

Introduction to Data Mining

Pang-Ning Tan
Michael Steinbach
Vipin Kumar

Pearson

Pearson Education Limited
Edinburgh Gate
Harlow
Essex CM20 2JE
England and Associated Companies throughout the world

Visit us on the World Wide Web at: www.pearsoned.co.uk

© Pearson Education Limited 2014

ISBN 10: 1-292-02615-4
ISBN 13: 978-1-292-02615-2

Printed in the United States of America

Contents

Chapter 1. Introduction

Pang-Ning Tan/Michael Steinbach/Vipin Kumar

Chapter 2. Data

Pang-Ning Tan/Michael Steinbach/Vipin Kumar

Chapter 3. Exploring Data

Pang-Ning Tan/Michael Steinbach/Vipin Kumar

Chapter 4. Classification: Basic Concepts, Decision Trees, and Model Evaluation

Pang-Ning Tan/Michael Steinbach/Vipin Kumar

Chapter 5. Classification: Alternative Techniques

Pang-Ning Tan/Michael Steinbach/Vipin Kumar

Chapter 6. Association Analysis: Basic Concepts and Algorithms

Pang-Ning Tan/Michael Steinbach/Vipin Kumar

Chapter 7. Association Analysis: Advanced Concepts

Pang-Ning Tan/Michael Steinbach/Vipin Kumar

Chapter 8. Cluster Analysis: Basic Concepts and Algorithms

Pang-Ning Tan/Michael Steinbach/Vipin Kumar

Chapter 9. Cluster Analysis: Additional Issues and Algorithms

Pang-Ning Tan/Michael Steinbach/Vipin Kumar

Chapter 10. Anomaly Detection

Pang-Ning Tan/Michael Steinbach/Vipin Kumar

Appendix B: Dimensionality Reduction

Pang-Ning Tan/Michael Steinbach/Vipin Kumar

Appendix D: Regression

Pang-Ning Tan/Michael Steinbach/Vipin Kumar

Appendix E: Optimization

Pang-Ning Tan/Michael Steinbach/Vipin Kumar

1

Introduction

Rapid advances in data collection and storage technology have enabled organizations to accumulate vast amounts of data. However, extracting useful information has proven extremely challenging. Often, traditional data analysis tools and techniques cannot be used because of the massive size of a data set. Sometimes, the non-traditional nature of the data means that traditional approaches cannot be applied even if the data set is relatively small. In other situations, the questions that need to be answered cannot be addressed using existing data analysis techniques, and thus, new methods need to be developed.

Data mining is a technology that blends traditional data analysis methods with sophisticated algorithms for processing large volumes of data. It has also opened up exciting opportunities for exploring and analyzing new types of data and for analyzing old types of data in new ways. In this introductory chapter, we present an overview of data mining and outline the key topics to be covered in this book. We start with a description of some well-known applications that require new techniques for data analysis.

Business Point-of-sale data collection (bar code scanners, radio frequency identification (RFID), and smart card technology) have allowed retailers to collect up-to-the-minute data about customer purchases at the checkout counters of their stores. Retailers can utilize this information, along with other business-critical data such as Web logs from e-commerce Web sites and customer service records from call centers, to help them better understand the needs of their customers and make more informed business decisions.

Data mining techniques can be used to support a wide range of business intelligence applications such as customer profiling, targeted marketing, workflow management, store layout, and fraud detection. It can also help retailers

answer important business questions such as “Who are the most profitable customers?” “What products can be cross-sold or up-sold?” and “What is the revenue outlook of the company for next year?” Some of these questions motivated the creation of association analysis (Chapters 6 and 7), a new data analysis technique.

Medicine, Science, and Engineering Researchers in medicine, science, and engineering are rapidly accumulating data that is key to important new discoveries. For example, as an important step toward improving our understanding of the Earth’s climate system, NASA has deployed a series of Earth-orbiting satellites that continuously generate global observations of the land surface, oceans, and atmosphere. However, because of the size and spatio-temporal nature of the data, traditional methods are often not suitable for analyzing these data sets. Techniques developed in data mining can aid Earth scientists in answering questions such as “What is the relationship between the frequency and intensity of ecosystem disturbances such as droughts and hurricanes to global warming?” “How is land surface precipitation and temperature affected by ocean surface temperature?” and “How well can we predict the beginning and end of the growing season for a region?”

As another example, researchers in molecular biology hope to use the large amounts of genomic data currently being gathered to better understand the structure and function of genes. In the past, traditional methods in molecular biology allowed scientists to study only a few genes at a time in a given experiment. Recent breakthroughs in microarray technology have enabled scientists to compare the behavior of thousands of genes under various situations. Such comparisons can help determine the function of each gene and perhaps isolate the genes responsible for certain diseases. However, the noisy and high-dimensional nature of data requires new types of data analysis. In addition to analyzing gene array data, data mining can also be used to address other important biological challenges such as protein structure prediction, multiple sequence alignment, the modeling of biochemical pathways, and phylogenetics.

1.1 What Is Data Mining?

Data mining is the process of automatically discovering useful information in large data repositories. Data mining techniques are deployed to scour large databases in order to find novel and useful patterns that might otherwise remain unknown. They also provide capabilities to predict the outcome of a

future observation, such as predicting whether a newly arrived customer will spend more than \$100 at a department store.

Not all information discovery tasks are considered to be data mining. For example, looking up individual records using a database management system or finding particular Web pages via a query to an Internet search engine are tasks related to the area of **information retrieval**. Although such tasks are important and may involve the use of the sophisticated algorithms and data structures, they rely on traditional computer science techniques and obvious features of the data to create index structures for efficiently organizing and retrieving information. Nonetheless, data mining techniques have been used to enhance information retrieval systems.

Data Mining and Knowledge Discovery

Data mining is an integral part of **knowledge discovery in databases (KDD)**, which is the overall process of converting raw data into useful information, as shown in Figure 1.1. This process consists of a series of transformation steps, from data preprocessing to postprocessing of data mining results.

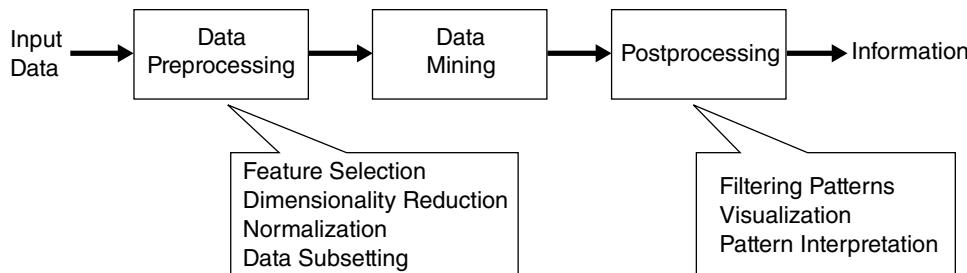


Figure 1.1. The process of knowledge discovery in databases (KDD).

The input data can be stored in a variety of formats (flat files, spreadsheets, or relational tables) and may reside in a centralized data repository or be distributed across multiple sites. The purpose of **preprocessing** is to transform the raw input data into an appropriate format for subsequent analysis. The steps involved in data preprocessing include fusing data from multiple sources, cleaning data to remove noise and duplicate observations, and selecting records and features that are relevant to the data mining task at hand. Because of the many ways data can be collected and stored, data

preprocessing is perhaps the most laborious and time-consuming step in the overall knowledge discovery process.

“Closing the loop” is the phrase often used to refer to the process of integrating data mining results into decision support systems. For example, in business applications, the insights offered by data mining results can be integrated with campaign management tools so that effective marketing promotions can be conducted and tested. Such integration requires a **postprocessing** step that ensures that only valid and useful results are incorporated into the decision support system. An example of postprocessing is visualization (see Chapter 3), which allows analysts to explore the data and the data mining results from a variety of viewpoints. Statistical measures or hypothesis testing methods can also be applied during postprocessing to eliminate spurious data mining results.

1.2 Motivating Challenges

As mentioned earlier, traditional data analysis techniques have often encountered practical difficulties in meeting the challenges posed by new data sets. The following are some of the specific challenges that motivated the development of data mining.

Scalability Because of advances in data generation and collection, data sets with sizes of gigabytes, terabytes, or even petabytes are becoming common. If data mining algorithms are to handle these massive data sets, then they must be scalable. Many data mining algorithms employ special search strategies to handle exponential search problems. Scalability may also require the implementation of novel data structures to access individual records in an efficient manner. For instance, out-of-core algorithms may be necessary when processing data sets that cannot fit into main memory. Scalability can also be improved by using sampling or developing parallel and distributed algorithms.

High Dimensionality It is now common to encounter data sets with hundreds or thousands of attributes instead of the handful common a few decades ago. In bioinformatics, progress in microarray technology has produced gene expression data involving thousands of features. Data sets with temporal or spatial components also tend to have high dimensionality. For example, consider a data set that contains measurements of temperature at various locations. If the temperature measurements are taken repeatedly for an extended period, the number of dimensions (features) increases in proportion to

the number of measurements taken. Traditional data analysis techniques that were developed for low-dimensional data often do not work well for such high-dimensional data. Also, for some data analysis algorithms, the computational complexity increases rapidly as the dimensionality (the number of features) increases.

Heterogeneous and Complex Data Traditional data analysis methods often deal with data sets containing attributes of the same type, either continuous or categorical. As the role of data mining in business, science, medicine, and other fields has grown, so has the need for techniques that can handle heterogeneous attributes. Recent years have also seen the emergence of more complex data objects. Examples of such non-traditional types of data include collections of Web pages containing semi-structured text and hyperlinks; DNA data with sequential and three-dimensional structure; and climate data that consists of time series measurements (temperature, pressure, etc.) at various locations on the Earth's surface. Techniques developed for mining such complex objects should take into consideration relationships in the data, such as temporal and spatial autocorrelation, graph connectivity, and parent-child relationships between the elements in semi-structured text and XML documents.

Data Ownership and Distribution Sometimes, the data needed for an analysis is not stored in one location or owned by one organization. Instead, the data is geographically distributed among resources belonging to multiple entities. This requires the development of distributed data mining techniques. Among the key challenges faced by distributed data mining algorithms include (1) how to reduce the amount of communication needed to perform the distributed computation, (2) how to effectively consolidate the data mining results obtained from multiple sources, and (3) how to address data security issues.

Non-traditional Analysis The traditional statistical approach is based on a hypothesize-and-test paradigm. In other words, a hypothesis is proposed, an experiment is designed to gather the data, and then the data is analyzed with respect to the hypothesis. Unfortunately, this process is extremely labor-intensive. Current data analysis tasks often require the generation and evaluation of thousands of hypotheses, and consequently, the development of some data mining techniques has been motivated by the desire to automate the process of hypothesis generation and evaluation. Furthermore, the data sets analyzed in data mining are typically not the result of a carefully designed

experiment and often represent opportunistic samples of the data, rather than random samples. Also, the data sets frequently involve non-traditional types of data and data distributions.

1.3 The Origins of Data Mining

Brought together by the goal of meeting the challenges of the previous section, researchers from different disciplines began to focus on developing more efficient and scalable tools that could handle diverse types of data. This work, which culminated in the field of data mining, built upon the methodology and algorithms that researchers had previously used. In particular, data mining draws upon ideas, such as (1) sampling, estimation, and hypothesis testing from statistics and (2) search algorithms, modeling techniques, and learning theories from artificial intelligence, pattern recognition, and machine learning. Data mining has also been quick to adopt ideas from other areas, including optimization, evolutionary computing, information theory, signal processing, visualization, and information retrieval.

A number of other areas also play key supporting roles. In particular, database systems are needed to provide support for efficient storage, indexing, and query processing. Techniques from high performance (parallel) computing are often important in addressing the massive size of some data sets. Distributed techniques can also help address the issue of size and are essential when the data cannot be gathered in one location.

Figure 1.2 shows the relationship of data mining to other areas.

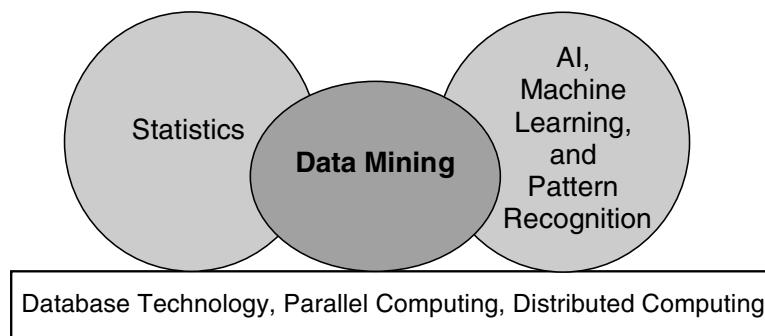


Figure 1.2. Data mining as a confluence of many disciplines.

1.4 Data Mining Tasks

Data mining tasks are generally divided into two major categories:

Predictive tasks. The objective of these tasks is to predict the value of a particular attribute based on the values of other attributes. The attribute to be predicted is commonly known as the **target** or **dependent variable**, while the attributes used for making the prediction are known as the **explanatory** or **independent variables**.

Descriptive tasks. Here, the objective is to derive patterns (correlations, trends, clusters, trajectories, and anomalies) that summarize the underlying relationships in data. Descriptive data mining tasks are often exploratory in nature and frequently require postprocessing techniques to validate and explain the results.

Figure 1.3 illustrates four of the core data mining tasks that are described in the remainder of this book.

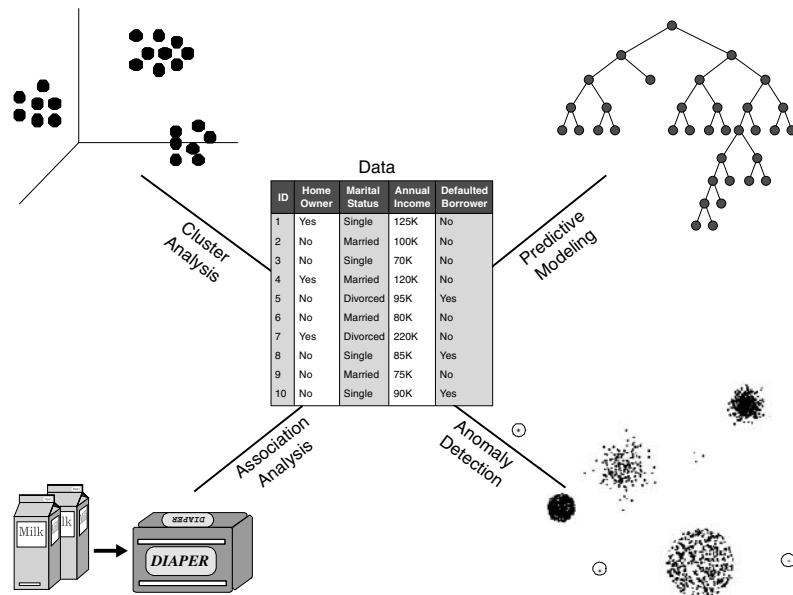


Figure 1.3. Four of the core data mining tasks.

Predictive modeling refers to the task of building a model for the target variable as a function of the explanatory variables. There are two types of predictive modeling tasks: **classification**, which is used for discrete target variables, and **regression**, which is used for continuous target variables. For example, predicting whether a Web user will make a purchase at an online bookstore is a classification task because the target variable is binary-valued. On the other hand, forecasting the future price of a stock is a regression task because price is a continuous-valued attribute. The goal of both tasks is to learn a model that minimizes the error between the predicted and true values of the target variable. Predictive modeling can be used to identify customers that will respond to a marketing campaign, predict disturbances in the Earth's ecosystem, or judge whether a patient has a particular disease based on the results of medical tests.

Example 1.1 (Predicting the Type of a Flower). Consider the task of predicting a species of flower based on the characteristics of the flower. In particular, consider classifying an Iris flower as to whether it belongs to one of the following three Iris species: Setosa, Versicolour, or Virginica. To perform this task, we need a data set containing the characteristics of various flowers of these three species. A data set with this type of information is the well-known Iris data set from the UCI Machine Learning Repository at <http://www.ics.uci.edu/~mlearn>. In addition to the species of a flower, this data set contains four other attributes: sepal width, sepal length, petal length, and petal width. (The Iris data set and its attributes are described further in Section 3.1.) Figure 1.4 shows a plot of petal width versus petal length for the 150 flowers in the Iris data set. Petal width is broken into the categories *low*, *medium*, and *high*, which correspond to the intervals $[0, 0.75]$, $[0.75, 1.75]$, $[1.75, \infty)$, respectively. Also, petal length is broken into categories *low*, *medium*, and *high*, which correspond to the intervals $[0, 2.5]$, $[2.5, 5]$, $[5, \infty)$, respectively. Based on these categories of petal width and length, the following rules can be derived:

Petal width low and petal length low implies Setosa.

Petal width medium and petal length medium implies Versicolour.

Petal width high and petal length high implies Virginica.

While these rules do not classify all the flowers, they do a good (but not perfect) job of classifying most of the flowers. Note that flowers from the Setosa species are well separated from the Versicolour and Virginica species with respect to petal width and length, but the latter two species overlap somewhat with respect to these attributes. ■

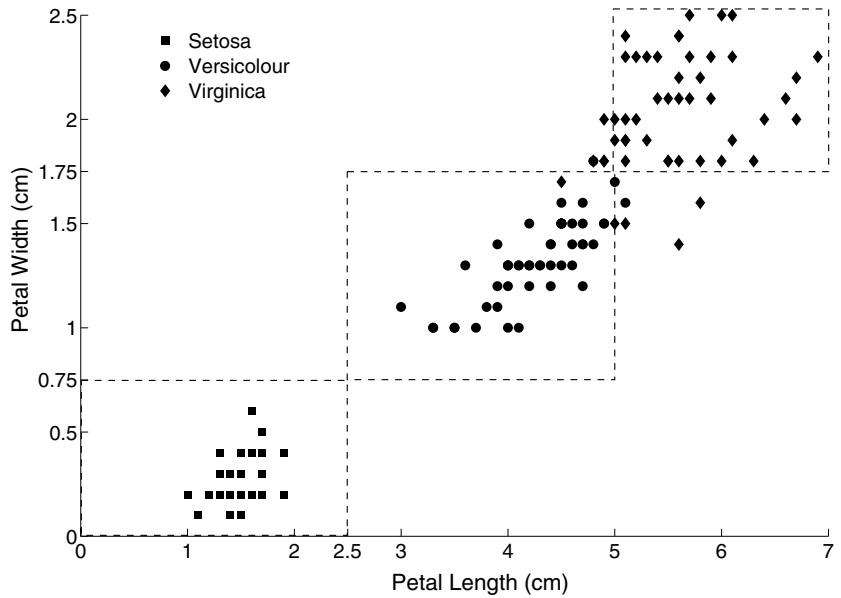


Figure 1.4. Petal width versus petal length for 150 Iris flowers.

Association analysis is used to discover patterns that describe strongly associated features in the data. The discovered patterns are typically represented in the form of implication rules or feature subsets. Because of the exponential size of its search space, the goal of association analysis is to extract the most interesting patterns in an efficient manner. Useful applications of association analysis include finding groups of genes that have related functionality, identifying Web pages that are accessed together, or understanding the relationships between different elements of Earth's climate system.

Example 1.2 (Market Basket Analysis). The transactions shown in Table 1.1 illustrate point-of-sale data collected at the checkout counters of a grocery store. Association analysis can be applied to find items that are frequently bought together by customers. For example, we may discover the rule $\{\text{Diapers}\} \rightarrow \{\text{Milk}\}$, which suggests that customers who buy diapers also tend to buy milk. This type of rule can be used to identify potential cross-selling opportunities among related items. ■

Cluster analysis seeks to find groups of closely related observations so that observations that belong to the same cluster are more similar to each other

Table 1.1. Market basket data.

Transaction ID	Items
1	{Bread, Butter, Diapers, Milk}
2	{Coffee, Sugar, Cookies, Salmon}
3	{Bread, Butter, Coffee, Diapers, Milk, Eggs}
4	{Bread, Butter, Salmon, Chicken}
5	{Eggs, Bread, Butter}
6	{Salmon, Diapers, Milk}
7	{Bread, Tea, Sugar, Eggs}
8	{Coffee, Sugar, Chicken, Eggs}
9	{Bread, Diapers, Milk, Salt}
10	{Tea, Eggs, Cookies, Diapers, Milk}

than observations that belong to other clusters. Clustering has been used to group sets of related customers, find areas of the ocean that have a significant impact on the Earth's climate, and compress data.

Example 1.3 (Document Clustering). The collection of news articles shown in Table 1.2 can be grouped based on their respective topics. Each article is represented as a set of word-frequency pairs (w, c) , where w is a word and c is the number of times the word appears in the article. There are two natural clusters in the data set. The first cluster consists of the first four articles, which correspond to news about the economy, while the second cluster contains the last four articles, which correspond to news about health care. A good clustering algorithm should be able to identify these two clusters based on the similarity between words that appear in the articles.

Table 1.2. Collection of news articles.

Article	Words
1	dollar: 1, industry: 4, country: 2, loan: 3, deal: 2, government: 2
2	machinery: 2, labor: 3, market: 4, industry: 2, work: 3, country: 1
3	job: 5, inflation: 3, rise: 2, jobless: 2, market: 3, country: 2, index: 3
4	domestic: 3, forecast: 2, gain: 1, market: 2, sale: 3, price: 2
5	patient: 4, symptom: 2, drug: 3, health: 2, clinic: 2, doctor: 2
6	pharmaceutical: 2, company: 3, drug: 2, vaccine: 1, flu: 3
7	death: 2, cancer: 4, drug: 3, public: 4, health: 3, director: 2
8	medical: 2, cost: 3, increase: 2, patient: 2, health: 3, care: 1

■

Anomaly detection is the task of identifying observations whose characteristics are significantly different from the rest of the data. Such observations are known as **anomalies** or **outliers**. The goal of an anomaly detection algorithm is to discover the real anomalies and avoid falsely labeling normal objects as anomalous. In other words, a good anomaly detector must have a high detection rate and a low false alarm rate. Applications of anomaly detection include the detection of fraud, network intrusions, unusual patterns of disease, and ecosystem disturbances.

Example 1.4 (Credit Card Fraud Detection). A credit card company records the transactions made by every credit card holder, along with personal information such as credit limit, age, annual income, and address. Since the number of fraudulent cases is relatively small compared to the number of legitimate transactions, anomaly detection techniques can be applied to build a profile of legitimate transactions for the users. When a new transaction arrives, it is compared against the profile of the user. If the characteristics of the transaction are very different from the previously created profile, then the transaction is flagged as potentially fraudulent. ■

1.5 Scope and Organization of the Book

This book introduces the major principles and techniques used in data mining from an algorithmic perspective. A study of these principles and techniques is essential for developing a better understanding of how data mining technology can be applied to various kinds of data. This book also serves as a starting point for readers who are interested in doing research in this field.

We begin the technical discussion of this book with a chapter on data (Chapter 2), which discusses the basic types of data, data quality, preprocessing techniques, and measures of similarity and dissimilarity. Although this material can be covered quickly, it provides an essential foundation for data analysis. Chapter 3, on data exploration, discusses summary statistics, visualization techniques, and On-Line Analytical Processing (OLAP). These techniques provide the means for quickly gaining insight into a data set.

Chapters 4 and 5 cover classification. Chapter 4 provides a foundation by discussing decision tree classifiers and several issues that are important to all classification: overfitting, performance evaluation, and the comparison of different classification models. Using this foundation, Chapter 5 describes a number of other important classification techniques: rule-based systems, nearest-neighbor classifiers, Bayesian classifiers, artificial neural networks, support vector machines, and ensemble classifiers, which are collections of classi-

fiers. The multiclass and imbalanced class problems are also discussed. These topics can be covered independently.

Association analysis is explored in Chapters 6 and 7. Chapter 6 describes the basics of association analysis: frequent itemsets, association rules, and some of the algorithms used to generate them. Specific types of frequent itemsets—maximal, closed, and hyperclique—that are important for data mining are also discussed, and the chapter concludes with a discussion of evaluation measures for association analysis. Chapter 7 considers a variety of more advanced topics, including how association analysis can be applied to categorical and continuous data or to data that has a concept hierarchy. (A concept hierarchy is a hierarchical categorization of objects, e.g., store items, clothing, shoes, sneakers.) This chapter also describes how association analysis can be extended to find sequential patterns (patterns involving order), patterns in graphs, and negative relationships (if one item is present, then the other is not).

Cluster analysis is discussed in Chapters 8 and 9. Chapter 8 first describes the different types of clusters and then presents three specific clustering techniques: K-means, agglomerative hierarchical clustering, and DBSCAN. This is followed by a discussion of techniques for validating the results of a clustering algorithm. Additional clustering concepts and techniques are explored in Chapter 9, including fuzzy and probabilistic clustering, Self-Organizing Maps (SOM), graph-based clustering, and density-based clustering. There is also a discussion of scalability issues and factors to consider when selecting a clustering algorithm.

The last chapter, Chapter 10, is on anomaly detection. After some basic definitions, several different types of anomaly detection are considered: statistical, distance-based, density-based, and clustering-based. Appendices A through E give a brief review of important topics that are used in portions of the book: linear algebra, dimensionality reduction, statistics, regression, and optimization.

The subject of data mining, while relatively young compared to statistics or machine learning, is already too large to cover in a single book. Selected references to topics that are only briefly covered, such as data quality, are provided in the bibliographic notes of the appropriate chapter. References to topics not covered in this book, such as data mining for streams and privacy-preserving data mining, are provided in the bibliographic notes of this chapter.

1.6 Bibliographic Notes

The topic of data mining has inspired many textbooks. Introductory textbooks include those by Dunham [10], Han and Kamber [21], Hand et al. [23], and Roiger and Geatz [36]. Data mining books with a stronger emphasis on business applications include the works by Berry and Linoff [2], Pyle [34], and Parr Rud [33]. Books with an emphasis on statistical learning include those by Cherkassky and Mulier [6], and Hastie et al. [24]. Some books with an emphasis on machine learning or pattern recognition are those by Duda et al. [9], Kantardzic [25], Mitchell [31], Webb [41], and Witten and Frank [42]. There are also some more specialized books: Chakrabarti [4] (web mining), Fayyad et al. [13] (collection of early articles on data mining), Fayyad et al. [11] (visualization), Grossman et al. [18] (science and engineering), Kargupta and Chan [26] (distributed data mining), Wang et al. [40] (bioinformatics), and Zaki and Ho [44] (parallel data mining).

There are several conferences related to data mining. Some of the main conferences dedicated to this field include the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), the IEEE International Conference on Data Mining (ICDM), the SIAM International Conference on Data Mining (SDM), the European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD), and the Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD). Data mining papers can also be found in other major conferences such as the ACM SIGMOD/PODS conference, the International Conference on Very Large Data Bases (VLDB), the Conference on Information and Knowledge Management (CIKM), the International Conference on Data Engineering (ICDE), the International Conference on Machine Learning (ICML), and the National Conference on Artificial Intelligence (AAAI).

Journal publications on data mining include *IEEE Transactions on Knowledge and Data Engineering*, *Data Mining and Knowledge Discovery*, *Knowledge and Information Systems*, *Intelligent Data Analysis*, *Information Systems*, and the *Journal of Intelligent Information Systems*.

There have been a number of general articles on data mining that define the field or its relationship to other fields, particularly statistics. Fayyad et al. [12] describe data mining and how it fits into the total knowledge discovery process. Chen et al. [5] give a database perspective on data mining. Ramakrishnan and Grama [35] provide a general discussion of data mining and present several viewpoints. Hand [22] describes how data mining differs from statistics, as does Friedman [14]. Lambert [29] explores the use of statistics for large data sets and provides some comments on the respective roles of data mining and statistics.

Glymour et al. [16] consider the lessons that statistics may have for data mining. Smyth et al. [38] describe how the evolution of data mining is being driven by new types of data and applications, such as those involving streams, graphs, and text. Emerging applications in data mining are considered by Han et al. [20] and Smyth [37] describes some research challenges in data mining. A discussion of how developments in data mining research can be turned into practical tools is given by Wu et al. [43]. Data mining standards are the subject of a paper by Grossman et al. [17]. Bradley [3] discusses how data mining algorithms can be scaled to large data sets.

With the emergence of new data mining applications have come new challenges that need to be addressed. For instance, concerns about privacy breaches as a result of data mining have escalated in recent years, particularly in application domains such as Web commerce and health care. As a result, there is growing interest in developing data mining algorithms that maintain user privacy. Developing techniques for mining encrypted or randomized data is known as **privacy-preserving data mining**. Some general references in this area include papers by Agrawal and Srikant [1], Clifton et al. [7] and Kargupta et al. [27]. Vassilios et al. [39] provide a survey.

Recent years have witnessed a growing number of applications that rapidly generate continuous streams of data. Examples of stream data include network traffic, multimedia streams, and stock prices. Several issues must be considered when mining data streams, such as the limited amount of memory available, the need for online analysis, and the change of the data over time. Data mining for stream data has become an important area in data mining. Some selected publications are Domingos and Hulten [8] (classification), Giannella et al. [15] (association analysis), Guha et al. [19] (clustering), Kifer et al. [28] (change detection), Papadimitriou et al. [32] (time series), and Law et al. [30] (dimensionality reduction).

Bibliography

- [1] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *Proc. of 2000 ACM-SIGMOD Intl. Conf. on Management of Data*, pages 439–450, Dallas, Texas, 2000. ACM Press.
- [2] M. J. A. Berry and G. Linoff. *Data Mining Techniques: For Marketing, Sales, and Customer Relationship Management*. Wiley Computer Publishing, 2nd edition, 2004.
- [3] P. S. Bradley, J. Gehrke, R. Ramakrishnan, and R. Srikant. Scaling mining algorithms to large databases. *Communications of the ACM*, 45(8):38–43, 2002.
- [4] S. Chakrabarti. *Mining the Web: Discovering Knowledge from Hypertext Data*. Morgan Kaufmann, San Francisco, CA, 2003.

- [5] M.-S. Chen, J. Han, and P. S. Yu. Data Mining: An Overview from a Database Perspective. *IEEE Transactions on Knowledge and Data Engineering*, 8(6):866–883, 1996.
- [6] V. Cherkassky and F. Mulier. *Learning from Data: Concepts, Theory, and Methods*. Wiley Interscience, 1998.
- [7] C. Clifton, M. Kantarcioglu, and J. Vaidya. Defining privacy for data mining. In *National Science Foundation Workshop on Next Generation Data Mining*, pages 126–133, Baltimore, MD, November 2002.
- [8] P. Domingos and G. Hulten. Mining high-speed data streams. In *Proc. of the 6th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 71–80, Boston, Massachusetts, 2000. ACM Press.
- [9] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. John Wiley & Sons, Inc., New York, 2nd edition, 2001.
- [10] M. H. Dunham. *Data Mining: Introductory and Advanced Topics*. Prentice Hall, 2002.
- [11] U. M. Fayyad, G. G. Grinstein, and A. Wierse, editors. *Information Visualization in Data Mining and Knowledge Discovery*. Morgan Kaufmann Publishers, San Francisco, CA, September 2001.
- [12] U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From Data Mining to Knowledge Discovery: An Overview. In *Advances in Knowledge Discovery and Data Mining*, pages 1–34. AAAI Press, 1996.
- [13] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press, 1996.
- [14] J. H. Friedman. Data Mining and Statistics: What's the Connection? Unpublished. www-stat.stanford.edu/~jhf/ftp/dm-stat.ps, 1997.
- [15] C. Giannella, J. Han, J. Pei, X. Yan, and P. S. Yu. Mining Frequent Patterns in Data Streams at Multiple Time Granularities. In H. Kargupta, A. Joshi, K. Sivakumar, and Y. Yesha, editors, *Next Generation Data Mining*, pages 191–212. AAAI/MIT, 2003.
- [16] C. Glymour, D. Madigan, D. Pregibon, and P. Smyth. Statistical Themes and Lessons for Data Mining. *Data Mining and Knowledge Discovery*, 1(1):11–28, 1997.
- [17] R. L. Grossman, M. F. Hornick, and G. Meyer. Data mining standards initiatives. *Communications of the ACM*, 45(8):59–61, 2002.
- [18] R. L. Grossman, C. Kamath, P. Kegelmeyer, V. Kumar, and R. Namburu, editors. *Data Mining for Scientific and Engineering Applications*. Kluwer Academic Publishers, 2001.
- [19] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering Data Streams: Theory and Practice. *IEEE Transactions on Knowledge and Data Engineering*, 15(3):515–528, May/June 2003.
- [20] J. Han, R. B. Altman, V. Kumar, H. Mannila, and D. Pregibon. Emerging scientific applications in data mining. *Communications of the ACM*, 45(8):54–58, 2002.
- [21] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, San Francisco, 2001.
- [22] D. J. Hand. Data Mining: Statistics and More? *The American Statistician*, 52(2):112–118, 1998.
- [23] D. J. Hand, H. Mannila, and P. Smyth. *Principles of Data Mining*. MIT Press, 2001.
- [24] T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, Prediction*. Springer, New York, 2001.
- [25] M. Kantardzic. *Data Mining: Concepts, Models, Methods, and Algorithms*. Wiley-IEEE Press, Piscataway, NJ, 2003.

- [26] H. Kargupta and P. K. Chan, editors. *Advances in Distributed and Parallel Knowledge Discovery*. AAAI Press, September 2002.
- [27] H. Kargupta, S. Datta, Q. Wang, and K. Sivakumar. On the Privacy Preserving Properties of Random Data Perturbation Techniques. In *Proc. of the 2003 IEEE Intl. Conf. on Data Mining*, pages 99–106, Melbourne, Florida, December 2003. IEEE Computer Society.
- [28] D. Kifer, S. Ben-David, and J. Gehrke. Detecting Change in Data Streams. In *Proc. of the 30th VLDB Conf.*, pages 180–191, Toronto, Canada, 2004. Morgan Kaufmann.
- [29] D. Lambert. What Use is Statistics for Massive Data? In *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 54–62, 2000.
- [30] M. H. C. Law, N. Zhang, and A. K. Jain. Nonlinear Manifold Learning for Data Streams. In *Proc. of the SIAM Intl. Conf. on Data Mining*, Lake Buena Vista, Florida, April 2004. SIAM.
- [31] T. Mitchell. *Machine Learning*. McGraw-Hill, Boston, MA, 1997.
- [32] S. Papadimitriou, A. Brockwell, and C. Faloutsos. Adaptive, unsupervised stream mining. *VLDB Journal*, 13(3):222–239, 2004.
- [33] O. Parr Rud. *Data Mining Cookbook: Modeling Data for Marketing, Risk and Customer Relationship Management*. John Wiley & Sons, New York, NY, 2001.
- [34] D. Pyle. *Business Modeling and Data Mining*. Morgan Kaufmann, San Francisco, CA, 2003.
- [35] N. Ramakrishnan and A. Grama. Data Mining: From Serendipity to Science—Guest Editors’ Introduction. *IEEE Computer*, 32(8):34–37, 1999.
- [36] R. Roiger and M. Geatz. *Data Mining: A Tutorial Based Primer*. Addison-Wesley, 2002.
- [37] P. Smyth. Breaking out of the Black-Box: Research Challenges in Data Mining. In *Proc. of the 2001 ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 2001.
- [38] P. Smyth, D. Pregibon, and C. Faloutsos. Data-driven evolution of data mining algorithms. *Communications of the ACM*, 45(8):33–37, 2002.
- [39] V. S. Verykios, E. Bertino, I. N. Fovino, L. P. Provenza, Y. Saygin, and Y. Theodoridis. State-of-the-art in privacy preserving data mining. *SIGMOD Record*, 33(1):50–57, 2004.
- [40] J. T. L. Wang, M. J. Zaki, H. Toivonen, and D. E. Shasha, editors. *Data Mining in Bioinformatics*. Springer, September 2004.
- [41] A. R. Webb. *Statistical Pattern Recognition*. John Wiley & Sons, 2nd edition, 2002.
- [42] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 1999.
- [43] X. Wu, P. S. Yu, and G. Piatetsky-Shapiro. Data Mining: How Research Meets Practical Development? *Knowledge and Information Systems*, 5(2):248–261, 2003.
- [44] M. J. Zaki and C.-T. Ho, editors. *Large-Scale Parallel Data Mining*. Springer, September 2002.

1.7 Exercises

1. Discuss whether or not each of the following activities is a data mining task.

- (a) Dividing the customers of a company according to their gender.
 - (b) Dividing the customers of a company according to their profitability.
 - (c) Computing the total sales of a company.
 - (d) Sorting a student database based on student identification numbers.
 - (e) Predicting the outcomes of tossing a (fair) pair of dice.
 - (f) Predicting the future stock price of a company using historical records.
 - (g) Monitoring the heart rate of a patient for abnormalities.
 - (h) Monitoring seismic waves for earthquake activities.
 - (i) Extracting the frequencies of a sound wave.
2. Suppose that you are employed as a data mining consultant for an Internet search engine company. Describe how data mining can help the company by giving specific examples of how techniques, such as clustering, classification, association rule mining, and anomaly detection can be applied.
3. For each of the following data sets, explain whether or not data privacy is an important issue.
- (a) Census data collected from 1900–1950.
 - (b) IP addresses and visit times of Web users who visit your Website.
 - (c) Images from Earth-orbiting satellites.
 - (d) Names and addresses of people from the telephone book.
 - (e) Names and email addresses collected from the Web.

