=1}^{k}\frac{n_{ij}(n_{ij}-1)}{2} = \frac{1}{2}\left(\sum_{i=1}^{r}\sum_{j=1}^{k}n_{ij}^2 - \sum_{i=1}^{r}\sum_{j=1}^{k}n_{ij}\right) \\ &= \frac{1}{2}\left(\left(\sum_{i=1}^{r}\sum_{j=1}^{k}n_{ij}^2\right) - n\right) \end{aligned} \tag{17.7}$$

This follows from the fact that each pair of points among the n_{ij} share the same cluster label (i) and the same partition label (j). The last step follows from the fact that the sum of all the entries in the contingency table must add to n, that is, $\sum_{i=1}^{r}\sum_{j=1}^{k} n_{ij} = n$.

To compute the total number of false negatives, we remove the number of true positives from the number of pairs that belong to the same partition. Because two points $\mathbf{x}_i$ and $\mathbf{x}_j$ that belong to the same partition have $y_i = y_j$, if we remove the true positives, that is, pairs with $\hat{y}_i = \hat{y}_j$, we are left with pairs for whom $\hat{y}_i \neq \hat{y}_j$, that is, the false negatives. We thus have

$$\begin{aligned} FN &= \sum_{j=1}^{k}\binom{m_j}{2} - TP = \frac{1}{2}\left(\sum_{j=1}^{k}m_j^2 - \sum_{j=1}^{k}m_j - \sum_{i=1}^{r}\sum_{j=1}^{k}n_{ij}^2 + n\right) \\ &= \frac{1}{2}\left(\sum_{j=1}^{k}m_j^2 - \sum_{i=1}^{r}\sum_{j=1}^{k}n_{ij}^2\right) \end{aligned} \tag{17.8}$$

The last step follows from the fact that $\sum_{j=1}^{k} m_j = n$.

The number of false positives can be obtained in a similar manner by subtracting the number of true positives from the number of point pairs that are in the same cluster:

$$FP = \sum_{i=1}^{r} \binom{n_i}{2} - TP = \frac{1}{2}\left(\sum_{i=1}^{r} n_i^2 - \sum_{i=1}^{r}\sum_{j=1}^{k} n_{ij}^2\right) \tag{17.9}$$

Finally, the number of true negatives can be obtained via Eq. (17.6) as follows:

$$TN = N - (TP + FN + FP) = \frac{1}{2}\left(n^2 - \sum_{i=1}^{r} n_i^2 - \sum_{j=1}^{k} m_j^2 + \sum_{i=1}^{r}\sum_{j=1}^{k} n_{ij}^2\right) \tag{17.10}$$

Each of the four values can be computed in $O(rk)$ time. Because the contingency table can be obtained in linear time, the total time to compute the four values is $O(n + rk)$, which is much better than the naive $O(n^2)$ bound. We next consider pairwise assessment measures based on these four values.

Jaccard Coefficient

The Jaccard Coefficient measures the fraction of true positive point pairs, but after ignoring the true negatives. It is defined as follows:

$$Jaccard = \frac{TP}{TP + FN + FP} \tag{17.11}$$

For a perfect clustering $\mathcal{C}$ (i.e., total agreement with the partitioning $\mathcal{T}$), the Jaccard Coefficient has value 1, as in that case there are no false positives or false negatives. The Jaccard coefficient is asymmetric in terms of the true positives and negatives because it ignores the true negatives. In other words, it emphasizes the similarity in terms of the point pairs that belong together in both the clustering and ground-truth partitioning, but it discounts the point pairs that do not belong together.

Rand Statistic

The Rand statistic measures the fraction of true positives and true negatives over all point pairs; it is defined as

$$Rand = \frac{TP + TN}{N} \tag{17.12}$$

The Rand statistic, which is symmetric, measures the fraction of point pairs where both $\mathcal{C}$ and $\mathcal{T}$ agree. A prefect clustering has a value of 1 for the statistic.

Fowlkes-Mallows Measure

Define the overall *pairwise precision* and *pairwise recall* values for a clustering $\mathcal{C}$, as follows:

$$prec = \frac{TP}{TP + FP} \qquad recall = \frac{TP}{TP + FN}$$

Precision measures the fraction of true or correctly clustered point pairs compared to all the point pairs in the same cluster. On the other hand, recall measures the fraction of correctly labeled points pairs compared to all the point pairs in the same partition.

The Fowlkes–Mallows (FM) measure is defined as the geometric mean of the pairwise precision and recall

$$FM = \sqrt{prec \cdot recall} = \frac{TP}{\sqrt{(TP+FN)(TP+FP)}} \tag{17.13}$$

The FM measure is also asymmetric in terms of the true positives and negatives because it ignores the true negatives. Its highest value is also 1, achieved when there are no false positives or negatives.

Example 17.3. Let us continue with Example 1. Consider again the contingency table for the clustering in Figure 17.1a:

	iris-setosa T_1	**iris-versicolor** T_2	**iris-virginica** T_3
C_1	0	47	14
C_2	50	0	0
C_3	0	3	36

Using Eq. (17.7), we can obtain the number of true positives as follows:

$$TP = \binom{47}{2} + \binom{14}{2} + \binom{50}{2} + \binom{3}{2} + \binom{36}{2}$$
$$= 1081 + 91 + 1225 + 3 + 630 = 3030$$

Using Eqs. (17.8), (17.9), and (17.10), we obtain

$$FN = 645 \qquad FP = 766 \qquad TN = 6734$$

Note that there are a total of $N = \binom{150}{2} = 11175$ point pairs.

We can now compute the different pairwise measures for clustering evaluation. The Jaccard coefficient [Eq. (17.11)], Rand statistic [Eq. (17.12)], and Fowlkes–Mallows measure [Eq. (17.13)], are given as

$$Jaccard = \frac{3030}{3030+645+766} = \frac{3030}{4441} = 0.68$$
$$Rand = \frac{3030+6734}{11175} = \frac{9764}{11175} = 0.87$$
$$FM = \frac{3030}{\sqrt{3675 \cdot 3796}} = \frac{3030}{3735} = 0.81$$

Using the contingency table for the clustering in Figure 17.1b from Example 1, we obtain

$$TP = 2891 \qquad FN = 784 \qquad FP = 2380 \qquad TN = 5120$$

The table below compares the different contingency based measures on the two clusterings in Figure 17.1.

	Jaccard	*Rand*	*FM*
(a) Good	0.682	0.873	0.811
(b) Bad	0.477	0.717	0.657

As expected, the clustering in Figure 17.1a has higher scores for all three measures.

17.1.4 Correlation Measures

Let $\mathbf{X}$ and $\mathbf{Y}$ be two symmetric $n \times n$ matrices, and let $N = \binom{n}{2}$. Let $\mathbf{x}, \mathbf{y} \in \mathbb{R}^N$ denote the vectors obtained by linearizing the upper triangular elements (excluding the main diagonal) of $\mathbf{X}$ and $\mathbf{Y}$ (e.g., in a row-wise manner), respectively. Let μ_X denote the element-wise mean of $\mathbf{x}$, given as

$$\mu_X = \frac{1}{N} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathbf{X}(i,j) = \frac{1}{N} \mathbf{x}^T \mathbf{x}$$

and let $\mathbf{z}_x$ denote the centered $\mathbf{x}$ vector, defined as

$$\mathbf{z}_x = \mathbf{x} - \mathbf{1} \cdot \mu_X$$

where $\mathbf{1} \in \mathbb{R}^N$ is the vector of all ones. Likewise, let μ_Y be the element-wise mean of $\mathbf{y}$, and $\mathbf{z}_y$ the centered $\mathbf{y}$ vector.

The Hubert statistic is defined as the averaged element-wise product between $\mathbf{X}$ and $\mathbf{Y}$

$$\Gamma = \frac{1}{N} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathbf{X}(i,j) \cdot \mathbf{Y}(i,j) = \frac{1}{N} \mathbf{x}^T \mathbf{y} \tag{17.14}$$

The normalized Hubert statistic is defined as the element-wise correlation between $\mathbf{X}$ and $\mathbf{Y}$

$$\Gamma_n = \frac{\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \big(\mathbf{X}(i,j) - \mu_X\big)\big(\cdot \mathbf{Y}(i,j) - \mu_Y\big)}{\sqrt{\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \big(\mathbf{X}(i,j) - \mu_X\big)^2 \quad \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \big(\mathbf{Y}[i] - \mu_Y\big)^2}} = \frac{\sigma_{XY}}{\sqrt{\sigma_X^2 \sigma_Y^2}}$$

where σ_X^2 and σ_Y^2 are the variances, and σ_{XY} the covariance, for the vectors $\mathbf{x}$ and $\mathbf{y}$, defined as

$$\sigma_X^2 = \frac{1}{N} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \big(\mathbf{X}(i,j) - \mu_X\big)^2 = \frac{1}{N} \mathbf{z}_x^T \mathbf{z}_x = \frac{1}{N} \|\mathbf{z}_x\|^2$$

$$\sigma_Y^2 = \frac{1}{N} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \big(\mathbf{Y}(i,j) - \mu_Y\big)^2 = \frac{1}{N} \mathbf{z}_y^T \mathbf{z}_y = \frac{1}{N} \|\mathbf{z}_y\|^2$$

$$\sigma_{XY} = \frac{1}{N} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \big(\mathbf{X}(i,j) - \mu_X\big)\big(\mathbf{Y}(i,j) - \mu_Y\big) = \frac{1}{N} \mathbf{z}_x^T \mathbf{z}_y$$

Thus, the normalized Hubert statistic can be rewritten as

$$\Gamma_n = \frac{\mathbf{z}_x^T \mathbf{z}_y}{\|\mathbf{z}_x\| \cdot \|\mathbf{z}_y\|} = \cos\theta \tag{17.15}$$

where θ is the angle between the two centered vectors $\mathbf{z}_x$ and $\mathbf{z}_y$. It follows immediately that Γ_n ranges from -1 to $+1$.

When $\mathbf{X}$ and $\mathbf{Y}$ are arbitrary $n \times n$ matrices the above expressions can be easily modified to range over all the n^2 elements of the two matrices. The (normalized) Hubert statistic can be used as an external evaluation measure, with appropriately defined matrices $\mathbf{X}$ and $\mathbf{Y}$, as described next.

Discretized Hubert Statistic

Let $\mathbf{T}$ and $\mathbf{C}$ be the $n \times n$ matrices defined as

$$\mathbf{T}(i,j) = \begin{cases} 1 & \text{if } y_i = y_j, i \neq j \\ 0 & \text{otherwise} \end{cases} \qquad \mathbf{C}(i,j) = \begin{cases} 1 & \text{if } \hat{y}_i = \hat{y}_j, i \neq j \\ 0 & \text{otherwise} \end{cases}$$

Also, let $\mathbf{t}, \mathbf{c} \in \mathbb{R}^N$ denote the N-dimensional vectors comprising the upper triangular elements (excluding the diagonal) of $\mathbf{T}$ and $\mathbf{C}$, respectively, where $N = \binom{n}{2}$ denotes the number of distinct point pairs. Finally, let $\mathbf{z}_t$ and $\mathbf{z}_c$ denote the centered $\mathbf{t}$ and $\mathbf{c}$ vectors.

The discretized Hubert statistic is computed via Eq. (17.14), by setting $\mathbf{x} = \mathbf{t}$ and $\mathbf{y} = \mathbf{c}$:

$$\Gamma = \frac{1}{N}\mathbf{t}^T\mathbf{c} = \frac{TP}{N} \tag{17.16}$$

Because the ith element of $\mathbf{t}$ is 1 only when the ith pair of points belongs to the same partition, and, likewise, the ith element of $\mathbf{c}$ is 1 only when the ith pair of points also belongs to the same cluster, the dot product $\mathbf{t}^T\mathbf{c}$ is simply the number of true positives, and thus the Γ value is equivalent to the fraction of all pairs that are true positives. It follows that the higher the agreement between the ground-truth partitioning $\mathcal{T}$ and clustering $\mathcal{C}$, the higher the Γ value.

Normalized Discretized Hubert Statistic

The normalized version of the discretized Hubert statistic is simply the correlation between $\mathbf{t}$ and $\mathbf{c}$ [Eq. (17.15)]:

$$\Gamma_n = \frac{\mathbf{z}_t^T\mathbf{z}_c}{\|\mathbf{z}_t\| \cdot \|\mathbf{z}_c\|} = \cos\theta \tag{17.17}$$

Note that $\mu_T = \frac{1}{N}\mathbf{t}^T\mathbf{t}$ is the fraction of point pairs that belong to the same partition, that is, with $y_i = y_j$, regardless of whether $\hat{y}_i$ matches $\hat{y}_j$ or not. Thus, we have

$$\mu_T = \frac{\mathbf{t}^T\mathbf{t}}{N} = \frac{TP + FN}{N}$$

Similarly, $\mu_C = \frac{1}{N}\mathbf{c}^T\mathbf{c}$ is the fraction of point pairs that belong to the same cluster, that is, with $\hat{y}_i = \hat{y}_j$, regardless of whether y_i matches y_j or not, so that

$$\mu_C = \frac{\mathbf{c}^T\mathbf{c}}{N} = \frac{TP + FP}{N}$$

Substituting these into the numerator in Eq. (17.17), we get

$$\begin{aligned}\mathbf{z}_t^T\mathbf{z}_c &= (\mathbf{t}-\mathbf{1}\cdot\mu_T)^T(\mathbf{c}-\mathbf{1}\cdot\mu_C)\\ &= \mathbf{t}^T\mathbf{c}-\mu_C\mathbf{t}^T\mathbf{1}-\mu_T\mathbf{c}^T\mathbf{1}+\mathbf{1}^T\mathbf{1}\mu_T\mu_C\\ &= \mathbf{t}^T\mathbf{c}-N\mu_C\mu_T-N\mu_T\mu_C+N\mu_T\mu_C\\ &= \mathbf{t}^T\mathbf{c}-N\mu_T\mu_C\\ &= TP-N\mu_T\mu_C \end{aligned} \tag{17.18}$$

where $\mathbf{1}\in\mathbb{R}^N$ is the vector of all 1's. We also made use of identities $\mathbf{t}^T\mathbf{1}=\mathbf{t}^T\mathbf{t}$ and $\mathbf{c}^T\mathbf{1}=\mathbf{c}^T\mathbf{c}$. Likewise, we can derive

$$\|\mathbf{z}_t\|^2=\mathbf{z}_t^T\mathbf{z}_t=\mathbf{t}^T\mathbf{t}-N\mu_T^2=N\mu_T-N\mu_T^2=N\mu_T(1-\mu_T) \tag{17.19}$$

$$\|\mathbf{z}_c\|^2=\mathbf{z}_c^T\mathbf{z}_c=\mathbf{c}^T\mathbf{c}-N\mu_C^2=N\mu_C-N\mu_C^2=N\mu_C(1-\mu_C) \tag{17.20}$$

Plugging Eqs. (17.18), (17.19), and (17.20) into Eq. (17.17) the normalized, discretized Hubert statistic can be written as

$$\Gamma_n=\frac{\frac{TP}{N}-\mu_T\mu_C}{\sqrt{\mu_T\mu_C(1-\mu_T)(1-\mu_C)}} \tag{17.21}$$

because $\mu_T=\frac{TP+FN}{N}$ and $\mu_C=\frac{TP+FP}{N}$, the normalized Γ_n statistic can be computed using only the TP, FN, and FP values. The maximum value of $\Gamma_n=+1$ is obtained when there are no false positives or negatives, that is, when $FN=FP=0$. The minimum value of $\Gamma_n=-1$ is when there are no true positives and negatives, that is, when $TP=TN=0$.

Example 17.4. Continuing Example 17.3, for the good clustering in Figure 17.1a, we have

$$TP=3030 \qquad FN=645 \qquad FP=766 \qquad TN=6734$$

From these values, we obtain

$$\mu_T=\frac{TP+FN}{N}=\frac{3675}{11175}=0.33$$

$$\mu_C=\frac{TP+FP}{N}=\frac{3796}{11175}=0.34$$

Using Eqs. (17.16) and (17.21) the Hubert statistic values are

$$\Gamma=\frac{3030}{11175}=0.271$$

$$\Gamma_n=\frac{0.27-0.33\cdot0.34}{\sqrt{0.33\cdot0.34\cdot(1-0.33)\cdot(1-0.34)}}=\frac{0.159}{0.222}=0.717$$

Likewise, for the bad clustering in Figure 17.1b, we have

$$TP=2891 \qquad FN=784 \qquad FP=2380 \qquad TN=5120$$

and the values for the discretized Hubert statistic are given as

$$\Gamma = 0.258 \qquad \Gamma_n = 0.442$$

We observe that the good clustering has higher values, though the normalized statistic is more discerning than the unnormalized version, that is, the good clustering has a much higher value of Γ_n than the bad clustering, whereas the difference in Γ for the two clusterings is not that high.

17.2 INTERNAL MEASURES

Internal evaluation measures do not have recourse to the ground-truth partitioning, which is the typical scenario when clustering a dataset. To evaluate the quality of the clustering, internal measures therefore have to utilize notions of intracluster similarity or compactness, contrasted with notions of intercluster separation, with usually a trade-off in maximizing these two aims. The internal measures are based on the $n \times n$ *distance matrix*, also called the *proximity matrix*, of all pairwise distances among the n points:

$$\mathbf{W} = \Big\{\delta(\mathbf{x}_i, \mathbf{x}_j)\Big\}_{i,j=1}^{n} \tag{17.22}$$

where

$$\delta(\mathbf{x}_i, \mathbf{x}_j) = \big\|\mathbf{x}_i - \mathbf{x}_j\big\|_2$$

is the Euclidean distance between $\mathbf{x}_i, \mathbf{x}_j \in \mathbf{D}$, although other distance metrics can also be used. Because $\mathbf{W}$ is symmetric and $\delta(\mathbf{x}_i, \mathbf{x}_i) = 0$, usually only the upper triangular elements of $\mathbf{W}$ (excluding the diagonal) are used in the internal measures.

The proximity matrix $\mathbf{W}$ can also be considered as the adjacency matrix of the weighted complete graph G over the n points, that is, with nodes $V = \{\mathbf{x}_i \mid \mathbf{x}_i \in \mathbf{D}\}$, edges $E = \{(\mathbf{x}_i, \mathbf{x}_j) \mid \mathbf{x}_i, \mathbf{x}_j \in \mathbf{D}\}$, and edge weights $w_{ij} = \mathbf{W}(i, j)$ for all $\mathbf{x}_i, \mathbf{x}_j \in \mathbf{D}$. There is thus a close connection between the internal evaluation measures and the graph clustering objectives we examined in Chapter 16.

For internal measures, we assume that we do not have access to a ground-truth partitioning. Instead, we assume that we are given a clustering $\mathcal{C} = \{C_1, \ldots, C_k\}$ comprising $r = k$ clusters, with cluster C_i containing $n_i = |C_i|$ points. Let $\hat{y}_i \in \{1, 2, \ldots, k\}$ denote the cluster label for point $\mathbf{x}_i$. The clustering $\mathcal{C}$ can be considered as a k-way cut in G because $C_i \neq \emptyset$ for all i, $C_i \cap C_j = \emptyset$ for all i, j, and $\bigcup_i C_i = V$. Given any subsets $S, R \subset V$, define $W(S, R)$ as the sum of the weights on all edges with one vertex in S and the other in R, given as

$$W(S, R) = \sum_{\mathbf{x}_i \in S} \sum_{\mathbf{x}_j \in R} w_{ij}$$

Also, given $S \subseteq V$, we denote by $\overline{S}$ the complementary set of vertices, that is, $\overline{S} = V - S$.

The internal measures are based on various functions over the intracluster and intercluster weights. In particular, note that the sum of all the intracluster weights over

all clusters is given as

$$W_{in} = \frac{1}{2}\sum_{i=1}^{k} W(C_i, C_i) \tag{17.23}$$

We divide by 2 because each edge within C_i is counted twice in the summation given by $W(C_i, C_i)$. Also note that the sum of all intercluster weights is given as

$$W_{out} = \frac{1}{2}\sum_{i=1}^{k} W(C_i, \overline{C_i}) = \sum_{i=1}^{k-1}\sum_{j>i} W(C_i, C_j) \tag{17.24}$$

Here too we divide by 2 because each edge is counted twice in the summation across clusters. The number of distinct intracluster edges, denoted N_{in}, and intercluster edges, denoted N_{out}, are given as

$$N_{in} = \sum_{i=1}^{k}\binom{n_i}{2} = \frac{1}{2}\sum_{i=1}^{k} n_i(n_i - 1)$$

$$N_{out} = \sum_{i=1}^{k-1}\sum_{j=i+1}^{k} n_i \cdot n_j = \frac{1}{2}\sum_{i=1}^{k}\sum_{\substack{j=1 \\ j\neq i}}^{k} n_i \cdot n_j$$

Note that the total number of distinct pairs of points N satisfies the identity

$$N = N_{in} + N_{out} = \binom{n}{2} = \frac{1}{2}n(n-1)$$

Example 17.5. Figure 17.2 shows the graphs corresponding to the two K-means clusterings shown in Figure 17.1. Here, each vertex corresponds to a point $\mathbf{x}_i \in \mathbf{D}$, and an edge $(\mathbf{x}_i, \mathbf{x}_j)$ exists between each pair of points. However, only the intracluster edges are shown (with intercluster edges omitted) to avoid clutter. Because internal measures do not have access to a ground truth labeling, the goodness of a clustering is measured based on intracluster and intercluster statistics.

BetaCV Measure

The BetaCV measure is the ratio of the mean intracluster distance to the mean intercluster distance:

$$BetaCV = \frac{W_{in}/N_{in}}{W_{out}/N_{out}} = \frac{N_{out}}{N_{in}} \cdot \frac{W_{in}}{W_{out}} = \frac{N_{out}}{N_{in}} \frac{\sum_{i=1}^{k} W(C_i, C_i)}{\sum_{i=1}^{k} W(C_i, \overline{C_i})}$$

The smaller the BetaCV ratio, the better the clustering, as it indicates that intracluster distances are on average smaller than intercluster distances.

C-index

Let $W_{\min}(N_{in})$ be the sum of the smallest N_{in} distances in the proximity matrix $\mathbf{W}$, where N_{in} is the total number of intracluster edges, or point pairs. Let $W_{\max}(N_{in})$ be the sum of the largest N_{in} distances in $\mathbf{W}$.

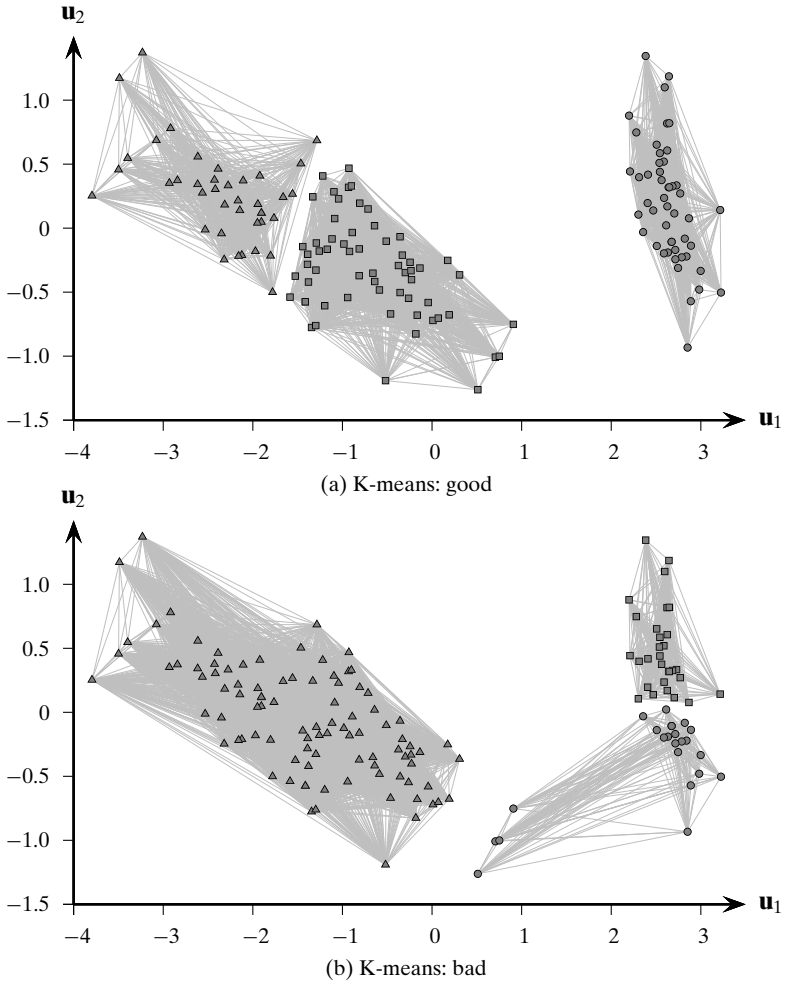


Figure 17.2. Clusterings as graphs: Iris.

The C-index measures to what extent the clustering puts together the N_{in} points that are the closest across the k clusters. It is defined as

$$Cindex = \frac{W_{in} - W_{\min}(N_{in})}{W_{\max}(N_{in}) - W_{\min}(N_{in})}$$

where W_{in} is the sum of all the intracluster distances [Eq. (17.23)]. The C-index lies in the range $[0, 1]$. The smaller the C-index, the better the clustering, as it indicates more compact clusters with relatively smaller distances within clusters rather than between clusters.

Normalized Cut Measure

The normalized cut objective [Eq. (16.17)] for graph clustering can also be used as an internal clustering evaluation measure:

$$NC = \sum_{i=1}^{k} \frac{W(C_i, \overline{C_i})}{vol(C_i)} = \sum_{i=1}^{k} \frac{W(C_i, \overline{C_i})}{W(C_i, V)}$$

where $vol(C_i) = W(C_i, V)$ is the volume of cluster C_i, that is, the total weights on edges with at least one end in the cluster. However, because we are using the proximity or distance matrix $\mathbf{W}$, instead of the affinity or similarity matrix $\mathbf{A}$, the higher the normalized cut value the better.

To see this, we make use of the observation that $W(C_i, V) = W(C_i, C_i) + W(C_i, \overline{C_i})$, so that

$$NC = \sum_{i=1}^{k} \frac{W(C_i, \overline{C_i})}{W(C_i, C_i) + W(C_i, \overline{C_i})} = \sum_{i=1}^{k} \frac{1}{\frac{W(C_i, C_i)}{W(C_i, \overline{C_i})} + 1}$$

We can see that NC is maximized when the ratios $\frac{W(C_i, C_i)}{W(C_i, \overline{C_i})}$ (across the k clusters) are as small as possible, which happens when the intracluster distances are much smaller compared to intercluster distances, that is, when the clustering is good. The maximum possible value of NC is k.

Modularity

The modularity objective for graph clustering [Eq. (16.26)] can also be used as an internal measure:

$$Q = \sum_{i=1}^{k} \left(\frac{W(C_i, C_i)}{W(V, V)} - \left(\frac{W(C_i, V)}{W(V, V)} \right)^2 \right)$$

where

$$\begin{aligned} W(V, V) &= \sum_{i=1}^{k} W(C_i, V) \\ &= \sum_{i=1}^{k} W(C_i, C_i) + \sum_{i=1}^{k} W(C_i, \overline{C_i}) \\ &= 2(W_{in} + W_{out}) \end{aligned}$$

The last step follows from Eqs. (17.23) and (17.24). Modularity measures the difference between the observed and expected fraction of weights on edges within the clusters. Since we are using the distance matrix, the smaller the modularity measure the better the clustering, which indicates that the intracluster distances are lower than expected.

Dunn Index

The Dunn index is defined as the ratio between the minimum distance between point pairs from different clusters and the maximum distance between point pairs from the same cluster. More formally, we have

$$Dunn = \frac{W_{out}^{\min}}{W_{in}^{\max}}$$

where $W_{out}^{\min}$ is the minimum intercluster distance:

$$W_{out}^{\min} = \min_{i, j>i} \left\{ w_{ab} | \mathbf{x}_a \in C_i, \mathbf{x}_b \in C_j \right\}$$

and $W_{in}^{\max}$ is the maximum intracluster distance:

$$W_{in}^{\max} = \max_i \left\{ w_{ab} | \mathbf{x}_a, \mathbf{x}_b \in C_i \right\}$$

The larger the Dunn index the better the clustering because it means even the closest distance between points in different clusters is much larger than the farthest distance between points in the same cluster. However, the Dunn index may be insensitive because the minimum intercluster and maximum intracluster distances do not capture all the information about a clustering.

Davies–Bouldin Index

Let μ_i denote the cluster mean, given as

$$\mu_i = \frac{1}{n_i} \sum_{\mathbf{x}_j \in C_i} \mathbf{x}_j \tag{17.25}$$

Further, let σ_{μ_i} denote the dispersion or spread of the points around the cluster mean, given as

$$\sigma_{\mu_i} = \sqrt{\frac{\sum_{\mathbf{x}_j \in C_i} \delta(\mathbf{x}_j, \mu_i)^2}{n_i}} = \sqrt{var(C_i)}$$

where $var(C_i)$ is the total variance [Eq. (1.4)] of cluster C_i.

The Davies–Bouldin measure for a pair of clusters C_i and C_j is defined as the ratio

$$DB_{ij} = \frac{\sigma_{\mu_i} + \sigma_{\mu_j}}{\delta(\mu_i, \mu_j)}$$

DB_{ij} measures how compact the clusters are compared to the distance between the cluster means. The Davies–Bouldin index is then defined as

$$DB = \frac{1}{k} \sum_{i=1}^{k} \max_{j \neq i} \{DB_{ij}\}$$

That is, for each cluster C_i, we pick the cluster C_j that yields the largest DB_{ij} ratio. The smaller the DB value the better the clustering, as it means that the clusters are well separated (i.e., the distance between cluster means is large), and each cluster is well represented by its mean (i.e., has a small spread).

Silhouette Coefficient

The silhouette coefficient is a measure of both cohesion and separation of clusters, and is based on the difference between the average distance to points in the closest cluster and to points in the same cluster. For each point $\mathbf{x}_i$ we calculate its silhouette coefficient s_i as

$$s_i = \frac{\mu_{out}^{\min}(\mathbf{x}_i) - \mu_{in}(\mathbf{x}_i)}{\max\left\{ \mu_{out}^{\min}(\mathbf{x}_i), \mu_{in}(\mathbf{x}_i) \right\}} \tag{17.26}$$

where $\mu_{in}(\mathbf{x}_i)$ is the mean distance from $\mathbf{x}_i$ to points in its own cluster $\hat{y}_i$:

$$\mu_{in}(\mathbf{x}_i) = \frac{\sum_{\mathbf{x}_j \in C_{\hat{y}_i}, j \neq i} \delta(\mathbf{x}_i, \mathbf{x}_j)}{n_{\hat{y}_i} - 1}$$

and $\mu_{out}^{\min}(\mathbf{x}_i)$ is the mean of the distances from $\mathbf{x}_i$ to points in the closest cluster:

$$\mu_{out}^{\min}(\mathbf{x}_i) = \min_{j \neq \hat{y}_i} \left\{ \frac{\sum_{\mathbf{y} \in C_j} \delta(\mathbf{x}_i, \mathbf{y})}{n_j} \right\}$$

The s_i value of a point lies in the interval $[-1, +1]$. A value close to $+1$ indicates that $\mathbf{x}_i$ is much closer to points in its own cluster and is far from other clusters. A value close to zero indicates that $\mathbf{x}_i$ is close to the boundary between two clusters. Finally, a value close to -1 indicates that $\mathbf{x}_i$ is much closer to another cluster than its own cluster, and therefore, the point may be mis-clustered.

The silhouette coefficient is defined as the mean s_i value across all the points:

$$SC = \frac{1}{n} \sum_{i=1}^{n} s_i \tag{17.27}$$

A value close to $+1$ indicates a good clustering.

Hubert Statistic

The Hubert Γ statistic [Eq. (17.14)], and its normalized version Γ_n [Eq. (17.15)], can both be used as internal evaluation measures by letting $\mathbf{X} = \mathbf{W}$ be the pairwise distance matrix, and by defining $\mathbf{Y}$ as the matrix of distances between the cluster means:

$$\mathbf{Y} = \left\{ \delta(\mu_{\hat{y}_i}, \mu_{\hat{y}_j}) \right\}_{i,j=1}^{n} \tag{17.28}$$

Because both $\mathbf{W}$ and $\mathbf{Y}$ are symmetric, both Γ and Γ_n are computed over their upper triangular elements.

Example 17.6. Consider the two clusterings for the Iris principal components dataset shown in Figure 17.1, along with their corresponding graph representations in Figure 17.2. Let us evaluate these two clusterings using internal measures.

The good clustering shown in Figure 17.1a and Figure 17.2a has clusters with the following sizes:

$$n_1 = 61 \qquad n_2 = 50 \qquad n_3 = 39$$

Thus, the number of intracluster and intercluster edges (i.e., point pairs) is given as

$$N_{in} = \binom{61}{2} + \binom{50}{2} + \binom{31}{2} = 1830 + 1225 + 741 = 3796$$

$$N_{out} = 61 \cdot 50 + 61 \cdot 39 + 50 \cdot 39 = 3050 + 2379 + 1950 = 7379$$

In total there are $N = N_{in} + N_{out} = 3796 + 7379 = 11175$ distinct point pairs.

The weights on edges within each cluster $W(C_i, C_i)$, and those from a cluster to another $W(C_i, C_j)$, are as given in the intercluster weight matrix

$$\begin{pmatrix} W & C_1 & C_2 & C_3 \\ C_1 & 3265.69 & 10402.30 & 4418.62 \\ C_2 & 10402.30 & 1523.10 & 9792.45 \\ C_3 & 4418.62 & 9792.45 & 1252.36 \end{pmatrix} \tag{17.29}$$

Thus, the sum of all the intracluster and intercluster edge weights is

$$W_{in} = \frac{1}{2}(3265.69 + 1523.10 + 1252.36) = 3020.57$$
$$W_{out} = (10402.30 + 4418.62 + 9792.45) = 24613.37$$

The BetaCV measure can then be computed as

$$BetaCV = \frac{N_{out} \cdot W_{in}}{N_{in} \cdot W_{out}} = \frac{7379 \times 3020.57}{3796 \times 24613.37} = 0.239$$

For the C-index, we first compute the sum of the N_{in} smallest and largest pair-wise distances, given as

$$W_{\min}(N_{in}) = 2535.96 \qquad W_{\max}(N_{in}) = 16889.57$$

Thus, C-index is given as

$$Cindex = \frac{W_{in} - W_{\min}(N_{in})}{W_{\max}(N_{in}) - W_{\min}(N_{in})} = \frac{3020.57 - 2535.96}{16889.57 - 2535.96} = \frac{484.61}{14535.61} = 0.0338$$

For the normalized cut and modularity measures, we compute $W(C_i, \overline{C_i})$, $W(C_i, V) = \sum_{j=1}^{k} W(C_i, C_j)$ and $W(V, V) = \sum_{i=1}^{k} W(C_i, V)$, using the intercluster weight matrix [Eq. (17.29)]:

$$W(C_1, \overline{C_1}) = 10402.30 + 4418.62 = 14820.91$$
$$W(C_2, \overline{C_2}) = 10402.30 + 9792.45 = 20194.75$$
$$W(C_3, \overline{C_3}) = 4418.62 + 9792.45 = 14211.07$$
$$W(C_1, V) = 3265.69 + W(C_1, \overline{C_1}) = 18086.61$$
$$W(C_2, V) = 1523.10 + W(C_2, \overline{C_2}) = 21717.85$$
$$W(C_3, V) = 1252.36 + W(C_3, \overline{C_3}) = 15463.43$$
$$W(V, V) = W(C_1, V) + W(C_2, V) + W(C_3, V) = 55267.89$$

The normalized cut and modularity values are given as

$$NC = \frac{14820.91}{18086.61} + \frac{20194.75}{21717.85} + \frac{14211.07}{15463.43} = 0.819 + 0.93 + 0.919 = 2.67$$

$$\begin{aligned} Q &= \left(\frac{3265.69}{55267.89} - \left(\frac{18086.61}{55267.89}\right)^2\right) + \left(\frac{1523.10}{55267.89} - \left(\frac{21717.85}{55267.89}\right)^2\right) \\ &\quad + \left(\frac{1252.36}{55267.89} - \left(\frac{15463.43}{55267.89}\right)^2\right) \\ &= -0.048 - 0.1269 - 0.0556 = -0.2305 \end{aligned}$$

The Dunn index can be computed from the minimum and maximum distances between pairs of points from two clusters C_i and C_j, computed as follows:

$$\left(\begin{array}{c|ccc} W^{\min} & C_1 & C_2 & C_3 \\ \hline C_1 & 0 & 1.62 & 0.198 \\ C_2 & 1.62 & 0 & 3.49 \\ C_3 & 0.198 & 3.49 & 0 \end{array}\right) \qquad \left(\begin{array}{c|ccc} W^{\max} & C_1 & C_2 & C_3 \\ \hline C_1 & 2.50 & 4.85 & 4.81 \\ C_2 & 4.85 & 2.33 & 7.06 \\ C_3 & 4.81 & 7.06 & 2.55 \end{array}\right)$$

The Dunn index value for the clustering is given as

$$Dunn = \frac{W_{out}^{\min}}{W_{in}^{\max}} = \frac{0.198}{2.55} = 0.078$$

To compute the Davies–Bouldin index, we compute the cluster mean and dispersion values:

$$\mu_1 = \begin{pmatrix} -0.664 \\ -0.33 \end{pmatrix} \qquad \mu_2 = \begin{pmatrix} 2.64 \\ 0.19 \end{pmatrix} \qquad \mu_3 = \begin{pmatrix} -2.35 \\ 0.27 \end{pmatrix}$$

$$\sigma_{\mu_1} = 0.723 \qquad \sigma_{\mu_2} = 0.512 \qquad \sigma_{\mu_3} = 0.695$$

and the DB_{ij} values for pairs of clusters:

$$\left(\begin{array}{c|ccc} DB_{ij} & C_1 & C_2 & C_3 \\ \hline C_1 & - & 0.369 & 0.794 \\ C_2 & 0.369 & - & 0.242 \\ C_3 & 0.794 & 0.242 & - \end{array}\right)$$

For example, $DB_{12} = \frac{\sigma_{\mu_1} + \sigma_{\mu_2}}{\delta(\mu_1, \mu_2)} = \frac{1.235}{3.346} = 0.369$. Finally, the DB index is given as

$$DB = \frac{1}{3}(0.794 + 0.369 + 0.794) = 0.652$$

The silhouette coefficient [Eq. (17.26)] for a chosen point, say $\mathbf{x}_1$, is given as

$$s_1 = \frac{1.902 - 0.701}{\max\{1.902, 0.701\}} = \frac{1.201}{1.902} = 0.632$$

The average value across all points is $SC = 0.598$

The Hubert statistic can be computed by taking the dot product over the upper triangular elements of the proximity matrix $\mathbf{W}$ [Eq. (17.22)] and the $n \times n$ matrix of distances among cluster means $\mathbf{Y}$ [Eq. (17.28)], and then dividing by the number of distinct point pairs N:

$$\Gamma = \frac{\mathbf{w}^T\mathbf{y}}{N} = \frac{91545.85}{11175} = 8.19$$

where $\mathbf{w}, \mathbf{y} \in \mathbb{R}^N$ are vectors comprising the upper triangular elements of $\mathbf{W}$ and $\mathbf{Y}$. The normalized Hubert statistic can be obtained as the correlation between $\mathbf{w}$ and $\mathbf{y}$ [Eq. (17.15)]:

$$\Gamma_n = \frac{\mathbf{z}_w^T\mathbf{z}_y}{\|\mathbf{x}_w\| \cdot \|\mathbf{z}_y\|} = 0.918$$

where $\mathbf{z}_w, \mathbf{z}_y$ are the centered vectors corresponding to $\mathbf{w}$ and $\mathbf{y}$, respectively.

The following table summarizes the various internal measure values for the good and bad clusterings shown in Figure 17.1 and Figure 17.2.

	Lower better				Higher better				
	BetaCV	*Cindex*	*Q*	*DB*	*NC*	*Dunn*	*SC*	Γ	Γ_n
(a) Good	0.24	0.034	−0.23	0.65	2.67	0.08	0.60	8.19	0.92
(b) Bad	0.33	0.08	−0.20	1.11	2.56	0.03	0.55	7.32	0.83

Despite the fact that these internal measures do not have access to the ground-truth partitioning, we can observe that the good clustering has higher values for normalized cut, Dunn, silhouette coefficient, and the Hubert statistics, and lower values for BetaCV, C-index, modularity, and Davies–Bouldin measures. These measures are thus capable of discerning good versus bad clusterings of the data.

17.3 RELATIVE MEASURES

Relative measures are used to compare different clusterings obtained by varying different parameters for the same algorithm, for example, to choose the number of clusters k.

Silhouette Coefficient

The silhouette coefficient [Eq. (17.26)] for each point s_j, and the average SC value [Eq. (17.27)], can be used to estimate the number of clusters in the data. The approach consists of plotting the s_j values in descending order for each cluster, and to note the overall SC value for a particular value of k, as well as clusterwise SC values:

$$SC_i = \frac{1}{n_i} \sum_{\mathbf{x}_j \in C_i} s_j$$

We can then pick the value k that yields the best clustering, with many points having high s_j values within each cluster, as well as high values for SC and SC_i $(1 \le i \le k)$.

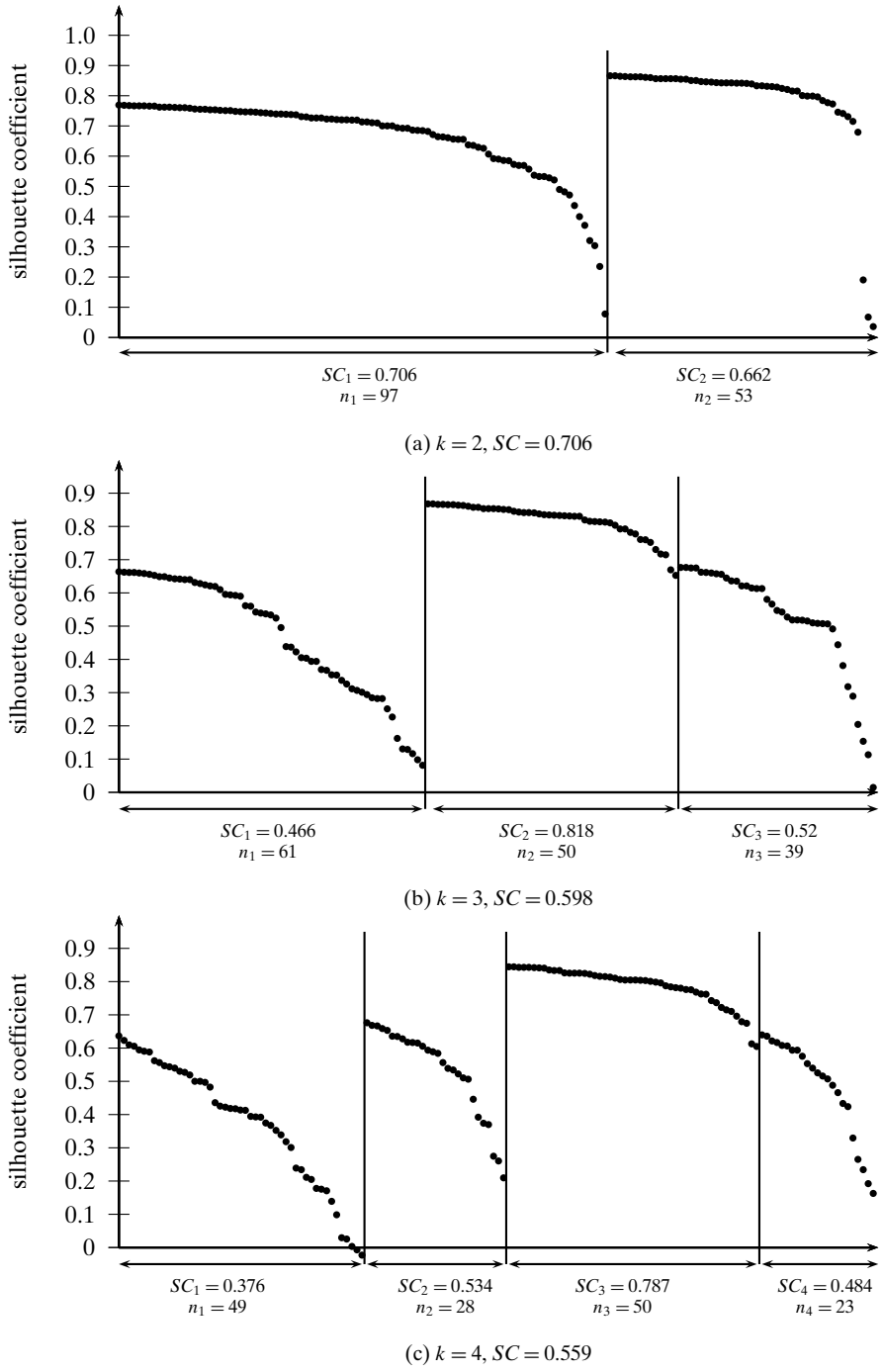


Figure 17.3. Iris K-means: silhouette coefficient plot.

Example 17.7. Figure 17.3 shows the silhouette coefficient plot for the best clustering results for the K-means algorithm on the Iris principal components dataset for three different values of k, namely $k = 2, 3, 4$. The silhouette coefficient values s_i for points

within each cluster are plotted in decreasing order. The overall average (SC) and clusterwise averages (SC_i, for $1 \leq i \leq k$) are also shown, along with the cluster sizes.

Figure 17.3a shows that $k = 2$ has the highest average silhouette coefficient, $SC = 0.706$. It shows two well separated clusters. The points in cluster C_1 start out with high s_i values, which gradually drop as we get to border points. The second cluster C_2 is even better separated, since it has a higher silhouette coefficient and the pointwise scores are all high, except for the last three points, suggesting that almost all the points are well clustered.

The silhouette plot in Figure 17.3b, with $k = 3$, corresponds to the "good" clustering shown in Figure 17.1a. We can see that cluster C_1 from Figure 17.3a has been split into two clusters for $k = 3$, namely C_1 and C_3. Both of these have many bordering points, whereas C_2 is well separated with high silhouette coefficients across all points.

Finally, the silhouette plot for $k = 4$ is shown in Figure 17.3c. Here C_3 is the well separated cluster, corresponding to C_2 above, and the remaining clusters are essentially subclusters of C_1 for $k = 2$ (Figure 17.3a). Cluster C_1 also has two points with negative s_i values, indicating that they are probably misclustered.

Because $k = 2$ yields the highest silhouette coefficient, and the two clusters are essentially well separated, in the absence of prior knowledge, we would choose $k = 2$ as the best number of clusters for this dataset.

Calinski–Harabasz Index

Given the dataset $\mathbf{D} = \{\mathbf{x}_i\}_{i=1}^n$, the scatter matrix for $\mathbf{D}$ is given as

$$\mathbf{S} = n\boldsymbol{\Sigma} = \sum_{j=1}^{n} \left(\mathbf{x}_j - \boldsymbol{\mu}\right)\left(\mathbf{x}_j - \boldsymbol{\mu}\right)^T$$

where $\boldsymbol{\mu} = \frac{1}{n}\sum_{j=1}^{n} \mathbf{x}_j$ is the mean and $\boldsymbol{\Sigma}$ is the covariance matrix. The scatter matrix can be decomposed into two matrices $\mathbf{S} = \mathbf{S}_W + \mathbf{S}_B$, where $\mathbf{S}_W$ is the within-cluster scatter matrix and $\mathbf{S}_B$ is the between-cluster scatter matrix, given as

$$\mathbf{S}_W = \sum_{i=1}^{k} \sum_{\mathbf{x}_j \in C_i} \left(\mathbf{x}_j - \boldsymbol{\mu}_i\right)\left(\mathbf{x}_j - \boldsymbol{\mu}_i\right)^T$$

$$\mathbf{S}_B = \sum_{i=1}^{k} n_i \left(\boldsymbol{\mu}_i - \boldsymbol{\mu}\right)\left(\boldsymbol{\mu}_i - \boldsymbol{\mu}\right)^T$$

where $\boldsymbol{\mu}_i = \frac{1}{n_i}\sum_{\mathbf{x}_j \in C_i} \mathbf{x}_j$ is the mean for cluster C_i.

The Calinski–Harabasz (CH) variance ratio criterion for a given value of k is defined as follows:

$$CH(k) = \frac{tr(\mathbf{S}_B)/(k-1)}{tr(\mathbf{S}_W)/(n-k)} = \frac{n-k}{k-1} \cdot \frac{tr(\mathbf{S}_B)}{tr(\mathbf{S}_W)}$$

where $tr(\mathbf{S}_W)$ and $tr(\mathbf{S}_B)$ are the traces (the sum of the diagonal elements) of the within-cluster and between-cluster scatter matrices.

For a good value of k, we expect the within-cluster scatter to be smaller relative to the between-cluster scatter, which should result in a higher $CH(k)$ value. On the other

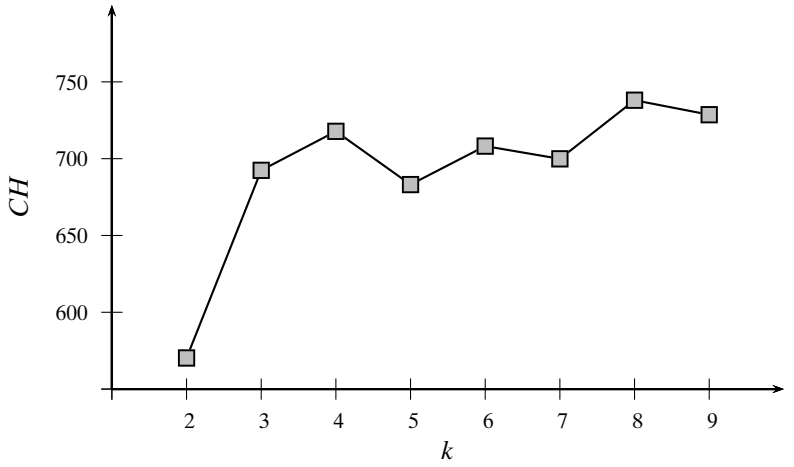


Figure 17.4. Calinski–Harabasz variance ratio criterion.

hand, we do not desire a very large value of k; thus the term $\frac{n-k}{k-1}$ penalizes larger values of k. We could choose a value of k that maximizes $CH(k)$. Alternatively, we can plot the CH values and look for a large increase in the value followed by little or no gain. For instance, we can choose the value $k > 3$ that minimizes the term

$$\Delta(k) = \Big(CH(k+1) - CH(k)\Big) - \Big(CH(k) - CH(k-1)\Big)$$

The intuition is that we want to find the value of k for which $CH(k)$ is much higher than $CH(k-1)$ and there is only a little improvement or a decrease in the $CH(k+1)$ value.

Example 17.8. Figure 17.4 shows the CH ratio for various values of k on the Iris principal components dataset, using the K-means algorithm, with the best results chosen from 200 runs.

For $k = 3$, the within-cluster and between-cluster scatter matrices are given as

$$\mathbf{S}_W = \begin{pmatrix} 39.14 & -13.62 \\ -13.62 & 24.73 \end{pmatrix} \qquad \mathbf{S}_B = \begin{pmatrix} 590.36 & 13.62 \\ 13.62 & 11.36 \end{pmatrix}$$

Thus, we have

$$CH(3) = \frac{(150-3)}{(3-1)} \cdot \frac{(590.36+11.36)}{(39.14+24.73)} = (147/2) \cdot \frac{601.72}{63.87} = 73.5 \cdot 9.42 = 692.4$$

The successive $CH(k)$ and $\Delta(k)$ values are as follows:

k	2	3	4	5	6	7	8	9
$CH(k)$	570.25	692.40	717.79	683.14	708.26	700.17	738.05	728.63
$\Delta(k)$	–	−96.78	−60.03	59.78	−33.22	45.97	−47.30	–

If we choose the first large peak before a decrease we would choose $k = 4$. However, $\Delta(k)$ suggests $k = 3$ as the best (lowest) value, representing the "knee-of-the-curve". One limitation of the $\Delta(k)$ criteria is that values less than $k = 3$ cannot be evaluated, since $\Delta(2)$ depends on $CH(1)$, which is not defined.

Gap Statistic

The gap statistic compares the sum of intracluster weights W_{in} [Eq. (17.23)] for different values of k with their expected values assuming no apparent clustering structure, which forms the null hypothesis.

Let $\mathcal{C}_k$ be the clustering obtained for a specified value of k, using a chosen clustering algorithm. Let $W_{in}^k(\mathbf{D})$ denote the sum of intracluster weights (over all clusters) for $\mathcal{C}_k$ on the input dataset $\mathbf{D}$. We would like to compute the probability of the observed W_{in}^k value under the null hypothesis that the points are randomly placed in the same data space as $\mathbf{D}$. Unfortunately, the sampling distribution of W_{in} is not known. Further, it depends on the number of clusters k, the number of points n, and other characteristics of $\mathbf{D}$.

To obtain an empirical distribution for W_{in}, we resort to Monte Carlo simulations of the sampling process. That is, we generate t random samples comprising n randomly distributed points within the same d-dimensional data space as the input dataset $\mathbf{D}$. That is, for each dimension of $\mathbf{D}$, say X_j, we compute its range $[\min(X_j), \max(X_j)]$ and generate values for the n points (for the jth dimension) uniformly at random within the given range. Let $\mathbf{R}_i \in \mathbb{R}^{n \times d}$, $1 \le i \le t$ denote the ith sample. Let $W_{in}^k(\mathbf{R}_i)$ denote the sum of intracluster weights for a given clustering of $\mathbf{R}_i$ into k clusters. From each sample dataset $\mathbf{R}_i$, we generate clusterings for different values of k using the same algorithm and record the intracluster values $W_{in}^k(\mathbf{R}_i)$. Let $\mu_W(k)$ and $\sigma_W(k)$ denote the mean and standard deviation of these intracluster weights for each value of k, given as

$$\mu_W(k) = \frac{1}{t}\sum_{i=1}^{t} \log W_{in}^k(\mathbf{R}_i)$$

$$\sigma_W(k) = \sqrt{\frac{1}{t}\sum_{i=1}^{t}\Big(\log W_{in}^k(\mathbf{R}_i) - \mu_W(k)\Big)^2}$$

where we use the logarithm of the W_{in} values, as they can be quite large.

The *gap statistic* for a given k is then defined as

$$gap(k) = \mu_W(k) - \log W_{in}^k(\mathbf{D})$$

It measures the deviation of the observed W_{in}^k value from its expected value under the null hypothesis. We can select the value of k that yields the largest gap statistic because that indicates a clustering structure far away from the uniform distribution of points. A more robust approach is to choose k as follows:

$$k^* = \arg\min_k \Big\{ gap(k) \ge gap(k+1) - \sigma_W(k+1) \Big\}$$

That is, we select the least value of k such that the gap statistic is within one standard deviation of the gap at $k+1$.