2

Data

This chapter discusses several data-related issues that are important for successful data mining:

The Type of Data Data sets differ in a number of ways. For example, the attributes used to describe data objects can be of different types—quantitative or qualitative—and data sets may have special characteristics; e.g., some data sets contain time series or objects with explicit relationships to one another. Not surprisingly, the type of data determines which tools and techniques can be used to analyze the data. Furthermore, new research in data mining is often driven by the need to accommodate new application areas and their new types of data.

The Quality of the Data Data is often far from perfect. While most data mining techniques can tolerate some level of imperfection in the data, a focus on understanding and improving data quality typically improves the quality of the resulting analysis. Data quality issues that often need to be addressed include the presence of noise and outliers; missing, inconsistent, or duplicate data; and data that is biased or, in some other way, unrepresentative of the phenomenon or population that the data is supposed to describe.

Preprocessing Steps to Make the Data More Suitable for Data Mining Often, the raw data must be processed in order to make it suitable for analysis. While one objective may be to improve data quality, other goals focus on modifying the data so that it better fits a specified data mining technique or tool. For example, a continuous attribute, e.g., length, may need to be transformed into an attribute with discrete categories, e.g., *short*, *medium*, or *long*, in order to apply a particular technique. As another example, the

number of attributes in a data set is often reduced because many techniques are more effective when the data has a relatively small number of attributes.

Analyzing Data in Terms of Its Relationships One approach to data analysis is to find relationships among the data objects and then perform the remaining analysis using these relationships rather than the data objects themselves. For instance, we can compute the similarity or distance between pairs of objects and then perform the analysis—clustering, classification, or anomaly detection—based on these similarities or distances. There are many such similarity or distance measures, and the proper choice depends on the type of data and the particular application.

Example 2.1 (An Illustration of Data-Related Issues). To further illustrate the importance of these issues, consider the following hypothetical situation. You receive an email from a medical researcher concerning a project that you are eager to work on.

Hi,

I've attached the data file that I mentioned in my previous email.

Each line contains the information for a single patient and consists of five fields. We want to predict the last field using the other fields.

I don't have time to provide any more information about the data since I'm going out of town for a couple of days, but hopefully that won't slow you down too much. And if you don't mind, could we meet when I get back to discuss your preliminary results? I might invite a few other members of my team.

Thanks and see you in a couple of days.

Despite some misgivings, you proceed to analyze the data. The first few rows of the file are as follows:

012	232	33.5	0	10.7
020	121	16.9	2	210.1
027	165	24.0	0	427.6
⋮				

A brief look at the data reveals nothing strange. You put your doubts aside and start the analysis. There are only 1000 lines, a smaller data file than you had hoped for, but two days later, you feel that you have made some progress. You arrive for the meeting, and while waiting for others to arrive, you strike

up a conversation with a statistician who is working on the project. When she learns that you have also been analyzing the data from the project, she asks if you would mind giving her a brief overview of your results.

Statistician: So, you got the data for all the patients?

Data Miner: Yes. I haven't had much time for analysis, but I do have a few interesting results.

Statistician: Amazing. There were so many data issues with this set of patients that I couldn't do much.

Data Miner: Oh? I didn't hear about any possible problems.

Statistician: Well, first there is field 5, the variable we want to predict. It's common knowledge among people who analyze this type of data that results are better if you work with the log of the values, but I didn't discover this until later. Was it mentioned to you?

Data Miner: No.

Statistician: But surely you heard about what happened to field 4? It's supposed to be measured on a scale from 1 to 10, with 0 indicating a missing value, but because of a data entry error, all 10's were changed into 0's. Unfortunately, since some of the patients have missing values for this field, it's impossible to say whether a 0 in this field is a real 0 or a 10. Quite a few of the records have that problem.

Data Miner: Interesting. Were there any other problems?

Statistician: Yes, fields 2 and 3 are basically the same, but I assume that you probably noticed that.

Data Miner: Yes, but these fields were only weak predictors of field 5.

Statistician: Anyway, given all those problems, I'm surprised you were able to accomplish anything.

Data Miner: True, but my results are really quite good. Field 1 is a very strong predictor of field 5. I'm surprised that this wasn't noticed before.

Statistician: What? Field 1 is just an identification number.

Data Miner: Nonetheless, my results speak for themselves.

Statistician: Oh, no! I just remembered. We assigned ID numbers after we sorted the records based on field 5. There is a strong connection, but it's meaningless. Sorry.

Although this scenario represents an extreme situation, it emphasizes the importance of “knowing your data.” To that end, this chapter will address each of the four issues mentioned above, outlining some of the basic challenges and standard approaches.

2.1 Types of Data

A **data set** can often be viewed as a collection of **data objects**. Other names for a data object are *record*, *point*, *vector*, *pattern*, *event*, *case*, *sample*, *observation*, or *entity*. In turn, data objects are described by a number of **attributes** that capture the basic characteristics of an object, such as the mass of a physical object or the time at which an event occurred. Other names for an attribute are *variable*, *characteristic*, *field*, *feature*, or *dimension*.

Example 2.2 (Student Information). Often, a data set is a file, in which the objects are records (or rows) in the file and each field (or column) corresponds to an attribute. For example, Table 2.1 shows a data set that consists of student information. Each row corresponds to a student and each column is an attribute that describes some aspect of a student, such as grade point average (GPA) or identification number (ID).

Table 2.1. A sample data set containing student information.

Student ID	Year	Grade Point Average (GPA)	...
	:		
1034262	Senior	3.24	...
1052663	Sophomore	3.51	...
1082246	Freshman	3.62	...
	:		

Although record-based data sets are common, either in flat files or relational database systems, there are other important types of data sets and systems for storing data. In Section 2.1.2, we will discuss some of the types of data sets that are commonly encountered in data mining. However, we first consider attributes.

2.1.1 Attributes and Measurement

In this section we address the issue of describing data by considering what types of attributes are used to describe data objects. We first define an attribute, then consider what we mean by the type of an attribute, and finally describe the types of attributes that are commonly encountered.

What Is an attribute?

We start with a more detailed definition of an attribute.

Definition 2.1. An **attribute** is a property or characteristic of an object that may vary, either from one object to another or from one time to another.

For example, eye color varies from person to person, while the temperature of an object varies over time. Note that eye color is a symbolic attribute with a small number of possible values *{brown, black, blue, green, hazel, etc.}*, while temperature is a numerical attribute with a potentially unlimited number of values.

At the most basic level, attributes are not about numbers or symbols. However, to discuss and more precisely analyze the characteristics of objects, we assign numbers or symbols to them. To do this in a well-defined way, we need a measurement scale.

Definition 2.2. A **measurement scale** is a rule (function) that associates a numerical or symbolic value with an attribute of an object.

Formally, the process of **measurement** is the application of a measurement scale to associate a value with a particular attribute of a specific object. While this may seem a bit abstract, we engage in the process of measurement all the time. For instance, we step on a bathroom scale to determine our weight, we classify someone as male or female, or we count the number of chairs in a room to see if there will be enough to seat all the people coming to a meeting. In all these cases, the “physical value” of an attribute of an object is mapped to a numerical or symbolic value.

With this background, we can now discuss the type of an attribute, a concept that is important in determining if a particular data analysis technique is consistent with a specific type of attribute.

The Type of an Attribute

It should be apparent from the previous discussion that the properties of an attribute need not be the same as the properties of the values used to mea-

sure it. In other words, the values used to represent an attribute may have properties that are not properties of the attribute itself, and vice versa. This is illustrated with two examples.

Example 2.3 (Employee Age and ID Number). Two attributes that might be associated with an employee are *ID* and *age* (in years). Both of these attributes can be represented as integers. However, while it is reasonable to talk about the average age of an employee, it makes no sense to talk about the average employee ID. Indeed, the only aspect of employees that we want to capture with the ID attribute is that they are distinct. Consequently, the only valid operation for employee IDs is to test whether they are equal. There is no hint of this limitation, however, when integers are used to represent the employee ID attribute. For the age attribute, the properties of the integers used to represent age are very much the properties of the attribute. Even so, the correspondence is not complete since, for example, ages have a maximum, while integers do not. ■

Example 2.4 (Length of Line Segments). Consider Figure 2.1, which shows some objects—line segments—and how the length attribute of these objects can be mapped to numbers in two different ways. Each successive line segment, going from the top to the bottom, is formed by appending the topmost line segment to itself. Thus, the second line segment from the top is formed by appending the topmost line segment to itself twice, the third line segment from the top is formed by appending the topmost line segment to itself three times, and so forth. In a very real (physical) sense, all the line segments are multiples of the first. This fact is captured by the measurements on the right-hand side of the figure, but not by those on the left hand-side. More specifically, the measurement scale on the left-hand side captures only the ordering of the length attribute, while the scale on the right-hand side captures both the ordering and additivity properties. Thus, an attribute can be measured in a way that does not capture all the properties of the attribute. ■

The type of an attribute should tell us what properties of the attribute are reflected in the values used to measure it. Knowing the type of an attribute is important because it tells us which properties of the measured values are consistent with the underlying properties of the attribute, and therefore, it allows us to avoid foolish actions, such as computing the average employee ID. Note that it is common to refer to the type of an attribute as the **type of a measurement scale**.

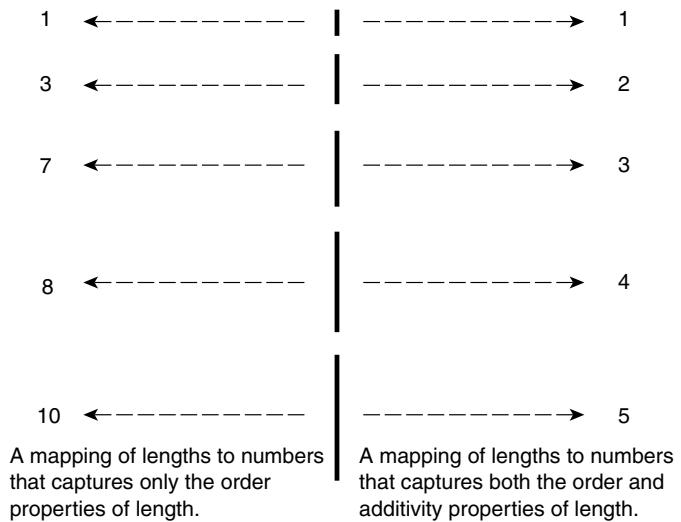


Figure 2.1. The measurement of the length of line segments on two different scales of measurement.

The Different Types of Attributes

A useful (and simple) way to specify the type of an attribute is to identify the properties of numbers that correspond to underlying properties of the attribute. For example, an attribute such as length has many of the properties of numbers. It makes sense to compare and order objects by length, as well as to talk about the differences and ratios of length. The following properties (operations) of numbers are typically used to describe attributes.

1. **Distinctness** = and \neq
2. **Order** $<$, \leq , $>$, and \geq
3. **Addition** $+$ and $-$
4. **Multiplication** $*$ and $/$

Given these properties, we can define four types of attributes: **nominal**, **ordinal**, **interval**, and **ratio**. Table 2.2 gives the definitions of these types, along with information about the statistical operations that are valid for each type. Each attribute type possesses all of the properties and operations of the attribute types above it. Consequently, any property or operation that is valid for nominal, ordinal, and interval attributes is also valid for ratio attributes. In other words, the definition of the attribute types is cumulative. However,

Table 2.2. Different attribute types.

Attribute Type	Description	Examples	Operations
Categorical (Qualitative)	Nominal	The values of a nominal attribute are just different names; i.e., nominal values provide only enough information to distinguish one object from another. $(=, \neq)$	zip codes, employee ID numbers, eye color, gender
	Ordinal	The values of an ordinal attribute provide enough information to order objects. $(<, >)$	hardness of minerals, $\{good, better, best\}$, grades, street numbers
Numeric (Quantitative)	Interval	For interval attributes, the differences between values are meaningful, i.e., a unit of measurement exists. $(+, -)$	calendar dates, temperature in Celsius or Fahrenheit
	Ratio	For ratio variables, both differences and ratios are meaningful. $(*, /)$	temperature in Kelvin, monetary quantities, counts, age, mass, length, electrical current

this does not mean that the operations appropriate for one attribute type are appropriate for the attribute types above it.

Nominal and ordinal attributes are collectively referred to as **categorical** or **qualitative** attributes. As the name suggests, qualitative attributes, such as employee ID, lack most of the properties of numbers. Even if they are represented by numbers, i.e., integers, they should be treated more like symbols. The remaining two types of attributes, interval and ratio, are collectively referred to as **quantitative** or **numeric** attributes. Quantitative attributes are represented by numbers and have most of the properties of numbers. Note that quantitative attributes can be integer-valued or continuous.

The types of attributes can also be described in terms of transformations that do not change the meaning of an attribute. Indeed, S. Smith Stevens, the psychologist who originally defined the types of attributes shown in Table 2.2, defined them in terms of these **permissible transformations**. For example,

Table 2.3. Transformations that define attribute levels.

Attribute Type		Transformation	Comment
Categorical (Qualitative)	Nominal	Any one-to-one mapping, e.g., a permutation of values	If all employee ID numbers are reassigned, it will not make any difference.
	Ordinal	An order-preserving change of values, i.e., $new_value = f(old_value)$, where f is a monotonic function.	An attribute encompassing the notion of good, better, best can be represented equally well by the values $\{1, 2, 3\}$ or by $\{0.5, 1, 10\}$.
Numeric (Quantitative)	Interval	$new_value = a * old_value + b$, a and b constants.	The Fahrenheit and Celsius temperature scales differ in the location of their zero value and the size of a degree (unit).
	Ratio	$new_value = a * old_value$	Length can be measured in meters or feet.

the meaning of a length attribute is unchanged if it is measured in meters instead of feet.

The statistical operations that make sense for a particular type of attribute are those that will yield the same results when the attribute is transformed using a transformation that preserves the attribute's meaning. To illustrate, the average length of a set of objects is different when measured in meters rather than in feet, but both averages represent the same length. Table 2.3 shows the permissible (meaning-preserving) transformations for the four attribute types of Table 2.2.

Example 2.5 (Temperature Scales). Temperature provides a good illustration of some of the concepts that have been described. First, temperature can be either an interval or a ratio attribute, depending on its measurement scale. When measured on the Kelvin scale, a temperature of 2° is, in a physically meaningful way, twice that of a temperature of 1° . This is not true when temperature is measured on either the Celsius or Fahrenheit scales, because, physically, a temperature of 1° Fahrenheit (Celsius) is not much different than a temperature of 2° Fahrenheit (Celsius). The problem is that the zero points of the Fahrenheit and Celsius scales are, in a physical sense, arbitrary, and therefore, the ratio of two Celsius or Fahrenheit temperatures is not physically meaningful. ■

Describing Attributes by the Number of Values

An independent way of distinguishing between attributes is by the number of values they can take.

Discrete A discrete attribute has a finite or countably infinite set of values.

Such attributes can be categorical, such as zip codes or ID numbers, or numeric, such as counts. Discrete attributes are often represented using integer variables. **Binary attributes** are a special case of discrete attributes and assume only two values, e.g., true/false, yes/no, male/female, or 0/1. Binary attributes are often represented as Boolean variables, or as integer variables that only take the values 0 or 1.

Continuous A continuous attribute is one whose values are real numbers. Examples include attributes such as temperature, height, or weight. Continuous attributes are typically represented as floating-point variables. Practically, real values can only be measured and represented with limited precision.

In theory, any of the measurement scale types—nominal, ordinal, interval, and ratio—could be combined with any of the types based on the number of attribute values—binary, discrete, and continuous. However, some combinations occur only infrequently or do not make much sense. For instance, it is difficult to think of a realistic data set that contains a continuous binary attribute. Typically, nominal and ordinal attributes are binary or discrete, while interval and ratio attributes are continuous. However, **count attributes**, which are discrete, are also ratio attributes.

Asymmetric Attributes

For asymmetric attributes, only presence—a non-zero attribute value—is regarded as important. Consider a data set where each object is a student and each attribute records whether or not a student took a particular course at a university. For a specific student, an attribute has a value of 1 if the student took the course associated with that attribute and a value of 0 otherwise. Because students take only a small fraction of all available courses, most of the values in such a data set would be 0. Therefore, it is more meaningful and more efficient to focus on the non-zero values. To illustrate, if students are compared on the basis of the courses they don't take, then most students would seem very similar, at least if the number of courses is large. Binary attributes where only non-zero values are important are called **asymmetric**

binary attributes. This type of attribute is particularly important for association analysis, which is discussed in Chapter 6. It is also possible to have discrete or continuous asymmetric features. For instance, if the number of credits associated with each course is recorded, then the resulting data set will consist of **asymmetric discrete** or **continuous attributes**.

2.1.2 Types of Data Sets

There are many types of data sets, and as the field of data mining develops and matures, a greater variety of data sets become available for analysis. In this section, we describe some of the most common types. For convenience, we have grouped the types of data sets into three groups: record data, graph-based data, and ordered data. These categories do not cover all possibilities and other groupings are certainly possible.

General Characteristics of Data Sets

Before providing details of specific kinds of data sets, we discuss three characteristics that apply to many data sets and have a significant impact on the data mining techniques that are used: dimensionality, sparsity, and resolution.

Dimensionality The dimensionality of a data set is the number of attributes that the objects in the data set possess. Data with a small number of dimensions tends to be qualitatively different than moderate or high-dimensional data. Indeed, the difficulties associated with analyzing high-dimensional data are sometimes referred to as the **curse of dimensionality**. Because of this, an important motivation in preprocessing the data is **dimensionality reduction**. These issues are discussed in more depth later in this chapter and in Appendix B.

Sparsity For some data sets, such as those with asymmetric features, most attributes of an object have values of 0; in many cases, fewer than 1% of the entries are non-zero. In practical terms, sparsity is an advantage because usually only the non-zero values need to be stored and manipulated. This results in significant savings with respect to computation time and storage. Furthermore, some data mining algorithms work well only for sparse data.

Resolution It is frequently possible to obtain data at different levels of resolution, and often the properties of the data are different at different resolutions. For instance, the surface of the Earth seems very uneven at a resolution of a

few meters, but is relatively smooth at a resolution of tens of kilometers. The patterns in the data also depend on the level of resolution. If the resolution is too fine, a pattern may not be visible or may be buried in noise; if the resolution is too coarse, the pattern may disappear. For example, variations in atmospheric pressure on a scale of hours reflect the movement of storms and other weather systems. On a scale of months, such phenomena are not detectable.

Record Data

Much data mining work assumes that the data set is a collection of records (data objects), each of which consists of a fixed set of data fields (attributes). See Figure 2.2(a). For the most basic form of record data, there is no explicit relationship among records or data fields, and every record (object) has the same set of attributes. Record data is usually stored either in **flat** files or in relational databases. Relational databases are certainly more than a collection of records, but data mining often does not use any of the additional information available in a relational database. Rather, the database serves as a convenient place to find records. Different types of record data are described below and are illustrated in Figure 2.2.

Transaction or Market Basket Data Transaction data is a special type of record data, where each record (transaction) involves a set of items. Consider a grocery store. The set of products purchased by a customer during one shopping trip constitutes a transaction, while the individual products that were purchased are the items. This type of data is called **market basket data** because the items in each record are the products in a person’s “market basket.” Transaction data is a collection of sets of items, but it can be viewed as a set of records whose fields are asymmetric attributes. Most often, the attributes are binary, indicating whether or not an item was purchased, but more generally, the attributes can be discrete or continuous, such as the number of items purchased or the amount spent on those items. Figure 2.2(b) shows a sample transaction data set. Each row represents the purchases of a particular customer at a particular time.

The Data Matrix If the data objects in a collection of data all have the same fixed set of numeric attributes, then the data objects can be thought of as points (vectors) in a multidimensional space, where each dimension represents a distinct attribute describing the object. A set of such data objects can be interpreted as an m by n matrix, where there are m rows, one for each object,

<i>Tid</i>	Refund	Marital Status	Taxable Income	Defaulted Borrower
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

(a) Record data.

<i>TID</i>	ITEMS
1	Bread, Soda, Milk
2	Beer, Bread
3	Beer, Soda, Diaper, Milk
4	Beer, Bread, Diaper, Milk
5	Soda, Diaper, Milk

(b) Transaction data.

Projection of x Load	Projection of y Load	Distance	Load	Thickness
10.23	5.27	15.22	27	1.2
12.65	6.25	16.22	22	1.1
13.54	7.23	17.34	23	1.2
14.27	8.43	18.45	25	0.9

(c) Data matrix.

	team	coach	play	ball	score	game	win	lost	timeout	season
Document 1	3	0	5	0	2	6	0	2	0	2
Document 2	0	7	0	2	1	0	0	3	0	0
Document 3	0	1	0	0	1	2	2	0	3	0

(d) Document-term matrix.

Figure 2.2. Different variations of record data.

and n columns, one for each attribute. (A representation that has data objects as columns and attributes as rows is also fine.) This matrix is called a **data matrix** or a **pattern matrix**. A data matrix is a variation of record data, but because it consists of numeric attributes, standard matrix operation can be applied to transform and manipulate the data. Therefore, the data matrix is the standard data format for most statistical data. Figure 2.2(c) shows a sample data matrix.

The Sparse Data Matrix A sparse data matrix is a special case of a data matrix in which the attributes are of the same type and are asymmetric; i.e., only non-zero values are important. Transaction data is an example of a sparse data matrix that has only 0–1 entries. Another common example is document data. In particular, if the order of the terms (words) in a document is ignored,

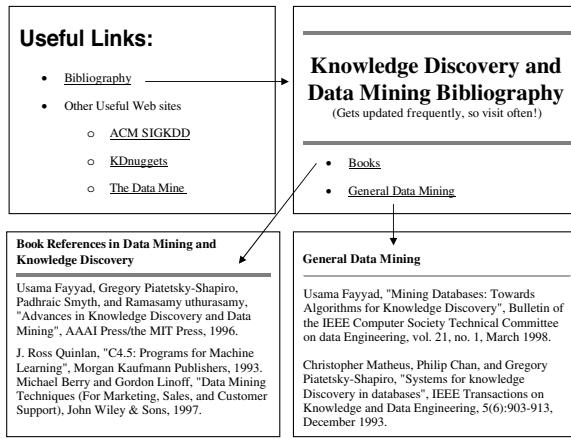
then a document can be represented as a term vector, where each term is a component (attribute) of the vector and the value of each component is the number of times the corresponding term occurs in the document. This representation of a collection of documents is often called a **document-term matrix**. Figure 2.2(d) shows a sample document-term matrix. The documents are the rows of this matrix, while the terms are the columns. In practice, only the non-zero entries of sparse data matrices are stored.

Graph-Based Data

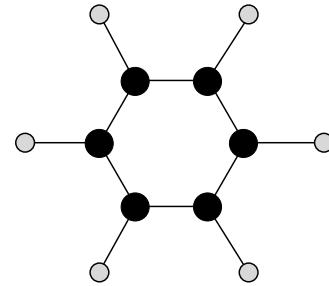
A graph can sometimes be a convenient and powerful representation for data. We consider two specific cases: (1) the graph captures relationships among data objects and (2) the data objects themselves are represented as graphs.

Data with Relationships among Objects The relationships among objects frequently convey important information. In such cases, the data is often represented as a graph. In particular, the data objects are mapped to nodes of the graph, while the relationships among objects are captured by the links between objects and link properties, such as direction and weight. Consider Web pages on the World Wide Web, which contain both text and links to other pages. In order to process search queries, Web search engines collect and process Web pages to extract their contents. It is well known, however, that the links to and from each page provide a great deal of information about the relevance of a Web page to a query, and thus, must also be taken into consideration. Figure 2.3(a) shows a set of linked Web pages.

Data with Objects That Are Graphs If objects have structure, that is, the objects contain subobjects that have relationships, then such objects are frequently represented as graphs. For example, the structure of chemical compounds can be represented by a graph, where the nodes are atoms and the links between nodes are chemical bonds. Figure 2.3(b) shows a ball-and-stick diagram of the chemical compound benzene, which contains atoms of carbon (black) and hydrogen (gray). A graph representation makes it possible to determine which substructures occur frequently in a set of compounds and to ascertain whether the presence of any of these substructures is associated with the presence or absence of certain chemical properties, such as melting point or heat of formation. Substructure mining, which is a branch of data mining that analyzes such data, is considered in Section 7.5.



(a) Linked Web pages.



(b) Benzene molecule.

Figure 2.3. Different variations of graph data.

Ordered Data

For some types of data, the attributes have relationships that involve order in time or space. Different types of ordered data are described next and are shown in Figure 2.4.

Sequential Data Sequential data, also referred to as **temporal data**, can be thought of as an extension of record data, where each record has a time associated with it. Consider a retail transaction data set that also stores the time at which the transaction took place. This time information makes it possible to find patterns such as “candy sales peak before Halloween.” A time can also be associated with each attribute. For example, each record could be the purchase history of a customer, with a listing of items purchased at different times. Using this information, it is possible to find patterns such as “people who buy DVD players tend to buy DVDs in the period immediately following the purchase.”

Figure 2.4(a) shows an example of sequential transaction data. There are five different times— t_1, t_2, t_3, t_4 , and t_5 ; three different customers—C1,

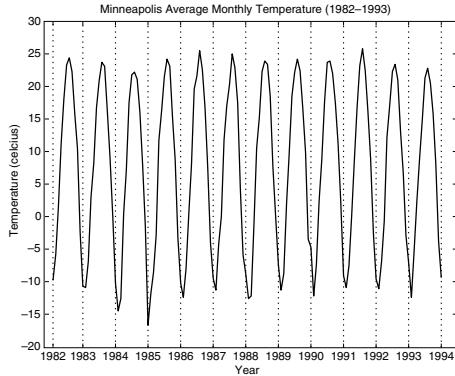
Time	Customer	Items Purchased
t1	C1	A, B
t2	C3	A, C
t2	C1	C, D
t3	C2	A, D
t4	C2	E
t5	C1	A, E

Customer	Time and Items Purchased
C1	(t1: A,B) (t2:C,D) (t5:A,E)
C2	(t3: A, D) (t4: E)
C3	(t2: A, C)

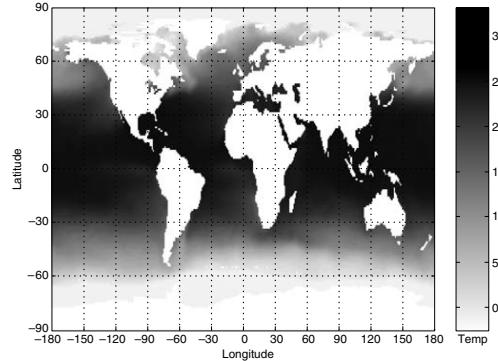
(a) Sequential transaction data.

GGTTCCGCCTTCAGCCCCGCGCC
CGCAGGGCCCGCCCCGCGCCGTC
GAGAAGGGCCCGCCTGGCGGGCG
GGGGGAGGCGGGGCCGCCCAGAC
CCAACCGAGTCCGACCAGGTGCC
CCCTCTGCTGGCCTAGACCTGA
GCTCATTAGGCGGCAGCGGACAG
GCCAAGTAGAACACACGCGAAGCGC
TGGGCTGCCTGCTGCGACCAGGG

(b) Genomic sequence data.



(c) Temperature time series.



(d) Spatial temperature data.

Figure 2.4. Different variations of ordered data.

C2, and C3; and five different items—A, B, C, D, and E. In the top table, each row corresponds to the items purchased at a particular time by each customer. For instance, at time t_3 , customer C2 purchased items A and D. In the bottom table, the same information is displayed, but each row corresponds to a particular customer. Each row contains information on each transaction involving the customer, where a transaction is considered to be a set of items and the time at which those items were purchased. For example, customer C3 bought items A and C at time t_2 .

Sequence Data Sequence data consists of a data set that is a sequence of individual entities, such as a sequence of words or letters. It is quite similar to sequential data, except that there are no time stamps; instead, there are positions in an ordered sequence. For example, the genetic information of plants and animals can be represented in the form of sequences of nucleotides that are known as genes. Many of the problems associated with genetic sequence data involve predicting similarities in the structure and function of genes from similarities in nucleotide sequences. Figure 2.4(b) shows a section of the human genetic code expressed using the four nucleotides from which all DNA is constructed: A, T, G, and C.

Time Series Data Time series data is a special type of sequential data in which each record is a **time series**, i.e., a series of measurements taken over time. For example, a financial data set might contain objects that are time series of the daily prices of various stocks. As another example, consider Figure 2.4(c), which shows a time series of the average monthly temperature for Minneapolis during the years 1982 to 1994. When working with temporal data, it is important to consider **temporal autocorrelation**; i.e., if two measurements are close in time, then the values of those measurements are often very similar.

Spatial Data Some objects have spatial attributes, such as positions or areas, as well as other types of attributes. An example of spatial data is weather data (precipitation, temperature, pressure) that is collected for a variety of geographical locations. An important aspect of spatial data is **spatial autocorrelation**; i.e., objects that are physically close tend to be similar in other ways as well. Thus, two points on the Earth that are close to each other usually have similar values for temperature and rainfall.

Important examples of spatial data are the science and engineering data sets that are the result of measurements or model output taken at regularly or irregularly distributed points on a two- or three-dimensional grid or mesh. For instance, Earth science data sets record the temperature or pressure measured at points (grid cells) on latitude-longitude spherical grids of various resolutions, e.g., 1° by 1° . (See Figure 2.4(d).) As another example, in the simulation of the flow of a gas, the speed and direction of flow can be recorded for each grid point in the simulation.

Handling Non-Record Data

Most data mining algorithms are designed for record data or its variations, such as transaction data and data matrices. Record-oriented techniques can be applied to non-record data by extracting features from data objects and using these features to create a record corresponding to each object. Consider the chemical structure data that was described earlier. Given a set of common substructures, each compound can be represented as a record with binary attributes that indicate whether a compound contains a specific substructure. Such a representation is actually a transaction data set, where the transactions are the compounds and the items are the substructures.

In some cases, it is easy to represent the data in a record format, but this type of representation does not capture all the information in the data. Consider spatio-temporal data consisting of a time series from each point on a spatial grid. This data is often stored in a data matrix, where each row represents a location and each column represents a particular point in time. However, such a representation does not explicitly capture the time relationships that are present among attributes and the spatial relationships that exist among objects. This does not mean that such a representation is inappropriate, but rather that these relationships must be taken into consideration during the analysis. For example, it would not be a good idea to use a data mining technique that assumes the attributes are statistically independent of one another.

2.2 Data Quality

Data mining applications are often applied to data that was collected for another purpose, or for future, but unspecified applications. For that reason, data mining cannot usually take advantage of the significant benefits of “addressing quality issues at the source.” In contrast, much of statistics deals with the design of experiments or surveys that achieve a prespecified level of data quality. Because preventing data quality problems is typically not an option, data mining focuses on (1) the detection and correction of data quality problems and (2) the use of algorithms that can tolerate poor data quality. The first step, detection and correction, is often called **data cleaning**.

The following sections discuss specific aspects of data quality. The focus is on measurement and data collection issues, although some application-related issues are also discussed.

2.2.1 Measurement and Data Collection Issues

It is unrealistic to expect that data will be perfect. There may be problems due to human error, limitations of measuring devices, or flaws in the data collection process. Values or even entire data objects may be missing. In other cases, there may be spurious or duplicate objects; i.e., multiple data objects that all correspond to a single “real” object. For example, there might be two different records for a person who has recently lived at two different addresses. Even if all the data is present and “looks fine,” there may be inconsistencies—a person has a height of 2 meters, but weighs only 2 kilograms.

In the next few sections, we focus on aspects of data quality that are related to data measurement and collection. We begin with a definition of measurement and data collection errors and then consider a variety of problems that involve measurement error: noise, artifacts, bias, precision, and accuracy. We conclude by discussing data quality issues that may involve both measurement and data collection problems: outliers, missing and inconsistent values, and duplicate data.

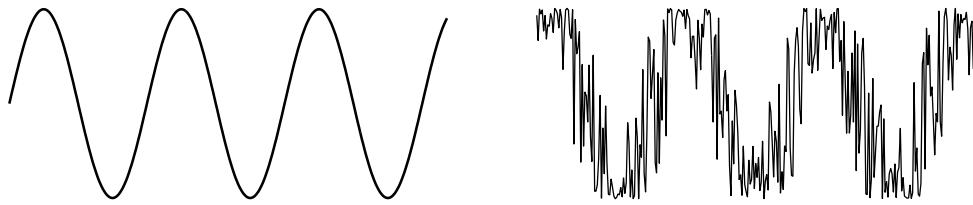
Measurement and Data Collection Errors

The term **measurement error** refers to any problem resulting from the measurement process. A common problem is that the value recorded differs from the true value to some extent. For continuous attributes, the numerical difference of the measured and true value is called the **error**. The term **data collection error** refers to errors such as omitting data objects or attribute values, or inappropriately including a data object. For example, a study of animals of a certain species might include animals of a related species that are similar in appearance to the species of interest. Both measurement errors and data collection errors can be either systematic or random.

We will only consider general types of errors. Within particular domains, there are certain types of data errors that are commonplace, and there often exist well-developed techniques for detecting and/or correcting these errors. For example, keyboard errors are common when data is entered manually, and as a result, many data entry programs have techniques for detecting and, with human intervention, correcting such errors.

Noise and Artifacts

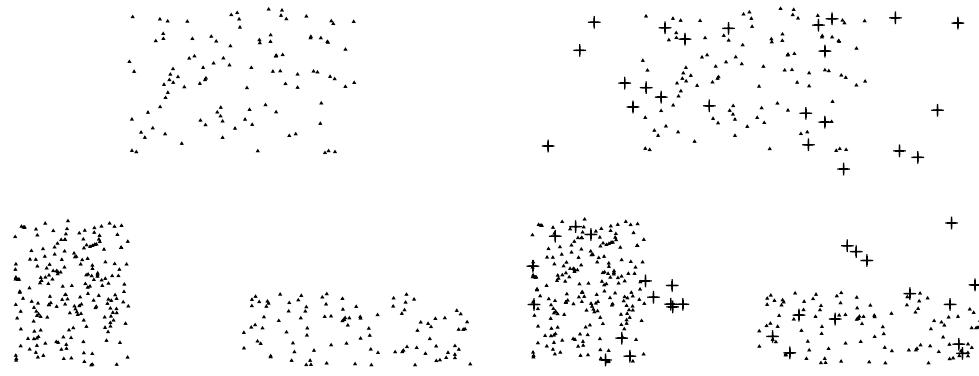
Noise is the random component of a measurement error. It may involve the distortion of a value or the addition of spurious objects. Figure 2.5 shows a time series before and after it has been disrupted by random noise. If a bit



(a) Time series.

(b) Time series with noise.

Figure 2.5. Noise in a time series context.



(a) Three groups of points.

(b) With noise points (+) added.

Figure 2.6. Noise in a spatial context.

more noise were added to the time series, its shape would be lost. Figure 2.6 shows a set of data points before and after some noise points (indicated by '+'s) have been added. Notice that some of the noise points are intermixed with the non-noise points.

The term noise is often used in connection with data that has a spatial or temporal component. In such cases, techniques from signal or image processing can frequently be used to reduce noise and thus, help to discover patterns (signals) that might be “lost in the noise.” Nonetheless, the elimination of noise is frequently difficult, and much work in data mining focuses on devising **robust algorithms** that produce acceptable results even when noise is present.

Data errors may be the result of a more deterministic phenomenon, such as a streak in the same place on a set of photographs. Such deterministic distortions of the data are often referred to as **artifacts**.

Precision, Bias, and Accuracy

In statistics and experimental science, the quality of the measurement process and the resulting data are measured by precision and bias. We provide the standard definitions, followed by a brief discussion. For the following definitions, we assume that we make repeated measurements of the same underlying quantity and use this set of values to calculate a mean (average) value that serves as our estimate of the true value.

Definition 2.3 (Precision). The closeness of repeated measurements (of the same quantity) to one another.

Definition 2.4 (Bias). A systematic variation of measurements from the quantity being measured.

Precision is often measured by the standard deviation of a set of values, while bias is measured by taking the difference between the mean of the set of values and the known value of the quantity being measured. Bias can only be determined for objects whose measured quantity is known by means external to the current situation. Suppose that we have a standard laboratory weight with a mass of 1g and want to assess the precision and bias of our new laboratory scale. We weigh the mass five times, and obtain the following five values: $\{1.015, 0.990, 1.013, 1.001, 0.986\}$. The mean of these values is 1.001, and hence, the bias is 0.001. The precision, as measured by the standard deviation, is 0.013.

It is common to use the more general term, **accuracy**, to refer to the degree of measurement error in data.

Definition 2.5 (Accuracy). The closeness of measurements to the true value of the quantity being measured.

Accuracy depends on precision and bias, but since it is a general concept, there is no specific formula for accuracy in terms of these two quantities.