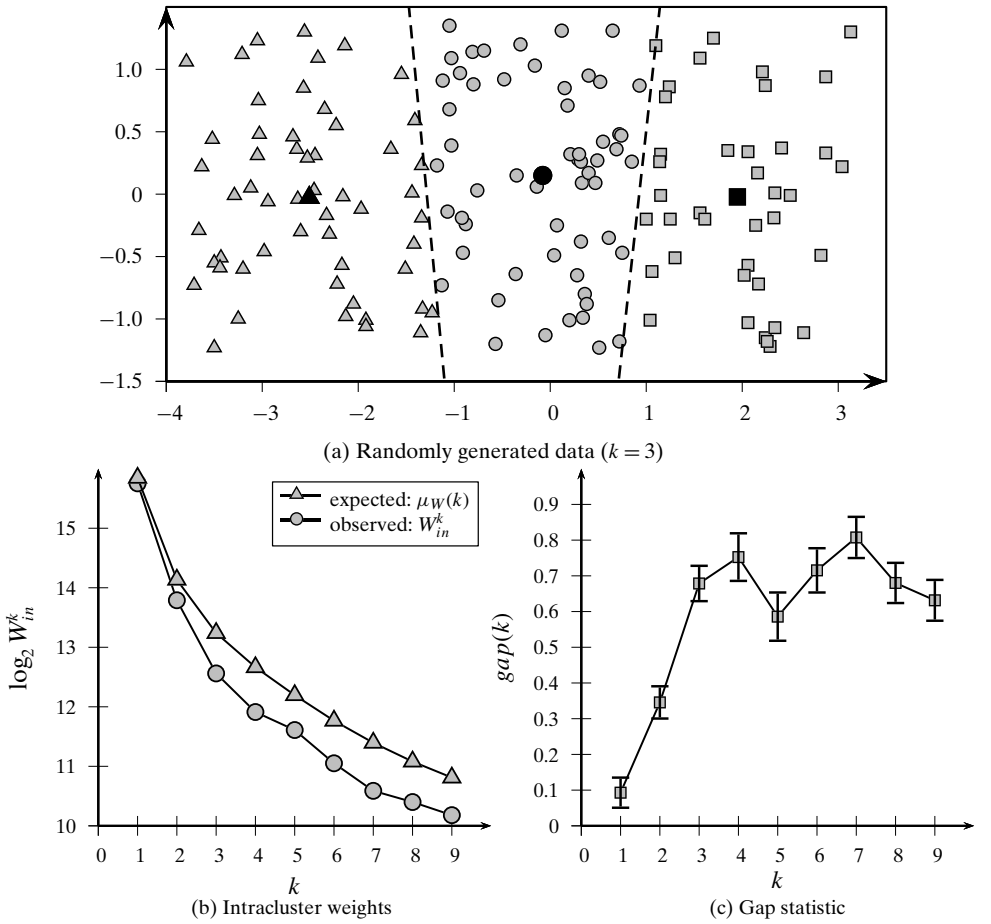


Figure 17.5. Gap statistic. (a) Randomly generated data. (b) Intracluster weights for different k. (c) Gap statistic as a function of k.

Example 17.9. To compute the gap statistic we have to generate t random samples of n points drawn from the same data space as the Iris principal components dataset. A random sample of $n = 150$ points is shown in Figure 17.5a, which does not have any apparent cluster structure. However, when we run K-means on this dataset it will output some clustering, an example of which is also shown, with $k = 3$. From this clustering, we can compute the $\log_2 W_{in}^k(\mathbf{R}_i)$ value; we use base 2 for all logarithms.

For Monte Carlo sampling, we generate $t = 200$ such random datasets, and compute the mean or expected intracluster weight $\mu_W(k)$ under the null hypothesis, for each value of k. Figure 17.5b shows the expected intracluster weights for different values of k. It also shows the observed value of $\log_2 W_{in}^k$ computed from the K-means clustering of the Iris principal components dataset. For the Iris dataset, and each of the uniform random samples, we run K-means 100 times and select the best possible clustering, from which the $W_{in}^k(\mathbf{R}_i)$ values are computed. We can see that the observed $W_{in}^k(\mathbf{D})$ values are smaller than the expected values $\mu_W(k)$.

Table 17.1. Gap statistic values as a function of k

k	$gap(k)$	$\sigma_W(k)$	$gap(k) - \sigma_W(k)$
1	0.093	0.0456	0.047
2	0.346	0.0486	0.297
3	0.679	0.0529	0.626
4	0.753	0.0701	0.682
5	0.586	0.0711	0.515
6	0.715	0.0654	0.650
7	0.808	0.0611	0.746
8	0.680	0.0597	0.620
9	0.632	0.0606	0.571

From these values, we then compute the gap statistic $gap(k)$ for different values of k, which are plotted in Figure 17.5c. Table 17.1 lists the gap statistic and standard deviation values. The optimal value for the number of clusters is $k = 4$ because

$$gap(4) = 0.753 > gap(5) - \sigma_W(5) = 0.515$$

However, if we had relaxed the gap test to be within two standard deviations, then the optimal value would have been $k = 3$ because

$$gap(3) = 0.679 > gap(4) - 2\sigma_W(4) = 0.753 - 2 \cdot 0.0701 = 0.613$$

Essentially, there is still some subjectivity in selecting the right number of clusters, but the gap statistic plot can help in this task.

17.3.1 Cluster Stability

The main idea behind cluster stability is that the clusterings obtained from several datasets sampled from the same underlying distribution as $\mathbf{D}$ should be similar or "stable." The cluster stability approach can be used to find good parameter values for a given clustering algorithm; we will focus on the task of finding a good value for k, the correct number of clusters.

The joint probability distribution for $\mathbf{D}$ is typically unknown. Therefore, to sample a dataset from the same distribution we can try a variety of methods, including random perturbations, subsampling, or bootstrap resampling. Let us consider the bootstrapping approach; we generate t samples of size n by sampling from $\mathbf{D}$ with replacement, which allows the same point to be chosen possibly multiple times, and thus each sample $\mathbf{D}_i$ will be different. Next, for each sample $\mathbf{D}_i$ we run the same clustering algorithm with different cluster values k ranging from 2 to $k^{\max}$.

Let $\mathcal{C}_k(\mathbf{D}_i)$ denote the clustering obtained from sample $\mathbf{D}_i$, for a given value of k. Next, the method compares the distance between all pairs of clusterings $\mathcal{C}_k(\mathbf{D}_i)$ and $\mathcal{C}_k(\mathbf{D}_j)$ via some distance function. Several of the external cluster evaluation measures can be used as distance measures, by setting, for example, $\mathcal{C} = \mathcal{C}_k(\mathbf{D}_i)$ and $\mathcal{T} = \mathcal{C}_k(\mathbf{D}_j)$, or vice versa. From these values we compute the expected pairwise distance for each value of k. Finally, the value k^* that exhibits the least deviation between the clusterings

ALGORITHM 17.1. Clustering Stability Algorithm for Choosing k

```
CLUSTERINGSTABILITY (A, t, k^max, D):
n ← |D|
// Generate t samples
for i = 1, 2, ..., t do
    D_i ← sample n points from D with replacement
// Generate clusterings for different values of k
for i = 1, 2, ..., t do
    for k = 2, 3, ..., k^max do
        C_k(D_i) ← cluster D_i into k clusters using algorithm A
// Compute mean difference between clusterings for each k
foreach pair D_i, D_j with j > i do
    D_ij ← D_i ∩ D_j // create common dataset using Eq. (17.30)
    for k = 2, 3, ..., k^max do
        d_ij(k) ← d(C_k(D_i), C_k(D_j), D_ij) // distance between
            clusterings
for k = 2, 3, ..., k^max do
    μ_d(k) ← 2/(t(t−1)) Σ_{i=1}^{t} Σ_{j>i} d_ij(k) // expected pairwise distance
// Choose best k
k* ← arg min_k {μ_d(k)}
```

obtained from the resampled datasets is the best choice for k because it exhibits the most stability.

There is, however, one complication when evaluating the distance between a pair of clusterings $\mathcal{C}_k(\mathbf{D}_i)$ and $\mathcal{C}_k(\mathbf{D}_j)$, namely that the underlying datasets $\mathbf{D}_i$ and $\mathbf{D}_j$ are different. That is, the set of points being clustered is different because each sample $\mathbf{D}_i$ is different. Before computing the distance between the two clusterings, we have to restrict the clusterings only to the points common to both $\mathbf{D}_i$ and $\mathbf{D}_j$, denoted as $\mathbf{D}_{ij}$. Because sampling with replacement allows multiple instances of the same point, we also have to account for this when creating $\mathbf{D}_{ij}$. For each point $\mathbf{x}_a$ in the input dataset $\mathbf{D}$, let m_i^a and m_j^a denote the number of occurrences of $\mathbf{x}_a$ in $\mathbf{D}_i$ and $\mathbf{D}_j$, respectively. Define

$$\mathbf{D}_{ij} = \mathbf{D}_i \cap \mathbf{D}_j = \left\{ m^a \text{ instances of } \mathbf{x}_a \mid \mathbf{x}_a \in \mathbf{D}, m^a = \min\{m_i^a, m_j^a\} \right\} \tag{17.30}$$

That is, the common dataset $\mathbf{D}_{ij}$ is created by selecting the minimum number of instances of the point $\mathbf{x}_a$ in $\mathbf{D}_i$ or $\mathbf{D}_j$.

Algorithm 17.1 shows the pseudo-code for the clustering stability method for choosing the best k value. It takes as input the clustering algorithm A, the number of samples t, the maximum number of clusters $k^{\max}$, and the input dataset $\mathbf{D}$.

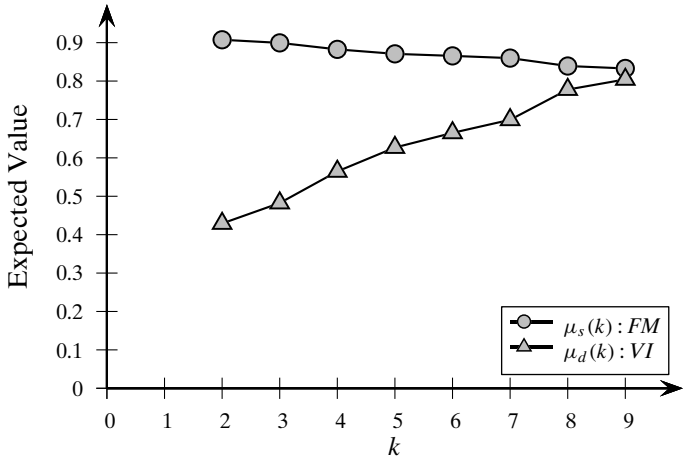


Figure 17.6. Clustering stability: Iris dataset.

It first generates the t bootstrap samples and clusters them using algorithm A. Next, it computes the distance between the clusterings for each pair of datasets $\mathbf{D}_i$ and $\mathbf{D}_j$, for each value of k. Finally, the method computes the expected pairwise distance $\mu_d(k)$ in line 12. We assume that the clustering distance function d is symmetric. If d is not symmetric, then the expected difference should be computed over all ordered pairs, that is, $\mu_d(k) = \frac{1}{t(t-1)} \sum_{i=1}^{r} \sum_{j \neq i} d_{ij}(k)$.

Instead of a distance function d, we can also evaluate clustering stability via a similarity measure, in which case, after computing the average similarity between pairs of clusterings for a given k, we can choose the best value k^* as the one that maximizes the expected similarity $\mu_s(k)$. In general, those external measures that yield lower values for better agreement between $\mathcal{C}_k(\mathbf{D}_i)$ and $\mathcal{C}_k(\mathbf{D}_j)$ can be used as distance functions, whereas those that yield higher values for better agreement can be used as similarity functions. Examples of distance functions include normalized mutual information, variation of information, and conditional entropy (which is asymmetric). Examples of similarity functions include Jaccard, Fowlkes–Mallows, Hubert Γ statistic, and so on.

Example 17.10. We study the clustering stability for the Iris principal components dataset, with $n = 150$, using the K-means algorithm. We use $t = 500$ bootstrap samples. For each dataset $\mathbf{D}_i$, and each value of k, we run K-means with 100 initial starting configurations, and select the best clustering.

For the distance function, we used the variation of information [Eq. (17.5)] between each pair of clusterings. We also used the Fowlkes–Mallows measure [Eq. (17.13)] as an example of a similarity measure. The expected values of the pairwise distance $\mu_d(k)$ for the VI measure, and the pairwise similarity $\mu_s(k)$ for the FM measure are plotted in Figure 17.6. Both the measures indicate that $k = 2$ is the best value, as for the VI measure this leads to the least expected distance between pairs of clusterings, and for the FM measure this choice leads to the most expected similarity between clusterings.

17.3.2 Clustering Tendency

Clustering tendency or clusterability aims to determine whether the dataset $\mathbf{D}$ has any meaningful groups to begin with. This is usually a hard task given the different definitions of what it means to be a cluster, for example, partitional, hierarchical, density-based, graph-based and so on. Even if we fix the cluster type, it is still a hard task to define the appropriate null model (e.g., the one without any clustering structure) for a given dataset $\mathbf{D}$. Furthermore, if we do determine that the data is clusterable, then we are still faced with the question of how many clusters there are. Nevertheless, it is still worthwhile to assess the clusterability of a dataset; we look at some approaches to answer the question whether the data is clusterable or not.

Spatial Histogram

One simple approach is to contrast the d-dimensional spatial histogram of the input dataset $\mathbf{D}$ with the histogram from samples generated randomly in the same data space. Let $X_1, X_2, \ldots, X_d$ denote the d dimensions. Given b, the number of bins for each dimension, we divide each dimension X_j into b equi-width bins, and simply count how many points lie in each of the b^d d-dimensional cells. From this spatial histogram, we can obtain the empirical joint probability mass function (EPMF) for the dataset $\mathbf{D}$, which is an approximation of the unknown joint probability density function. The EPMF is given as

$$f(\mathbf{i}) = P(\mathbf{x}_j \in \text{cell } \mathbf{i}) = \frac{\big|\{\mathbf{x}_j \in \text{cell } \mathbf{i}\}\big|}{n}$$

where $\mathbf{i} = (i_1, i_2, \ldots, i_d)$ denotes a cell index, with i_j denoting the bin index along dimension X_j.

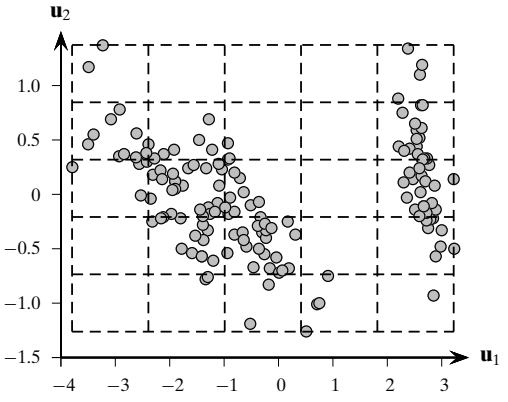
Next, we generate t random samples, each comprising n points within the same d-dimensional space as the input dataset $\mathbf{D}$. That is, for each dimension X_j, we compute its range $[\min(X_j), \max(X_j)]$, and generate values uniformly at random within the given range. Let $\mathbf{R}_j$ denote the jth such random sample. We can then compute the corresponding EPMF $g_j(\mathbf{i})$ for each $\mathbf{R}_j$, $1 \le j \le t$.

Finally, we can compute how much the distribution f differs from g_j (for $j = 1, \ldots, t$), using the Kullback–Leibler (KL) divergence from f to g_j, defined as

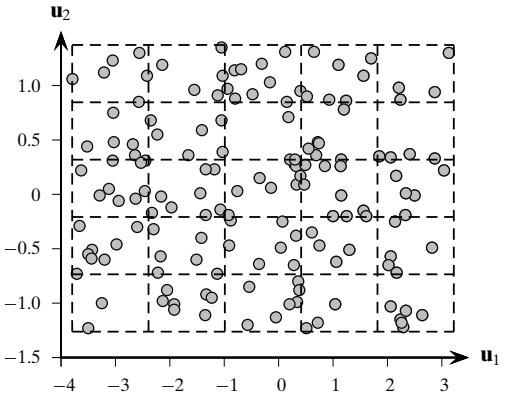
$$KL(f|g_j) = \sum_{\mathbf{i}} f(\mathbf{i}) \log\left(\frac{f(\mathbf{i})}{g_j(\mathbf{i})}\right) \tag{17.31}$$

The KL divergence is zero only when f and g_j are the same distributions. Using these divergence values, we can compute how much the dataset $\mathbf{D}$ differs from a random dataset.

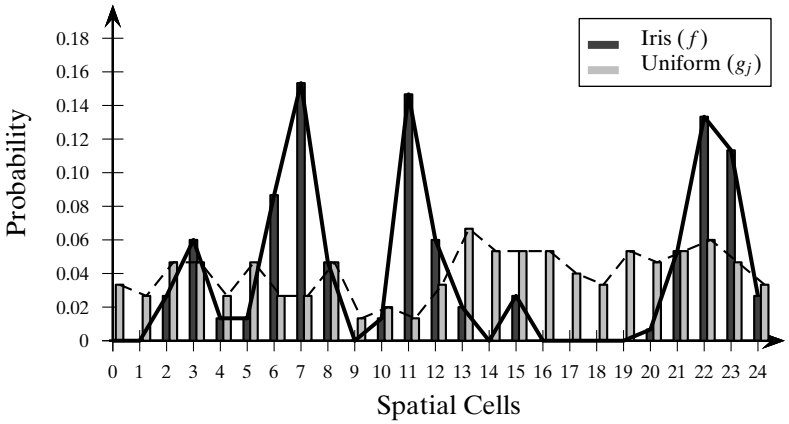
The main limitation of this approach is that as dimensionality increases, the number of cells (b^d) increases exponentially, and with a fixed sample size n, most of the cells will be empty, or will have only one point, making it hard to estimate the divergence. The method is also sensitive to the choice of parameter b. Instead of histograms, and the corresponding EPMF, we can also use density estimation methods (see Section 15.2) to determine the joint probability density function (PDF) for the



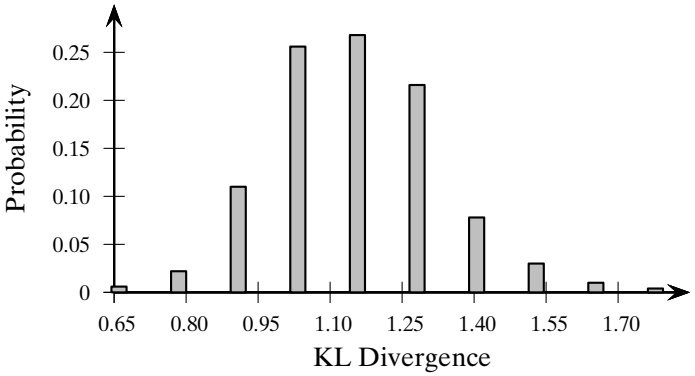
(a) Iris: spatial cells



(b) Uniform: spatial cells



(c) Empirical probability mass function



(d) KL-divergence distribution

Figure 17.7. Iris dataset: spatial histogram.

dataset **D**, and see how it differs from the PDF for the random datasets. However, the curse of dimensionality also causes problems for density estimation.

Example 17.11. Figure 17.7c shows the empirical joint probability mass function for the Iris principal components dataset that has $n = 150$ points in $d = 2$ dimensions. It also shows the EPMF for one of the datasets generated uniformly at random in the same data space. Both EPMFs were computed using $b = 5$ bins in each dimension, for a total of 25 spatial cells. The spatial grids/cells for the Iris dataset **D**, and the random sample **R**, are shown in Figures 17.7a and 17.7b, respectively. The cells are numbered starting from 0, from bottom to top, and then left to right. Thus, the bottom left cell is 0, top left is 4, bottom right is 19, and top right is 24. These indices are used along the x-axis in the EPMF plot in Figure 17.7c.

We generated $t = 500$ random samples from the null distribution, and computed the KL divergence from f to g_j for each $1 \leq j \leq t$ (using logarithm with base 2). The distribution of the KL values is plotted in Figure 17.7d. The mean KL value was $\mu_{KL} = 1.17$, with a standard deviation of $\sigma_{KL} = 0.18$, indicating that the Iris data is indeed far from the randomly generated data, and thus is clusterable.

Distance Distribution

Instead of trying to estimate the density, another approach to determine clusterability is to compare the pairwise point distances from **D**, with those from the randomly generated samples $\mathbf{R}_i$ from the null distribution. That is, we create the EPMF from the proximity matrix **W** for **D** [Eq. (17.22)] by binning the distances into b bins:

$$f(i) = P(w_{pq} \in \text{ bin } i \mid \mathbf{x}_p, \mathbf{x}_q \in \mathbf{D}, p < q) = \frac{\left|\{w_{pq} \in \text{ bin } i\}\right|}{n(n-1)/2}$$

Likewise, for each of the samples $\mathbf{R}_j$, we can determine the EPMF for the pairwise distances, denoted g_j. Finally, we can compute the KL divergences between f and g_j using Eq. (17.31). The expected divergence indicates the extent to which **D** differs from the null (random) distribution.

Example 17.12. Figure 17.8a shows the distance distribution for the Iris principal components dataset **D** and the random sample $\mathbf{R}_j$ from Figure 17.7b. The distance distribution is obtained by binning the edge weights between all pairs of points using $b = 25$ bins.

We then compute the KL divergence from **D** to each $\mathbf{R}_j$, over $t = 500$ samples. The distribution of the KL divergences (using logarithm with base 2) is shown in Figure 17.8b. The mean divergence is $\mu_{KL} = 0.18$, with standard deviation $\sigma_{KL} = 0.017$. Even though the Iris dataset has a good clustering tendency, the KL divergence is not very large. We conclude that, at least for the Iris dataset, the distance distribution is not as discriminative as the spatial histogram approach for clusterability analysis.

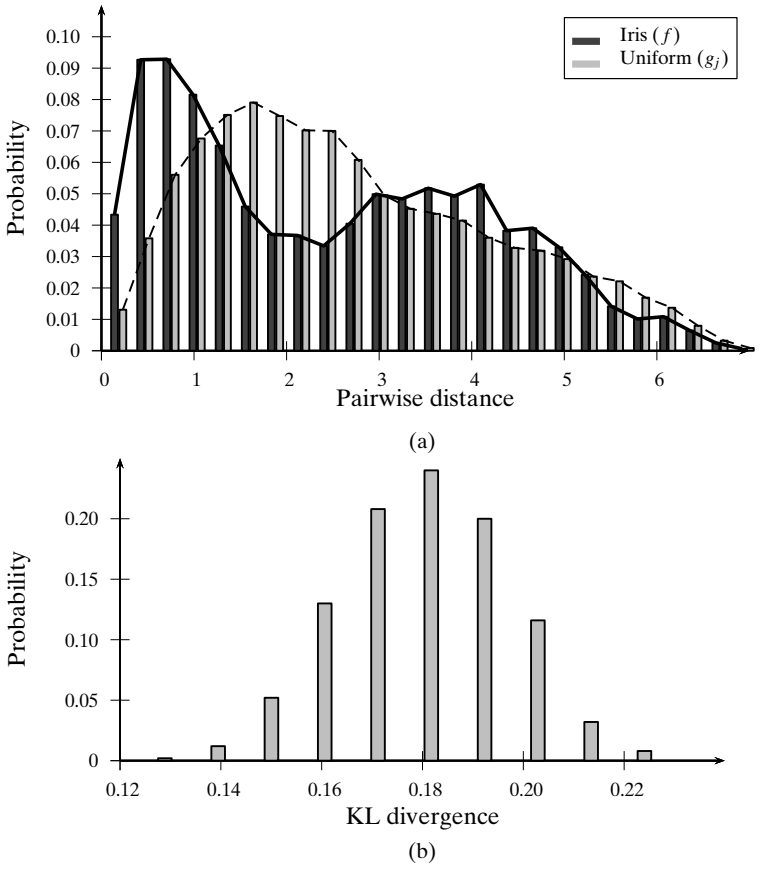


Figure 17.8. Iris dataset: distance distribution.

Hopkins Statistic

The Hopkins statistic is a sparse sampling test for spatial randomness. Given a dataset $\mathbf{D}$ comprising n points, we generate t random subsamples $\mathbf{R}_i$ of m points each, where $m \ll n$. These samples are drawn from the same data space as $\mathbf{D}$, generated uniformly at random along each dimension. Further, we also generate t subsamples of m points directly from $\mathbf{D}$, using sampling without replacement. Let $\mathbf{D}_i$ denote the ith direct subsample. Next, we compute the minimum distance between each point $\mathbf{x}_j \in \mathbf{D}_i$ and points in $\mathbf{D}$

$$\delta_{\min}(\mathbf{x}_j) = \min_{\mathbf{x}_i \in \mathbf{D}, \mathbf{x}_i \neq \mathbf{x}_j} \left\{ \delta(\mathbf{x}_j, \mathbf{x}_i) \right\}$$

Likewise, we compute the minimum distance $\delta_{\min}(\mathbf{y}_j)$ between a point $\mathbf{y}_j \in \mathbf{R}_i$ and points in $\mathbf{D}$.

The Hopkins statistic (in d dimensions) for the ith pair of samples $\mathbf{R}_i$ and $\mathbf{D}_i$ is then defined as

$$HS_i = \frac{\sum_{\mathbf{y}_j \in \mathbf{R}_i} \left(\delta_{\min}(\mathbf{y}_j)\right)^d}{\sum_{\mathbf{y}_j \in \mathbf{R}_i} \left(\delta_{\min}(\mathbf{y}_j)\right)^d + \sum_{\mathbf{x}_j \in \mathbf{D}_i} \left(\delta_{\min}(\mathbf{x}_j)\right)^d}$$

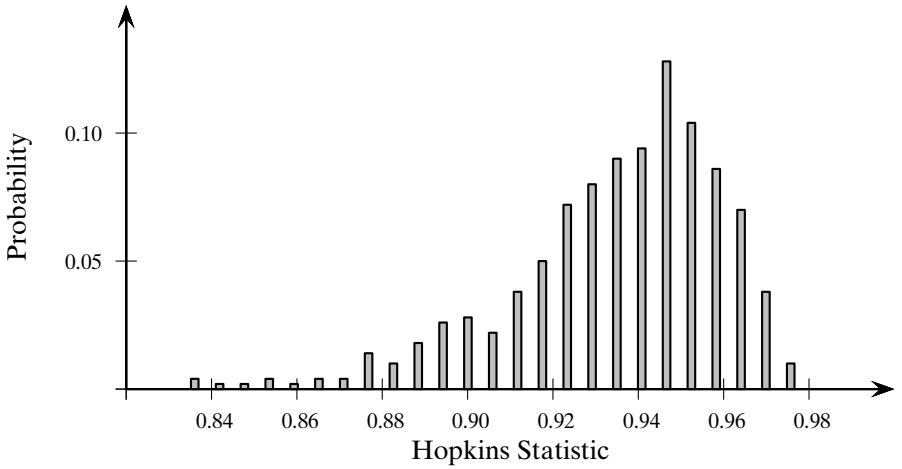


Figure 17.9. Iris dataset: Hopkins statistic distribution.

This statistic compares the nearest-neighbor distribution of randomly generated points to the same distribution for random subsets of points from $\mathbf{D}$. If the data is well clustered we expect $\delta_{\min}(\mathbf{x}_j)$ values to be smaller compared to the $\delta_{\min}(\mathbf{y}_j)$ values, and in this case HS_i tends to 1. If both nearest-neighbor distances are similar, then HS_i takes on values close to 0.5, which indicates that the data is essentially random, and there is no apparent clustering. Finally, if $\delta_{\min}(\mathbf{x}_j)$ values are larger compared to $\delta_{\min}(\mathbf{y}_j)$ values, then HS_i tends to 0, and it indicates point repulsion, with no clustering. From the t different values of HS_i we may then compute the mean and variance of the statistic to determine whether $\mathbf{D}$ is clusterable or not.

Example 17.13. Figure 17.9 plots the distribution of the Hopkins statistic values over $t = 500$ pairs of samples: $\mathbf{R}_j$ generated uniformly at random, and $\mathbf{D}_j$ subsampled from the input dataset $\mathbf{D}$. The subsample size was set as $m = 30$, using 20% of the points in $\mathbf{D}$, that is, the Iris principal components dataset, which has $n = 150$ points in $d = 2$ dimensions. The mean of the Hopkins statistic is $\mu_{HS} = 0.935$, with a standard deviation of $\sigma_{HS} = 0.025$. Given the high value of the statistic, we conclude that the Iris dataset has a good clustering tendency.

17.4 FURTHER READING

For an excellent introduction to clustering validation see Jain and Dubes (1988); the book describes many of the external, internal, and relative measures discussed in this chapter, including clustering tendency. Other good reviews appear in Halkidi, Batistakis, and Vazirgiannis (2001) and Theodoridis and Koutroumbas (2008). For recent work on formal properties for comparing clusterings via external measures see Amigó et al. (2009) and Meilă (2007). For the silhouette plot see Rousseeuw (1987), and for gap statistic see Tibshirani, Walther, and Hastie (2001). For an overview of cluster stability methods see Luxburg (2009). A recent review of clusterability appears

in Ackerman and Ben-David (2009). Overall reviews of clustering methods appear in Xu and Wunsch (2005) and Jain, Murty, and Flynn (1999). See Kriegel, Kröger, and Zimek (2009) for a review of subspace clustering methods.

Ackerman, M. and Ben-David, S. (2009). "Clusterability: A theoretical study." *In Proceedings of 12th International Conference on Artificial Intelligence and Statistics.*

Amigó, E., Gonzalo, J., Artiles, J., and Verdejo, F. (2009). "A comparison of extrinsic clustering evaluation metrics based on formal constraints." *Information Retrieval*, 12 (4): 461–486.

Halkidi, M., Batistakis, Y., and Vazirgiannis, M. (2001). "On clustering validation techniques." *Journal of Intelligent Information Systems*, 17 (2–3): 107–145.

Jain, A. K. and Dubes, R. C. (1988). *Algorithms for Clustering Data*. Upper Saddle River, NJ: Prentice-Hall.

Jain, A. K., Murty, M. N., and Flynn, P. J. (1999). "Data clustering: A review." *ACM Computing Surveys*, 31 (3): 264–323.

Kriegel, H.-P., Kröger, P., and Zimek, A. (2009). "Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering." *ACM Transactions on Knowledge Discovery from Data*, 3 (1): 1.

Luxburg, U. von (2009). "Clustering stability: An overview." *Foundations and Trends in Machine Learning*, 2 (3): 235–274.

Meilă, M. (2007). "Comparing clusterings – an information based distance." *Journal of Multivariate Analysis*, 98 (5): 873–895.

Rousseeuw, P. J. (1987). "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis." *Journal of Computational and Applied Mathematics*, 20: 53–65.

Theodoridis, S. and Koutroumbas, K. (2008). *Pattern Recognition,* 4th ed. San Diego: Academic Press.

Tibshirani, R., Walther, G., and Hastie, T. (2001). "Estimating the number of clusters in a dataset via the gap statistic." *Journal of the Royal Statistical Society B*, 63: 411–423.

Xu, R. and Wunsch, D. (2005). "Survey of clustering algorithms." *IEEE Transactions on Neural Networks*, 16 (3): 645–678.

17.5 EXERCISES

Q1. Prove that the maximum value of the entropy measure in Eq. (17.2) is $\log k$.

Q2. Show that if $\mathcal{C}$ and $\mathcal{T}$ are independent of each other then $H(\mathcal{T}|\mathcal{C}) = H(\mathcal{T})$, and further that $H(\mathcal{C}, \mathcal{T}) = H(\mathcal{C}) + H(\mathcal{T})$.

Q3. Show that $H(\mathcal{T}|\mathcal{C}) = 0$ if and only if $\mathcal{T}$ is completely determined by $\mathcal{C}$.

Q4. Show that $I(\mathcal{C}, \mathcal{T}) = H(\mathcal{C}) + H(\mathcal{T}) - H(\mathcal{T}, \mathcal{C})$.

Q5. Show that the variation of information is 0 only when $\mathcal{C}$ and $\mathcal{T}$ are identical.

Q6. Prove that the maximum value of the normalized discretized Hubert statistic in Eq. (17.21) is obtained when $FN = FP = 0$, and the minimum value is obtained when $TP = TN = 0$.

Q7. Show that the Fowlkes–Mallows measure can be considered as the correlation between the pairwise indicator matrices for $\mathcal{C}$ and $\mathcal{T}$, respectively. Define $\mathbf{C}(i, j) = 1$ if $\mathbf{x}_i$ and $\mathbf{x}_j$ (with $i \neq j$) are in the same cluster, and 0 otherwise. Define $\mathbf{T}$ similarly for the ground-truth partitions. Define $\langle \mathbf{C}, \mathbf{T} \rangle = \sum_{i,j=1}^{n} \mathbf{C}_{ij}\mathbf{T}_{ij}$. Show that $FM = \frac{\langle \mathbf{C}, \mathbf{T} \rangle}{\sqrt{\langle \mathbf{T}, \mathbf{T} \rangle \langle \mathbf{C}, \mathbf{C} \rangle}}$

Q8. Show that the silhouette coefficient of a point lies in the interval $[-1, +1]$.

Q9. Show that the scatter matrix can be decomposed as $\mathbf{S} = \mathbf{S}_W + \mathbf{S}_B$, where $\mathbf{S}_W$ and $\mathbf{S}_B$ are the within-cluster and between-cluster scatter matrices.

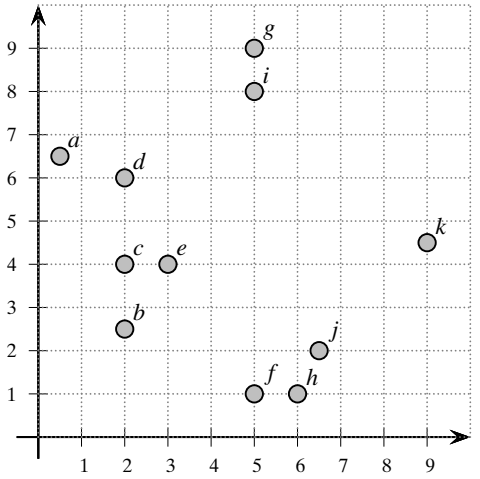


Figure 17.10. Data for Q10 .

Q10. Consider the dataset in Figure 17.10. Compute the silhouette coefficient for the point labeled c.

Q11. Describe how one may apply the gap statistic methodology for determining the parameters of density-based clustering algorithms, such as DBSCAN and DENCLUE (see Chapter 15).

PART FOUR CLASSIFICATION

CHAPTER 18 Probabilistic Classification

Classification refers to the task of predicting a class label for a given unlabeled point. In this chapter we consider three examples of the probabilistic classification approach. The (full) Bayes classifier uses the Bayes theorem to predict the class as the one that maximizes the posterior probability. The main task is to estimate the joint probability density function for each class, which is modeled via a multivariate normal distribution. The naive Bayes classifier assumes that attributes are independent, but it is still surprisingly powerful for many applications. We also describe the nearest neighbors classifier, which uses a non-parametric approach to estimate the density.

18.1 BAYES CLASSIFIER

Let the training dataset $\mathbf{D}$ consist of n points $\mathbf{x}_i$ in a d-dimensional space, and let y_i denote the class for each point, with $y_i \in \{c_1, c_2, \ldots, c_k\}$. The Bayes classifier directly uses the Bayes theorem to predict the class for a new test instance, $\mathbf{x}$. It estimates the posterior probability $P(c_i|\mathbf{x})$ for each class c_i, and chooses the class that has the largest probability. The predicted class for $\mathbf{x}$ is given as

$$\hat{y} = \arg\max_{c_i}\{P(c_i|\mathbf{x})\} \tag{18.1}$$

The Bayes theorem allows us to invert the posterior probability in terms of the likelihood and prior probability, as follows:

$$P(c_i|\mathbf{x}) = \frac{P(\mathbf{x}|c_i) \cdot P(c_i)}{P(\mathbf{x})}$$

where $P(\mathbf{x}|c_i)$ is the *likelihood*, defined as the probability of observing $\mathbf{x}$ assuming that the true class is c_i, $P(c_i)$ is the *prior probability* of class c_i, and $P(\mathbf{x})$ is the probability of observing $\mathbf{x}$ from any of the k classes, given as

$$P(\mathbf{x}) = \sum_{j=1}^{k} P(\mathbf{x}|c_j) \cdot P(c_j)$$

Because $P(\mathbf{x})$ is fixed for a given point, Bayes rule [Eq. (18.1)] can be rewritten as

$$\begin{aligned}\hat{y} &= \arg\max_{c_i}\{P(c_i|\mathbf{x})\} \\ &= \arg\max_{c_i}\left\{\frac{P(\mathbf{x}|c_i)P(c_i)}{P(\mathbf{x})}\right\} = \arg\max_{c_i}\big\{P(\mathbf{x}|c_i)P(c_i)\big\}\end{aligned} \tag{18.2}$$

In other words, the predicted class essentially depends on the likelihood of that class taking its prior probability into account.

18.1.1 Estimating the Prior Probability

To classify points, we have to estimate the likelihood and prior probabilities directly from the training dataset $\mathbf{D}$. Let $\mathbf{D}_i$ denote the subset of points in $\mathbf{D}$ that are labeled with class c_i:

$$\mathbf{D}_i = \{\mathbf{x}_j \in \mathbf{D} \mid \mathbf{x}_j \text{ has class } y_j = c_i\}$$

Let the size of the dataset $\mathbf{D}$ be given as $|\mathbf{D}| = n$, and let the size of each class-specific subset $\mathbf{D}_i$ be given as $|\mathbf{D}_i| = n_i$. The prior probability for class c_i can be estimated as follows:

$$\hat{P}(c_i) = \frac{n_i}{n}$$

18.1.2 Estimating the Likelihood

To estimate the likelihood $P(\mathbf{x}|c_i)$, we have to estimate the joint probability of $\mathbf{x}$ across all the d dimensions, that is, we have to estimate $P\big(\mathbf{x} = (x_1, x_2, \ldots, x_d)|c_i\big)$.

Numeric Attributes

Assuming all dimensions are numeric, we can estimate the joint probability of $\mathbf{x}$ via either a nonparametric or a parametric approach. We consider the non-parametric approach in Section 18.3.

In the parametric approach we typically assume that each class c_i is normally distributed around some mean $\boldsymbol{\mu}_i$ with a corresponding covariance matrix $\boldsymbol{\Sigma}_i$, both of which are estimated from $\mathbf{D}_i$. For class c_i, the probability density at $\mathbf{x}$ is thus given as

$$f_i(\mathbf{x}) = f(\mathbf{x}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) = \frac{1}{(\sqrt{2\pi})^d\sqrt{|\boldsymbol{\Sigma}_i|}}\exp\left\{-\frac{(\mathbf{x}-\boldsymbol{\mu}_i)^T\boldsymbol{\Sigma}_i^{-1}(\mathbf{x}-\boldsymbol{\mu}_i)}{2}\right\} \tag{18.3}$$

Because c_i is characterized by a continuous distribution, the probability of any given point must be zero, i.e., $P(\mathbf{x}|c_i) = 0$. However, we can compute the likelihood by considering a small interval $\epsilon > 0$ centered at $\mathbf{x}$:

$$P(\mathbf{x}|c_i) = 2\epsilon \cdot f_i(\mathbf{x})$$

The posterior probability is then given as

$$P(c_i|\mathbf{x}) = \frac{2\epsilon \cdot f_i(\mathbf{x})P(c_i)}{\sum_{j=1}^{k} 2\epsilon \cdot f_j(\mathbf{x})P(c_j)} = \frac{f_i(\mathbf{x})P(c_i)}{\sum_{j=1}^{k} f_j(\mathbf{x})P(c_j)} \tag{18.4}$$

Further, because $\sum_{j=1}^{k} f_j(\mathbf{x})P(c_j)$ remains fixed for $\mathbf{x}$, we can predict the class for $\mathbf{x}$ by modifying Eq. (18.2) as follows:

$$\hat{y} = \arg\max_{c_i} \Big\{ f_i(\mathbf{x})P(c_i) \Big\}$$

To classify a numeric test point $\mathbf{x}$, the Bayes classifier estimates the parameters via the sample mean and sample covariance matrix. The sample mean for the class c_i can be estimated as

$$\hat{\boldsymbol{\mu}}_i = \frac{1}{n_i} \sum_{\mathbf{x}_j \in \mathbf{D}_i} \mathbf{x}_j$$

and the sample covariance matrix for each class can be estimated using Eq. (2.30), as follows

$$\widehat{\boldsymbol{\Sigma}}_i = \frac{1}{n_i} \mathbf{Z}_i^T \mathbf{Z}_i$$

where $\mathbf{Z}_i$ is the centered data matrix for class c_i given as $\mathbf{Z}_i = \mathbf{D}_i - \mathbf{1} \cdot \hat{\boldsymbol{\mu}}_i^T$. These values can be used to estimate the probability density in Eq. (18.3) as $\hat{f}_i(\mathbf{x}) = f(\mathbf{x}|\hat{\boldsymbol{\mu}}_i, \widehat{\boldsymbol{\Sigma}}_i)$.

Algorithm 18.1 shows the pseudo-code for the Bayes classifier. Given an input dataset $\mathbf{D}$, the method estimates the prior probability, mean and covariance matrix for each class. For testing, given a test point $\mathbf{x}$, it simply returns the class with the maximum posterior probability. The cost of training is dominated by the covariance matrix computation step which takes $O(nd^2)$ time.

ALGORITHM 18.1. Bayes Classifier

BAYESCLASSIFIER ($\mathbf{D} = \{(\mathbf{x}_j, y_j)\}_{j=1}^{n}$):

1 **for** $i = 1, \ldots, k$ **do**
2 $\quad \mathbf{D}_i \leftarrow \{\mathbf{x}_j \mid y_j = c_i, j = 1, \ldots, n\}$ `// class-specific subsets`
3 $\quad n_i \leftarrow |\mathbf{D}_i|$ `// cardinality`
4 $\quad \hat{P}(c_i) \leftarrow n_i/n$ `// prior probability`
5 $\quad \hat{\boldsymbol{\mu}}_i \leftarrow \frac{1}{n_i} \sum_{\mathbf{x}_j \in \mathbf{D}_i} \mathbf{x}_j$ `// mean`
6 $\quad \mathbf{Z}_i \leftarrow \mathbf{D}_i - \mathbf{1}_{n_i} \hat{\boldsymbol{\mu}}_i^T$ `// centered data`
7 $\quad \widehat{\boldsymbol{\Sigma}}_i \leftarrow \frac{1}{n_i} \mathbf{Z}_i^T \mathbf{Z}_i$ `// covariance matrix`
8 **return** $\hat{P}(c_i), \hat{\boldsymbol{\mu}}_i, \widehat{\boldsymbol{\Sigma}}_i$ *for all* $i = 1, \ldots, k$

TESTING (x and $\hat{P}(c_i)$, $\hat{\boldsymbol{\mu}}_i$, $\widehat{\boldsymbol{\Sigma}}_i$, for all $i \in [1, k]$):

9 $\hat{y} \leftarrow \arg\max_{c_i} \{ f(\mathbf{x}|\hat{\boldsymbol{\mu}}_i, \widehat{\boldsymbol{\Sigma}}_i) \cdot P(c_i) \}$
10 **return** $\hat{y}$

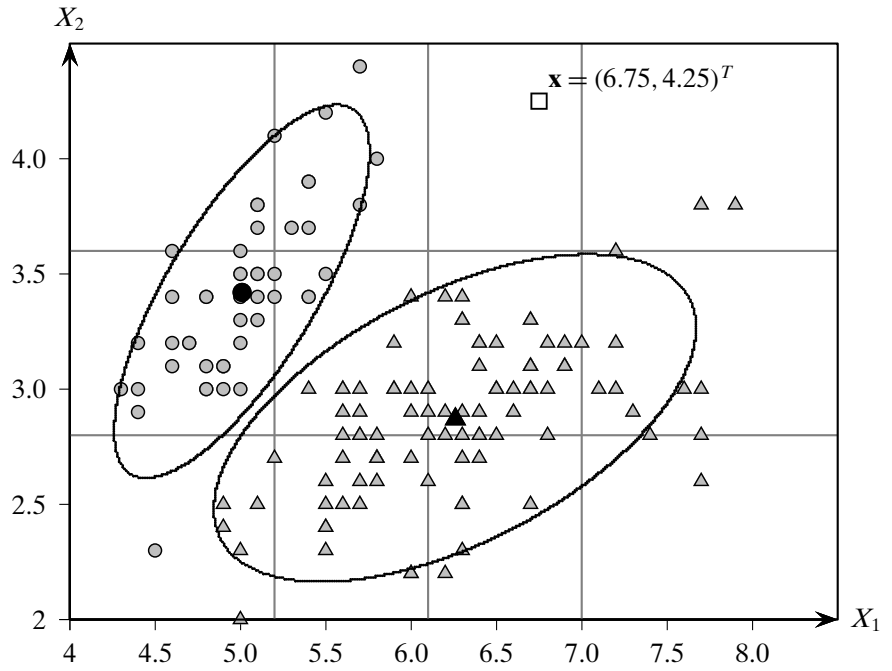


Figure 18.1. Iris data: X_1:`sepal length` versus X_2:`sepal width`. The class means are show in black; the density contours are also shown. The square represents a test point labeled $\mathbf{x}$.

Example 18.1. Consider the 2-dimensional Iris data, with attributes `sepal length` and `sepal width`, shown in Figure 18.1. Class c_1, which corresponds to `iris-setosa` (shown as circles), has $n_1 = 50$ points, whereas the other class c_2 (shown as triangles) has $n_2 = 100$ points. The prior probabilities for the two classes are

$$\hat{P}(c_1) = \frac{n_1}{n} = \frac{50}{150} = 0.33 \qquad \hat{P}(c_2) = \frac{n_2}{n} = \frac{100}{150} = 0.67$$

The means for c_1 and c_2 (shown as black circle and triangle) are given as

$$\hat{\boldsymbol{\mu}}_1 = \begin{pmatrix} 5.01 \\ 3.42 \end{pmatrix} \qquad \hat{\boldsymbol{\mu}}_2 = \begin{pmatrix} 6.26 \\ 2.87 \end{pmatrix}$$

and the corresponding covariance matrices are as follows:

$$\widehat{\boldsymbol{\Sigma}}_1 = \begin{pmatrix} 0.122 & 0.098 \\ 0.098 & 0.142 \end{pmatrix} \qquad \widehat{\boldsymbol{\Sigma}}_2 = \begin{pmatrix} 0.435 & 0.121 \\ 0.121 & 0.110 \end{pmatrix}$$

Figure 18.1 shows the contour or level curve (corresponding to 1% of the peak density) for the multivariate normal distribution modeling the probability density for both classes.

Let $\mathbf{x} = (6.75, 4.25)^T$ be a test point (shown as white square). The posterior probabilities for c_1 and c_2 can be computed using Eq. (18.4):

$$\hat{P}(c_1|\mathbf{x}) \propto \hat{f}(\mathbf{x}|\hat{\boldsymbol{\mu}}_1, \widehat{\boldsymbol{\Sigma}}_1)\hat{P}(c_1) = (4.914 \times 10^{-7}) \times 0.33 = 1.622 \times 10^{-7}$$

$$\hat{P}(c_2|\mathbf{x}) \propto \hat{f}(\mathbf{x}|\hat{\boldsymbol{\mu}}_2, \widehat{\boldsymbol{\Sigma}}_2)\hat{P}(c_2) = (2.589 \times 10^{-5}) \times 0.67 = 1.735 \times 10^{-5}$$

Because $\hat{P}(c_2|\mathbf{x}) > \hat{P}(c_1|\mathbf{x})$ the class for $\mathbf{x}$ is predicted as $\hat{y} = c_2$.

Categorical Attributes

If the attributes are categorical, the likelihood can be computed using the categorical data modeling approach presented in Chapter 3. Formally, let X_j be a categorical attribute over the domain $dom(X_j) = \{a_{j1}, a_{j2}, \ldots, a_{jm_j}\}$, that is, attribute X_j can take on m_j distinct categorical values. Each categorical attribute X_j is modeled as an m_j-dimensional multivariate Bernoulli random variable $\mathbf{X}_j$ that takes on m_j distinct vector values $\mathbf{e}_{j1}, \mathbf{e}_{j2}, \ldots, \mathbf{e}_{jm_j}$, where $\mathbf{e}_{jr}$ is the rth standard basis vector in $\mathbb{R}^{m_j}$ and corresponds to the rth value or symbol $a_{jr} \in dom(X_j)$. The entire d-dimensional dataset is modeled as the vector random variable $\mathbf{X} = (\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_d)^T$. Let $d' = \sum_{j=1}^{d} m_j$; a categorical point $\mathbf{x} = (x_1, x_2, \ldots, x_d)^T$ is therefore represented as the d'-dimensional binary vector

$$\mathbf{v} = \begin{pmatrix} \mathbf{v}_1 \\ \vdots \\ \mathbf{v}_d \end{pmatrix} = \begin{pmatrix} \mathbf{e}_{1r_1} \\ \vdots \\ \mathbf{e}_{dr_d} \end{pmatrix}$$

where $\mathbf{v}_j = \mathbf{e}_{jr_j}$ provided $x_j = a_{jr_j}$ is the r_jth value in the domain of X_j. The probability of the categorical point $\mathbf{x}$ is obtained from the joint probability mass function (PMF) for the vector random variable $\mathbf{X}$:

$$P(\mathbf{x}|c_i) = f(\mathbf{v}|c_i) = f\big(\mathbf{X}_1 = \mathbf{e}_{1r_1}, \ldots, \mathbf{X}_d = \mathbf{e}_{dr_d} | \, c_i\big) \tag{18.5}$$

The above joint PMF can be estimated directly from the data $\mathbf{D}_i$ for each class c_i as follows:

$$\hat{f}(\mathbf{v}|c_i) = \frac{n_i(\mathbf{v})}{n_i}$$

where $n_i(\mathbf{v})$ is the number of times the value $\mathbf{v}$ occurs in class c_i. Unfortunately, if the probability mass at the point $\mathbf{v}$ is zero for one or both classes, it would lead to a zero value for the posterior probability. To avoid zero probabilities, one approach is to introduce a small prior probability for all the possible values of the vector random variable $\mathbf{X}$. One simple approach is to assume a *pseudo-count* of 1 for each value, that is, to assume that each value of $\mathbf{X}$ occurs at least one time, and to augment this base count of 1 with the actual number of occurrences of the observed value $\mathbf{v}$ in class c_i. The adjusted probability mass at $\mathbf{v}$ is then given as

$$\hat{f}(\mathbf{v}|c_i) = \frac{n_i(\mathbf{v}) + 1}{n_i + \prod_{j=1}^{d} m_j} \tag{18.6}$$

where $\prod_{j=1}^{d} m_j$ gives the number of possible values of $\mathbf{X}$. Extending the code in Algorithm 18.1 to incorporate categorical attributes is relatively straightforward; all that is required is to compute the joint PMF for each class using Eq. (18.6).

Table 18.1. Discretized `sepal length` and `sepal width` attributes

Bins	Domain
[4.3, 5.2]	Very Short (a_{11})
(5.2, 6.1]	Short (a_{12})
(6.1, 7.0]	Long (a_{13})
(7.0, 7.9]	Very Long (a_{14})

(a) Discretized `sepal length`

Bins	Domain
[2.0, 2.8]	Short (a_{21})
(2.8, 3.6]	Medium (a_{22})
(3.6, 4.4]	Long (a_{23})

(b) Discretized `sepal width`

Table 18.2. Class-specific empirical (joint) probability mass function

	Class: c_1	X_2			$\hat{f}_{X_1}$
		`Short` ($\mathbf{e}_{21}$)	`Medium` ($\mathbf{e}_{22}$)	`Long` ($\mathbf{e}_{23}$)	
X_1	`Very Short` ($\mathbf{e}_{11}$)	1/50	33/50	5/50	39/50
	`Short` ($\mathbf{e}_{12}$)	0	3/50	8/50	13/50
	`Long` ($\mathbf{e}_{13}$)	0	0	0	0
	`Very Long` ($\mathbf{e}_{14}$)	0	0	0	0
	$\hat{f}_{X_2}$	1/50	36/50	13/50	

	Class: c_2	X_2			$\hat{f}_{X_1}$
		`Short` ($\mathbf{e}_{21}$)	`Medium` ($\mathbf{e}_{22}$)	`Long` ($\mathbf{e}_{23}$)	
X_1	`Very Short` ($\mathbf{e}_{11}$)	6/100	0	0	6/100
	`Short` ($\mathbf{e}_{12}$)	24/100	15/100	0	39/100
	`Long` ($\mathbf{e}_{13}$)	13/100	30/100	0	43/100
	`Very Long` ($\mathbf{e}_{14}$)	3/100	7/100	2/100	12/100
	$\hat{f}_{X_2}$	46/100	52/100	2/100	

Example 18.2. Assume that the `sepal length` and `sepal width` attributes in the Iris dataset have been discretized as shown in Table 18.1a and Table 18.1b, respectively. We have $|dom(X_1)| = m_1 = 4$ and $|dom(X_2)| = m_2 = 3$. These intervals are also illustrated in Figure 18.1: via the gray grid lines. Table 18.2 shows the empirical joint PMF for both the classes. Also, as in Example 18.1, the prior probabilities of the classes are given as $\hat{P}(c_1) = 0.33$ and $\hat{P}(c_2) = 0.67$.

Consider a test point $\mathbf{x} = (5.3, 3.0)^T$ corresponding to the categorical point (`Short, Medium`), which is represented as $\mathbf{v} = \begin{pmatrix} \mathbf{e}_{12}^T & \mathbf{e}_{22}^T \end{pmatrix}^T$. The likelihood and posterior probability for each class is given as

$$\hat{P}(\mathbf{x}|c_1) = \hat{f}(\mathbf{v}|c_1) = 3/50 = 0.06$$
$$\hat{P}(\mathbf{x}|c_2) = \hat{f}(\mathbf{v}|c_2) = 15/100 = 0.15$$
$$P(c_1|\mathbf{x}) \propto 0.06 \times 0.33 = 0.0198$$
$$P(c_2|\mathbf{x}) \propto 0.15 \times 0.67 = 0.1005$$

In this case the predicted class is $\hat{y} = c_2$.

On the other hand, the test point $\mathbf{x} = (6.75, 4.25)^T$ corresponding to the categorical point (`Long, Long`) is represented as $\mathbf{v} = \begin{pmatrix} \mathbf{e}_{13}^T & \mathbf{e}_{23}^T \end{pmatrix}^T$. Unfortunately the

probability mass at $\mathbf{v}$ is zero for both classes. We adjust the PMF via pseudo-counts [Eq. (18.6)]; note that the number of possible values are $m_1 \times m_2 = 4 \times 3 = 12$. The likelihood and prior probability can then be computed as

$$\hat{P}(\mathbf{x}|c_1) = \hat{f}(\mathbf{v}|c_1) = \frac{0+1}{50+12} = 1.61 \times 10^{-2}$$

$$\hat{P}(\mathbf{x}|c_2) = \hat{f}(\mathbf{v}|c_2) = \frac{0+1}{100+12} = 8.93 \times 10^{-3}$$

$$\hat{P}(c_1|\mathbf{x}) \propto (1.61 \times 10^{-2}) \times 0.33 = 5.32 \times 10^{-3}$$

$$\hat{P}(c_2|\mathbf{x}) \propto (8.93 \times 10^{-3}) \times 0.67 = 5.98 \times 10^{-3}$$

Thus, the predicted class is $\hat{y} = c_2$.

Challenges

The main problem with the Bayes classifier is the lack of enough data to reliably estimate the joint probability density or mass function, especially for high-dimensional data. For instance, for numeric attributes we have to estimate $O(d^2)$ covariances, and as the dimensionality increases, this requires us to estimate too many parameters. For categorical attributes we have to estimate the joint probability for all the possible values of $\mathbf{v}$, given as $\prod_j |\text{dom}(X_j)|$. Even if each categorical attribute has only two values, we would need to estimate the probability for 2^d values. However, because there can be at most n distinct values for $\mathbf{v}$, most of the counts will be zero. To address some of these concerns we can use reduced set of parameters in practice, as described next.

18.2 NAIVE BAYES CLASSIFIER

We saw earlier that the full Bayes approach is fraught with estimation related problems, especially with large number of dimensions. The naive Bayes approach makes the simple assumption that all the attributes are independent. This leads to a much simpler, though surprisingly effective classifier in practice. The independence assumption immediately implies that the likelihood can be decomposed into a product of dimension-wise probabilities:

$$P(\mathbf{x}|c_i) = P(x_1, x_2, \ldots, x_d|c_i) = \prod_{j=1}^{d} P(x_j|c_i) \tag{18.7}$$

Numeric Attributes

For numeric attributes we make the default assumption that each of them is normally distributed for each class c_i. Let μ_{ij} and σ_{ij}^2 denote the mean and variance for attribute X_j, for class c_i. The likelihood for class c_i, for dimension X_j, is given as

$$P(x_j|c_i) \propto f(x_j|\mu_{ij}, \sigma_{ij}^2) = \frac{1}{\sqrt{2\pi}\sigma_{ij}} \exp\left\{-\frac{(x_j - \mu_{ij})^2}{2\sigma_{ij}^2}\right\}$$

Incidentally, the naive assumption corresponds to setting all the covariances to zero in $\mathbf{\Sigma}_i$, that is,

$$\mathbf{\Sigma}_i = \begin{pmatrix} \sigma_{i1}^2 & 0 & \dots & 0 \\ 0 & \sigma_{i2}^2 & \dots & 0 \\ \vdots & \vdots & \ddots & \\ 0 & 0 & \dots & \sigma_{id}^2 \end{pmatrix}$$

This yields

$$|\mathbf{\Sigma}_i| = \det(\mathbf{\Sigma}_i) = \sigma_{i1}^2 \sigma_{i2}^2 \cdots \sigma_{id}^2 = \prod_{j=1}^{d} \sigma_{ij}^2$$

Also, we have

$$\mathbf{\Sigma}_i^{-1} = \begin{pmatrix} \frac{1}{\sigma_{i1}^2} & 0 & \dots & 0 \\ 0 & \frac{1}{\sigma_{i2}^2} & \dots & 0 \\ \vdots & \vdots & \ddots & \\ 0 & 0 & \dots & \frac{1}{\sigma_{id}^2} \end{pmatrix}$$

assuming that $\sigma_{ij}^2 \neq 0$ for all j. Finally,

$$(\mathbf{x} - \boldsymbol{\mu}_i)^T \mathbf{\Sigma}_i^{-1} (\mathbf{x} - \boldsymbol{\mu}_i) = \sum_{j=1}^{d} \frac{(x_j - \mu_{ij})^2}{\sigma_{ij}^2}$$

Plugging these into Eq. (18.3) gives us

$$\begin{aligned} P(\mathbf{x}|c_i) &= \frac{1}{(\sqrt{2\pi})^d \sqrt{\prod_{j=1}^{d} \sigma_{ij}^2}} \exp\left\{ -\sum_{j=1}^{d} \frac{(x_j - \mu_{ij})^2}{2\sigma_{ij}^2} \right\} \\ &= \prod_{j=1}^{d} \left(\frac{1}{\sqrt{2\pi}\, \sigma_{ij}} \exp\left\{ -\frac{(x_j - \mu_{ij})^2}{2\sigma_{ij}^2} \right\} \right) \\ &= \prod_{j=1}^{d} P(x_j|c_i) \end{aligned}$$

which is equivalent to Eq. (18.7). In other words, the joint probability has been decomposed into a product of the probability along each dimension, as required by the independence assumption.

The naive Bayes classifier uses the sample mean $\hat{\boldsymbol{\mu}}_i = (\hat{\mu}_{i1}, \dots, \hat{\mu}_{id})^T$ and a *diagonal* sample covariance matrix $\widehat{\mathbf{\Sigma}}_i = diag(\sigma_{i1}^2, \dots, \sigma_{id}^2)$ for each class c_i. Thus, in total $2d$ parameters have to be estimated, corresponding to the sample mean and sample variance for each dimension X_j.

Algorithm 18.2 shows the pseudo-code for the naive Bayes classifier. Given an input dataset $\mathbf{D}$, the method estimates the prior probability and mean for each class. Next, it computes the variance $\hat{\sigma}_{ij}^2$ for each of the attributes X_j, with all the d variances for class c_i stored in the vector $\hat{\boldsymbol{\sigma}}_i$. The variance for attribute X_j is obtained by first

ALGORITHM 18.2. Naive Bayes Classifier

NAIVEBAYES ($\mathbf{D} = \{(\mathbf{x}_j, y_j)\}_{j=1}^n$):

for $i = 1, \ldots, k$ **do**
  $\mathbf{D}_i \leftarrow \{\mathbf{x}_j \mid y_j = c_i, j = 1, \ldots, n\}$ // class-specific subsets
  $n_i \leftarrow |\mathbf{D}_i|$ // cardinality
  $\hat{P}(c_i) \leftarrow n_i / n$ // prior probability
  $\hat{\boldsymbol{\mu}}_i \leftarrow \frac{1}{n_i} \sum_{\mathbf{x}_j \in \mathbf{D}_i} \mathbf{x}_j$ // mean
  $\mathbf{Z}_i = \mathbf{D}_i - \mathbf{1} \cdot \hat{\boldsymbol{\mu}}_i^T$ // centered data for class c_i
  for $j = 1, .., d$ **do** // class-specific variance for X_j
    $\hat{\sigma}_{ij}^2 \leftarrow \frac{1}{n_i} Z_{ij}^T Z_{ij}$ // variance
  $\hat{\boldsymbol{\sigma}}_i = \left(\hat{\sigma}_{i1}^2, \ldots, \hat{\sigma}_{id}^2\right)^T$ // class-specific attribute variances
return $\hat{P}(c_i), \hat{\boldsymbol{\mu}}_i, \hat{\boldsymbol{\sigma}}_i$ for all $i = 1, \ldots, k$

TESTING (x and $\hat{P}(c_i)$, $\hat{\boldsymbol{\mu}}_i$, $\hat{\boldsymbol{\sigma}}_i$, for all $i \in [1, k]$):

$$\hat{y} \leftarrow \arg\max_{c_i} \left\{ \hat{P}(c_i) \prod_{j=1}^{d} f(x_j | \hat{\mu}_{ij}, \hat{\sigma}_{ij}^2) \right\}$$

return $\hat{y}$

centering the data for class $\mathbf{D}_i$ via $\mathbf{Z}_i = \mathbf{D}_i - \mathbf{1} \cdot \hat{\boldsymbol{\mu}}_i^T$. We denote by Z_{ij} the centered data for class c_i corresponding to attribute X_j. The variance is then given as $\hat{\sigma} = \frac{1}{n_i} Z_{ij}^T Z_{ij}$.

Training the naive Bayes classifier is very fast, with $O(nd)$ computational complexity. For testing, given a test point $\mathbf{x}$, it simply returns the class with the maximum posterior probability obtained as a product of the likelihood for each dimension and the class prior probability.

Example 18.3. Consider Example 18.1. In the naive Bayes approach the prior probabilities $\hat{P}(c_i)$ and means $\hat{\boldsymbol{\mu}}_i$ remain unchanged. The key difference is that the covariance matrices are assumed to be diagonal, as follows:

$$\widehat{\boldsymbol{\Sigma}}_1 = \begin{pmatrix} 0.122 & 0 \\ 0 & 0.142 \end{pmatrix} \qquad \widehat{\boldsymbol{\Sigma}}_2 = \begin{pmatrix} 0.435 & 0 \\ 0 & 0.110 \end{pmatrix}$$

Figure 18.2 shows the contour or level curve (corresponding to 1% of the peak density) of the multivariate normal distribution for both classes. One can see that the diagonal assumption leads to contours that are axis-parallel ellipses; contrast these with the contours in Figure 18.1 for the full Bayes classifier.

For the test point $\mathbf{x} = (6.75, 4.25)^T$, the posterior probabilities for c_1 and c_2 are as follows:

$$\hat{P}(c_1|\mathbf{x}) \propto \hat{f}(\mathbf{x}|\hat{\boldsymbol{\mu}}_1, \widehat{\boldsymbol{\Sigma}}_1)\hat{P}(c_1) = (3.99 \times 10^{-7}) \times 0.33 = 1.32 \times 10^{-7}$$

$$\hat{P}(c_2|\mathbf{x}) \propto \hat{f}(\mathbf{x}|\hat{\boldsymbol{\mu}}_2, \widehat{\boldsymbol{\Sigma}}_2)\hat{P}(c_2) = (9.597 \times 10^{-5}) \times 0.67 = 6.43 \times 10^{-5}$$

Because $\hat{P}(c_2|\mathbf{x}) > \hat{P}(c_1|\mathbf{x})$ the class for $\mathbf{x}$ is predicted as $\hat{y} = c_2$.

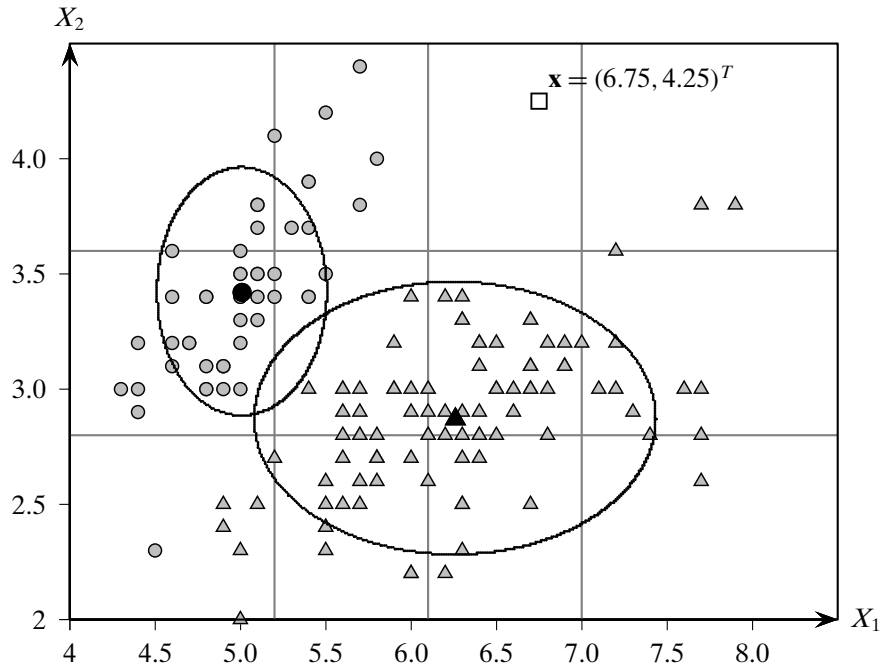


Figure 18.2. Naive Bayes: X_1:`sepal length` versus X_2:`sepal width`. The class means are shown in black; the density contours are also shown. The square represents a test point labeled $\mathbf{x}$.

Categorical Attributes

The independence assumption leads to a simplification of the joint probability mass function in Eq. (18.5), which can be rewritten as

$$P(\mathbf{x}|c_i) = \prod_{j=1}^{d} P(x_j|c_i) = \prod_{j=1}^{d} f\left(\mathbf{X}_j = \mathbf{e}_{jr_j} \mid c_i\right)$$

where $f(\mathbf{X}_j = \mathbf{e}_{jr_j}|c_i)$ is the probability mass function for $\mathbf{X}_j$, which can be estimated from $\mathbf{D}_i$ as follows:

$$\hat{f}(\mathbf{v}_j|c_i) = \frac{n_i(\mathbf{v}_j)}{n_i}$$

where $n_i(\mathbf{v}_j)$ is the observed frequency of the value $\mathbf{v}_j = \mathbf{e}_j r_j$ corresponding to the r_jth categorical value a_{jr_j} for the attribute X_j for class c_i. As in the full Bayes case, if the count is zero, we can use the pseudo-count method to obtain a prior probability. The adjusted estimates with pseudo-counts are given as

$$\hat{f}(\mathbf{v}_j|c_i) = \frac{n_i(\mathbf{v}_j) + 1}{n_i + m_j}$$

where $m_j = |dom(X_j)|$. Extending the code in Algorithm 18.2 to incorporate categorical attributes is straightforward.

Example 18.4. Continuing Example 18.2, the class-specific PMF for each discretized attribute is shown in Table 18.2. In particular, these correspond to the row and column marginal probabilities $\hat{f}_{X_1}$ and $\hat{f}_{X_2}$, respectively.

The test point $\mathbf{x} = (6.75, 4.25)$, corresponding to (Long, Long) or $\mathbf{v} = (\mathbf{e}_{13}, \mathbf{e}_{23})$, is classified as follows:

$$\hat{P}(\mathbf{v}|c_1) = \hat{P}(\mathbf{e}_{13}|c_1) \cdot \hat{P}(\mathbf{e}_{23}|c_1) = \left(\frac{0+1}{50+4}\right) \cdot \left(\frac{13}{50}\right) = 4.81 \times 10^{-3}$$

$$\hat{P}(\mathbf{v}|c_2) = \hat{P}(\mathbf{e}_{13}|c_2) \cdot \hat{P}(\mathbf{e}_{23}|c_2) = \left(\frac{43}{100}\right) \cdot \left(\frac{2}{100}\right) = 8.60 \times 10^{-3}$$

$$\hat{P}(c_1|\mathbf{v}) \propto (4.81 \times 10^{-3}) \times 0.33 = 1.59 \times 10^{-3}$$

$$\hat{P}(c_2|\mathbf{v}) \propto (8.6 \times 10^{-3}) \times 0.67 = 5.76 \times 10^{-3}$$

Thus, the predicted class is $\hat{y} = c_2$.

18.3 K NEAREST NEIGHBORS CLASSIFIER

In the preceding sections we considered a parametric approach for estimating the likelihood $P(\mathbf{x}|c_i)$. In this section, we consider a non-parametric approach, which does not make any assumptions about the underlying joint probability density function. Instead, it directly uses the data sample to estimate the density, for example, using the density estimation methods from Chapter 15. We illustrate the non-parametric approach using nearest neighbors density estimation from Section 15.2.3, which leads to the *K nearest neighbors* (KNN) classifier.

Let $\mathbf{D}$ be a training dataset comprising n points $\mathbf{x}_i \in \mathbb{R}^d$, and let $\mathbf{D}_i$ denote the subset of points in $\mathbf{D}$ that are labeled with class c_i, with $n_i = |\mathbf{D}_i|$. Given a test point $\mathbf{x} \in \mathbb{R}^d$, and K, the number of neighbors to consider, let r denote the distance from $\mathbf{x}$ to its Kth nearest neighbor in $\mathbf{D}$.

Consider the d-dimensional hyperball of radius r around the test point $\mathbf{x}$, defined as

$$B_d(\mathbf{x}, r) = \{\mathbf{x}_i \in \mathbf{D} \mid \delta(\mathbf{x}, \mathbf{x}_i) \leq r\}$$

Here $\delta(\mathbf{x}, \mathbf{x}_i)$ is the distance between $\mathbf{x}$ and $\mathbf{x}_i$, which is usually assumed to be the Euclidean distance, i.e., $\delta(\mathbf{x}, \mathbf{x}_i) = \|\mathbf{x} - \mathbf{x}_i\|_2$. However, other distance metrics can also be used. We assume that $|B_d(\mathbf{x}, r)| = K$.

Let K_i denote the number of points among the K nearest neighbors of $\mathbf{x}$ that are labeled with class c_i, that is

$$K_i = \{\mathbf{x}_j \in B_d(\mathbf{x}, r) \mid y_j = c_i\}$$

The class conditional probability density at $\mathbf{x}$ can be estimated as the fraction of points from class c_i that lie within the hyperball divided by its volume, that is

$$\hat{f}(\mathbf{x}|c_i) = \frac{K_i/n_i}{V} = \frac{K_i}{n_i V}$$

where $V = \text{vol}(B_d(\mathbf{x}, r))$ is the volume of the d-dimensional hyperball [Eq. (6.4)].

Using Eq. (18.4), the posterior probability $P(c_i|\mathbf{x})$ can be estimated as

$$P(c_i|\mathbf{x}) = \frac{\hat{f}(\mathbf{x}|c_i)\hat{P}(c_i)}{\sum_{j=1}^{k} \hat{f}(\mathbf{x}|c_j)\hat{P}(c_j)}$$

However, because $\hat{P}(c_i) = \frac{n_i}{n}$, we have

$$\hat{f}(\mathbf{x}|c_i)\hat{P}(c_i) = \frac{K_i}{n_i V} \cdot \frac{n_i}{n} = \frac{K_i}{nV}$$

Thus the posterior probability is given as

$$P(c_i|\mathbf{x}) = \frac{\frac{K_i}{nV}}{\sum_{j=1}^{k} \frac{K_j}{nV}} = \frac{K_i}{K}$$

Finally, the predicted class for $\mathbf{x}$ is

$$\hat{y} = \arg\max_{c_i}\{P(c_i|\mathbf{x})\} = \arg\max_{c_i}\left\{\frac{K_i}{K}\right\} = \arg\max_{c_i}\{K_i\}$$

Because K is fixed, the KNN classifier predicts the class of $\mathbf{x}$ as the majority class among its K nearest neighbors.

Example 18.5. Consider the 2D Iris dataset shown in Figure 18.3. The two classes are: c_1 (circles) with $n_1 = 50$ points and c_2 (triangles) with $n_2 = 100$ points.

Let us classify the test point $\mathbf{x} = (6.75, 4.25)^T$ using its $K = 5$ nearest neighbors. The distance from $\mathbf{x}$ to its 5th nearest neighbor, namely $(6.2, 3.4)^T$, is given as $r = \sqrt{1.025} = 1.012$. The enclosing ball or circle of radius r is shown in the figure. It encompasses $K_1 = 1$ point from class c_1 and $K_2 = 4$ points from class c_2. Therefore, the predicted class for $\mathbf{x}$ is $\hat{y} = c_2$.

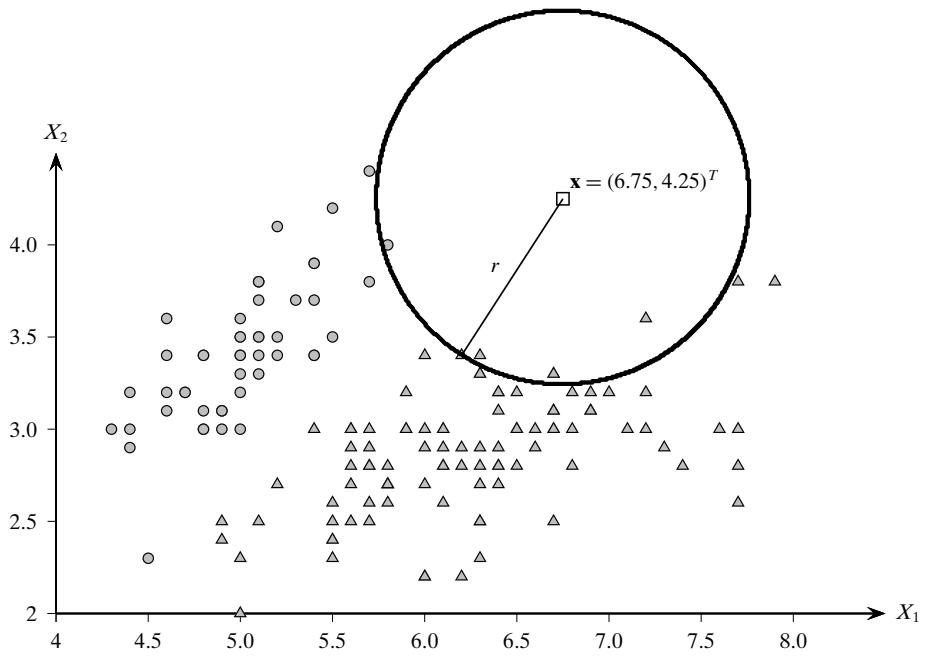


Figure 18.3. Iris Data: K Nearest Neighbors Classifier

18.4 FURTHER READING

The naive Bayes classifier is surprisingly effective even though the independence assumption is usually violated in real datasets. Comparison of the naive Bayes classifier against other classification approaches and reasons for why is works well have appeared in Langley, Iba, and Thompson (1992); Domingos and Pazzani (1997); Zhang (2005); Hand and Yu (2001) and Rish (2001). For the long history of naive Bayes in information retrieval see Lewis (1998). The K nearest neighbor classification approach was first proposed in Fix and Hodges, Jr. (1951).

Domingos, P. and Pazzani, M. (1997). "On the optimality of the simple Bayesian classifier under zero-one loss." *Machine Learning*, 29 (2–3): 103–130.

Fix, E. and Hodges Jr., J. L. (1951). Discriminatory analysis, nonparametric discrimination. *USAF School of Aviation Medicine, Randolph Field, TX, Project 21-49-004, Report 4, Contract AF41(128)-31*.

Hand, D. J. and Yu, K. (2001). "Idiot's Bayes-not so stupid after all?" *International Statistical Review*, 69 (3): 385–398.

Langley, P., Iba, W., and Thompson, K. (1992). "An analysis of Bayesian classifiers." *In Proceedings of the National Conference on Artificial Intelligence*, pp. 223–223.

Lewis, D. D. (1998). "Naive (Bayes) at forty: The independence assumption in information retrieval." *In Proceedings of the 10th European Conference on Machine Learning.* pp. 4–15.

Rish, I. (2001). "An empirical study of the naive Bayes classifier." *In Proceedings of the IJCAI Workshop on Empirical Methods in Artificial Intelligence*, pp. 41–46.

Zhang, H. (2005). "Exploring conditions for the optimality of naive Bayes." *International Journal of Pattern Recognition and Artificial Intelligence*, 19 (2): 183–198.

18.5 EXERCISES

Q1. Consider the dataset in Table 18.3. Classify the new point: (Age=23, Car=truck) via the full and naive Bayes approach. You may assume that the domain of Car is given as {sports, vintage, suv, truck}.

Table 18.3. Data for Q1

$\mathbf{x}_i$	Age	Car	Class
$\mathbf{x}_1$	25	sports	L
$\mathbf{x}_2$	20	vintage	H
$\mathbf{x}_3$	25	sports	L
$\mathbf{x}_4$	45	suv	H
$\mathbf{x}_5$	20	sports	H
$\mathbf{x}_6$	25	suv	H

Q2. Given the dataset in Table 18.4, use the naive Bayes classifier to classify the new point $(T, F, 1.0)$.

Table 18.4. Data for Q2

$\mathbf{x}_i$	a_1	a_2	a_3	Class
$\mathbf{x}_1$	T	T	5.0	Y
$\mathbf{x}_2$	T	T	7.0	Y
$\mathbf{x}_3$	T	F	8.0	N
$\mathbf{x}_4$	F	F	3.0	Y
$\mathbf{x}_5$	F	T	7.0	N
$\mathbf{x}_6$	F	T	4.0	N
$\mathbf{x}_7$	F	F	5.0	N
$\mathbf{x}_8$	T	F	6.0	Y
$\mathbf{x}_9$	F	T	1.0	N

Q3. Consider the class means and covariance matrices for classes c_1 and c_2:

$$\boldsymbol{\mu}_1 = (1,3) \qquad \boldsymbol{\mu}_2 = (5,5)$$

$$\boldsymbol{\Sigma}_1 = \begin{pmatrix} 5 & 3 \\ 3 & 2 \end{pmatrix} \qquad \boldsymbol{\Sigma}_2 = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}$$

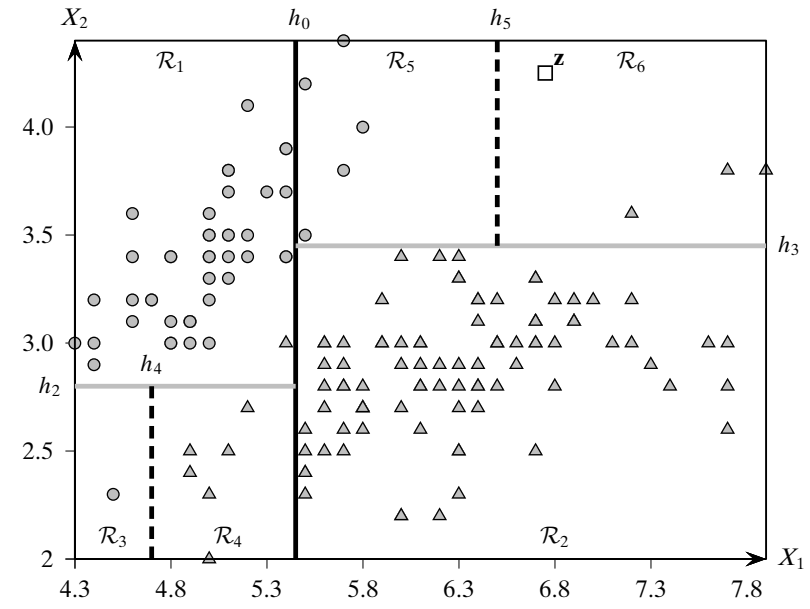
Classify the point $(3,4)^T$ via the (full) Bayesian approach, assuming normally distributed classes, and $P(c_1) = P(c_2) = 0.5$. Show all steps. Recall that the inverse of a 2×2 matrix $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ is given as $A^{-1} = \frac{1}{\det(A)} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$.

CHAPTER 19 Decision Tree Classifier

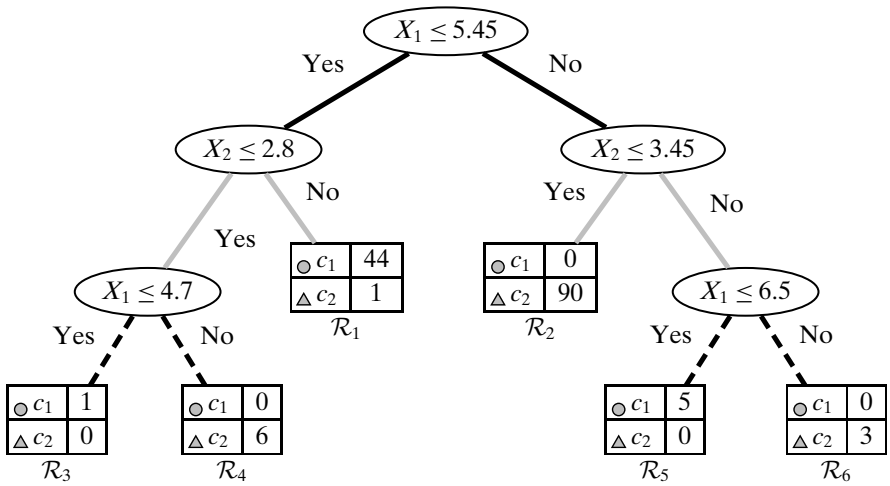
Let the training dataset $\mathbf{D} = \{\mathbf{x}_i, y_i\}_{i=1}^n$ consist of n points in a d-dimensional space, with y_i being the class label for point $\mathbf{x}_i$. We assume that the dimensions or the attributes X_j are numeric or categorical, and that there are k distinct classes, so that $y_i \in \{c_1, c_2, \ldots, c_k\}$. A decision tree classifier is a recursive, partition-based tree model that predicts the class $\hat{y}_i$ for each point $\mathbf{x}_i$. Let $\mathcal{R}$ denote the data space that encompasses the set of input points $\mathbf{D}$. A decision tree uses an axis-parallel hyperplane to split the data space $\mathcal{R}$ into two resulting half-spaces or regions, say $\mathcal{R}_1$ and $\mathcal{R}_2$, which also induces a partition of the input points into $\mathbf{D}_1$ and $\mathbf{D}_2$, respectively. Each of these regions is recursively split via axis-parallel hyperplanes until the points within an induced partition are relatively pure in terms of their class labels, that is, most of the points belong to the same class. The resulting hierarchy of split decisions constitutes the decision tree model, with the leaf nodes labeled with the majority class among points in those regions. To classify a new *test* point we have to recursively evaluate which half-space it belongs to until we reach a leaf node in the decision tree, at which point we predict its class as the label of the leaf.

Example 19.1. Consider the Iris dataset shown in Figure 19.1a, which plots the attributes `sepal length` (X_1) and `sepal width` (X_2). The classification task is to discriminate between c_1, corresponding to `iris-setosa` (in circles), and c_2, corresponding to the other two types of Irises (in triangles). The input dataset $\mathbf{D}$ has $n = 150$ points that lie in the data space which is given as the rectangle, $\mathcal{R} = range(X_1) \times range(X_2) = [4.3, 7.9] \times [2.0, 4.4]$.

The recursive partitioning of the space $\mathcal{R}$ via axis-parallel hyperplanes is illustrated in Figure 19.1a. In two dimensions a hyperplane is simply a line. The first split corresponds to hyperplane h_0 shown as a black line. The resulting left and right half-spaces are further split via hyperplanes h_2 and h_3, respectively (shown as gray lines). The bottom half-space for h_2 is further split via h_4, and the top half-space for h_3 is split via h_5; these third level hyperplanes, h_4 and h_5, are shown as dashed lines. The set of hyperplanes and the set of six leaf regions, namely $\mathcal{R}_1, \ldots, \mathcal{R}_6$, constitute the decision tree model. Note also the induced partitioning of the input points into these six regions.



(a) Recursive Splits



(b) Decision Tree

Figure 19.1. Decision trees: recursive partitioning via axis-parallel hyperplanes.

Consider the test point $\mathbf{z} = (6.75, 4.25)^T$ (shown as a white square). To predict its class, the decision tree first checks which side of h_0 it lies in. Because the point lies in the right half-space, the decision tree next checks h_3 to determine that $\mathbf{z}$ is in the top half-space. Finally, we check and find that $\mathbf{z}$ is in the right half-space of h_5, and we reach the leaf region $\mathcal{R}_6$. The predicted class is c_2, as that leaf region has all points (three of them) with class c_2 (triangles).

19.1 DECISION TREES

A decision tree consists of internal nodes that represent the decisions corresponding to the hyperplanes or split points (i.e., which half-space a given point lies in), and leaf nodes that represent regions or partitions of the data space, which are labeled with the majority class. A region is characterized by the subset of data points that lie in that region.

Axis-Parallel Hyperplanes

A hyperplane $h(\mathbf{x})$ is defined as the set of all points $\mathbf{x}$ that satisfy the following equation

$$h(\mathbf{x}): \mathbf{w}^T\mathbf{x} + b = 0 \tag{19.1}$$

Here $\mathbf{w} \in \mathbb{R}^d$ is a *weight vector* that is normal to the hyperplane, and b is the offset of the hyperplane from the origin. A decision tree considers only *axis-parallel hyperplanes*, that is, the weight vector must be parallel to one of the original dimensions or axes X_j. Put differently, the weight vector $\mathbf{w}$ is restricted *a priori* to one of the standard basis vectors $\{\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_d\}$, where $\mathbf{e}_i \in \mathbb{R}^d$ has a 1 for the jth dimension, and 0 for all other dimensions. If $\mathbf{x} = (x_1, x_2, \ldots, x_d)^T$ and assuming $\mathbf{w} = \mathbf{e}_j$, we can rewrite Eq. (19.1) as

$$h(\mathbf{x}): \mathbf{e}_j^T\mathbf{x} + b = 0, \text{ which implies that}$$
$$h(\mathbf{x}): x_j + b = 0$$

where the choice of the offset b yields different hyperplanes along dimension X_j.

Split Points

A hyperplane specifies a decision or *split point* because it splits the data space $\mathcal{R}$ into two half-spaces. All points $\mathbf{x}$ such that $h(\mathbf{x}) \leq 0$ are on the hyperplane or to one side of the hyperplane, whereas all points such that $h(\mathbf{x}) > 0$ are on the other side. The split point associated with an axis-parallel hyperplane can be written as $h(\mathbf{x}) \leq 0$, which implies that $x_i + b \leq 0$, or $x_i \leq -b$. Because x_i is some value from dimension X_j and the offset b can be chosen to be any value, the generic form of a split point for a numeric attribute X_j is given as

$$X_j \leq v$$

where $v = -b$ is some value in the domain of attribute X_j. The decision or split point $X_j \leq v$ thus splits the input data space $\mathcal{R}$ into two regions $\mathcal{R}_Y$ and $\mathcal{R}_N$, which denote the set of *all possible points* that satisfy the decision and those that do not.

Data Partition

Each split of $\mathcal{R}$ into $\mathcal{R}_Y$ and $\mathcal{R}_N$ also induces a binary partition of the corresponding input data points $\mathbf{D}$. That is, a split point of the form $X_j \leq v$ induces the data partition

$$\mathbf{D}_Y = \{\mathbf{x} \mid \mathbf{x} \in \mathbf{D}, x_j \leq v\}$$
$$\mathbf{D}_N = \{\mathbf{x} \mid \mathbf{x} \in \mathbf{D}, x_j > v\}$$

where $\mathbf{D}_Y$ is the subset of data points that lie in region $\mathcal{R}_Y$ and $\mathbf{D}_N$ is the subset of input points that line in $\mathcal{R}_N$.

Purity

The purity of a region $\mathcal{R}_j$ is defined in terms of the mixture of classes for points in the corresponding data partition $\mathbf{D}_j$. Formally, purity is the fraction of points with the majority label in $\mathbf{D}_j$, that is,

$$purity(\mathbf{D}_j) = \max_i \left\{ \frac{n_{ji}}{n_j} \right\} \tag{19.2}$$

where $n_j = |\mathbf{D}_j|$ is the total number of data points in the region $\mathcal{R}_j$, and n_{ji} is the number of points in $\mathbf{D}_j$ with class label c_i.

Example 19.2. Figure 19.1b shows the resulting decision tree that corresponds to the recursive partitioning of the space via axis-parallel hyperplanes illustrated in Figure 19.1a. The recursive splitting terminates when appropriate stopping conditions are met, usually taking into account the size and purity of the regions. In this example, we use a size threshold of 5 and a purity threshold of 0.95. That is, a region will be split further only if the number of points is more than five and the purity is less than 0.95.

The very first hyperplane to be considered is $h_1(\mathbf{x}) : x_1 - 5.45 = 0$ which corresponds to the decision

$$X_1 \leq 5.45$$

at the root of the decision tree. The two resulting half-spaces are recursively split into smaller half-spaces.

For example, the region $X_1 \leq 5.45$ is further split using the hyperplane $h_2(\mathbf{x}) : x_2 - 2.8 = 0$ corresponding to the decision

$$X_2 \leq 2.8$$

which forms the left child of the root. Notice how this hyperplane is restricted only to the region $X_1 \leq 5.45$. This is because each region is considered independently after the split, as if it were a separate dataset. There are seven points that satisfy the condition $X_2 \leq 2.8$, out of which one is from class c_1 (circle) and six are from class c_2 (triangles). The purity of this region is therefore $6/7 = 0.857$. Because the region has more than five points, and its purity is less than 0.95, it is further split via the hyperplane $h_4(\mathbf{x}) : x_1 - 4.7 = 0$ yielding the left-most decision node

$$X_1 \leq 4.7$$

in the decision tree shown in Figure 19.1b.

Returning back to the right half-space corresponding to h_2, namely the region $X_2 > 2.8$, it has 45 points, of which only one is a triangle. The size of the region is 45, but the purity is $44/45 = 0.98$. Because the region exceeds the purity threshold it is not split further. Instead, it becomes a leaf node in the decision tree, and the entire region ($\mathcal{R}_1$) is labeled with the majority class c_1. The frequency for each class is also noted at a leaf node so that the potential error rate for that leaf can be computed. For example, we can expect that the probability of misclassification in region $\mathcal{R}_1$ is $1/45 = 0.022$, which is the error rate for that leaf.

Categorical Attributes

In addition to numeric attributes, a decision tree can also handle categorical data. For a categorical attribute X_j, the split points or decisions are of the $X_j \in V$, where $V \subset dom(X_j)$, and $dom(X_j)$ denotes the domain for X_j. Intuitively, this split can be considered to be the categorical analog of a hyperplane. It results in two "half-spaces," one region $\mathcal{R}_Y$ consisting of points $\mathbf{x}$ that satisfy the condition $x_i \in V$, and the other region $\mathcal{R}_N$ comprising points that satisfy the condition $x_i \notin V$.

Decision Rules

One of the advantages of decision trees is that they produce models that are relatively easy to interpret. In particular, a tree can be read as set of decision rules, with each rule's antecedent comprising the decisions on the internal nodes along a path to a leaf, and its consequent being the label of the leaf node. Further, because the regions are all disjoint and cover the entire space, the set of rules can be interpreted as a set of alternatives or disjunctions.

Example 19.3. Consider the decision tree in Figure 19.1b. It can be interpreted as the following set of disjunctive rules, one per leaf region $\mathcal{R}_i$

$\mathcal{R}_3$: If $X_1 \leq 5.45$ and $X_2 \leq 2.8$ and $X_1 \leq 4.7$, then class is c_1, or

$\mathcal{R}_4$: If $X_1 \leq 5.45$ and $X_2 \leq 2.8$ and $X_1 > 4.7$, then class is c_2, or

$\mathcal{R}_1$: If $X_1 \leq 5.45$ and $X_2 > 2.8$, then class is c_1, or

$\mathcal{R}_2$: If $X_1 > 5.45$ and $X_2 \leq 3.45$, then class is c_2, or

$\mathcal{R}_5$: If $X_1 > 5.45$ and $X_2 > 3.45$ and $X_1 \leq 6.5$, then class is c_1, or

$\mathcal{R}_6$: If $X_1 > 5.45$ and $X_2 > 3.45$ and $X_1 > 6.5$, then class is c_2

19.2 DECISION TREE ALGORITHM

The pseudo-code for decision tree model construction is shown in Algorithm 19.1. It takes as input a training dataset $\mathbf{D}$, and two parameters η and π, where η is the leaf size and π the leaf purity threshold. Different split points are evaluated for each attribute in $\mathbf{D}$. Numeric decisions are of the form $X_j \leq v$ for some value v in the value range for attribute X_j, and categorical decisions are of the form $X_j \in V$ for some subset of values in the domain of X_j. The best split point is chosen to partition the data into two subsets, $\mathbf{D}_Y$ and $\mathbf{D}_N$, where $\mathbf{D}_Y$ corresponds to all points $\mathbf{x} \in \mathbf{D}$ that satisfy the split decision, and $\mathbf{D}_N$ corresponds to all points that do not satisfy the split decision. The decision tree method is then called recursively on $\mathbf{D}_Y$ and $\mathbf{D}_N$. A number of stopping conditions can be used to stop the recursive partitioning process. The simplest condition is based on the size of the partition $\mathbf{D}$. If the number of points n in $\mathbf{D}$ drops below the user-specified size threshold η, then we stop the partitioning process and make $\mathbf{D}$ a leaf. This condition prevents over-fitting the model to the training set, by avoiding to model very small subsets of the data. Size alone is not sufficient because if

ALGORITHM 19.1. Decision Tree Algorithm

```
DECISIONTREE (D, η, π):
1  n ← |D| // partition size
2  n_i ← |{x_j | x_j ∈ D, y_j = c_i}| // size of class c_i
3  purity(D) ← max_i {n_i / n}
4  if n ≤ η or purity(D) ≥ π then // stopping condition
5      c* ← arg max_{c_i} {n_i / n} // majority class
6      create leaf node, and label it with class c*
7      return
8  (split point*, score*) ← (∅, 0) // initialize best split point
9  foreach (attribute X_j) do
10     if (X_j is numeric) then
11         (v, score) ← EVALUATE-NUMERIC-ATTRIBUTE(D, X_j)
12         if score > score* then (split point*, score*) ← (X_j ≤ v, score)
13     else if (X_j is categorical) then
14         (V, score) ← EVALUATE-CATEGORICAL-ATTRIBUTE(D, X_j)
15         if score > score* then (split point*, score*) ← (X_j ∈ V, score)
   // partition D into D_Y and D_N using split point*, and call
      recursively
16 D_Y ← {x ∈ D | x satisfies split point*}
17 D_N ← {x ∈ D | x does not satisfy split point*}
18 create internal node split point*, with two child nodes, D_Y and D_N
19 DECISIONTREE(D_Y); DECISIONTREE(D_N)
```

the partition is already pure then it does not make sense to split it further. Thus, the recursive partitioning is also terminated if the purity of $\mathbf{D}$ is above the purity threshold π. Details of how the split points are evaluated and chosen are given next.

19.2.1 Split Point Evaluation Measures

Given a split point of the form $X_j \leq v$ or $X_j \in V$ for a numeric or categorical attribute, respectively, we need an objective criterion for scoring the split point. Intuitively, we want to select a split point that gives the best separation or discrimination between the different class labels.

Entropy

Entropy, in general, measures the amount of disorder or uncertainty in a system. In the classification setting, a partition has lower entropy (or low disorder) if it is relatively pure, that is, if most of the points have the same label. On the other hand, a partition has higher entropy (or more disorder) if the class labels are mixed, and there is no majority class as such.

The entropy of a set of labeled points $\mathbf{D}$ is defined as follows:

$$H(\mathbf{D}) = -\sum_{i=1}^{k} P(c_i|\mathbf{D}) \log_2 P(c_i|\mathbf{D}) \tag{19.3}$$

where $P(c_i|\mathbf{D})$ is the probability of class c_i in $\mathbf{D}$, and k is the number of classes. If a region is pure, that is, has points from the same class, then the entropy is zero. On the other hand, if the classes are all mixed up, and each appears with equal probability $P(c_i|\mathbf{D}) = \frac{1}{k}$, then the entropy has the highest value, $H(\mathbf{D}) = \log_2 k$.

Assume that a split point partitions $\mathbf{D}$ into $\mathbf{D}_Y$ and $\mathbf{D}_N$. Define the *split entropy* as the weighted entropy of each of the resulting partitions, given as

$$H(\mathbf{D}_Y, \mathbf{D}_N) = \frac{n_Y}{n} H(\mathbf{D}_Y) + \frac{n_N}{n} H(\mathbf{D}_N) \tag{19.4}$$

where $n = |\mathbf{D}|$ is the number of points in $\mathbf{D}$, and $n_Y = |\mathbf{D}_Y|$ and $n_N = |\mathbf{D}_N|$ are the number of points in $\mathbf{D}_Y$ and $\mathbf{D}_N$.

To see if the split point results in a reduced overall entropy, we define the *information gain* for a given split point as follows:

$$Gain(\mathbf{D}, \mathbf{D}_Y, \mathbf{D}_N) = H(\mathbf{D}) - H(\mathbf{D}_Y, \mathbf{D}_N) \tag{19.5}$$

The higher the information gain, the more the reduction in entropy, and the better the split point. Thus, given split points and their corresponding partitions, we can score each split point and choose the one that gives the highest information gain.

Gini Index

Another common measure to gauge the purity of a split point is the *Gini index*, defined as follows:

$$G(\mathbf{D}) = 1 - \sum_{i=1}^{k} P(c_i|\mathbf{D})^2 \tag{19.6}$$

If the partition is pure, then the probability of the majority class is 1 and the probability of all other classes is 0, and thus, the Gini index is 0. On the other hand, when each class is equally represented, with probability $P(c_i|\mathbf{D}) = \frac{1}{k}$, then the Gini index has value $\frac{k-1}{k}$. Thus, higher values of the Gini index indicate more disorder, and lower values indicate more order in terms of the class labels.

We can compute the weighted Gini index of a split point as follows:

$$G(\mathbf{D}_Y, \mathbf{D}_N) = \frac{n_Y}{n} G(\mathbf{D}_Y) + \frac{n_N}{n} G(\mathbf{D}_N)$$

where n, n_Y, and n_N denote the number of points in regions $\mathbf{D}$, $\mathbf{D}_Y$, and $\mathbf{D}_N$, respectively. The lower the Gini index value, the better the split point.

Other measures can also be used instead of entropy and Gini index to evaluate the splits. For example, the Classification And Regression Trees (CART) measure is given as

$$CART(\mathbf{D}_Y, \mathbf{D}_N) = 2\frac{n_Y}{n}\frac{n_N}{n} \sum_{i=1}^{k} \left| P(c_i|\mathbf{D}_Y) - P(c_i|\mathbf{D}_N) \right| \tag{19.7}$$

This measure thus prefers a split point that maximizes the difference between the class probability mass function for the two partitions; the higher the CART measure, the better the split point.

19.2.2 Evaluating Split Points

All of the split point evaluation measures, such as entropy [Eq. (19.3)], Gini-index [Eq. (19.6)], and CART [Eq. (19.7)], considered in the preceding section depend on the class probability mass function (PMF) for $\mathbf{D}$, namely, $P(c_i|\mathbf{D})$, and the class PMFs for the resulting partitions $\mathbf{D}_Y$ and $\mathbf{D}_N$, namely $P(c_i|\mathbf{D}_Y)$ and $P(c_i|\mathbf{D}_N)$. Note that we have to compute the class PMFs for all possible split points; scoring each of them independently would result in significant computational overhead. Instead, one can incrementally compute the PMFs as described in the following paragraphs.

Numeric Attributes

If X is a numeric attribute, we have to evaluate split points of the form $X \leq v$. Even if we restrict v to lie within the value range of attribute X, there are still an infinite number of choices for v. One reasonable approach is to consider only the midpoints between two successive distinct values for X in the sample $\mathbf{D}$. This is because split points of the form $X \leq v$, for $v \in [x_a, x_b)$, where x_a and x_b are two successive distinct values of X in $\mathbf{D}$, produce the same partitioning of $\mathbf{D}$ into $\mathbf{D}_Y$ and $\mathbf{D}_N$, and thus yield the same scores. Because there can be at most n distinct values for X, there are at most $n-1$ midpoint values to consider.

Let $\{v_1, \ldots, v_m\}$ denote the set of all such midpoints, such that $v_1 < v_2 < \cdots < v_m$. For each split point $X \leq v$, we have to estimate the class PMFs:

$$\hat{P}(c_i|\mathbf{D}_Y) = \hat{P}(c_i|X \leq v) \tag{19.8}$$

$$\hat{P}(c_i|\mathbf{D}_N) = \hat{P}(c_i|X > v) \tag{19.9}$$

Let $I()$ be an indicator variable that takes on the value 1 only when its argument is true, and is 0 otherwise. Using the Bayes theorem, we have

$$\hat{P}(c_i|X \leq v) = \frac{\hat{P}(X \leq v|c_i)\hat{P}(c_i)}{\hat{P}(X \leq v)} = \frac{\hat{P}(X \leq v|c_i)\hat{P}(c_i)}{\sum_{j=1}^{k} \hat{P}(X \leq v|c_j)\hat{P}(c_j)} \tag{19.10}$$

The prior probability for each class in $\mathbf{D}$ can be estimated as follows:

$$\hat{P}(c_i) = \frac{1}{n}\sum_{j=1}^{n} I(y_j = c_i) = \frac{n_i}{n} \tag{19.11}$$

where y_j is the class for point $\mathbf{x}_j$, $n = |\mathbf{D}|$ is the total number of points, and n_i is the number of points in $\mathbf{D}$ with class c_i. Define N_{vi} as the number of points $x_j \leq v$ with class c_i, where x_j is the value of data point $\mathbf{x}_j$ for the attribute X, given as

$$N_{vi} = \sum_{j=1}^{n} I(x_j \leq v \text{ and } y_j = c_i) \tag{19.12}$$

We can then estimate $P(X \leq v|c_i)$ as follows:

$$\hat{P}(X \leq v|c_i) = \frac{\hat{P}(X \leq v \text{ and } c_i)}{\hat{P}(c_i)} = \left(\frac{1}{n}\sum_{j=1}^{n} I(x_j \leq v \text{ and } y_j = c_i)\right) \Big/ \left(n_i/n\right)$$
$$= \frac{N_{vi}}{n_i} \tag{19.13}$$

Plugging Eqs. (19.11) and (19.13) into Eq. (19.10), and using Eq. (19.8), we have

$$\hat{P}(c_i|\mathbf{D}_Y) = \hat{P}(c_i|X \leq v) = \frac{N_{vi}}{\sum_{j=1}^{k} N_{vj}} \tag{19.14}$$

We can estimate $\hat{P}(X > v|c_i)$ as follows:

$$\hat{P}(X > v|c_i) = 1 - \hat{P}(X \leq v|c_i) = 1 - \frac{N_{vi}}{n_i} = \frac{n_i - N_{vi}}{n_i} \tag{19.15}$$

Using Eqs. (19.11) and (19.15), the class PMF $\hat{P}(c_i|\mathbf{D}_N)$ is given as

$$\hat{P}(c_i|\mathbf{D}_N) = \hat{P}(c_i|X > v) = \frac{\hat{P}(X > v|c_i)\hat{P}(c_i)}{\sum_{j=1}^{k} \hat{P}(X > v|c_j)\hat{P}(c_j)} = \frac{n_i - N_{vi}}{\sum_{j=1}^{k}(n_j - N_{vj})} \tag{19.16}$$

Algorithm 19.2 shows the split point evaluation method for numeric attributes. The for loop on line 4 iterates through all the points and computes the midpoint values v and the number of points N_{vi} from class c_i such that $x_j \leq v$. The for loop on line 12 enumerates all possible split points of the form $X \leq v$, one for each midpoint v, and scores them using the gain criterion [Eq. (19.5)]; the best split point and score are recorded and returned. Any of the other evaluation measures can also be used. However, for Gini index and CART a lower score is better unlike for gain where a higher score is better.

In terms of computational complexity, the initial sorting of values of X (line 1) takes time $O(n \log n)$. The cost of computing the midpoints and the class-specific counts N_{vi} takes time $O(nk)$ (for loop on line 4). The cost of computing the score is also bounded by $O(nk)$, because the total number of midpoints v can be at most n (for loop on line 12). The total cost of evaluating a numeric attribute is therefore $O(n \log n + nk)$. Ignoring k, because it is usually a small constant, the total cost of numeric split point evaluation is $O(n \log n)$.

Example 19.4 (Numeric Attributes). Consider the 2-dimensional Iris dataset shown in Figure 19.1a. In the initial invocation of Algorithm 19.1, the entire dataset $\mathbf{D}$ with $n = 150$ points is considered at the root of the decision tree. The task is to find the best split point considering both the attributes, X_1 (`sepal length`) and X_2 (`sepal width`). Because there are $n_1 = 50$ points labeled c_1 (`iris-setosa`), the other class c_2 has $n_2 = 100$ points. We thus have

$$\hat{P}(c_1) = 50/150 = 1/3$$
$$\hat{P}(c_2) = 100/150 = 2/3$$

ALGORITHM 19.2. Evaluate Numeric Attribute (Using Gain)

```
EVALUATE-NUMERIC-ATTRIBUTE (D, X):
1  sort D on attribute X, so that x_j ≤ x_{j+1}, ∀j = 1, ..., n−1
2  M ← ∅ // set of midpoints
3  for i = 1, ..., k do n_i ← 0
4  for j = 1, ..., n−1 do
5      if y_j = c_i then n_i ← n_i + 1 // running count for class c_i
6      if x_{j+1} ≠ x_j then
7          v ← (x_{j+1} + x_j)/2; M ← M ∪ {v} // midpoints
8          for i = 1, ..., k do
9              N_vi ← n_i // Number of points such that x_j ≤ v and y_j = c_i
10 if y_n = c_i then n_i ← n_i + 1
   // evaluate split points of the form X ≤ v
11 v* ← ∅; score* ← 0 // initialize best split point
12 forall v ∈ M do
13     for i = 1, ..., k do
14         P̂(c_i|D_Y) ← N_vi / Σ_{j=1}^{k} N_vj
15         P̂(c_i|D_N) ← (n_i − N_vi) / Σ_{j=1}^{k} (n_j − N_vj)
16     score(X ≤ v) ← Gain(D, D_Y, D_N) // use Eq. (19.5)
17     if score(X ≤ v) > score* then
18         v* ← v; score* ← score(X ≤ v)
19 return (v*, score*)
```

The entropy [Eq. (19.3)] of the dataset $\mathbf{D}$ is therefore

$$H(\mathbf{D}) = -\left(\frac{1}{3}\log_2\frac{1}{3} + \frac{2}{3}\log_2\frac{2}{3}\right) = 0.918$$

Consider split points for attribute X_1. To evaluate the splits we first compute the frequencies N_{vi} using Eq. (19.12), which are plotted in Figure 19.2 for both the classes. For example, consider the split point $X_1 \leq 5.45$. From Figure 19.2, we see that

$$N_{v1} = 45 \qquad\qquad N_{v2} = 7$$

Plugging in these values into Eq. (19.14) we get

$$\hat{P}(c_1|\mathbf{D}_Y) = \frac{N_{v1}}{N_{v1}+N_{v2}} = \frac{45}{45+7} = 0.865$$

$$\hat{P}(c_2|\mathbf{D}_Y) = \frac{N_{v2}}{N_{v1}+N_{v2}} = \frac{7}{45+7} = 0.135$$

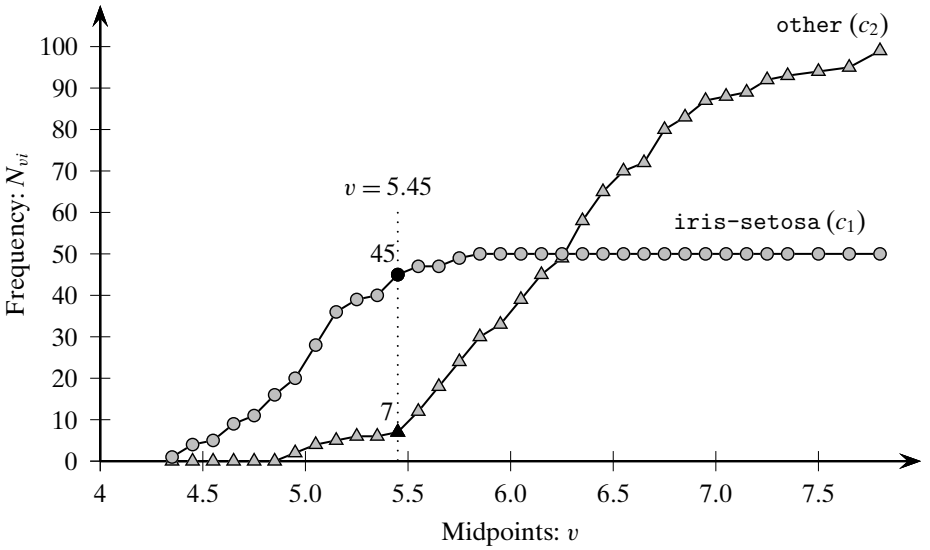


Figure 19.2. Iris: frequencies N_{vi} for classes c_1 and c_2 for attribute `sepal length`.

and using Eq. (19.16), we obtain

$$\hat{P}(c_1|\mathbf{D}_N) = \frac{n_1 - N_{v1}}{(n_1 - N_{v1}) + (n_2 - N_{v2})} = \frac{50 - 45}{(50 - 45) + (100 - 7)} = 0.051$$

$$\hat{P}(c_2|\mathbf{D}_N) = \frac{n_2 - N_{v2}}{(n_1 - N_{v1}) + (n_2 - N_{v2})} = \frac{(100 - 7)}{(50 - 45) + (100 - 7)} = 0.949$$

We can now compute the entropy of the partitions $\mathbf{D}_Y$ and $\mathbf{D}_N$ as follows:

$$H(\mathbf{D}_Y) = -(0.865 \log_2 0.865 + 0.135 \log_2 0.135) = 0.571$$

$$H(\mathbf{D}_N) = -(0.051 \log_2 0.051 + 0.949 \log_2 0.949) = 0.291$$

The entropy of the split point $X \leq 5.45$ is given via Eq. (19.4)

$$H(\mathbf{D}_Y, \mathbf{D}_N) = \frac{52}{150} H(\mathbf{D}_Y) + \frac{98}{150} H(\mathbf{D}_N) = 0.388$$

where $n_Y = |\mathbf{D}_Y| = 52$ and $n_N = |\mathbf{D}_N| = 98$. The information gain for the split point is therefore

$$Gain = H(\mathbf{D}) - H(\mathbf{D}_Y, \mathbf{D}_N) = 0.918 - 0.388 = 0.53$$

In a similar manner, we can evaluate all of the split points for both attributes X_1 and X_2. Figure 19.3 plots the gain values for the different split points for the two attributes. We can observe that $X \leq 5.45$ is the best split point and it is thus chosen as the root of the decision tree in Figure 19.1b.