One important aspect of accuracy is the use of **significant digits**. The goal is to use only as many digits to represent the result of a measurement or calculation as are justified by the precision of the data. For example, if the length of an object is measured with a meter stick whose smallest markings are millimeters, then we should only record the length of data to the nearest millimeter. The precision of such a measurement would be $\pm 0.5\text{mm}$. We do not

review the details of working with significant digits, as most readers will have encountered them in previous courses, and they are covered in considerable depth in science, engineering, and statistics textbooks.

Issues such as significant digits, precision, bias, and accuracy are sometimes overlooked, but they are important for data mining as well as statistics and science. Many times, data sets do not come with information on the precision of the data, and furthermore, the programs used for analysis return results without any such information. Nonetheless, without some understanding of the accuracy of the data and the results, an analyst runs the risk of committing serious data analysis blunders.

Outliers

Outliers are either (1) data objects that, in some sense, have characteristics that are different from most of the other data objects in the data set, or (2) values of an attribute that are unusual with respect to the typical values for that attribute. Alternatively, we can speak of **anomalous** objects or values. There is considerable leeway in the definition of an outlier, and many different definitions have been proposed by the statistics and data mining communities. Furthermore, it is important to distinguish between the notions of noise and outliers. Outliers can be legitimate data objects or values. Thus, unlike noise, outliers may sometimes be of interest. In fraud and network intrusion detection, for example, the goal is to find unusual objects or events from among a large number of normal ones. Chapter 10 discusses anomaly detection in more detail.

Missing Values

It is not unusual for an object to be missing one or more attribute values. In some cases, the information was not collected; e.g., some people decline to give their age or weight. In other cases, some attributes are not applicable to all objects; e.g., often, forms have conditional parts that are filled out only when a person answers a previous question in a certain way, but for simplicity, all fields are stored. Regardless, missing values should be taken into account during the data analysis.

There are several strategies (and variations on these strategies) for dealing with missing data, each of which may be appropriate in certain circumstances. These strategies are listed next, along with an indication of their advantages and disadvantages.

Eliminate Data Objects or Attributes A simple and effective strategy is to eliminate objects with missing values. However, even a partially specified data object contains some information, and if many objects have missing values, then a reliable analysis can be difficult or impossible. Nonetheless, if a data set has only a few objects that have missing values, then it may be expedient to omit them. A related strategy is to eliminate attributes that have missing values. This should be done with caution, however, since the eliminated attributes may be the ones that are critical to the analysis.

Estimate Missing Values Sometimes missing data can be reliably estimated. For example, consider a time series that changes in a reasonably smooth fashion, but has a few, widely scattered missing values. In such cases, the missing values can be estimated (interpolated) by using the remaining values. As another example, consider a data set that has many similar data points. In this situation, the attribute values of the points closest to the point with the missing value are often used to estimate the missing value. If the attribute is continuous, then the average attribute value of the nearest neighbors is used; if the attribute is categorical, then the most commonly occurring attribute value can be taken. For a concrete illustration, consider precipitation measurements that are recorded by ground stations. For areas not containing a ground station, the precipitation can be estimated using values observed at nearby ground stations.

Ignore the Missing Value during Analysis Many data mining approaches can be modified to ignore missing values. For example, suppose that objects are being clustered and the similarity between pairs of data objects needs to be calculated. If one or both objects of a pair have missing values for some attributes, then the similarity can be calculated by using only the attributes that do not have missing values. It is true that the similarity will only be approximate, but unless the total number of attributes is small or the number of missing values is high, this degree of inaccuracy may not matter much. Likewise, many classification schemes can be modified to work with missing values.

Inconsistent Values

Data can contain inconsistent values. Consider an address field, where both a zip code and city are listed, but the specified zip code area is not contained in that city. It may be that the individual entering this information transposed two digits, or perhaps a digit was misread when the information was scanned

from a handwritten form. Regardless of the cause of the inconsistent values, it is important to detect and, if possible, correct such problems.

Some types of inconsistencies are easy to detect. For instance, a person's height should not be negative. In other cases, it can be necessary to consult an external source of information. For example, when an insurance company processes claims for reimbursement, it checks the names and addresses on the reimbursement forms against a database of its customers.

Once an inconsistency has been detected, it is sometimes possible to correct the data. A product code may have "check" digits, or it may be possible to double-check a product code against a list of known product codes, and then correct the code if it is incorrect, but close to a known code. The correction of an inconsistency requires additional or redundant information.

Example 2.6 (Inconsistent Sea Surface Temperature). This example illustrates an inconsistency in actual time series data that measures the sea surface temperature (SST) at various points on the ocean. SST data was originally collected using ocean-based measurements from ships or buoys, but more recently, satellites have been used to gather the data. To create a long-term data set, both sources of data must be used. However, because the data comes from different sources, the two parts of the data are subtly different. This discrepancy is visually displayed in Figure 2.7, which shows the correlation of SST values between pairs of years. If a pair of years has a positive correlation, then the location corresponding to the pair of years is colored white; otherwise it is colored black. (Seasonal variations were removed from the data since, otherwise, all the years would be highly correlated.) There is a distinct change in behavior where the data has been put together in 1983. Years within each of the two groups, 1958–1982 and 1983–1999, tend to have a positive correlation with one another, but a negative correlation with years in the other group. This does not mean that this data should not be used, only that the analyst should consider the potential impact of such discrepancies on the data mining analysis. ■

Duplicate Data

A data set may include data objects that are duplicates, or almost duplicates, of one another. Many people receive duplicate mailings because they appear in a database multiple times under slightly different names. To detect and eliminate such duplicates, two main issues must be addressed. First, if there are two objects that actually represent a single object, then the values of corresponding attributes may differ, and these inconsistent values must be

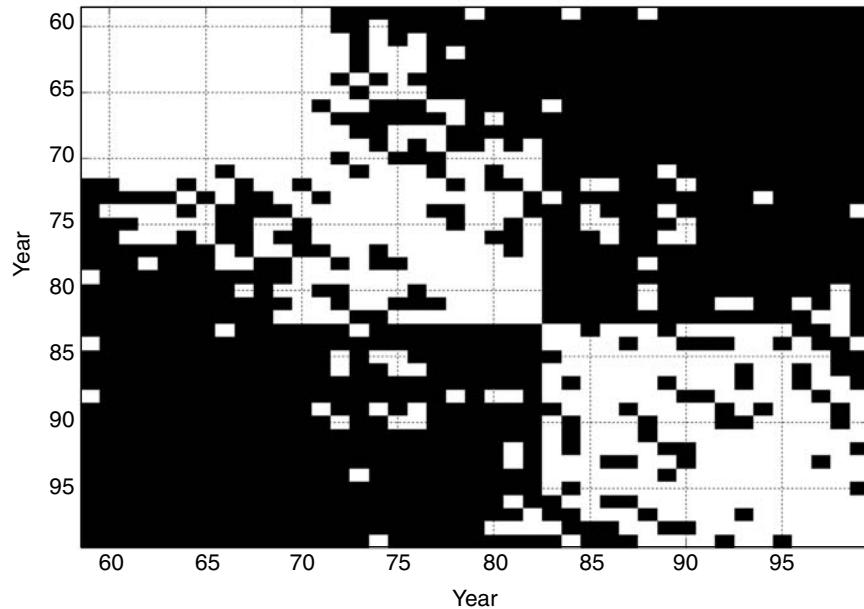


Figure 2.7. Correlation of SST data between pairs of years. White areas indicate positive correlation. Black areas indicate negative correlation.

resolved. Second, care needs to be taken to avoid accidentally combining data objects that are similar, but not duplicates, such as two distinct people with identical names. The term **deduplication** is often used to refer to the process of dealing with these issues.

In some cases, two or more objects are identical with respect to the attributes measured by the database, but they still represent different objects. Here, the duplicates are legitimate, but may still cause problems for some algorithms if the possibility of identical objects is not specifically accounted for in their design. An example of this is given in Exercise 13 on page 91.

2.2.2 Issues Related to Applications

Data quality issues can also be considered from an application viewpoint as expressed by the statement “data is of high quality if it is suitable for its intended use.” This approach to data quality has proven quite useful, particularly in business and industry. A similar viewpoint is also present in statistics and the experimental sciences, with their emphasis on the careful design of experiments to collect the data relevant to a specific hypothesis. As with quality

issues at the measurement and data collection level, there are many issues that are specific to particular applications and fields. Again, we consider only a few of the general issues.

Timeliness Some data starts to age as soon as it has been collected. In particular, if the data provides a snapshot of some ongoing phenomenon or process, such as the purchasing behavior of customers or Web browsing patterns, then this snapshot represents reality for only a limited time. If the data is out of date, then so are the models and patterns that are based on it.

Relevance The available data must contain the information necessary for the application. Consider the task of building a model that predicts the accident rate for drivers. If information about the age and gender of the driver is omitted, then it is likely that the model will have limited accuracy unless this information is indirectly available through other attributes.

Making sure that the objects in a data set are relevant is also challenging. A common problem is **sampling bias**, which occurs when a sample does not contain different types of objects in proportion to their actual occurrence in the population. For example, survey data describes only those who respond to the survey. (Other aspects of sampling are discussed further in Section 2.3.2.) Because the results of a data analysis can reflect only the data that is present, sampling bias will typically result in an erroneous analysis.

Knowledge about the Data Ideally, data sets are accompanied by documentation that describes different aspects of the data; the quality of this documentation can either aid or hinder the subsequent analysis. For example, if the documentation identifies several attributes as being strongly related, these attributes are likely to provide highly redundant information, and we may decide to keep just one. (Consider sales tax and purchase price.) If the documentation is poor, however, and fails to tell us, for example, that the missing values for a particular field are indicated with a -9999, then our analysis of the data may be faulty. Other important characteristics are the precision of the data, the type of features (nominal, ordinal, interval, ratio), the scale of measurement (e.g., meters or feet for length), and the origin of the data.

2.3 Data Preprocessing

In this section, we address the issue of which preprocessing steps should be applied to make the data more suitable for data mining. Data preprocessing

is a broad area and consists of a number of different strategies and techniques that are interrelated in complex ways. We will present some of the most important ideas and approaches, and try to point out the interrelationships among them. Specifically, we will discuss the following topics:

- Aggregation
- Sampling
- Dimensionality reduction
- Feature subset selection
- Feature creation
- Discretization and binarization
- Variable transformation

Roughly speaking, these items fall into two categories: selecting data objects and attributes for the analysis or creating/changing the attributes. In both cases the goal is to improve the data mining analysis with respect to time, cost, and quality. Details are provided in the following sections.

A quick note on terminology: In the following, we sometimes use synonyms for attribute, such as feature or variable, in order to follow common usage.

2.3.1 Aggregation

Sometimes “less is more” and this is the case with **aggregation**, the combining of two or more objects into a single object. Consider a data set consisting of transactions (data objects) recording the daily sales of products in various store locations (Minneapolis, Chicago, Paris, ...) for different days over the course of a year. See Table 2.4. One way to aggregate transactions for this data set is to replace all the transactions of a single store with a single storewide transaction. This reduces the hundreds or thousands of transactions that occur daily at a specific store to a single daily transaction, and the number of data objects is reduced to the number of stores.

An obvious issue is how an aggregate transaction is created; i.e., how the values of each attribute are combined across all the records corresponding to a particular location to create the aggregate transaction that represents the sales of a single store or date. Quantitative attributes, such as price, are typically aggregated by taking a sum or an average. A qualitative attribute, such as item, can either be omitted or summarized as the set of all the items that were sold at that location.

The data in Table 2.4 can also be viewed as a multidimensional array, where each attribute is a dimension. From this viewpoint, aggregation is the

Table 2.4. Data set containing information about customer purchases.

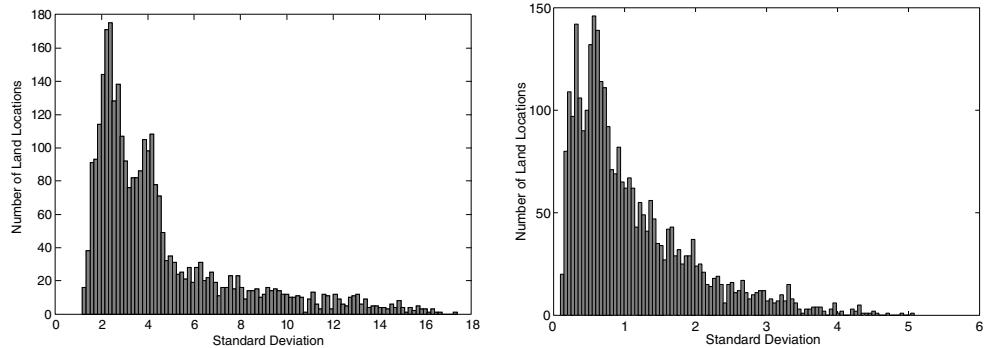
Transaction ID	Item	Store Location	Date	Price	...
:	:	:	:	:	
101123	Watch	Chicago	09/06/04	\$25.99	...
101123	Battery	Chicago	09/06/04	\$5.99	...
101124	Shoes	Minneapolis	09/06/04	\$75.00	...
:	:	:	:	:	

process of eliminating attributes, such as the type of item, or reducing the number of values for a particular attribute; e.g., reducing the possible values for date from 365 days to 12 months. This type of aggregation is commonly used in Online Analytical Processing (OLAP), which is discussed further in Chapter 3.

There are several motivations for aggregation. First, the smaller data sets resulting from data reduction require less memory and processing time, and hence, aggregation may permit the use of more expensive data mining algorithms. Second, aggregation can act as a change of scope or scale by providing a high-level view of the data instead of a low-level view. In the previous example, aggregating over store locations and months gives us a monthly, per store view of the data instead of a daily, per item view. Finally, the behavior of groups of objects or attributes is often more stable than that of individual objects or attributes. This statement reflects the statistical fact that aggregate quantities, such as averages or totals, have less variability than the individual objects being aggregated. For totals, the actual amount of variation is larger than that of individual objects (on average), but the percentage of the variation is smaller, while for means, the actual amount of variation is less than that of individual objects (on average). A disadvantage of aggregation is the potential loss of interesting details. In the store example aggregating over months loses information about which day of the week has the highest sales.

Example 2.7 (Australian Precipitation). This example is based on precipitation in Australia from the period 1982 to 1993. Figure 2.8(a) shows a histogram for the standard deviation of average monthly precipitation for 3,030 0.5° by 0.5° grid cells in Australia, while Figure 2.8(b) shows a histogram for the standard deviation of the average yearly precipitation for the same locations. The average yearly precipitation has less variability than the average monthly precipitation. All precipitation measurements (and their standard deviations) are in centimeters.

■



(a) Histogram of standard deviation of average monthly precipitation

(b) Histogram of standard deviation of average yearly precipitation

Figure 2.8. Histograms of standard deviation for monthly and yearly precipitation in Australia for the period 1982 to 1993.

2.3.2 Sampling

Sampling is a commonly used approach for selecting a subset of the data objects to be analyzed. In statistics, it has long been used for both the preliminary investigation of the data and the final data analysis. Sampling can also be very useful in data mining. However, the motivations for sampling in statistics and data mining are often different. Statisticians use sampling because obtaining the entire set of data of interest is too expensive or time consuming, while data miners sample because it is too expensive or time consuming to process all the data. In some cases, using a sampling algorithm can reduce the data size to the point where a better, but more expensive algorithm can be used.

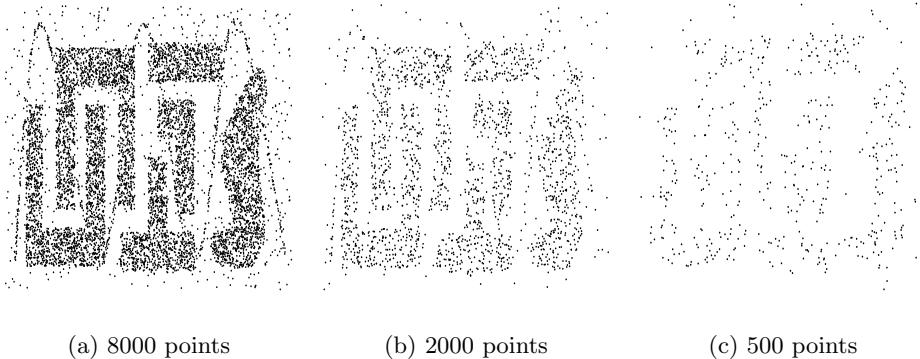
The key principle for effective sampling is the following: Using a sample will work almost as well as using the entire data set if the sample is representative. In turn, a **sample is representative** if it has approximately the same property (of interest) as the original set of data. If the mean (average) of the data objects is the property of interest, then a sample is representative if it has a mean that is close to that of the original data. Because sampling is a statistical process, the representativeness of any particular sample will vary, and the best that we can do is choose a sampling scheme that guarantees a high probability of getting a representative sample. As discussed next, this involves choosing the appropriate sample size and sampling techniques.

Sampling Approaches

There are many sampling techniques, but only a few of the most basic ones and their variations will be covered here. The simplest type of sampling is **simple random sampling**. For this type of sampling, there is an equal probability of selecting any particular item. There are two variations on random sampling (and other sampling techniques as well): (1) **sampling without replacement**—as each item is selected, it is removed from the set of all objects that together constitute the **population**, and (2) **sampling with replacement**—objects are not removed from the population as they are selected for the sample. In sampling with replacement, the same object can be picked more than once. The samples produced by the two methods are not much different when samples are relatively small compared to the data set size, but sampling with replacement is simpler to analyze since the probability of selecting any object remains constant during the sampling process.

When the population consists of different types of objects, with widely different numbers of objects, simple random sampling can fail to adequately represent those types of objects that are less frequent. This can cause problems when the analysis requires proper representation of all object types. For example, when building classification models for rare classes, it is critical that the rare classes be adequately represented in the sample. Hence, a sampling scheme that can accommodate differing frequencies for the items of interest is needed. **Stratified sampling**, which starts with prespecified groups of objects, is such an approach. In the simplest version, equal numbers of objects are drawn from each group even though the groups are of different sizes. In another variation, the number of objects drawn from each group is proportional to the size of that group.

Example 2.8 (Sampling and Loss of Information). Once a sampling technique has been selected, it is still necessary to choose the sample size. Larger sample sizes increase the probability that a sample will be representative, but they also eliminate much of the advantage of sampling. Conversely, with smaller sample sizes, patterns may be missed or erroneous patterns can be detected. Figure 2.9(a) shows a data set that contains 8000 two-dimensional points, while Figures 2.9(b) and 2.9(c) show samples from this data set of size 2000 and 500, respectively. Although most of the structure of this data set is present in the sample of 2000 points, much of the structure is missing in the sample of 500 points. ■



(a) 8000 points

(b) 2000 points

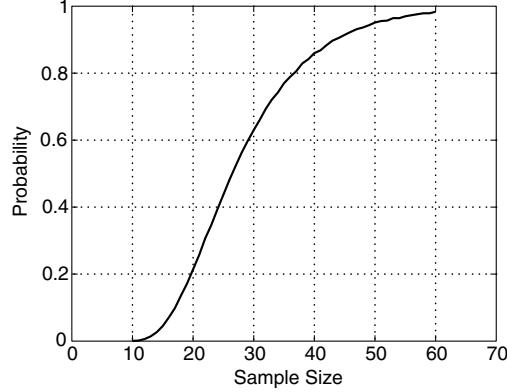
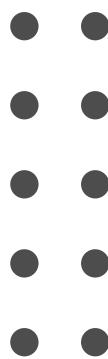
(c) 500 points

Figure 2.9. Example of the loss of structure with sampling.

Example 2.9 (Determining the Proper Sample Size). To illustrate that determining the proper sample size requires a methodical approach, consider the following task.

Given a set of data that consists of a small number of almost equal-sized groups, find at least one representative point for each of the groups. Assume that the objects in each group are highly similar to each other, but not very similar to objects in different groups. Also assume that there are a relatively small number of groups, e.g., 10. Figure 2.10(a) shows an idealized set of clusters (groups) from which these points might be drawn.

This problem can be efficiently solved using sampling. One approach is to take a small sample of data points, compute the pairwise similarities between points, and then form groups of points that are highly similar. The desired set of representative points is then obtained by taking one point from each of these groups. To follow this approach, however, we need to determine a sample size that would guarantee, with a high probability, the desired outcome; that is, that at least one point will be obtained from each cluster. Figure 2.10(b) shows the probability of getting one object from each of the 10 groups as the sample size runs from 10 to 60. Interestingly, with a sample size of 20, there is little chance (20%) of getting a sample that includes all 10 clusters. Even with a sample size of 30, there is still a moderate chance (almost 40%) of getting a sample that doesn't contain objects from all 10 clusters. This issue is further explored in the context of clustering by Exercise 4 on page 559.



(a) Ten groups of points.

(b) Probability a sample contains points from each of 10 groups.

Figure 2.10. Finding representative points from 10 groups.

Progressive Sampling

The proper sample size can be difficult to determine, so **adaptive or progressive sampling** schemes are sometimes used. These approaches start with a small sample, and then increase the sample size until a sample of sufficient size has been obtained. While this technique eliminates the need to determine the correct sample size initially, it requires that there be a way to evaluate the sample to judge if it is large enough.

Suppose, for instance, that progressive sampling is used to learn a predictive model. Although the accuracy of predictive models increases as the sample size increases, at some point the increase in accuracy levels off. We want to stop increasing the sample size at this leveling-off point. By keeping track of the change in accuracy of the model as we take progressively larger samples, and by taking other samples close to the size of the current one, we can get an estimate as to how close we are to this leveling-off point, and thus, stop sampling.

2.3.3 Dimensionality Reduction

Data sets can have a large number of features. Consider a set of documents, where each document is represented by a vector whose components are the frequencies with which each word occurs in the document. In such cases,

there are typically thousands or tens of thousands of attributes (components), one for each word in the vocabulary. As another example, consider a set of time series consisting of the daily closing price of various stocks over a period of 30 years. In this case, the attributes, which are the prices on specific days, again number in the thousands.

There are a variety of benefits to dimensionality reduction. A key benefit is that many data mining algorithms work better if the dimensionality—the number of attributes in the data—is lower. This is partly because dimensionality reduction can eliminate irrelevant features and reduce noise and partly because of the curse of dimensionality, which is explained below. Another benefit is that a reduction of dimensionality can lead to a more understandable model because the model may involve fewer attributes. Also, dimensionality reduction may allow the data to be more easily visualized. Even if dimensionality reduction doesn't reduce the data to two or three dimensions, data is often visualized by looking at pairs or triplets of attributes, and the number of such combinations is greatly reduced. Finally, the amount of time and memory required by the data mining algorithm is reduced with a reduction in dimensionality.

The term dimensionality reduction is often reserved for those techniques that reduce the dimensionality of a data set by creating new attributes that are a combination of the old attributes. The reduction of dimensionality by selecting new attributes that are a subset of the old is known as feature subset selection or feature selection. It will be discussed in Section 2.3.4.

In the remainder of this section, we briefly introduce two important topics: the curse of dimensionality and dimensionality reduction techniques based on linear algebra approaches such as principal components analysis (PCA). More details on dimensionality reduction can be found in Appendix B.

The Curse of Dimensionality

The curse of dimensionality refers to the phenomenon that many types of data analysis become significantly harder as the dimensionality of the data increases. Specifically, as dimensionality increases, the data becomes increasingly sparse in the space that it occupies. For classification, this can mean that there are not enough data objects to allow the creation of a model that reliably assigns a class to all possible objects. For clustering, the definitions of density and the distance between points, which are critical for clustering, become less meaningful. (This is discussed further in Sections 9.1.2, 9.4.5, and 9.4.7.) As a result, many clustering and classification algorithms (and other

data analysis algorithms) have trouble with high-dimensional data—reduced classification accuracy and poor quality clusters.

Linear Algebra Techniques for Dimensionality Reduction

Some of the most common approaches for dimensionality reduction, particularly for continuous data, use techniques from linear algebra to project the data from a high-dimensional space into a lower-dimensional space. **Principal Components Analysis (PCA)** is a linear algebra technique for continuous attributes that finds new attributes (principal components) that (1) are linear combinations of the original attributes, (2) are **orthogonal** (perpendicular) to each other, and (3) capture the maximum amount of variation in the data. For example, the first two principal components capture as much of the variation in the data as is possible with two orthogonal attributes that are linear combinations of the original attributes. **Singular Value Decomposition (SVD)** is a linear algebra technique that is related to PCA and is also commonly used for dimensionality reduction. For additional details, see Appendices A and B.

2.3.4 Feature Subset Selection

Another way to reduce the dimensionality is to use only a subset of the features. While it might seem that such an approach would lose information, this is not the case if redundant and irrelevant features are present. **Redundant features** duplicate much or all of the information contained in one or more other attributes. For example, the purchase price of a product and the amount of sales tax paid contain much of the same information. **Irrelevant features** contain almost no useful information for the data mining task at hand. For instance, students' ID numbers are irrelevant to the task of predicting students' grade point averages. Redundant and irrelevant features can reduce classification accuracy and the quality of the clusters that are found.

While some irrelevant and redundant attributes can be eliminated immediately by using common sense or domain knowledge, selecting the best subset of features frequently requires a systematic approach. The ideal approach to feature selection is to try all possible subsets of features as input to the data mining algorithm of interest, and then take the subset that produces the best results. This method has the advantage of reflecting the objective and bias of the data mining algorithm that will eventually be used. Unfortunately, since the number of subsets involving n attributes is 2^n , such an approach is impractical in most situations and alternative strategies are needed. There are three standard approaches to feature selection: embedded, filter, and wrapper.

Embedded approaches Feature selection occurs naturally as part of the data mining algorithm. Specifically, during the operation of the data mining algorithm, the algorithm itself decides which attributes to use and which to ignore. Algorithms for building decision tree classifiers, which are discussed in Chapter 4, often operate in this manner.

Filter approaches Features are selected before the data mining algorithm is run, using some approach that is independent of the data mining task. For example, we might select sets of attributes whose pairwise correlation is as low as possible.

Wrapper approaches These methods use the target data mining algorithm as a black box to find the best subset of attributes, in a way similar to that of the ideal algorithm described above, but typically without enumerating all possible subsets.

Since the embedded approaches are algorithm-specific, only the filter and wrapper approaches will be discussed further here.

An Architecture for Feature Subset Selection

It is possible to encompass both the filter and wrapper approaches within a common architecture. The feature selection process is viewed as consisting of four parts: a measure for evaluating a subset, a search strategy that controls the generation of a new subset of features, a stopping criterion, and a validation procedure. Filter methods and wrapper methods differ only in the way in which they evaluate a subset of features. For a wrapper method, subset evaluation uses the target data mining algorithm, while for a filter approach, the evaluation technique is distinct from the target data mining algorithm. The following discussion provides some details of this approach, which is summarized in Figure 2.11.

Conceptually, feature subset selection is a search over all possible subsets of features. Many different types of search strategies can be used, but the search strategy should be computationally inexpensive and should find optimal or near optimal sets of features. It is usually not possible to satisfy both requirements, and thus, tradeoffs are necessary.

An integral part of the search is an evaluation step to judge how the current subset of features compares to others that have been considered. This requires an evaluation measure that attempts to determine the goodness of a subset of attributes with respect to a particular data mining task, such as classification

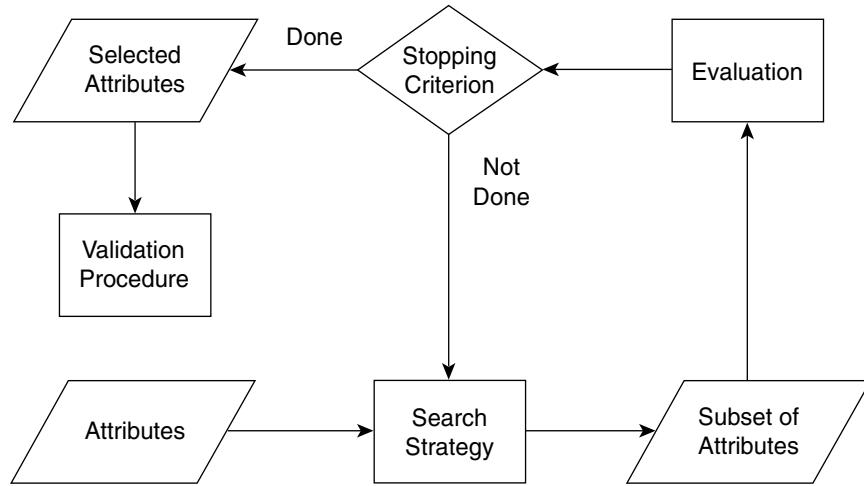


Figure 2.11. Flowchart of a feature subset selection process.

or clustering. For the filter approach, such measures attempt to predict how well the actual data mining algorithm will perform on a given set of attributes. For the wrapper approach, where evaluation consists of actually running the target data mining application, the subset evaluation function is simply the criterion normally used to measure the result of the data mining.

Because the number of subsets can be enormous and it is impractical to examine them all, some sort of stopping criterion is necessary. This strategy is usually based on one or more conditions involving the following: the number of iterations, whether the value of the subset evaluation measure is optimal or exceeds a certain threshold, whether a subset of a certain size has been obtained, whether simultaneous size and evaluation criteria have been achieved, and whether any improvement can be achieved by the options available to the search strategy.

Finally, once a subset of features has been selected, the results of the target data mining algorithm on the selected subset should be validated. A straightforward evaluation approach is to run the algorithm with the full set of features and compare the full results to results obtained using the subset of features. Hopefully, the subset of features will produce results that are better than or almost as good as those produced when using all features. Another validation approach is to use a number of different feature selection algorithms to obtain subsets of features and then compare the results of running the data mining algorithm on each subset.

Feature Weighting

Feature weighting is an alternative to keeping or eliminating features. More important features are assigned a higher weight, while less important features are given a lower weight. These weights are sometimes assigned based on domain knowledge about the relative importance of features. Alternatively, they may be determined automatically. For example, some classification schemes, such as support vector machines (Chapter 5), produce classification models in which each feature is given a weight. Features with larger weights play a more important role in the model. The normalization of objects that takes place when computing the cosine similarity (Section 2.4.5) can also be regarded as a type of feature weighting.

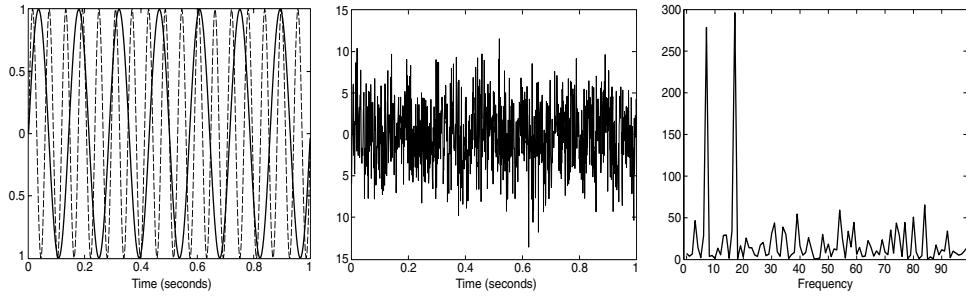
2.3.5 Feature Creation

It is frequently possible to create, from the original attributes, a new set of attributes that captures the important information in a data set much more effectively. Furthermore, the number of new attributes can be smaller than the number of original attributes, allowing us to reap all the previously described benefits of dimensionality reduction. Three related methodologies for creating new attributes are described next: feature extraction, mapping the data to a new space, and feature construction.

Feature Extraction

The creation of a new set of features from the original raw data is known as **feature extraction**. Consider a set of photographs, where each photograph is to be classified according to whether or not it contains a human face. The raw data is a set of pixels, and as such, is not suitable for many types of classification algorithms. However, if the data is processed to provide higher-level features, such as the presence or absence of certain types of edges and areas that are highly correlated with the presence of human faces, then a much broader set of classification techniques can be applied to this problem.

Unfortunately, in the sense in which it is most commonly used, feature extraction is highly domain-specific. For a particular field, such as image processing, various features and the techniques to extract them have been developed over a period of time, and often these techniques have limited applicability to other fields. Consequently, whenever data mining is applied to a relatively new area, a key task is the development of new features and feature extraction methods.



(a) Two time series.

(b) Noisy time series.

(c) Power spectrum

Figure 2.12. Application of the Fourier transform to identify the underlying frequencies in time series data.

Mapping the Data to a New Space

A totally different view of the data can reveal important and interesting features. Consider, for example, time series data, which often contains periodic patterns. If there is only a single periodic pattern and not much noise, then the pattern is easily detected. If, on the other hand, there are a number of periodic patterns and a significant amount of noise is present, then these patterns are hard to detect. Such patterns can, nonetheless, often be detected by applying a **Fourier transform** to the time series in order to change to a representation in which frequency information is explicit. In the example that follows, it will not be necessary to know the details of the Fourier transform. It is enough to know that, for each time series, the Fourier transform produces a new data object whose attributes are related to frequencies.

Example 2.10 (Fourier Analysis). The time series presented in Figure 2.12(b) is the sum of three other time series, two of which are shown in Figure 2.12(a) and have frequencies of 7 and 17 cycles per second, respectively. The third time series is random noise. Figure 2.12(c) shows the power spectrum that can be computed after applying a Fourier transform to the original time series. (Informally, the power spectrum is proportional to the square of each frequency attribute.) In spite of the noise, there are two peaks that correspond to the periods of the two original, non-noisy time series. Again, the main point is that better features can reveal important aspects of the data. ■

Many other sorts of transformations are also possible. Besides the Fourier transform, the **wavelet transform** has also proven very useful for time series and other types of data.

Feature Construction

Sometimes the features in the original data sets have the necessary information, but it is not in a form suitable for the data mining algorithm. In this situation, one or more new features constructed out of the original features can be more useful than the original features.

Example 2.11 (Density). To illustrate this, consider a data set consisting of information about historical artifacts, which, along with other information, contains the volume and mass of each artifact. For simplicity, assume that these artifacts are made of a small number of materials (wood, clay, bronze, gold) and that we want to classify the artifacts with respect to the material of which they are made. In this case, a density feature constructed from the mass and volume features, i.e., $density = mass/volume$, would most directly yield an accurate classification. Although there have been some attempts to automatically perform feature construction by exploring simple mathematical combinations of existing attributes, the most common approach is to construct features using domain expertise. ■

2.3.6 Discretization and Binarization

Some data mining algorithms, especially certain classification algorithms, require that the data be in the form of categorical attributes. Algorithms that find association patterns require that the data be in the form of binary attributes. Thus, it is often necessary to transform a continuous attribute into a categorical attribute (**discretization**), and both continuous and discrete attributes may need to be transformed into one or more binary attributes (**binarization**). Additionally, if a categorical attribute has a large number of values (categories), or some values occur infrequently, then it may be beneficial for certain data mining tasks to reduce the number of categories by combining some of the values.

As with feature selection, the best discretization and binarization approach is the one that “produces the best result for the data mining algorithm that will be used to analyze the data.” It is typically not practical to apply such a criterion directly. Consequently, discretization or binarization is performed in

Table 2.5. Conversion of a categorical attribute to three binary attributes.

Categorical Value	Integer Value	x_1	x_2	x_3
<i>awful</i>	0	0	0	0
<i>poor</i>	1	0	0	1
<i>OK</i>	2	0	1	0
<i>good</i>	3	0	1	1
<i>great</i>	4	1	0	0

Table 2.6. Conversion of a categorical attribute to five asymmetric binary attributes.

Categorical Value	Integer Value	x_1	x_2	x_3	x_4	x_5
<i>awful</i>	0	1	0	0	0	0
<i>poor</i>	1	0	1	0	0	0
<i>OK</i>	2	0	0	1	0	0
<i>good</i>	3	0	0	0	1	0
<i>great</i>	4	0	0	0	0	1

a way that satisfies a criterion that is thought to have a relationship to good performance for the data mining task being considered.

Binarization

A simple technique to binarize a categorical attribute is the following: If there are m categorical values, then uniquely assign each original value to an integer in the interval $[0, m - 1]$. If the attribute is ordinal, then order must be maintained by the assignment. (Note that even if the attribute is originally represented using integers, this process is necessary if the integers are not in the interval $[0, m - 1]$.) Next, convert each of these m integers to a binary number. Since $n = \lceil \log_2(m) \rceil$ binary digits are required to represent these integers, represent these binary numbers using n binary attributes. To illustrate, a categorical variable with 5 values $\{\text{awful}, \text{poor}, \text{OK}, \text{good}, \text{great}\}$ would require three binary variables x_1 , x_2 , and x_3 . The conversion is shown in Table 2.5.

Such a transformation can cause complications, such as creating unintended relationships among the transformed attributes. For example, in Table 2.5, attributes x_2 and x_3 are correlated because information about the *good* value is encoded using both attributes. Furthermore, association analysis requires asymmetric binary attributes, where only the presence of the attribute (value = 1) is important. For association problems, it is therefore necessary to introduce one binary attribute for each categorical value, as in Table 2.6. If the

number of resulting attributes is too large, then the techniques described below can be used to reduce the number of categorical values before binarization.

Likewise, for association problems, it may be necessary to replace a single binary attribute with two asymmetric binary attributes. Consider a binary attribute that records a person's gender, male or female. For traditional association rule algorithms, this information needs to be transformed into two asymmetric binary attributes, one that is a 1 only when the person is male and one that is a 1 only when the person is female. (For asymmetric binary attributes, the information representation is somewhat inefficient in that two bits of storage are required to represent each bit of information.)

Discretization of Continuous Attributes

Discretization is typically applied to attributes that are used in classification or association analysis. In general, the best discretization depends on the algorithm being used, as well as the other attributes being considered. Typically, however, the discretization of an attribute is considered in isolation.

Transformation of a continuous attribute to a categorical attribute involves two subtasks: deciding how many categories to have and determining how to map the values of the continuous attribute to these categories. In the first step, after the values of the continuous attribute are sorted, they are then divided into n intervals by specifying $n - 1$ **split points**. In the second, rather trivial step, all the values in one interval are mapped to the same categorical value. Therefore, the problem of discretization is one of deciding how many split points to choose and where to place them. The result can be represented either as a set of intervals $\{(x_0, x_1], (x_1, x_2], \dots, (x_{n-1}, x_n)\}$, where x_0 and x_n may be $+\infty$ or $-\infty$, respectively, or equivalently, as a series of inequalities $x_0 < x \leq x_1, \dots, x_{n-1} < x < x_n$.

Unsupervised Discretization A basic distinction between discretization methods for classification is whether class information is used (supervised) or not (unsupervised). If class information is not used, then relatively simple approaches are common. For instance, the **equal width** approach divides the range of the attribute into a user-specified number of intervals each having the same width. Such an approach can be badly affected by outliers, and for that reason, an **equal frequency (equal depth)** approach, which tries to put the same number of objects into each interval, is often preferred. As another example of unsupervised discretization, a clustering method, such as K-means (see Chapter 8), can also be used. Finally, visually inspecting the data can sometimes be an effective approach.

Example 2.12 (Discretization Techniques). This example demonstrates how these approaches work on an actual data set. Figure 2.13(a) shows data points belonging to four different groups, along with two outliers—the large dots on either end. The techniques of the previous paragraph were applied to discretize the x values of these data points into four categorical values. (Points in the data set have a random y component to make it easy to see how many points are in each group.) Visually inspecting the data works quite well, but is not automatic, and thus, we focus on the other three approaches. The split points produced by the techniques equal width, equal frequency, and K-means are shown in Figures 2.13(b), 2.13(c), and 2.13(d), respectively. The split points are represented as dashed lines. If we measure the performance of a discretization technique by the extent to which different objects in different groups are assigned the same categorical value, then K-means performs best, followed by equal frequency, and finally, equal width. ■

Supervised Discretization The discretization methods described above are usually better than no discretization, but keeping the end purpose in mind and using additional information (class labels) often produces better results. This should not be surprising, since an interval constructed with no knowledge of class labels often contains a mixture of class labels. A conceptually simple approach is to place the splits in a way that maximizes the purity of the intervals. In practice, however, such an approach requires potentially arbitrary decisions about the purity of an interval and the minimum size of an interval. To overcome such concerns, some statistically based approaches start with each attribute value as a separate interval and create larger intervals by merging adjacent intervals that are similar according to a statistical test. Entropy-based approaches are one of the most promising approaches to discretization, and a simple approach based on entropy will be presented.

First, it is necessary to define **entropy**. Let k be the number of different class labels, m_i be the number of values in the i^{th} interval of a partition, and m_{ij} be the number of values of class j in interval i . Then the entropy e_i of the i^{th} interval is given by the equation

$$e_i = \sum_{j=1}^k p_{ij} \log_2 p_{ij},$$

where $p_{ij} = m_{ij}/m_i$ is the probability (fraction of values) of class j in the i^{th} interval. The total entropy, e , of the partition is the weighted average of the individual interval entropies, i.e.,

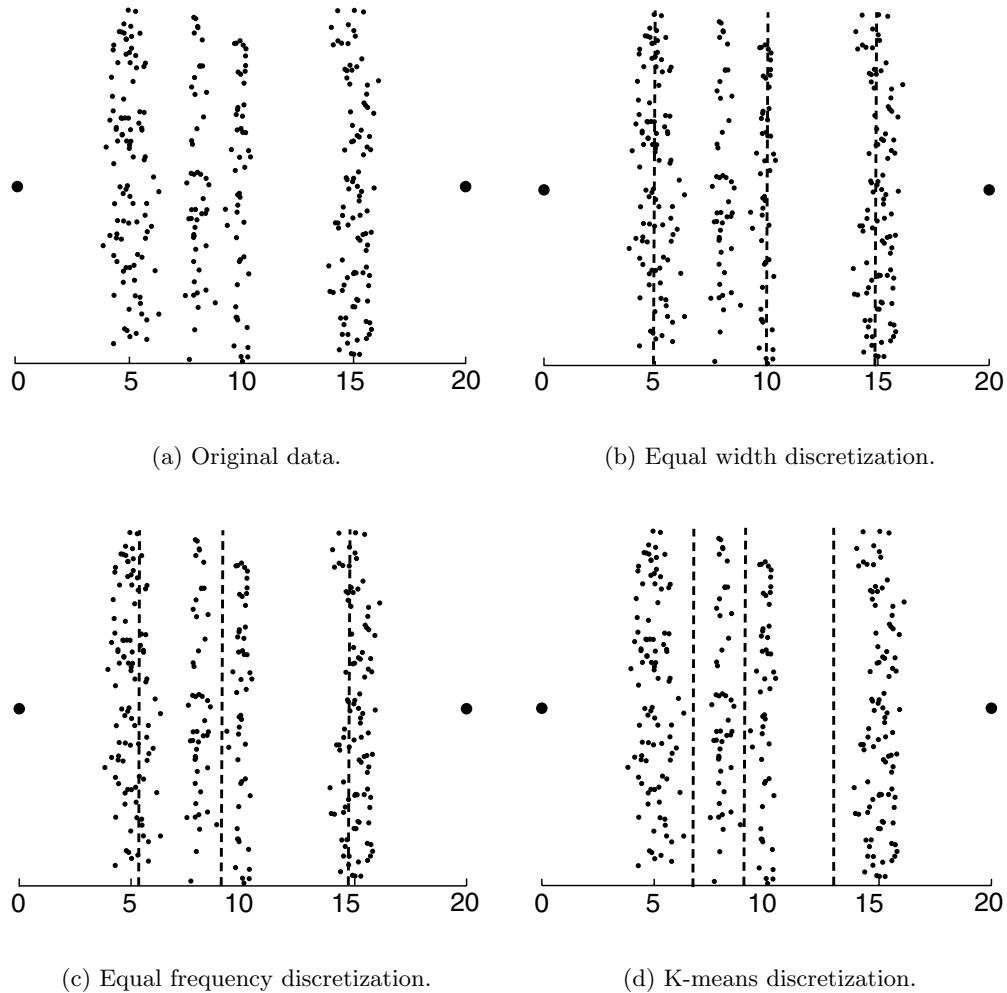


Figure 2.13. Different discretization techniques.

$$e = \sum_{i=1}^n w_i e_i,$$

where m is the number of values, $w_i = m_i/m$ is the fraction of values in the i^{th} interval, and n is the number of intervals. Intuitively, the entropy of an interval is a measure of the purity of an interval. If an interval contains only values of one class (is perfectly pure), then the entropy is 0 and it contributes

nothing to the overall entropy. If the classes of values in an interval occur equally often (the interval is as impure as possible), then the entropy is a maximum.

A simple approach for partitioning a continuous attribute starts by bisecting the initial values so that the resulting two intervals give minimum entropy. This technique only needs to consider each value as a possible split point, because it is assumed that intervals contain ordered sets of values. The splitting process is then repeated with another interval, typically choosing the interval with the worst (highest) entropy, until a user-specified number of intervals is reached, or a stopping criterion is satisfied.

Example 2.13 (Discretization of Two Attributes). This method was used to independently discretize both the x and y attributes of the two-dimensional data shown in Figure 2.14. In the first discretization, shown in Figure 2.14(a), the x and y attributes were both split into three intervals. (The dashed lines indicate the split points.) In the second discretization, shown in Figure 2.14(b), the x and y attributes were both split into five intervals. ■

This simple example illustrates two aspects of discretization. First, in two dimensions, the classes of points are well separated, but in one dimension, this is not so. In general, discretizing each attribute separately often guarantees suboptimal results. Second, five intervals work better than three, but six intervals do not improve the discretization much, at least in terms of entropy. (Entropy values and results for six intervals are not shown.) Consequently, it is desirable to have a stopping criterion that automatically finds the right number of partitions.

Categorical Attributes with Too Many Values

Categorical attributes can sometimes have too many values. If the categorical attribute is an ordinal attribute, then techniques similar to those for continuous attributes can be used to reduce the number of categories. If the categorical attribute is nominal, however, then other approaches are needed. Consider a university that has a large number of departments. Consequently, a *department name* attribute might have dozens of different values. In this situation, we could use our knowledge of the relationships among different departments to combine departments into larger groups, such as *engineering*, *social sciences*, or *biological sciences*. If domain knowledge does not serve as a useful guide or such an approach results in poor classification performance, then it is necessary to use a more empirical approach, such as grouping values

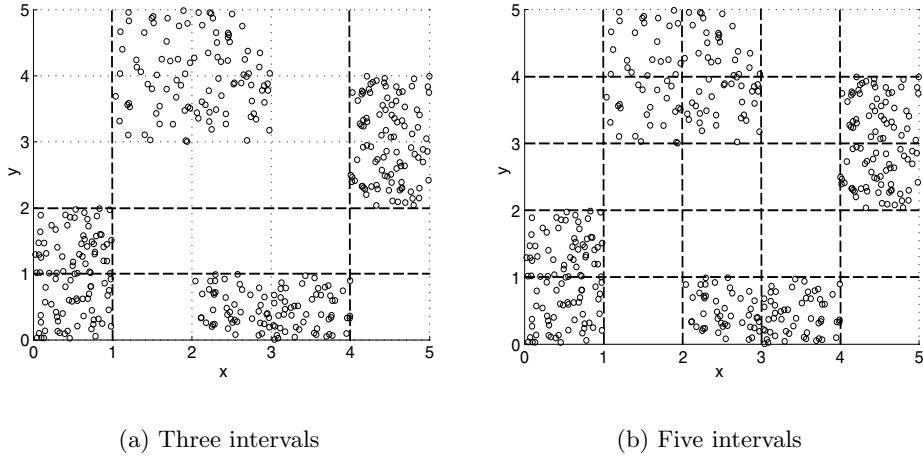


Figure 2.14. Discretizing x and y attributes for four groups (classes) of points.

together only if such a grouping results in improved classification accuracy or achieves some other data mining objective.

2.3.7 Variable Transformation

A **variable transformation** refers to a transformation that is applied to all the values of a variable. (We use the term variable instead of attribute to adhere to common usage, although we will also refer to attribute transformation on occasion.) In other words, for each object, the transformation is applied to the value of the variable for that object. For example, if only the magnitude of a variable is important, then the values of the variable can be transformed by taking the absolute value. In the following section, we discuss two important types of variable transformations: simple functional transformations and normalization.

Simple Functions

For this type of variable transformation, a simple mathematical function is applied to each value individually. If x is a variable, then examples of such transformations include x^k , $\log x$, e^x , \sqrt{x} , $1/x$, $\sin x$, or $|x|$. In statistics, variable transformations, especially *sqrt*, *log*, and $1/x$, are often used to transform data that does not have a Gaussian (normal) distribution into data that does. While this can be important, other reasons often take precedence in data min-

ing. Suppose the variable of interest is the number of data bytes in a session, and the number of bytes ranges from 1 to 1 billion. This is a huge range, and it may be advantageous to compress it by using a \log_{10} transformation. In this case, sessions that transferred 10^8 and 10^9 bytes would be more similar to each other than sessions that transferred 10 and 1000 bytes ($9 - 8 = 1$ versus $3 - 1 = 2$). For some applications, such as network intrusion detection, this may be what is desired, since the first two sessions most likely represent transfers of large files, while the latter two sessions could be two quite distinct types of sessions.

Variable transformations should be applied with caution since they change the nature of the data. While this is what is desired, there can be problems if the nature of the transformation is not fully appreciated. For instance, the transformation $1/x$ reduces the magnitude of values that are 1 or larger, but increases the magnitude of values between 0 and 1. To illustrate, the values $\{1, 2, 3\}$ go to $\{1, \frac{1}{2}, \frac{1}{3}\}$, but the values $\{1, \frac{1}{2}, \frac{1}{3}\}$ go to $\{1, 2, 3\}$. Thus, for all sets of values, the transformation $1/x$ reverses the order. To help clarify the effect of a transformation, it is important to ask questions such as the following: Does the order need to be maintained? Does the transformation apply to all values, especially negative values and 0? What is the effect of the transformation on the values between 0 and 1? Exercise 17 on page 92 explores other aspects of variable transformation.

Normalization or Standardization

Another common type of variable transformation is the **standardization** or **normalization** of a variable. (In the data mining community the terms are often used interchangeably. In statistics, however, the term normalization can be confused with the transformations used for making a variable **normal**, i.e., **Gaussian**.) The goal of standardization or normalization is to make an entire set of values have a particular property. A traditional example is that of “standardizing a variable” in statistics. If \bar{x} is the mean (average) of the attribute values and s_x is their standard deviation, then the transformation $x' = (x - \bar{x})/s_x$ creates a new variable that has a mean of 0 and a standard deviation of 1. If different variables are to be combined in some way, then such a transformation is often necessary to avoid having a variable with large values dominate the results of the calculation. To illustrate, consider comparing people based on two variables: age and income. For any two people, the difference in income will likely be much higher in absolute terms (hundreds or thousands of dollars) than the difference in age (less than 150). If the differences in the range of values of age and income are not taken into account, then

the comparison between people will be dominated by differences in income. In particular, if the similarity or dissimilarity of two people is calculated using the similarity or dissimilarity measures defined later in this chapter, then in many cases, such as that of Euclidean distance, the income values will dominate the calculation.

The mean and standard deviation are strongly affected by outliers, so the above transformation is often modified. First, the mean is replaced by the **median**, i.e., the middle value. Second, the standard deviation is replaced by the **absolute standard deviation**. Specifically, if x is a variable, then the absolute standard deviation of x is given by $\sigma_A = \sum_{i=1}^m |x_i - \mu|$, where x_i is the i^{th} value of the variable, m is the number of objects, and μ is either the mean or median. Other approaches for computing estimates of the location (center) and spread of a set of values in the presence of outliers are described in Sections 3.2.3 and 3.2.4, respectively. These measures can also be used to define a standardization transformation.

2.4 Measures of Similarity and Dissimilarity

Similarity and dissimilarity are important because they are used by a number of data mining techniques, such as clustering, nearest neighbor classification, and anomaly detection. In many cases, the initial data set is not needed once these similarities or dissimilarities have been computed. Such approaches can be viewed as transforming the data to a similarity (dissimilarity) space and then performing the analysis.

We begin with a discussion of the basics: high-level definitions of similarity and dissimilarity, and a discussion of how they are related. For convenience, the term **proximity** is used to refer to either similarity or dissimilarity. Since the proximity between two objects is a function of the proximity between the corresponding attributes of the two objects, we first describe how to measure the proximity between objects having only one simple attribute, and then consider proximity measures for objects with multiple attributes. This includes measures such as correlation and Euclidean distance, which are useful for dense data such as time series or two-dimensional points, as well as the Jaccard and cosine similarity measures, which are useful for sparse data like documents. Next, we consider several important issues concerning proximity measures. The section concludes with a brief discussion of how to select the right proximity measure.

2.4.1 Basics

Definitions

Informally, the **similarity** between two objects is a numerical measure of the degree to which the two objects are alike. Consequently, similarities are *higher* for pairs of objects that are more alike. Similarities are usually non-negative and are often between 0 (no similarity) and 1 (complete similarity).

The **dissimilarity** between two objects is a numerical measure of the degree to which the two objects are different. Dissimilarities are *lower* for more similar pairs of objects. Frequently, the term **distance** is used as a synonym for dissimilarity, although, as we shall see, distance is often used to refer to a special class of dissimilarities. Dissimilarities sometimes fall in the interval $[0, 1]$, but it is also common for them to range from 0 to ∞ .

Transformations

Transformations are often applied to convert a similarity to a dissimilarity, or vice versa, or to transform a proximity measure to fall within a particular range, such as $[0,1]$. For instance, we may have similarities that range from 1 to 10, but the particular algorithm or software package that we want to use may be designed to only work with dissimilarities, or it may only work with similarities in the interval $[0,1]$. We discuss these issues here because we will employ such transformations later in our discussion of proximity. In addition, these issues are relatively independent of the details of specific proximity measures.

Frequently, proximity measures, especially similarities, are defined or transformed to have values in the interval $[0,1]$. Informally, the motivation for this is to use a scale in which a proximity value indicates the fraction of similarity (or dissimilarity) between two objects. Such a transformation is often relatively straightforward. For example, if the similarities between objects range from 1 (not at all similar) to 10 (completely similar), we can make them fall within the range $[0, 1]$ by using the transformation $s' = (s - 1)/9$, where s and s' are the original and new similarity values, respectively. In the more general case, the transformation of similarities to the interval $[0, 1]$ is given by the expression $s' = (s - \text{min_}s)/(\text{max_}s - \text{min_}s)$, where $\text{max_}s$ and $\text{min_}s$ are the maximum and minimum similarity values, respectively. Likewise, dissimilarity measures with a finite range can be mapped to the interval $[0,1]$ by using the formula $d' = (d - \text{min_}d)/(\text{max_}d - \text{min_}d)$.

There can be various complications in mapping proximity measures to the interval $[0, 1]$, however. If, for example, the proximity measure originally takes

values in the interval $[0, \infty]$, then a non-linear transformation is needed and values will not have the same relationship to one another on the new scale. Consider the transformation $d' = d/(1 + d)$ for a dissimilarity measure that ranges from 0 to ∞ . The dissimilarities 0, 0.5, 2, 10, 100, and 1000 will be transformed into the new dissimilarities 0, 0.33, 0.67, 0.90, 0.99, and 0.999, respectively. Larger values on the original dissimilarity scale are compressed into the range of values near 1, but whether or not this is desirable depends on the application. Another complication is that the meaning of the proximity measure may be changed. For example, correlation, which is discussed later, is a measure of similarity that takes values in the interval $[-1, 1]$. Mapping these values to the interval $[0, 1]$ by taking the absolute value loses information about the sign, which can be important in some applications. See Exercise 22 on page 94.

Transforming similarities to dissimilarities and vice versa is also relatively straightforward, although we again face the issues of preserving meaning and changing a linear scale into a non-linear scale. If the similarity (or dissimilarity) falls in the interval $[0, 1]$, then the dissimilarity can be defined as $d = 1 - s$ ($s = 1 - d$). Another simple approach is to define similarity as the negative of the dissimilarity (or vice versa). To illustrate, the dissimilarities 0, 1, 10, and 100 can be transformed into the similarities 0, -1, -10, and -100, respectively.

The similarities resulting from the negation transformation are not restricted to the range $[0, 1]$, but if that is desired, then transformations such as $s = \frac{1}{d+1}$, $s = e^{-d}$, or $s = 1 - \frac{d-\min_d}{\max_d-\min_d}$ can be used. For the transformation $s = \frac{1}{d+1}$, the dissimilarities 0, 1, 10, 100 are transformed into 1, 0.5, 0.09, 0.01, respectively. For $s = e^{-d}$, they become 1.00, 0.37, 0.00, 0.00, respectively, while for $s = 1 - \frac{d-\min_d}{\max_d-\min_d}$ they become 1.00, 0.99, 0.00, 0.00, respectively. In this discussion, we have focused on converting dissimilarities to similarities. Conversion in the opposite direction is considered in Exercise 23 on page 94.

In general, any monotonic decreasing function can be used to convert dissimilarities to similarities, or vice versa. Of course, other factors also must be considered when transforming similarities to dissimilarities, or vice versa, or when transforming the values of a proximity measure to a new scale. We have mentioned issues related to preserving meaning, distortion of scale, and requirements of data analysis tools, but this list is certainly not exhaustive.

2.4.2 Similarity and Dissimilarity between Simple Attributes

The proximity of objects with a number of attributes is typically defined by combining the proximities of individual attributes, and thus, we first discuss

proximity between objects having a single attribute. Consider objects described by one nominal attribute. What would it mean for two such objects to be similar? Since nominal attributes only convey information about the distinctness of objects, all we can say is that two objects either have the same value or they do not. Hence, in this case similarity is traditionally defined as 1 if attribute values match, and as 0 otherwise. A dissimilarity would be defined in the opposite way: 0 if the attribute values match, and 1 if they do not.

For objects with a single ordinal attribute, the situation is more complicated because information about order should be taken into account. Consider an attribute that measures the quality of a product, e.g., a candy bar, on the scale $\{\text{poor}, \text{fair}, \text{OK}, \text{good}, \text{wonderful}\}$. It would seem reasonable that a product, P1, which is rated *wonderful*, would be closer to a product P2, which is rated *good*, than it would be to a product P3, which is rated *OK*. To make this observation quantitative, the values of the ordinal attribute are often mapped to successive integers, beginning at 0 or 1, e.g., $\{\text{poor}=0, \text{fair}=1, \text{OK}=2, \text{good}=3, \text{wonderful}=4\}$. Then, $d(P1, P2) = 3 - 2 = 1$ or, if we want the dissimilarity to fall between 0 and 1, $d(P1, P2) = \frac{3-2}{4} = 0.25$. A similarity for ordinal attributes can then be defined as $s = 1 - d$.

This definition of similarity (dissimilarity) for an ordinal attribute should make the reader a bit uneasy since this assumes equal intervals, and this is not so. Otherwise, we would have an interval or ratio attribute. Is the difference between the values *fair* and *good* really the same as that between the values *OK* and *wonderful*? Probably not, but in practice, our options are limited, and in the absence of more information, this is the standard approach for defining proximity between ordinal attributes.

For interval or ratio attributes, the natural measure of dissimilarity between two objects is the absolute difference of their values. For example, we might compare our current weight and our weight a year ago by saying “I am ten pounds heavier.” In cases such as these, the dissimilarities typically range from 0 to ∞ , rather than from 0 to 1. The similarity of interval or ratio attributes is typically expressed by transforming a similarity into a dissimilarity, as previously described.

Table 2.7 summarizes this discussion. In this table, x and y are two objects that have one attribute of the indicated type. Also, $d(x, y)$ and $s(x, y)$ are the dissimilarity and similarity between x and y , respectively. Other approaches are possible; these are the most common ones.

The following two sections consider more complicated measures of proximity between objects that involve multiple attributes: (1) dissimilarities between data objects and (2) similarities between data objects. This division

Table 2.7. Similarity and dissimilarity for simple attributes

Attribute Type	Dissimilarity	Similarity
Nominal	$d = \begin{cases} 0 & \text{if } x = y \\ 1 & \text{if } x \neq y \end{cases}$	$s = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{if } x \neq y \end{cases}$
Ordinal	$d = x - y /(n - 1)$ (values mapped to integers 0 to $n - 1$, where n is the number of values)	$s = 1 - d$
Interval or Ratio	$d = x - y $	$s = -d, s = \frac{1}{1+d}, s = e^{-d}, s = 1 - \frac{d - \min_d}{\max_d - \min_d}$

allows us to more naturally display the underlying motivations for employing various proximity measures. We emphasize, however, that similarities can be transformed into dissimilarities and vice versa using the approaches described earlier.

2.4.3 Dissimilarities between Data Objects

In this section, we discuss various kinds of dissimilarities. We begin with a discussion of distances, which are dissimilarities with certain properties, and then provide examples of more general kinds of dissimilarities.

Distances

We first present some examples, and then offer a more formal description of distances in terms of the properties common to all distances. The **Euclidean distance**, d , between two points, \mathbf{x} and \mathbf{y} , in one-, two-, three-, or higher-dimensional space, is given by the following familiar formula:

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{k=1}^n (x_k - y_k)^2}, \quad (2.1)$$

where n is the number of dimensions and x_k and y_k are, respectively, the k^{th} attributes (components) of x and y . We illustrate this formula with Figure 2.15 and Tables 2.8 and 2.9, which show a set of points, the x and y coordinates of these points, and the **distance matrix** containing the pairwise distances of these points.

The Euclidean distance measure given in Equation 2.1 is generalized by the **Minkowski** distance metric shown in Equation 2.2,

$$d(\mathbf{x}, \mathbf{y}) = \left(\sum_{k=1}^n |x_k - y_k|^r \right)^{1/r}, \quad (2.2)$$

where r is a parameter. The following are the three most common examples of Minkowski distances.

- $r = 1$. City block (Manhattan, taxicab, L_1 norm) distance. A common example is the **Hamming distance**, which is the number of bits that are different between two objects that have only binary attributes, i.e., between two binary vectors.
- $r = 2$. Euclidean distance (L_2 norm).
- $r = \infty$. Supremum (L_{\max} or L_∞ norm) distance. This is the maximum difference between any attribute of the objects. More formally, the L_∞ distance is defined by Equation 2.3

$$d(\mathbf{x}, \mathbf{y}) = \lim_{r \rightarrow \infty} \left(\sum_{k=1}^n |x_k - y_k|^r \right)^{1/r}. \quad (2.3)$$

The r parameter should not be confused with the number of dimensions (attributes) n . The Euclidean, Manhattan, and supremum distances are defined for all values of n : 1, 2, 3, ..., and specify different ways of combining the differences in each dimension (attribute) into an overall distance.

Tables 2.10 and 2.11, respectively, give the proximity matrices for the L_1 and L_∞ distances using data from Table 2.8. Notice that all these distance matrices are symmetric; i.e., the ij^{th} entry is the same as the ji^{th} entry. In Table 2.9, for instance, the fourth row of the first column and the fourth column of the first row both contain the value 5.1.

Distances, such as the Euclidean distance, have some well-known properties. If $d(\mathbf{x}, \mathbf{y})$ is the distance between two points, \mathbf{x} and \mathbf{y} , then the following properties hold.

1. Positivity

- (a) $d(\mathbf{x}, \mathbf{x}) \geq 0$ for all \mathbf{x} and \mathbf{y} ,
- (b) $d(\mathbf{x}, \mathbf{y}) = 0$ only if $\mathbf{x} = \mathbf{y}$.

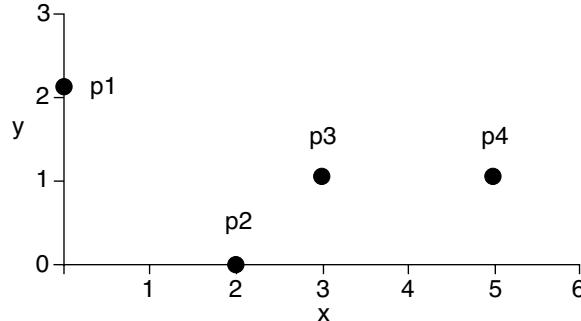


Figure 2.15. Four two-dimensional points.

Table 2.8. x and y coordinates of four points.

point	x coordinate	y coordinate
p1	0	2
p2	2	0
p3	3	1
p4	5	1

Table 2.9. Euclidean distance matrix for Table 2.8.

	p1	p2	p3	p4
p1	0.0	2.8	3.2	5.1
p2	2.8	0.0	1.4	3.2
p3	3.2	1.4	0.0	2.0
p4	5.1	3.2	2.0	0.0

Table 2.10. L_1 distance matrix for Table 2.8.

L_1	p1	p2	p3	p4
p1	0.0	4.0	4.0	6.0
p2	4.0	0.0	2.0	4.0
p3	4.0	2.0	0.0	2.0
p4	6.0	4.0	2.0	0.0

Table 2.11. L_∞ distance matrix for Table 2.8.

L_∞	p1	p2	p3	p4
p1	0.0	2.0	3.0	5.0
p2	2.0	0.0	1.0	3.0
p3	3.0	1.0	0.0	2.0
p4	5.0	3.0	2.0	0.0

2. Symmetry

$$d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x}) \text{ for all } \mathbf{x} \text{ and } \mathbf{y}.$$

3. Triangle Inequality

$$d(\mathbf{x}, \mathbf{z}) \leq d(\mathbf{x}, \mathbf{y}) + d(\mathbf{y}, \mathbf{z}) \text{ for all points } \mathbf{x}, \mathbf{y}, \text{ and } \mathbf{z}.$$

Measures that satisfy all three properties are known as **metrics**. Some people only use the term distance for dissimilarity measures that satisfy these properties, but that practice is often violated. The three properties described here are useful, as well as mathematically pleasing. Also, if the triangle inequality holds, then this property can be used to increase the efficiency of techniques (including clustering) that depend on distances possessing this property. (See Exercise 25.) Nonetheless, many dissimilarities do not satisfy one or more of the metric properties. We give two examples of such measures.

Example 2.14 (Non-metric Dissimilarities: Set Differences). This example is based on the notion of the difference of two sets, as defined in set theory. Given two sets A and B , $A - B$ is the set of elements of A that are not in B . For example, if $A = \{1, 2, 3, 4\}$ and $B = \{2, 3, 4\}$, then $A - B = \{1\}$ and $B - A = \emptyset$, the empty set. We can define the distance d between two sets A and B as $d(A, B) = \text{size}(A - B)$, where size is a function returning the number of elements in a set. This distance measure, which is an integer value greater than or equal to 0, does not satisfy the second part of the positivity property, the symmetry property, or the triangle inequality. However, these properties can be made to hold if the dissimilarity measure is modified as follows: $d(A, B) = \text{size}(A - B) + \text{size}(B - A)$. See Exercise 21 on page 94. ■

Example 2.15 (Non-metric Dissimilarities: Time). This example gives a more everyday example of a dissimilarity measure that is not a metric, but that is still useful. Define a measure of the distance between times of the day as follows:

$$d(t_1, t_2) = \begin{cases} t_2 - t_1 & \text{if } t_1 \leq t_2 \\ 24 + (t_2 - t_1) & \text{if } t_1 \geq t_2 \end{cases}. \quad (2.4)$$

To illustrate, $d(1\text{PM}, 2\text{PM}) = 1$ hour, while $d(2\text{PM}, 1\text{PM}) = 23$ hours. Such a definition would make sense, for example, when answering the question: “If an event occurs at 1PM every day, and it is now 2PM, how long do I have to wait for that event to occur again?” ■

2.4.4 Similarities between Data Objects

For similarities, the triangle inequality (or the analogous property) typically does not hold, but symmetry and positivity typically do. To be explicit, if $s(\mathbf{x}, \mathbf{y})$ is the similarity between points \mathbf{x} and \mathbf{y} , then the typical properties of similarities are the following:

1. $s(\mathbf{x}, \mathbf{y}) = 1$ only if $\mathbf{x} = \mathbf{y}$. ($0 \leq s \leq 1$)
2. $s(\mathbf{x}, \mathbf{y}) = s(\mathbf{y}, \mathbf{x})$ for all \mathbf{x} and \mathbf{y} . (Symmetry)

There is no general analog of the triangle inequality for similarity measures. It is sometimes possible, however, to show that a similarity measure can easily be converted to a metric distance. The cosine and Jaccard similarity measures, which are discussed shortly, are two examples. Also, for specific similarity measures, it is possible to derive mathematical bounds on the similarity between two objects that are similar in spirit to the triangle inequality.

Example 2.16 (A Non-symmetric Similarity Measure). Consider an experiment in which people are asked to classify a small set of characters as they flash on a screen. The **confusion matrix** for this experiment records how often each character is classified as itself, and how often each is classified as another character. For instance, suppose that “0” appeared 200 times and was classified as a “0” 160 times, but as an “o” 40 times. Likewise, suppose that ‘o’ appeared 200 times and was classified as an “o” 170 times, but as “0” only 30 times. If we take these counts as a measure of the similarity between two characters, then we have a similarity measure, but one that is not symmetric. In such situations, the similarity measure is often made symmetric by setting $s'(\mathbf{x}, \mathbf{y}) = s'(\mathbf{y}, \mathbf{x}) = (s(\mathbf{x}, \mathbf{y}) + s(\mathbf{y}, \mathbf{x}))/2$, where s' indicates the new similarity measure. ■

2.4.5 Examples of Proximity Measures

This section provides specific examples of some similarity and dissimilarity measures.

Similarity Measures for Binary Data

Similarity measures between objects that contain only binary attributes are called **similarity coefficients**, and typically have values between 0 and 1. A value of 1 indicates that the two objects are completely similar, while a value of 0 indicates that the objects are not at all similar. There are many rationales for why one coefficient is better than another in specific instances.

Let \mathbf{x} and \mathbf{y} be two objects that consist of n binary attributes. The comparison of two such objects, i.e., two binary vectors, leads to the following four quantities (frequencies):

- f_{00} = the number of attributes where \mathbf{x} is 0 and \mathbf{y} is 0
- f_{01} = the number of attributes where \mathbf{x} is 0 and \mathbf{y} is 1
- f_{10} = the number of attributes where \mathbf{x} is 1 and \mathbf{y} is 0
- f_{11} = the number of attributes where \mathbf{x} is 1 and \mathbf{y} is 1

Simple Matching Coefficient One commonly used similarity coefficient is the **simple matching coefficient (SMC)**, which is defined as

$$SMC = \frac{\text{number of matching attribute values}}{\text{number of attributes}} = \frac{f_{11} + f_{00}}{f_{01} + f_{10} + f_{11} + f_{00}}. \quad (2.5)$$

This measure counts both presences and absences equally. Consequently, the *SMC* could be used to find students who had answered questions similarly on a test that consisted only of true/false questions.

Jaccard Coefficient Suppose that \mathbf{x} and \mathbf{y} are data objects that represent two rows (two transactions) of a transaction matrix (see Section 2.1.2). If each asymmetric binary attribute corresponds to an item in a store, then a 1 indicates that the item was purchased, while a 0 indicates that the product was not purchased. Since the number of products not purchased by any customer far outnumbers the number of products that were purchased, a similarity measure such as *SMC* would say that all transactions are very similar. As a result, the Jaccard coefficient is frequently used to handle objects consisting of asymmetric binary attributes. The **Jaccard coefficient**, which is often symbolized by J , is given by the following equation:

$$J = \frac{\text{number of matching presences}}{\text{number of attributes not involved in 00 matches}} = \frac{f_{11}}{f_{01} + f_{10} + f_{11}}. \quad (2.6)$$

Example 2.17 (The SMC and Jaccard Similarity Coefficients). To illustrate the difference between these two similarity measures, we calculate *SMC* and J for the following two binary vectors.

$$\begin{aligned}\mathbf{x} &= (1, 0, 0, 0, 0, 0, 0, 0, 0) \\ \mathbf{y} &= (0, 0, 0, 0, 0, 0, 1, 0, 0)\end{aligned}$$

- $f_{01} = 2$ the number of attributes where \mathbf{x} was 0 and \mathbf{y} was 1
- $f_{10} = 1$ the number of attributes where \mathbf{x} was 1 and \mathbf{y} was 0
- $f_{00} = 7$ the number of attributes where \mathbf{x} was 0 and \mathbf{y} was 0
- $f_{11} = 0$ the number of attributes where \mathbf{x} was 1 and \mathbf{y} was 1

$$SMC = \frac{f_{11} + f_{00}}{f_{01} + f_{10} + f_{11} + f_{00}} = \frac{0+7}{2+1+0+7} = 0.7$$

$$J = \frac{f_{11}}{f_{01} + f_{10} + f_{11}} = \frac{0}{2+1+0} = 0$$

■

Cosine Similarity

Documents are often represented as vectors, where each attribute represents the frequency with which a particular term (word) occurs in the document. It is more complicated than this, of course, since certain common words are ig-

nored and various processing techniques are used to account for different forms of the same word, differing document lengths, and different word frequencies.

Even though documents have thousands or tens of thousands of attributes (terms), each document is sparse since it has relatively few non-zero attributes. (The normalizations used for documents do not create a non-zero entry where there was a zero entry; i.e., they preserve sparsity.) Thus, as with transaction data, similarity should not depend on the number of shared 0 values since any two documents are likely to “not contain” many of the same words, and therefore, if 0–0 matches are counted, most documents will be highly similar to most other documents. Therefore, a similarity measure for documents needs to ignore 0–0 matches like the Jaccard measure, but also must be able to handle non-binary vectors. The **cosine similarity**, defined next, is one of the most common measure of document similarity. If \mathbf{x} and \mathbf{y} are two document vectors, then

$$\cos(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|}, \quad (2.7)$$

where \cdot indicates the vector dot product, $\mathbf{x} \cdot \mathbf{y} = \sum_{k=1}^n x_k y_k$, and $\|\mathbf{x}\|$ is the length of vector \mathbf{x} , $\|\mathbf{x}\| = \sqrt{\sum_{k=1}^n x_k^2} = \sqrt{\mathbf{x} \cdot \mathbf{x}}$.