The recursive tree growth process continues and yields the final decision tree and the split points as shown in Figure 19.1b. In this example, we use a leaf size threshold of 5 and a purity threshold of 0.95.

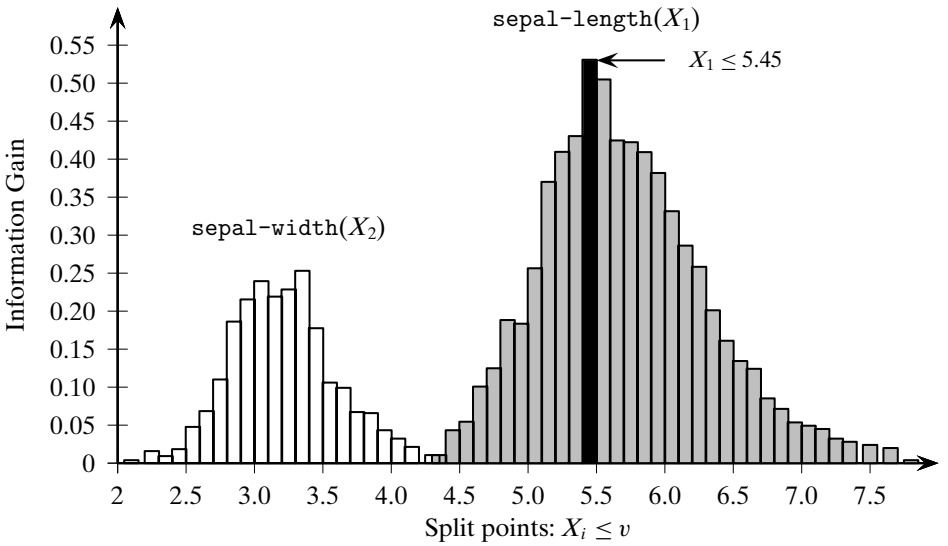


Figure 19.3. Iris: gain for different split points, for `sepal length` and `sepal width`.

Categorical Attributes

If X is a categorical attribute we evaluate split points of the form $X \in V$, where $V \subset dom(X)$ and $V \neq \emptyset$. In words, all distinct partitions of the set of values of X are considered. Because the split point $X \in V$ yields the same partition as $X \in \overline{V}$, where $\overline{V} = dom(X) \setminus V$ is the complement of V, the total number of distinct partitions is given as

$$\sum_{i=1}^{\lfloor m/2 \rfloor} \binom{m}{i} = O(2^{m-1}) \tag{19.17}$$

where m is the number of values in the domain of X, that is, $m = |dom(X)|$. The number of possible split points to consider is therefore exponential in m, which can pose problems if m is large. One simplification is to restrict V to be of size one, so that there are only m split points of the form $X_j \in \{v\}$, where $v \in dom(X_j)$.

To evaluate a given split point $X \in V$ we have to compute the following class probability mass functions:

$$P(c_i|\mathbf{D}_Y) = P(c_i|X \in V) \qquad P(c_i|\mathbf{D}_N) = P(c_i|X \notin V)$$

Making use of the Bayes theorem, we have

$$P(c_i|X \in V) = \frac{P(X \in V|c_i)P(c_i)}{P(X \in V)} = \frac{P(X \in V|c_i)P(c_i)}{\sum_{j=1}^{k} P(X \in V|c_j)P(c_j)}$$

However, note that a given point $\mathbf{x}$ can take on only one value in the domain of X, and thus the values $v \in dom(X)$ are mutually exclusive. Therefore, we have

$$P(X \in V|c_i) = \sum_{v \in V} P(X = v|c_i)$$

and we can rewrite $P(c_i|\mathbf{D}_Y)$ as

$$P(c_i|\mathbf{D}_Y) = \frac{\sum_{v\in V} P(X=v|c_i)P(c_i)}{\sum_{j=1}^{k}\sum_{v\in V} P(X=v|c_j)P(c_j)} \tag{19.18}$$

Define n_{vi} as the number of points $\mathbf{x}_j \in \mathbf{D}$, with value $x_j = v$ for attribute X and having class $y_j = c_i$:

$$n_{vi} = \sum_{j=1}^{n} I(x_j = v \text{ and } y_j = c_i) \tag{19.19}$$

The class conditional empirical PMF for X is then given as

$$\begin{aligned}\hat{P}(X=v|c_i) &= \frac{\hat{P}\big(X=v \text{ and } c_i\big)}{\hat{P}(c_i)} \\ &= \left(\frac{1}{n}\sum_{j=1}^{n} I(x_j = v \text{ and } y_j = c_i)\right) \Big/ \big(n_i/n\big) \\ &= \frac{n_{vi}}{n_i}\end{aligned} \tag{19.20}$$

Note that the class prior probabilities can be estimated using Eq. (19.11) as discussed earlier, that is, $\hat{P}(c_i) = n_i/n$. Thus, substituting Eq. (19.20) in Eq. (19.18), the class PMF for the partition $\mathbf{D}_Y$ for the split point $X \in V$ is given as

$$\hat{P}(c_i|\mathbf{D}_Y) = \frac{\sum_{v\in V} \hat{P}(X=v|c_i)\hat{P}(c_i)}{\sum_{j=1}^{k}\sum_{v\in V} \hat{P}(X=v|c_j)\hat{P}(c_j)} = \frac{\sum_{v\in V} n_{vi}}{\sum_{j=1}^{k}\sum_{v\in V} n_{vj}} \tag{19.21}$$

In a similar manner, the class PMF for the partition $\mathbf{D}_N$ is given as

$$\hat{P}(c_i|\mathbf{D}_N) = \hat{P}(c_i|X \notin V) = \frac{\sum_{v\notin V} n_{vi}}{\sum_{j=1}^{k}\sum_{v\notin V} n_{vj}} \tag{19.22}$$

Algorithm 19.3 shows the split point evaluation method for categorical attributes. The for loop on line 4 iterates through all the points and computes n_{vi}, that is, the number of points having value $v \in dom(X)$ and class c_i. The for loop on line 7 enumerates all possible split points of the form $X \in V$ for $V \subset dom(X)$, such that $|V| \leq l$, where l is a user specified parameter denoting the maximum cardinality of V. For example, to control the number of split points, we can also restrict V to be a single item, that is, $l = 1$, so that splits are of the form $V \in \{v\}$, with $v \in dom(X)$. If $l = \lfloor m/2 \rfloor$, we have to consider all possible distinct partitions V. Given a split point $X \in V$, the method scores it using information gain [Eq. (19.5)], although any of the other scoring criteria can also be used. The best split point and score are recorded and returned.

In terms of computational complexity the class-specific counts for each value n_{vi} takes $O(n)$ time (for loop on line 4). With $m = |dom(X)|$, the maximum number of partitions V is $O(2^{m-1})$, and because each split point can be evaluated in time $O(mk)$, the for loop in line 7 takes time $O(mk2^{m-1})$. The total cost for categorical attributes is therefore $O(n + mk2^{m-1})$. If we make the assumption that $2^{m-1} = O(n)$, that is, if we bound the maximum size of V to $l = O(\log n)$, then the cost of categorical splits is bounded as $O(n \log n)$, ignoring k.

ALGORITHM 19.3. Evaluate Categorical Attribute (Using Gain)

```
EVALUATE-CATEGORICAL-ATTRIBUTE (D, X, l):
for i = 1, ..., k do
    n_i ← 0
    forall v ∈ dom(X) do n_vi ← 0
for j = 1, ..., n do
    if x_j = v and y_j = c_i then n_vi ← n_vi + 1 // frequency statistics
// evaluate split points of the form X ∈ V
V* ← ∅; score* ← 0 // initialize best split point
forall V ⊂ dom(X), such that 1 ≤ |V| ≤ l do
    for i = 1, ..., k do
        P̂(c_i|D_Y) ← (Σ_{v∈V} n_vi) / (Σ_{j=1}^{k} Σ_{v∈V} n_vj)
        P̂(c_i|D_N) ← (Σ_{v∉V} n_vi) / (Σ_{j=1}^{k} Σ_{v∉V} n_vj)
    score(X ∈ V) ← Gain(D, D_Y, D_N) // use Eq. (19.5)
    if score(X ∈ V) > score* then
        V* ← V; score* ← score(X ∈ V)
return (V*, score*)
```

Example 19.5 (Categorical Attributes). Consider the 2-dimensional Iris dataset comprising the `sepal length` and `sepal width` attributes. Let us assume that `sepal length` has been discretized as shown in Table 19.1. The class frequencies n_{vi} are also shown. For instance $n_{a_1 2} = 6$ denotes the fact that there are 6 points in $\mathbf{D}$ with value $v = a_1$ and class c_2.

Consider the split point $X_1 \in \{a_1, a_3\}$. From Table 19.1 we can compute the class PMF for partition $\mathbf{D}_Y$ using Eq. (19.21)

$$\hat{P}(c_1|\mathbf{D}_Y) = \frac{n_{a_1 1} + n_{a_3 1}}{(n_{a_1 1} + n_{a_3 1}) + (n_{a_1 2} + n_{a_3 2})} = \frac{39 + 0}{(39 + 0) + (6 + 43)} = 0.443$$

$$\hat{P}(c_2|\mathbf{D}_Y) = 1 - \hat{P}(c_1|\mathbf{D}_Y) = 0.557$$

with the entropy given as

$$H(\mathbf{D}_Y) = -(0.443 \log_2 0.443 + 0.557 \log_2 0.557) = 0.991$$

To compute the class PMF for $\mathbf{D}_N$ [Eq. (19.22)], we sum up the frequencies over values $v \notin V = \{a_1, a_3\}$, that is, we sum over $v = a_2$ and $v = a_4$, as follows:

$$\hat{P}(c_1|\mathbf{D}_N) = \frac{n_{a_2 1} + n_{a_4 1}}{(n_{a_2 1} + n_{a_4 1}) + (n_{a_2 2} + n_{a_4 2})} = \frac{11 + 0}{(11 + 0) + (39 + 12)} = 0.177$$

$$\hat{P}(c_2|\mathbf{D}_N) = 1 - \hat{P}(c_1|\mathbf{D}_N) = 0.823$$

Table 19.1. Discretized `sepal length` attribute: class frequencies

Bins	v: values	Class frequencies (n_{vi})	
		c_1:`iris-setosa`	c_2:`other`
[4.3, 5.2]	Very Short (a_1)	39	6
(5.2, 6.1]	Short (a_2)	11	39
(6.1, 7.0]	Long (a_3)	0	43
(7.0, 7.9]	Very Long (a_4)	0	12

Table 19.2. Categorical split points for `sepal length`

V	Split entropy	Info. gain
$\{a_1\}$	0.509	0.410
$\{a_2\}$	0.897	0.217
$\{a_3\}$	0.711	0.207
$\{a_4\}$	0.869	0.049
$\{a_1, a_2\}$	0.632	0.286
$\{a_1, a_3\}$	0.860	0.058
$\{a_1, a_4\}$	0.667	0.251
$\{a_2, a_3\}$	0.667	0.251
$\{a_2, a_4\}$	0.860	0.058
$\{a_3, a_4\}$	0.632	0.286

with the entropy given as

$$H(\mathbf{D}_N) = -(0.177 \log_2 0.177 + 0.823 \log_2 0.823) = 0.673$$

We can see from Table 19.1 that $V \in \{a_1, a_3\}$ splits the input data $\mathbf{D}$ into partitions of size $|\mathbf{D}_Y| = 39 + 6 + 43 = 88$, and $\mathbf{D}_N = 150 - 88 = 62$. The entropy of the split is therefore given as

$$H(\mathbf{D}_Y, \mathbf{D}_N) = \frac{88}{150} H(\mathbf{D}_Y) + \frac{62}{150} H(\mathbf{D}_N) = 0.86$$

As noted in Example 19.4, the entropy of the whole dataset $\mathbf{D}$ is $H(\mathbf{D}) = 0.918$. The gain is then given as

$$Gain = H(\mathbf{D}) - H(\mathbf{D}_Y, \mathbf{D}_N) = 0.918 - 0.86 = 0.058$$

The split entropy and gain values for all the categorical split points are given in Table 19.2. We can see that $X_1 \in \{a_1\}$ is the best split point on the discretized attribute X_1.

19.2.3 Computational Complexity

To analyze the computational complexity of the decision tree method in Algorithm 19.1, we assume that the cost of evaluating all the split points for a numeric or categorical

attribute is $O(n \log n)$, where $n = |\mathbf{D}|$ is the size of the dataset. Given $\mathbf{D}$, the decision tree algorithm evaluates all d attributes, with cost $(dn \log n)$. The total cost depends on the depth of the decision tree. In the worst case, the tree can have depth n, and thus the total cost is $O(dn^2 \log n)$.

19.3 FURTHER READING

Among the earliest works on decision trees are Hunt, Marin, and Stone (1966); Breiman et al. (1984); and Quinlan (1986). The description in this chapter is largely based on the C4.5 method described in Quinlan (1993), which is an excellent reference for further details, such as how to prune decision trees to prevent overfitting, how to handle missing attribute values, and other implementation issues. A survey of methods for simplifying decision trees appears in Breslow and Aha (1997). Scalable implementation techniques are described in Mehta, Agrawal, and Rissanen (1996) and Gehrke et al. (1999).

Breiman, L., Friedman, J., Stone, C. J., and Olshen, R. (1984). *Classification and Regression Trees*. Boca Raton, FL: Chapman and Hall/CRC Press.

Breslow, L. A. and Aha, D. W. (1997). "Simplifying decision trees: A survey." *Knowledge Engineering Review*, 12 (1): 1–40.

Gehrke, J., Ganti, V., Ramakrishnan, R., and Loh, W.-Y. (1999). "BOAT-optimistic decision tree construction." *ACM SIGMOD Record*, 28 (2): 169–180.

Hunt, E. B., Marin, J., and Stone, P. J. (1966). *Experiments in Induction*. New York: Academic Press.

Mehta, M., Agrawal, R., and Rissanen, J. (1996). "SLIQ: A fast scalable classifier for data mining." *In Proceedings of the International Conference on Extending Database Technology* (pp. 18–32). New York: Springer-Verlag.

Quinlan, J. R. (1986). "Induction of decision trees." *Machine Learning*, 1 (1): 81–106.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. New York: Morgan Kaufmann.

19.4 EXERCISES

Q1. True or False:

(a) High entropy means that the partitions in classification are "pure."

(b) Multiway split of a categorical attribute generally results in more pure partitions than a binary split.

Q2. Given Table 19.3, construct a decision tree using a purity threshold of 100%. Use information gain as the split point evaluation measure. Next, classify the point (Age=27,Car=Vintage).

Q3. What is the maximum and minimum value of the CART measure [Eq. (19.7)] and under what conditions?

Table 19.3. Data for Q2: `Age` is numeric and `Car` is categorical. `Risk` gives the class label for each point: high (H) or low (L)

Point	Age	Car	**Risk**
$\mathbf{x}_1$	25	Sports	L
$\mathbf{x}_2$	20	Vintage	H
$\mathbf{x}_3$	25	Sports	L
$\mathbf{x}_4$	45	SUV	H
$\mathbf{x}_5$	20	Sports	H
$\mathbf{x}_6$	25	SUV	H

Q4. Given the dataset in Table 19.4. Answer the following questions:

Table 19.4. Data for Q4

Instance	a_1	a_2	a_3	Class
1	T	T	5.0	Y
2	T	T	7.0	Y
3	T	F	8.0	N
4	F	F	3.0	Y
5	F	T	7.0	N
6	F	T	4.0	N
7	F	F	5.0	N
8	T	F	6.0	Y
9	F	T	1.0	N

(a) Show which decision will be chosen at the root of the decision tree using information gain [Eq. (19.5)], Gini index [Eq. (19.6)], and CART [Eq. (19.7)] measures. Show all split points for all attributes.

(b) What happens to the purity if we use Instance as another attribute? Do you think this attribute should be used for a decision in the tree?

Q5. Consider Table 19.5. Let us make a nonlinear split instead of an axis parallel split, given as follows: $AB - B^2 \leq 0$. Compute the information gain of this split based on entropy (use $\log_2$, i.e., log to the base 2).

Table 19.5. Data for Q5

	A	B	Class
$\boldsymbol{x}_1$	3.5	4	H
$\boldsymbol{x}_2$	2	4	H
$\boldsymbol{x}_3$	9.1	4.5	L
$\boldsymbol{x}_4$	2	6	H
$\boldsymbol{x}_5$	1.5	7	H
$\boldsymbol{x}_6$	7	6.5	H
$\boldsymbol{x}_7$	2.1	2.5	L
$\boldsymbol{x}_8$	8	4	L

CHAPTER 20 Linear Discriminant Analysis

Given labeled data consisting of d-dimensional points $\mathbf{x}_i$ along with their classes y_i, the goal of linear discriminant analysis (LDA) is to find a vector $\mathbf{w}$ that maximizes the separation between the classes after projection onto $\mathbf{w}$. Recall from Chapter 7 that the first principal component is the vector that maximizes the projected variance of the points. The key difference between principal component analysis and LDA is that the former deals with unlabeled data and tries to maximize variance, whereas the latter deals with labeled data and tries to maximize the discrimination between the classes.

20.1 OPTIMAL LINEAR DISCRIMINANT

Let us assume that the dataset $\mathbf{D}$ consists of n labeled points $\{\mathbf{x}_i, y_i\}$, where $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{c_1, c_2, \ldots, c_k\}$. Let $\mathbf{D}_i$ denote the subset of points labeled with class c_i, i.e., $\mathbf{D}_i = \{\mathbf{x}_j | y_j = c_i\}$, and let $|\mathbf{D}_i| = n_i$ denote the number of points with class c_i. We assume that there are only $k = 2$ classes. Thus, the dataset $\mathbf{D}$ can be partitioned into $\mathbf{D}_1$ and $\mathbf{D}_2$.

Let $\mathbf{w}$ be a unit vector, that is, $\mathbf{w}^T\mathbf{w} = 1$. By Eq. (1.7), the projection of any d-dimensional point $\mathbf{x}_i$ onto the vector $\mathbf{w}$ is given as

$$\mathbf{x}_i' = \left(\frac{\mathbf{w}^T\mathbf{x}_i}{\mathbf{w}^T\mathbf{w}}\right)\mathbf{w} = \left(\mathbf{w}^T\mathbf{x}_i\right)\mathbf{w} = a_i\mathbf{w}$$

where a_i specifies the offset or coordinate of $\mathbf{x}_i'$ along the line $\mathbf{w}$:

$$a_i = \mathbf{w}^T\mathbf{x}_i$$

Thus, the set of n scalars $\{a_1, a_2, \ldots, a_n\}$ represents the mapping from $\mathbb{R}^d$ to $\mathbb{R}$, that is, from the original d-dimensional space to a 1-dimensional space (along $\mathbf{w}$).

Example 20.1. Consider Figure 20.1, which shows the 2-dimensional Iris dataset with `sepal length` and `sepal width` as the attributes, and `iris-setosa` as class c_1 (circles), and the other two Iris types as class c_2 (triangles). There are $n_1 = 50$ points in c_1 and $n_2 = 100$ points in c_2. One possible vector $\mathbf{w}$ is shown, along with the projection

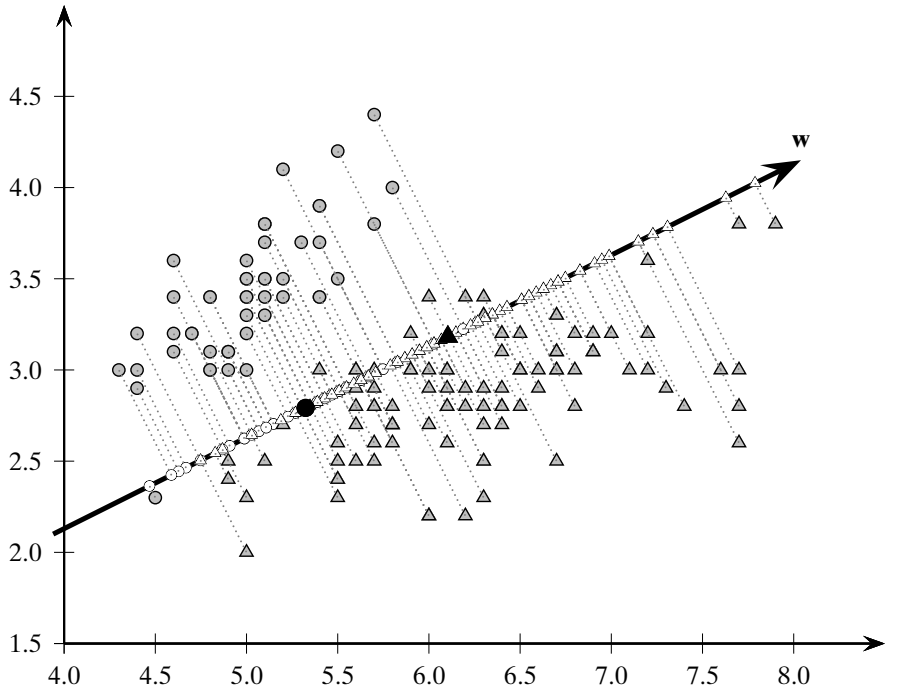


Figure 20.1. Projection onto **w**.

of all the points onto **w**. The projected means of the two classes are shown in black. Here **w** has been translated so that it passes through the mean of the entire data. One can observe that **w** is not very good in discriminating between the two classes because the projection of the points onto **w** are all mixed up in terms of their class labels. The optimal linear discriminant direction is shown in Figure 20.2.

Each point coordinate a_i has associated with it the original class label y_i, and thus we can compute, for each of the two classes, the mean of the projected points as follows:

$$\begin{aligned} m_1 &= \frac{1}{n_1} \sum_{\mathbf{x}_i \in \mathbf{D}_1} a_i \\ &= \frac{1}{n_1} \sum_{\mathbf{x}_i \in \mathbf{D}_1} \mathbf{w}^T \mathbf{x}_i \\ &= \mathbf{w}^T \left(\frac{1}{n_1} \sum_{\mathbf{x}_i \in \mathbf{D}_1} \mathbf{x}_i \right) \\ &= \mathbf{w}^T \boldsymbol{\mu}_1 \end{aligned}$$

where $\boldsymbol{\mu}_1$ is the mean of all point in $\mathbf{D}_1$. Likewise, we can obtain

$$m_2 = \mathbf{w}^T \boldsymbol{\mu}_2$$

In other words, the mean of the projected points is the same as the projection of the mean.

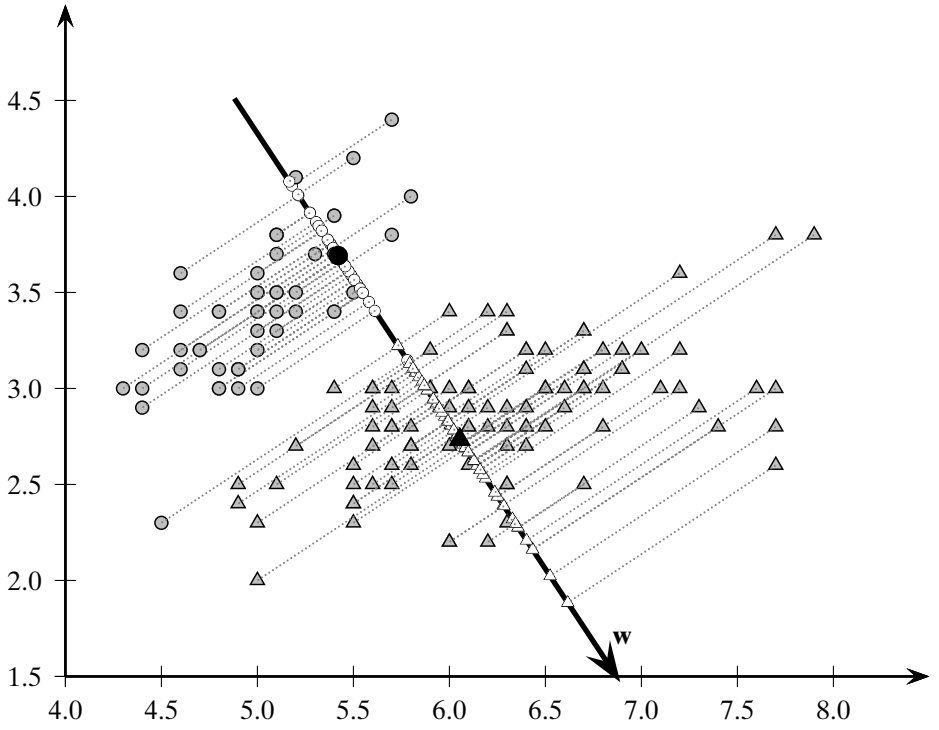


Figure 20.2. Linear discriminant direction **w**.

To maximize the separation between the classes, it seems reasonable to maximize the difference between the projected means, $|m_1 - m_2|$. However, this is not enough. For good separation, the variance of the projected points for each class should also not be too large. A large variance would lead to possible overlaps among the points of the two classes due to the large spread of the points, and thus we may fail to have a good separation. LDA maximizes the separation by ensuring that the *scatter* s_i^2 for the projected points within each class is small, where scatter is defined as

$$s_i^2 = \sum_{\mathbf{x}_j \in \mathbf{D}_i} (a_j - m_i)^2$$

Scatter is the total squared deviation from the mean, as opposed to the variance, which is the average deviation from mean. In other words

$$s_i^2 = n_i \sigma_i^2$$

where $n_i = |\mathbf{D}_i|$ is the size, and σ_i^2 is the variance, for class c_i.

We can incorporate the two LDA criteria, namely, maximizing the distance between projected means and minimizing the sum of projected scatter, into a single maximization criterion called the *Fisher LDA objective*:

$$\max_{\mathbf{w}} \; J(\mathbf{w}) = \frac{(m_1 - m_2)^2}{s_1^2 + s_2^2} \tag{20.1}$$

The goal of LDA is to find the vector $\mathbf{w}$ that maximizes $J(\mathbf{w})$, that is, the direction that maximizes the separation between the two means m_1 and m_2, and minimizes the total scatter $s_1^2 + s_2^2$ of the two classes. The vector $\mathbf{w}$ is also called the *optimal linear discriminant (LD)*. The optimization objective [Eq. (20.1)] is in the projected space. To solve it, we have to rewrite it in terms of the input data, as described next.

Note that we can rewrite $(m_1 - m_2)^2$ as follows:

$$\begin{aligned}(m_1 - m_2)^2 &= \left(\mathbf{w}^T(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)\right)^2 \\ &= \mathbf{w}^T\left((\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T\right)\mathbf{w} \\ &= \mathbf{w}^T\mathbf{B}\mathbf{w} \end{aligned} \tag{20.2}$$

where $\mathbf{B} = (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T$ is a $d \times d$ rank-one matrix called the *between-class scatter matrix*.

As for the projected scatter for class c_1, we can compute it as follows:

$$\begin{aligned} s_1^2 &= \sum_{\mathbf{x}_i \in \mathbf{D}_1} (a_i - m_1)^2 \\ &= \sum_{\mathbf{x}_i \in \mathbf{D}_1} (\mathbf{w}^T\mathbf{x}_i - \mathbf{w}^T\boldsymbol{\mu}_1)^2 \\ &= \sum_{\mathbf{x}_i \in \mathbf{D}_1} \left(\mathbf{w}^T(\mathbf{x}_i - \boldsymbol{\mu}_1)\right)^2 \\ &= \mathbf{w}^T\left(\sum_{\mathbf{x}_i \in \mathbf{D}_1} (\mathbf{x}_i - \boldsymbol{\mu}_1)(\mathbf{x}_i - \boldsymbol{\mu}_1)^T\right)\mathbf{w} \\ &= \mathbf{w}^T\mathbf{S}_1\mathbf{w} \end{aligned} \tag{20.3}$$

where $\mathbf{S}_1$ is the *scatter matrix* for $\mathbf{D}_1$. Likewise, we can obtain

$$s_2^2 = \mathbf{w}^T\mathbf{S}_2\mathbf{w} \tag{20.4}$$

Notice again that the scatter matrix is essentially the same as the covariance matrix, but instead of recording the average deviation from the mean, it records the total deviation, that is,

$$\mathbf{S}_i = n_i\boldsymbol{\Sigma}_i \tag{20.5}$$

Combining Eqs. (20.3) and (20.4), the denominator in Eq. (20.1) can be rewritten as

$$s_1^2 + s_2^2 = \mathbf{w}^T\mathbf{S}_1\mathbf{w} + \mathbf{w}^T\mathbf{S}_2\mathbf{w} = \mathbf{w}^T(\mathbf{S}_1 + \mathbf{S}_2)\mathbf{w} = \mathbf{w}^T\mathbf{S}\mathbf{w} \tag{20.6}$$

where $\mathbf{S} = \mathbf{S}_1 + \mathbf{S}_2$ denotes the *within-class scatter matrix* for the pooled data. Because both $\mathbf{S}_1$ and $\mathbf{S}_2$ are $d \times d$ symmetric positive semidefinite matrices, $\mathbf{S}$ has the same properties.

Using Eqs. (20.2) and (20.6), we write the LDA objective function [Eq. (20.1)] as follows:

$$\max_{\mathbf{w}} \; J(\mathbf{w}) = \frac{\mathbf{w}^T\mathbf{B}\mathbf{w}}{\mathbf{w}^T\mathbf{S}\mathbf{w}} \tag{20.7}$$

To solve for the best direction $\mathbf{w}$, we differentiate the objective function with respect to $\mathbf{w}$, and set the result to zero. We do not explicitly have to deal with the constraint that $\mathbf{w}^T\mathbf{w} = 1$ because in Eq. (20.7) the terms related to the magnitude of $\mathbf{w}$ cancel out in the numerator and the denominator.

Recall that if $f(x)$ and $g(x)$ are two functions then we have

$$\frac{d}{dx}\left(\frac{f(x)}{g(x)}\right) = \frac{f'(x)g(x) - g'(x)f(x)}{g(x)^2}$$

where $f'(x)$ denotes the derivative of $f(x)$. Taking the derivative of Eq. (20.7) with respect to the vector $\mathbf{w}$, and setting the result to the zero vector, gives us

$$\frac{d}{d\mathbf{w}}J(\mathbf{w}) = \frac{2\mathbf{B}\mathbf{w}(\mathbf{w}^T\mathbf{S}\mathbf{w}) - 2\mathbf{S}\mathbf{w}(\mathbf{w}^T\mathbf{B}\mathbf{w})}{(\mathbf{w}^T\mathbf{S}\mathbf{w})^2} = \mathbf{0}$$

which yields

$$\begin{aligned}
\mathbf{B}\,\mathbf{w}(\mathbf{w}^T\mathbf{S}\mathbf{w}) &= \mathbf{S}\,\mathbf{w}(\mathbf{w}^T\mathbf{B}\mathbf{w}) \\
\mathbf{B}\,\mathbf{w} &= \mathbf{S}\,\mathbf{w}\left(\frac{\mathbf{w}^T\mathbf{B}\mathbf{w}}{\mathbf{w}^T\mathbf{S}\mathbf{w}}\right) \\
\mathbf{B}\,\mathbf{w} &= J(\mathbf{w})\mathbf{S}\mathbf{w} \\
\mathbf{B}\mathbf{w} &= \lambda\mathbf{S}\mathbf{w} \qquad (20.8)
\end{aligned}$$

where $\lambda = J(\mathbf{w})$. Eq. (20.8) represents a *generalized eigenvalue problem* where λ is a generalized eigenvalue of $\mathbf{B}$ and $\mathbf{S}$; the eigenvalue λ satisfies the equation $\det(\mathbf{B} - \lambda\mathbf{S}) = 0$. Because the goal is to maximize the objective [Eq. (20.7)], $J(\mathbf{w}) = \lambda$ should be chosen to be the largest generalized eigenvalue, and $\mathbf{w}$ to be the corresponding eigenvector. If $\mathbf{S}$ is *nonsingular*, that is, if $\mathbf{S}^{-1}$ exists, then Eq. (20.8) leads to the regular eigenvalue–eigenvector equation, as

$$\begin{aligned}
\mathbf{B}\mathbf{w} &= \lambda\mathbf{S}\mathbf{w} \\
\mathbf{S}^{-1}\mathbf{B}\mathbf{w} &= \lambda\mathbf{S}^{-1}\mathbf{S}\mathbf{w} \\
(\mathbf{S}^{-1}\mathbf{B})\mathbf{w} &= \lambda\mathbf{w} \qquad (20.9)
\end{aligned}$$

Thus, if $\mathbf{S}^{-1}$ exists, then $\lambda = J(\mathbf{w})$ is an eigenvalue, and $\mathbf{w}$ is an eigenvector of the matrix $\mathbf{S}^{-1}\mathbf{B}$. To maximize $J(\mathbf{w})$ we look for the largest eigenvalue λ, and the corresponding dominant eigenvector $\mathbf{w}$ specifies the best linear discriminant vector.

Algorithm 20.1 shows the pseudo-code for linear discriminant analysis. Here, we assume that there are two classes, and that $\mathbf{S}$ is nonsingular (i.e., $\mathbf{S}^{-1}$ exists). The vector $\mathbf{1}_{n_i}$ is the vector of all ones, with the appropriate dimension for each class, i.e., $\mathbf{1}_{n_i} \in \mathbb{R}^{n_i}$ for class $i = 1, 2$. After dividing $\mathbf{D}$ into the two groups $\mathbf{D}_1$ and $\mathbf{D}_2$, LDA proceeds to compute the between-class and within-class scatter matrices, $\mathbf{B}$ and $\mathbf{S}$. The optimal LD vector is obtained as the dominant eigenvector of $\mathbf{S}^{-1}\mathbf{B}$. In terms of computational complexity, computing $\mathbf{S}$ takes $O(nd^2)$ time, and computing the dominant eigenvalue-eigenvector pair takes $O(d^3)$ time in the worst case. Thus, the total time is $O(d^3 + nd^2)$.

ALGORITHM 20.1. Linear Discriminant Analysis

```
LINEARDISCRIMINANT (D = {(x_i, y_i)}_{i=1}^n):
1 D_i ← {x_j | y_j = c_i, j = 1,...,n}, i = 1,2 // class-specific subsets
2 μ_i ← mean(D_i), i = 1,2 // class means
3 B ← (μ_1 − μ_2)(μ_1 − μ_2)^T // between-class scatter matrix
4 Z_i ← D_i − 1_{n_i} μ_i^T, i = 1,2 // center class matrices
5 S_i ← Z_i^T Z_i, i = 1,2 // class scatter matrices
6 S ← S_1 + S_2 // within-class scatter matrix
7 λ_1, w ← eigen(S^{-1}B) // compute dominant eigenvector
```

Example 20.2 (Linear Discriminant Analysis). Consider the 2-dimensional Iris data (with attributes `sepal length` and `sepal width`) shown in Example 20.1. Class c_1, corresponding to `iris-setosa`, has $n_1 = 50$ points, whereas the other class c_2 has $n_2 = 100$ points. The means for the two classes c_1 and c_2, and their difference is given as

$$\boldsymbol{\mu}_1 = \begin{pmatrix} 5.01 \\ 3.42 \end{pmatrix}^T \qquad \boldsymbol{\mu}_2 = \begin{pmatrix} 6.26 \\ 2.87 \end{pmatrix}^T \qquad \boldsymbol{\mu}_1 - \boldsymbol{\mu}_2 = \begin{pmatrix} -1.256 \\ 0.546 \end{pmatrix}^T$$

The between-class scatter matrix is

$$\mathbf{B} = (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T = \begin{pmatrix} -1.256 \\ 0.546 \end{pmatrix} \begin{pmatrix} -1.256 & 0.546 \end{pmatrix} = \begin{pmatrix} 1.587 & -0.693 \\ -0.693 & 0.303 \end{pmatrix}$$

and the within-class scatter matrix is

$$\mathbf{S}_1 = \begin{pmatrix} 6.09 & 4.91 \\ 4.91 & 7.11 \end{pmatrix} \qquad \mathbf{S}_2 = \begin{pmatrix} 43.5 & 12.09 \\ 12.09 & 10.96 \end{pmatrix} \qquad \mathbf{S} = \mathbf{S}_1 + \mathbf{S}_2 = \begin{pmatrix} 49.58 & 17.01 \\ 17.01 & 18.08 \end{pmatrix}$$

$\mathbf{S}$ is nonsingular, with its inverse given as

$$\mathbf{S}^{-1} = \begin{pmatrix} 0.0298 & -0.028 \\ -0.028 & 0.0817 \end{pmatrix}$$

Therefore, we have

$$\mathbf{S}^{-1}\mathbf{B} = \begin{pmatrix} 0.0298 & -0.028 \\ -0.028 & 0.0817 \end{pmatrix} \begin{pmatrix} 1.587 & -0.693 \\ -0.693 & 0.303 \end{pmatrix} = \begin{pmatrix} 0.066 & -0.029 \\ -0.100 & 0.044 \end{pmatrix}$$

The direction of most separation between c_1 and c_2 is the dominant eigenvector corresponding to the largest eigenvalue of the matrix $\mathbf{S}^{-1}\mathbf{B}$. The solution is

$$J(\mathbf{w}) = \lambda_1 = 0.11$$

$$\mathbf{w} = \begin{pmatrix} 0.551 \\ -0.834 \end{pmatrix}$$

Figure 20.2 plots the optimal linear discriminant direction $\mathbf{w}$, translated to the mean of the data. The projected means for the two classes are shown in black. We can

clearly observe that along $\mathbf{w}$ the circles appear together as a group, and are quite well separated from the triangles. Except for one outlying circle corresponding to the point $(4.5, 2.3)^T$, all points in c_1 are perfectly separated from points in c_2.

For the two class scenario, if $\mathbf{S}$ is nonsingular, we can directly solve for $\mathbf{w}$ without computing the eigenvalues and eigenvectors. Note that $\mathbf{B} = (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T$ is a $d \times d$ rank-one matrix, and thus $\mathbf{Bw}$ must point in the same direction as $(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$ because

$$\begin{aligned}\mathbf{Bw} &= \Big((\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T\Big)\mathbf{w} \\ &= (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)\Big((\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T\mathbf{w}\Big) \\ &= b(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)\end{aligned}$$

where $b = (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T\mathbf{w}$ is just a scalar multiplier.

We can then rewrite Eq. (20.9) as

$$\begin{aligned}\mathbf{Bw} &= \lambda\mathbf{Sw} \\ b(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) &= \lambda\mathbf{Sw} \\ \mathbf{w} &= \frac{b}{\lambda}\mathbf{S}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)\end{aligned}$$

Because $\frac{b}{\lambda}$ is just a scalar, we can solve for the best linear discriminant as

$$\mathbf{w} = \mathbf{S}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) \tag{20.10}$$

Once the direction $\mathbf{w}$ has been found we can normalize it to be a unit vector. Thus, instead of solving for the eigenvalue/eigenvector, in the two class case, we immediately obtain the direction $\mathbf{w}$ using Eq. (20.10). Intuitively, the direction that maximizes the separation between the classes can be viewed as a linear transformation (by $\mathbf{S}^{-1}$) of the vector joining the two class means $(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$.

Example 20.3. Continuing Example 20.2, we can directly compute $\mathbf{w}$ as follows:

$$\begin{aligned}\mathbf{w} &= \mathbf{S}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) \\ &= \begin{pmatrix} 0.066 & -0.029 \\ -0.100 & 0.044 \end{pmatrix}\begin{pmatrix} -1.246 \\ 0.546 \end{pmatrix} = \begin{pmatrix} -0.0527 \\ 0.0798 \end{pmatrix}\end{aligned}$$

After normalizing, we have

$$\mathbf{w} = \frac{\mathbf{w}}{\|\mathbf{w}\|} = \frac{1}{0.0956}\begin{pmatrix} -0.0527 \\ 0.0798 \end{pmatrix} = \begin{pmatrix} -0.551 \\ 0.834 \end{pmatrix}$$

Note that even though the sign is reversed for $\mathbf{w}$, compared to that in Example 20.2, they represent the same direction; only the scalar multiplier is different.

20.2 KERNEL DISCRIMINANT ANALYSIS

Kernel discriminant analysis, like linear discriminant analysis, tries to find a direction that maximizes the separation between the classes. However, it does so in *feature space* via the use of kernel functions.

Given a dataset $\mathbf{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, where $\mathbf{x}_i$ is a point in input space and $y_i \in \{c_1, c_2\}$ is the class label, let $\mathbf{D}_i = \{\mathbf{x}_j | y_j = c_i\}$ denote the data subset restricted to class c_i, and let $n_i = |\mathbf{D}_i|$. Further, let $\phi(\mathbf{x}_i)$ denote the corresponding point in feature space, and let K be a kernel function.

The goal of kernel LDA is to find the direction vector $\mathbf{w}$ in feature space that maximizes

$$\max_{\mathbf{w}} \; J(\mathbf{w}) = \frac{(m_1 - m_2)^2}{s_1^2 + s_2^2} \tag{20.11}$$

where m_1 and m_2 are the projected means, and s_1^2 and s_2^2 are projected scatter values in feature space. We first show that $\mathbf{w}$ can be expressed as a linear combination of the points in feature space, and then we transform the LDA objective in terms of the kernel matrix.

Optimal LD: Linear Combination of Feature Points

The mean for class c_i in feature space is given as

$$\boldsymbol{\mu}_i^\phi = \frac{1}{n_i} \sum_{\mathbf{x}_j \in \mathbf{D}_i} \phi(\mathbf{x}_j) \tag{20.12}$$

and the covariance matrix for class c_i in feature space is

$$\boldsymbol{\Sigma}_i^\phi = \frac{1}{n_i} \sum_{\mathbf{x}_j \in \mathbf{D}_i} \left(\phi(\mathbf{x}_j) - \boldsymbol{\mu}_i^\phi\right)\left(\phi(\mathbf{x}_j) - \boldsymbol{\mu}_i^\phi\right)^T$$

Using a derivation similar to Eq. (20.2) we obtain an expression for the between-class scatter matrix in feature space

$$\mathbf{B}_\phi = \left(\boldsymbol{\mu}_1^\phi - \boldsymbol{\mu}_2^\phi\right)\left(\boldsymbol{\mu}_1^\phi - \boldsymbol{\mu}_2^\phi\right)^T = \mathbf{d}_\phi \mathbf{d}_\phi^T \tag{20.13}$$

where $\mathbf{d}_\phi = \boldsymbol{\mu}_1^\phi - \boldsymbol{\mu}_2^\phi$ is the difference between the two class mean vectors. Likewise, using Eqs. (20.5) and (20.6) the within-class scatter matrix in feature space is given as

$$\mathbf{S}_\phi = n_1 \boldsymbol{\Sigma}_1^\phi + n_2 \boldsymbol{\Sigma}_2^\phi$$

$\mathbf{S}_\phi$ is a $d \times d$ symmetric, positive semidefinite matrix, where d is the dimensionality of the feature space. From Eq. (20.9), we conclude that the best linear discriminant vector $\mathbf{w}$ in feature space is the dominant eigenvector, which satisfies the expression

$$\left(\mathbf{S}_\phi^{-1} \mathbf{B}_\phi\right)\mathbf{w} = \lambda \mathbf{w} \tag{20.14}$$

where we assume that $\mathbf{S}_\phi$ is non-singular. Let δ_i denote the ith eigenvalue and $\mathbf{u}_i$ the ith eigenvector of $\mathbf{S}_\phi$, for $i = 1, \ldots, d$. The eigen-decomposition of $\mathbf{S}_\phi$ yields $\mathbf{S}_\phi = \mathbf{U}\boldsymbol{\Delta}\mathbf{U}^T$,

with the inverse of $\mathbf{S}_\phi$ given as $\mathbf{S}_\phi^{-1} = \mathbf{U}\mathbf{\Delta}^{-1}\mathbf{U}^T$. Here $\mathbf{U}$ is the matrix whose columns are the eigenvectors of $\mathbf{S}_\phi$ and $\mathbf{\Delta}$ is the diagonal matrix of eigenvalues of $\mathbf{S}_\phi$. The inverse $\mathbf{S}_\phi^{-1}$ can thus be expressed as the spectral sum

$$\mathbf{S}_\phi^{-1} = \sum_{r=1}^{d} \frac{1}{\delta_r} \mathbf{u}_r \mathbf{u}_r^T \tag{20.15}$$

Plugging Eqs. (20.13) and (20.15) into Eq. (20.14), we obtain

$$\lambda \mathbf{w} = \left(\sum_{r=1}^{d} \frac{1}{\delta_r} \mathbf{u}_r \mathbf{u}_r^T \right) \mathbf{d}_\phi \mathbf{d}_\phi^T \mathbf{w} = \sum_{r=1}^{d} \frac{1}{\delta_r} \left(\mathbf{u}_r (\mathbf{u}_r^T \mathbf{d}_\phi)(\mathbf{d}_\phi^T \mathbf{w}) \right) = \sum_{r=1}^{d} b_r \mathbf{u}_r$$

where $b_r = \frac{1}{\delta_r}(\mathbf{u}_r^T \mathbf{d}_\phi)(\mathbf{d}_\phi^T \mathbf{w})$ is a scalar value. Using a derivation similar to that in Eq. (7.32), the rth eigenvector of $\mathbf{S}_\phi$ can be expressed as a linear combination of the feature points, say $\mathbf{u}_r = \sum_{j=1}^{n} c_{rj} \phi(\mathbf{x}_j)$, where c_{rj} is a scalar coefficient. Thus, we can rewrite $\mathbf{w}$ as

$$\begin{aligned} \mathbf{w} &= \frac{1}{\lambda} \sum_{r=1}^{d} b_r \left(\sum_{j=1}^{n} c_{rj} \phi(\mathbf{x}_j) \right) \\ &= \sum_{j=1}^{n} \phi(\mathbf{x}_j) \left(\sum_{r=1}^{d} \frac{b_r c_{rj}}{\lambda} \right) \\ &= \sum_{j=1}^{n} a_j \phi(\mathbf{x}_j) \end{aligned}$$

where $a_j = \sum_{r=1}^{d} b_r c_{rj}/\lambda$ is a scalar value for the feature point $\phi(\mathbf{x}_j)$. Therefore, the direction vector $\mathbf{w}$ can be expressed as a linear combination of the points in feature space.

LDA Objective via Kernel Matrix

We now rewrite the kernel LDA objective [Eq. (20.11)] in terms of the kernel matrix. Projecting the mean for class c_i given in Eq. (20.12) onto the LD direction $\mathbf{w}$, we have

$$\begin{aligned} m_i = \mathbf{w}^T \boldsymbol{\mu}_i^\phi &= \left(\sum_{j=1}^{n} a_j \, \phi(\mathbf{x}_j) \right)^T \left(\frac{1}{n_i} \sum_{\mathbf{x}_k \in \mathbf{D}_i} \phi(\mathbf{x}_k) \right) \\ &= \frac{1}{n_i} \sum_{j=1}^{n} \sum_{\mathbf{x}_k \in \mathbf{D}_i} a_j \, \phi(\mathbf{x}_j)^T \phi(\mathbf{x}_k) \\ &= \frac{1}{n_i} \sum_{j=1}^{n} \sum_{\mathbf{x}_k \in \mathbf{D}_i} a_j K(\mathbf{x}_j, \mathbf{x}_k) \\ &= \mathbf{a}^T \mathbf{m}_i \end{aligned} \tag{20.16}$$

where $\mathbf{a} = (a_1, a_2, \ldots, a_n)^T$ is the weight vector, and

$$\mathbf{m}_i = \frac{1}{n_i} \begin{pmatrix} \sum_{\mathbf{x}_k \in \mathbf{D}_i} K(\mathbf{x}_1, \mathbf{x}_k) \\ \sum_{\mathbf{x}_k \in \mathbf{D}_i} K(\mathbf{x}_2, \mathbf{x}_k) \\ \vdots \\ \sum_{\mathbf{x}_k \in \mathbf{D}_i} K(\mathbf{x}_n, \mathbf{x}_k) \end{pmatrix} = \frac{1}{n_i} \mathbf{K}^{c_i} \mathbf{1}_{n_i} \tag{20.17}$$

where $\mathbf{K}^{c_i}$ is the $n \times n_i$ subset of the kernel matrix, restricted to columns belonging to points only in $\mathbf{D}_i$, and $\mathbf{1}_{n_i}$ is the n_i-dimensional vector all of whose entries are one. The n-length vector $\mathbf{m}_i$ thus stores for each point in $\mathbf{D}$ its average kernel value with respect to the points in $\mathbf{D}_i$.

We can rewrite the separation between the projected means in feature space as follows:

$$\begin{aligned} (m_1 - m_2)^2 &= \left(\mathbf{w}^T \boldsymbol{\mu}_1^\phi - \mathbf{w}^T \boldsymbol{\mu}_2^\phi\right)^2 \\ &= \left(\mathbf{a}^T \mathbf{m}_1 - \mathbf{a}^T \mathbf{m}_2\right)^2 \\ &= \mathbf{a}^T (\mathbf{m}_1 - \mathbf{m}_2)(\mathbf{m}_1 - \mathbf{m}_2)^T \mathbf{a} \\ &= \mathbf{a}^T \mathbf{M} \mathbf{a} \end{aligned} \tag{20.18}$$

where $\mathbf{M} = (\mathbf{m}_1 - \mathbf{m}_2)(\mathbf{m}_1 - \mathbf{m}_2)^T$ is the between-class scatter matrix.

We can also compute the projected scatter for each class, s_1^2 and s_2^2, purely in terms of the kernel function, as

$$\begin{aligned} s_1^2 &= \sum_{\mathbf{x}_i \in \mathbf{D}_1} \left\| \mathbf{w}^T \phi(\mathbf{x}_i) - \mathbf{w}^T \boldsymbol{\mu}_1^\phi \right\|^2 \\ &= \sum_{\mathbf{x}_i \in \mathbf{D}_1} \left\| \mathbf{w}^T \phi(\mathbf{x}_i) \right\|^2 - 2 \sum_{\mathbf{x}_i \in \mathbf{D}_1} \mathbf{w}^T \phi(\mathbf{x}_i) \cdot \mathbf{w}^T \boldsymbol{\mu}_1^\phi + \sum_{\mathbf{x}_i \in \mathbf{D}_1} \left\| \mathbf{w}^T \boldsymbol{\mu}_1^\phi \right\|^2 \\ &= \left(\sum_{\mathbf{x}_i \in \mathbf{D}_1} \left\| \sum_{j=1}^{n} a_j \phi(\mathbf{x}_j)^T \phi(\mathbf{x}_i) \right\|^2 \right) - 2 \cdot n_1 \cdot \left\| \mathbf{w}^T \boldsymbol{\mu}_1^\phi \right\|^2 + n_1 \cdot \left\| \mathbf{w}^T \boldsymbol{\mu}_1^\phi \right\|^2 \\ &= \left(\sum_{\mathbf{x}_i \in \mathbf{D}_1} \mathbf{a}^T \mathbf{K}_i \mathbf{K}_i^T \mathbf{a} \right) - n_1 \cdot \mathbf{a}^T \mathbf{m}_1 \mathbf{m}_1^T \mathbf{a} \qquad \text{by using Eq. (20.16)} \\ &= \mathbf{a}^T \left(\left(\sum_{\mathbf{x}_i \in \mathbf{D}_1} \mathbf{K}_i \mathbf{K}_i^T \right) - n_1 \mathbf{m}_1 \mathbf{m}_1^T \right) \mathbf{a} \\ &= \mathbf{a}^T \mathbf{N}_1 \mathbf{a} \end{aligned}$$

where $\mathbf{K}_i$ is the ith column of the kernel matrix, and $\mathbf{N}_1$ is the class scatter matrix for c_1. Let $K(\mathbf{x}_i, \mathbf{x}_j) = K_{ij}$. We can express $\mathbf{N}_1$ more compactly in matrix notation as follows:

$$\begin{aligned} \mathbf{N}_1 &= \left(\sum_{\mathbf{x}_i \in \mathbf{D}_1} \mathbf{K}_i \mathbf{K}_i^T \right) - n_1 \mathbf{m}_1 \mathbf{m}_1^T \\ &= (\mathbf{K}^{c_1}) \left(\mathbf{I}_{n_1} - \frac{1}{n_1} \mathbf{1}_{n_1 \times n_1} \right) (\mathbf{K}^{c_1})^T \end{aligned} \tag{20.19}$$

where $\mathbf{I}_{n_1}$ is the $n_1 \times n_1$ identity matrix and $\mathbf{1}_{n_1 \times n_1}$ is the $n_1 \times n_1$ matrix, all of whose entries are 1's.

In a similar manner we get $s_2^2 = \mathbf{a}^T \mathbf{N}_2 \mathbf{a}$, where

$$\mathbf{N}_2 = (\mathbf{K}^{c_2}) \left(\mathbf{I}_{n_2} - \frac{1}{n_2} \mathbf{1}_{n_2 \times n_2} \right) (\mathbf{K}^{c_2})^T$$

where $\mathbf{I}_{n_2}$ is the $n_2 \times n_2$ identity matrix and $\mathbf{1}_{n_2 \times n_2}$ is the $n_2 \times n_2$ matrix, all of whose entries are 1's.

The sum of projected scatter values is then given as

$$s_1^2 + s_2^2 = \mathbf{a}^T (\mathbf{N}_1 + \mathbf{N}_2) \mathbf{a} = \mathbf{a}^T \mathbf{N} \mathbf{a} \tag{20.20}$$

where $\mathbf{N}$ is the $n \times n$ within-class scatter matrix.

Substituting Eqs. (20.18) and (20.20) in Eq. (20.11), we obtain the kernel LDA maximization condition

$$\max_{\mathbf{w}} J(\mathbf{w}) = \max_{\mathbf{a}} J(\mathbf{a}) = \frac{\mathbf{a}^T \mathbf{M} \mathbf{a}}{\mathbf{a}^T \mathbf{N} \mathbf{a}}$$

Notice how all the terms in the expression above involve only kernel functions. The weight vector $\mathbf{a}$ is the eigenvector corresponding to the largest eigenvalue of the generalized eigenvalue problem:

$$\mathbf{M}\mathbf{a} = \lambda_1 \mathbf{N}\mathbf{a} \tag{20.21}$$

If $\mathbf{N}$ is nonsingular, $\mathbf{a}$ is the dominant eigenvector corresponding to the largest eigenvalue for the system

$$(\mathbf{N}^{-1}\mathbf{M})\mathbf{a} = \lambda_1 \mathbf{a}$$

As in the case of linear discriminant analysis [Eq. (20.10)], when there are only two classes we do not have to solve for the eigenvector because $\mathbf{a}$ can be obtained directly:

$$\mathbf{a} = \mathbf{N}^{-1}(\mathbf{m}_1 - \mathbf{m}_2)$$

Once $\mathbf{a}$ has been obtained, we can normalize $\mathbf{w}$ to be a unit vector by ensuring that

$$\mathbf{w}^T \mathbf{w} = 1, \text{ which implies that}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{n} a_i a_j \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) = 1, \text{ or}$$

$$\mathbf{a}^T \mathbf{K} \mathbf{a} = 1$$

Put differently, we can ensure that $\mathbf{w}$ is a unit vector if we scale $\mathbf{a}$ by $\frac{1}{\sqrt{\mathbf{a}^T \mathbf{K} \mathbf{a}}}$.

Finally, we can project any point $\mathbf{x}$ onto the discriminant direction, as follows:

$$\mathbf{w}^T \phi(\mathbf{x}) = \sum_{j=1}^{n} a_j \phi(\mathbf{x}_j)^T \phi(\mathbf{x}) = \sum_{j=1}^{n} a_j K(\mathbf{x}_j, \mathbf{x}) \tag{20.22}$$

Algorithm 20.2 shows the pseudo-code for kernel discriminant analysis. The method proceeds by computing the $n \times n$ kernel matrix $\mathbf{K}$, and the $n \times n_i$ class

ALGORITHM 20.2. Kernel Discriminant Analysis

KERNELDISCRIMINANT $(\mathbf{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n, K)$:

1 $\mathbf{K} \leftarrow \left\{K(\mathbf{x}_i, \mathbf{x}_j)\right\}_{i,j=1,\ldots,n}$ `// compute` $n \times n$ `kernel matrix`

2 $\mathbf{K}^{c_i} \leftarrow \left\{\mathbf{K}(j,k) \mid y_k = c_i, 1 \le j,k \le n\right\}, i = 1,2$ `// class kernel matrix`

3 $\mathbf{m}_i \leftarrow \frac{1}{n_i}\mathbf{K}^{c_i}\mathbf{1}_{n_i}, i = 1,2$ `// class means`

4 $\mathbf{M} \leftarrow (\mathbf{m}_1 - \mathbf{m}_2)(\mathbf{m}_1 - \mathbf{m}_2)^T$ `// between-class scatter matrix`

5 $\mathbf{N}_i \leftarrow \mathbf{K}^{c_i}(\mathbf{I}_{n_i} - \frac{1}{n_i}\mathbf{1}_{n_i \times n_i})(\mathbf{K}^{c_i})^T, i = 1,2$ `// class scatter matrices`

6 $\mathbf{N} \leftarrow \mathbf{N}_1 + \mathbf{N}_2$ `// within-class scatter matrix`

7 $\lambda_1, \mathbf{a} \leftarrow \text{eigen}(\mathbf{N}^{-1}\mathbf{M})$ `// compute weight vector`

8 $\mathbf{a} \leftarrow \frac{\mathbf{a}}{\sqrt{\mathbf{a}^T\mathbf{K}\mathbf{a}}}$ `// normalize` $\mathbf{w}$ `to be unit vector`

specific kernel matrices $\mathbf{K}^{c_i}$ for each class c_i. After computing the between-class and within-class scatter matrices $\mathbf{M}$ and $\mathbf{N}$, the weight vector $\mathbf{a}$ is obtained as the dominant eigenvector of $\mathbf{N}^{-1}\mathbf{M}$. The last step scales $\mathbf{a}$ so that $\mathbf{w}$ will be normalized to be unit length. The complexity of kernel discriminant analysis is $O(n^3)$, with the dominant steps being the computation of $\mathbf{N}$ and solving for the dominant eigenvector of $\mathbf{N}^{-1}\mathbf{M}$, both of which take $O(n^3)$ time.

Example 20.4 (Kernel Discriminant Analysis). Consider the 2-dimensional Iris dataset comprising the `sepal length` and `sepal width` attributes. Figure 20.3a shows the points projected onto the first two principal components. The points have been divided into two classes: c_1 (circles) corresponds to *iris-virginica* and c_2 (triangles) corresponds to the other two Iris types. Here $n_1 = 50$ and $n_2 = 100$, with a total of $n = 150$ points.

Because c_1 is surrounded by points in c_2 a good linear discriminant will not be found. Instead, we apply kernel discriminant analysis using the homogeneous quadratic kernel

$$K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T\mathbf{x}_j)^2$$

Solving for $\mathbf{a}$ via Eq. (20.21) yields

$$\lambda_1 = 0.0511$$

However, we do not show $\mathbf{a}$ because it lies in $\mathbb{R}^{150}$. Figure 20.3a shows the contours of constant projections onto the best kernel discriminant. The contours are obtained by solving Eq. (20.22), that is, by solving $\mathbf{w}^T\phi(\mathbf{x}) = \sum_{j=1}^n a_j K(\mathbf{x}_j, \mathbf{x}) = c$ for different values of the scalars c. The contours are hyperbolic, and thus form pairs starting from the center. For instance, the first curve on the left and right of the origin $(0,0)^T$ forms the same contour, that is, points along both the curves have the same value when projected onto $\mathbf{w}$. We can see that contours or pairs of curves starting with the fourth curve (on the left and right) from the center all relate to class c_2, whereas the first three contours deal mainly with class c_1, indicating good discrimination with the homogeneous quadratic kernel.

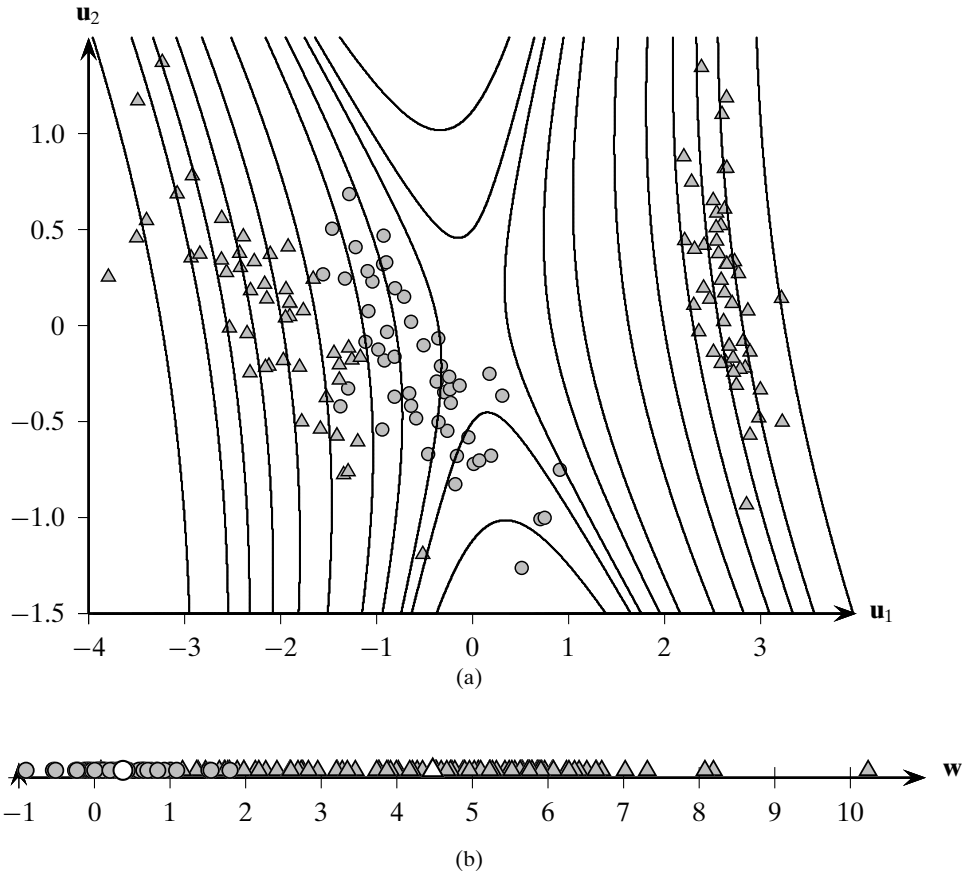


Figure 20.3. Kernel discriminant analysis: quadratic homogeneous kernel.

A better picture emerges when we plot the coordinates of all the points $\mathbf{x}_i \in \mathbf{D}$ when projected onto $\mathbf{w}$, as shown in Figure 20.3b. We can observe that $\mathbf{w}$ is able to separate the two classes reasonably well; all the circles (c_1) are concentrated on the left, whereas the triangles (c_2) are spread out on the right. The projected means are shown in white. The scatters and means for both classes after projection are as follows:

$$m_1 = 0.338 \qquad m_2 = 4.476$$
$$s_1^2 = 13.862 \qquad s_2^2 = 320.934$$

The value of $J(\mathbf{w})$ is given as

$$J(\mathbf{w}) = \frac{(m_1 - m_2)^2}{s_1^2 + s_2^2} = \frac{(0.338 - 4.476)^2}{13.862 + 320.934} = \frac{17.123}{334.796} = 0.0511$$

which, as expected, matches $\lambda_1 = 0.0511$ from above.

In general, it is not desirable or possible to obtain an explicit discriminant vector $\mathbf{w}$, since it lies in feature space. However, because each point $\mathbf{x} = (x_1, x_2)^T \in \mathbb{R}^2$ in

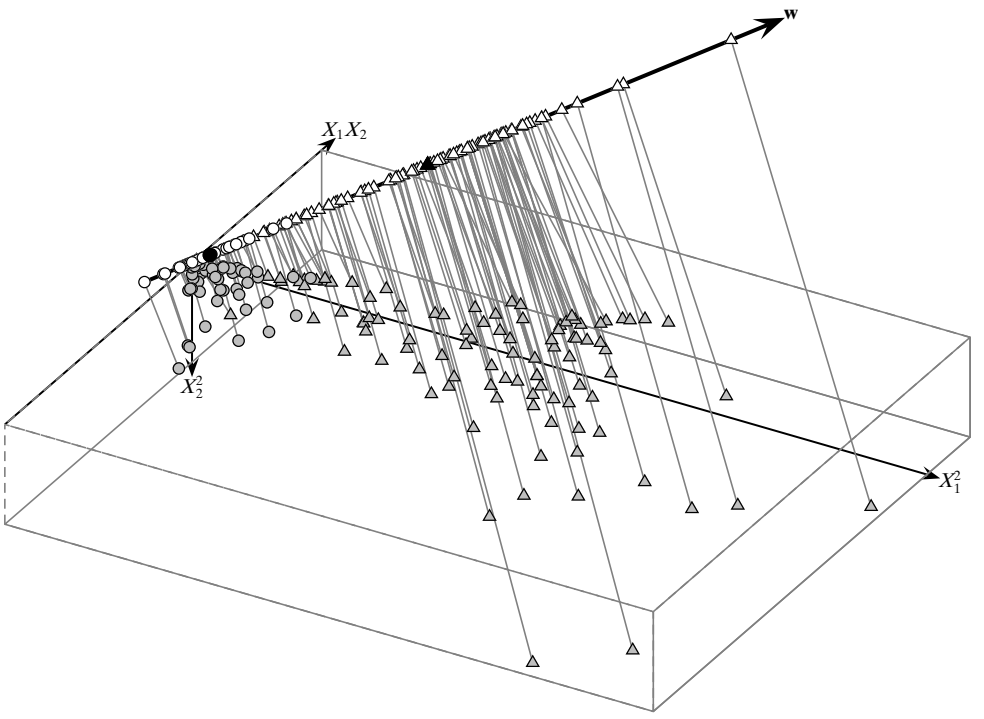


Figure 20.4. Homogeneous quadratic kernel feature space.

input space is mapped to the point $\phi(\mathbf{x}) = (\sqrt{2}x_1x_2, x_1^2, x_2^2)^T \in \mathbb{R}^3$ in feature space via the homogeneous quadratic kernel, for our example it is possible to visualize the feature space, as illustrated in Figure 20.4. The projection of each point $\phi(\mathbf{x}_i)$ onto the discriminant vector $\mathbf{w}$ is also shown, where

$$\mathbf{w} = 0.511x_1x_2 + 0.761x_1^2 - 0.4x_2^2$$

The projections onto $\mathbf{w}$ are identical to those shown in Figure 20.3b.

20.3 FURTHER READING

Linear discriminant analysis was introduced in Fisher (1936). Its extension to kernel discriminant analysis was proposed in Mika et al. (1999). The 2-class LDA approach can be generalized to $k > 2$ classes by finding the optimal $(k-1)$-dimensional subspace projection that best discriminates between the k classes; see Duda, Hart, and Stork (2012) for details.

Duda, R. O., Hart, P. E., and Stork, D. G. (2012). *Pattern Classification*. New York: Wiley-Interscience.

Fisher, R. A. (1936). "The use of multiple measurements in taxonomic problems." *Annals of Eugenics*, 7 (2): 179–188.

Mika, S., Ratsch, G., Weston, J., Scholkopf, B., and Mullers, K. (1999). "Fisher discriminant analysis with kernels." *In Proceedings of the IEEE Neural Networks for Signal Processing Workshop,* IEEE, pp. 41–48.

20.4 EXERCISES

Q1. Consider the data shown in Table 20.1. Answer the following questions:

(a) Compute $\boldsymbol{\mu}_{+1}$ and $\boldsymbol{\mu}_{-1}$, and $\mathbf{S}_B$, the between-class scatter matrix.

(b) Compute $\mathbf{S}_{+1}$ and $\mathbf{S}_{-1}$, and $\mathbf{S}_W$, the within-class scatter matrix.

(c) Find the best direction $\mathbf{w}$ that discriminates between the classes. Use the fact that the inverse of the matrix $\mathbf{A} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ is given as $\mathbf{A}^{-1} = \frac{1}{\det(\mathbf{A})} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$.

(d) Having found the direction $\mathbf{w}$, find the point on $\mathbf{w}$ that best separates the two classes.

Table 20.1. Dataset for Q1

i	$\mathbf{x}_i$	y_i
$\mathbf{x}_1$	(4,2.9)	1
$\mathbf{x}_2$	(3.5,4)	1
$\mathbf{x}_3$	(2.5,1)	−1
$\mathbf{x}_4$	(2,2.1)	−1

Q2. Given the labeled points (from two classes) shown in Figure 20.5, and given that the inverse of the within-class scatter matrix is

$$\begin{pmatrix} 0.056 & -0.029 \\ -0.029 & 0.052 \end{pmatrix}$$

Find the best linear discriminant line $\mathbf{w}$, and sketch it.

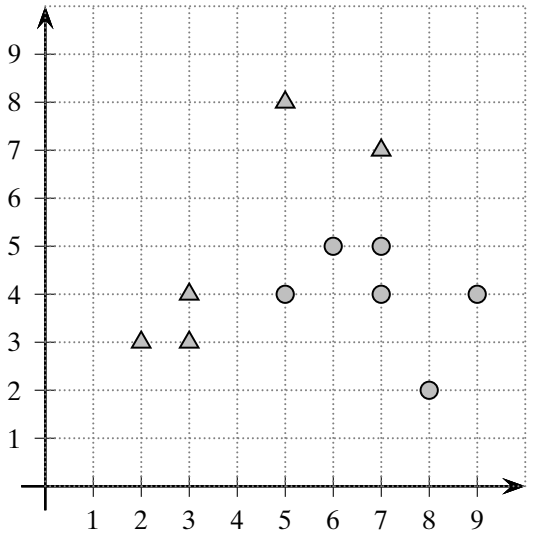


Figure 20.5. Dataset for Q2.

Q3. Maximize the objective in Eq. (20.7) by explicitly considering the constraint $\mathbf{w}^T\mathbf{w} = 1$, that is, by using a Lagrange multiplier for that constraint.

Q4. Prove the equality in Eq. (20.19). That is, show that

$$\mathbf{N}_1 = \left(\sum_{\mathbf{x}_i \in \mathbf{D}_1} \mathbf{K}_i \mathbf{K}_i^T \right) - n_1 \mathbf{m}_1 \mathbf{m}_1^T = (\mathbf{K}^{c_1}) \left(\mathbf{I}_{n_1} - \frac{1}{n_1} \mathbf{1}_{n_1 \times n_1} \right) (\mathbf{K}^{c_1})^T$$

CHAPTER 21 Support Vector Machines

In this chapter we describe Support Vector Machines (SVMs), a classification method based on maximum margin linear discriminants, that is, the goal is to find the optimal hyperplane that maximizes the gap or margin between the classes. Further, we can use the kernel trick to find the optimal nonlinear decision boundary between classes, which corresponds to a hyperplane in some high-dimensional "nonlinear" space.

21.1 SUPPORT VECTORS AND MARGINS

Let $\mathbf{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ be a classification dataset, with n points in a d-dimensional space. Further, let us assume that there are only two class labels, that is, $y_i \in \{+1, -1\}$, denoting the positive and negative classes.

Hyperplanes

A hyperplane in d dimensions is given as the set of all points $\mathbf{x} \in \mathbb{R}^d$ that satisfy the equation $h(\mathbf{x}) = 0$, where $h(\mathbf{x})$ is the *hyperplane function*, defined as follows:

$$\begin{aligned} h(\mathbf{x}) &= \mathbf{w}^T\mathbf{x} + b \\ &= w_1x_1 + w_2x_2 + \cdots + w_dx_d + b \end{aligned} \tag{21.1}$$

Here, $\mathbf{w}$ is a d dimensional *weight vector* and b is a scalar, called the *bias*. For points that lie on the hyperplane, we have

$$h(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b = 0 \tag{21.2}$$

The hyperplane is thus defined as the set of all points such that $\mathbf{w}^T\mathbf{x} = -b$. To see the role played by b, assuming that $w_1 \neq 0$, and setting $x_i = 0$ for all $i > 1$, we can obtain the offset where the hyperplane intersects the first axis, as by Eq. (21.2), we have

$$w_1x_1 = -b \quad \text{or} \quad x_1 = \frac{-b}{w_1}$$

In other words, the point $(\frac{-b}{w_1}, 0, \ldots, 0)$ lies on the hyperplane. In a similar manner, we can obtain the offset where the hyperplane intersects each of the axes, which is given as $\frac{-b}{w_i}$ (provided $w_i \neq 0$).

Separating Hyperplane

A hyperplane splits the original d-dimensional space into two *half-spaces*. A dataset is said to be *linearly separable* if each half-space has points only from a single class. If the input dataset is linearly separable, then we can find a *separating* hyperplane $h(\mathbf{x}) = 0$, such that for all points labeled $y_i = -1$, we have $h(\mathbf{x}_i) < 0$, and for all points labeled $y_i = +1$, we have $h(\mathbf{x}_i) > 0$. In fact, the hyperplane function $h(\mathbf{x})$ serves as a linear classifier or a linear discriminant, which predicts the class y for any given point $\mathbf{x}$, according to the decision rule:

$$y = \begin{cases} +1 & \text{if } h(\mathbf{x}) > 0 \\ -1 & \text{if } h(\mathbf{x}) < 0 \end{cases} \tag{21.3}$$

Let $\mathbf{a}_1$ and $\mathbf{a}_2$ be two arbitrary points that lie on the hyperplane. From Eq. (21.2) we have

$$h(\mathbf{a}_1) = \mathbf{w}^T\mathbf{a}_1 + b = 0$$
$$h(\mathbf{a}_2) = \mathbf{w}^T\mathbf{a}_2 + b = 0$$

Subtracting one from the other we obtain

$$\mathbf{w}^T(\mathbf{a}_1 - \mathbf{a}_2) = 0$$

This means that the weight vector $\mathbf{w}$ is orthogonal to the hyperplane because it is orthogonal to any arbitrary vector $(\mathbf{a}_1 - \mathbf{a}_2)$ on the hyperplane. In other words, the weight vector $\mathbf{w}$ specifies the direction that is normal to the hyperplane, which fixes the orientation of the hyperplane, whereas the bias b fixes the offset of the hyperplane in the d-dimensional space. Because both $\mathbf{w}$ and $-\mathbf{w}$ are normal to the hyperplane, we remove this ambiguity by requiring that $h(\mathbf{x}_i) > 0$ when $y_i = 1$, and $h(\mathbf{x}_i) < 0$ when $y_i = -1$.

Distance of a Point to the Hyperplane

Consider a point $\mathbf{x} \in \mathbb{R}^d$, such that $\mathbf{x}$ does not lie on the hyperplane. Let $\mathbf{x}_p$ be the orthogonal projection of $\mathbf{x}$ on the hyperplane, and let $\mathbf{r} = \mathbf{x} - \mathbf{x}_p$, then as shown in Figure 21.1 we can write $\mathbf{x}$ as

$$\mathbf{x} = \mathbf{x}_p + \mathbf{r}$$
$$\mathbf{x} = \mathbf{x}_p + r\frac{\mathbf{w}}{\|\mathbf{w}\|} \tag{21.4}$$

where r is the *directed distance* of the point $\mathbf{x}$ from $\mathbf{x}_p$, that is, r gives the offset of $\mathbf{x}$ from $\mathbf{x}_p$ in terms of the unit weight vector $\frac{\mathbf{w}}{\|\mathbf{w}\|}$. The offset r is positive if $\mathbf{r}$ is in the same direction as $\mathbf{w}$, and r is negative if $\mathbf{r}$ is in a direction opposite to $\mathbf{w}$.

Plugging Eq. (21.4) into the hyperplane function [Eq. (21.1)], we get

$$h(\mathbf{x}) = h\left(\mathbf{x}_p + r\frac{\mathbf{w}}{\|\mathbf{w}\|}\right)$$
$$= \mathbf{w}^T\left(\mathbf{x}_p + r\frac{\mathbf{w}}{\|\mathbf{w}\|}\right) + b$$

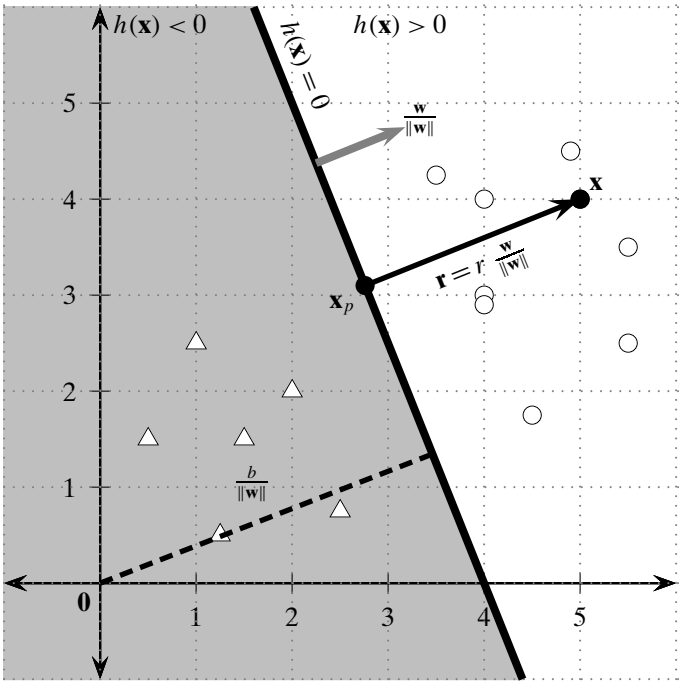


Figure 21.1. Geometry of a separating hyperplane in 2D. Points labeled $+1$ are shown as circles, and those labeled -1 are shown as triangles. The hyperplane $h(\mathbf{x}) = 0$ divides the space into two half-spaces. The shaded region comprises all points $\mathbf{x}$ satisfying $h(\mathbf{x}) < 0$, whereas the unshaded region consists of all points satisfying $h(\mathbf{x}) > 0$. The unit weight vector $\frac{\mathbf{w}}{\|\mathbf{w}\|}$ (in gray) is orthogonal to the hyperplane. The directed distance of the origin to the hyperplane is $\frac{b}{\|\mathbf{w}\|}$.

$$= \underbrace{\mathbf{w}^T\mathbf{x}_p + b}_{h(\mathbf{x}_p)} + r\frac{\mathbf{w}^T\mathbf{w}}{\|\mathbf{w}\|}$$

$$= \underbrace{h(\mathbf{x}_p)}_{0} + r\|\mathbf{w}\|$$

$$= r\|\mathbf{w}\|$$

The last step follows from the fact that $h(\mathbf{x}_p) = 0$ because $\mathbf{x}_p$ lies on the hyperplane. Using the result above, we obtain an expression for the directed distance of a point to the hyperplane:

$$r = \frac{h(\mathbf{x})}{\|\mathbf{w}\|}$$

To obtain distance, which must be non-negative, we can conveniently multiply r by the class label y of the point because when $h(\mathbf{x}) < 0$, the class is -1, and when $h(\mathbf{x}) > 0$ the class is $+1$. The distance of a point $\mathbf{x}$ from the hyperplane $h(\mathbf{x}) = 0$ is thus given as

$$\delta = y\, r = \frac{y\, h(\mathbf{x})}{\|\mathbf{w}\|} \tag{21.5}$$

In particular, for the origin $\mathbf{x} = \mathbf{0}$, the directed distance is

$$r = \frac{h(\mathbf{0})}{\|\mathbf{w}\|} = \frac{\mathbf{w}^T\mathbf{0} + b}{\|\mathbf{w}\|} = \frac{b}{\|\mathbf{w}\|}$$

as illustrated in Figure 21.1.

Example 21.1. Consider the example shown in Figure 21.1. In this 2-dimensional example, the hyperplane is just a line, defined as the set of all points $\mathbf{x} = (x_1, x_2)^T$ that satisfy the following equation:

$$h(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b = w_1x_1 + w_2x_2 + b = 0$$

Rearranging the terms we get

$$x_2 = -\frac{w_1}{w_2}x_1 - \frac{b}{w_2}$$

where $-\frac{w_1}{w_2}$ is the slope of the line, and $-\frac{b}{w_2}$ is the intercept along the second dimension.

Consider any two points on the hyperplane, say $\mathbf{p} = (p_1, p_2) = (4, 0)$, and $\mathbf{q} = (q_1, q_2) = (2, 5)$. The slope is given as

$$-\frac{w_1}{w_2} = \frac{q_2 - p_2}{q_1 - p_1} = \frac{5 - 0}{2 - 4} = -\frac{5}{2}$$

which implies that $w_1 = 5$ and $w_2 = 2$. Given any point on the hyperplane, say $(4, 0)$, we can compute the offset b directly as follows:

$$b = -5x_1 - 2x_2 = -5 \cdot 4 - 2 \cdot 0 = -20$$

Thus, $\mathbf{w} = \begin{pmatrix} 5 \\ 2 \end{pmatrix}$ is the weight vector, and $b = -20$ is the bias, and the equation of the hyperplane is given as

$$h(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b = \begin{pmatrix} 5 & 2 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} - 20 = 0$$

One can verify that the distance of the origin $\mathbf{0}$ from the hyperplane is given as

$$\delta = y\, r = -1\, r = \frac{-b}{\|\mathbf{w}\|} = \frac{-(-20)}{\sqrt{29}} = 3.71$$

Margin and Support Vectors of a Hyperplane

Given a training dataset of labeled points, $\mathbf{D} = \{\mathbf{x}_i, y_i\}_{i=1}^n$ with $y_i \in \{+1, -1\}$, and given a separating hyperplane $h(\mathbf{x}) = 0$, for each point $\mathbf{x}_i$ we can find its distance to the hyperplane by Eq. (21.5):

$$\delta_i = \frac{y_i\, h(\mathbf{x}_i)}{\|\mathbf{w}\|} = \frac{y_i(\mathbf{w}^T\mathbf{x}_i + b)}{\|\mathbf{w}\|}$$

Over all the n points, we define the *margin* of the linear classifier as the minimum distance of a point from the separating hyperplane, given as

$$\delta^* = \min_{\mathbf{x}_i} \left\{ \frac{y_i(\mathbf{w}^T\mathbf{x}_i + b)}{\|\mathbf{w}\|} \right\} \tag{21.6}$$

Note that $\delta^* \neq 0$, since $h(\mathbf{x})$ is assumed to be a separating hyperplane, and Eq. (21.3) must be satisfied.

All the points (or vectors) that achieve this minimum distance are called *support vectors* for the hyperplane. In other words, a support vector $\mathbf{x}^*$ is a point that lies precisely on the margin of the classifier, and thus satisfies the condition

$$\delta^* = \frac{y^*(\mathbf{w}^T\mathbf{x}^* + b)}{\|\mathbf{w}\|}$$

where y^* is the class label for $\mathbf{x}^*$. The numerator $y^*(\mathbf{w}^T\mathbf{x}^* + b)$ gives the absolute distance of the support vector to the hyperplane, whereas the denominator $\|\mathbf{w}\|$ makes it a relative distance in terms of $\mathbf{w}$.

Canonical Hyperplane

Consider the equation of the hyperplane [Eq. (21.2)]. Multiplying on both sides by some scalar s yields an equivalent hyperplane:

$$s\, h(\mathbf{x}) = s\, \mathbf{w}^T\mathbf{x} + s\, b = (s\mathbf{w})^T\mathbf{x} + (sb) = 0$$

To obtain the unique or *canonical* hyperplane, we choose the scalar s such that the absolute distance of a support vector from the hyperplane is 1. That is,

$$s y^*(\mathbf{w}^T\mathbf{x}^* + b) = 1$$

which implies

$$s = \frac{1}{y^*(\mathbf{w}^T\mathbf{x}^* + b)} = \frac{1}{y^* h(\mathbf{x}^*)} \tag{21.7}$$

Henceforth, we will assume that any separating hyperplane is canonical. That is, it has already been suitably rescaled so that $y^* h(\mathbf{x}^*) = 1$ for a support vector $\mathbf{x}^*$, and the margin is given as

$$\delta^* = \frac{y^* h(\mathbf{x}^*)}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|}$$

For the canonical hyperplane, for each support vector $\mathbf{x}_i^*$ (with label y_i^*), we have $y_i^* h(\mathbf{x}_i^*) = 1$, and for any point that is not a support vector we have $y_i h(\mathbf{x}_i) > 1$, because, by definition, it must be farther from the hyperplane than a support vector. Over all the n points in the dataset $\mathbf{D}$, we thus obtain the following set of inequalities:

$$y_i\ (\mathbf{w}^T\mathbf{x}_i + b) \geq 1, \text{ for all points } \mathbf{x}_i \in \mathbf{D} \tag{21.8}$$

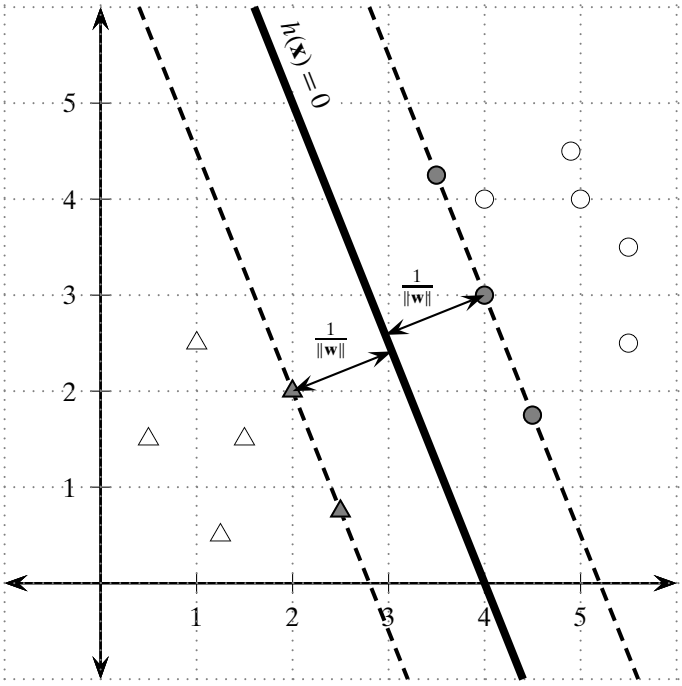


Figure 21.2. Margin of a separating hyperplane: $\frac{1}{\|\mathbf{w}\|}$ is the margin, and the shaded points are the support vectors.

Example 21.2. Figure 21.2 gives an illustration of the support vectors and the margin of a hyperplane. The equation of the separating hyperplane is

$$h(\mathbf{x}) = \begin{pmatrix} 5 \\ 2 \end{pmatrix}^T \mathbf{x} - 20 = 0$$

Consider the support vector $\mathbf{x}^* = (2,2)^T$, with class $y^* = -1$. To find the canonical hyperplane equation, we have to rescale the weight vector and bias by the scalar s, obtained using Eq. (21.7):

$$s = \frac{1}{y^* h(\mathbf{x}^*)} = \frac{1}{-1\left(\begin{pmatrix} 5 \\ 2 \end{pmatrix}^T \begin{pmatrix} 2 \\ 2 \end{pmatrix} - 20\right)} = \frac{1}{6}$$

Thus, the rescaled weight vector is

$$\mathbf{w} = \frac{1}{6}\begin{pmatrix} 5 \\ 2 \end{pmatrix} = \begin{pmatrix} 5/6 \\ 2/6 \end{pmatrix}$$

and the rescaled bias is

$$b = \frac{-20}{6}$$

The canonical form of the hyperplane is therefore

$$h(\mathbf{x}) = \begin{pmatrix} 5/6 \\ 2/6 \end{pmatrix}^T \mathbf{x} - 20/6 = \begin{pmatrix} 0.833 \\ 0.333 \end{pmatrix}^T \mathbf{x} - 3.33$$

and the margin of the canonical hyperplane is

$$\delta^* = \frac{y^* \, h(\mathbf{x}^*)}{\|\mathbf{w}\|} = \frac{1}{\sqrt{\left(\frac{5}{6}\right)^2 + \left(\frac{2}{6}\right)^2}} = \frac{6}{\sqrt{29}} = 1.114$$

In this example there are five support vectors (shown as shaded points), namely, $(2, 2)^T$ and $(2.5, 0.75)^T$ with class $y = -1$ (shown as triangles), and $(3.5, 4.25)^T$, $(4, 3)^T$, and $(4.5, 1.75)^T$ with class $y = +1$ (shown as circles), as illustrated in Figure 21.2.

21.2 SVM: LINEAR AND SEPARABLE CASE

Given a dataset $\mathbf{D} = \{\mathbf{x}_i, y_i\}_{i=1}^n$ with $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{+1, -1\}$, let us assume for the moment that the points are linearly separable, that is, there exists a separating hyperplane that perfectly classifies each point. In other words, all points labeled $y_i = +1$ lie on one side ($h(\mathbf{x}) > 0$) and all points labeled $y_i = -1$ lie on the other side ($h(\mathbf{x}) < 0$) of the hyperplane. It is obvious that in the linearly separable case, there are in fact an infinite number of such separating hyperplanes. Which one should we choose?

Maximum Margin Hyperplane

The fundamental idea behind SVMs is to choose the canonical hyperplane, specified by the weight vector $\mathbf{w}$ and the bias b, that yields the maximum margin among all possible separating hyperplanes. If δ_h^* represents the margin for hyperplane $h(\mathbf{x}) = 0$, then the goal is to find the optimal hyperplane h^*:

$$h^* = \arg\max_h \left\{ \delta_h^* \right\} = \arg\max_{\mathbf{w},b} \left\{ \frac{1}{\|\mathbf{w}\|} \right\}$$

The SVM task is to find the hyperplane that maximizes the margin $\frac{1}{\|\mathbf{w}\|}$, subject to the n constraints given in Eq. (21.8), namely, $y_i \ (\mathbf{w}^T\mathbf{x}_i + b) \geq 1, \ \text{for all points } \mathbf{x}_i \in \mathbf{D}$. Notice that instead of maximizing the margin $\frac{1}{\|\mathbf{w}\|}$, we can minimize $\|\mathbf{w}\|$. In fact, we can obtain an equivalent minimization formulation given as follows:

$$\textbf{Objective Function: } \min_{\mathbf{w},b} \left\{ \frac{\|\mathbf{w}\|^2}{2} \right\}$$

$$\textbf{Linear Constraints: } y_i \ (\mathbf{w}^T\mathbf{x}_i + b) \geq 1, \ \ \forall \mathbf{x}_i \in \mathbf{D}$$

We can directly solve the above *primal* convex minimization problem with the n linear constraints using standard optimization algorithms, as outlined later in Section 21.5. However, it is more common to solve the *dual* problem, which is obtained via the use of *Lagrange multipliers*. The main idea is to introduce a Lagrange multiplier

α_i for each constraint, which satisfies the Karush–Kuhn–Tucker (KKT) conditions at the optimal solution:

$$\alpha_i \left(y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1\right) = 0$$
$$\text{and } \alpha_i \geq 0$$

Incorporating all the n constraints, the new objective function, called the *Lagrangian*, then becomes

$$\min\ L = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n} \alpha_i \left(y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1\right) \tag{21.9}$$

L should be minimized with respect to $\mathbf{w}$ and b, and it should be maximized with respect to α_i.

Taking the derivative of L with respect to $\mathbf{w}$ and b, and setting those to zero, we obtain

$$\frac{\partial}{\partial \mathbf{w}} L = \mathbf{w} - \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i = \mathbf{0} \quad \text{or} \quad \mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \tag{21.10}$$

$$\frac{\partial}{\partial b} L = \sum_{i=1}^{n} \alpha_i y_i = 0 \tag{21.11}$$

The above equations give important intuition about the optimal weight vector $\mathbf{w}$. In particular, Eq. (21.10) implies that $\mathbf{w}$ can be expressed as a linear combination of the data points $\mathbf{x}_i$, with the signed Lagrange multipliers, $\alpha_i y_i$, serving as the coefficients. Further, Eq. (21.11) implies that the sum of the signed Lagrange multipliers, $\alpha_i y_i$, must be zero.

Plugging these into Eq. (21.9), we obtain the *dual Lagrangian* objective function, which is specified purely in terms of the Lagrange multipliers:

$$\begin{aligned} L_{dual} &= \frac{1}{2}\mathbf{w}^T\mathbf{w} - \mathbf{w}^T\Bigg(\underbrace{\sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i}_{\mathbf{w}}\Bigg) - b\underbrace{\sum_{i=1}^{n} \alpha_i y_i}_{0} + \sum_{i=1}^{n} \alpha_i \\ &= -\frac{1}{2}\mathbf{w}^T\mathbf{w} + \sum_{i=1}^{n} \alpha_i \\ &= \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j \end{aligned}$$

The dual objective is thus given as

$$\begin{aligned} &\textbf{Objective Function: } \max_{\boldsymbol{\alpha}}\ L_{dual} = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j \\ &\textbf{Linear Constraints: } \alpha_i \geq 0,\ \forall i \in \mathbf{D},\ \text{and } \sum_{i=1}^{n} \alpha_i y_i = 0 \end{aligned} \tag{21.12}$$

where $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_n)^T$ is the vector comprising the Lagrange multipliers. L_{dual} is a convex quadratic programming problem (note the $\alpha_i \alpha_j$ terms), which can be solved using standard optimization techniques. See Section 21.5 for a gradient-based method for solving the dual formulation.

Weight Vector and Bias

Once we have obtained the α_i values for $i = 1, \ldots, n$, we can solve for the weight vector $\mathbf{w}$ and the bias b. Note that according to the KKT conditions, we have

$$\alpha_i \left(y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1 \right) = 0$$

which gives rise to two cases:

(1) $\alpha_i = 0$, or

(2) $y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1 = 0$, which implies $y_i (\mathbf{w}^T \mathbf{x}_i + b) = 1$

This is a very important result because if $\alpha_i > 0$, then $y_i (\mathbf{w}^T \mathbf{x}_i + b) = 1$, and thus the point $\mathbf{x}_i$ must be a support vector. On the other hand if $y_i (\mathbf{w}^T \mathbf{x}_i + b) > 1$, then $\alpha_i = 0$, that is, if a point is not a support vector, then $\alpha_i = 0$.

Once we know α_i for all points, we can compute the weight vector $\mathbf{w}$ using Eq. (21.10), but by taking the summation only for the support vectors:

$$\mathbf{w} = \sum_{i, \alpha_i > 0} \alpha_i y_i \mathbf{x}_i \tag{21.13}$$

In other words, $\mathbf{w}$ is obtained as a linear combination of the support vectors, with the $\alpha_i y_i$'s representing the weights. The rest of the points (with $\alpha_i = 0$) are not support vectors and thus do not play a role in determining $\mathbf{w}$.

To compute the bias b, we first compute one solution b_i, per support vector, as follows:

$$\begin{aligned} &\alpha_i \left(y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1 \right) = 0 \\ &y_i (\mathbf{w}^T \mathbf{x}_i + b) = 1 \\ &b_i = \frac{1}{y_i} - \mathbf{w}^T \mathbf{x}_i = y_i - \mathbf{w}^T \mathbf{x}_i \end{aligned} \tag{21.14}$$

We can take b as the average bias value over all the support vectors:

$$b = \text{avg}_{\alpha_i > 0} \{ b_i \} \tag{21.15}$$

SVM Classifier

Given the optimal hyperplane function $h(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$, for any new point $\mathbf{z}$, we predict its class as

$$\hat{y} = \text{sign}(h(\mathbf{z})) = \text{sign}(\mathbf{w}^T \mathbf{z} + b) \tag{21.16}$$

where the sign$(\cdot)$ function returns $+1$ if its argument is positive, and -1 if its argument is negative.

Table 21.1. Dataset corresponding to Figure 21.2

$\mathbf{x}_i$	x_{i1}	x_{i2}	y_i
$\mathbf{x}_1$	3.5	4.25	+1
$\mathbf{x}_2$	4	3	+1
$\mathbf{x}_3$	4	4	+1
$\mathbf{x}_4$	4.5	1.75	+1
$\mathbf{x}_5$	4.9	4.5	+1
$\mathbf{x}_6$	5	4	+1
$\mathbf{x}_7$	5.5	2.5	+1
$\mathbf{x}_8$	5.5	3.5	+1
$\mathbf{x}_9$	0.5	1.5	−1
$\mathbf{x}_{10}$	1	2.5	−1
$\mathbf{x}_{11}$	1.25	0.5	−1
$\mathbf{x}_{12}$	1.5	1.5	−1
$\mathbf{x}_{13}$	2	2	−1
$\mathbf{x}_{14}$	2.5	0.75	−1

Example 21.3. Let us continue with the example dataset shown in Figure 21.2. The dataset has 14 points as shown in Table 21.1.

Solving the L_{dual} quadratic program yields the following nonzero values for the Lagrangian multipliers, which determine the support vectors

$\mathbf{x}_i$	x_{i1}	x_{i2}	y_i	α_i
$\mathbf{x}_1$	3.5	4.25	+1	0.0437
$\mathbf{x}_2$	4	3	+1	0.2162
$\mathbf{x}_4$	4.5	1.75	+1	0.1427
$\mathbf{x}_{13}$	2	2	−1	0.3589
$\mathbf{x}_{14}$	2.5	0.75	−1	0.0437

All other points have $\alpha_i = 0$ and therefore they are not support vectors. Using Eq. (21.13), we can compute the weight vector for the hyperplane:

$$\begin{aligned}\mathbf{w} &= \sum_{i,\alpha_i>0} \alpha_i y_i \mathbf{x}_i \\ &= 0.0437\begin{pmatrix}3.5\\4.25\end{pmatrix} + 0.2162\begin{pmatrix}4\\3\end{pmatrix} + 0.1427\begin{pmatrix}4.5\\1.75\end{pmatrix} - 0.3589\begin{pmatrix}2\\2\end{pmatrix} - 0.0437\begin{pmatrix}2.5\\0.75\end{pmatrix} \\ &= \begin{pmatrix}0.833\\0.334\end{pmatrix}\end{aligned}$$

The final bias is the average of the bias obtained from each support vector using Eq. (21.14):

$\mathbf{x}_i$	$\mathbf{w}^T\mathbf{x}_i$	$b_i = y_i - \mathbf{w}^T\mathbf{x}_i$
$\mathbf{x}_1$	4.332	−3.332
$\mathbf{x}_2$	4.331	−3.331
$\mathbf{x}_4$	4.331	−3.331
$\mathbf{x}_{13}$	2.333	−3.333
$\mathbf{x}_{14}$	2.332	−3.332
$b = \text{avg}\{b_i\}$		−3.332

Thus, the optimal hyperplane is given as follows:

$$h(\mathbf{x}) = \begin{pmatrix} 0.833 \\ 0.334 \end{pmatrix}^T \mathbf{x} - 3.332 = 0$$

which matches the canonical hyperplane in Example 21.2.

21.3 SOFT MARGIN SVM: LINEAR AND NONSEPARABLE CASE

So far we have assumed that the dataset is perfectly linearly separable. Here we consider the case where the classes overlap to some extent so that a perfect separation is not possible, as depicted in Figure 21.3.

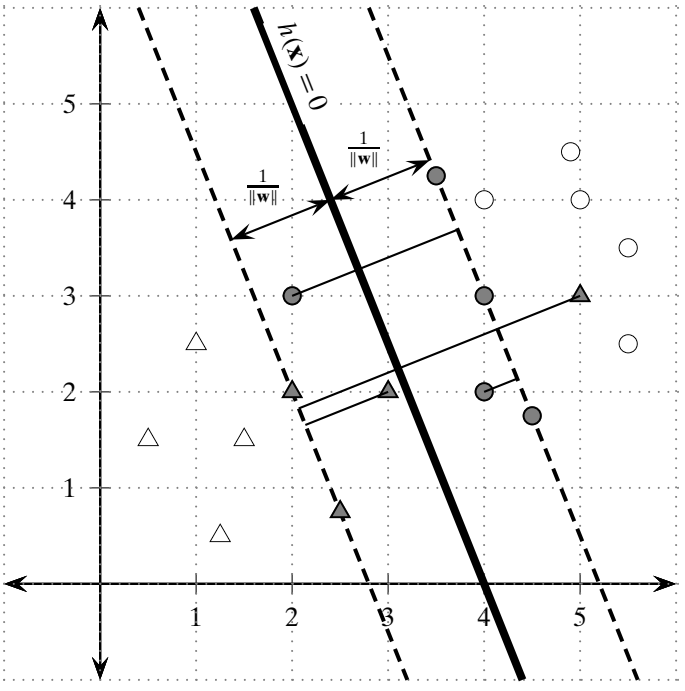


Figure 21.3. Soft margin hyperplane: the shaded points are the support vectors. The margin is $1/\|\mathbf{w}\|$ as illustrated, and points with positive slack values are also shown (thin black line).

Recall that when points are linearly separable we can find a separating hyperplane so that all points satisfy the condition $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$. SVMs can handle non-separable points by introducing *slack variables* ξ_i in Eq. (21.8), as follows:

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i$$

where $\xi_i \geq 0$ is the slack variable for point $\mathbf{x}_i$, which indicates how much the point violates the separability condition, that is, the point may no longer be at least $1/\|\mathbf{w}\|$ away from the hyperplane. The slack values indicate three types of points. If $\xi_i = 0$, then the corresponding point $\mathbf{x}_i$ is at least $\frac{1}{\|\mathbf{w}\|}$ away from the hyperplane. If $0 < \xi_i < 1$, then the point is within the margin and still correctly classified, that is, it is on the correct side of the hyperplane. However, if $\xi_i \geq 1$ then the point is misclassified and appears on the wrong side of the hyperplane.

In the nonseparable case, also called the *soft margin* case, the goal of SVM classification is to find the hyperplane with maximum margin that also minimizes the slack terms. The new objective function is given as

$$\begin{aligned} \textbf{Objective Function: } & \min_{\mathbf{w},b,\xi_i} \left\{ \frac{\|\mathbf{w}\|^2}{2} + C\sum_{i=1}^{n}(\xi_i)^k \right\} \\ \textbf{Linear Constraints: } & y_i\,(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i,\ \ \forall \mathbf{x}_i \in \mathbf{D} \\ & \xi_i \geq 0\ \ \forall \mathbf{x}_i \in \mathbf{D} \end{aligned} \tag{21.17}$$

where C and k are constants that incorporate the cost of misclassification. The term $\sum_{i=1}^{n}(\xi_i)^k$ gives the *loss*, that is, an estimate of the deviation from the separable case. The scalar C, which is chosen empirically, is a *regularization constant* that controls the trade-off between maximizing the margin (corresponding to minimizing $\|\mathbf{w}\|^2/2$) or minimizing the loss (corresponding to minimizing the sum of the slack terms $\sum_{i=1}^{n}(\xi_i)^k$). For example, if $C \to 0$, then the loss component essentially disappears, and the objective defaults to maximizing the margin. On the other hand, if $C \to \infty$, then the margin ceases to have much effect, and the objective function tries to minimize the loss. The constant k governs the form of the loss. Typically k is set to 1 or 2. When $k = 1$, called *hinge loss*, the goal is to minimize the sum of the slack variables, whereas when $k = 2$, called *quadratic loss*, the goal is to minimize the sum of the squared slack variables.

21.3.1 Hinge Loss

Assuming $k = 1$, we can compute the Lagrangian for the optimization problem in Eq. (21.17) by introducing Lagrange multipliers α_i and β_i that satisfy the following KKT conditions at the optimal solution:

$$\begin{aligned} \alpha_i\left(y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 + \xi_i\right) = 0 \text{ with } \alpha_i \geq 0 \\ \beta_i(\xi_i - 0) = 0 \text{ with } \beta_i \geq 0 \end{aligned} \tag{21.18}$$

The Lagrangian is then given as

$$L = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\alpha_i\left(y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 + \xi_i\right) - \sum_{i=1}^{n}\beta_i\xi_i \tag{21.19}$$

We turn this into a dual Lagrangian by taking its partial derivative with respect to $\mathbf{w}$, b and ξ_i, and setting those to zero:

$$\begin{aligned}
\frac{\partial}{\partial \mathbf{w}} L &= \mathbf{w} - \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i = \mathbf{0} \quad \text{or} \quad \mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \\
\frac{\partial}{\partial b} L &= \sum_{i=1}^{n} \alpha_i y_i = 0 \\
\frac{\partial}{\partial \xi_i} L &= C - \alpha_i - \beta_i = 0 \quad \text{or} \quad \beta_i = C - \alpha_i
\end{aligned} \tag{21.20}$$

Plugging these values into Eq. (21.19), we get

$$\begin{aligned}
L_{dual} &= \frac{1}{2}\mathbf{w}^T\mathbf{w} - \mathbf{w}^T\Big(\underbrace{\sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i}_{\mathbf{w}}\Big) - b\underbrace{\sum_{i=1}^{n} \alpha_i y_i}_{0} + \sum_{i=1}^{n} \alpha_i + \sum_{i=1}^{n} \underbrace{(C - \alpha_i + \beta_i)}_{0} \xi_i \\
&= \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j
\end{aligned}$$

The dual objective is thus given as

$$\begin{aligned}
&\textbf{Objective Function: } \max_{\boldsymbol{\alpha}} \; L_{dual} = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \\
&\textbf{Linear Constraints: } 0 \le \alpha_i \le C, \;\; \forall i \in \mathbf{D} \text{ and } \sum_{i=1}^{n} \alpha_i y_i = 0
\end{aligned} \tag{21.21}$$

Notice that the objective is the same as the dual Lagrangian in the linearly separable case [Eq. (21.12)]. However, the constraints on α_i's are different because we now require that $\alpha_i + \beta_i = C$ with $\alpha_i \ge 0$ and $\beta_i \ge 0$, which implies that $0 \le \alpha_i \le C$. Section 21.5 describes a gradient ascent approach for solving this dual objective function.

Weight Vector and Bias

Once we solve for α_i, we have the same situation as before, namely, $\alpha_i = 0$ for points that are not support vectors, and $\alpha_i > 0$ only for the support vectors, which comprise all points $\mathbf{x}_i$ for which we have

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) = 1 - \xi_i \tag{21.22}$$

Notice that the support vectors now include all points that are on the margin, which have zero slack ($\xi_i = 0$), as well as all points with positive slack ($\xi_i > 0$).

We can obtain the weight vector from the support vectors as before:

$$\mathbf{w} = \sum_{i,\alpha_i > 0} \alpha_i y_i \mathbf{x}_i \tag{21.23}$$

We can also solve for the β_i using Eq. (21.20):

$$\beta_i = C - \alpha_i$$

Replacing β_i in the KKT conditions [Eq. (21.18)] with the expression from above we obtain

$$(C - \alpha_i)\xi_i = 0 \tag{21.24}$$

Thus, for the support vectors with $\alpha_i > 0$, we have two cases to consider:

(1) $\xi_i > 0$, which implies that $C - \alpha_i = 0$, that is, $\alpha_i = C$, or

(2) $C - \alpha_i > 0$, that is $\alpha_i < C$. In this case, from Eq. (21.24) we must have $\xi_i = 0$. In other words, these are precisely those support vectors that are on the margin.