Example 2.18 (Cosine Similarity of Two Document Vectors). This example calculates the cosine similarity for the following two data objects, which might represent document vectors:

$$\mathbf{x} = (3, 2, 0, 5, 0, 0, 0, 2, 0, 0)$$

$$\mathbf{y} = (1, 0, 0, 0, 0, 0, 0, 1, 0, 2)$$

$$\mathbf{x} \cdot \mathbf{y} = 3 * 1 + 2 * 0 + 0 * 0 + 5 * 0 + 0 * 0 + 0 * 0 + 0 * 0 + 2 * 1 + 0 * 0 + 0 * 2 = 5$$

$$\|\mathbf{x}\| = \sqrt{3 * 3 + 2 * 2 + 0 * 0 + 5 * 5 + 0 * 0 + 0 * 0 + 0 * 0 + 2 * 2 + 0 * 0 + 0 * 0} = 6.48$$

$$\|\mathbf{y}\| = \sqrt{1 * 1 + 0 * 0 + 0 * 0 + 0 * 0 + 0 * 0 + 0 * 0 + 0 * 0 + 1 * 1 + 0 * 0 + 2 * 2} = 2.24$$

$$\cos(\mathbf{x}, \mathbf{y}) = 0.31$$

■

As indicated by Figure 2.16, cosine similarity really is a measure of the (cosine of the) angle between \mathbf{x} and \mathbf{y} . Thus, if the cosine similarity is 1, the angle between \mathbf{x} and \mathbf{y} is 0° , and \mathbf{x} and \mathbf{y} are the same except for magnitude (length). If the cosine similarity is 0, then the angle between \mathbf{x} and \mathbf{y} is 90° , and they do not share any terms (words).

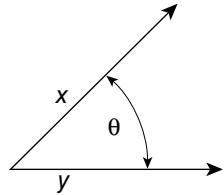


Figure 2.16. Geometric illustration of the cosine measure.

Equation 2.7 can be written as Equation 2.8.

$$\cos(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x}}{\|\mathbf{x}\|} \cdot \frac{\mathbf{y}}{\|\mathbf{y}\|} = \mathbf{x}' \cdot \mathbf{y}', \quad (2.8)$$

where $\mathbf{x}' = \mathbf{x}/\|\mathbf{x}\|$ and $\mathbf{y}' = \mathbf{y}/\|\mathbf{y}\|$. Dividing \mathbf{x} and \mathbf{y} by their lengths normalizes them to have a length of 1. This means that cosine similarity does not take the *magnitude* of the two data objects into account when computing similarity. (Euclidean distance might be a better choice when magnitude is important.) For vectors with a length of 1, the cosine measure can be calculated by taking a simple dot product. Consequently, when many cosine similarities between objects are being computed, normalizing the objects to have unit length can reduce the time required.

Extended Jaccard Coefficient (Tanimoto Coefficient)

The extended Jaccard coefficient can be used for document data and that reduces to the Jaccard coefficient in the case of binary attributes. The extended Jaccard coefficient is also known as the Tanimoto coefficient. (However, there is another coefficient that is also known as the Tanimoto coefficient.) This coefficient, which we shall represent as EJ , is defined by the following equation:

$$EJ(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\|^2 + \|\mathbf{y}\|^2 - \mathbf{x} \cdot \mathbf{y}}. \quad (2.9)$$

Correlation

The correlation between two data objects that have binary or continuous variables is a measure of the linear relationship between the attributes of the objects. (The calculation of correlation between attributes, which is more common, can be defined similarly.) More precisely, **Pearson's correlation**

coefficient between two data objects, \mathbf{x} and \mathbf{y} , is defined by the following equation:

$$\text{corr}(\mathbf{x}, \mathbf{y}) = \frac{\text{covariance}(\mathbf{x}, \mathbf{y})}{\text{standard_deviation}(\mathbf{x}) * \text{standard_deviation}(\mathbf{y})} = \frac{s_{xy}}{s_x s_y}, \quad (2.10)$$

where we are using the following standard statistical notation and definitions:

$$\text{covariance}(\mathbf{x}, \mathbf{y}) = s_{xy} = \frac{1}{n-1} \sum_{k=1}^n (x_k - \bar{x})(y_k - \bar{y}) \quad (2.11)$$

$$\begin{aligned} \text{standard_deviation}(\mathbf{x}) &= s_x = \sqrt{\frac{1}{n-1} \sum_{k=1}^n (x_k - \bar{x})^2} \\ \text{standard_deviation}(\mathbf{y}) &= s_y = \sqrt{\frac{1}{n-1} \sum_{k=1}^n (y_k - \bar{y})^2} \end{aligned}$$

$$\begin{aligned} \bar{x} &= \frac{1}{n} \sum_{k=1}^n x_k \text{ is the mean of } \mathbf{x} \\ \bar{y} &= \frac{1}{n} \sum_{k=1}^n y_k \text{ is the mean of } \mathbf{y} \end{aligned}$$

Example 2.19 (Perfect Correlation). Correlation is always in the range -1 to 1 . A correlation of 1 (-1) means that \mathbf{x} and \mathbf{y} have a perfect positive (negative) linear relationship; that is, $x_k = ay_k + b$, where a and b are constants. The following two sets of values for \mathbf{x} and \mathbf{y} indicate cases where the correlation is -1 and $+1$, respectively. In the first case, the means of \mathbf{x} and \mathbf{y} were chosen to be 0 , for simplicity.

$$\begin{aligned} \mathbf{x} &= (-3, 6, 0, 3, -6) \\ \mathbf{y} &= (1, -2, 0, -1, 2) \end{aligned}$$

$$\begin{aligned} \mathbf{x} &= (3, 6, 0, 3, 6) \\ \mathbf{y} &= (1, 2, 0, 1, 2) \end{aligned}$$

■

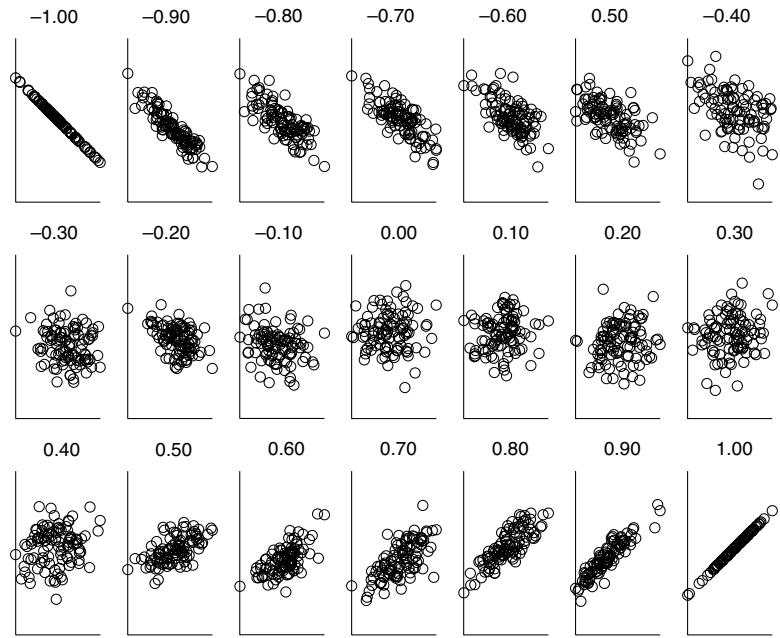


Figure 2.17. Scatter plots illustrating correlations from -1 to 1 .

Example 2.20 (Non-linear Relationships). If the correlation is 0 , then there is no linear relationship between the attributes of the two data objects. However, non-linear relationships may still exist. In the following example, $x_k = y_k^2$, but their correlation is 0 .

$$\begin{aligned} \mathbf{x} &= (-3, -2, -1, 0, 1, 2, 3) \\ \mathbf{y} &= (9, 4, 1, 0, 1, 4, 9) \end{aligned}$$

Example 2.21 (Visualizing Correlation). It is also easy to judge the correlation between two data objects \mathbf{x} and \mathbf{y} by plotting pairs of corresponding attribute values. Figure 2.17 shows a number of these plots when \mathbf{x} and \mathbf{y} have 30 attributes and the values of these attributes are randomly generated (with a normal distribution) so that the correlation of \mathbf{x} and \mathbf{y} ranges from -1 to 1 . Each circle in a plot represents one of the 30 attributes; its x coordinate is the value of one of the attributes for \mathbf{x} , while its y coordinate is the value of the same attribute for \mathbf{y} .

If we transform \mathbf{x} and \mathbf{y} by subtracting off their means and then normalizing them so that their lengths are 1 , then their correlation can be calculated by

taking the dot product. Notice that this is not the same as the standardization used in other contexts, where we make the transformations, $x'_k = (x_k - \bar{x})/s_x$ and $y'_k = (y_k - \bar{y})/s_y$.

Bregman Divergence* This section provides a brief description of Bregman divergences, which are a family of proximity functions that share some common properties. As a result, it is possible to construct general data mining algorithms, such as clustering algorithms, that work with any Bregman divergence. A concrete example is the K-means clustering algorithm (Section 8.2). Note that this section requires knowledge of vector calculus.

Bregman divergences are loss or distortion functions. To understand the idea of a loss function, consider the following. Let \mathbf{x} and \mathbf{y} be two points, where \mathbf{y} is regarded as the original point and \mathbf{x} is some distortion or approximation of it. For example, \mathbf{x} may be a point that was generated, for example, by adding random noise to \mathbf{y} . The goal is to measure the resulting distortion or loss that results if \mathbf{y} is approximated by \mathbf{x} . Of course, the more similar \mathbf{x} and \mathbf{y} are, the smaller the loss or distortion. Thus, Bregman divergences can be used as dissimilarity functions.

More formally, we have the following definition.

Definition 2.6 (Bregman Divergence). Given a strictly convex function ϕ (with a few modest restrictions that are generally satisfied), the Bregman divergence (loss function) $D(\mathbf{x}, \mathbf{y})$ generated by that function is given by the following equation:

$$D(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x}) - \phi(\mathbf{y}) - \langle \nabla \phi(\mathbf{y}), (\mathbf{x} - \mathbf{y}) \rangle \quad (2.12)$$

where $\nabla \phi(\mathbf{y})$ is the gradient of ϕ evaluated at \mathbf{y} , $\mathbf{x} - \mathbf{y}$, is the vector difference between \mathbf{x} and \mathbf{y} , and $\langle \nabla \phi(\mathbf{y}), (\mathbf{x} - \mathbf{y}) \rangle$ is the inner product between $\nabla \phi(\mathbf{y})$ and $(\mathbf{x} - \mathbf{y})$. For points in Euclidean space, the inner product is just the dot product.

$D(\mathbf{x}, \mathbf{y})$ can be written as $D(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x}) - L(\mathbf{x})$, where $L(\mathbf{x}) = \phi(\mathbf{y}) + \langle \nabla \phi(\mathbf{y}), (\mathbf{x} - \mathbf{y}) \rangle$ and represents the equation of a plane that is tangent to the function ϕ at \mathbf{y} . Using calculus terminology, $L(\mathbf{x})$ is the linearization of ϕ around the point \mathbf{y} and the Bregman divergence is just the difference between a function and a linear approximation to that function. Different Bregman divergences are obtained by using different choices for ϕ .

Example 2.22. We provide a concrete example using squared Euclidean distance, but restrict ourselves to one dimension to simplify the mathematics. Let

x and y be real numbers and $\phi(t)$ be the real valued function, $\phi(t) = t^2$. In that case, the gradient reduces to the derivative and the dot product reduces to multiplication. Specifically, Equation 2.12 becomes Equation 2.13.

$$D(x, y) = x^2 - y^2 - 2y(x - y) = (x - y)^2 \quad (2.13)$$

The graph for this example, with $y = 1$, is shown in Figure 2.18. The Bregman divergence is shown for two values of x : $x = 2$ and $x = 3$. ■

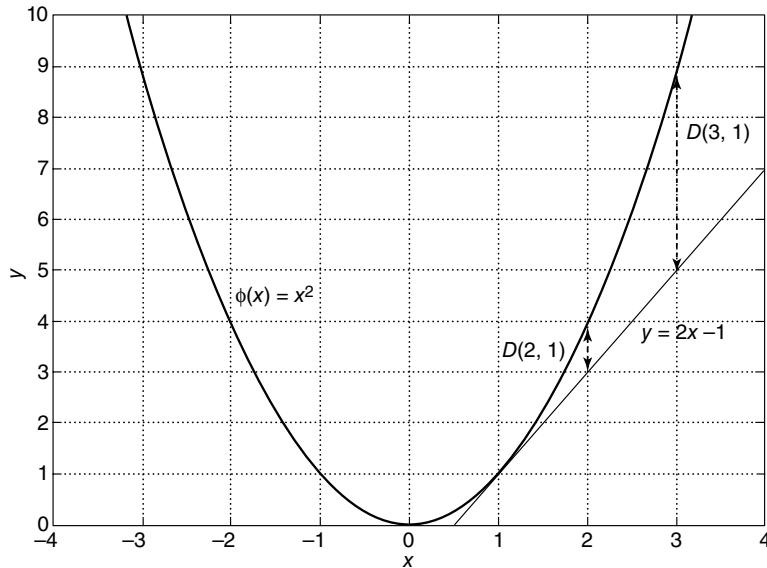


Figure 2.18. Illustration of Bregman divergence.

2.4.6 Issues in Proximity Calculation

This section discusses several important issues related to proximity measures: (1) how to handle the case in which attributes have different scales and/or are correlated, (2) how to calculate proximity between objects that are composed of different types of attributes, e.g., quantitative and qualitative, (3) and how to handle proximity calculation when attributes have different weights; i.e., when not all attributes contribute equally to the proximity of objects.

Standardization and Correlation for Distance Measures

An important issue with distance measures is how to handle the situation when attributes do not have the same range of values. (This situation is often described by saying that “the variables have different scales.”) Earlier, Euclidean distance was used to measure the distance between people based on two attributes: age and income. Unless these two attributes are standardized, the distance between two people will be dominated by income.

A related issue is how to compute distance when there is correlation between some of the attributes, perhaps in addition to differences in the ranges of values. A generalization of Euclidean distance, the **Mahalanobis distance**, is useful when attributes are correlated, have different ranges of values (different variances), and the distribution of the data is approximately Gaussian (normal). Specifically, the Mahalanobis distance between two objects (vectors) \mathbf{x} and \mathbf{y} is defined as

$$\text{mahalanobis}(\mathbf{x}, \mathbf{y}) = (\mathbf{x} - \mathbf{y})\Sigma^{-1}(\mathbf{x} - \mathbf{y})^T, \quad (2.14)$$

where Σ^{-1} is the inverse of the covariance matrix of the data. Note that the covariance matrix Σ is the matrix whose ij^{th} entry is the covariance of the i^{th} and j^{th} attributes as defined by Equation 2.11.

Example 2.23. In Figure 2.19, there are 1000 points, whose x and y attributes have a correlation of 0.6. The distance between the two large points at the opposite ends of the long axis of the ellipse is 14.7 in terms of Euclidean distance, but only 6 with respect to Mahalanobis distance. In practice, computing the Mahalanobis distance is expensive, but can be worthwhile for data whose attributes are correlated. If the attributes are relatively uncorrelated, but have different ranges, then standardizing the variables is sufficient. ■

Combining Similarities for Heterogeneous Attributes

The previous definitions of similarity were based on approaches that assumed all the attributes were of the same type. A general approach is needed when the attributes are of different types. One straightforward approach is to compute the similarity between each attribute separately using Table 2.7, and then combine these similarities using a method that results in a similarity between 0 and 1. Typically, the overall similarity is defined as the average of all the individual attribute similarities.

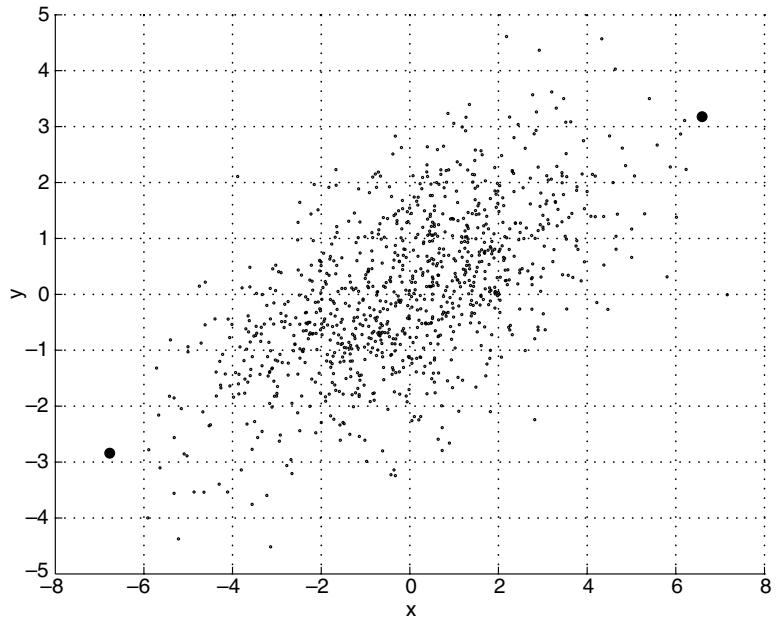


Figure 2.19. Set of two-dimensional points. The Mahalanobis distance between the two points represented by large dots is 6; their Euclidean distance is 14.7.

Unfortunately, this approach does not work well if some of the attributes are asymmetric attributes. For example, if all the attributes are asymmetric binary attributes, then the similarity measure suggested previously reduces to the simple matching coefficient, a measure that is not appropriate for asymmetric binary attributes. The easiest way to fix this problem is to omit asymmetric attributes from the similarity calculation when their values are 0 for both of the objects whose similarity is being computed. A similar approach also works well for handling missing values.

In summary, Algorithm 2.1 is effective for computing an overall similarity between two objects, \mathbf{x} and \mathbf{y} , with different types of attributes. This procedure can be easily modified to work with dissimilarities.

Using Weights

In much of the previous discussion, all attributes were treated equally when computing proximity. This is not desirable when some attributes are more important to the definition of proximity than others. To address these situations,

Algorithm 2.1 Similarities of heterogeneous objects.

1: For the k^{th} attribute, compute a similarity, $s_k(\mathbf{x}, \mathbf{y})$, in the range [0, 1].

2: Define an indicator variable, δ_k , for the k^{th} attribute as follows:

$$\delta_k = \begin{cases} 0 & \text{if the } k^{th} \text{ attribute is an asymmetric attribute and} \\ & \text{both objects have a value of 0, or if one of the objects} \\ & \text{has a missing value for the } k^{th} \text{ attribute} \\ 1 & \text{otherwise} \end{cases}$$

3: Compute the overall similarity between the two objects using the following formula:

$$\text{similarity}(\mathbf{x}, \mathbf{y}) = \frac{\sum_{k=1}^n \delta_k s_k(\mathbf{x}, \mathbf{y})}{\sum_{k=1}^n \delta_k} \quad (2.15)$$

the formulas for proximity can be modified by weighting the contribution of each attribute.

If the weights w_k sum to 1, then (2.15) becomes

$$\text{similarity}(\mathbf{x}, \mathbf{y}) = \frac{\sum_{k=1}^n w_k \delta_k s_k(\mathbf{x}, \mathbf{y})}{\sum_{k=1}^n \delta_k}. \quad (2.16)$$

The definition of the Minkowski distance can also be modified as follows:

$$d(\mathbf{x}, \mathbf{y}) = \left(\sum_{k=1}^n w_k |x_k - y_k|^r \right)^{1/r}. \quad (2.17)$$

2.4.7 Selecting the Right Proximity Measure

The following are a few general observations that may be helpful. First, the type of proximity measure should fit the type of data. For many types of dense, continuous data, metric distance measures such as Euclidean distance are often used. Proximity between continuous attributes is most often expressed in terms of differences, and distance measures provide a well-defined way of combining these differences into an overall proximity measure. Although attributes can have different scales and be of differing importance, these issues can often be dealt with as described earlier.

For sparse data, which often consists of asymmetric attributes, we typically employ similarity measures that ignore 0–0 matches. Conceptually, this reflects the fact that, for a pair of complex objects, similarity depends on the number of characteristics they both share, rather than the number of characteristics they both lack. More specifically, for sparse, asymmetric data, most

objects have only a few of the characteristics described by the attributes, and thus, are highly similar in terms of the characteristics they do not have. The cosine, Jaccard, and extended Jaccard measures are appropriate for such data.

There are other characteristics of data vectors that may need to be considered. Suppose, for example, that we are interested in comparing time series. If the magnitude of the time series is important (for example, each time series represent total sales of the same organization for a different year), then we could use Euclidean distance. If the time series represent different quantities (for example, blood pressure and oxygen consumption), then we usually want to determine if the time series have the same shape, not the same magnitude. Correlation, which uses a built-in normalization that accounts for differences in magnitude and level, would be more appropriate.

In some cases, transformation or normalization of the data is important for obtaining a proper similarity measure since such transformations are not always present in proximity measures. For instance, time series may have trends or periodic patterns that significantly impact similarity. Also, a proper computation of similarity may require that time lags be taken into account. Finally, two time series may only be similar over specific periods of time. For example, there is a strong relationship between temperature and the use of natural gas, but only during the heating season.

Practical consideration can also be important. Sometimes, a one or more proximity measures are already in use in a particular field, and thus, others will have answered the question of which proximity measures should be used. Other times, the software package or clustering algorithm being used may drastically limit the choices. If efficiency is a concern, then we may want to choose a proximity measure that has a property, such as the triangle inequality, that can be used to reduce the number of proximity calculations. (See Exercise 25.)

However, if common practice or practical restrictions do not dictate a choice, then the proper choice of a proximity measure can be a time-consuming task that requires careful consideration of both domain knowledge and the purpose for which the measure is being used. A number of different similarity measures may need to be evaluated to see which ones produce results that make the most sense.

2.5 Bibliographic Notes

It is essential to understand the nature of the data that is being analyzed, and at a fundamental level, this is the subject of measurement theory. In

particular, one of the initial motivations for defining types of attributes was to be precise about which statistical operations were valid for what sorts of data. We have presented the view of measurement theory that was initially described in a classic paper by S. S. Stevens [79]. (Tables 2.2 and 2.3 are derived from those presented by Stevens [80].) While this is the most common view and is reasonably easy to understand and apply, there is, of course, much more to measurement theory. An authoritative discussion can be found in a three-volume series on the foundations of measurement theory [63, 69, 81]. Also of interest is a wide-ranging article by Hand [55], which discusses measurement theory and statistics, and is accompanied by comments from other researchers in the field. Finally, there are many books and articles that describe measurement issues for particular areas of science and engineering.

Data quality is a broad subject that spans every discipline that uses data. Discussions of precision, bias, accuracy, and significant figures can be found in many introductory science, engineering, and statistics textbooks. The view of data quality as “fitness for use” is explained in more detail in the book by Redman [76]. Those interested in data quality may also be interested in MIT’s Total Data Quality Management program [70, 84]. However, the knowledge needed to deal with specific data quality issues in a particular domain is often best obtained by investigating the data quality practices of researchers in that field.

Aggregation is a less well-defined subject than many other preprocessing tasks. However, aggregation is one of the main techniques used by the database area of Online Analytical Processing (OLAP), which is discussed in Chapter 3. There has also been relevant work in the area of symbolic data analysis (Bock and Diday [47]). One of the goals in this area is to summarize traditional record data in terms of symbolic data objects whose attributes are more complex than traditional attributes. Specifically, these attributes can have values that are sets of values (categories), intervals, or sets of values with weights (histograms). Another goal of symbolic data analysis is to be able to perform clustering, classification, and other kinds of data analysis on data that consists of symbolic data objects.

Sampling is a subject that has been well studied in statistics and related fields. Many introductory statistics books, such as the one by Lindgren [65], have some discussion on sampling, and there are entire books devoted to the subject, such as the classic text by Cochran [49]. A survey of sampling for data mining is provided by Gu and Liu [54], while a survey of sampling for databases is provided by Olken and Rotem [72]. There are a number of other data mining and database-related sampling references that may be of interest,

including papers by Palmer and Faloutsos [74], Provost et al. [75], Toivonen [82], and Zaki et al. [85].

In statistics, the traditional techniques that have been used for dimensionality reduction are multidimensional scaling (MDS) (Borg and Groenen [48], Kruskal and Uslaner [64]) and principal component analysis (PCA) (Jolliffe [58]), which is similar to singular value decomposition (SVD) (Demmel [50]). Dimensionality reduction is discussed in more detail in Appendix B.

Discretization is a topic that has been extensively investigated in data mining. Some classification algorithms only work with categorical data, and association analysis requires binary data, and thus, there is a significant motivation to investigate how to best binarize or discretize continuous attributes. For association analysis, we refer the reader to work by Srikant and Agrawal [78], while some useful references for discretization in the area of classification include work by Dougherty et al. [51], Elomaa and Rousu [52], Fayyad and Irani [53], and Hussain et al. [56].

Feature selection is another topic well investigated in data mining. A broad coverage of this topic is provided in a survey by Molina et al. [71] and two books by Liu and Motoda [66, 67]. Other useful papers include those by Blum and Langley [46], Kohavi and John [62], and Liu et al. [68].

It is difficult to provide references for the subject of feature transformations because practices vary from one discipline to another. Many statistics books have a discussion of transformations, but typically the discussion is restricted to a particular purpose, such as ensuring the normality of a variable or making sure that variables have equal variance. We offer two references: Osborne [73] and Tukey [83].

While we have covered some of the most commonly used distance and similarity measures, there are hundreds of such measures and more are being created all the time. As with so many other topics in this chapter, many of these measures are specific to particular fields; e.g., in the area of time series see papers by Kalpakis et al. [59] and Keogh and Pazzani [61]. Clustering books provide the best general discussions. In particular, see the books by Anderberg [45], Jain and Dubes [57], Kaufman and Rousseeuw [60], and Sneath and Sokal [77].

Bibliography

- [45] M. R. Anderberg. *Cluster Analysis for Applications*. Academic Press, New York, December 1973.
- [46] A. Blum and P. Langley. Selection of Relevant Features and Examples in Machine Learning. *Artificial Intelligence*, 97(1–2):245–271, 1997.

- [47] H. H. Bock and E. Diday. *Analysis of Symbolic Data: Exploratory Methods for Extracting Statistical Information from Complex Data (Studies in Classification, Data Analysis, and Knowledge Organization)*. Springer-Verlag Telos, January 2000.
- [48] I. Borg and P. Groenen. *Modern Multidimensional Scaling—Theory and Applications*. Springer-Verlag, February 1997.
- [49] W. G. Cochran. *Sampling Techniques*. John Wiley & Sons, 3rd edition, July 1977.
- [50] J. W. Demmel. *Applied Numerical Linear Algebra*. Society for Industrial & Applied Mathematics, September 1997.
- [51] J. Dougherty, R. Kohavi, and M. Sahami. Supervised and Unsupervised Discretization of Continuous Features. In *Proc. of the 12th Intl. Conf. on Machine Learning*, pages 194–202, 1995.
- [52] T. Elomaa and J. Rousu. General and Efficient Multisplitting of Numerical Attributes. *Machine Learning*, 36(3):201–244, 1999.
- [53] U. M. Fayyad and K. B. Irani. Multi-interval discretization of continuousvalued attributes for classification learning. In *Proc. 13th Int. Joint Conf. on Artificial Intelligence*, pages 1022–1027. Morgan Kaufman, 1993.
- [54] F. H. Gaohua Gu and H. Liu. Sampling and Its Application in Data Mining: A Survey. Technical Report TRA6/00, National University of Singapore, Singapore, 2000.
- [55] D. J. Hand. Statistics and the Theory of Measurement. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 159(3):445–492, 1996.
- [56] F. Hussain, H. Liu, C. L. Tan, and M. Dash. TRC6/99: Discretization: an enabling technique. Technical report, National University of Singapore, Singapore, 1999.
- [57] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice Hall Advanced Reference Series. Prentice Hall, March 1988. Book available online at http://www.cse.msu.edu/~jain/Clustering_Jain_Dubes.pdf.
- [58] I. T. Jolliffe. *Principal Component Analysis*. Springer Verlag, 2nd edition, October 2002.
- [59] K. Kalpakis, D. Gada, and V. Puttagunta. Distance Measures for Effective Clustering of ARIMA Time-Series. In *Proc. of the 2001 IEEE Intl. Conf. on Data Mining*, pages 273–280. IEEE Computer Society, 2001.
- [60] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley Series in Probability and Statistics. John Wiley and Sons, New York, November 1990.
- [61] E. J. Keogh and M. J. Pazzani. Scaling up dynamic time warping for datamining applications. In *KDD*, pages 285–289, 2000.
- [62] R. Kohavi and G. H. John. Wrappers for Feature Subset Selection. *Artificial Intelligence*, 97(1–2):273–324, 1997.
- [63] D. Krantz, R. D. Luce, P. Suppes, and A. Tversky. *Foundations of Measurements: Volume 1: Additive and polynomial representations*. Academic Press, New York, 1971.
- [64] J. B. Kruskal and E. M. Uslaner. *Multidimensional Scaling*. Sage Publications, August 1978.
- [65] B. W. Lindgren. *Statistical Theory*. CRC Press, January 1993.
- [66] H. Liu and H. Motoda, editors. *Feature Extraction, Construction and Selection: A Data Mining Perspective*. Kluwer International Series in Engineering and Computer Science, 453. Kluwer Academic Publishers, July 1998.
- [67] H. Liu and H. Motoda. *Feature Selection for Knowledge Discovery and Data Mining*. Kluwer International Series in Engineering and Computer Science, 454. Kluwer Academic Publishers, July 1998.

- [68] H. Liu, H. Motoda, and L. Yu. Feature Extraction, Selection, and Construction. In N. Ye, editor, *The Handbook of Data Mining*, pages 22–41. Lawrence Erlbaum Associates, Inc., Mahwah, NJ, 2003.
- [69] R. D. Luce, D. Krantz, P. Suppes, and A. Tversky. *Foundations of Measurements: Volume 3: Representation, Axiomatization, and Invariance*. Academic Press, New York, 1990.
- [70] MIT Total Data Quality Management Program. web.mit.edu/tdqm/www/index.shtml, 2003.
- [71] L. C. Molina, L. Belanche, and A. Nebot. Feature Selection Algorithms: A Survey and Experimental Evaluation. In *Proc. of the 2002 IEEE Intl. Conf. on Data Mining*, 2002.
- [72] F. Olken and D. Rotem. Random Sampling from Databases—A Survey. *Statistics & Computing*, 5(1):25–42, March 1995.
- [73] J. Osborne. Notes on the Use of Data Transformations. *Practical Assessment, Research & Evaluation*, 28(6), 2002.
- [74] C. R. Palmer and C. Faloutsos. Density biased sampling: An improved method for data mining and clustering. *ACM SIGMOD Record*, 29(2):82–92, 2000.
- [75] F. J. Provost, D. Jensen, and T. Oates. Efficient Progressive Sampling. In *Proc. of the 5th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 23–32, 1999.
- [76] T. C. Redman. *Data Quality: The Field Guide*. Digital Press, January 2001.
- [77] P. H. A. Sneath and R. R. Sokal. *Numerical Taxonomy*. Freeman, San Francisco, 1971.
- [78] R. Srikant and R. Agrawal. Mining Quantitative Association Rules in Large Relational Tables. In *Proc. of 1996 ACM-SIGMOD Intl. Conf. on Management of Data*, pages 1–12, Montreal, Quebec, Canada, August 1996.
- [79] S. S. Stevens. On the Theory of Scales of Measurement. *Science*, 103(2684):677–680, June 1946.
- [80] S. S. Stevens. Measurement. In G. M. Maranell, editor, *Scaling: A Sourcebook for Behavioral Scientists*, pages 22–41. Aldine Publishing Co., Chicago, 1974.
- [81] P. Suppes, D. Krantz, R. D. Luce, and A. Tversky. *Foundations of Measurements: Volume 2: Geometrical, Threshold, and Probabilistic Representations*. Academic Press, New York, 1989.
- [82] H. Toivonen. Sampling Large Databases for Association Rules. In *VLDB96*, pages 134–145. Morgan Kaufman, September 1996.
- [83] J. W. Tukey. On the Comparative Anatomy of Transformations. *Annals of Mathematical Statistics*, 28(3):602–632, September 1957.
- [84] R. Y. Wang, M. Ziad, Y. W. Lee, and Y. R. Wang. *Data Quality*. The Kluwer International Series on Advances in Database Systems, Volume 23. Kluwer Academic Publishers, January 2001.
- [85] M. J. Zaki, S. Parthasarathy, W. Li, and M. Ogihara. Evaluation of Sampling for Data Mining of Association Rules. Technical Report TR617, Rensselaer Polytechnic Institute, 1996.

2.6 Exercises

1. In the initial example of Chapter 2, the statistician says, “Yes, fields 2 and 3 are basically the same.” Can you tell from the three lines of sample data that are shown why she says that?

2. Classify the following attributes as binary, discrete, or continuous. Also classify them as qualitative (nominal or ordinal) or quantitative (interval or ratio). Some cases may have more than one interpretation, so briefly indicate your reasoning if you think there may be some ambiguity.

Example: Age in years. **Answer:** Discrete, quantitative, ratio

- (a) Time in terms of AM or PM.
 - (b) Brightness as measured by a light meter.
 - (c) Brightness as measured by people's judgments.
 - (d) Angles as measured in degrees between 0 and 360.
 - (e) Bronze, Silver, and Gold medals as awarded at the Olympics.
 - (f) Height above sea level.
 - (g) Number of patients in a hospital.
 - (h) ISBN numbers for books. (Look up the format on the Web.)
 - (i) Ability to pass light in terms of the following values: opaque, translucent, transparent.
 - (j) Military rank.
 - (k) Distance from the center of campus.
 - (l) Density of a substance in grams per cubic centimeter.
 - (m) Coat check number. (When you attend an event, you can often give your coat to someone who, in turn, gives you a number that you can use to claim your coat when you leave.)
3. You are approached by the marketing director of a local company, who believes that he has devised a foolproof way to measure customer satisfaction. He explains his scheme as follows: "It's so simple that I can't believe that no one has thought of it before. I just keep track of the number of customer complaints for each product. I read in a data mining book that counts are ratio attributes, and so, my measure of product satisfaction must be a ratio attribute. But when I rated the products based on my new customer satisfaction measure and showed them to my boss, he told me that I had overlooked the obvious, and that my measure was worthless. I think that he was just mad because our best-selling product had the worst satisfaction since it had the most complaints. Could you help me set him straight?"
- (a) Who is right, the marketing director or his boss? If you answered, his boss, what would you do to fix the measure of satisfaction?
 - (b) What can you say about the attribute type of the original product satisfaction attribute?

4. A few months later, you are again approached by the same marketing director as in Exercise 3. This time, he has devised a better approach to measure the extent to which a customer prefers one product over other, similar products. He explains, "When we develop new products, we typically create several variations and evaluate which one customers prefer. Our standard procedure is to give our test subjects all of the product variations at one time and then ask them to rank the product variations in order of preference. However, our test subjects are very indecisive, especially when there are more than two products. As a result, testing takes forever. I suggested that we perform the comparisons in pairs and then use these comparisons to get the rankings. Thus, if we have three product variations, we have the customers compare variations 1 and 2, then 2 and 3, and finally 3 and 1. Our testing time with my new procedure is a third of what it was for the old procedure, but the employees conducting the tests complain that they cannot come up with a consistent ranking from the results. And my boss wants the latest product evaluations, yesterday. I should also mention that he was the person who came up with the old product evaluation approach. Can you help me?"
- (a) Is the marketing director in trouble? Will his approach work for generating an ordinal ranking of the product variations in terms of customer preference? Explain.
 - (b) Is there a way to fix the marketing director's approach? More generally, what can you say about trying to create an ordinal measurement scale based on pairwise comparisons?
 - (c) For the original product evaluation scheme, the overall rankings of each product variation are found by computing its average over all test subjects. Comment on whether you think that this is a reasonable approach. What other approaches might you take?
5. Can you think of a situation in which identification numbers would be useful for prediction?
6. An educational psychologist wants to use association analysis to analyze test results. The test consists of 100 questions with four possible answers each.
- (a) How would you convert this data into a form suitable for association analysis?
 - (b) In particular, what type of attributes would you have and how many of them are there?
7. Which of the following quantities is likely to show more temporal autocorrelation: daily rainfall or daily temperature? Why?
8. Discuss why a document-term matrix is an example of a data set that has asymmetric discrete or asymmetric continuous features.

9. Many sciences rely on observation instead of (or in addition to) designed experiments. Compare the data quality issues involved in observational science with those of experimental science and data mining.
10. Discuss the difference between the precision of a measurement and the terms single and double precision, as they are used in computer science, typically to represent floating-point numbers that require 32 and 64 bits, respectively.
11. Give at least two advantages to working with data stored in text files instead of in a binary format.
12. Distinguish between noise and outliers. Be sure to consider the following questions.
 - (a) Is noise ever interesting or desirable? Outliers?
 - (b) Can noise objects be outliers?
 - (c) Are noise objects always outliers?
 - (d) Are outliers always noise objects?
 - (e) Can noise make a typical value into an unusual one, or vice versa?
13. Consider the problem of finding the K nearest neighbors of a data object. A programmer designs Algorithm 2.2 for this task.