Using those support vectors that are on the margin, that is, have $0 < \alpha_i < C$ and $\xi_i = 0$, we can solve for b_i:

$$\begin{aligned} &\alpha_i \left(y_i(\mathbf{w}^T\mathbf{x}_i + b_i) - 1\right) = 0 \\ &y_i(\mathbf{w}^T\mathbf{x}_i + b_i) = 1 \\ &b_i = \frac{1}{y_i} - \mathbf{w}^T\mathbf{x}_i = y_i - \mathbf{w}^T\mathbf{x}_i \end{aligned} \tag{21.25}$$

To obtain the final bias b, we can take the average over all the b_i values. From Eqs. (21.23) and (21.25), both the weight vector $\mathbf{w}$ and the bias term b can be computed without explicitly computing the slack terms ξ_i for each point.

Once the optimal hyperplane plane has been determined, the SVM model predicts the class for a new point $\mathbf{z}$ as follows:

$$\hat{y} = \text{sign}(h(\mathbf{z})) = \text{sign}(\mathbf{w}^T\mathbf{z} + b)$$

Example 21.4. Let us consider the data points shown in Figure 21.3. There are four new points in addition to the 14 points from Table 21.1 that we considered in Example 21.3; these points are

$\mathbf{x}_i$	x_{i1}	x_{i2}	y_i
$\mathbf{x}_{15}$	4	2	+1
$\mathbf{x}_{16}$	2	3	+1
$\mathbf{x}_{17}$	3	2	−1
$\mathbf{x}_{18}$	5	3	−1

Let $k = 1$ and $C = 1$, then solving the L_{dual} yields the following support vectors and Lagrangian values α_i:

$\mathbf{x}_i$	x_{i1}	x_{i2}	y_i	α_i
$\mathbf{x}_1$	3.5	4.25	+1	0.0271
$\mathbf{x}_2$	4	3	+1	0.2162
$\mathbf{x}_4$	4.5	1.75	+1	0.9928
$\mathbf{x}_{13}$	2	2	−1	0.9928
$\mathbf{x}_{14}$	2.5	0.75	−1	0.2434
$\mathbf{x}_{15}$	4	2	+1	1
$\mathbf{x}_{16}$	2	3	+1	1
$\mathbf{x}_{17}$	3	2	−1	1
$\mathbf{x}_{18}$	5	3	−1	1

All other points are not support vectors, having $\alpha_i = 0$. Using Eq. (21.23) we compute the weight vector for the hyperplane:

$$\begin{aligned}\mathbf{w} &= \sum_{i,\alpha_i>0} \alpha_i y_i \mathbf{x}_i \\ &= 0.0271\begin{pmatrix}3.5\\4.25\end{pmatrix} + 0.2162\begin{pmatrix}4\\3\end{pmatrix} + 0.9928\begin{pmatrix}4.5\\1.75\end{pmatrix} - 0.9928\begin{pmatrix}2\\2\end{pmatrix} \\ &\quad - 0.2434\begin{pmatrix}2.5\\0.75\end{pmatrix} + \begin{pmatrix}4\\2\end{pmatrix} + \begin{pmatrix}2\\3\end{pmatrix} - \begin{pmatrix}3\\2\end{pmatrix} - \begin{pmatrix}5\\3\end{pmatrix} \\ &= \begin{pmatrix}0.834\\0.333\end{pmatrix}\end{aligned}$$

The final bias is the average of the biases obtained from each support vector using Eq. (21.25). Note that we compute the per-point bias only for the support vectors that lie precisely on the margin. These support vectors have $\xi_i = 0$ and have $0 < \alpha_i < C$. Put another way, we do not compute the bias for support vectors with $\alpha_i = C = 1$, which include the points $\mathbf{x}_{15}$, $\mathbf{x}_{16}$, $\mathbf{x}_{17}$, and $\mathbf{x}_{18}$. From the remaining support vectors, we get

$\mathbf{x}_i$	$\mathbf{w}^T\mathbf{x}_i$	$b_i = y_i - \mathbf{w}^T\mathbf{x}_i$
$\mathbf{x}_1$	4.334	−3.334
$\mathbf{x}_2$	4.334	−3.334
$\mathbf{x}_4$	4.334	−3.334
$\mathbf{x}_{13}$	2.334	−3.334
$\mathbf{x}_{14}$	2.334	−3.334
$b = \text{avg}\{b_i\}$		−3.334

Thus, the optimal hyperplane is given as follows:

$$h(\mathbf{x}) = \begin{pmatrix}0.834\\0.333\end{pmatrix}^T \mathbf{x} - 3.334 = 0$$

One can see that this is essentially the same as the canonical hyperplane we found in Example 21.3.

It is instructive to see what the slack variables are in this case. Note that $\xi_i = 0$ for all points that are not support vectors, and also for those support vectors that are on the margin. So the slack is positive only for the remaining support vectors, for whom the slack can be computed directly from Eq. (21.22), as follows:

$$\xi_i = 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b)$$

Thus, for all support vectors not on the margin, we have

$\mathbf{x}_i$	$\mathbf{w}^T\mathbf{x}_i$	$\mathbf{w}^T\mathbf{x}_i + b$	$\xi_i = 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b)$
$\mathbf{x}_{15}$	4.001	0.667	0.333
$\mathbf{x}_{16}$	2.667	−0.667	1.667
$\mathbf{x}_{17}$	3.167	−0.167	0.833
$\mathbf{x}_{18}$	5.168	1.834	2.834

As expected, the slack variable $\xi_i > 1$ for those points that are misclassified (i.e., are on the wrong side of the hyperplane), namely $\mathbf{x}_{16} = (3, 3)^T$ and $\mathbf{x}_{18} = (5, 3)^T$. The other two points are correctly classified, but lie within the margin, and thus satisfy $0 < \xi_i < 1$. The total slack is given as

$$\sum_i \xi_i = \xi_{15} + \xi_{16} + \xi_{17} + \xi_{18} = 0.333 + 1.667 + 0.833 + 2.834 = 5.667$$

21.3.2 Quadratic Loss

For quadratic loss, we have $k = 2$ in the objective function [Eq. (21.17)]. In this case we can drop the positivity constraint $\xi_i \geq 0$ due to the fact that (1) the sum of the slack terms $\sum_{i=1}^n \xi_i^2$ is always positive, and (2) a potential negative value of slack will be ruled out during optimization because a choice of $\xi_i = 0$ leads to a smaller value of the primary objective, and it still satisfies the constraint $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i$ whenever $\xi_i < 0$. In other words, the optimization process will replace any negative slack variables by zero values. Thus, the SVM objective for quadratic loss is given as

$$\textbf{Objective Function: } \min_{\mathbf{w}, b, \xi_i} \left\{ \frac{\|\mathbf{w}\|^2}{2} + C \sum_{i=1}^n \xi_i^2 \right\}$$

$$\textbf{Linear Constraints: } y_i\ (\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i,\ \ \forall \mathbf{x}_i \in \mathbf{D}$$

The Lagrangian is then given as:

$$L = \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^n \xi_i^2 - \sum_{i=1}^n \alpha_i \left(y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 + \xi_i \right) \tag{21.26}$$

Differentiating with respect to $\mathbf{w}$, b, and ξ_i and setting them to zero results in the following conditions, respectively:

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

$$\xi_i = \frac{1}{2C}\alpha_i$$

Substituting these back into Eq. (21.26) yields the dual objective

$$\begin{aligned} L_{dual} &= \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j - \frac{1}{4C}\sum_{i=1}^{n} \alpha_i^2 \\ &= \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \left(\mathbf{x}_i^T \mathbf{x}_j + \frac{1}{2C}\delta_{ij}\right) \end{aligned}$$

where δ is the *Kronecker delta* function, defined as $\delta_{ij} = 1$ if $i = j$, and $\delta_{ij} = 0$ otherwise. Thus, the dual objective is given as

$$\begin{aligned} &\max_{\boldsymbol{\alpha}} \; L_{dual} = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \left(\mathbf{x}_i^T \mathbf{x}_j + \frac{1}{2C}\delta_{ij}\right) \\ &\text{subject to the constraints } \alpha_i \geq 0, \forall i \in \mathbf{D}, \text{and } \sum_{i=1}^{n} \alpha_i y_i = 0 \end{aligned} \tag{21.27}$$

Once we solve for α_i using the methods from Section 21.5, we can recover the weight vector and bias as follows:

$$\mathbf{w} = \sum_{i, \alpha_i > 0} \alpha_i y_i \mathbf{x}_i$$

$$b = \text{avg}_{i, C > \alpha_i > 0} \left\{ y_i - \mathbf{w}^T \mathbf{x}_i \right\}$$

21.4 KERNEL SVM: NONLINEAR CASE

The linear SVM approach can be used for datasets with a nonlinear decision boundary via the kernel trick from Chapter 5. Conceptually, the idea is to map the original d-dimensional points $\mathbf{x}_i$ in the input space to points $\phi(\mathbf{x}_i)$ in a high-dimensional feature space via some nonlinear transformation ϕ. Given the extra flexibility, it is more likely that the points $\phi(\mathbf{x}_i)$ might be linearly separable in the feature space. Note, however, that a linear decision surface in feature space actually corresponds to a nonlinear decision surface in the input space. Further, the kernel trick allows us to carry out all operations via the kernel function computed in input space, rather than having to map the points into feature space.

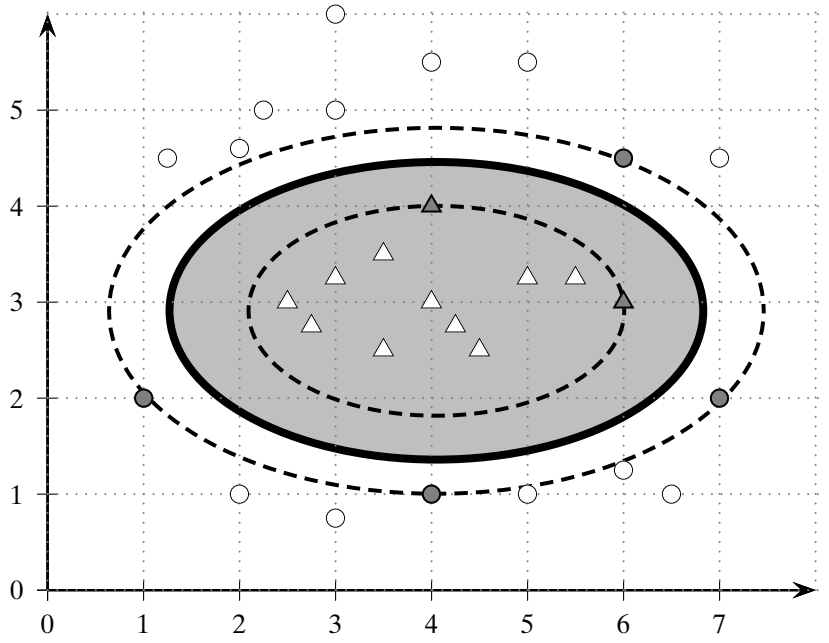


Figure 21.4. Nonlinear SVM: shaded points are the support vectors.

Example 21.5. Consider the set of points shown in Figure 21.4. There is no linear classifier that can discriminate between the points. However, there exists a perfect quadratic classifier that can separate the two classes. Given the input space over the two dimensions X_1 and X_2, if we transform each point $\mathbf{x} = (x_1, x_2)^T$ into a point in the feature space consisting of the dimensions $(X_1, X_2, X_1^2, X_2^2, X_1X_2)$, via the transformation $\phi(\mathbf{x}) = (\sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2)^T$, then it is possible to find a separating hyperplane in feature space. For this dataset, it is possible to map the hyperplane back to the input space, where it is seen as an ellipse (thick black line) that separates the two classes (circles and triangles). The support vectors are those points (shown in gray) that lie on the margin (dashed ellipses).

To apply the kernel trick for nonlinear SVM classification, we have to show that all operations require only the kernel function:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

Let the original database be given as $\mathbf{D} = \{\mathbf{x}_i, y_i\}_{i=1}^n$. Applying ϕ to each point, we can obtain the new dataset in the feature space $\mathbf{D}_\phi = \{\phi(\mathbf{x}_i), y_i\}_{i=1}^n$.

The SVM objective function [Eq. (21.17)] in feature space is given as

$$\textbf{Objective Function: } \min_{\mathbf{w},b,\xi_i} \left\{ \frac{\|\mathbf{w}\|^2}{2} + C \sum_{i=1}^{n} (\xi_i)^k \right\} \tag{21.28}$$

$$\textbf{Linear Constraints: } y_i \ (\mathbf{w}^T \phi(\mathbf{x}_i) + b) \geq 1 - \xi_i, \text{and } \xi_i \geq 0, \ \ \forall \mathbf{x}_i \in \mathbf{D}$$

where $\mathbf{w}$ is the weight vector, b is the bias, and ξ_i are the slack variables, all in feature space.

Hinge Loss

For hinge loss, the dual Lagrangian [Eq. (21.21)] in feature space is given as

$$\begin{aligned}\max_{\boldsymbol{\alpha}} L_{dual} &= \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j) \\ &= \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j K(\mathbf{x}_i,\mathbf{x}_j)\end{aligned} \tag{21.29}$$

Subject to the constraints that $0 \leq \alpha_i \leq C$, and $\sum_{i=1}^{n}\alpha_i y_i = 0$. Notice that the dual Lagrangian depends only on the dot product between two vectors in feature space $\phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j) = K(\mathbf{x}_i,\mathbf{x}_j)$, and thus we can solve the optimization problem using the kernel matrix $\mathbf{K} = \{K(\mathbf{x}_i,\mathbf{x}_j)\}_{i,j=1,\ldots,n}$. Section 21.5 describes a stochastic gradient-based approach for solving the dual objective function.

Quadratic Loss

For quadratic loss, the dual Lagrangian [Eq. (21.27)] corresponds to a change of kernel. Define a new kernel function K_q, as follows:

$$K_q(\mathbf{x}_i,\mathbf{x}_j) = \mathbf{x}_i^T\mathbf{x}_j + \frac{1}{2C}\delta_{ij} = K(\mathbf{x}_i,\mathbf{x}_j) + \frac{1}{2C}\delta_{ij}$$

which affects only the diagonal entries of the kernel matrix $\mathbf{K}$, as $\delta_{ij} = 1$ iff $i = j$, and zero otherwise. Thus, the dual Lagrangian is given as

$$\max_{\boldsymbol{\alpha}} L_{dual} = \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j K_q(\mathbf{x}_i,\mathbf{x}_j) \tag{21.30}$$

subject to the constraints that $\alpha_i \geq 0$, and $\sum_{i=1}^{n}\alpha_i y_i = 0$. The above optimization can be solved using the same approach as for hinge loss, with a simple change of kernel.

Weight Vector and Bias

We can solve for $\mathbf{w}$ in feature space as follows:

$$\mathbf{w} = \sum_{\alpha_i > 0}\alpha_i y_i \phi(\mathbf{x}_i) \tag{21.31}$$

Because $\mathbf{w}$ uses $\phi(\mathbf{x}_i)$ directly, in general, we may not be able or willing to compute $\mathbf{w}$ explicitly. However, as we shall see next, it is not necessary to explicitly compute $\mathbf{w}$ for classifying the points.

Let us now see how to compute the bias via kernel operations. Using Eq. (21.25), we compute b as the average over the support vectors that are on the margin, that is, those with $0 < \alpha_i < C$, and $\xi_i = 0$:

$$b = \text{avg}_{i,\,0<\alpha_i<C}\{b_i\} = \text{avg}_{i,\,0<\alpha_i<C}\left\{y_i - \mathbf{w}^T\phi(\mathbf{x}_i)\right\} \tag{21.32}$$

Substituting $\mathbf{w}$ from Eq. (21.31), we obtain a new expression for b_i as

$$\begin{aligned} b_i &= y_i - \sum_{\alpha_j > 0} \alpha_j y_j \phi(\mathbf{x}_j)^T \phi(\mathbf{x}_i) \\ &= y_i - \sum_{\alpha_j > 0} \alpha_j y_j K(\mathbf{x}_j, \mathbf{x}_i) \end{aligned} \tag{21.33}$$

Notice that b_i is a function of the dot product between two vectors in feature space and therefore it can be computed via the kernel function in the input space.

Kernel SVM Classifier

We can predict the class for a new point $\mathbf{z}$ as follows:

$$\begin{aligned} \hat{y} &= \text{sign}(\mathbf{w}^T \phi(\mathbf{z}) + b) \\ &= \text{sign}\left(\sum_{\alpha_i > 0} \alpha_i y_i \phi(\mathbf{x}_i)^T \phi(\mathbf{z}) + b \right) \\ &= \text{sign}\left(\sum_{\alpha_i > 0} \alpha_i y_i K(\mathbf{x}_i, \mathbf{z}) + b \right) \end{aligned}$$

Once again we see that $\hat{y}$ uses only dot products in feature space.

Based on the above derivations, we can see that, to train and test the SVM classifier, the mapped points $\phi(\mathbf{x}_i)$ are never needed in isolation. Instead, all operations can be carried out in terms of the kernel function $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$. Thus, any nonlinear kernel function can be used to do nonlinear classification in the input space. Examples of such nonlinear kernels include the polynomial kernel [Eq. (5.9)], and the Gaussian kernel [Eq. (5.10)], among others.

Example 21.6. Let us consider the example dataset shown in Figure 21.4; it has 29 points in total. Although it is generally too expensive or infeasible (depending on the choice of the kernel) to compute an explicit representation of the hyperplane in feature space, and to map it back into input space, we will illustrate the application of SVMs in both input and feature space to aid understanding.

We use an inhomogeneous polynomial kernel [Eq. (5.9)] of degree $q = 2$, that is, we use the kernel:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^2$$

With $C = 4$, solving the L_{dual} quadratic program [Eq. (21.30)] in input space yields the following six support vectors, shown as the shaded (gray) points in Figure 21.4.

$\mathbf{x}_i$	$(x_{i1}, x_{i2})^T$	$\phi(\mathbf{x}_i)$	y_i	α_i
$\mathbf{x}_1$	$(1, 2)^T$	$(1, 1.41, 2.83, 1, 4, 2.83)^T$	$+1$	0.6198
$\mathbf{x}_2$	$(4, 1)^T$	$(1, 5.66, 1.41, 16, 1, 5.66)^T$	$+1$	2.069
$\mathbf{x}_3$	$(6, 4.5)^T$	$(1, 8.49, 6.36, 36, 20.25, 38.18)^T$	$+1$	3.803
$\mathbf{x}_4$	$(7, 2)^T$	$(1, 9.90, 2.83, 49, 4, 19.80)^T$	$+1$	0.3182
$\mathbf{x}_5$	$(4, 4)^T$	$(1, 5.66, 5.66, 16, 16, 15.91)^T$	-1	2.9598
$\mathbf{x}_6$	$(6, 3)^T$	$(1, 8.49, 4.24, 36, 9, 25.46)^T$	-1	3.8502

For the inhomogeneous quadratic kernel, the mapping ϕ maps an input point $\mathbf{x}_i$ into feature space as follows:

$$\phi\left(\mathbf{x} = (x_1, x_2)^T\right) = \left(1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2\right)^T$$

The table above shows all the mapped points, which reside in feature space. For example, $\mathbf{x}_1 = (1, 2)^T$ is transformed into

$$\phi(\mathbf{x}_i) = \left(1, \sqrt{2} \cdot 1, \sqrt{2} \cdot 2, 1^2, 2^2, \sqrt{2} \cdot 1 \cdot 2\right)^T = (1, 1.41, 2.83, 1, 2, 2.83)^T$$

We compute the weight vector for the hyperplane using Eq. (21.31):

$$\mathbf{w} = \sum_{i, \alpha_i > 0} \alpha_i y_i \phi(\mathbf{x}_i) = (0, -1.413, -3.298, 0.256, 0.82, -0.018)^T$$

and the bias is computed using Eq. (21.32), which yields

$$b = -8.841$$

For the quadratic polynomial kernel, the decision boundary in input space corresponds to an ellipse. For our example, the center of the ellipse is given as $(4.046, 2.907)$, and the semimajor axis length is 2.78 and the semiminor axis length is 1.55. The resulting decision boundary is the ellipse shown in Figure 21.4. We emphasize that in this example we explicitly transformed all the points into the feature space just for illustration purposes. The kernel trick allows us to achieve the same goal using only the kernel function.

21.5 SVM TRAINING ALGORITHMS

We now turn our attention to algorithms for solving the SVM optimization problems. We will consider simple optimization approaches for solving the dual as well as the primal formulations. It is important to note that these methods are not the most efficient. However, since they are relatively simple, they can serve as a starting point for more sophisticated methods.

For the SVM algorithms in this section, instead of dealing explicitly with the bias b, we map each point $\mathbf{x}_i \in \mathbb{R}^d$ to the point $\mathbf{x}_i' \in \mathbb{R}^{d+1}$ as follows:

$$\mathbf{x}_i' = (x_{i1}, \ldots, x_{id}, 1)^T \tag{21.34}$$

Furthermore, we also map the weight vector to $\mathbb{R}^{d+1}$, with $w_{d+1} = b$, so that

$$\mathbf{w} = (w_1, \ldots, w_d, b)^T \tag{21.35}$$

The equation of the hyperplane [Eq. (21.1)] is then given as follows:

$$h(\mathbf{x}') : \mathbf{w}^T\mathbf{x}' = 0$$

$$h(\mathbf{x}') : \begin{pmatrix} w_1 & \cdots & w_d & b \end{pmatrix} \begin{pmatrix} x_{i1} \\ \vdots \\ x_{id} \\ 1 \end{pmatrix} = 0$$

$$h(\mathbf{x}') : w_1 x_{i1} + \cdots + w_d x_{id} + b = 0$$

In the discussion below we assume that the bias term has been included in $\mathbf{w}$, and that each point has been mapped to $\mathbb{R}^{d+1}$ as per Eqs. (21.34) and (21.35). Thus, the last component of $\mathbf{w}$ yields the bias b. Another consequence of mapping the points to $\mathbb{R}^{d+1}$ is that the constraint $\sum_{i=1}^{n} \alpha_i y_i = 0$ does not apply in the SVM dual formulations given in Eqs. (21.21), (21.27), (21.29), and (21.30), as there is no explicit bias term b for the linear constraints in the SVM objective given in Eq. (21.17). The new set of constraints is given as

$$y_i \mathbf{w}^T \mathbf{x} \geq 1 - \xi_i$$

21.5.1 Dual Solution: Stochastic Gradient Ascent

We consider only the hinge loss case because quadratic loss can be handled by a change of kernel, as shown in Eq. (21.30). The dual optimization objective for hinge loss [Eq. (21.29)] is given as

$$\max_{\boldsymbol{\alpha}} J(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

subject to the constraints $0 \leq \alpha_i \leq C$ for all $i = 1, \ldots, n$. Here $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \cdots, \alpha_n)^T \in \mathbb{R}^n$.

Let us consider the terms in $J(\boldsymbol{\alpha})$ that involve the Lagrange multiplier α_k:

$$J(\alpha_k) = \alpha_k - \frac{1}{2}\alpha_k^2 y_k^2 K(\mathbf{x}_k, \mathbf{x}_k) - \alpha_k y_k \sum_{\substack{i=1 \\ i \neq k}}^{n} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}_k)$$

The gradient or the rate of change in the objective function at $\boldsymbol{\alpha}$ is given as the partial derivative of $J(\boldsymbol{\alpha})$ with respect to $\boldsymbol{\alpha}$, that is, with respect to each α_k:

$$\nabla J(\boldsymbol{\alpha}) = \left(\frac{\partial J(\boldsymbol{\alpha})}{\partial \alpha_1}, \frac{\partial J(\boldsymbol{\alpha})}{\partial \alpha_2}, \ldots, \frac{\partial J(\boldsymbol{\alpha})}{\partial \alpha_n} \right)^T$$

where the kth component of the gradient is obtained by differentiating $J(\alpha_k)$ with respect to α_k:

$$\frac{\partial J(\boldsymbol{\alpha})}{\partial \alpha_k} = \frac{\partial J(\alpha_k)}{\partial \alpha_k} = 1 - y_k \left(\sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}_k) \right) \tag{21.36}$$

Because we want to maximize the objective function $J(\boldsymbol{\alpha})$, we should move in the direction of the gradient $\nabla J(\boldsymbol{\alpha})$. Starting from an initial $\boldsymbol{\alpha}$, the gradient ascent approach successively updates it as follows:

$$\boldsymbol{\alpha}_{t+1} = \boldsymbol{\alpha}_t + \eta_t \nabla J(\boldsymbol{\alpha}_t)$$

where $\boldsymbol{\alpha}_t$ is the estimate at the tth step, and η_t is the step size.

Instead of updating the entire $\boldsymbol{\alpha}$ vector in each step, in the stochastic gradient ascent approach, we update each component α_k independently and immediately use the new value to update other components. This can result in faster convergence. The update rule for the k-th component is given as

$$\alpha_k = \alpha_k + \eta_k \frac{\partial J(\boldsymbol{\alpha})}{\partial \alpha_k} = \alpha_k + \eta_k \left(1 - y_k \sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}_k)\right) \tag{21.37}$$

where η_k is the step size. We also have to ensure that the constraints $\alpha_k \in [0, C]$ are satisfied. Thus, in the update step above, if $\alpha_k < 0$ we reset it to $\alpha_k = 0$, and if $\alpha_k > C$ we reset it to $\alpha_k = C$. The pseudo-code for stochastic gradient ascent is given in Algorithm 21.1.

ALGORITHM 21.1. Dual SVM Algorithm: Stochastic Gradient Ascent

SVM-DUAL (D, K, C, ϵ):

foreach $\mathbf{x}_i \in \mathbf{D}$ **do** $\mathbf{x}_i \leftarrow \begin{pmatrix} \mathbf{x}_i \\ 1 \end{pmatrix}$ `// map to` $\mathbb{R}^{d+1}$

if *loss* = `hinge` **then**

  $\mathbf{K} \leftarrow \{K(\mathbf{x}_i, \mathbf{x}_j)\}_{i,j=1,\ldots,n}$ `// kernel matrix, hinge loss`

else if *loss* = `quadratic` **then**

  $\mathbf{K} \leftarrow \{K(\mathbf{x}_i, \mathbf{x}_j) + \frac{1}{2C}\delta_{ij}\}_{i,j=1,\ldots,n}$ `// kernel matrix, quadratic loss`

for $k = 1, \ldots, n$ **do** $\eta_k \leftarrow \frac{1}{K(\mathbf{x}_k, \mathbf{x}_k)}$ `// set step size`

$t \leftarrow 0$

$\boldsymbol{\alpha}_0 \leftarrow (0, \ldots, 0)^T$

repeat

  $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha}_t$

  for $k = 1$ *to* n **do**

    `// update` k`th component of` $\boldsymbol{\alpha}$

    $\alpha_k \leftarrow \alpha_k + \eta_k \left(1 - y_k \sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}_k)\right)$

    if $\alpha_k < 0$ **then** $\alpha_k \leftarrow 0$

    if $\alpha_k > C$ **then** $\alpha_k \leftarrow C$

  $\boldsymbol{\alpha}_{t+1} = \boldsymbol{\alpha}$

  $t \leftarrow t + 1$

until $\|\boldsymbol{\alpha}_t - \boldsymbol{\alpha}_{t-1}\| \leq \epsilon$

To determine the step size η_k, ideally, we would like to choose it so that the gradient at α_k goes to zero, which happens when

$$\eta_k = \frac{1}{K(\mathbf{x}_k, \mathbf{x}_k)} \tag{21.38}$$

To see why, note that when only α_k is updated, the other α_i do not change. Thus, the new $\boldsymbol{\alpha}$ has a change only in α_k, and from Eq. (21.36) we get

$$\frac{\partial J(\boldsymbol{\alpha})}{\partial \alpha_k} = \left(1 - y_k \sum_{i \neq k} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}_k)\right) - y_k \alpha_k y_k K(\mathbf{x}_k, \mathbf{x}_k)$$

Plugging in the value of α_k from Eq. (21.37), we have

$$\begin{aligned}
\frac{\partial J(\boldsymbol{\alpha})}{\partial \alpha_k} &= \left(1 - y_k \sum_{i \neq k} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}_k)\right) - \left(\alpha_k + \eta_k \left(1 - y_k \sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}_k)\right)\right) K(\mathbf{x}_k, \mathbf{x}_k) \\
&= \left(1 - y_k \sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}_k)\right) - \eta_k K(\mathbf{x}_k, \mathbf{x}_k) \left(1 - y_k \sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}_k)\right) \\
&= \left(1 - \eta_k K(\mathbf{x}_k, \mathbf{x}_k)\right) \left(1 - y_k \sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}_k)\right)
\end{aligned}$$

Substituting η_k from Eq. (21.38), we have

$$\frac{\partial J(\boldsymbol{\alpha})}{\partial a_k} = \left(1 - \frac{1}{K(\mathbf{x}_k, \mathbf{x}_k)} K(\mathbf{x}_k, \mathbf{x}_k)\right) \left(1 - y_k \sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}_k)\right) = 0$$

In Algorithm 21.1, for better convergence, we thus choose η_k according to Eq. (21.38). The method successively updates $\boldsymbol{\alpha}$ and stops when the change falls below a given threshold ϵ. Since the above description assumes a general kernel function between any two points, we can recover the linear, nonseparable case by simply setting $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$. The computational complexity of the method is $O(n^2)$ per iteration.

Note that once we obtain the final $\boldsymbol{\alpha}$, we classify a new point $\mathbf{z} \in \mathbb{R}^{d+1}$ as follows:

$$\hat{y} = \text{sign}\Big(h(\phi(\mathbf{z}))\Big) = \text{sign}\Big(\mathbf{w}^T \phi(\mathbf{z})\Big) = \text{sign}\left(\sum_{\alpha_i > 0} \alpha_i y_i K(\mathbf{x}_i, \mathbf{z})\right)$$

Example 21.7 (Dual SVM: Linear Kernel). Figure 21.5 shows the $n = 150$ points from the Iris dataset, using `sepal length` and `sepal width` as the two attributes. The goal is to discriminate between `Iris-setosa` (shown as circles) and other types of Iris flowers (shown as triangles). Algorithm 21.1 was used to train the SVM classifier with a linear kernel $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$ and convergence threshold $\epsilon = 0.0001$, with hinge loss. Two different values of C were used; hyperplane h_{10} is obtained by using $C = 10$, whereas h_{1000} uses $C = 1000$; the hyperplanes are given as follows:

$$\begin{aligned}
h_{10}(\mathbf{x}): &\quad 2.74x_1 - 3.74x_2 - 3.09 = 0 \\
h_{1000}(\mathbf{x}): &\quad 8.56x_1 - 7.14x_2 - 23.12 = 0
\end{aligned}$$

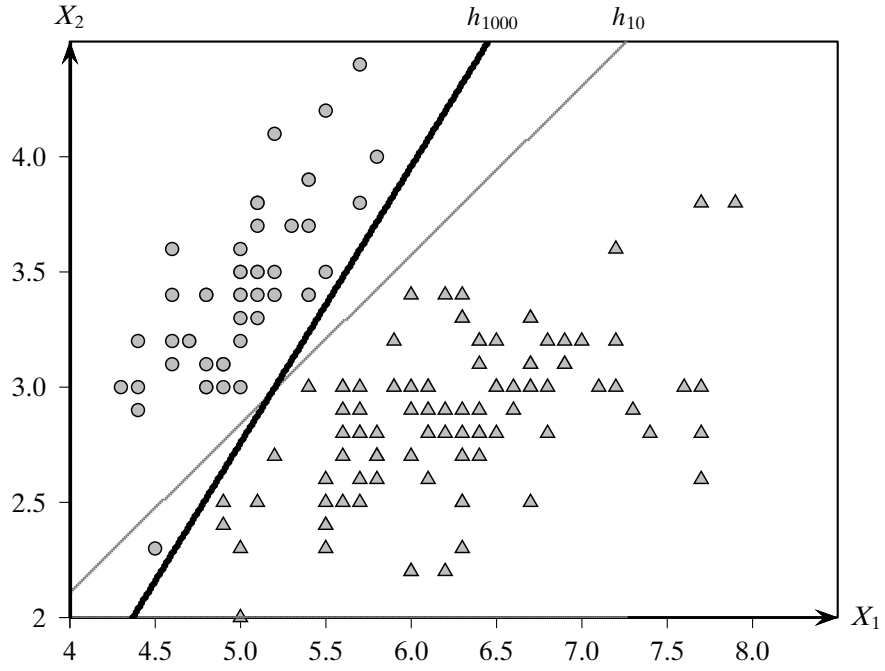


Figure 21.5. SVM dual algorithm with linear kernel.

The hyperplane h_{10} has a larger margin, but it has a larger slack; it misclassifies one of the circles. On the other hand, the hyperplane h_{1000} has a smaller margin, but it minimizes the slack; it is a separating hyperplane. This example illustrates the fact that the higher the value of C the more the emphasis on minimizing the slack.

Example 21.8 (Dual SVM: Quadratic Kernel). Figure 21.6 shows the $n = 150$ points from the Iris dataset projected on the first two principal components. The task is to separate `Iris-versicolor` (in circles) from the other two types of Irises (in triangles). The figure plots the decision boundaries obtained when using the linear kernel $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$, and the homogeneous quadratic kernel $K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T \mathbf{x}_j)^2$, where $\mathbf{x}_i \in \mathbb{R}^{d+1}$, as per Eq. (21.34). The optimal hyperplane in both cases was found via the gradient ascent approach in Algorithm 21.1, with $C = 10$, $\epsilon = 0.0001$ and using hinge loss.

The optimal hyperplane h_l (shown in gray) for the linear kernel is given as

$$h_l(\mathbf{x}) : 0.16x_1 + 1.9x_2 + 0.8 = 0$$

As expected, h_l is unable to separate the classes. On the other hand, the optimal hyperplane h_q (shown as clipped black ellipse) for the quadratic kernel is given as

$$h_q(\mathbf{x}) : \mathbf{w}^T \phi(\mathbf{x}) = 1.86x_1^2 + 1.87x_1x_2 + 0.14x_1 + 0.85x_2^2 - 1.22x_2 - 3.25 = 0$$

where $\mathbf{x} = (x_1, x_2)^T$, $\mathbf{w} = \big(1.86, 1.32, 0.099, 0.85, -0.87, -3.25\big)^T$ and $\phi(\mathbf{x}) = \Big(x_1^2, \sqrt{2}x_1x_2, \sqrt{2}x_1, x_2^2, \sqrt{2}x_2, 1\Big)^T$.

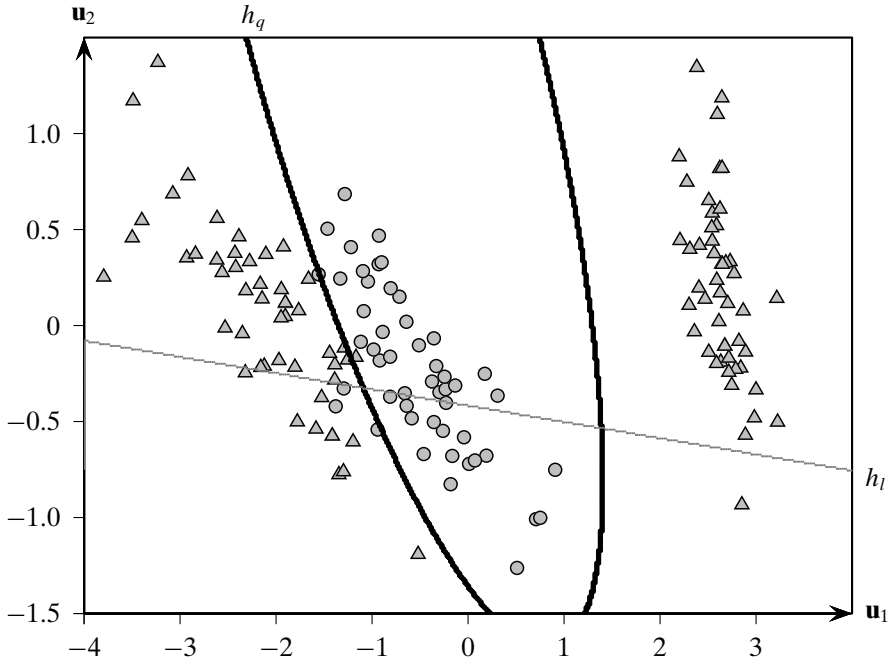


Figure 21.6. SVM dual algorithm with quadratic kernel.

The hyperplane h_q is able to separate the two classes quite well. Here we explicitly reconstructed $\mathbf{w}$ for illustration purposes; note that the last element of $\mathbf{w}$ gives the bias term $b = -3.25$.

21.5.2 Primal Solution: Newton Optimization

The dual approach is the one most commonly used to train SVMs, but it is also possible to train using the primal formulation.

Consider the primal optimization function for the linear, but nonseparable case [Eq. (21.17)]. With $\mathbf{w}, \mathbf{x}_i \in \mathbb{R}^{d+1}$ as discussed earlier, we have to minimize the objective function:

$$\min_{\mathbf{w}} J(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}(\xi_i)^k \tag{21.39}$$

subject to the linear constraints:

$$y_i\ (\mathbf{w}^T\mathbf{x}_i) \geq 1 - \xi_i \text{ and } \xi_i \geq 0 \text{ for all } i = 1, \ldots, n$$

Rearranging the above, we obtain an expression for ξ_i

$$\begin{aligned} &\xi_i \geq 1 - y_i\ (\mathbf{w}^T\mathbf{x}_i) \text{ and } \xi_i \geq 0, \text{ which implies that} \\ &\xi_i = \max\left\{0, 1 - y_i\ (\mathbf{w}^T\mathbf{x}_i)\right\} \end{aligned} \tag{21.40}$$

Plugging Eq. (21.40) into the objective function [Eq. (21.39)], we obtain

$$J(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\max\left\{0, 1 - y_i\ (\mathbf{w}^T\mathbf{x}_i)\right\}^k$$
$$= \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{y_i(\mathbf{w}^T\mathbf{x}_i)<1}\left(1 - y_i(\mathbf{w}^T\mathbf{x}_i)\right)^k \tag{21.41}$$

The last step follows from Eq. (21.40) because $\xi_i > 0$ if and only if $1 - y_i(\mathbf{w}^T\mathbf{x}_i) > 0$, that is, $y_i(\mathbf{w}^T\mathbf{x}_i) < 1$. Unfortunately, the hinge loss formulation, with $k = 1$, is not differentiable. One could use a differentiable approximation to the hinge loss, but here we describe the quadratic loss formulation.

Quadratic Loss

For quadratic loss, we have $k = 2$, and the primal objective [Eq. (21.41)] can be written as

$$J(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{y_i(\mathbf{w}^T\mathbf{x}_i)<1}\left(1 - y_i(\mathbf{w}^T\mathbf{x}_i)\right)^2$$

The gradient or the rate of change of the objective function at $\mathbf{w}$ is given as the partial derivative of $J(\mathbf{w})$ with respect to $\mathbf{w}$:

$$\nabla_{\mathbf{w}} = \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} = \mathbf{w} - 2C\left(\sum_{y_i(\mathbf{w}^T\mathbf{x}_i)<1} y_i\mathbf{x}_i\left(1 - y_i(\mathbf{w}^T\mathbf{x}_i)\right)\right)$$
$$= \mathbf{w} - 2C\left(\sum_{y_i(\mathbf{w}^T\mathbf{x}_i)<1} y_i\mathbf{x}_i\right) + 2C\left(\sum_{y_i(\mathbf{w}^T\mathbf{x}_i)<1}\mathbf{x}_i\mathbf{x}_i^T\right)\mathbf{w}$$
$$= \mathbf{w} - 2C\mathbf{v} + 2C\mathbf{S}\mathbf{w}$$

where the vector $\mathbf{v}$ and the matrix $\mathbf{S}$ are given as

$$\mathbf{v} = \sum_{y_i(\mathbf{w}^T\mathbf{x}_i)<1} y_i\mathbf{x}_i \qquad \mathbf{S} = \sum_{y_i(\mathbf{w}^T\mathbf{x}_i)<1}\mathbf{x}_i\mathbf{x}_i^T$$

Note that the matrix $\mathbf{S}$ is the scatter matrix, and the vector $\mathbf{v}$ is m times the mean of the, say m, signed points $y_i\mathbf{x}_i$ that satisfy the condition $y_i h(\mathbf{x}_i) < 1$.

The *Hessian matrix* is defined as the matrix of second-order partial derivatives of $J(\mathbf{w})$ with respect to $\mathbf{w}$, which is given as

$$\mathbf{H}_{\mathbf{w}} = \frac{\partial \nabla_{\mathbf{w}}}{\partial \mathbf{w}} = \mathbf{I} + 2C\mathbf{S}$$

Because we want to minimize the objective function $J(\mathbf{w})$, we should move in the direction opposite to the gradient. The Newton optimization update rule for $\mathbf{w}$ is given as

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t\mathbf{H}_{\mathbf{w}_t}^{-1}\nabla_{\mathbf{w}_t} \tag{21.42}$$

where $\eta_t > 0$ is a scalar value denoting the step size at iteration t. Normally one needs to use a line search method to find the optimal step size η_t, but the default value of $\eta_t = 1$ usually works for quadratic loss.

ALGORITHM 21.2. Primal SVM Algorithm: Newton Optimization, Quadratic Loss

SVM-PRIMAL $(\mathbf{D}, C, \epsilon)$:

1 **foreach** $\mathbf{x}_i \in \mathbf{D}$ **do**

2 $\quad \mathbf{x}_i \leftarrow \begin{pmatrix} \mathbf{x}_i \\ 1 \end{pmatrix}$ `// map to` $\mathbb{R}^{d+1}$

3 $t \leftarrow 0$

4 $\mathbf{w}_0 \leftarrow (0, \ldots, 0)^T$ `// initialize` $\mathbf{w}_t \in \mathbb{R}^{d+1}$

5 **repeat**

6 $\quad \mathbf{v} \leftarrow \sum_{y_i(\mathbf{w}_t^T \mathbf{x}_i) < 1} y_i \mathbf{x}_i$

7 $\quad \mathbf{S} \leftarrow \sum_{y_i(\mathbf{w}_t^T \mathbf{x}_i) < 1} \mathbf{x}_i \mathbf{x}_i^T$

8 $\quad \nabla \leftarrow (\mathbf{I} + 2C\mathbf{S})\mathbf{w}_t - 2C\mathbf{v}$ `// gradient`

9 $\quad \mathbf{H} \leftarrow \mathbf{I} + 2C\mathbf{S}$ `// Hessian`

10 $\quad \mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta_t \mathbf{H}^{-1} \nabla$ `// Newton update rule [Eq. (21.42)]`

11 $\quad t \leftarrow t + 1$

12 **until** $\|\mathbf{w}_t - \mathbf{w}_{t-1}\| \leq \epsilon$

The Newton optimization algorithm for training linear, nonseparable SVMs in the primal is given in Algorithm 21.2. The step size η_t is set to 1 by default. After computing the gradient and Hessian at $\mathbf{w}_t$ (lines 6–9), the Newton update rule is used to obtain the new weight vector $\mathbf{w}_{t+1}$ (line 10). The iterations continue until there is very little change in the weight vector. Computing $\mathbf{S}$ requires $O(nd^2)$ steps; computing the gradient ∇, the Hessian matrix $\mathbf{H}$ and updating the weight vector $\mathbf{w}_{t+1}$ takes time $O(d^2)$; and inverting the Hessian takes $O(d^3)$ operations, for a total computational complexity of $O(nd^2 + d^3)$ per iteration in the worst case.

Example 21.9 (Primal SVM). Figure 21.7 plots the hyperplanes obtained using the dual and primal approaches for the 2-dimensional Iris dataset comprising the `sepal length` versus `sepal width` attributes. We used $C = 1000$ and $\epsilon = 0.0001$ with the quadratic loss function. The dual solution h_d (gray line) and the primal solution h_p (thick black line) are essentially identical; they are as follows:

$$h_d(\mathbf{x}): \quad 7.47x_1 - 6.34x_2 - 19.89 = 0$$
$$h_p(\mathbf{x}): \quad 7.47x_1 - 6.34x_2 - 19.91 = 0$$

Primal Kernel SVMs

In the preceding discussion we considered the linear, nonseparable case for primal SVM learning. We now generalize the primal approach to learn kernel-based SVMs, again for quadratic loss.

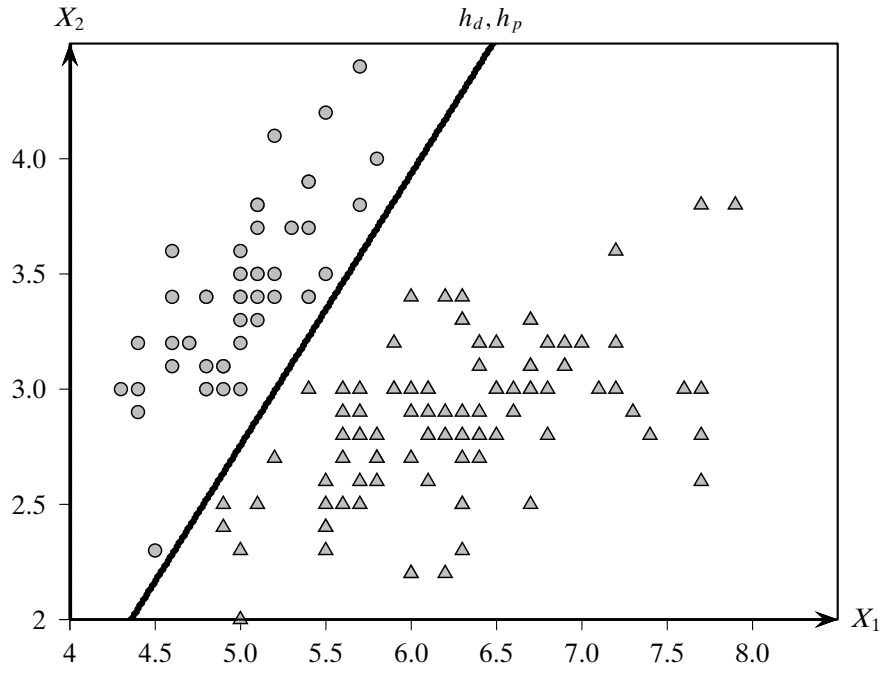


Figure 21.7. SVM primal algorithm with linear kernel.

Let ϕ denote a mapping from the input space to the feature space; each input point $\mathbf{x}_i$ is mapped to the feature point $\phi(\mathbf{x}_i)$. Let $K(\mathbf{x}_i, \mathbf{x}_j)$ denote the kernel function, and let $\mathbf{w}$ denote the weight vector in feature space. The hyperplane in feature space is then given as

$$h(\mathbf{x}) : \mathbf{w}^T \phi(\mathbf{x}) = 0$$

Using Eqs. (21.28) and (21.40), the primal objective function in feature space can be written as

$$\min_{\mathbf{w}} J(\mathbf{w}) = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^{n} L(y_i, h(\mathbf{x}_i)) \tag{21.43}$$

where $L(y_i, h(\mathbf{x}_i)) = \max\{0, 1 - y_i h(\mathbf{x}_i)\}^k$ is the loss function.

The gradient at $\mathbf{w}$ is given as

$$\nabla_{\mathbf{w}} = \mathbf{w} + C \sum_{i=1}^{n} \frac{\partial L(y_i, h(\mathbf{x}_i))}{\partial h(\mathbf{x}_i)} \cdot \frac{\partial h(\mathbf{x}_i)}{\partial \mathbf{w}}$$

where

$$\frac{\partial h(\mathbf{x}_i)}{\partial \mathbf{w}} = \frac{\partial \mathbf{w}^T \phi(\mathbf{x}_i)}{\partial \mathbf{w}} = \phi(\mathbf{x}_i)$$

At the optimal solution, the gradient vanishes, that is, $\nabla_{\mathbf{w}} = \mathbf{0}$, which yields

$$\begin{aligned}\mathbf{w} &= -C\sum_{i=1}^{n}\frac{\partial L(y_i, h(\mathbf{x}_i))}{\partial h(\mathbf{x}_i)}\cdot\phi(\mathbf{x}_i)\\ &= \sum_{i=1}^{n}\beta_i\phi(\mathbf{x}_i)\end{aligned} \tag{21.44}$$

where β_i is the coefficient of the point $\phi(\mathbf{x}_i)$ in feature space. In other words, the optimal weight vector in feature space is expressed as a linear combination of the points $\phi(\mathbf{x}_i)$ in feature space.

Using Eq. (21.44), the distance to the hyperplane in feature space can be expressed as

$$y_i h(\mathbf{x}_i) = y_i \mathbf{w}^T\phi(\mathbf{x}_i) = y_i\sum_{j=1}^{n}\beta_j K(\mathbf{x}_j, \mathbf{x}_i) = y_i\mathbf{K}_i^T\boldsymbol{\beta} \tag{21.45}$$

where $\mathbf{K} = \{K(\mathbf{x}_i, \mathbf{x}_j)\}_{i,j=1}^{n}$ is the $n \times n$ kernel matrix, $\mathbf{K}_i$ is the ith column of $\mathbf{K}$, and $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_n)^T$ is the coefficient vector.

Plugging Eqs. (21.44) and (21.45) into Eq. (21.43), with quadratic loss ($k = 2$), yields the primal kernel SVM formulation purely in terms of the kernel matrix:

$$\begin{aligned}\min_{\boldsymbol{\beta}} J(\boldsymbol{\beta}) &= \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\beta_i\beta_j K(\mathbf{x}_i, \mathbf{x}_j) + C\sum_{i=1}^{n}\max\{0, 1 - y_i\mathbf{K}_i^T\boldsymbol{\beta}\}^2\\ &= \frac{1}{2}\boldsymbol{\beta}^T\mathbf{K}\boldsymbol{\beta} + C\sum_{y_i\mathbf{K}_i^T\boldsymbol{\beta}<1}(1 - y_i\mathbf{K}_i^T\boldsymbol{\beta})^2\end{aligned}$$

The gradient of $J(\boldsymbol{\beta})$ with respect to $\boldsymbol{\beta}$ is given as

$$\begin{aligned}\nabla_{\boldsymbol{\beta}} = \frac{\partial J(\boldsymbol{\beta})}{\partial\boldsymbol{\beta}} &= \mathbf{K}\boldsymbol{\beta} - 2C\sum_{y_i\mathbf{K}_i^T\boldsymbol{\beta}<1} y_i\mathbf{K}_i(1 - y_i\mathbf{K}_i^T\boldsymbol{\beta})\\ &= \mathbf{K}\boldsymbol{\beta} + 2C\sum_{y_i\mathbf{K}_i^T\boldsymbol{\beta}<1}(\mathbf{K}_i\mathbf{K}_i^T)\,\boldsymbol{\beta} - 2C\sum_{y_i\mathbf{K}_i^T\boldsymbol{\beta}<1} y_i\mathbf{K}_i\\ &= (\mathbf{K} + 2C\mathbf{S})\,\boldsymbol{\beta} - 2C\mathbf{v}\end{aligned}$$

where the vector $\mathbf{v} \in \mathbb{R}^n$ and the matrix $\mathbf{S} \in \mathbb{R}^{n\times n}$ are given as

$$\mathbf{v} = \sum_{y_i\mathbf{K}_i^T\boldsymbol{\beta}<1} y_i\mathbf{K}_i \qquad\qquad \mathbf{S} = \sum_{y_i\mathbf{K}_i^T\boldsymbol{\beta}<1}\mathbf{K}_i\mathbf{K}_i^T$$

Furthermore, the *Hessian matrix* is given as

$$\mathbf{H}_{\boldsymbol{\beta}} = \frac{\partial\nabla_{\boldsymbol{\beta}}}{\partial\boldsymbol{\beta}} = \mathbf{K} + 2C\mathbf{S}$$

We can now minimize $J(\boldsymbol{\beta})$ by Newton optimization using the following update rule:

$$\boldsymbol{\beta}_{t+1} = \boldsymbol{\beta}_t - \eta_t\mathbf{H}_{\boldsymbol{\beta}}^{-1}\nabla_{\boldsymbol{\beta}}$$

ALGORITHM 21.3. Primal Kernel SVM Algorithm: Newton Optimization, Quadratic Loss

SVM-PRIMAL-KERNEL $(\mathbf{D}, K, C, \epsilon)$:

1 **foreach** $\mathbf{x}_i \in \mathbf{D}$ **do**

2 $\quad \mathbf{x}_i \leftarrow \begin{pmatrix} \mathbf{x}_i \\ 1 \end{pmatrix}$ `// map to` $\mathbb{R}^{d+1}$

3 $\mathbf{K} \leftarrow \{K(\mathbf{x}_i, \mathbf{x}_j)\}_{i,j=1,\ldots,n}$ `// compute kernel matrix`

4 $t \leftarrow 0$

5 $\boldsymbol{\beta}_0 \leftarrow (0, \ldots, 0)^T$ `// initialize` $\boldsymbol{\beta}_t \in \mathbb{R}^n$

6 **repeat**

7 $\quad \mathbf{v} \leftarrow \sum_{y_i(\mathbf{K}_i^T \boldsymbol{\beta}_t) < 1} y_i \mathbf{K}_i$

8 $\quad \mathbf{S} \leftarrow \sum_{y_i(\mathbf{K}_i^T \boldsymbol{\beta}_t) < 1} \mathbf{K}_i \mathbf{K}_i^T$

9 $\quad \nabla \leftarrow (\mathbf{K} + 2C\mathbf{S})\boldsymbol{\beta}_t - 2C\mathbf{v}$ `// gradient`

10 $\quad \mathbf{H} \leftarrow \mathbf{K} + 2C\mathbf{S}$ `// Hessian`

11 $\quad \boldsymbol{\beta}_{t+1} \leftarrow \boldsymbol{\beta}_t - \eta_t \mathbf{H}^{-1} \nabla$ `// Newton update rule`

12 $\quad t \leftarrow t + 1$

13 **until** $\|\boldsymbol{\beta}_t - \boldsymbol{\beta}_{t-1}\| \leq \epsilon$

Note that if $\mathbf{H}_{\boldsymbol{\beta}}$ is singular, that is, if it does not have an inverse, then we add a small *ridge* to the diagonal to regularize it. That is, we make $\mathbf{H}$ invertible as follows:

$$\mathbf{H}_{\boldsymbol{\beta}} = \mathbf{H}_{\boldsymbol{\beta}} + \lambda \mathbf{I}$$

where $\lambda > 0$ is some small positive ridge value.

Once $\boldsymbol{\beta}$ has been found, it is easy to classify any test point $\mathbf{z}$ as follows:

$$\hat{y} = \text{sign}\left(\mathbf{w}^T \phi(\mathbf{z})\right) = \text{sign}\left(\sum_{i=1}^{n} \beta_i \phi(\mathbf{x}_i)^T \phi(\mathbf{z})\right) = \text{sign}\left(\sum_{i=1}^{n} \beta_i K(\mathbf{x}_i, \mathbf{z})\right)$$

The Newton optimization algorithm for kernel SVM in the primal is given in Algorithm 21.3. The step size η_t is set to 1 by default, as in the linear case. In each iteration, the method first computes the gradient and Hessian (lines 7–10). Next, the Newton update rule is used to obtain the updated coefficient vector $\boldsymbol{\beta}_{t+1}$ (line 11). The iterations continue until there is very little change in $\boldsymbol{\beta}$. The computational complexity of the method is $O(n^3)$ per iteration in the worst case.