Algorithm 2.2 Algorithm for finding K nearest neighbors.

```

1: for  $i = 1$  to number of data objects do
2:   Find the distances of the  $i^{th}$  object to all other objects.
3:   Sort these distances in decreasing order.
   (Keep track of which object is associated with each distance.)
4:   return the objects associated with the first  $K$  distances of the sorted list
5: end for

```

- (a) Describe the potential problems with this algorithm if there are duplicate objects in the data set. Assume the distance function will only return a distance of 0 for objects that are the same.
- (b) How would you fix this problem?
14. The following attributes are measured for members of a herd of Asian elephants: *weight*, *height*, *tusk length*, *trunk length*, and *ear area*. Based on these measurements, what sort of similarity measure from Section 2.4 would you use to compare or group these elephants? Justify your answer and explain any special circumstances.

15. You are given a set of m objects that is divided into K groups, where the i^{th} group is of size m_i . If the goal is to obtain a sample of size $n < m$, what is the difference between the following two sampling schemes? (Assume sampling with replacement.)

- (a) We randomly select $n * m_i / m$ elements from each group.
- (b) We randomly select n elements from the data set, without regard for the group to which an object belongs.

16. Consider a document-term matrix, where tf_{ij} is the frequency of the i^{th} word (term) in the j^{th} document and m is the number of documents. Consider the variable transformation that is defined by

$$tf'_{ij} = tf_{ij} * \log \frac{m}{df_i}, \quad (2.18)$$

where df_i is the number of documents in which the i^{th} term appears, which is known as the **document frequency** of the term. This transformation is known as the **inverse document frequency** transformation.

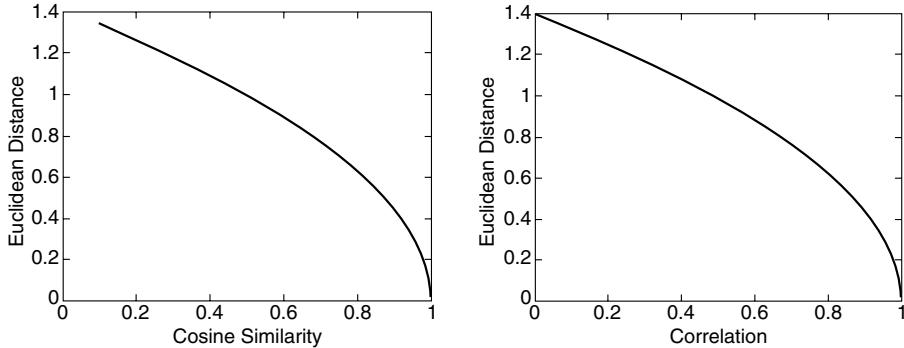
- (a) What is the effect of this transformation if a term occurs in one document?
In every document?
 - (b) What might be the purpose of this transformation?
17. Assume that we apply a square root transformation to a ratio attribute x to obtain the new attribute x^* . As part of your analysis, you identify an interval (a, b) in which x^* has a linear relationship to another attribute y .
- (a) What is the corresponding interval (a, b) in terms of x ?
 - (b) Give an equation that relates y to x .
18. This exercise compares and contrasts some similarity and distance measures.

- (a) For binary data, the L1 distance corresponds to the Hamming distance; that is, the number of bits that are different between two binary vectors. The Jaccard similarity is a measure of the similarity between two binary vectors. Compute the Hamming distance and the Jaccard similarity between the following two binary vectors.

$$\begin{aligned}\mathbf{x} &= 0101010001 \\ \mathbf{y} &= 0100011000\end{aligned}$$

- (b) Which approach, Jaccard or Hamming distance, is more similar to the Simple Matching Coefficient, and which approach is more similar to the cosine measure? Explain. (Note: The Hamming measure is a distance, while the other three measures are similarities, but don't let this confuse you.)

- (c) Suppose that you are comparing how similar two organisms of different species are in terms of the number of genes they share. Describe which measure, Hamming or Jaccard, you think would be more appropriate for comparing the genetic makeup of two organisms. Explain. (Assume that each animal is represented as a binary vector, where each attribute is 1 if a particular gene is present in the organism and 0 otherwise.)
- (d) If you wanted to compare the genetic makeup of two organisms of the same species, e.g., two human beings, would you use the Hamming distance, the Jaccard coefficient, or a different measure of similarity or distance? Explain. (Note that two human beings share > 99.9% of the same genes.)
19. For the following vectors, \mathbf{x} and \mathbf{y} , calculate the indicated similarity or distance measures.
- (a) $\mathbf{x} = (1, 1, 1, 1)$, $\mathbf{y} = (2, 2, 2, 2)$ cosine, correlation, Euclidean
 - (b) $\mathbf{x} = (0, 1, 0, 1)$, $\mathbf{y} = (1, 0, 1, 0)$ cosine, correlation, Euclidean, Jaccard
 - (c) $\mathbf{x} = (0, -1, 0, 1)$, $\mathbf{y} = (1, 0, -1, 0)$ cosine, correlation, Euclidean
 - (d) $\mathbf{x} = (1, 1, 0, 1, 0, 1)$, $\mathbf{y} = (1, 1, 1, 0, 0, 1)$ cosine, correlation, Jaccard
 - (e) $\mathbf{x} = (2, -1, 0, 2, 0, -3)$, $\mathbf{y} = (-1, 1, -1, 0, 0, -1)$ cosine, correlation
20. Here, we further explore the cosine and correlation measures.
- (a) What is the range of values that are possible for the cosine measure?
 - (b) If two objects have a cosine measure of 1, are they identical? Explain.
 - (c) What is the relationship of the cosine measure to correlation, if any? (Hint: Look at statistical measures such as mean and standard deviation in cases where cosine and correlation are the same and different.)
 - (d) Figure 2.20(a) shows the relationship of the cosine measure to Euclidean distance for 100,000 randomly generated points that have been normalized to have an L₂ length of 1. What general observation can you make about the relationship between Euclidean distance and cosine similarity when vectors have an L₂ norm of 1?
 - (e) Figure 2.20(b) shows the relationship of correlation to Euclidean distance for 100,000 randomly generated points that have been standardized to have a mean of 0 and a standard deviation of 1. What general observation can you make about the relationship between Euclidean distance and correlation when the vectors have been standardized to have a mean of 0 and a standard deviation of 1?
 - (f) Derive the mathematical relationship between cosine similarity and Euclidean distance when each data object has an L₂ length of 1.
 - (g) Derive the mathematical relationship between correlation and Euclidean distance when each data point has been been standardized by subtracting its mean and dividing by its standard deviation.



(a) Relationship between Euclidean distance and the cosine measure.

(b) Relationship between Euclidean distance and correlation.

Figure 2.20. Graphs for Exercise 20.

21. Show that the set difference metric given by

$$d(A, B) = \text{size}(A - B) + \text{size}(B - A) \quad (2.19)$$

satisfies the metric axioms given on page 70. A and B are sets and $A - B$ is the set difference.

22. Discuss how you might map correlation values from the interval $[-1,1]$ to the interval $[0,1]$. Note that the type of transformation that you use might depend on the application that you have in mind. Thus, consider two applications: clustering time series and predicting the behavior of one time series given another.
23. Given a similarity measure with values in the interval $[0,1]$ describe two ways to transform this similarity value into a dissimilarity value in the interval $[0,\infty]$.
24. Proximity is typically defined between a pair of objects.
- (a) Define two ways in which you might define the proximity among a group of objects.
 - (b) How might you define the distance between two sets of points in Euclidean space?
 - (c) How might you define the proximity between two sets of data objects? (Make no assumption about the data objects, except that a proximity measure is defined between any pair of objects.)
25. You are given a set of points S in Euclidean space, as well as the distance of each point in S to a point \mathbf{x} . (It does not matter if $\mathbf{x} \in S$.)

- (a) If the goal is to find all points within a specified distance ε of point \mathbf{y} , $\mathbf{y} \neq \mathbf{x}$, explain how you could use the triangle inequality and the already calculated distances to \mathbf{x} to potentially reduce the number of distance calculations necessary? Hint: The triangle inequality, $d(\mathbf{x}, \mathbf{z}) \leq d(\mathbf{x}, \mathbf{y}) + d(\mathbf{y}, \mathbf{z})$, can be rewritten as $d(\mathbf{x}, \mathbf{y}) \geq d(\mathbf{x}, \mathbf{z}) - d(\mathbf{y}, \mathbf{z})$.
 - (b) In general, how would the distance between \mathbf{x} and \mathbf{y} affect the number of distance calculations?
 - (c) Suppose that you can find a small subset of points S' , from the original data set, such that every point in the data set is within a specified distance ε of at least one of the points in S' , and that you also have the pairwise distance matrix for S' . Describe a technique that uses this information to compute, with a minimum of distance calculations, the set of all points within a distance of β of a specified point from the data set.
26. Show that 1 minus the Jaccard similarity is a distance measure between two data objects, \mathbf{x} and \mathbf{y} , that satisfies the metric axioms given on page 70. Specifically, $d(\mathbf{x}, \mathbf{y}) = 1 - J(\mathbf{x}, \mathbf{y})$.
27. Show that the distance measure defined as the angle between two data vectors, \mathbf{x} and \mathbf{y} , satisfies the metric axioms given on page 70. Specifically, $d(\mathbf{x}, \mathbf{y}) = \arccos(\cos(\mathbf{x}, \mathbf{y}))$.
28. Explain why computing the proximity between two attributes is often simpler than computing the similarity between two objects.

3

Exploring Data

The previous chapter addressed high-level data issues that are important in the knowledge discovery process. This chapter provides an introduction to **data exploration**, which is a preliminary investigation of the data in order to better understand its specific characteristics. Data exploration can aid in selecting the appropriate preprocessing and data analysis techniques. It can even address some of the questions typically answered by data mining. For example, patterns can sometimes be found by visually inspecting the data. Also, some of the techniques used in data exploration, such as visualization, can be used to understand and interpret data mining results.

This chapter covers three major topics: summary statistics, visualization, and On-Line Analytical Processing (OLAP). Summary statistics, such as the mean and standard deviation of a set of values, and visualization techniques, such as histograms and scatter plots, are standard methods that are widely employed for data exploration. OLAP, which is a more recent development, consists of a set of techniques for exploring multidimensional arrays of values. OLAP-related analysis functions focus on various ways to create summary data tables from a multidimensional data array. These techniques include aggregating data either across various dimensions or across various attribute values. For instance, if we are given sales information reported according to product, location, and date, OLAP techniques can be used to create a summary that describes the sales activity at a particular location by month and product category.

The topics covered in this chapter have considerable overlap with the area known as **Exploratory Data Analysis** (EDA), which was created in the 1970s by the prominent statistician, John Tukey. This chapter, like EDA, places a heavy emphasis on visualization. Unlike EDA, this chapter does not include topics such as cluster analysis or anomaly detection. There are two

reasons for this. First, data mining views descriptive data analysis techniques as an end in themselves, whereas statistics, from which EDA originated, tends to view hypothesis-based testing as the final goal. Second, cluster analysis and anomaly detection are large areas and require full chapters for an in-depth discussion. Hence, cluster analysis is covered in Chapters 8 and 9, while anomaly detection is discussed in Chapter 10.

3.1 The Iris Data Set

In the following discussion, we will often refer to the Iris data set that is available from the University of California at Irvine (UCI) Machine Learning Repository. It consists of information on 150 Iris flowers, 50 each from one of three Iris species: Setosa, Versicolour, and Virginica. Each flower is characterized by five attributes:

1. sepal length in centimeters
2. sepal width in centimeters
3. petal length in centimeters
4. petal width in centimeters
5. class (Setosa, Versicolour, Virginica)

The sepals of a flower are the outer structures that protect the more fragile parts of the flower, such as the petals. In many flowers, the sepals are green, and only the petals are colorful. For Irises, however, the sepals are also colorful. As illustrated by the picture of a Virginica Iris in Figure 3.1, the sepals of an Iris are larger than the petals and are drooping, while the petals are upright.

3.2 Summary Statistics

Summary statistics are quantities, such as the mean and standard deviation, that capture various characteristics of a potentially large set of values with a single number or a small set of numbers. Everyday examples of summary statistics are the average household income or the fraction of college students who complete an undergraduate degree in four years. Indeed, for many people, summary statistics are the most visible manifestation of statistics. We will concentrate on summary statistics for the values of a single attribute, but will provide a brief description of some multivariate summary statistics.

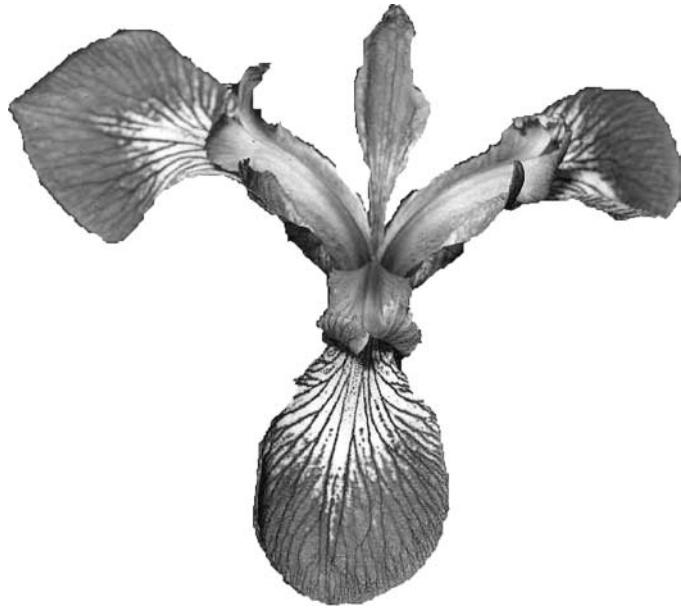


Figure 3.1. Picture of Iris Virginica. Robert H. Mohlenbrock @ USDA-NRCS PLANTS Database/USDA NRCS. 1995. Northeast wetland flora: Field office guide to plant species. Northeast National Technical Center, Chester, PA. Background removed.

This section considers only the descriptive nature of summary statistics. However, as described in Appendix C, statistics views data as arising from an underlying statistical process that is characterized by various parameters, and some of the summary statistics discussed here can be viewed as estimates of statistical parameters of the underlying distribution that generated the data.

3.2.1 Frequencies and the Mode

Given a set of unordered categorical values, there is not much that can be done to further characterize the values except to compute the frequency with which each value occurs for a particular set of data. Given a categorical attribute x , which can take values $\{v_1, \dots, v_i, \dots, v_k\}$ and a set of m objects, the frequency of a value v_i is defined as

$$\text{frequency}(v_i) = \frac{\text{number of objects with attribute value } v_i}{m}. \quad (3.1)$$

The **mode** of a categorical attribute is the value that has the highest frequency.

Example 3.1. Consider a set of students who have an attribute, *class*, which can take values from the set $\{\text{freshman}, \text{sophomore}, \text{junior}, \text{senior}\}$. Table 3.1 shows the number of students for each value of the *class* attribute. The mode of the *class* attribute is *freshman*, with a frequency of 0.33. This may indicate dropouts due to attrition or a larger than usual freshman class.

Table 3.1. Class size for students in a hypothetical college.

Class	Size	Frequency
freshman	140	0.33
sophomore	160	0.27
junior	130	0.22
senior	170	0.18

Categorical attributes often, but not always, have a small number of values, and consequently, the mode and frequencies of these values can be interesting and useful. Notice, though, that for the Iris data set and the *class* attribute, the three types of flower all have the same frequency, and therefore, the notion of a mode is not interesting.

For continuous data, the mode, as currently defined, is often not useful because a single value may not occur more than once. Nonetheless, in some cases, the mode may indicate important information about the nature of the values or the presence of missing values. For example, the heights of 20 people measured to the nearest millimeter will typically not repeat, but if the heights are measured to the nearest tenth of a meter, then some people may have the same height. Also, if a unique value is used to indicate a missing value, then this value will often show up as the mode.

3.2.2 Percentiles

For ordered data, it is more useful to consider the **percentiles** of a set of values. In particular, given an ordinal or continuous attribute x and a number p between 0 and 100, the p^{th} percentile x_p is a value of x such that $p\%$ of the observed values of x are less than x_p . For instance, the 50^{th} percentile is the value $x_{50\%}$ such that 50% of all values of x are less than $x_{50\%}$. Table 3.2 shows the percentiles for the four quantitative attributes of the Iris data set.

Table 3.2. Percentiles for sepal length, sepal width, petal length, and petal width. (All values are in centimeters.)

Percentile	Sepal Length	Sepal Width	Petal Length	Petal Width
0	4.3	2.0	1.0	0.1
10	4.8	2.5	1.4	0.2
20	5.0	2.7	1.5	0.2
30	5.2	2.8	1.7	0.4
40	5.6	3.0	3.9	1.2
50	5.8	3.0	4.4	1.3
60	6.1	3.1	4.6	1.5
70	6.3	3.2	5.0	1.8
80	6.6	3.4	5.4	1.9
90	6.9	3.6	5.8	2.2
100	7.9	4.4	6.9	2.5

Example 3.2. The percentiles, $x_{0\%}, x_{10\%}, \dots, x_{90\%}, x_{100\%}$ of the integers from 1 to 10 are, in order, the following: 1.0, 1.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5, 9.5, 10.0. By tradition, $\min(x) = x_{0\%}$ and $\max(x) = x_{100\%}$. ■

3.2.3 Measures of Location: Mean and Median

For continuous data, two of the most widely used summary statistics are the **mean** and **median**, which are measures of the *location* of a set of values. Consider a set of m objects and an attribute x . Let $\{x_1, \dots, x_m\}$ be the attribute values of x for these m objects. As a concrete example, these values might be the heights of m children. Let $\{x_{(1)}, \dots, x_{(m)}\}$ represent the values of x after they have been sorted in non-decreasing order. Thus, $x_{(1)} = \min(x)$ and $x_{(m)} = \max(x)$. Then, the mean and median are defined as follows:

$$\text{mean}(x) = \bar{x} = \frac{1}{m} \sum_{i=1}^m x_i \quad (3.2)$$

$$\text{median}(x) = \begin{cases} x_{(r+1)} & \text{if } m \text{ is odd, i.e., } m = 2r + 1 \\ \frac{1}{2}(x_{(r)} + x_{(r+1)}) & \text{if } m \text{ is even, i.e., } m = 2r \end{cases} \quad (3.3)$$

To summarize, the median is the middle value if there are an odd number of values, and the average of the two middle values if the number of values is even. Thus, for seven values, the median is $x_{(4)}$, while for ten values, the median is $\frac{1}{2}(x_{(5)} + x_{(6)})$.

Although the mean is sometimes interpreted as the middle of a set of values, this is only correct if the values are distributed in a symmetric manner. If the distribution of values is skewed, then the median is a better indicator of the middle. Also, the mean is sensitive to the presence of outliers. For data with outliers, the median again provides a more robust estimate of the middle of a set of values.

To overcome problems with the traditional definition of a mean, the notion of a **trimmed mean** is sometimes used. A percentage p between 0 and 100 is specified, the top and bottom $(p/2)\%$ of the data is thrown out, and the mean is then calculated in the normal way. The median is a trimmed mean with $p = 100\%$, while the standard mean corresponds to $p = 0\%$.

Example 3.3. Consider the set of values $\{1, 2, 3, 4, 5, 90\}$. The mean of these values is 17.5, while the median is 3.5. The trimmed mean with $p = 40\%$ is also 3.5. ■

Example 3.4. The means, medians, and trimmed means ($p = 20\%$) of the four quantitative attributes of the Iris data are given in Table 3.3. The three measures of location have similar values except for the attribute *petal length*.

Table 3.3. Means and medians for sepal length, sepal width, petal length, and petal width. (All values are in centimeters.)

Measure	Sepal Length	Sepal Width	Petal Length	Petal Width
mean	5.84	3.05	3.76	1.20
median	5.80	3.00	4.35	1.30
trimmed mean (20%)	5.79	3.02	3.72	1.12

3.2.4 Measures of Spread: Range and Variance

Another set of commonly used summary statistics for continuous data are those that measure the dispersion or spread of a set of values. Such measures indicate if the attribute values are widely spread out or if they are relatively concentrated around a single point such as the mean.

The simplest measure of spread is the **range**, which, given an attribute x with a set of m values $\{x_1, \dots, x_m\}$, is defined as

$$\text{range}(x) = \max(x) - \min(x) = x_{(m)} - x_{(1)}. \quad (3.4)$$

Table 3.4. Range, standard deviation (std), absolute average difference (AAD), median absolute difference (MAD), and interquartile range (IQR) for sepal length, sepal width, petal length, and petal width. (All values are in centimeters.)

Measure	Sepal Length	Sepal Width	Petal Length	Petal Width
range	3.6	2.4	5.9	2.4
std	0.8	0.4	1.8	0.8
AAD	0.7	0.3	1.6	0.6
MAD	0.7	0.3	1.2	0.7
IQR	1.3	0.5	3.5	1.5

Although the range identifies the maximum spread, it can be misleading if most of the values are concentrated in a narrow band of values, but there are also a relatively small number of more extreme values. Hence, the **variance** is preferred as a measure of spread. The variance of the (observed) values of an attribute x is typically written as s_x^2 and is defined below. The **standard deviation**, which is the square root of the variance, is written as s_x and has the same units as x .

$$\text{variance}(x) = s_x^2 = \frac{1}{m-1} \sum_{i=1}^m (x_i - \bar{x})^2 \quad (3.5)$$

The mean can be distorted by outliers, and since the variance is computed using the mean, it is also sensitive to outliers. Indeed, the variance is particularly sensitive to outliers since it uses the squared difference between the mean and other values. As a result, more robust estimates of the spread of a set of values are often used. Following are the definitions of three such measures: the **absolute average deviation** (AAD), the **median absolute deviation** (MAD), and the **interquartile range**(IQR). Table 3.4 shows these measures for the Iris data set.

$$\text{AAD}(x) = \frac{1}{m} \sum_{i=1}^m |x_i - \bar{x}| \quad (3.6)$$

$$\text{MAD}(x) = \text{median}\left(\{|x_1 - \bar{x}|, \dots, |x_m - \bar{x}|\}\right) \quad (3.7)$$

$$\text{interquartile range}(x) = x_{75\%} - x_{25\%} \quad (3.8)$$

3.2.5 Multivariate Summary Statistics

Measures of location for data that consists of several attributes (multivariate data) can be obtained by computing the mean or median separately for each attribute. Thus, given a data set the mean of the data objects, $\bar{\mathbf{x}}$, is given by

$$\bar{\mathbf{x}} = (\bar{x}_1, \dots, \bar{x}_n), \quad (3.9)$$

where \bar{x}_i is the mean of the i^{th} attribute x_i .

For multivariate data, the spread of each attribute can be computed independently of the other attributes using any of the approaches described in Section 3.2.4. However, for data with continuous variables, the spread of the data is most commonly captured by the **covariance matrix** \mathbf{S} , whose ij^{th} entry s_{ij} is the covariance of the i^{th} and j^{th} attributes of the data. Thus, if x_i and x_j are the i^{th} and j^{th} attributes, then

$$s_{ij} = \text{covariance}(x_i, x_j). \quad (3.10)$$

In turn, $\text{covariance}(x_i, x_j)$ is given by

$$\text{covariance}(x_i, x_j) = \frac{1}{m-1} \sum_{k=1}^m (x_{ki} - \bar{x}_i)(x_{kj} - \bar{x}_j), \quad (3.11)$$

where x_{ki} and x_{kj} are the values of the i^{th} and j^{th} attributes for the k^{th} object. Notice that $\text{covariance}(x_i, x_i) = \text{variance}(x_i)$. Thus, the covariance matrix has the variances of the attributes along the diagonal.

The covariance of two attributes is a measure of the degree to which two attributes vary together and depends on the magnitudes of the variables. A value near 0 indicates that two attributes do not have a (linear) relationship, but it is not possible to judge the degree of relationship between two variables by looking only at the value of the covariance. Because the correlation of two attributes immediately gives an indication of how strongly two attributes are (linearly) related, correlation is preferred to covariance for data exploration. (Also see the discussion of correlation in Section 2.4.5.) The ij^{th} entry of the **correlation matrix** \mathbf{R} , is the correlation between the i^{th} and j^{th} attributes of the data. If x_i and x_j are the i^{th} and j^{th} attributes, then

$$r_{ij} = \text{correlation}(x_i, x_j) = \frac{\text{covariance}(x_i, x_j)}{s_i s_j}, \quad (3.12)$$

where s_i and s_j are the variances of x_i and x_j , respectively. The diagonal entries of \mathbf{R} are $\text{correlation}(x_i, x_i) = 1$, while the other entries are between -1 and 1 . It is also useful to consider correlation matrices that contain the pairwise correlations of objects instead of attributes.

3.2.6 Other Ways to Summarize the Data

There are, of course, other types of summary statistics. For instance, the **skewness** of a set of values measures the degree to which the values are symmetrically distributed around the mean. There are also other characteristics of the data that are not easy to measure quantitatively, such as whether the distribution of values is multimodal; i.e., the data has multiple “bumps” where most of the values are concentrated. In many cases, however, the most effective approach to understanding the more complicated or subtle aspects of how the values of an attribute are distributed, is to view the values graphically in the form of a histogram. (Histograms are discussed in the next section.)

3.3 Visualization

Data visualization is the display of information in a graphic or tabular format. Successful visualization requires that the data (information) be converted into a visual format so that the characteristics of the data and the relationships among data items or attributes can be analyzed or reported. The goal of visualization is the interpretation of the visualized information by a person and the formation of a mental model of the information.

In everyday life, visual techniques such as graphs and tables are often the preferred approach used to explain the weather, the economy, and the results of political elections. Likewise, while algorithmic or mathematical approaches are often emphasized in most technical disciplines—data mining included—visual techniques can play a key role in data analysis. In fact, sometimes the use of visualization techniques in data mining is referred to as **visual data mining**.

3.3.1 Motivations for Visualization

The overriding motivation for using visualization is that people can quickly absorb large amounts of visual information and find patterns in it. Consider Figure 3.2, which shows the Sea Surface Temperature (SST) in degrees Celsius for July, 1982. This picture summarizes the information from approximately 250,000 numbers and is readily interpreted in a few seconds. For example, it

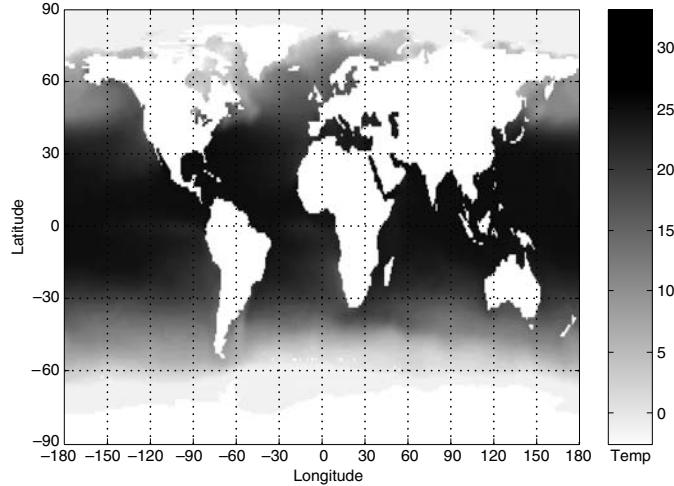


Figure 3.2. Sea Surface Temperature (SST) for July, 1982.

is easy to see that the ocean temperature is highest at the equator and lowest at the poles.

Another general motivation for visualization is to make use of the domain knowledge that is “locked up in people’s heads.” While the use of domain knowledge is an important task in data mining, it is often difficult or impossible to fully utilize such knowledge in statistical or algorithmic tools. In some cases, an analysis can be performed using non-visual tools, and then the results presented visually for evaluation by the domain expert. In other cases, having a domain specialist examine visualizations of the data may be the best way of finding patterns of interest since, by using domain knowledge, a person can often quickly eliminate many uninteresting patterns and direct the focus to the patterns that are important.

3.3.2 General Concepts

This section explores some of the general concepts related to visualization, in particular, general approaches for visualizing the data and its attributes. A number of visualization techniques are mentioned briefly and will be described in more detail when we discuss specific approaches later on. We assume that the reader is familiar with line graphs, bar charts, and scatter plots.

Representation: Mapping Data to Graphical Elements

The first step in visualization is the mapping of information to a visual format; i.e., mapping the objects, attributes, and relationships in a set of information to visual objects, attributes, and relationships. That is, data objects, their attributes, and the relationships among data objects are translated into graphical elements such as points, lines, shapes, and colors.

Objects are usually represented in one of three ways. First, if only a single categorical attribute of the object is being considered, then objects are often lumped into categories based on the value of that attribute, and these categories are displayed as an entry in a table or an area on a screen. (Examples shown later in this chapter are a cross-tabulation table and a bar chart.) Second, if an object has multiple attributes, then the object can be displayed as a row (or column) of a table or as a line on a graph. Finally, an object is often interpreted as a point in two- or three-dimensional space, where graphically, the point might be represented by a geometric figure, such as a circle, cross, or box.

For attributes, the representation depends on the type of attribute, i.e., nominal, ordinal, or continuous (interval or ratio). Ordinal and continuous attributes can be mapped to continuous, ordered graphical features such as location along the x , y , or z axes; intensity; color; or size (diameter, width, height, etc.). For categorical attributes, each category can be mapped to a distinct position, color, shape, orientation, embellishment, or column in a table. However, for nominal attributes, whose values are unordered, care should be taken when using graphical features, such as color and position that have an inherent ordering associated with their values. In other words, the graphical elements used to represent the ordinal values often have an order, but ordinal values do not.

The representation of relationships via graphical elements occurs either explicitly or implicitly. For graph data, the standard graph representation—a set of nodes with links between the nodes—is normally used. If the nodes (data objects) or links (relationships) have attributes or characteristics of their own, then this is represented graphically. To illustrate, if the nodes are cities and the links are highways, then the diameter of the nodes might represent population, while the width of the links might represent the volume of traffic.

In most cases, though, mapping objects and attributes to graphical elements implicitly maps the relationships in the data to relationships among graphical elements. To illustrate, if the data object represents a physical object that has a location, such as a city, then the relative positions of the graphical objects corresponding to the data objects tend to naturally preserve the actual

relative positions of the objects. Likewise, if there are two or three continuous attributes that are taken as the coordinates of the data points, then the resulting plot often gives considerable insight into the relationships of the attributes and the data points because data points that are visually close to each other have similar values for their attributes.

In general, it is difficult to ensure that a mapping of objects and attributes will result in the relationships being mapped to easily observed relationships among graphical elements. Indeed, this is one of the most challenging aspects of visualization. In any given set of data, there are many implicit relationships, and hence, a key challenge of visualization is to choose a technique that makes the relationships of interest easily observable.

Arrangement

As discussed earlier, the proper choice of visual representation of objects and attributes is essential for good visualization. The arrangement of items within the visual display is also crucial. We illustrate this with two examples.

Example 3.5. This example illustrates the importance of rearranging a table of data. In Table 3.5, which shows nine objects with six binary attributes, there is no clear relationship between objects and attributes, at least at first glance. If the rows and columns of this table are permuted, however, as shown in Table 3.6, then it is clear that there are really only two types of objects in the table—one that has all ones for the first three attributes and one that has only ones for the last three attributes. ■

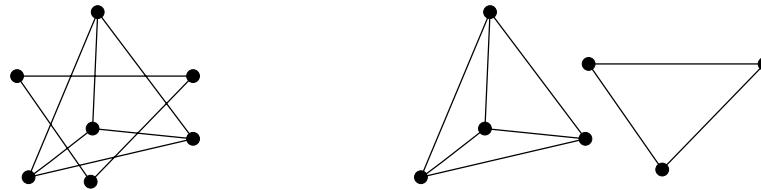
Table 3.5. A table of nine objects (rows) with six binary attributes (columns).

	1	2	3	4	5	6
1	0	1	0	1	1	0
2	1	0	1	0	0	1
3	0	1	0	1	1	0
4	1	0	1	0	0	1
5	0	1	0	1	1	0
6	1	0	1	0	0	1
7	0	1	0	1	1	0
8	1	0	1	0	0	1
9	0	1	0	1	1	0

Table 3.6. A table of nine objects (rows) with six binary attributes (columns) permuted so that the relationships of the rows and columns are clear.

	6	1	3	2	5	4
4	1	1	1	0	0	0
2	1	1	1	0	0	0
6	1	1	1	0	0	0
8	1	1	1	0	0	0
5	0	0	0	1	1	1
3	0	0	0	1	1	1
9	0	0	0	1	1	1
1	0	0	0	1	1	1
7	0	0	0	1	1	1

Example 3.6. Consider Figure 3.3(a), which shows a visualization of a graph. If the connected components of the graph are separated, as in Figure 3.3(b), then the relationships between nodes and graphs become much simpler to understand. ■



(a) Original view of a graph.

(b) Uncoupled view of connected components of the graph.

Figure 3.3. Two visualizations of a graph.

Selection

Another key concept in visualization is **selection**, which is the elimination or the de-emphasis of certain objects and attributes. Specifically, while data objects that only have a few dimensions can often be mapped to a two- or three-dimensional graphical representation in a straightforward way, there is no completely satisfactory and general approach to represent data with many attributes. Likewise, if there are many data objects, then visualizing all the objects can result in a display that is too crowded. If there are many attributes and many objects, then the situation is even more challenging.

The most common approach to handling many attributes is to choose a subset of attributes—usually two—for display. If the dimensionality is not too high, a matrix of bivariate (two-attribute) plots can be constructed for simultaneous viewing. (Figure 3.16 shows a matrix of scatter plots for the pairs of attributes of the Iris data set.) Alternatively, a visualization program can automatically show a series of two-dimensional plots, in which the sequence is user directed or based on some predefined strategy. The hope is that visualizing a collection of two-dimensional plots will provide a more complete view of the data.

The technique of selecting a pair (or small number) of attributes is a type of dimensionality reduction, and there are many more sophisticated dimensionality reduction techniques that can be employed, e.g., principal components analysis (PCA). Consult Appendices A (Linear Algebra) and B (Dimensionality Reduction) for more information.

When the number of data points is high, e.g., more than a few hundred, or if the range of the data is large, it is difficult to display enough information about each object. Some data points can obscure other data points, or a data object may not occupy enough pixels to allow its features to be clearly displayed. For example, the shape of an object cannot be used to encode a characteristic of that object if there is only one pixel available to display it. In these situations, it is useful to be able to eliminate some of the objects, either by zooming in on a particular region of the data or by taking a sample of the data points.

3.3.3 Techniques

Visualization techniques are often specialized to the type of data being analyzed. Indeed, new visualization techniques and approaches, as well as specialized variations of existing approaches, are being continuously created, typically in response to new kinds of data and visualization tasks.

Despite this specialization and the ad hoc nature of visualization, there are some generic ways to classify visualization techniques. One such classification is based on the number of attributes involved (1, 2, 3, or many) or whether the data has some special characteristic, such as a hierarchical or graph structure. Visualization methods can also be classified according to the type of attributes involved. Yet another classification is based on the type of application: scientific, statistical, or information visualization. The following discussion will use three categories: visualization of a small number of attributes, visualization of data with spatial and/or temporal attributes, and visualization of data with many attributes.

Most of the visualization techniques discussed here can be found in a wide variety of mathematical and statistical packages, some of which are freely available. There are also a number of data sets that are freely available on the World Wide Web. Readers are encouraged to try these visualization techniques as they proceed through the following sections.

Visualizing Small Numbers of Attributes

This section examines techniques for visualizing data with respect to a small number of attributes. Some of these techniques, such as histograms, give insight into the distribution of the observed values for a single attribute. Other techniques, such as scatter plots, are intended to display the relationships between the values of two attributes.

Stem and Leaf Plots Stem and leaf plots can be used to provide insight into the distribution of one-dimensional integer or continuous data. (We will assume integer data initially, and then explain how stem and leaf plots can be applied to continuous data.) For the simplest type of stem and leaf plot, we split the values into groups, where each group contains those values that are the same except for the last digit. Each group becomes a stem, while the last digits of a group are the leaves. Hence, if the values are two-digit integers, e.g., 35, 36, 42, and 51, then the stems will be the high-order digits, e.g., 3, 4, and 5, while the leaves are the low-order digits, e.g., 1, 2, 5, and 6. By plotting the stems vertically and leaves horizontally, we can provide a visual representation of the distribution of the data.

Example 3.7. The set of integers shown in Figure 3.4 is the sepal length in centimeters (multiplied by 10 to make the values integers) taken from the Iris data set. For convenience, the values have also been sorted.

The stem and leaf plot for this data is shown in Figure 3.5. Each number in Figure 3.4 is first put into one of the vertical groups—4, 5, 6, or 7—according to its ten’s digit. Its last digit is then placed to the right of the colon. Often, especially if the amount of data is larger, it is desirable to split the stems. For example, instead of placing all values whose ten’s digit is 4 in the same “bucket,” the stem 4 is repeated twice; all values 40–44 are put in the bucket corresponding to the first stem and all values 45–49 are put in the bucket corresponding to the second stem. This approach is shown in the stem and leaf plot of Figure 3.6. Other variations are also possible. ■

Histograms Stem and leaf plots are a type of **histogram**, a plot that displays the distribution of values for attributes by dividing the possible values into bins and showing the number of objects that fall into each bin. For categorical data, each value is a bin. If this results in too many values, then values are combined in some way. For continuous attributes, the range of values is divided into bins—typically, but not necessarily, of equal width—and the values in each bin are counted.

```

43 44 44 44 45 46 46 46 47 47 48 48 48 48 48 49 49 49 49 49 49 49 50
50 50 50 50 50 50 50 51 51 51 51 51 51 51 51 51 51 52 52 52 52 52 53
54 54 54 54 54 55 55 55 55 55 55 55 56 56 56 56 56 57 57 57 57 57
57 57 57 57 58 58 58 58 58 58 59 59 59 60 60 60 60 60 60 61 61 61
61 61 61 62 62 62 63 63 63 63 63 63 63 64 64 64 64 64 64 64
65 65 65 65 66 66 67 67 67 67 67 67 68 68 68 69 69 69 69 70
71 72 72 72 73 74 76 77 77 77 77 79

```

Figure 3.4. Sepal length data from the Iris data set.

```

4 : 34444566667788888999999
5 : 0000000001111111122223444445555556666667777777888888999
6 : 000001111112222333333333444444555556677777778889999
7 : 0122234677779

```

Figure 3.5. Stem and leaf plot for the sepal length from the Iris data set.

```

4 : 3444
4 : 566667788888999999
5 : 0000000001111111122223444444
5 : 5555556666667777777888888999
6 : 0000011111122223333333334444444
6 : 5555566777777778889999
7 : 0122234
7 : 677779

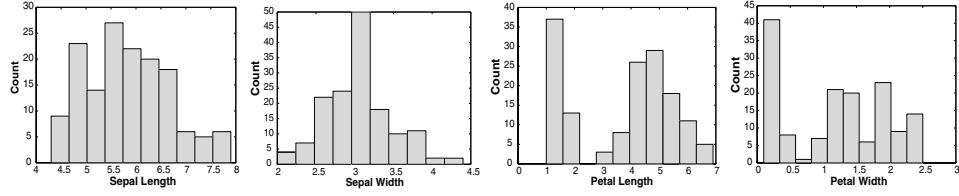
```

Figure 3.6. Stem and leaf plot for the sepal length from the Iris data set when buckets corresponding to digits are split.

Once the counts are available for each bin, a **bar plot** is constructed such that each bin is represented by one bar and the area of each bar is proportional to the number of values (objects) that fall into the corresponding range. If all intervals are of equal width, then all bars are the same width and the height of a bar is proportional to the number of values in the corresponding bin.

Example 3.8. Figure 3.7 shows histograms (with 10 bins) for sepal length, sepal width, petal length, and petal width. Since the shape of a histogram can depend on the number of bins, histograms for the same data, but with 20 bins, are shown in Figure 3.8. ■

There are variations of the histogram plot. A **relative (frequency) histogram** replaces the count by the relative frequency. However, this is just a

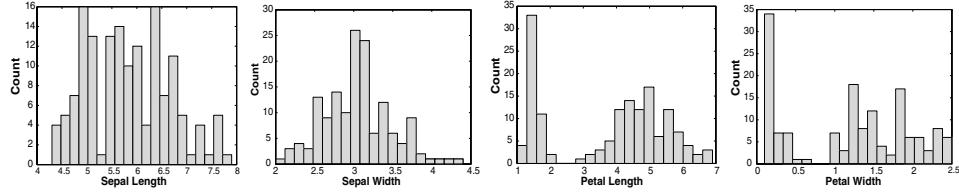


(a) Sepal length.

(b) Sepal width.

(c) Petal length.

(d) Petal width.

Figure 3.7. Histograms of four Iris attributes (10 bins).

(a) Sepal length.

(b) Sepal width.

(c) Petal length.

(d) Petal width.

Figure 3.8. Histograms of four Iris attributes (20 bins).

change in scale of the y axis, and the shape of the histogram does not change. Another common variation, especially for unordered categorical data, is the **Pareto histogram**, which is the same as a normal histogram except that the categories are sorted by count so that the count is decreasing from left to right.

Two-Dimensional Histograms Two-dimensional histograms are also possible. Each attribute is divided into intervals and the two sets of intervals define two-dimensional rectangles of values.

Example 3.9. Figure 3.9 shows a two-dimensional histogram of petal length and petal width. Because each attribute is split into three bins, there are nine rectangular two-dimensional bins. The height of each rectangular bar indicates the number of objects (flowers in this case) that fall into each bin. Most of the flowers fall into only three of the bins—those along the diagonal. It is not possible to see this by looking at the one-dimensional distributions. ■

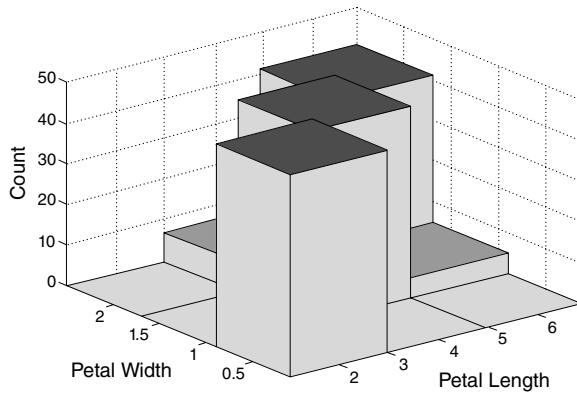


Figure 3.9. Two-dimensional histogram of petal length and width in the Iris data set.

While two-dimensional histograms can be used to discover interesting facts about how the values of two attributes co-occur, they are visually more complicated. For instance, it is easy to imagine a situation in which some of the columns are hidden by others.

Box Plots Box plots are another method for showing the distribution of the values of a single numerical attribute. Figure 3.10 shows a labeled box plot for sepal length. The lower and upper ends of the box indicate the 25th and 75th percentiles, respectively, while the line inside the box indicates the value of the 50th percentile. The top and bottom lines of the tails indicate the 10th and 90th percentiles. Outliers are shown by “+” marks. Box plots are relatively compact, and thus, many of them can be shown on the same plot. Simplified versions of the box plot, which take less space, can also be used.

Example 3.10. The box plots for the first four attributes of the Iris data set are shown in Figure 3.11. Box plots can also be used to compare how attributes vary between different classes of objects, as shown in Figure 3.12. ■

Pie Chart A pie chart is similar to a histogram, but is typically used with categorical attributes that have a relatively small number of values. Instead of showing the relative frequency of different values with the area or height of a bar, as in a histogram, a pie chart uses the relative area of a circle to indicate relative frequency. Although pie charts are common in popular articles, they

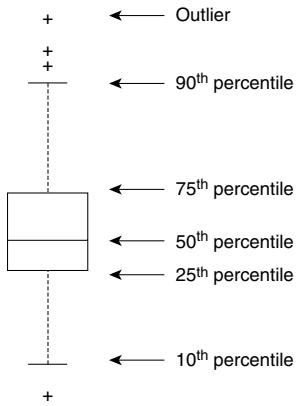


Figure 3.10. Description of box plot for sepal length.

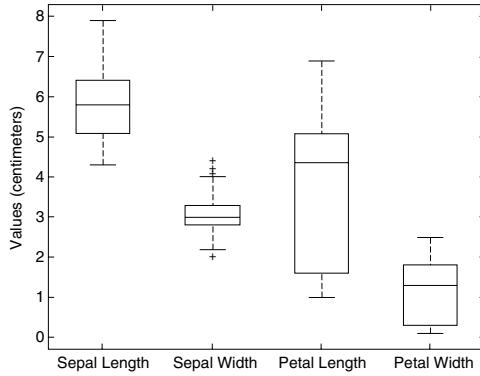
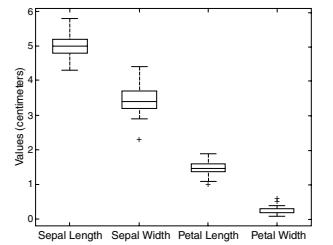
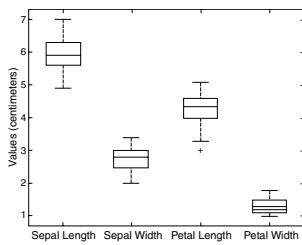


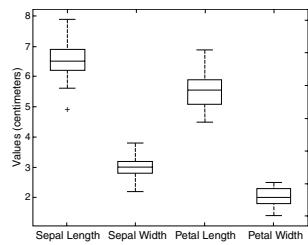
Figure 3.11. Box plot for Iris attributes.



(a) Setosa.



(b) Versicolour.



(c) Virginica.

Figure 3.12. Box plots of attributes by Iris species.

are used less frequently in technical publications because the size of relative areas can be hard to judge. Histograms are preferred for technical work.

Example 3.11. Figure 3.13 displays a pie chart that shows the distribution of Iris species in the Iris data set. In this case, all three flower types have the same frequency. ■

Percentile Plots and Empirical Cumulative Distribution Functions

A type of diagram that shows the distribution of the data more quantitatively is the plot of an empirical cumulative distribution function. While this type of plot may sound complicated, the concept is straightforward. For each value of a statistical distribution, a **cumulative distribution function** (CDF) shows

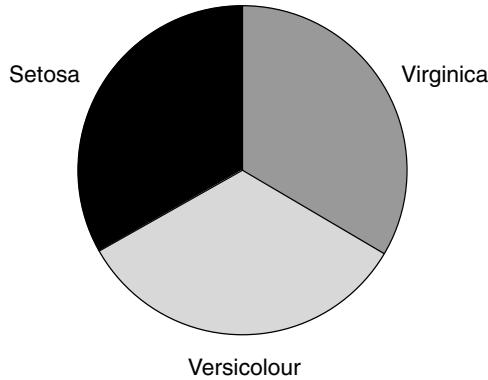


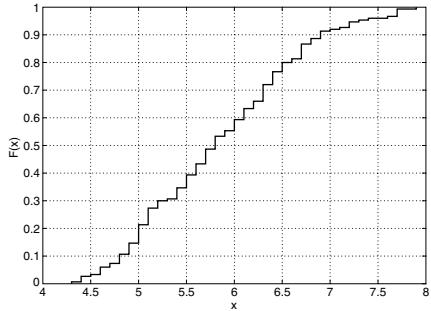
Figure 3.13. Distribution of the types of Iris flowers.

the probability that a point is less than that value. For each observed value, an **empirical cumulative distribution function** (ECDF) shows the fraction of points that are less than this value. Since the number of points is finite, the empirical cumulative distribution function is a step function.

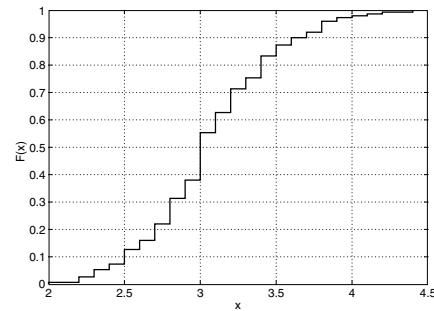
Example 3.12. Figure 3.14 shows the ECDFs of the Iris attributes. The percentiles of an attribute provide similar information. Figure 3.15 shows the **percentile plots** of the four continuous attributes of the Iris data set from Table 3.2. The reader should compare these figures with the histograms given in Figures 3.7 and 3.8. ■

Scatter Plots Most people are familiar with scatter plots to some extent, and they were used in Section 2.4.5 to illustrate linear correlation. Each data object is plotted as a point in the plane using the values of the two attributes as x and y coordinates. It is assumed that the attributes are either integer- or real-valued.

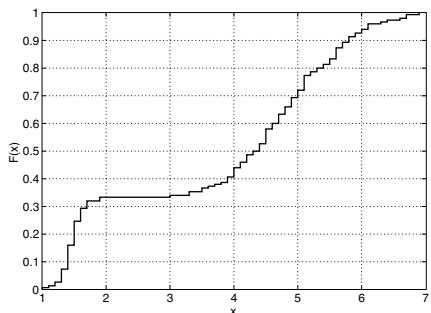
Example 3.13. Figure 3.16 shows a scatter plot for each pair of attributes of the Iris data set. The different species of Iris are indicated by different markers. The arrangement of the scatter plots of pairs of attributes in this type of tabular format, which is known as a **scatter plot matrix**, provides an organized way to examine a number of scatter plots simultaneously. ■



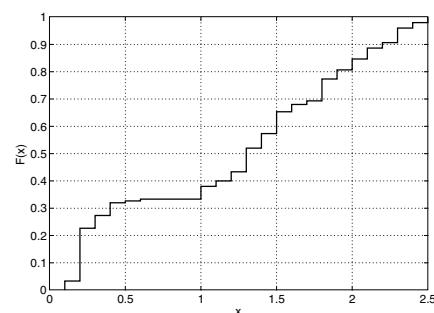
(a) Sepal Length.



(b) Sepal Width.



(c) Petal Length.



(d) Petal Width.

Figure 3.14. Empirical CDFs of four Iris attributes.

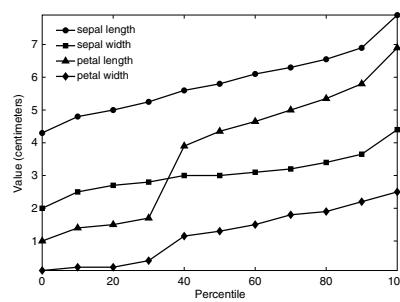


Figure 3.15. Percentile plots for sepal length, sepal width, petal length, and petal width.

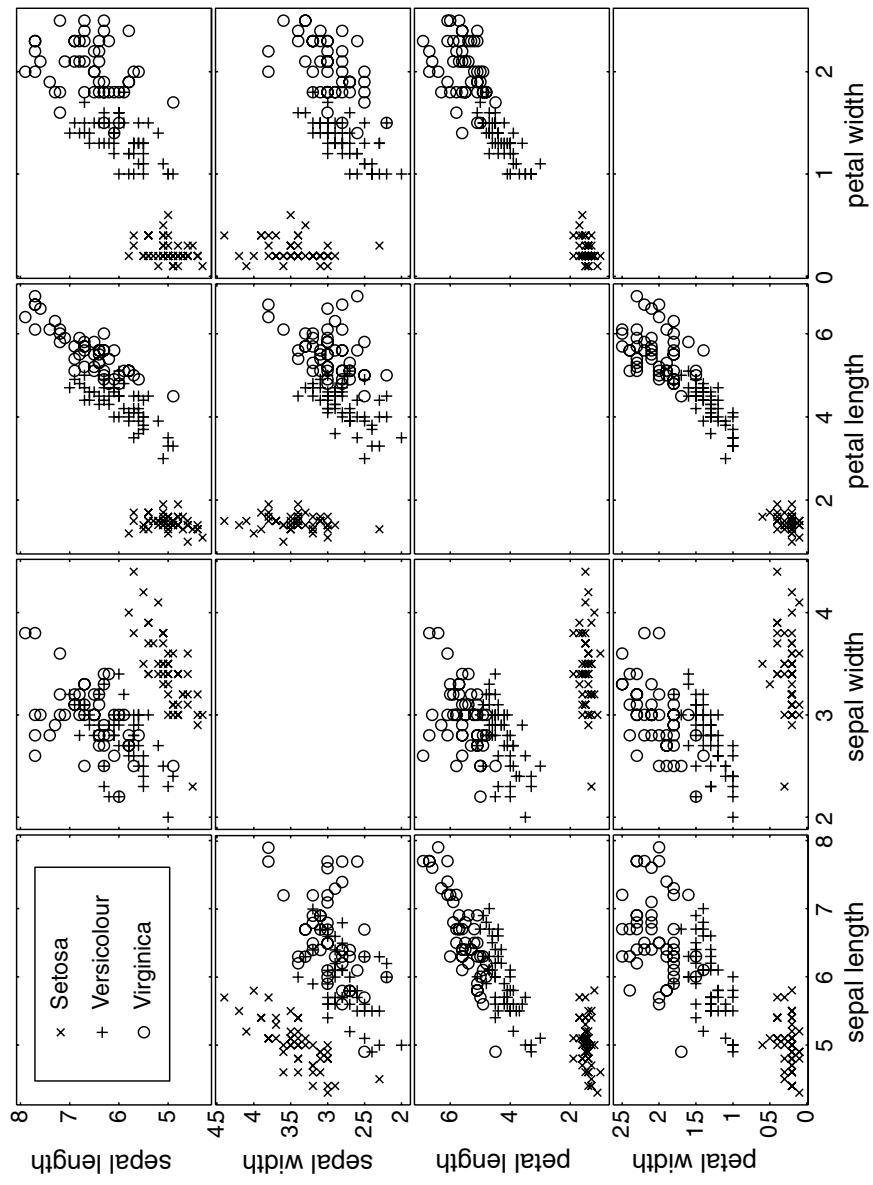


Figure 3.16. Matrix of scatter plots for the Iris data set.

There are two main uses for scatter plots. First, they graphically show the relationship between two attributes. In Section 2.4.5, we saw how scatter plots could be used to judge the degree of linear correlation. (See Figure 2.17.) Scatter plots can also be used to detect non-linear relationships, either directly or by using a scatter plot of the transformed attributes.

Second, when class labels are available, they can be used to investigate the degree to which two attributes separate the classes. If it is possible to draw a line (or a more complicated curve) that divides the plane defined by the two attributes into separate regions that contain mostly objects of one class, then it is possible to construct an accurate classifier based on the specified pair of attributes. If not, then more attributes or more sophisticated methods are needed to build a classifier. In Figure 3.16, many of the pairs of attributes (for example, petal width and petal length) provide a moderate separation of the Iris species.

Example 3.14. There are two separate approaches for displaying three attributes of a data set with a scatter plot. First, each object can be displayed according to the values of three, instead of two attributes. Figure 3.17 shows a three-dimensional scatter plot for three attributes in the Iris data set. Second, one of the attributes can be associated with some characteristic of the marker, such as its size, color, or shape. Figure 3.18 shows a plot of three attributes of the Iris data set, where one of the attributes, sepal width, is mapped to the size of the marker. ■

Extending Two- and Three-Dimensional Plots As illustrated by Figure 3.18, two- or three-dimensional plots can be extended to represent a few additional attributes. For example, scatter plots can display up to three additional attributes using color or shading, size, and shape, allowing five or six dimensions to be represented. There is a need for caution, however. As the complexity of a visual representation of the data increases, it becomes harder for the intended audience to interpret the information. There is no benefit in packing six dimensions' worth of information into a two- or three-dimensional plot, if doing so makes it impossible to understand.

Visualizing Spatio-temporal Data

Data often has spatial or temporal attributes. For instance, the data may consist of a set of observations on a spatial grid, such as observations of pressure on the surface of the Earth or the modeled temperature at various grid points in the simulation of a physical object. These observations can also be

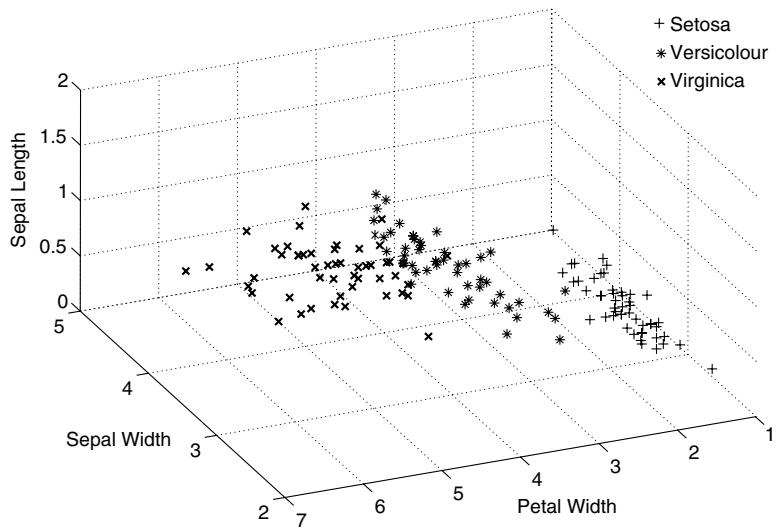


Figure 3.17. Three-dimensional scatter plot of sepal width, sepal length, and petal width.

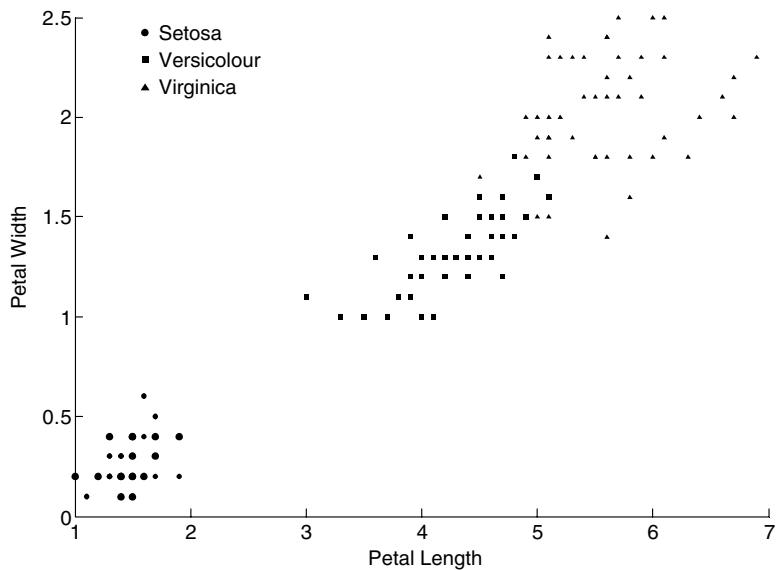


Figure 3.18. Scatter plot of petal length versus petal width, with the size of the marker indicating sepal width.

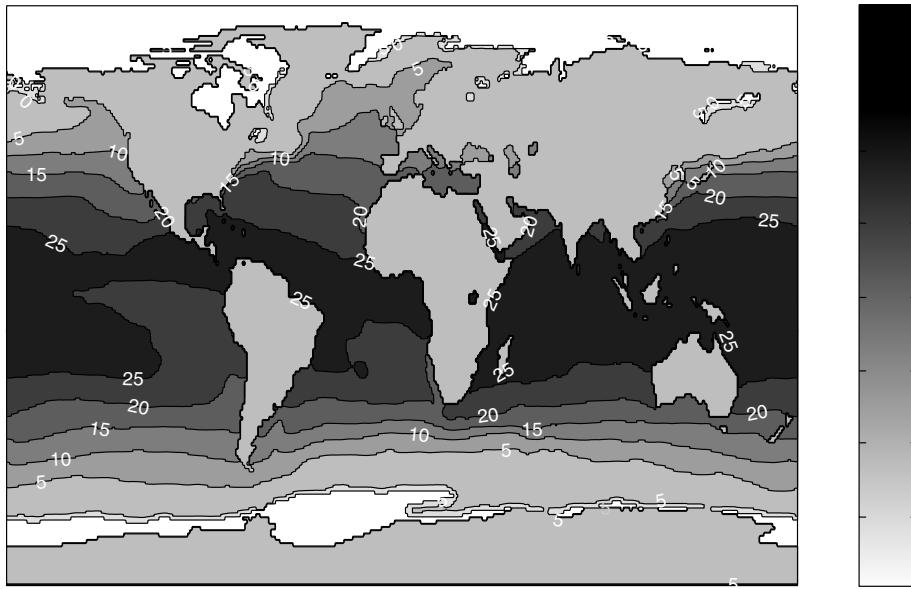


Figure 3.19. Contour plot of SST for December 1998.

made at various points in time. In addition, data may have only a temporal component, such as time series data that gives the daily prices of stocks.

Contour Plots For some three-dimensional data, two attributes specify a position in a plane, while the third has a continuous value, such as temperature or elevation. A useful visualization for such data is a **contour plot**, which breaks the plane into separate regions where the values of the third attribute (temperature, elevation) are roughly the same. A common example of a contour plot is a contour map that shows the elevation of land locations.

Example 3.15. Figure 3.19 shows a contour plot of the average sea surface temperature (SST) for December 1998. The land is arbitrarily set to have a temperature of 0°C . In many contour maps, such as that of Figure 3.19, the **contour lines** that separate two regions are labeled with the value used to separate the regions. For clarity, some of these labels have been deleted. ■

Surface Plots Like contour plots, **surface plots** use two attributes for the x and y coordinates. The third attribute is used to indicate the height above

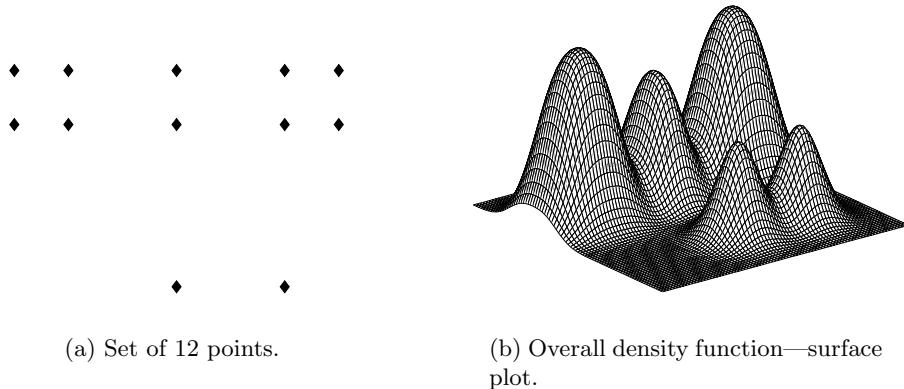


Figure 3.20. Density of a set of 12 points.

the plane defined by the first two attributes. While such graphs can be useful, they require that a value of the third attribute be defined for all combinations of values for the first two attributes, at least over some range. Also, if the surface is too irregular, then it can be difficult to see all the information, unless the plot is viewed interactively. Thus, surface plots are often used to describe mathematical functions or physical surfaces that vary in a relatively smooth manner.

Example 3.16. Figure 3.20 shows a surface plot of the density around a set of 12 points. This example is further discussed in Section 9.3.3. ■

Vector Field Plots In some data, a characteristic may have both a magnitude and a direction associated with it. For example, consider the flow of a substance or the change of density with location. In these situations, it can be useful to have a plot that displays both direction and magnitude. This type of plot is known as a **vector plot**.

Example 3.17. Figure 3.21 shows a contour plot of the density of the two smaller density peaks from Figure 3.20(b), annotated with the density gradient vectors. ■

Lower-Dimensional Slices Consider a spatio-temporal data set that records some quantity, such as temperature or pressure, at various locations over time. Such a data set has four dimensions and cannot be easily displayed by the types

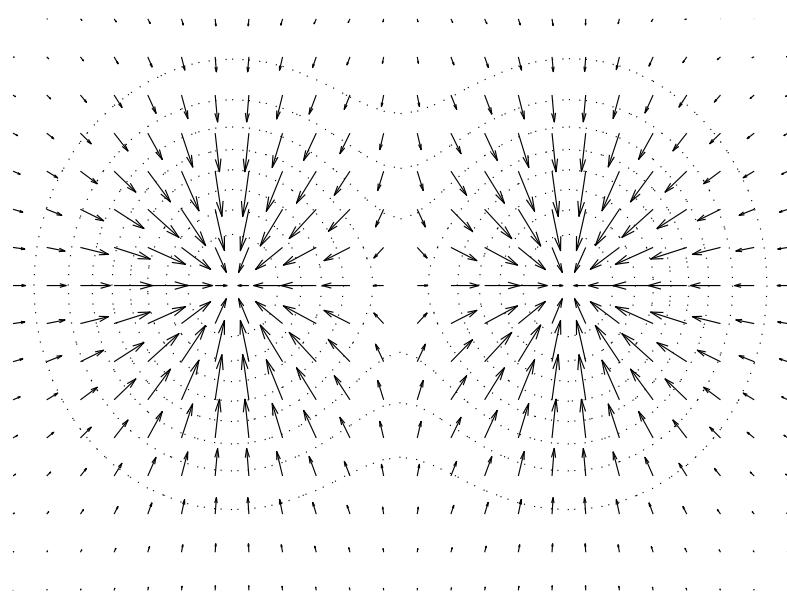


Figure 3.21. Vector plot of the gradient (change) in density for the bottom two density peaks of Figure 3.20.

of plots that we have described so far. However, separate “slices” of the data can be displayed by showing a set of plots, one for each month. By examining the change in a particular area from one month to another, it is possible to notice changes that occur, including those that may be due to seasonal factors.

Example 3.18. The underlying data set for this example consists of the average monthly sea level pressure (SLP) from 1982 to 1999 on a 2.5° by 2.5° latitude-longitude grid. The twelve monthly plots of pressure for one year are shown in Figure 3.22. In this example, we are interested in slices for a particular month in the year 1982. More generally, we can consider slices of the data along any arbitrary dimension. ■

Animation Another approach to dealing with slices of data, whether or not time is involved, is to employ animation. The idea is to display successive two-dimensional slices of the data. The human visual system is well suited to detecting visual changes and can often notice changes that might be difficult to detect in another manner. Despite the visual appeal of animation, a set of still plots, such as those of Figure 3.22, can be more useful since this type of visualization allows the information to be studied in arbitrary order and for arbitrary amounts of time.

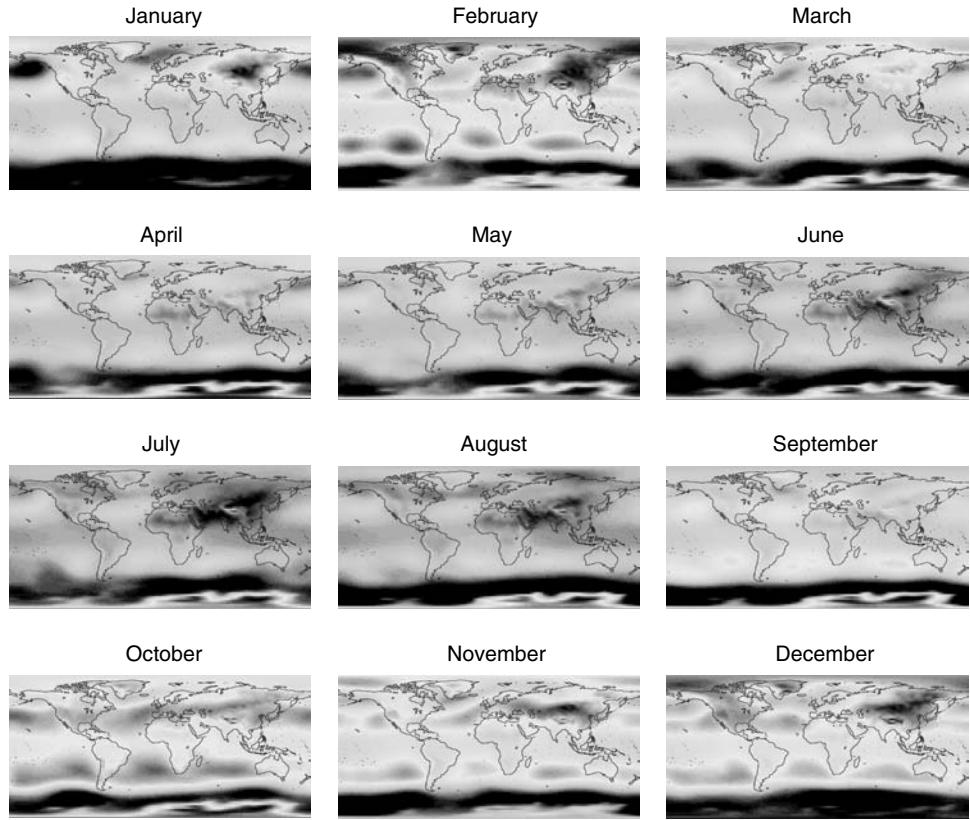


Figure 3.22. Monthly plots of sea level pressure over the 12 months of 1982.

3.3.4 Visualizing Higher-Dimensional Data

This section considers visualization techniques that can display more than the handful of dimensions that can be observed with the techniques just discussed. However, even these techniques are somewhat limited in that they only show some aspects of the data.

Matrices An image can be regarded as a rectangular array of pixels, where each pixel is characterized by its color and brightness. A data matrix is a rectangular array of values. Thus, a data matrix can be visualized as an image by associating each entry of the data matrix with a pixel in the image. The brightness or color of the pixel is determined by the value of the corresponding entry of the matrix.

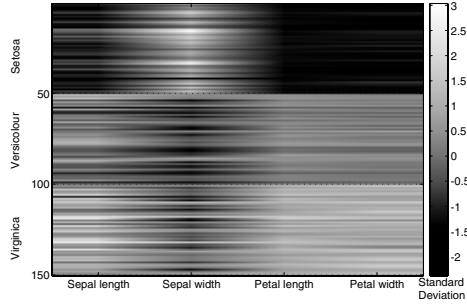


Figure 3.23. Plot of the Iris data matrix where columns have been standardized to have a mean of 0 and standard deviation of 1.

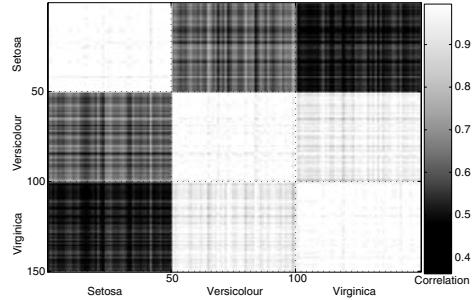


Figure 3.24. Plot of the Iris correlation matrix.

There are some important practical considerations when visualizing a data matrix. If class labels are known, then it is useful to reorder the data matrix so that all objects of a class are together. This makes it easier, for example, to detect if all objects in a class have similar attribute values for some attributes. If different attributes have different ranges, then the attributes are often standardized to have a mean of zero and a standard deviation of 1. This prevents the attribute with the largest magnitude values from visually dominating the plot.

Example 3.19. Figure 3.23 shows the standardized data matrix for the Iris data set. The first 50 rows represent Iris flowers of the species Setosa, the next 50 Versicolour, and the last 50 Virginica. The Setosa flowers have petal width and length well below the average, while the Versicolour flowers have petal width and length around average. The Virginica flowers have petal width and length above average. ■

It can also be useful to look for structure in the plot of a proximity matrix for a set of data objects. Again, it is useful to sort the rows and columns of the similarity matrix (when class labels are known) so that all the objects of a class are together. This allows a visual evaluation of the cohesiveness of each class and its separation from other classes.

Example 3.20. Figure 3.24 shows the correlation matrix for the Iris data set. Again, the rows and columns are organized so that all the flowers of a particular species are together. The flowers in each group are most similar

to each other, but Versicolour and Virginica are more similar to one another than to Setosa. ■

If class labels are not known, various techniques (matrix reordering and seriation) can be used to rearrange the rows and columns of the similarity matrix so that groups of highly similar objects and attributes are together and can be visually identified. Effectively, this is a simple kind of clustering. See Section 8.5.3 for a discussion of how a proximity matrix can be used to investigate the cluster structure of data.

Parallel Coordinates Parallel coordinates have one coordinate axis for each attribute, but the different axes are parallel to one other instead of perpendicular, as is traditional. Furthermore, an object is represented as a line instead of as a point. Specifically, the value of each attribute of an object is mapped to a point on the coordinate axis associated with that attribute, and these points are then connected to form the line that represents the object.

It might be feared that this would yield quite a mess. However, in many cases, objects tend to fall into a small number of groups, where the points in each group have similar values for their attributes. If so, and if the number of data objects is not too large, then the resulting parallel coordinates plot can reveal interesting patterns.

Example 3.21. Figure 3.25 shows a parallel coordinates plot of the four numerical attributes of the Iris data set. The lines representing objects of different classes are distinguished by their shading and the use of three different line styles—solid, dotted, and dashed. The parallel coordinates plot shows that the classes are reasonably well separated for petal width and petal length, but less well separated for sepal length and sepal width. Figure 3.25 is another parallel coordinates plot of the same data, but with a different ordering of the axes. ■

One of the drawbacks of parallel coordinates is that the detection of patterns in such a plot may depend on the order. For instance, if lines cross a lot, the picture can become confusing, and thus, it can be desirable to order the coordinate axes to obtain sequences of axes with less crossover. Compare Figure 3.26, where sepal width (the attribute that is most mixed) is at the left of the figure, to Figure 3.25, where this attribute is in the middle.