Example 21.10 (Primal SVM: Quadratic Kernel). Figure 21.8 plots the hyperplanes obtained using the dual and primal approaches on the Iris dataset projected onto the first two principal components. The task is to separate `iris versicolor` from the others, the same as in Example 21.8. Because a linear kernel is not suitable for this task, we employ the quadratic kernel. We further set $C = 10$ and $\epsilon = 0.0001$, with

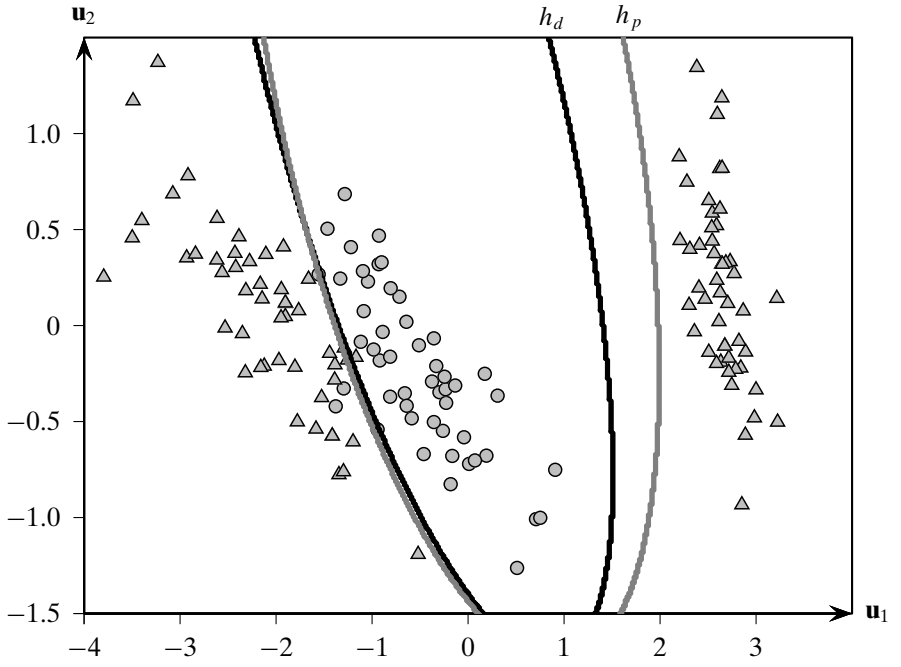


Figure 21.8. SVM quadratic kernel: dual and primal.

the quadratic loss function. The dual solution h_d (black contours) and the primal solution h_p (gray contours) are given as follows:

$$h_d(\mathbf{x}): \quad 1.4x_1^2 + 1.34x_1x_2 - 0.05x_1 + 0.66x_2^2 - 0.96x_2 - 2.66 = 0$$
$$h_p(\mathbf{x}): \quad 0.87x_1^2 + 0.64x_1x_2 - 0.5x_1 + 0.43x_2^2 - 1.04x_2 - 2.398 = 0$$

Although the solutions are not identical, they are close, especially on the left decision boundary.

21.6 FURTHER READING

The origins of support vector machines can be found in Vapnik (1982). In particular, it introduced the generalized portrait approach for constructing an optimal separating hyperplane. The use of the kernel trick for SVMs was introduced in Boser, Guyon, and Vapnik (1992), and the soft margin SVM approach for nonseparable data was proposed in Cortes and Vapnik (1995). For a good introduction to support vector machines, including implementation techniques, see Cristianini and Shawe-Taylor (2000) and Schölkopf and Smola (2002). The primal training approach described in this chapter is from Chapelle (2007).

Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). "A training algorithm for optimal margin classifiers." *In Proceedings of the 5th Annual Workshop on Computational Learning Theory,* ACM, pp. 144–152.

Chapelle, O. (2007). "Training a support vector machine in the primal." *Neural Computation*, 19 (5): 1155–1178.

Cortes, C. and Vapnik, V. (1995). "Support-vector networks." *Machine Learning*, 20 (3): 273–297.

Cristianini, N. and Shawe-Taylor, J. (2000). *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press.

Schölkopf, B. and Smola, A. J. (2002). *Learning with Kernels: Support Vector Machines, Regularization, Optimization and Beyond*. Cambridge, MA: MIT Press.

Vapnik, V. N. (1982). *Estimation of Dependences Based on Empirical Data,* vol. 41. New York: Springer-Verlag.

21.7 EXERCISES

Q1. Consider the dataset in Figure 21.9, which has points from two classes c_1 (triangles) and c_2 (circles). Answer the questions below.

- **(a)** Find the equations for the two hyperplanes h_1 and h_2.
- **(b)** Show all the support vectors for h_1 and h_2.
- **(c)** Which of the two hyperplanes is better at separating the two classes based on the margin computation?
- **(d)** Find the equation of the best separating hyperplane for this dataset, and show the corresponding support vectors. You can do this witout having to solve the Lagrangian by considering the convex hull of each class and the possible hyperplanes at the boundary of the two classes.

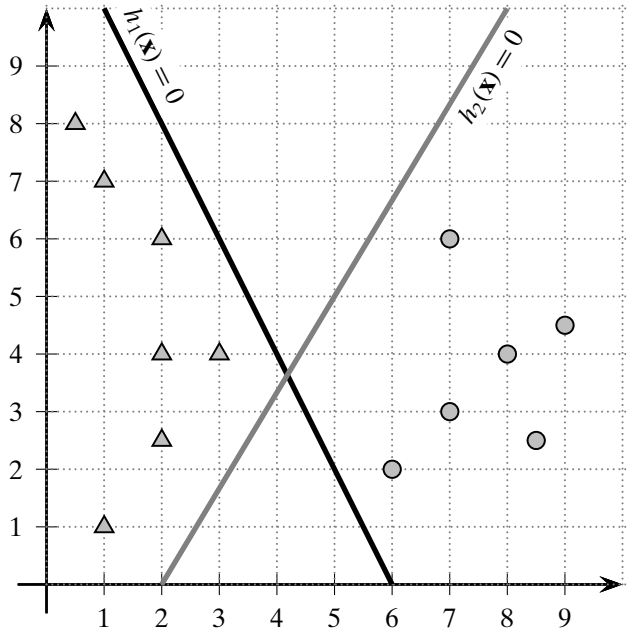


Figure 21.9. Dataset for Q1.

Table 21.2. Dataset for Q2

i	x_{i1}	x_{i2}	y_i	α_i
$\mathbf{x}_1$	4	2.9	1	0.414
$\mathbf{x}_2$	4	4	1	0
$\mathbf{x}_3$	1	2.5	−1	0
$\mathbf{x}_4$	2.5	1	−1	0.018
$\mathbf{x}_5$	4.9	4.5	1	0
$\mathbf{x}_6$	1.9	1.9	−1	0
$\mathbf{x}_7$	3.5	4	1	0.018
$\mathbf{x}_8$	0.5	1.5	−1	0
$\mathbf{x}_9$	2	2.1	−1	0.414
$\mathbf{x}_{10}$	4.5	2.5	1	0

Q2. Given the 10 points in Table 21.2, along with their classes and their Lagranian multipliers (α_i), answer the following questions:

(a) What is the equation of the SVM hyperplane $h(\mathbf{x})$?

(b) What is the distance of $\mathbf{x}_6$ from the hyperplane? Is it within the margin of the classifier?

(c) Classify the point $\mathbf{z} = (3, 3)^T$ using $h(\mathbf{x})$ from above.

CHAPTER 22 Classification Assessment

We have seen different classifiers in the preceding chapters, such as decision trees, full and naive Bayes classifiers, nearest neighbors classifier, support vector machines, and so on. In general, we may think of the classifier as a model or function M that predicts the class label $\hat{y}$ for a given input example $\mathbf{x}$:

$$\hat{y} = M(\mathbf{x})$$

where $\mathbf{x} = (x_1, x_2, \ldots, x_d)^T \in \mathbb{R}^d$ is a point in d-dimensional space and $\hat{y} \in \{c_1, c_2, \ldots, c_k\}$ is its predicted class.

To build the classification model M we need a *training set* of points along with their known classes. Different classifiers are obtained depending on the assumptions used to build the model M. For instance, support vector machines use the maximum margin hyperplane to construct M. On the other hand, the Bayes classifier directly computes the posterior probability $P(c_j|\mathbf{x})$ for each class c_j, and predicts the class of $\mathbf{x}$ as the one with the maximum posterior probability, $\hat{y} = \arg\max_{c_j} \{P(c_j|\mathbf{x})\}$. Once the model M has been trained, we assess its performance over a separate *testing set* of points for which we know the true classes. Finally, the model can be deployed to predict the class for future points whose class we typically do not know.

In this chapter we look at methods to assess a classifier, and to compare multiple classifiers. We start by defining metrics of classifier accuracy. We then discuss how to determine bounds on the expected error. We finally discuss how to assess the performance of classifiers and compare them.

22.1 CLASSIFICATION PERFORMANCE MEASURES

Let $\mathbf{D}$ be the testing set comprising n points in a d dimensional space, let $\{c_1, c_2, \ldots, c_k\}$ denote the set of k class labels, and let M be a classifier. For $\mathbf{x}_i \in \mathbf{D}$, let y_i denote its true class, and let $\hat{y}_i = M(\mathbf{x}_i)$ denote its predicted class.

Error Rate

The error rate is the fraction of incorrect predictions for the classifier over the testing set, defined as

$$Error\ Rate = \frac{1}{n}\sum_{i=1}^{n} I(y_i \neq \hat{y}_i) \tag{22.1}$$

where I is an indicator function that has the value 1 when its argument is true, and 0 otherwise. Error rate is an estimate of the probability of misclassification. The lower the error rate the better the classifier.

Accuracy

The accuracy of a classifier is the fraction of correct predictions over the testing set:

$$Accuracy = \frac{1}{n}\sum_{i=1}^{n} I(y_i = \hat{y}_i) = 1 - Error\ Rate \tag{22.2}$$

Accuracy gives an estimate of the probability of a correct prediction; thus, the higher the accuracy, the better the classifier.

Example 22.1. Figure 22.1 shows the 2-dimensional Iris dataset, with the two attributes being `sepal length` and `sepal width`. It has 150 points, and has three equal-sized classes: `Iris-setosa` (c_1; circles), `Iris-versicolor` (c_2; squares) and `Iris-virginica` (c_3; triangles). The dataset is partitioned into training and testing sets, in the ratio 80:20. Thus, the training set has 120 points (shown in light gray), and

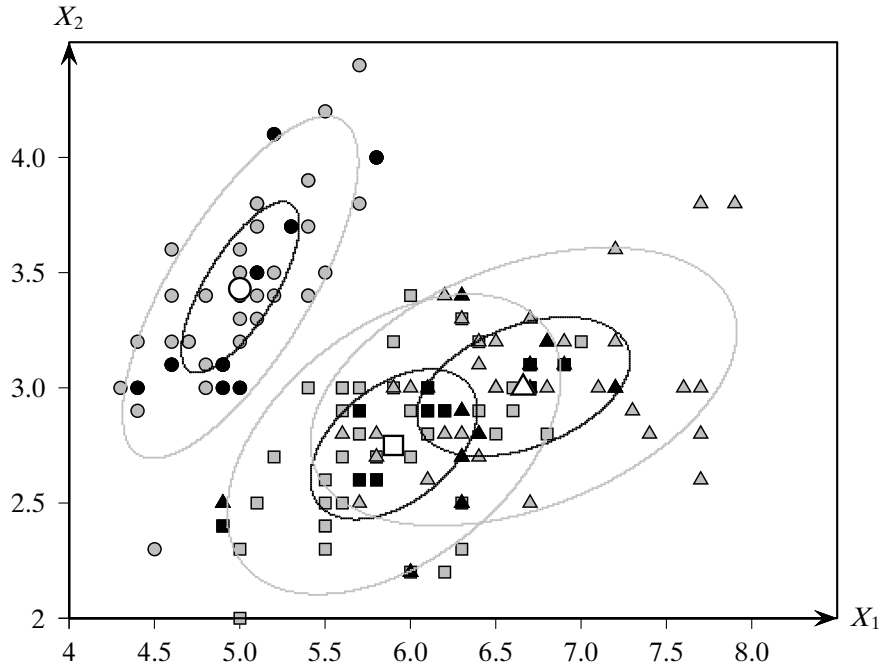


Figure 22.1. Iris dataset: three classes.

the testing set $\mathbf{D}$ has $n = 30$ points (shown in black). One can see that whereas c_1 is well separated from the other classes, c_2 and c_3 are not easy to separate. In fact, some points are labeled as both c_2 and c_3 (e.g., the point $(6, 2.2)^T$ appears twice, labeled as c_2 and c_3).

We classify the test points using the full Bayes classifier (see Chapter 18). Each class is modeled using a single normal distribution, whose mean (in white) and density contours (corresponding to one and two standard deviations) are also plotted in Figure 22.1. The classifier misclassifies 8 out of the 30 test cases. Thus, we have

$$\textit{Error Rate} = 8/30 = 0.267$$
$$\textit{Accuracy} = 22/30 = 0.733$$

22.1.1 Contingency Table–based Measures

The error rate (and, thus also the accuracy) is a global measure in that it does not explicitly consider the classes that contribute to the error. More informative measures can be obtained by tabulating the class specific agreement and disagreement between the true and predicted labels over the testing set. Let $\mathcal{D} = \{\mathbf{D}_1, \mathbf{D}_2, \ldots, \mathbf{D}_k\}$ denote a partitioning of the testing points based on their true class labels, where

$$\mathbf{D}_j = \{\mathbf{x}_i \in \mathbf{D} \mid y_i = c_j\}$$

Let $n_i = |\mathbf{D}_i|$ denote the size of true class c_i.

Let $\mathcal{R} = \{\mathbf{R}_1, \mathbf{R}_2, \ldots, \mathbf{R}_k\}$ denote a partitioning of the testing points based on the predicted labels, that is,

$$\mathbf{R}_j = \{\mathbf{x}_i \in \mathbf{D} \mid \hat{y}_i = c_j\}$$

Let $m_j = |\mathbf{R}_j|$ denote the size of the predicted class c_j.

$\mathcal{R}$ and $\mathcal{D}$ induce a $k \times k$ contingency table $\mathbf{N}$, also called a *confusion matrix*, defined as follows:

$$\mathbf{N}(i, j) = n_{ij} = \left|\mathbf{R}_i \cap \mathbf{D}_j\right| = \left|\left\{\mathbf{x}_a \in \mathbf{D} \mid \hat{y}_a = c_i \text{ and } y_a = c_j\right\}\right|$$

where $1 \le i, j \le k$. The count n_{ij} denotes the number of points with predicted class c_i whose true label is c_j. Thus, n_{ii} (for $1 \le i \le k$) denotes the number of cases where the classifier agrees on the true label c_i. The remaining counts n_{ij}, with $i \ne j$, are cases where the classifier and true labels disagree.

Accuracy/Precision

The class-specific *accuracy* or *precision* of the classifier M for class c_i is given as the fraction of correct predictions over all points predicted to be in class c_i

$$acc_i = prec_i = \frac{n_{ii}}{m_i}$$

where m_i is the number of examples predicted as c_i by classifier M. The higher the accuracy on class c_i the better the classifier.

The overall precision or accuracy of the classifier is the weighted average of the class-specific accuracy:

$$Accuracy = Precision = \sum_{i=1}^{k} \left(\frac{m_i}{n}\right) acc_i = \frac{1}{n} \sum_{i=1}^{k} n_{ii}$$

This is identical to the expression in Eq. (22.2).

Coverage/Recall

The class-specific *coverage* or *recall* of M for class c_i is the fraction of correct predictions over all points in class c_i:

$$coverage_i = recall_i = \frac{n_{ii}}{n_i}$$

where n_i is the number of points in class c_i. The higher the coverage the better the classifier.

F-measure

Often there is a trade-off between the precision and recall of a classifier. For example, it is easy to make $recall_i = 1$, by predicting all testing points to be in class c_i. However, in this case $prec_i$ will be low. On the other hand, we can make $prec_i$ very high by predicting only a few points as c_i, for instance, for those predictions where M has the most confidence, but in this case $recall_i$ will be low. Ideally, we would like both precision and recall to be high.

The *class-specific F-measure* tries to balance the precision and recall values, by computing their harmonic mean for class c_i:

$$F_i = \frac{2}{\frac{1}{prec_i} + \frac{1}{recall_i}} = \frac{2 \cdot prec_i \cdot recall_i}{prec_i + recall_i} = \frac{2\, n_{ii}}{n_i + m_i}$$

The higher the F_i value the better the classifier.

The overall *F-measure* for the classifier M is the mean of the class-specific values:

$$F = \frac{1}{k} \sum_{i=1}^{r} F_i$$

For a perfect classifier, the maximum value of the F-measure is 1.

Example 22.2. Consider the 2-dimensional Iris dataset shown in Figure 22.1. In Example 22.1 we saw that the error rate was 26.7%. However, the error rate measure does not give much information about the classes or instances that are more difficult to classify. From the class-specific normal distribution in the figure, it is clear that the Bayes classifier should perform well for c_1, but it is likely to have problems discriminating some test cases that lie close to the decision boundary between c_2 and c_3. This information is better captured by the confusion matrix obtained on the testing set, as shown in Table 22.1. We can observe that all 10 points in c_1 are classified correctly. However, only 7 out of the 10 for c_2 and 5 out of the 10 for c_3 are classified correctly.

Table 22.1. Contingency table for Iris dataset: testing set

	True			
Predicted	Iris-setosa (c_1)	Iris-versicolor (c_2)	Iris-virginica(c_3)	
Iris-setosa (c_1)	10	0	0	$m_1 = 10$
Iris-versicolor (c_2)	0	7	5	$m_2 = 12$
Iris-virginica (c_3)	0	3	5	$m_3 = 8$
	$n_1 = 10$	$n_2 = 10$	$n_3 = 10$	$n = 30$

From the confusion matrix we can compute the class-specific precision (or accuracy) values:

$$prec_1 = \frac{n_{11}}{m_1} = 10/10 = 1.0$$

$$prec_2 = \frac{n_{22}}{m_2} = 7/12 = 0.583$$

$$prec_3 = \frac{n_{33}}{m_3} = 5/8 = 0.625$$

The overall accuracy tallies with that reported in Example 22.1:

$$Accuracy = \frac{(n_{11} + n_{22} + n_{33})}{n} = \frac{(10 + 7 + 5)}{30} = 22/30 = 0.733$$

The class-specific recall (or coverage) values are given as

$$recall_1 = \frac{n_{11}}{n_1} = 10/10 = 1.0$$

$$recall_2 = \frac{n_{22}}{n_2} = 7/10 = 0.7$$

$$recall_3 = \frac{n_{33}}{n_3} = 5/10 = 0.5$$

From these we can compute the class-specific F-measure values:

$$F_1 = \frac{2 \cdot n_{11}}{(n_1 + m_1)} = 20/20 = 1.0$$

$$F_2 = \frac{2 \cdot n_{22}}{(n_2 + m_2)} = 14/22 = 0.636$$

$$F_3 = \frac{2 \cdot n_{33}}{(n_3 + m_3)} = 10/18 = 0.556$$

Thus, the overall F-measure for the classifier is

$$F = \frac{1}{3}(1.0 + 0.636 + 0.556) = \frac{2.192}{3} = 0.731$$

Table 22.2. Confusion matrix for two classes

	True Class	
Predicted Class	Positive (c_1)	Negative (c_2)
Positive (c_1)	True Positive (TP)	False Positive (FP)
Negative (c_2)	False Negative (FN)	True Negative (TN)

22.1.2 Binary Classification: Positive and Negative Class

When there are only $k = 2$ classes, we call class c_1 the positive class and c_2 the negative class. The entries of the resulting 2×2 confusion matrix, shown in Table 22.2, are given special names, as follows:

- *True Positives (TP):* The number of points that the classifier correctly predicts as positive:

$$TP = n_{11} = \left|\{\mathbf{x}_i \mid \hat{y}_i = y_i = c_1\}\right|$$

- *False Positives (FP):* The number of points the classifier predicts to be positive, which in fact belong to the negative class:

$$FP = n_{12} = \left|\{\mathbf{x}_i \mid \hat{y}_i = c_1 \text{ and } y_i = c_2\}\right|$$

- *False Negatives (FN):* The number of points the classifier predicts to be in the negative class, which in fact belong to the positive class:

$$FN = n_{21} = \left|\{\mathbf{x}_i \mid \hat{y}_i = c_2 \text{ and } y_i = c_1\}\right|$$

- *True Negatives (TN):* The number of points that the classifier correctly predicts as negative:

$$TN = n_{22} = \left|\{\mathbf{x}_i \mid \hat{y}_i = y_i = c_2\}\right|$$

Error Rate

The error rate [Eq. (22.1)] for the binary classification case is given as the fraction of mistakes (or false predictions):

$$Error\ Rate = \frac{FP + FN}{n}$$

Accuracy

The accuracy [Eq. (22.2)] is the fraction of correct predictions:

$$Accuracy = \frac{TP + TN}{n}$$

The above are global measures of classifier performance. We can obtain class-specific measures as follows.

Class-specific Precision

The precision for the positive and negative class is given as

$$prec_P = \frac{TP}{TP + FP} = \frac{TP}{m_1}$$
$$prec_N = \frac{TN}{TN + FN} = \frac{TN}{m_2}$$

where $m_i = |\mathbf{R}_i|$ is the number of points predicted by M as having class c_i.

Sensitivity: True Positive Rate

The true positive rate, also called *sensitivity*, is the fraction of correct predictions with respect to all points in the positive class, that is, it is simply the recall for the positive class

$$TPR = recall_P = \frac{TP}{TP + FN} = \frac{TP}{n_1}$$

where n_1 is the size of the positive class.

Specificity: True Negative Rate

The true negative rate, also called *specificity*, is simply the recall for the negative class:

$$TNR = specificity = recall_N = \frac{TN}{FP + TN} = \frac{TN}{n_2}$$

where n_2 is the size of the negative class.

False Negative Rate

The false negative rate is defined as

$$FNR = \frac{FN}{TP + FN} = \frac{FN}{n_1} = 1 - sensitivity$$

False Positive Rate

The false positive rate is defined as

$$FPR = \frac{FP}{FP + TN} = \frac{FP}{n_2} = 1 - specificity$$

Example 22.3. Consider the Iris dataset projected onto its first two principal components, as shown in Figure 22.2. The task is to separate `Iris-versicolor` (class c_1; in circles) from the other two Irises (class c_2; in triangles). The points from class c_1 lie in-between the points from class c_2, making this is a hard problem for (linear) classification. The dataset has been randomly split into 80% training (in gray) and 20% testing points (in black). Thus, the training set has 120 points and the testing set has $n = 30$ points.

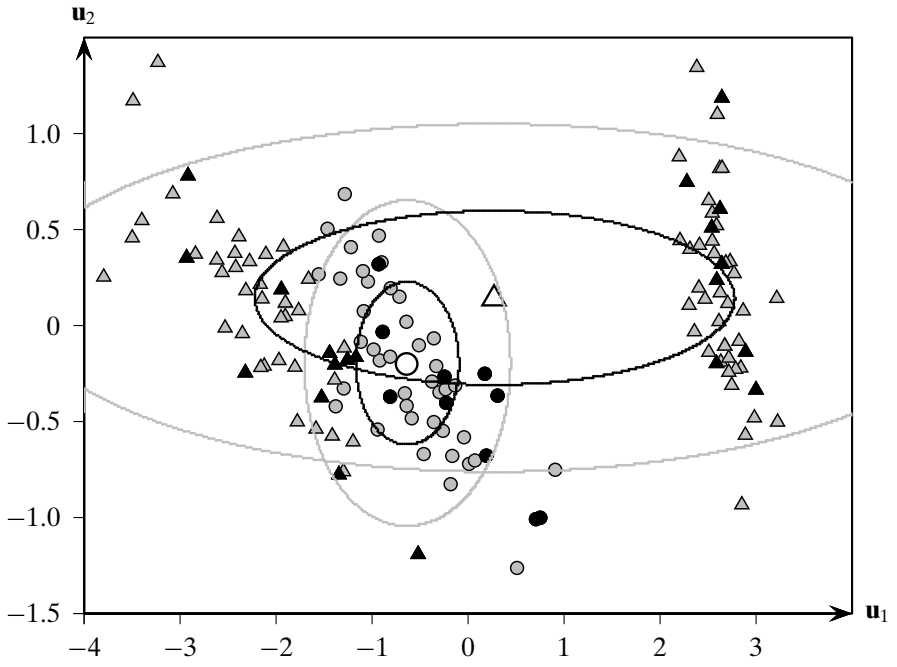


Figure 22.2. Iris principal component dataset: training and testing sets.

Applying the naive Bayes classifier (with one normal per class) on the training set yields the following estimates for the mean, covariance matrix and prior probability for each class:

$$\hat{P}(c_1) = 40/120 = 0.33 \qquad \hat{P}(c_2) = 80/120 = 0.67$$

$$\hat{\mu}_1 = \begin{pmatrix} -0.641 & -0.204 \end{pmatrix}^T \qquad \hat{\mu}_2 = \begin{pmatrix} 0.27 & 0.14 \end{pmatrix}^T$$

$$\widehat{\mathbf{\Sigma}}_1 = \begin{pmatrix} 0.29 & 0 \\ 0 & 0.18 \end{pmatrix} \qquad \widehat{\mathbf{\Sigma}}_2 = \begin{pmatrix} 6.14 & 0 \\ 0 & 0.206 \end{pmatrix}$$

The mean (in white) and the contour plot of the normal distribution for each class are also shown in the figure; the contours are shown for one and two standard deviations along each axis.

For each of the 30 testing points, we classify them using the above parameter estimates (see Chapter 18). The naive Bayes classifier misclassified 10 out of the 30 test instances, resulting in an error rate and accuracy of

$$Error\ Rate = 10/30 = 0.33$$

$$Accuracy = 20/30 = 0.67$$

The confusion matrix for this binary classification problem is shown in Table 22.3. From this table, we can compute the various performance measures:

$$prec_P = \frac{TP}{TP + FP} = \frac{7}{14} = 0.5$$

Table 22.3. Iris PC dataset: contingency table for binary classification

	True		
Predicted	Positive (c_1)	Negative (c_2)	
Positive (c_1)	$TP = 7$	$FP = 7$	$m_1 = 14$
Negative (c_2)	$FN = 3$	$TN = 13$	$m_2 = 16$
	$n_1 = 10$	$n_2 = 20$	$n = 30$

$$prec_N = \frac{TN}{TN + FN} = \frac{13}{16} = 0.8125$$

$$recall_P = sensitivity = TPR = \frac{TP}{TP + FN} = \frac{7}{10} = 0.7$$

$$recall_N = specificity = TNR = \frac{TN}{TN + FP} = \frac{13}{20} = 0.65$$

$$FNR = 1 - sensitivity = 1 - 0.7 = 0.3$$

$$FPR = 1 - specificity = 1 - 0.65 = 0.35$$

We can observe that the precision for the positive class is rather low. The true positive rate is also low, and the false positive rate is relatively high. Thus, the naive Bayes classifier is not particularly effective on this testing dataset.

22.1.3 ROC Analysis

Receiver Operating Characteristic (ROC) analysis is a popular strategy for assessing the performance of classifiers when there are two classes. ROC analysis requires that a classifier output a score value for the positive class for each point in the testing set. These scores can then be used to order points in decreasing order. For instance, we can use the posterior probability $P(c_1|\mathbf{x}_i)$ as the score, for example, for the Bayes classifiers. For SVM classifiers, we can use the signed distance from the hyperplane as the score because large positive distances are high confidence predictions for c_1, and large negative distances are very low confidence predictions for c_1 (they are, in fact, high confidence predictions for the negative class c_2).

Typically, a binary classifier chooses some positive score threshold ρ, and classifies all points with score above ρ as positive, with the remaining points classified as negative. However, such a threshold is likely to be somewhat arbitrary. Instead, ROC analysis plots the performance of the classifier over all possible values of the threshold parameter ρ. In particular, for each value of ρ, it plots the false positive rate (1-specificity) on the x-axis versus the true positive rate (sensitivity) on the y-axis. The resulting plot is called the *ROC curve* or *ROC plot* for the classifier.

Let $S(\mathbf{x}_i)$ denote the real-valued score for the positive class output by a classifier M for the point $\mathbf{x}_i$. Let the maximum and minimum score thresholds observed on testing dataset $\mathbf{D}$ be as follows:

$$\rho^{\min} = \min_i \{S(\mathbf{x}_i)\} \qquad \rho^{\max} = \max_i \{S(\mathbf{x}_i)\}$$

Table 22.4. Different cases for 2×2 confusion matrix

	True	
Predicted	Pos	Neg
Pos	0	0
Neg	*FN*	*TN*

(a) Initial: all negative

	True	
Predicted	Pos	Neg
Pos	*TP*	*FP*
Neg	0	0

(b) Final: all positive

	True	
Predicted	Pos	Neg
Pos	*TP*	0
Neg	0	*TN*

(c) Ideal classifier

Initially, we classify all points as negative. Both *TP* and *FP* are thus initially zero (as shown in Table 22.4a), resulting in *TPR* and *FPR* rates of zero, which correspond to the point $(0, 0)$ at the lower left corner in the ROC plot. Next, for each distinct value of ρ in the range $[\rho^{\min}, \rho^{\max}]$, we tabulate the set of positive points:

$$\mathbf{R}_1(\rho) = \{\mathbf{x}_i \in \mathbf{D} : S(\mathbf{x}_i) > \rho\}$$

and we compute the corresponding true and false positive rates, to obtain a new point in the ROC plot. Finally, in the last step, we classify all points as positive. Both *FN* and *TN* are thus zero (as shown in Table 22.4b), resulting in *TPR* and *FPR* values of 1. This results in the point $(1, 1)$ at the top right-hand corner in the ROC plot. An ideal classifier corresponds to the top left point $(0, 1)$, which corresponds to the case $FPR = 0$ and $TPR = 1$, that is, the classifier has no false positives, and identifies all true positives (as a consequence, it also correctly predicts all the points in the negative class). This case is shown in Table 22.4c. As such, a ROC curve indicates the extent to which the classifier ranks positive instances higher than the negative instances. An ideal classifier should score all positive points higher than any negative point. Thus, a classifier with a curve closer to the ideal case, that is, closer to the upper left corner, is a better classifier.

Area Under ROC Curve

The area under the ROC curve, abbreviated AUC, can be used as a measure of classifier performance. Because the total area of the plot is 1, the AUC lies in the interval $[0, 1]$ – the higher the better. The AUC value is essentially the probability that the classifier will rank a random positive test case higher than a random negative test instance.

ROC/AUC Algorithm

Algorithm 22.1 shows the steps for plotting a ROC curve, and for computing the area under the curve. It takes as input the testing set $\mathbf{D}$, and the classifier M. The first step is to predict the score $S(\mathbf{x}_i)$ for the positive class (c_1) for each test point $\mathbf{x}_i \in \mathbf{D}$. Next, we sort the $(S(\mathbf{x}_i), y_i)$ pairs, that is, the score and the true class pairs, in decreasing order of the scores (line 3). Initially, we set the positive score threshold $\rho = \infty$ (line 7). The for loop (line 8) examines each pair $(S(\mathbf{x}_i), y_i)$ in sorted order, and for each distinct value of the score, it sets $\rho = S(\mathbf{x}_i)$ and plots the point

$$(FPR, TPR) = \left(\frac{FP}{n_2}, \frac{TP}{n_1}\right)$$

ALGORITHM 22.1. ROC Curve and Area under the Curve

ROC-CURVE(D, M):
1 $n_1 \leftarrow \left|\{\mathbf{x}_i \in \mathbf{D} \,|\, y_i = c_1\}\right|$ // size of positive class
2 $n_2 \leftarrow \left|\{\mathbf{x}_i \in \mathbf{D} \,|\, y_i = c_2\}\right|$ // size of negative class
// classify, score, and sort all test points
3 $L \leftarrow$ sort the set $\{(S(\mathbf{x}_i), y_i) : \mathbf{x}_i \in \mathbf{D}\}$ by decreasing scores
4 $FP \leftarrow TP \leftarrow 0$
5 $FP_{prev} \leftarrow TP_{prev} \leftarrow 0$
6 $AUC \leftarrow 0$
7 $\rho \leftarrow \infty$
8 **foreach** $(S(\mathbf{x}_i), y_i) \in L$ **do**
9 **if** $\rho > S(\mathbf{x}_i)$ **then**
10 plot point $\left(\frac{FP}{n_2}, \frac{TP}{n_1}\right)$
11 $AUC \leftarrow AUC + \text{TRAPEZOID-AREA}\left(\left(\frac{FP_{prev}}{n_2}, \frac{TP_{prev}}{n_1}\right), \left(\frac{FP}{n_2}, \frac{TP}{n_1}\right)\right)$
12 $\rho \leftarrow S(\mathbf{x}_i)$
13 $FP_{prev} \leftarrow FP$
14 $TP_{prev} \leftarrow TP$
15 **if** $y_i = c_1$ **then** $TP \leftarrow TP + 1$
16 **else** $FP \leftarrow FP + 1$
17 plot point $\left(\frac{FP}{n_2}, \frac{TP}{n_1}\right)$
18 $AUC \leftarrow AUC + \text{TRAPEZOID-AREA}\left(\left(\frac{FP_{prev}}{n_2}, \frac{TP_{prev}}{n_1}\right), \left(\frac{FP}{n_2}, \frac{TP}{n_1}\right)\right)$

TRAPEZOID-AREA($(x_1, y_1), (x_2, y_2)$):
19 $b \leftarrow |x_2 - x_1|$ // base of trapezoid
20 $h \leftarrow \frac{1}{2}(y_2 + y_1)$ // average height of trapezoid
21 **return** $(b \cdot h)$

As each test point is examined, the true and false positive values are adjusted based on the true class y_i for the test point $\mathbf{x}_i$. If $y_1 = c_1$, we increment the true positives, otherwise, we increment the false positives (lines 15-16). At the end of the for loop we plot the final point in the ROC curve (line 17).

The AUC value is computed as each new point is added to the ROC plot. The algorithm maintains the previous values of the false and true positives, FP_{prev} and TP_{prev}, for the previous score threshold ρ. Given the current FP and TP values, we compute the area under the curve defined by the four points

$$(x_1, y_1) = \left(\frac{FP_{prev}}{n_2}, \frac{TP_{prev}}{n_1}\right) \qquad (x_2, y_2) = \left(\frac{FP}{n_2}, \frac{TP}{n_1}\right)$$

$$(x_1, 0) = \left(\frac{FP_{prev}}{n_2}, 0\right) \qquad (x_2, 0) = \left(\frac{FP}{n_2}, 0\right)$$

These four points define a trapezoid, whenever $x_2 > x_1$ and $y_2 > y_1$, otherwise, they define a rectangle (which may be degenerate, with zero area). The function

Table 22.5. Sorted scores and true classes

$S(\mathbf{x}_i)$	0.93	0.82	0.80	0.77	0.74	0.71	0.69	0.67	0.66	0.61
y_i	c_2	c_1	c_2	c_1	c_1	c_1	c_2	c_1	c_2	c_2

$S(\mathbf{x}_i)$	0.59	0.55	0.55	0.53	0.47	0.30	0.26	0.11	0.04	2.97e-03
y_i	c_2	c_2	c_1	c_1	c_1	c_1	c_1	c_2	c_2	c_2

$S(\mathbf{x}_i)$	1.28e-03	2.55e-07	6.99e-08	3.11e-08	3.109e-08
y_i	c_2	c_2	c_2	c_2	c_2

$S(\mathbf{x}_i)$	1.53e-08	9.76e-09	2.08e-09	1.95e-09	7.83e-10
y_i	c_2	c_2	c_2	c_2	c_2

TRAPEZOID-AREA computes the area under the trapezoid, which is given as $b \cdot h$, where $b = |x_2 - x_1|$ is the length of the base of the trapezoid and $h = \frac{1}{2}(y_2 + y_1)$ is the average height of the trapezoid.

Example 22.4. Consider the binary classification problem from Example 22.3 for the Iris principal components dataset. The test dataset $\mathbf{D}$ has $n = 30$ points, with $n_1 = 10$ points in the positive class and $n_2 = 20$ points in the negative class.

We use the naive Bayes classifier to compute the probability that each test point belongs to the positive class (c_1; `iris-versicolor`). The score of the classifier for test point $\mathbf{x}_i$ is therefore $S(\mathbf{x}_i) = P(c_1|\mathbf{x}_i)$. The sorted scores (in decreasing order) along with the true class labels are shown in Table 22.5.

The ROC curve for the test dataset is shown in Figure 22.3. Consider the positive score threshold $\rho = 0.71$. If we classify all points with a score above this value as positive, then we have the following counts for the true and false positives:

$$TP = 3 \qquad FP = 2$$

The false positive rate is therefore $\frac{FP}{n_2} = 2/20 = 0.1$, and the true positive rate is $\frac{TP}{n_1} = 3/10 = 0.3$. This corresponds to the point $(0.1, 0.3)$ in the ROC curve. Other points on the ROC curve are obtained in a similar manner as shown in Figure 22.3. The total area under the curve is 0.775.

Example 22.5 (AUC). To see why we need to account for trapezoids when computing the AUC, consider the following sorted scores, along with the true class, for some testing dataset with $n = 5$, $n_1 = 3$ and $n_2 = 2$.

$$(0.9, c_1), (0.8, c_2), (0.8, c_1), (0.8, c_1), (0.1, c_2)$$

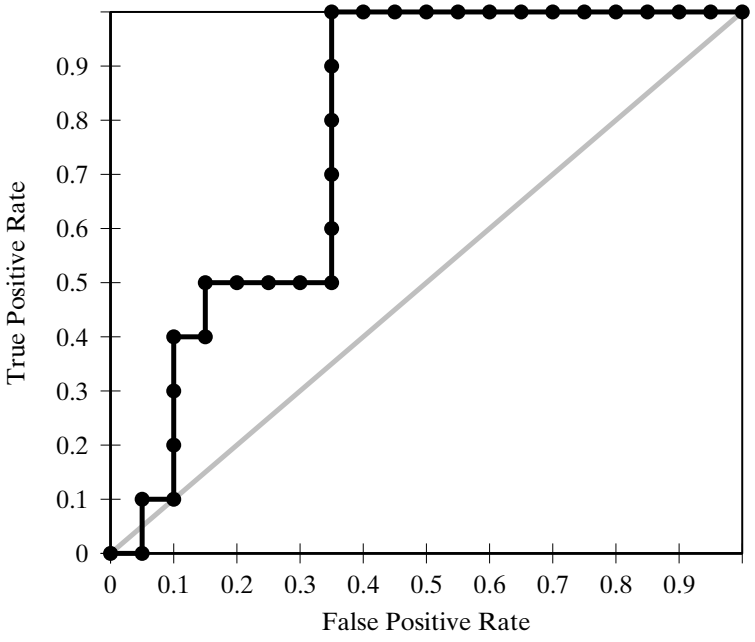


Figure 22.3. ROC plot for Iris principal components dataset. The ROC curves for the naive Bayes (black) and random (gray) classifiers are shown.

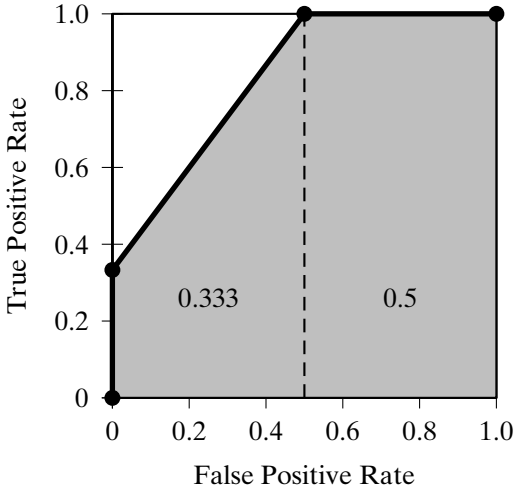


Figure 22.4. ROC plot and AUC: trapezoid region.

Algorithm 22.1 yields the following points that are added to the ROC plot, along with the running AUC:

ρ	FP	TP	(FPR, TPR)	AUC
∞	0	0	$(0, 0)$	0
0.9	0	1	$(0, 0.333)$	0
0.8	1	3	$(0.5, 1)$	0.333
0.1	2	3	$(1, 1)$	0.833

Figure 22.4 shows the ROC plot, with the shaded region representing the AUC. We can observe that a trapezoid is obtained whenever there is at least one positive and one negative point with the same score. The total AUC is 0.833, obtained as the sum of the trapezoidal region on the left (0.333) and the rectangular region on the right (0.5).

Random Classifier

It is interesting to note that a random classifier corresponds to a diagonal line in the ROC plot. To see this think of a classifier that randomly guesses the class of a point as positive half the time, and negative the other half. We then expect that half of the true positives and true negatives will be identified correctly, resulting in the point $(TPR, FPR) = (0.5, 0.5)$ for the ROC plot. If, on the other hand, the classifier guesses the class of a point as positive 90% of the time and as negative 10% of the time, then we expect 90% of the true positives and 10% of the true negatives to be labeled correctly, resulting in $TPR = 0.9$ and $FPR = 1 - TNR = 1 - 0.1 = 0.9$, that is, we get the point $(0.9, 0.9)$ in the ROC plot. In general, any fixed probability of prediction, say r, for the positive class, yields the point (r, r) in ROC space. The diagonal line thus represents the performance of a random classifier, over all possible positive class prediction thresholds r. If follows that if the ROC curve for any classifier is below the diagonal, it indicates performance worse than random guessing. For such cases, inverting the class assignment will produce a better classifier. As a consequence of the diagonal ROC curve, the AUC value for a random classifier is 0.5. Thus, if any classifier has an AUC value less than 0.5, that also indicates performance worse than random.

Example 22.6. In addition to the ROC curve for the naive Bayes classifier, Figure 22.3 also shows the ROC plot for the random classifier (the diagonal line in gray). We can see that the ROC curve for the naive Bayes classifier is much better than random. Its AUC value is 0.775, which is much better than the 0.5 AUC for a random classifier. However, at the very beginning naive Bayes performs worse than the random classifier because the highest scored point is from the negative class. As such, the ROC curve should be considered as a discrete approximation of a smooth curve that would be obtained for a very large (infinite) testing dataset.

Class Imbalance

It is worth remarking that ROC curves are insensitive to class skew. This is because the TPR, interpreted as the probability of predicting a positive point as positive, and the FPR, interpreted as the probability of predicting a negative point as positive, do not depend on the ratio of the positive to negative class size. This is a desirable property, since the ROC curve will essentially remain the same whether the classes are balanced (have relatively the same number of points) or skewed (when one class has many more points than the other).

22.2 CLASSIFIER EVALUATION

In this section we discuss how to evaluate a classifier M using some performance measure θ. Typically, the input dataset $\mathbf{D}$ is randomly split into a disjoint training set and testing set. The training set is used to learn the model M, and the testing set is used to evaluate the measure θ. However, how confident can we be about the classification performance? The results may be due to an artifact of the random split, for example, by random chance the testing set may have particularly easy (or hard) to classify points, leading to good (or poor) classifier performance. As such, a fixed, pre-defined partitioning of the dataset is not a good strategy for evaluating classifiers. Also note that, in general, $\mathbf{D}$ is itself a d-dimensional multivariate random sample drawn from the true (unknown) joint probability density function $f(\mathbf{x})$ that represents the population of interest. Ideally, we would like to know the expected value $E[\theta]$ of the performance measure over all possible testing sets drawn from f. However, because f is unknown, we have to estimate $E[\theta]$ from $\mathbf{D}$. Cross-validation and resampling are two common approaches to compute the expected value and variance of a given performance measure; we discuss these methods in the following sections.

22.2.1 K-fold Cross-Validation

Cross-validation divides the dataset $\mathbf{D}$ into K equal-sized parts, called *folds*, namely $\mathbf{D}_1, \mathbf{D}_2, \ldots, \mathbf{D}_K$. Each fold $\mathbf{D}_i$ is, in turn, treated as the testing set, with the remaining folds comprising the training set $\mathbf{D} \setminus \mathbf{D}_i = \bigcup_{j \neq i} \mathbf{D}_j$. After training the model M_i on $\mathbf{D} \setminus \mathbf{D}_i$, we assess its performance on the testing set $\mathbf{D}_i$ to obtain the i-th estimate θ_i. The expected value of the performance measure can then be estimated as

$$\hat{\mu}_\theta = E[\theta] = \frac{1}{K} \sum_{i=1}^{K} \theta_i \tag{22.3}$$

and its variance as

$$\hat{\sigma}^2_\theta = \frac{1}{K} \sum_{i=1}^{K} (\theta_i - \hat{\mu}_\theta)^2 \tag{22.4}$$

Algorithm 22.2 shows the pseudo-code for K-fold cross-validation. After randomly shuffling the dataset $\mathbf{D}$, we partition it into K equal folds (except for possibly the last one). Next, each fold $\mathbf{D}_i$ is used as the testing set on which we assess the performance θ_i of the classifier M_i trained on $\mathbf{D} \setminus \mathbf{D}_i$. The estimated mean and variance of θ can then be reported. Note that the K-fold cross-validation can be repeated multiple times; the initial random shuffling ensures that the folds are different each time.

Usually K is chosen to be 5 or 10. The special case, when $K = n$, is called *leave-one-out* cross-validation, where the testing set comprises a single point and the remaining data is used for training purposes.

ALGORITHM 22.2. *K*-fold Cross-Validation

```
CROSS-VALIDATION(K, D):
1 D ← randomly shuffle D
2 {D_1, D_2, ..., D_K} ← partition D in K equal parts
3 foreach i ∈ [1, K] do
4     M_i ← train classifier on D \ D_i
5     θ_i ← assess M_i on D_i
6 μ̂_θ = (1/K) Σ_{i=1}^{K} θ_i
7 σ̂²_θ = (1/K) Σ_{i=1}^{K} (θ_i − μ̂_θ)²
8 return μ̂_θ, σ̂²_θ
```

Example 22.7. Consider the 2-dimensional Iris dataset from Example 22.1 with $k = 3$ classes. We assess the error rate of the full Bayes classifier via 5-fold cross-validation, obtaining the following error rates when testing on each fold:

$$\theta_1 = 0.267 \qquad \theta_2 = 0.133 \qquad \theta_3 = 0.233 \qquad \theta_4 = 0.367 \qquad \theta_5 = 0.167$$

Using Eqs. (22.3) and (22.4), the mean and variance for the error rate are as follows:

$$\hat{\mu}_\theta = \frac{1.167}{5} = 0.233 \qquad \hat{\sigma}^2_\theta = 0.00833$$

We can repeat the whole cross-validation approach multiple times, with a different permutation of the input points, and then we can compute the mean of the average error rate, and mean of the variance. Performing ten 5-fold cross-validation runs for the Iris dataset results in the mean of the expected error rate as 0.232, and the mean of the variance as 0.00521, with the variance in both these estimates being less than 10^{-3}.

22.2.2 Bootstrap Resampling

Another approach to estimate the expected performance of a classifier is to use the bootstrap resampling method. Instead of partitioning the input dataset $\mathbf{D}$ into disjoint folds, the bootstrap method draws K random samples of size n *with replacement* from $\mathbf{D}$. Each sample $\mathbf{D}_i$ is thus the same size as $\mathbf{D}$, and has several repeated points. Consider the probability that a point $\mathbf{x}_j \in \mathbf{D}$ is not selected for the ith bootstrap sample $\mathbf{D}_i$. Due to sampling with replacement, the probability that a given point is selected is given as $p = \frac{1}{n}$, and thus the probability that it is not selected is

$$q = 1 - p = \left(1 - \frac{1}{n}\right)$$

Because $\mathbf{D}_i$ has n points, the probability that $\mathbf{x}_j$ is not selected even after n tries is given as

$$P(\mathbf{x}_j \notin \mathbf{D}_i) = q^n = \left(1 - \frac{1}{n}\right)^n \simeq e^{-1} = 0.368$$

ALGORITHM 22.3. Bootstrap Resampling Method

BOOTSTRAP-RESAMPLING(K, $\mathbf{D}$):
1 **for** $i \in [1, K]$ **do**
2 $\quad \mathbf{D}_i \leftarrow$ sample of size n with replacement from $\mathbf{D}$
3 $\quad M_i \leftarrow$ train classifier on $\mathbf{D}_i$
4 $\quad \theta_i \leftarrow$ assess M_i on $\mathbf{D}$
5 $\hat{\mu}_\theta = \frac{1}{K}\sum_{i=1}^{K} \theta_i$
6 $\hat{\sigma}^2_\theta = \frac{1}{K}\sum_{i=1}^{K} (\theta_i - \hat{\mu}_\theta)^2$
7 **return** $\hat{\mu}_\theta, \hat{\sigma}^2_\theta$

On the other hand, the probability that $\mathbf{x}_j \in \mathbf{D}$ is given as

$$P(\mathbf{x}_j \in \mathbf{D}_i) = 1 - P(\mathbf{x}_j \notin \mathbf{D}_i) = 1 - 0.368 = 0.632$$

This means that each bootstrap sample contains approximately 63.2% of the points from $\mathbf{D}$.

The bootstrap samples can be used to evaluate the classifier by training it on each of samples $\mathbf{D}_i$ and then using the full input dataset $\mathbf{D}$ as the testing set, as shown in Algorithm 22.3. The expected value and variance of the performance measure θ can be obtained using Eqs. (22.3) and (22.4). However, it should be borne in mind that the estimates will be somewhat optimistic owing to the fairly large overlap between the training and testing datasets (63.2%). The cross-validation approach does not suffer from this limitation because it keeps the training and testing sets disjoint.

Example 22.8. We continue with the Iris dataset from Example 22.7. However, we now apply bootstrap sampling to estimate the error rate for the full Bayes classifier, using $K = 50$ samples. The sampling distribution of error rates is shown in Figure 22.5.

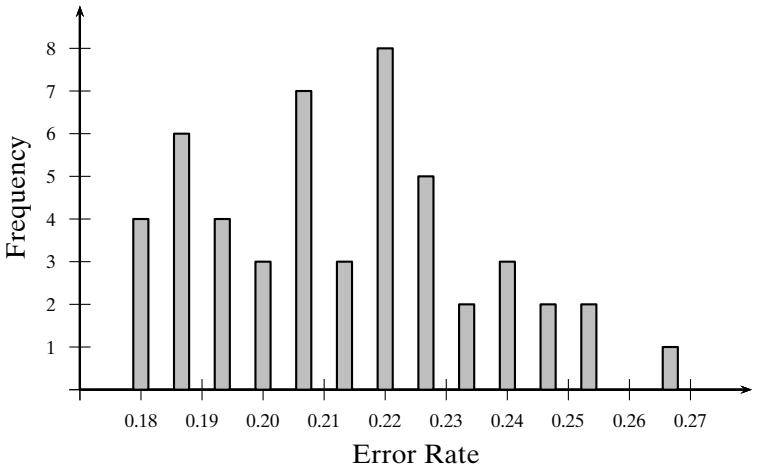


Figure 22.5. Sampling distribution of error rates.

The expected value and variance of the error rate are

$$\hat{\mu}_\theta = 0.213$$
$$\hat{\sigma}^2_\theta = 4.815 \times 10^{-4}$$

Due to the overlap between the training and testing sets, the estimates are more optimistic (i.e., lower) compared to those obtained via cross-validation in Example 22.7, where we had $\hat{\mu}_\theta = 0.233$ and $\hat{\sigma}^2_\theta = 0.00833$.

22.2.3 Confidence Intervals

Having estimated the expected value and variance for a chosen performance measure, we would like to derive confidence bounds on how much the estimate may deviate from the true value.

To answer this question we make use of the central limit theorem, which states that the sum of a large number of independent and identically distributed (IID) random variables has approximately a normal distribution, regardless of the distribution of the individual random variables. More formally, let $\theta_1, \theta_2, \ldots, \theta_K$ be a sequence of IID random variables, representing, for example, the error rate or some other performance measure over the K-folds in cross-validation or K bootstrap samples. Assume that each θ_i has a finite mean $E[\theta_i] = \mu$ and finite variance $var(\theta_i) = \sigma^2$.

Let $\hat{\mu}$ denote the sample mean:

$$\hat{\mu} = \frac{1}{K}(\theta_1 + \theta_2 + \cdots + \theta_K)$$

By linearity of expectation, we have

$$E[\hat{\mu}] = E\left[\frac{1}{K}(\theta_1 + \theta_2 + \cdots + \theta_K)\right] = \frac{1}{K}\sum_{i=1}^{K} E[\theta_i] = \frac{1}{K}(K\mu) = \mu$$

Utilizing the linearity of variance for independent random variables, and noting that $var(aX) = a^2 \cdot var(X)$ for $a \in \mathbb{R}$, the variance of $\hat{\mu}$ is given as

$$var(\hat{\mu}) = var\left(\frac{1}{K}(\theta_1 + \theta_2 + \cdots + \theta_K)\right) = \frac{1}{K^2}\sum_{i=1}^{K} var(\theta_i) = \frac{1}{K^2}\left(K\sigma^2\right) = \frac{\sigma^2}{K}$$

Thus, the standard deviation of $\hat{\mu}$ is given as

$$std(\hat{\mu}) = \sqrt{var(\hat{\mu})} = \frac{\sigma}{\sqrt{K}}$$

We are interested in the distribution of the z-score of $\hat{\mu}$, which is itself a random variable

$$Z_K = \frac{\hat{\mu} - E[\hat{\mu}]}{std(\hat{\mu})} = \frac{\hat{\mu} - \mu}{\frac{\sigma}{\sqrt{K}}} = \sqrt{K}\left(\frac{\hat{\mu} - \mu}{\sigma}\right)$$

Z_K specifies the deviation of the estimated mean from the true mean in terms of its standard deviation. The central limit theorem states that as the sample size increases,

the random variable Z_K *converges in distribution* to the standard normal distribution (which has mean 0 and variance 1). That is, as $K \to \infty$, for any $x \in \mathbb{R}$, we have

$$\lim_{K\to\infty} P(Z_K \leq x) = \Phi(x)$$

where $\Phi(x)$ is the cumulative distribution function for the standard normal density function $f(x|0,1)$. Let $z_{\alpha/2}$ denote the z-score value that encompasses $\alpha/2$ of the probability mass for a standard normal distribution, that is,

$$P(0 \leq Z_K \leq z_{\alpha/2}) = \Phi(z_{\alpha/2}) - \Phi(0) = \alpha/2$$

then, because the normal distribution is symmetric about the mean, we have

$$\lim_{K\to\infty} P(-z_{\alpha/2} \leq Z_K \leq z_{\alpha/2}) = 2 \cdot P(0 \leq Z_K \leq z_{\alpha/2}) = \alpha \tag{22.5}$$

Note that

$$\begin{aligned} -z_{\alpha/2} \leq Z_K \leq z_{\alpha/2} &\implies -z_{\alpha/2} \leq \sqrt{K}\left(\frac{\hat{\mu}-\mu}{\sigma}\right) \leq z_{\alpha/2} \\ &\implies -z_{\alpha/2}\frac{\sigma}{\sqrt{K}} \leq \hat{\mu}-\mu \leq z_{\alpha/2}\frac{\sigma}{\sqrt{K}} \\ &\implies \left(\hat{\mu}-z_{\alpha/2}\frac{\sigma}{\sqrt{K}}\right) \leq \mu \leq \left(\hat{\mu}+z_{\alpha/2}\frac{\sigma}{\sqrt{K}}\right) \end{aligned}$$

Substituting the above into Eq. (22.5) we obtain bounds on the value of the true mean μ in terms of the estimated value $\hat{\mu}$, that is,

$$\lim_{K\to\infty} P\left(\hat{\mu}-z_{\alpha/2}\frac{\sigma}{\sqrt{K}} \leq \mu \leq \hat{\mu}+z_{\alpha/2}\frac{\sigma}{\sqrt{K}}\right) = \alpha \tag{22.6}$$

Thus, for any given level of confidence α, we can compute the probability that the true mean μ lies in the α% confidence interval $\left(\hat{\mu}-z_{\alpha/2}\frac{\sigma}{\sqrt{K}}, \hat{\mu}+z_{\alpha/2}\frac{\sigma}{\sqrt{K}}\right)$. In other words, even though we do not know the true mean μ, we can obtain a high-confidence estimate of the interval within which it must lie (e.g., by setting $\alpha = 0.95$ or $\alpha = 0.99$).

Unknown Variance

The analysis above assumes that we know the true variance σ^2, which is generally not the case. However, we can replace σ^2 by the sample variance

$$\hat{\sigma}^2 = \frac{1}{K}\sum_{i=1}^{K}(\theta_i - \hat{\mu})^2 \tag{22.7}$$

because $\hat{\sigma}^2$ is a *consistent* estimator for σ^2, that is, as $K \to \infty$, $\hat{\sigma}^2$ converges with probability 1, also called *converges almost surely*, to σ^2. The central limit theorem then states that the random variable Z_K^* defined below converges in distribution to the standard normal distribution:

$$Z_K^* = \sqrt{K}\left(\frac{\hat{\mu}-\mu}{\hat{\sigma}}\right) \tag{22.8}$$

and thus, we have

$$\lim_{K\to\infty} P\left(\hat{\mu} - z_{\alpha/2}\frac{\hat{\sigma}}{\sqrt{K}} \le \mu \le \hat{\mu} + z_{\alpha/2}\frac{\hat{\sigma}}{\sqrt{K}}\right) = \alpha \tag{22.9}$$

In other words, we say that $\left(\hat{\mu} - z_{\alpha/2}\frac{\hat{\sigma}}{\sqrt{K}},\ \hat{\mu} + z_{\alpha/2}\frac{\hat{\sigma}}{\sqrt{K}}\right)$ is the α% confidence interval for μ.

Example 22.9. Consider Example 22.7, where we applied 5-fold cross-validation ($K = 5$) to assess the error rate of the full Bayes classifier. The estimated expected value and variance for the error rate were as follows:

$$\hat{\mu}_\theta = 0.233 \qquad \hat{\sigma}_\theta^2 = 0.00833 \qquad \hat{\sigma}_\theta = \sqrt{0.00833} = 0.0913$$

Let $\alpha = 0.95$ be the confidence value. It is known that the standard normal distribution has 95% of the probability density within $z_{\alpha/2} = 1.96$ standard deviations from the mean. Thus, in the limit of large sample size, we have

$$P\left(\mu \in \left(\hat{\mu}_\theta - z_{\alpha/2}\frac{\hat{\sigma}_\theta}{\sqrt{K}}, \hat{\mu}_\theta + z_{\alpha/2}\frac{\hat{\sigma}_\theta}{\sqrt{K}}\right)\right) = 0.95$$

Because $z_{\alpha/2}\frac{\hat{\sigma}_\theta}{\sqrt{K}} = \frac{1.96\times 0.0913}{\sqrt{5}} = 0.08$, we have

$$P\Big(\mu \in (0.233 - 0.08, 0.233 + 0.08)\Big) = P\Big(\mu \in (0.153, 0.313)\Big) = 0.95$$

Put differently, with 95% confidence, the true expected error rate lies in the interval $(0.153, 0.313)$.

If we want greater confidence, for example, for $\alpha = 0.99$, then the corresponding z-score value is $z_{\alpha/2} = 2.58$, and thus $z_{\alpha/2}\frac{\hat{\sigma}_\theta}{\sqrt{K}} = \frac{2.58\times 0.0913}{\sqrt{5}} = 0.105$. The 99% confidence interval for μ is therefore wider $(0.128, 0.338)$.

Nevertheless, $K = 5$ is not a large sample size, and thus the above confidence intervals are not that reliable.

Small Sample Size

The confidence interval in Eq. (22.9) applies only when the sample size $K \to \infty$. We would like to obtain more precise confidence intervals for small samples. Consider the random variables V_i, for $i = 1, \ldots, K$, defined as

$$V_i = \frac{\theta_i - \hat{\mu}}{\sigma}$$

Further, consider the sum of their squares:

$$S = \sum_{i=1}^{K} V_i^2 = \sum_{i=1}^{K}\left(\frac{\theta_i - \hat{\mu}}{\sigma}\right)^2 = \frac{1}{\sigma^2}\sum_{i=1}^{K}(\theta_i - \hat{\mu})^2 = \frac{K\hat{\sigma}^2}{\sigma^2} \tag{22.10}$$

The last step follows from the definition of sample variance in Eq. (22.7).

If we assume that the V_i's are IID with the standard normal distribution, then the sum S follows a chi-squared distribution with $K - 1$ degrees of freedom, denoted

$\chi^2(K-1)$, since S is the sum of the squares of K random variables V_i. There are only $K-1$ degrees of freedom because each V_i depends on $\hat{\mu}$ and the sum of the θ_i's is thus fixed.

Consider the random variable Z_K^* in Eq. (22.8). We have,

$$Z_K^* = \sqrt{K}\left(\frac{\hat{\mu}-\mu}{\hat{\sigma}}\right) = \left(\frac{\hat{\mu}-\mu}{\hat{\sigma}/\sqrt{K}}\right)$$

Dividing the numerator and denominator in the expression above by $\sigma/\sqrt{K}$, we get

$$Z_K^* = \left(\frac{\hat{\mu}-\mu}{\sigma/\sqrt{K}} \Big/ \frac{\hat{\sigma}/\sqrt{K}}{\sigma/\sqrt{K}}\right) = \left(\frac{\frac{\hat{\mu}-\mu}{\sigma/\sqrt{K}}}{\hat{\sigma}/\sigma}\right) = \frac{Z_K}{\sqrt{S/K}} \tag{22.11}$$

The last step follows from Eq. (22.10) because

$$S = \frac{K\hat{\sigma}^2}{\sigma^2} \text{ implies that } \frac{\hat{\sigma}}{\sigma} = \sqrt{S/K}$$

Assuming that Z_K follows a standard normal distribution, and noting that S follows a chi-squared distribution with $K-1$ degrees of freedom, then the distribution of Z_K^* is precisely the Student's t distribution with $K-1$ degrees of freedom. Thus, in the small sample case, instead of using the standard normal density to derive the confidence interval, we use the t distribution. In particular, we choose the value $t_{\alpha/2,K-1}$ such that the cumulative t distribution function with $K-1$ degrees of freedom encompasses $\alpha/2$ of the probability mass, that is,

$$P(0 \le Z_K^* \le t_{\alpha/2,K-1}) = T_{K-1}(t_{\alpha/2}) - T_{K-1}(0) = \alpha/2$$

where T_{K-1} is the cumulative distribution function for the Student's t distribution with $K-1$ degrees of freedom. Because the t distribution is symmetric about the mean, we have

$$P\left(\hat{\mu} - t_{\alpha/2,K-1}\frac{\hat{\sigma}}{\sqrt{K}} \le \mu \le \hat{\mu} + t_{\alpha/2,K-1}\frac{\hat{\sigma}}{\sqrt{K}}\right) = \alpha \tag{22.12}$$

The α% confidence interval for the true mean μ is thus

$$\left(\hat{\mu} - t_{\alpha/2,K-1}\frac{\hat{\sigma}}{\sqrt{K}} \le \mu \le \hat{\mu} + t_{\alpha/2,K-1}\frac{\hat{\sigma}}{\sqrt{K}}\right)$$

Note the dependence of the interval on both α and the sample size K.

Figure 22.6 shows the t distribution density function for different values of K. It also shows the standard normal density function. We can observe that the t distribution has more probability concentrated in its tails compared to the standard normal distribution. Further, as K increases, the t distribution very rapidly converges in distribution to the standard normal distribution, consistent with the large sample case. Thus, for large samples, we may use the usual $z_{\alpha/2}$ threshold.

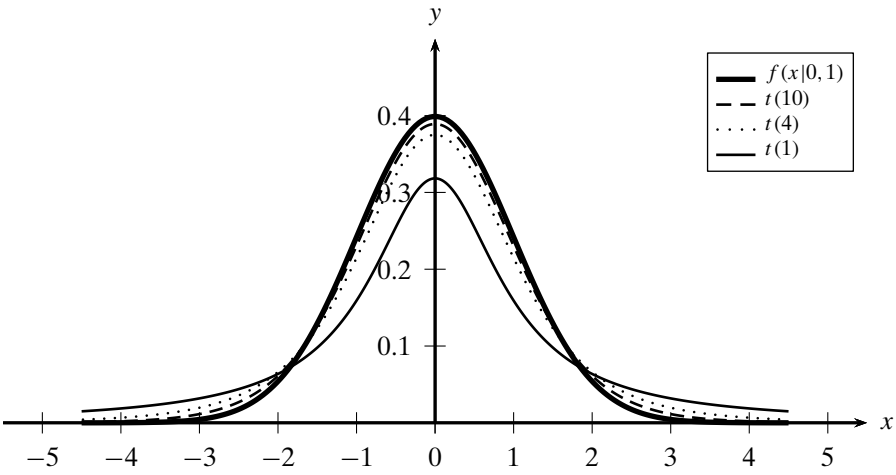


Figure 22.6. Student's t distribution: K degrees of freedom. The thick solid line is standard normal distribution.

Example 22.10. Consider Example 22.9. For 5-fold cross-validation, the estimated mean error rate is $\hat{\mu}_\theta = 0.233$, and the estimated variance is $\hat{\sigma}_\theta = 0.0913$.

Due to the small sample size ($K = 5$), we can get a better confidence interval by using the t distribution. For $K - 1 = 4$ degrees of freedom, for $\alpha = 0.95$, we use the quantile function for the Student's t-distribution to obtain $t_{\alpha/2,K-1} = 2.776$. Thus,

$$t_{\alpha/2,K-1}\frac{\hat{\sigma}_\theta}{\sqrt{K}} = 2.776 \times \frac{0.0913}{\sqrt{5}} = 0.113$$

The 95% confidence interval is therefore

$$(0.233 - 0.113, 0.233 + 0.113) = (0.12, 0.346)$$

which is much wider than the overly optimistic confidence interval $(0.153, 0.313)$ obtained for the large sample case in Example 22.9.

For $\alpha = 0.99$, we have $t_{\alpha/2,K-1} = 4.604$, and thus

$$t_{\alpha/2,K-1}\frac{\hat{\sigma}_\theta}{\sqrt{K}} = 4.604 \times \frac{0.0913}{\sqrt{5}} = 0.188$$

and the 99% confidence interval is

$$(0.233 - 0.188, 0.233 + 0.188) = (0.045, 0.421)$$

This is also much wider than the 99% confidence interval $(0.128, 0.338)$ obtained for the large sample case in Example 22.9.

22.2.4 Comparing Classifiers: Paired t-Test

In this section we look at a method that allows us to test for a significant difference in the classification performance of two alternative classifiers, M^A and M^B. We want to assess which of them has a superior classification performance on a given dataset $\mathbf{D}$.

Following the evaluation methodology above, we can apply K-fold cross-validation (or bootstrap resampling) and tabulate their performance over each of the K folds, with identical folds for both classifiers. That is, we perform a *paired test*, with both classifiers trained and tested on the same data. Let $\theta_1^A, \theta_2^A, \ldots, \theta_K^A$ and $\theta_1^B, \theta_2^B, \ldots, \theta_K^B$ denote the performance values for M_A and M_B, respectively. To determine if the two classifiers have different or similar performance, define the random variable δ_i as the difference in their performance on the ith dataset:

$$\delta_i = \theta_i^A - \theta_i^B$$

Now consider the estimates for the expected difference and the variance of the differences:

$$\hat{\mu}_\delta = \frac{1}{K}\sum_{i=1}^{K}\delta_i \qquad\qquad \hat{\sigma}_\delta^2 = \frac{1}{K}\sum_{i=1}^{K}(\delta_i - \hat{\mu}_\delta)^2$$

We can set up a hypothesis testing framework to determine if there is a statistically significant difference between the performance of M^A and M^B. The null hypothesis H_0 is that their performance is the same, that is, the true expected difference is zero, whereas the alternative hypothesis H_a is that they are not the same, that is, the true expected difference μ_δ is not zero:

$$H_0: \quad \mu_\delta = 0 \qquad\qquad H_a: \quad \mu_\delta \neq 0$$

Let us define the z-score random variable for the estimated expected difference as

$$Z_\delta^* = \sqrt{K}\left(\frac{\hat{\mu}_\delta - \mu_\delta}{\hat{\sigma}_\delta}\right)$$

Following a similar argument as in Eq. (22.11), Z_δ^* follows a t distribution with $K-1$ degrees of freedom. However, under the null hypothesis we have $\mu_\delta = 0$, and thus

$$Z_\delta^* = \frac{\sqrt{K}\hat{\mu}_\delta}{\hat{\sigma}_\delta} \sim t_{K-1}$$

where the notation $Z_\delta^* \sim t_{K-1}$ means that Z_δ^* follows the t distribution with $K-1$ degrees of freedom.

Given a desired confidence level α, we conclude that

$$P\left(-t_{\alpha/2,K-1} \leq Z_\delta^* \leq t_{\alpha/2,K-1}\right) = \alpha$$

Put another way, if $Z_\delta^* \notin \left(-t_{\alpha/2,K-1}, t_{\alpha/2,K-1}\right)$, then we may reject the null hypothesis with α% confidence. In this case, we conclude that there is a significant difference between the performance of M^A and M^B. On the other hand, if Z_δ^* does lie in the above confidence interval, then we accept the null hypothesis that both M^A and M^B have essentially the same performance. The pseudo-code for the paired t-test is shown in Algorithm 22.4.

ALGORITHM 22.4. Paired *t*-Test via Cross-Validation

PAIRED *t*-TEST(α, K, $\mathbf{D}$):
1 $\mathbf{D} \leftarrow$ randomly shuffle $\mathbf{D}$
2 $\{\mathbf{D}_1, \mathbf{D}_2, \ldots, \mathbf{D}_K\} \leftarrow$ partition $\mathbf{D}$ in K equal parts
3 **foreach** $i \in [1, K]$ **do**
4 $\quad M_i^A, M_i^B \leftarrow$ train the two different classifiers on $\mathbf{D} \setminus \mathbf{D}_i$
5 $\quad \theta_i^A, \theta_i^B \leftarrow$ assess M_i^A and M_i^B on $\mathbf{D}_i$
6 $\quad \delta_i = \theta_i^A - \theta_i^B$
7 $\hat{\mu}_\delta = \frac{1}{K}\sum_{i=1}^{K} \delta_i$
8 $\hat{\sigma}_\delta^2 = \frac{1}{K}\sum_{i=1}^{K} (\delta_i - \hat{\mu}_\delta)^2$
9 $Z_\delta^* = \frac{\sqrt{K}\hat{\mu}_\delta}{\hat{\sigma}_\delta}$
10 **if** $Z_\delta^* \in \left(-t_{\alpha/2,K-1}, t_{\alpha/2,K-1}\right)$ **then**
11 $\quad$ Accept H_0; both classifiers have similar performance
12 **else**
13 $\quad$ Reject H_0; classifiers have significantly different performance

Example 22.11. Consider the 2-dimensional Iris dataset from Example 22.1, with $k = 3$ classes. We compare the naive Bayes (M^A) with the full Bayes (M^B) classifier via cross-validation using $K = 5$ folds. Using error rate as the performance measure, we obtain the following values for the error rates and their difference over each of the K folds:

i	1	2	3	4	5
θ_i^A	0.233	0.267	0.1	0.4	0.3
θ_i^B	0.2	0.2	0.167	0.333	0.233
δ_i	0.033	0.067	−0.067	0.067	0.067

The estimated expected difference and variance of the differences are

$$\hat{\mu}_\delta = \frac{0.167}{5} = 0.033 \qquad \hat{\sigma}_\delta^2 = 0.00333 \qquad \hat{\sigma}_\delta = \sqrt{0.00333} = 0.0577$$

The z-score value is given as

$$Z_\delta^* = \frac{\sqrt{K}\hat{\mu}_\delta}{\hat{\sigma}_\delta} = \frac{\sqrt{5} \times 0.033}{0.0577} = 1.28$$

From Example 22.10, for $\alpha = 0.95$ and $K - 1 = 4$ degrees of freedom, we have $t_{\alpha/2,K-1} = 2.776$. Because

$$Z_\delta^* = 1.28 \in (-2.776, 2.776) = \left(-t_{\alpha/2,K-1}, t_{\alpha/2,K-1}\right)$$

we cannot reject the null hypothesis. Instead, we accept the null hypothesis that $\mu_\delta = 0$, that is, there is no significant difference between the naive and full Bayes classifier for this dataset.

22.3 BIAS-VARIANCE DECOMPOSITION

Given a training set $\mathbf{D} = \{\mathbf{x}_i, y_i\}_{i=1}^n$, comprising n points $\mathbf{x}_i \in \mathbb{R}^d$, with their corresponding classes y_i, a learned classification model M predicts the class for a given test point $\mathbf{x}$. The various performance measures we described above mainly focus on minimizing the prediction error by tabulating the fraction of misclassified points. However, in many applications, there may be costs associated with making wrong predictions. A *loss function* specifies the cost or penalty of predicting the class to be $\hat{y} = M(\mathbf{x})$, when the true class is y. A commonly used loss function for classification is the *zero-one loss*, defined as

$$L(y, M(\mathbf{x})) = I(M(\mathbf{x}) \neq y) = \begin{cases} 0 & \text{if } M(\mathbf{x}) = y \\ 1 & \text{if } M(\mathbf{x}) \neq y \end{cases}$$

Thus, zero-one loss assigns a cost of zero if the prediction is correct, and one otherwise. Another commonly used loss function is the *squared loss*, defined as

$$L(y, M(\mathbf{x})) = (y - M(\mathbf{x}))^2$$

where we assume that the classes are discrete valued, and not categorical.

Expected Loss

An ideal or optimal classifier is the one that minimizes the loss function. Because the true class is not known for a test case $\mathbf{x}$, the goal of learning a classification model can be cast as minimizing the expected loss:

$$E_y[L(y, M(\mathbf{x})) \,|\mathbf{x}] = \sum_y L(y, M(\mathbf{x})) \cdot P(y|\mathbf{x}) \tag{22.13}$$

where $P(y|\mathbf{x})$ is the conditional probability of class y given test point $\mathbf{x}$, and E_y denotes that the expectation is taken over the different class values y.

Minimizing the expected zero–one loss corresponds to minimizing the error rate. This can be seen by expanding Eq. (22.13) with zero–one loss. Let $M(\mathbf{x}) = c_i$, then we have

$$\begin{aligned} E_y[L(y, M(\mathbf{x})) \,|\mathbf{x}] &= E_y[I(y \neq M(\mathbf{x})) \,|\mathbf{x}] \\ &= \sum_y I(y \neq c_i) \cdot P(y|\mathbf{x}) \\ &= \sum_{y \neq c_i} P(y|\mathbf{x}) \\ &= 1 - P(c_i|\mathbf{x}) \end{aligned}$$

Thus, to minimize the expected loss we should choose c_i as the class that maximizes the posterior probability, that is, $c_i = \arg\max_y P(y|\mathbf{x})$. Because by definition [Eq. (22.1)], the error rate is simply an estimate of the expected zero–one loss, this choice also minimizes the error rate.

Bias and Variance

The expected loss for the squared loss function offers important insight into the classification problem because it can be decomposed into bias and variance terms. Intuitively, the *bias* of a classifier refers to the systematic deviation of its predicted decision boundary from the true decision boundary, whereas the *variance* of a classifier refers to the deviation among the learned decision boundaries over different training sets. More formally, because M depends on the training set, given a test point $\mathbf{x}$, we denote its predicted value as $M(\mathbf{x}, \mathbf{D})$. Consider the expected square loss:

$$
\begin{aligned}
& E_y\Big[L\big(y, M(\mathbf{x},\mathbf{D})\big) \,\big|\, \mathbf{x}, \mathbf{D}\Big] \\
&\quad = E_y\Big[\big(y - M(\mathbf{x},\mathbf{D})\big)^2 \big| \mathbf{x}, \mathbf{D}\Big] \\
&\quad = E_y\Big[\big(y \underbrace{- E_y[y|\mathbf{x}] + E_y[y|\mathbf{x}]}_{\text{add and subtract same term}} - M(\mathbf{x},\mathbf{D})\big)^2 \,\big|\, \mathbf{x}, \mathbf{D}\Big] \\
&\quad = E_y\Big[\big(y - E_y[y|\mathbf{x}]\big)^2 \,\big|\, \mathbf{x}, \mathbf{D}\Big] + E_y\Big[\big(M(\mathbf{x},\mathbf{D}) - E_y[y|\mathbf{x}]\big)^2 \,\big|\, \mathbf{x}, \mathbf{D}\Big] \\
&\qquad + E_y\Big[2\big(y - E_y[y|\mathbf{x}]\big) \cdot \big(E_y[y|\mathbf{x}] - M(\mathbf{x},\mathbf{D})\big) \,\big|\, \mathbf{x}, \mathbf{D}\Big] \\
&\quad = E_y\Big[\big(y - E_y[y|\mathbf{x}]\big)^2 \,\big|\, \mathbf{x}, \mathbf{D}\Big] + \big(M(\mathbf{x},\mathbf{D}) - E_y[y|\mathbf{x}]\big)^2 \\
&\qquad + 2\big(E_y[y|\mathbf{x}] - M(\mathbf{x},\mathbf{D})\big) \cdot \underbrace{\big(E_y[y|\mathbf{x}] - E_y[y|\mathbf{x}]\big)}_{0} \\
&\quad = \underbrace{E_y\Big[\big(y - E_y[y|\mathbf{x}]\big)^2 \,\big|\, \mathbf{x}, \mathbf{D}\Big]}_{var(y|\mathbf{x})} + \underbrace{\Big(M(\mathbf{x},\mathbf{D}) - E_y[y|\mathbf{x}]\Big)^2}_{squared\text{-}error} \qquad (22.14)
\end{aligned}
$$

Above, we made use of the fact that for any random variables X and Y, and for any constant a, we have $E[X+Y] = E[X] + E[Y]$, $E[aX] = aE[X]$, and $E[a] = a$. The first term in Eq. (22.14) is simply the variance of y given $\mathbf{x}$. The second term is the squared error between the predicted value $M(\mathbf{x}, \mathbf{D})$ and the expected value $E_y[y|\mathbf{x}]$. Because this term depends on the training set, we can eliminate this dependence by averaging over all possible training tests of size n. The average or expected squared error for a given test point $\mathbf{x}$ over all training sets is then given as

$$
\begin{aligned}
& E_{\mathbf{D}}\Big[\big(M(\mathbf{x},\mathbf{D}) - E_y[y|\mathbf{x}]\big)^2\Big] \\
&\quad = E_{\mathbf{D}}\Big[\Big(M(\mathbf{x},\mathbf{D}) \underbrace{- E_{\mathbf{D}}[M(\mathbf{x},\mathbf{D})] + E_{\mathbf{D}}[M(\mathbf{x},\mathbf{D})]}_{\text{add and subtract same term}} - E_y[y|\mathbf{x}]\Big)^2\Big] \\
&\quad = E_{\mathbf{D}}\Big[\big(M(\mathbf{x},\mathbf{D}) - E_{\mathbf{D}}[M(\mathbf{x},\mathbf{D})]\big)^2\Big] + E_{\mathbf{D}}\Big[\big(E_{\mathbf{D}}[M(\mathbf{x},\mathbf{D})] - E_y[y|\mathbf{x}]\big)^2\Big] \\
&\qquad + 2\big(E_{\mathbf{D}}[M(\mathbf{x},\mathbf{D})] - E_y[y|\mathbf{x}]\big) \cdot \underbrace{\big(E_{\mathbf{D}}[M(\mathbf{x},\mathbf{D})] - E_{\mathbf{D}}[M(\mathbf{x},\mathbf{D})]\big)}_{0} \\
&\quad = \underbrace{E_{\mathbf{D}}\Big[\big(M(\mathbf{x},\mathbf{D}) - E_{\mathbf{D}}[M(\mathbf{x},\mathbf{D})]\big)^2\Big]}_{variance} + \underbrace{\Big(E_{\mathbf{D}}[M(\mathbf{x},\mathbf{D})] - E_y[y|\mathbf{x}]\Big)^2}_{bias} \qquad (22.15)
\end{aligned}
$$

This means that the expected squared error for a given test point can be decomposed into bias and variance terms. Combining Eqs. (22.14) and (22.15) the expected squared loss over all test points $\mathbf{x}$ and over all training sets $\mathbf{D}$ of size n yields the following decomposition into noise, variance and bias terms:

$$\begin{aligned}
&E_{\mathbf{x},\mathbf{D},y}\Big[\big(y - M(\mathbf{x},\mathbf{D})\big)^2\Big] \\
&\quad = E_{\mathbf{x},\mathbf{D},y}\Big[\big(y - E_y[y|\mathbf{x}]\big)^2 \,|\mathbf{x},\mathbf{D}\Big] + E_{\mathbf{x},\mathbf{D}}\Big[\big(M(\mathbf{x},\mathbf{D}) - E_y[y|\mathbf{x}]\big)^2\Big] \\
&\quad = \underbrace{E_{\mathbf{x},y}\Big[\big(y - E_y[y|\mathbf{x}]\big)^2\Big]}_{noise} + \underbrace{E_{\mathbf{x},\mathbf{D}}\Big[\big(M(\mathbf{x},\mathbf{D}) - E_{\mathbf{D}}[M(\mathbf{x},\mathbf{D})]\big)^2\Big]}_{average\ variance} \\
&\qquad + \underbrace{E_{\mathbf{x}}\Big[\big(E_{\mathbf{D}}[M(\mathbf{x},\mathbf{D})] - E_y[y|\mathbf{x}]\big)^2\Big]}_{average\ bias}
\end{aligned} \tag{22.16}$$

Thus, the expected square loss over all test points and training sets can be decomposed into three terms: noise, average bias, and average variance. The noise term is the average variance $var(y|\mathbf{x})$ over all test points $\mathbf{x}$. It contributes a fixed cost to the loss independent of the model, and can thus be ignored when comparing different classifiers. The classifier specific loss can then be attributed to the variance and bias terms. In general, bias indicates whether the model M is correct or incorrect. It also reflects our assumptions about the domain in terms of the decision boundary. For example, if the decision boundary is nonlinear, and we use a linear classifier, then it is likely to have high bias, that is, it will be consistently incorrect over different training sets. On the other hand, a nonlinear (or a more complex) classifier is more likely to capture the correct decision boundary, and is thus likely to have a low bias. Nevertheless, this does not necessarily mean that a complex classifier will be a better one, since we also have to consider the variance term, which measures the inconsistency of the classifier decisions. A complex classifier induces a more complex decision boundary and thus may be prone to *overfitting*, that is, it may try to model all the small nuances in the training data, and thus may be susceptible to small changes in training set, which may result in high variance.

In general, the expected loss can be attributed to high bias or high variance, with typically a trade-off between these two terms. Ideally, we seek a balance between these opposing trends, that is, we prefer a classifier with an acceptable bias (reflecting domain or dataset specific assumptions) and as low a variance as possible.

Example 22.12. Figure 22.7 illustrates the trade-off between bias and variance, using the Iris principal components dataset, which has $n = 150$ points and $k = 2$ classes ($c_1 = +1$, and $c_2 = -1$). We construct $K = 10$ training datasets via bootstrap sampling, and use them to train SVM classifiers using a quadratic (homogeneous) kernel, varying the regularization constant C from 10^{-2} to 10^2.

Recall that C controls the weight placed on the slack variables, as opposed to the margin of the hyperplane (see Section 21.3). A small value of C emphasizes the margin, whereas a large value of C tries to minimize the slack terms. Figures 22.7a, 22.7b, and 22.7c show that the variance of the SVM model increases

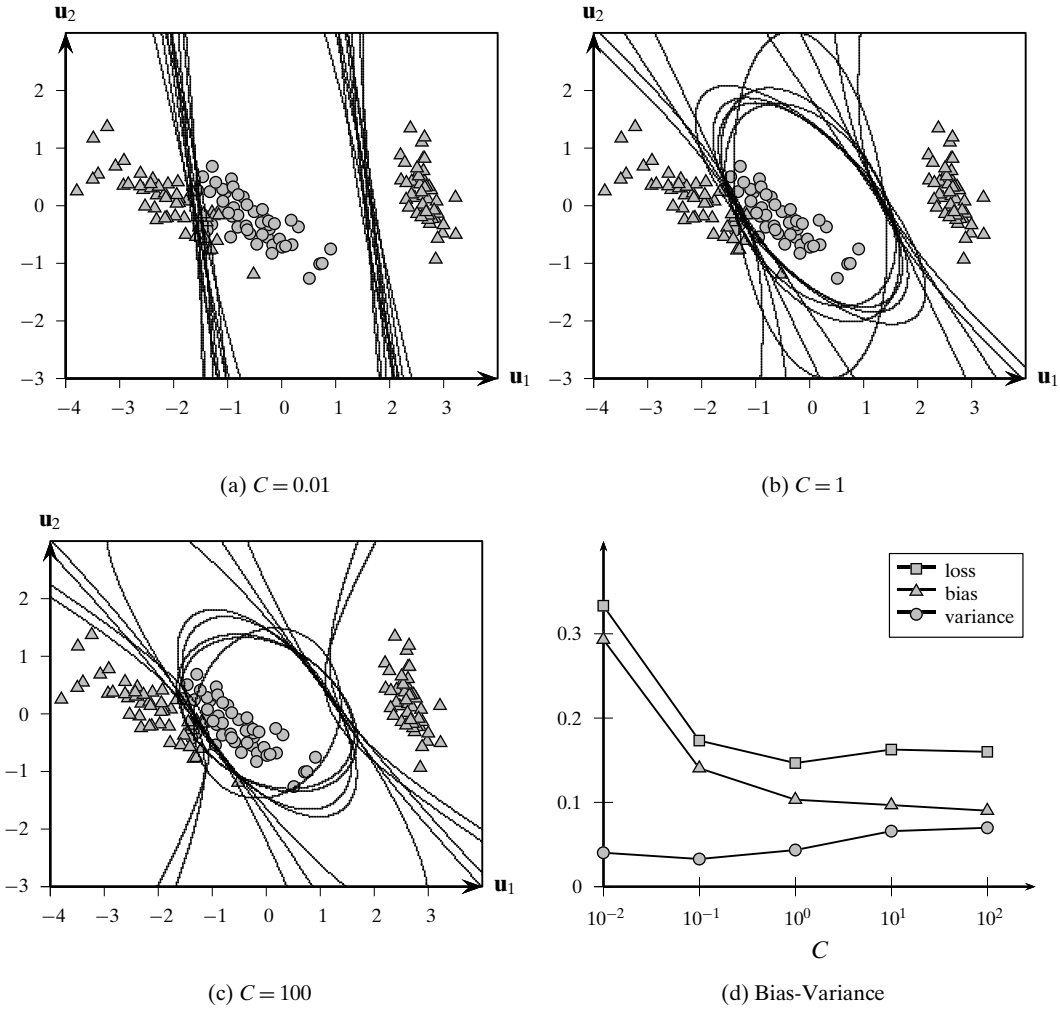


Figure 22.7. Bias-variance decomposition: SVM quadratic kernels. Decision boundaries plotted for $K = 10$ bootstrap samples.

as we increase C, as seen from the varying decision boundaries. Figure 22.7d plots the average variance and average bias for different values of C, as well as the expected loss. The bias-variance tradeoff is clearly visible, since as the bias reduces, the variance increases. The lowest expected loss is obtained when $C = 1$.

22.3.1 Ensemble Classifiers

A classifier is called *unstable* if small perturbations in the training set result in large changes in the prediction or decision boundary. High variance classifiers are inherently unstable, since they tend to overfit the data. On the other hand, high bias methods typically underfit the data, and usually have low variance. In either case, the aim of learning is to reduce classification error by reducing the variance or bias, ideally

both. Ensemble methods create a *combined classifier* using the output of multiple *base classifiers*, which are trained on different data subsets. Depending on how the training sets are selected, and on the stability of the base classifiers, ensemble classifiers can help reduce the variance and the bias, leading to a better overall performance.

Bagging

Bagging, which stands for *Bootstrap Aggregation*, is an ensemble classification method that employs multiple bootstrap samples (with replacement) from the input training data $\mathbf{D}$ to create slightly different training sets $\mathbf{D}_i$, $i = 1, 2, \ldots, K$. Different base classifiers M_i are learned, with M_i trained on $\mathbf{D}_i$. Given any test point $\mathbf{x}$, it is first classified using each of the K base classifiers, M_i. Let the number of classifiers that predict the class of $\mathbf{x}$ as c_j be given as

$$v_j(\mathbf{x}) = \Big|\big\{M_i(\mathbf{x}) = c_j \,\big|\, i = 1, \ldots, K\big\}\Big|$$

The combined classifier, denoted $\mathbf{M}^K$, predicts the class of a test point $\mathbf{x}$ by *majority voting* among the k classes:

$$\mathbf{M}^K(\mathbf{x}) = \arg\max_{c_j}\Big\{v_j(\mathbf{x}) \,\big|\, j = 1, \ldots, k\Big\}$$

For binary classification, assuming that the classes are given as $\{+1, -1\}$, the combined classifier $\mathbf{M}^K$ can be expressed more simply as

$$\mathbf{M}^K(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{K} M_i(\mathbf{x})\right)$$

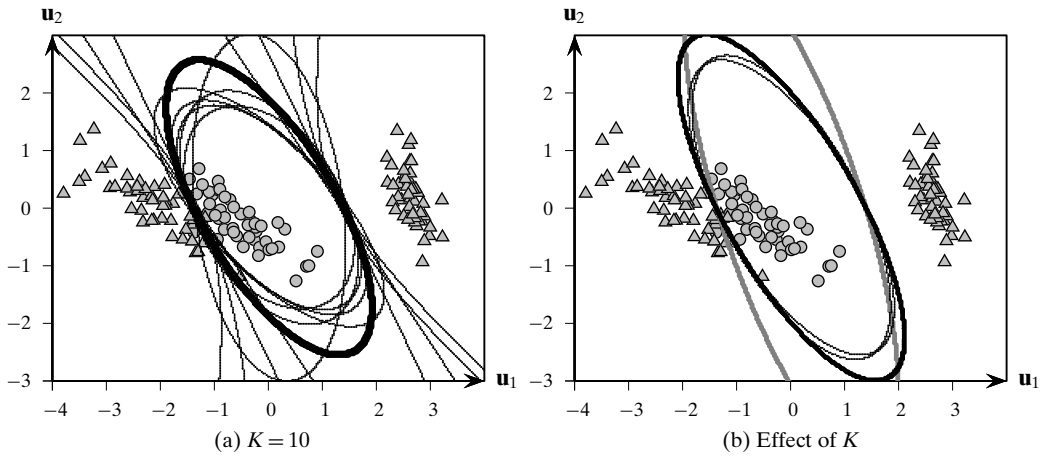
Bagging can help reduce the variance, especially if the base classifiers are unstable, due to the averaging effect of majority voting. It does not, in general, have much effect on the bias.

Example 22.13. Figure 22.8a shows the averaging effect of bagging for the Iris principal components dataset from Example 22.12. The figure shows the SVM decision boundaries for the quadratic kernel using $C = 1$. The base SVM classifiers are trained on $K = 10$ bootstrap samples. The combined (average) classifier is shown in bold.

Figure 22.8b shows the combined classifiers obtained for different values of K, keeping $C = 1$. The zero–one and squared loss for selected values of K are shown below

K	Zero–one loss	Squared loss
3	0.047	0.187
5	0.04	0.16
8	0.02	0.10
10	0.027	0.113
15	0.027	0.107

The worst training performance is obtained for $K = 3$ (in thick gray) and the best for $K = 8$ (in thick black).



(a) $K = 10$ (b) Effect of K

Figure 22.8. Bagging: combined classifiers. (a) uses $K = 10$ bootstrap samples. (b) shows average decision boundary for different values of K.

Boosting

Boosting is another ensemble technique that trains the base classifiers on different samples. However, the main idea is to carefully select the samples to *boost* the performance on hard to classify instances. Starting from an initial training sample $\mathbf{D}_1$, we train the base classifier M_1, and obtain its training error rate. To construct the next sample $\mathbf{D}_2$, we select the misclassified instances with higher probability, and after training M_2, we obtain its training error rate. To construct $\mathbf{D}_3$, those instances that are hard to classify by M_1 or M_2, have a higher probability of being selected. This process is repeated for K iterations. Thus, unlike bagging that uses independent random samples from the input dataset, boosting employs weighted or biased samples to construct the different training sets, with the current sample depending on the previous ones. Finally, the combined classifier is obtained via weighted voting over the output of the K base classifiers $M_1, M_2, \ldots, M_K$.

Boosting is most beneficial when the base classifiers are *weak*, that is, have an error rate that is slightly less than that for a random classifier. The idea is that whereas M_1 may not be particularly good on all test instances, by design M_2 may help classify some cases where M_1 fails, and M_3 may help classify instances where M_1 and M_2 fail, and so on. Thus, boosting has more of a bias reducing effect. Each of the weak learners is likely to have high bias (it is only slightly better than random guessing), but the final combined classifier can have much lower bias, since different weak learners learn to classify instances in different regions of the input space. Several variants of boosting can be obtained based on how the instance weights are computed for sampling, how the base classifiers are combined, and so on. We discuss *Adaptive Boosting (AdaBoost)*, which is one of the most popular variants.

Adaptive Boosting: AdaBoost Let $\mathbf{D}$ be the input training set, comprising n points $\mathbf{x}_i \in \mathbb{R}^d$. The boosting process will be repeated K times. Let t denote the iteration and let α_t denote the weight for the tth classifier M_t. Let w_i^t denote the weight for $\mathbf{x}_i$, with $\mathbf{w}^t = (w_1^t, w_2^t, \ldots, w_n^t)^T$ being the weight vector over all the points for the tth iteration.

ALGORITHM 22.5. Adaptive Boosting Algorithm: AdaBoost

ADABOOST(K, D):

$\mathbf{w}^0 \leftarrow \left(\frac{1}{n}\right) \cdot \mathbf{1} \in \mathbb{R}^n$
$t \leftarrow 1$
while $t \leq K$ **do**
  $\mathbf{D}_t \leftarrow$ weighted resampling with replacement from $\mathbf{D}$ using $\mathbf{w}^{t-1}$
  $M_t \leftarrow$ train classifier on $\mathbf{D}_t$
  $\epsilon_t \leftarrow \sum_{i=1}^{n} w_i^{t-1} \cdot I\big(M_t(\mathbf{x}_i) \neq y_i\big)$ `// weighted error rate on` $\mathbf{D}$
  if $\epsilon_t = 0$ **then break**
  else if $\epsilon_t < 0.5$ **then**
    $\alpha_t = \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$ `// classifier weight`
    foreach $i \in [1, n]$ **do**
      `// update point weights`
      $w_i^t = \begin{cases} w_i^{t-1} & \text{if } M_t(\mathbf{x}_i) = y_i \\ w_i^{t-1}\left(\frac{1-\epsilon_t}{\epsilon_t}\right) & \text{if } M_t(\mathbf{x}_i) \neq y_i \end{cases}$
    $\mathbf{w}^t = \frac{\mathbf{w}^t}{\mathbf{1}^T\mathbf{w}^t}$ `// normalize weights`
    $t \leftarrow t + 1$
return $\{M_1, M_2, \ldots, M_K\}$

In fact, $\mathbf{w}$ is a probability vector, whose elements sum to one. Initially all points have equal weights, that is,

$$\mathbf{w}^0 = \left(\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n}\right)^T = \frac{1}{n}\mathbf{1}$$

where $\mathbf{1} \in \mathbb{R}^n$ is the n-dimensional vector of all 1's.

The pseudo-code for AdaBoost is shown in Algorithm 22.5. During iteration t, the training sample $\mathbf{D}_t$ is obtained via weighted resampling using the distribution $\mathbf{w}^{t-1}$, that is, we draw a sample of size n with replacement, such that the ith point is chosen according to its probability w_i^{t-1}. Next, we train the classifier M_t using $\mathbf{D}_t$, and compute its weighted error rate ϵ_t on the entire input dataset $\mathbf{D}$:

$$\epsilon_t = \sum_{i=1}^{n} w_i^{t-1} \cdot I\big(M_t(\mathbf{x}_i) \neq y_i\big)$$

where I is an indicator function that is 1 when its argument is true, that is, when M_t misclassifies $\mathbf{x}_i$, and is 0 otherwise.

The weight for the tth classifier is then set as

$$\alpha_t = \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$$

and the weight for each point $\mathbf{x}_i \in \mathbf{D}$ is updated based on whether the point is misclassified or not

$$w_i^t = w_i^{t-1} \cdot \exp\Big\{\alpha_t \cdot I\big(M_t(\mathbf{x}_i) \neq y_i\big)\Big\}$$

Thus, if the predicted class matches the true class, that is, if $M_t(\mathbf{x}_i) = y_i$, then $I(M_t(\mathbf{x}_i) \neq y_i) = 0$, and the weight for point $\mathbf{x}_i$ remains unchanged. On the other hand, if the point is misclassified, that is, $M_t(\mathbf{x}_i) \neq y_i$, then we have $I(M_t(\mathbf{x}_i) \neq y_i) = 1$, and

$$w_i^t = w_i^{t-1} \cdot \exp\{\alpha_t\} = w_i^{t-1} \exp\left\{\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)\right\} = w_i^{t-1}\left(\frac{1}{\epsilon_t} - 1\right)$$

We can observe that if the error rate ϵ_t is small, then there is a greater weight increment for $\mathbf{x}_i$. The intuition is that a point that is misclassified by a good classifier (with a low error rate) should be more likely to be selected for the next training dataset. On the other hand, if the error rate of the base classifier is close to 0.5, then there is only a small change in the weight, since a bad classifier (with a high error rate) is expected to misclassify many instances. Note that for a binary class problem, an error rate of 0.5 corresponds to a random classifier, that is, one that makes a random guess. Thus, we require that a base classifier has an error rate at least slightly better than random guessing, that is, $\epsilon_t < 0.5$. If the error rate $\epsilon_t \geq 0.5$, then the boosting method discards the classifier, and returns to line 5 to try another data sample. Alternatively, one can simply invert the predictions for binary classification. It is worth emphasizing that for a multi-class problem (with $k > 2$), the requirement that $\epsilon_t < 0.5$ is a significantly stronger requirement than for the binary ($k = 2$) class problem because in the multiclass case a random classifier is expected to have an error rate of $\frac{k-1}{k}$. Note also that if the error rate of the base classifier $\epsilon_t = 0$, then we can stop the boosting iterations.

Once the point weights have been updated, we re-normalize the weights so that $\mathbf{w}^t$ is a probability vector (line 14):

$$\mathbf{w}^t = \frac{\mathbf{w}^t}{\mathbf{1}^T\mathbf{w}^t} = \frac{1}{\sum_{j=1}^n w_j^t}\left(w_1^t, w_2^t, \ldots, w_n^t\right)^T$$

Combined Classifier Given the set of boosted classifiers, $M_1, M_2, \ldots, M_K$, along with their weights $\alpha_1, \alpha_2, \ldots, \alpha_K$, the class for a test case $\mathbf{x}$ is obtained via weighted majority voting. Let $v_j(\mathbf{x})$ denote the weighted vote for class c_j over the K classifiers, given as

$$v_j(\mathbf{x}) = \sum_{t=1}^{K} \alpha_t \cdot I\big(M_t(\mathbf{x}) = c_j\big)$$

Because $I(M_t(\mathbf{x}) = c_j)$ is 1 only when $M_t(\mathbf{x}) = c_j$, the variable $v_j(\mathbf{x})$ simply obtains the tally for class c_j among the K base classifiers, taking into account the classifier weights. The combined classifier, denoted $\mathbf{M}^K$, then predicts the class for $\mathbf{x}$ as follows:

$$\mathbf{M}^K(\mathbf{x}) = \arg\max_{c_j}\Big\{v_j(\mathbf{x}) \,\big|\, j = 1, .., k\Big\}$$

In the case of binary classification, with classes $\{+1, -1\}$, the combined classifier $\mathbf{M}^K$ can be expressed more simply as

$$\mathbf{M}^K(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{K} \alpha_t M_t(\mathbf{x})\right)$$

Example 22.14. Figure 22.9a illustrates the boosting approach on the Iris principal components dataset, using linear SVMs as the base classifiers. The regularization constant was set to $C = 1$. The hyperplane learned in iteration t is denoted h_t, thus, the classifier model is given as $M_t(\mathbf{x}) = \text{sign}(h_t(\mathbf{x}))$. As such, no individual linear hyperplane can discriminate between the classes very well, as seen from their error rates on the training set:

M_t	h_1	h_2	h_3	h_4
ϵ_t	0.280	0.305	0.174	0.282
α_t	0.944	0.826	1.559	0.935

However, when we combine the decisions from successive hyperplanes weighted by α_t, we observe a marked drop in the error rate for the combined classifier $\mathbf{M}^K(\mathbf{x})$ as K increases:

combined model	$\mathbf{M}^1$	$\mathbf{M}^2$	$\mathbf{M}^3$	$\mathbf{M}^4$
training error rate	0.280	0.253	0.073	0.047

We can see, for example, that the combined classifier $\mathbf{M}^3$, comprising h_1, h_2 and h_3, has already captured the essential features of the nonlinear decision boundary between the two classes, yielding an error rate of 7.3%. Further reduction in the training error is obtained by increasing the number of boosting steps.

To assess the performance of the combined classifier on independent testing data, we employ 5-fold cross-validation, and plot the average testing and training error rates as a function of K in Figure 22.9b. We can see that as the number of base

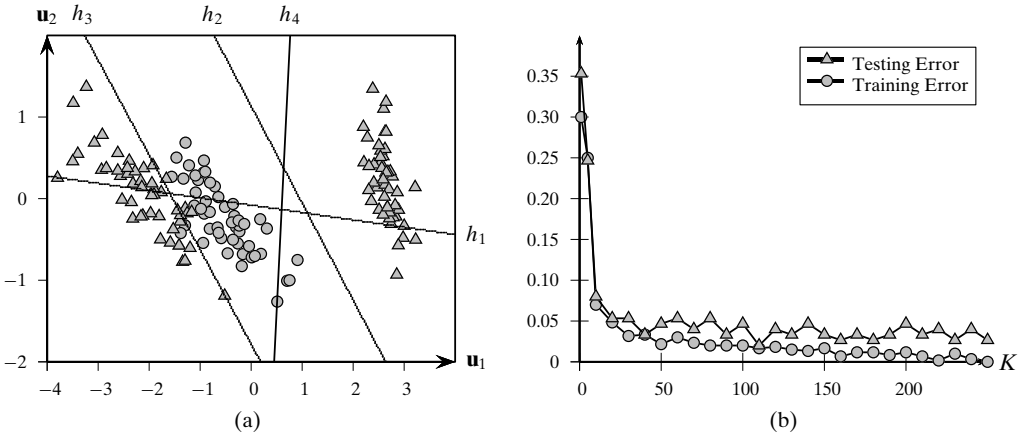


Figure 22.9. (a) Boosting SVMs with linear kernel. (b) Average testing and training error: 5-fold cross-validation.

classifiers K increases, both the training and testing error rates reduce. However, while the training error essentially goes to 0, the testing error does not reduce beyond 0.02, which happens at $K = 110$. This example illustrates the effectiveness of boosting in reducing the bias.

Bagging as a Special Case of AdaBoost: Bagging can be considered as a special case of AdaBoost, where $w^t = \frac{1}{n}\mathbf{1}$, and $\alpha_t = 1$ for all K iterations. In this case, the weighted resampling defaults to regular resampling with replacement, and the predicted class for a test case also defaults to simple majority voting.

22.4 FURTHER READING

The application of ROC analysis to classifier performance was introduced in Provost and Fawcett (1997), with an excellent introduction to ROC analysis given in Fawcett (2006). For an in-depth description of the bootstrap, cross-validation, and other methods for assessing classification accuracy see Efron and Tibshirani (1993). For many datasets simple rules, like one-level decision trees, can yield good classification performance; see Holte (1993) for details. For a recent review and comparison of classifiers over multiple datasets see Demšar (2006). A discussion of bias, variance, and zero–one loss for classification appears in Friedman (1997), with a unified decomposition of bias and variance for both squared and zero–one loss given in Domingos (2000). The concept of bagging was proposed in Breiman (1996), and that of adaptive boosting in Freund and Schapire (1997). Random forests is a tree-based ensemble approach that can be very effective; see Breiman (2001) for details. For a comprehensive overview on the evaluation of classification algorithms see Japkowicz and Shah (2011).

Breiman, L. (1996). "Bagging predictors." *Machine Learning*, 24 (2): 123–140.

Breiman, L. (2001). "Random forests." *Machine Learning*, 45 (1): 5–32.

Demšar, J. (2006). "Statistical comparisons of classifiers over multiple data sets." *The Journal of Machine Learning Research*, 7: 1–30.

Domingos, P. (2000). "A unified bias-variance decomposition for zero-one and squared loss." *In Proceedings of the National Conference on Artificial Intelligence*, 564–569.

Efron, B. and Tibshirani, R. (1993). *An Introduction to the Bootstrap,* vol. 57. Boca Raton, FL: Chapman & Hall/CRC.

Fawcett, T. (2006). "An introduction to ROC analysis." *Pattern Recognition Letters*, 27 (8): 861–874.

Freund, Y. and Schapire, R. E. (1997). "A decision-theoretic generalization of on-line learning and an application to boosting." *Journal of Computer and System Sciences*, 55 (1): 119–139.

Friedman, J. H. (1997). "On bias, variance, 0/1-loss, and the curse-of-dimensionality." *Data Mining and Knowledge Discovery*, 1 (1): 55–77.

Holte, R. C. (1993). "Very simple classification rules perform well on most commonly used datasets." *Machine Learning*, 11 (1): 63–90.

Japkowicz, N. and Shah, M. (2011). *Evaluating Learning Algorithms: A Classification Perspective*. New York: Cambridge University Press.

Provost, F. and Fawcett, T. (1997). "Analysis and visualization of classifier performance: Comparison under imprecise class and cost distributions." *In Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining,* Menlo Park, CA, 43–48.

22.5 EXERCISES

Q1. True or False:

(a) A classification model must have 100% accuracy (overall) on the training dataset.

(b) A classification model must have 100% coverage (overall) on the training dataset.

Q2. Given the training database in Table 22.6a and the testing data in Table 22.6b, answer the following questions:

(a) Build the complete decision tree using binary splits and Gini index as the evaluation measure (see Chapter 19).

(b) Compute the accuracy of the classifier on the test data. Also show the per class accuracy and coverage.

Table 22.6. Data for Q2

X	Y	Z	Class
15	1	A	1
20	3	B	2
25	2	A	1
30	4	A	1
35	2	B	2
25	4	A	1
15	2	B	2
20	3	B	2

(a) Training

X	Y	Z	Class
10	2	A	2
20	1	B	1
30	3	A	2
40	2	B	2
15	1	B	1

(b) Testing

Q3. Show that for binary classification the majority voting for the combined classifier decision in boosting can be expressed as

$$\mathbf{M}^K(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{K} \alpha_t M_t(\mathbf{x})\right)$$

Q4. Consider the 2-dimensional dataset shown in Figure 22.10, with the labeled points belonging to two classes: c_1 (triangles) and c_2 (circles). Assume that the six hyperplanes were learned from different bootstrap samples. Find the error rate for each of the six hyperplanes on the entire dataset. Then, compute the 95% confidence

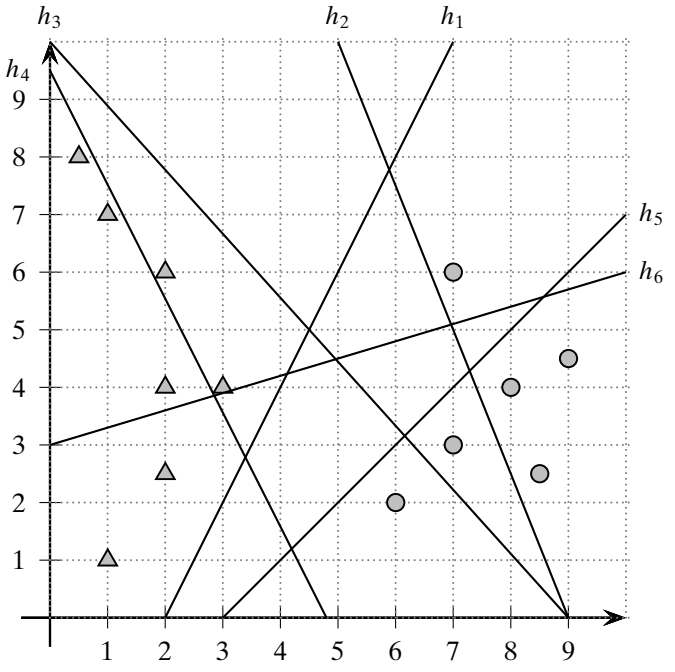


Figure 22.10. For Q4.

Table 22.7. Critical values for t-test

dof	1	2	3	4	5	6
$t_{\alpha/2}$	12.7065	4.3026	3.1824	2.7764	2.5706	2.4469

interval for the expected error rate, using the t-distribution critical values for different degrees of freedom (dof) given in Table 22.7.

Q5. Consider the probabilities $P(+1|x_i)$ for the positive class obtained for some classifier, and given the true class labels y_i

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}
y_i	$+1$	-1	$+1$	$+1$	-1	$+1$	-1	$+1$	-1	-1
$P(+1\|x_i)$	0.53	0.86	0.25	0.95	0.87	0.86	0.76	0.94	0.44	0.86

Plot the ROC curve for this classifier.

Index