Star Coordinates and Chernoff Faces

Another approach to displaying multidimensional data is to encode objects as **glyphs** or **icons**—symbols that impart information non-verbally. More

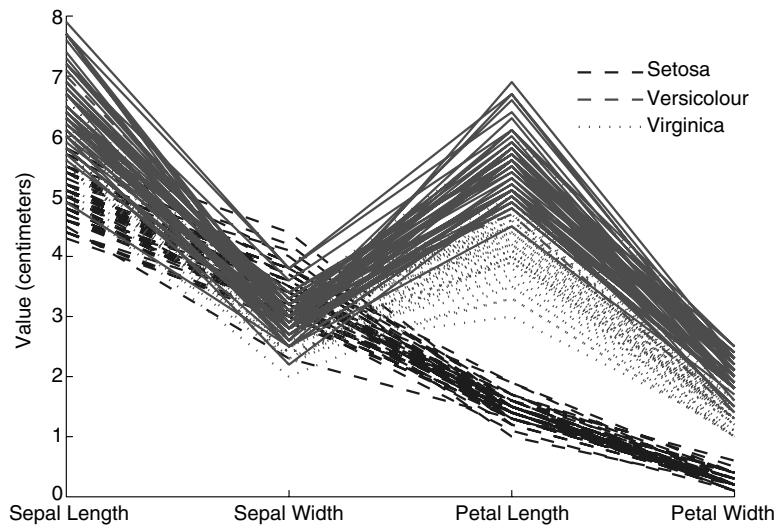


Figure 3.25. A parallel coordinates plot of the four Iris attributes.

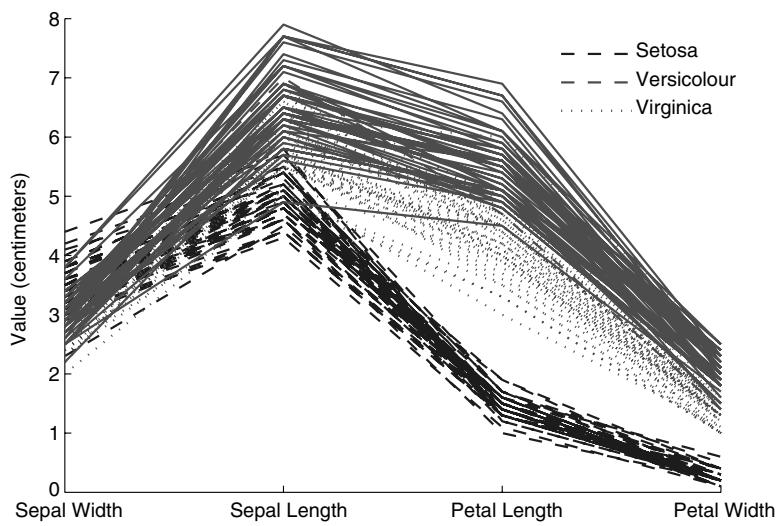


Figure 3.26. A parallel coordinates plot of the four Iris attributes with the attributes reordered to emphasize similarities and dissimilarities of groups.

specifically, each attribute of an object is mapped to a particular feature of a glyph, so that the value of the attribute determines the exact nature of the feature. Thus, at a glance, we can distinguish how two objects differ.

Star coordinates are one example of this approach. This technique uses one axis for each attribute. These axes all radiate from a center point, like the spokes of a wheel, and are evenly spaced. Typically, all the attribute values are mapped to the range [0,1].

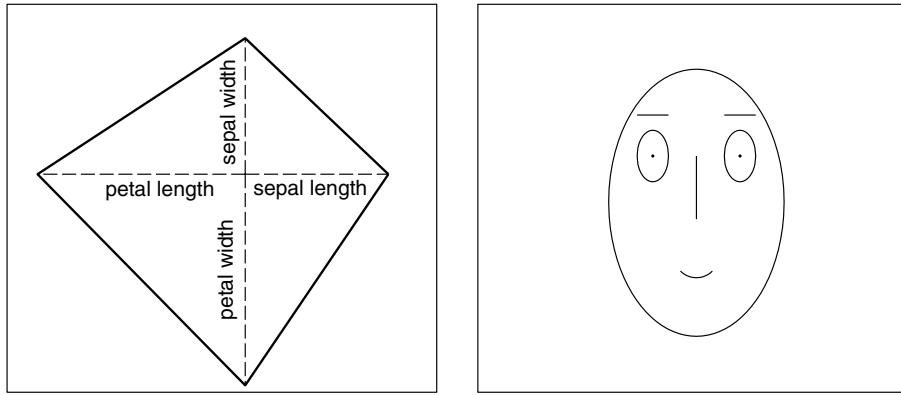
An object is mapped onto this star-shaped set of axes using the following process: Each attribute value of the object is converted to a fraction that represents its distance between the minimum and maximum values of the attribute. This fraction is mapped to a point on the axis corresponding to this attribute. Each point is connected with a line segment to the point on the axis preceding or following its own axis; this forms a polygon. The size and shape of this polygon gives a visual description of the attribute values of the object. For ease of interpretation, a separate set of axes is used for each object. In other words, each object is mapped to a polygon. An example of a star coordinates plot of flower 150 is given in Figure 3.27(a).

It is also possible to map the values of features to those of more familiar objects, such as faces. This technique is named **Chernoff faces** for its creator, Herman Chernoff. In this technique, each attribute is associated with a specific feature of a face, and the attribute value is used to determine the way that the facial feature is expressed. Thus, the shape of the face may become more elongated as the value of the corresponding data feature increases. An example of a Chernoff face for flower 150 is given in Figure 3.27(b).

The program that we used to make this face mapped the features to the four features listed below. Other features of the face, such as width between the eyes and length of the mouth, are given default values.

Data Feature	Facial Feature
sepal length	size of face
sepal width	forehead/jaw relative arc length
petal length	shape of forehead
petal width	shape of jaw

Example 3.22. A more extensive illustration of these two approaches to viewing multidimensional data is provided by Figures 3.28 and 3.29, which shows the star and face plots, respectively, of 15 flowers from the Iris data set. The first 5 flowers are of species Setosa, the second 5 are Versicolour, and the last 5 are Virginica. ■



(a) Star graph of Iris 150.

(b) Chernoff face of Iris 150.

Figure 3.27. Star coordinates graph and Chernoff face of the 150th flower of the Iris data set.

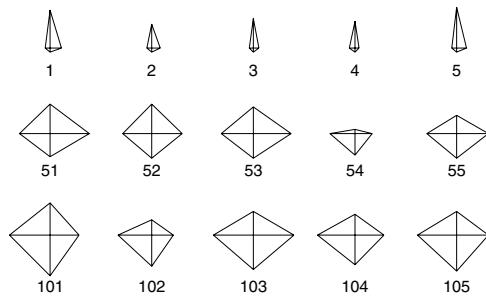


Figure 3.28. Plot of 15 Iris flowers using star coordinates.

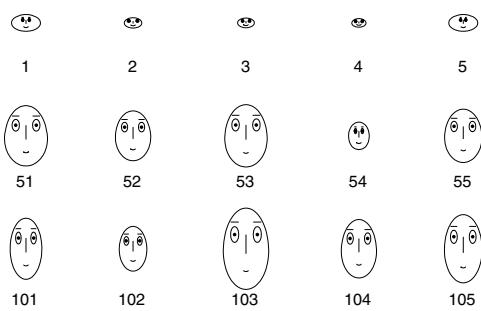


Figure 3.29. A plot of 15 Iris flowers using Chernoff faces.

Despite the visual appeal of these sorts of diagrams, they do not scale well, and thus, they are of limited use for many data mining problems. Nonetheless, they may still be of use as a means to quickly compare small sets of objects that have been selected by other techniques.

3.3.5 Do's and Don'ts

To conclude this section on visualization, we provide a short list of visualization do's and don'ts. While these guidelines incorporate a lot of visualization wisdom, they should not be followed blindly. As always, guidelines are no substitute for thoughtful consideration of the problem at hand.

ACCENT Principles The following are the *ACCENT* principles for effective graphical display put forth by D. A. Burn (as adapted by Michael Friendly):

Apprehension Ability to correctly perceive relations among variables. Does the graph maximize apprehension of the relations among variables?

Clarity Ability to visually distinguish all the elements of a graph. Are the most important elements or relations visually most prominent?

Consistency Ability to interpret a graph based on similarity to previous graphs. Are the elements, symbol shapes, and colors consistent with their use in previous graphs?

Efficiency Ability to portray a possibly complex relation in as simple a way as possible. Are the elements of the graph economically used? Is the graph easy to interpret?

Necessity The need for the graph, and the graphical elements. Is the graph a more useful way to represent the data than alternatives (table, text)? Are all the graph elements necessary to convey the relations?

Truthfulness Ability to determine the true value represented by any graphical element by its magnitude relative to the implicit or explicit scale. Are the graph elements accurately positioned and scaled?

Tufte's Guidelines Edward R. Tufte has also enumerated the following principles for graphical excellence:

- Graphical excellence is the well-designed presentation of interesting data—a matter of *substance*, of *statistics*, and of *design*.
- Graphical excellence consists of complex ideas communicated with clarity, precision, and efficiency.
- Graphical excellence is that which gives to the viewer the greatest number of ideas in the shortest time with the least ink in the smallest space.
- Graphical excellence is nearly always multivariate.
- And graphical excellence requires telling the truth about the data.

3.4 OLAP and Multidimensional Data Analysis

In this section, we investigate the techniques and insights that come from viewing data sets as multidimensional arrays. A number of database systems support such a viewpoint, most notably, On-Line Analytical Processing (OLAP) systems. Indeed, some of the terminology and capabilities of OLAP systems have made their way into spreadsheet programs that are used by millions of people. OLAP systems also have a strong focus on the interactive analysis of data and typically provide extensive capabilities for visualizing the data and generating summary statistics. For these reasons, our approach to multidimensional data analysis will be based on the terminology and concepts common to OLAP systems.

3.4.1 Representing Iris Data as a Multidimensional Array

Most data sets can be represented as a table, where each row is an object and each column is an attribute. In many cases, it is also possible to view the data as a multidimensional array. We illustrate this approach by representing the Iris data set as a multidimensional array.

Table 3.7 was created by discretizing the petal length and petal width attributes to have values of *low*, *medium*, and *high* and then counting the number of flowers from the Iris data set that have particular combinations of petal width, petal length, and species type. (For petal width, the categories *low*, *medium*, and *high* correspond to the intervals $[0, 0.75]$, $[0.75, 1.75]$, $[1.75, \infty)$, respectively. For petal length, the categories *low*, *medium*, and *high* correspond to the intervals $[0, 2.5]$, $[2.5, 5]$, $[5, \infty)$, respectively.)

Table 3.7. Number of flowers having a particular combination of petal width, petal length, and species type.

Petal Length	Petal Width	Species Type	Count
low	low	Setosa	46
low	medium	Setosa	2
medium	low	Setosa	2
medium	medium	Versicolour	43
medium	high	Versicolour	3
medium	high	Virginica	3
high	medium	Versicolour	2
high	medium	Virginica	3
high	high	Versicolour	2
high	high	Virginica	44

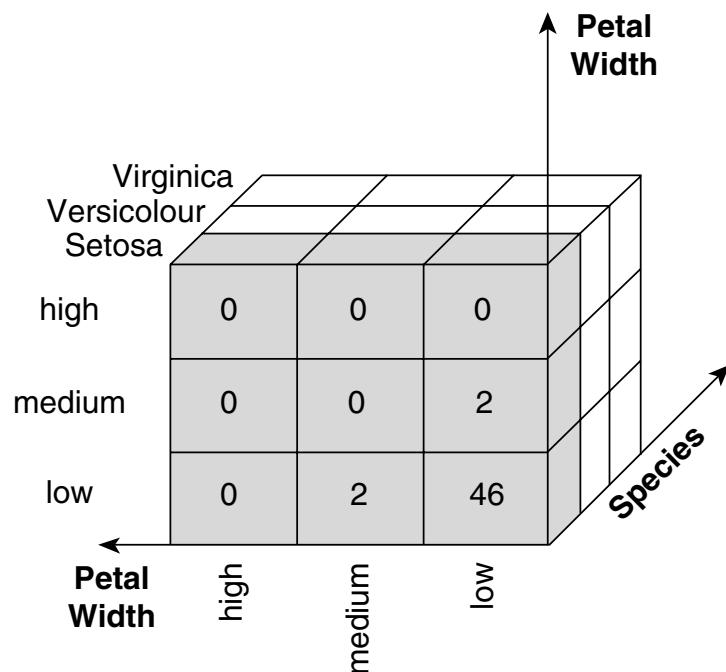


Figure 3.30. A multidimensional data representation for the Iris data set.

Table 3.8. Cross-tabulation of flowers according to petal length and width for flowers of the Setosa species.

		Width		
		low	medium	high
Length	low	46	2	0
	medium	2	0	0
	high	0	0	0

Table 3.9. Cross-tabulation of flowers according to petal length and width for flowers of the Versicolour species.

		Width		
		low	medium	high
Length	low	0	0	0
	medium	0	43	3
	high	0	2	2

Table 3.10. Cross-tabulation of flowers according to petal length and width for flowers of the Virginica species.

		Width		
		low	medium	high
Length	low	0	0	0
	medium	0	0	3
	high	0	3	44

Empty combinations—those combinations that do not correspond to at least one flower—are not shown.

The data can be organized as a multidimensional array with three dimensions corresponding to petal width, petal length, and species type, as illustrated in Figure 3.30. For clarity, slices of this array are shown as a set of three two-dimensional tables, one for each species—see Tables 3.8, 3.9, and 3.10. The information contained in both Table 3.7 and Figure 3.30 is the same. However, in the multidimensional representation shown in Figure 3.30 (and Tables 3.8, 3.9, and 3.10), the values of the attributes—petal width, petal length, and species type—are array indices.

What is important are the insights can be gained by looking at data from a multidimensional viewpoint. Tables 3.8, 3.9, and 3.10 show that each species of Iris is characterized by a different combination of values of petal length and width. Setosa flowers have low width and length, Versicolour flowers have medium width and length, and Virginica flowers have high width and length.

3.4.2 Multidimensional Data: The General Case

The previous section gave a specific example of using a multidimensional approach to represent and analyze a familiar data set. Here we describe the general approach in more detail.

The starting point is usually a tabular representation of the data, such as that of Table 3.7, which is called a **fact table**. Two steps are necessary in order to represent data as a multidimensional array: identification of the dimensions and identification of an attribute that is the focus of the analysis. The dimensions are categorical attributes or, as in the previous example, continuous attributes that have been converted to categorical attributes. The values of an attribute serve as indices into the array for the dimension corresponding to the attribute, and the number of attribute values is the size of that dimension. In the previous example, each attribute had three possible values, and thus, each dimension was of size three and could be indexed by three values. This produced a $3 \times 3 \times 3$ multidimensional array.

Each combination of attribute values (one value for each different attribute) defines a cell of the multidimensional array. To illustrate using the previous example, if petal length = *low*, petal width = *medium*, and species = Setosa, a specific cell containing the value 2 is identified. That is, there are only two flowers in the data set that have the specified attribute values. Notice that each row (object) of the data set in Table 3.7 corresponds to a cell in the multidimensional array.

The contents of each cell represents the value of a **target quantity** (target variable or attribute) that we are interested in analyzing. In the Iris example, the target quantity is the *number of flowers* whose petal width and length fall within certain limits. The target attribute is quantitative because a key goal of multidimensional data analysis is to look aggregate quantities, such as totals or averages.

The following summarizes the procedure for creating a multidimensional data representation from a data set represented in tabular form. First, identify the categorical attributes to be used as the dimensions and a quantitative attribute to be used as the target of the analysis. Each row (object) in the table is mapped to a cell of the multidimensional array. The indices of the cell are specified by the values of the attributes that were selected as dimensions, while the value of the cell is the value of the target attribute. Cells not defined by the data are assumed to have a value of 0.

Example 3.23. To further illustrate the ideas just discussed, we present a more traditional example involving the sale of products. The fact table for this example is given by Table 3.11. The dimensions of the multidimensional representation are the *product ID*, *location*, and *date* attributes, while the target attribute is the *revenue*. Figure 3.31 shows the multidimensional representation of this data set. This larger and more complicated data set will be used to illustrate additional concepts of multidimensional data analysis. ■

3.4.3 Analyzing Multidimensional Data

In this section, we describe different multidimensional analysis techniques. In particular, we discuss the creation of data cubes, and related operations, such as slicing, dicing, dimensionality reduction, roll-up, and drill down.

Data Cubes: Computing Aggregate Quantities

A key motivation for taking a multidimensional viewpoint of data is the importance of aggregating data in various ways. In the sales example, we might wish to find the total sales revenue for a specific year and a specific product. Or we might wish to see the yearly sales revenue for each location across all products. Computing aggregate totals involves fixing specific values for some of the attributes that are being used as dimensions and then summing over all possible values for the attributes that make up the remaining dimensions. There are other types of aggregate quantities that are also of interest, but for simplicity, this discussion will use totals (sums).

Table 3.12 shows the result of summing over all locations for various combinations of date and product. For simplicity, assume that all the dates are within one year. If there are 365 days in a year and 1000 products, then Table 3.12 has 365,000 entries (totals), one for each product-data pair. We could also specify the store location and date and sum over products, or specify the location and product and sum over all dates.

Table 3.13 shows the **marginal totals** of Table 3.12. These totals are the result of further summing over either dates or products. In Table 3.13, the total sales revenue due to product 1, which is obtained by summing across row 1 (over all dates), is \$370,000. The total sales revenue on January 1, 2004, which is obtained by summing down column 1 (over all products), is \$527,362. The total sales revenue, which is obtained by summing over all rows and columns (all times and products) is \$227,352,127. All of these totals are for all locations because the entries of Table 3.13 include all locations.

A key point of this example is that there are a number of different totals (aggregates) that can be computed for a multidimensional array, depending on how many attributes we sum over. Assume that there are n dimensions and that the i^{th} dimension (attribute) has s_i possible values. There are n different ways to sum only over a single attribute. If we sum over dimension j , then we obtain $s_1 * \dots * s_{j-1} * s_{j+1} * \dots * s_n$ totals, one for each possible combination of attribute values of the $n - 1$ other attributes (dimensions). The totals that result from summing over one attribute form a multidimensional array of $n - 1$ dimensions and there are n such arrays of totals. In the sales example, there

Table 3.11. Sales revenue of products (in dollars) for various locations and times.

Product ID	Location	Date	Revenue
:	:	:	:
1	Minneapolis	Oct. 18, 2004	\$250
1	Chicago	Oct. 18, 2004	\$79
:	:	:	:
1	Paris	Oct. 18, 2004	301
:	:	:	:
27	Minneapolis	Oct. 18, 2004	\$2,321
27	Chicago	Oct. 18, 2004	\$3,278
:	:	:	:
27	Paris	Oct. 18, 2004	\$1,325
:	:	:	:

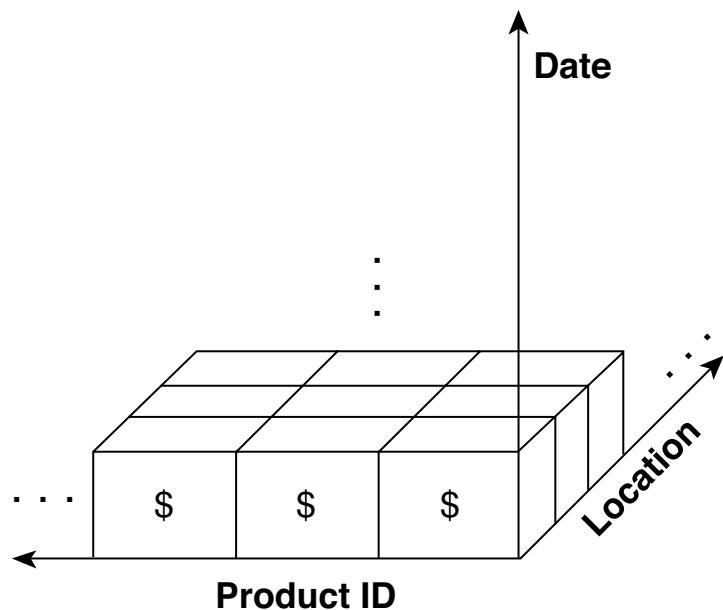


Figure 3.31. Multidimensional data representation for sales data.

Table 3.12. Totals that result from summing over all locations for a fixed time and product.

	date			
	Jan 1, 2004	Jan 2, 2004	...	Dec 31, 2004
1	\$1,001	\$987	...	\$891
:	:			:
27	\$10,265	\$10,225	...	\$9,325
:	:			:

Table 3.13. Table 3.12 with marginal totals.

	date				
	Jan 1, 2004	Jan 2, 2004	...	Dec 31, 2004	total
1	\$1,001	\$987	...	\$891	\$370,000
:	:			:	:
27	\$10,265	\$10,225	...	\$9,325	\$3,800,020
:	:			:	:
total	\$527,362	\$532,953	...	\$631,221	\$227,352,127

are three sets of totals that result from summing over only one dimension and each set of totals can be displayed as a two-dimensional table.

If we sum over two dimensions (perhaps starting with one of the arrays of totals obtained by summing over one dimension), then we will obtain a multidimensional array of totals with $n - 2$ dimensions. There will be $\binom{n}{2}$ distinct arrays of such totals. For the sales examples, there will be $\binom{3}{2} = 3$ arrays of totals that result from summing over location and product, location and time, or product and time. In general, summing over k dimensions yields $\binom{n}{k}$ arrays of totals, each with dimension $n - k$.

A multidimensional representation of the data, together with all possible totals (aggregates), is known as a **data cube**. Despite the name, the size of each dimension—the number of attribute values—does not need to be equal. Also, a data cube may have either more or fewer than three dimensions. More importantly, a data cube is a generalization of what is known in statistical terminology as a **cross-tabulation**. If marginal totals were added, Tables 3.8, 3.9, or 3.10 would be typical examples of cross tabulations.

Dimensionality Reduction and Pivoting

The aggregation described in the last section can be viewed as a form of **dimensionality reduction**. Specifically, the j^{th} dimension is eliminated by summing over it. Conceptually, this collapses each “column” of cells in the j^{th} dimension into a single cell. For both the sales and Iris examples, aggregating over one dimension reduces the dimensionality of the data from 3 to 2. If s_j is the number of possible values of the j^{th} dimension, the number of cells is reduced by a factor of s_j . Exercise 17 on page 143 asks the reader to explore the difference between this type of dimensionality reduction and that of PCA.

Pivoting refers to aggregating over all dimensions except two. The result is a two-dimensional cross tabulation with the two specified dimensions as the only remaining dimensions. Table 3.13 is an example of pivoting on date and product.

Slicing and Dicing

These two colorful names refer to rather straightforward operations. **Slicing** is selecting a group of cells from the entire multidimensional array by specifying a specific value for one or more dimensions. Tables 3.8, 3.9, and 3.10 are three slices from the Iris set that were obtained by specifying three separate values for the species dimension. **Dicing** involves selecting a subset of cells by specifying a range of attribute values. This is equivalent to defining a subarray from the complete array. In practice, both operations can also be accompanied by aggregation over some dimensions.

Roll-Up and Drill-Down

In Chapter 2, attribute values were regarded as being “atomic” in some sense. However, this is not always the case. In particular, each date has a number of properties associated with it such as the year, month, and week. The data can also be identified as belonging to a particular business quarter, or if the application relates to education, a school quarter or semester. A location also has various properties: continent, country, state (province, etc.), and city. Products can also be divided into various categories, such as clothing, electronics, and furniture.

Often these categories can be organized as a hierarchical tree or lattice. For instance, years consist of months or weeks, both of which consist of days. Locations can be divided into nations, which contain states (or other units of local government), which in turn contain cities. Likewise, any category

of products can be further subdivided. For example, the product category, furniture, can be subdivided into the subcategories, chairs, tables, sofas, etc.

This hierarchical structure gives rise to the roll-up and drill-down operations. To illustrate, starting with the original sales data, which is a multidimensional array with entries for each date, we can aggregate (**roll up**) the sales across all the dates in a month. Conversely, given a representation of the data where the time dimension is broken into months, we might want to split the monthly sales totals (**drill down**) into daily sales totals. Of course, this requires that the underlying sales data be available at a daily granularity.

Thus, roll-up and drill-down operations are related to aggregation. Notice, however, that they differ from the aggregation operations discussed until now in that they aggregate cells within a dimension, not across the entire dimension.

3.4.4 Final Comments on Multidimensional Data Analysis

Multidimensional data analysis, in the sense implied by OLAP and related systems, consists of viewing the data as a multidimensional array and aggregating data in order to better analyze the structure of the data. For the Iris data, the differences in petal width and length are clearly shown by such an analysis. The analysis of business data, such as sales data, can also reveal many interesting patterns, such as profitable (or unprofitable) stores or products.

As mentioned, there are various types of database systems that support the analysis of multidimensional data. Some of these systems are based on relational databases and are known as ROLAP systems. More specialized database systems that specifically employ a multidimensional data representation as their fundamental data model have also been designed. Such systems are known as MOLAP systems. In addition to these types of systems, statistical databases (SDBs) have been developed to store and analyze various types of statistical data, e.g., census and public health data, that are collected by governments or other large organizations. References to OLAP and SDBs are provided in the bibliographic notes.

3.5 Bibliographic Notes

Summary statistics are discussed in detail in most introductory statistics books, such as [92]. References for exploratory data analysis are the classic text by Tukey [104] and the book by Velleman and Hoaglin [105].

The basic visualization techniques are readily available, being an integral part of most spreadsheets (Microsoft EXCEL [95]), statistics programs (SAS

[99], SPSS [102], R [96], and S-PLUS [98]), and mathematics software (MATLAB [94] and Mathematica [93]). Most of the graphics in this chapter were generated using MATLAB. The statistics package R is freely available as an open source software package from the R project.

The literature on visualization is extensive, covering many fields and many decades. One of the classics of the field is the book by Tufte [103]. The book by Spence [101], which strongly influenced the visualization portion of this chapter, is a useful reference for information visualization—both principles and techniques. This book also provides a thorough discussion of many dynamic visualization techniques that were not covered in this chapter. Two other books on visualization that may also be of interest are those by Card et al. [87] and Fayyad et al. [89].

Finally, there is a great deal of information available about data visualization on the World Wide Web. Since Web sites come and go frequently, the best strategy is a search using “information visualization,” “data visualization,” or “statistical graphics.” However, we do want to single out for attention “The Gallery of Data Visualization,” by Friendly [90]. The ACCENT Principles for effective graphical display as stated in this chapter can be found there, or as originally presented in the article by Burn [86].

There are a variety of graphical techniques that can be used to explore whether the distribution of the data is Gaussian or some other specified distribution. Also, there are plots that display whether the observed values are statistically significant in some sense. We have not covered any of these techniques here and refer the reader to the previously mentioned statistical and mathematical packages.

Multidimensional analysis has been around in a variety of forms for some time. One of the original papers was a white paper by Codd [88], the father of relational databases. The data cube was introduced by Gray et al. [91], who described various operations for creating and manipulating data cubes within a relational database framework. A comparison of statistical databases and OLAP is given by Shoshani [100]. Specific information on OLAP can be found in documentation from database vendors and many popular books. Many database textbooks also have general discussions of OLAP, often in the context of data warehousing. For example, see the text by Ramakrishnan and Gehrke [97].

Bibliography

- [86] D. A. Burn. Designing Effective Statistical Graphs. In C. R. Rao, editor, *Handbook of Statistics 9*. Elsevier/North-Holland, Amsterdam, The Netherlands, September 1993.

- [87] S. K. Card, J. D. MacKinlay, and B. Shneiderman, editors. *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann Publishers, San Francisco, CA, January 1999.
- [88] E. F. Codd, S. B. Codd, and C. T. Smalley. Providing OLAP (On-line Analytical Processing) to User- Analysts: An IT Mandate. White Paper, E.F. Codd and Associates, 1993.
- [89] U. M. Fayyad, G. G. Grinstein, and A. Wierse, editors. *Information Visualization in Data Mining and Knowledge Discovery*. Morgan Kaufmann Publishers, San Francisco, CA, September 2001.
- [90] M. Friendly. Gallery of Data Visualization. <http://www.math.yorku.ca/SCS/Gallery/>, 2005.
- [91] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Rechert, M. Venkatrao, F. Pellow, and H. Pirahesh. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals. *Journal Data Mining and Knowledge Discovery*, 1(1): 29–53, 1997.
- [92] B. W. Lindgren. *Statistical Theory*. CRC Press, January 1993.
- [93] Mathematica 5.1. Wolfram Research, Inc. <http://www.wolfram.com/>, 2005.
- [94] MATLAB 7.0. The MathWorks, Inc. <http://www.mathworks.com>, 2005.
- [95] Microsoft Excel 2003. Microsoft, Inc. [http://www.microsoft.com/](http://www.microsoft.com), 2003.
- [96] R: A language and environment for statistical computing and graphics. The R Project for Statistical Computing. <http://www.r-project.org/>, 2005.
- [97] R. Ramakrishnan and J. Gehrke. *Database Management Systems*. McGraw-Hill, 3rd edition, August 2002.
- [98] S-PLUS. Insightful Corporation. <http://www.insightful.com>, 2005.
- [99] SAS: Statistical Analysis System. SAS Institute Inc. [http://www.sas.com/](http://www.sas.com), 2005.
- [100] A. Shoshani. OLAP and statistical databases: similarities and differences. In *Proc. of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems*, pages 185–196. ACM Press, 1997.
- [101] R. Spence. *Information Visualization*. ACM Press, New York, December 2000.
- [102] SPSS: Statistical Package for the Social Sciences. SPSS, Inc. [http://www.spss.com/](http://www.spss.com), 2005.
- [103] E. R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, Cheshire, CT, March 1986.
- [104] J. W. Tukey. *Exploratory data analysis*. Addison-Wesley, 1977.
- [105] P. Velleman and D. Hoaglin. *The ABC's of EDA: Applications, Basics, and Computing of Exploratory Data Analysis*. Duxbury, 1981.

3.6 Exercises

1. Obtain one of the data sets available at the UCI Machine Learning Repository and apply as many of the different visualization techniques described in the chapter as possible. The bibliographic notes and book Web site provide pointers to visualization software.

2. Identify at least two advantages and two disadvantages of using color to visually represent information.
3. What are the arrangement issues that arise with respect to three-dimensional plots?
4. Discuss the advantages and disadvantages of using sampling to reduce the number of data objects that need to be displayed. Would simple random sampling (without replacement) be a good approach to sampling? Why or why not?
5. Describe how you would create visualizations to display information that describes the following types of systems.
 - (a) Computer networks. Be sure to include both the static aspects of the network, such as connectivity, and the dynamic aspects, such as traffic.
 - (b) The distribution of specific plant and animal species around the world for a specific moment in time.
 - (c) The use of computer resources, such as processor time, main memory, and disk, for a set of benchmark database programs.
 - (d) The change in occupation of workers in a particular country over the last thirty years. Assume that you have yearly information about each person that also includes gender and level of education.

Be sure to address the following issues:

- **Representation.** How will you map objects, attributes, and relationships to visual elements?
 - **Arrangement.** Are there any special considerations that need to be taken into account with respect to how visual elements are displayed? Specific examples might be the choice of viewpoint, the use of transparency, or the separation of certain groups of objects.
 - **Selection.** How will you handle a large number of attributes and data objects?
6. Describe one advantage and one disadvantage of a stem and leaf plot with respect to a standard histogram.
 7. How might you address the problem that a histogram depends on the number and location of the bins?
 8. Describe how a box plot can give information about whether the value of an attribute is symmetrically distributed. What can you say about the symmetry of the distributions of the attributes shown in Figure 3.11?
 9. Compare sepal length, sepal width, petal length, and petal width, using Figure 3.12.

10. Comment on the use of a box plot to explore a data set with four attributes: age, weight, height, and income.
11. Give a possible explanation as to why most of the values of petal length and width fall in the buckets along the diagonal in Figure 3.9.
12. Use Figures 3.14 and 3.15 to identify a characteristic shared by the petal width and petal length attributes.
13. Simple line plots, such as that displayed in Figure 2.12 on page 56, which shows two time series, can be used to effectively display high-dimensional data. For example, in Figure 2.12 it is easy to tell that the frequencies of the two time series are different. What characteristic of time series allows the effective visualization of high-dimensional data?
14. Describe the types of situations that produce sparse or dense data cubes. Illustrate with examples other than those used in the book.
15. How might you extend the notion of multidimensional data analysis so that the target variable is a qualitative variable? In other words, what sorts of summary statistics or data visualizations would be of interest?
16. Construct a data cube from Table 3.14. Is this a dense or sparse data cube? If it is sparse, identify the cells that empty.

Table 3.14. Fact table for Exercise 16.

Product ID	Location ID	Number Sold
1	1	10
1	3	6
2	1	5
2	2	22

17. Discuss the differences between dimensionality reduction based on aggregation and dimensionality reduction based on techniques such as PCA and SVD.

4

Classification: Basic Concepts, Decision Trees, and Model Evaluation

Classification, which is the task of assigning objects to one of several predefined categories, is a pervasive problem that encompasses many diverse applications. Examples include detecting spam email messages based upon the message header and content, categorizing cells as malignant or benign based upon the results of MRI scans, and classifying galaxies based upon their shapes (see Figure 4.1).



(a) A spiral galaxy.



(b) An elliptical galaxy.

Figure 4.1. Classification of galaxies. The images are from the NASA website.

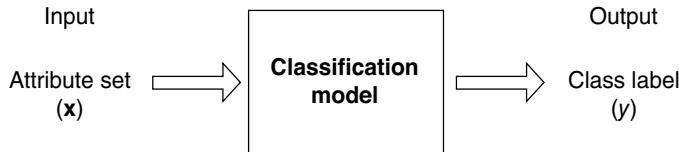


Figure 4.2. Classification as the task of mapping an input attribute set x into its class label y .

This chapter introduces the basic concepts of classification, describes some of the key issues such as model overfitting, and presents methods for evaluating and comparing the performance of a classification technique. While it focuses mainly on a technique known as decision tree induction, most of the discussion in this chapter is also applicable to other classification techniques, many of which are covered in Chapter 5.

4.1 Preliminaries

The input data for a classification task is a collection of records. Each record, also known as an instance or example, is characterized by a tuple (\mathbf{x}, y) , where \mathbf{x} is the attribute set and y is a special attribute, designated as the class label (also known as category or target attribute). Table 4.1 shows a sample data set used for classifying vertebrates into one of the following categories: mammal, bird, fish, reptile, or amphibian. The attribute set includes properties of a vertebrate such as its body temperature, skin cover, method of reproduction, ability to fly, and ability to live in water. Although the attributes presented in Table 4.1 are mostly discrete, the attribute set can also contain continuous features. The class label, on the other hand, must be a discrete attribute. This is a key characteristic that distinguishes classification from **regression**, a predictive modeling task in which y is a continuous attribute. Regression techniques are covered in Appendix D.

Definition 4.1 (Classification). Classification is the task of learning a **target function** f that maps each attribute set \mathbf{x} to one of the predefined class labels y .

The target function is also known informally as a **classification model**. A classification model is useful for the following purposes.

Descriptive Modeling A classification model can serve as an explanatory tool to distinguish between objects of different classes. For example, it would be useful—for both biologists and others—to have a descriptive model that

Table 4.1. The vertebrate data set.

Name	Body Temperature	Skin Cover	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hibernates	Class Label
human	warm-blooded	hair	yes	no	no	yes	no	mammal
python	cold-blooded	scales	no	no	no	no	yes	reptile
salmon	cold-blooded	scales	no	yes	no	no	no	fish
whale	warm-blooded	hair	yes	yes	no	no	no	mammal
frog	cold-blooded	none	no	semi	no	yes	yes	amphibian
komodo	cold-blooded	scales	no	no	no	yes	no	reptile
dragon								
bat	warm-blooded	hair	yes	no	yes	yes	yes	mammal
pigeon	warm-blooded	feathers	no	no	yes	yes	no	bird
cat	warm-blooded	fur	yes	no	no	yes	no	mammal
leopard	cold-blooded	scales	yes	yes	no	no	no	fish
shark								
turtle	cold-blooded	scales	no	semi	no	yes	no	reptile
penguin	warm-blooded	feathers	no	semi	no	yes	no	bird
porcupine	warm-blooded	quills	yes	no	no	yes	yes	mammal
eel	cold-blooded	scales	no	yes	no	no	no	fish
salamander	cold-blooded	none	no	semi	no	yes	yes	amphibian

summarizes the data shown in Table 4.1 and explains what features define a vertebrate as a mammal, reptile, bird, fish, or amphibian.

Predictive Modeling A classification model can also be used to predict the class label of unknown records. As shown in Figure 4.2, a classification model can be treated as a black box that automatically assigns a class label when presented with the attribute set of an unknown record. Suppose we are given the following characteristics of a creature known as a gila monster:

Name	Body Temperature	Skin Cover	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hibernates	Class Label
gila monster	cold-blooded	scales	no	no	no	yes	yes	?

We can use a classification model built from the data set shown in Table 4.1 to determine the class to which the creature belongs.

Classification techniques are most suited for predicting or describing data sets with binary or nominal categories. They are less effective for ordinal categories (e.g., to classify a person as a member of high-, medium-, or low-income group) because they do not consider the implicit order among the categories. Other forms of relationships, such as the subclass–superclass relationships among categories (e.g., humans and apes are primates, which in

turn, is a subclass of mammals) are also ignored. The remainder of this chapter focuses only on binary or nominal class labels.

4.2 General Approach to Solving a Classification Problem

A classification technique (or classifier) is a systematic approach to building classification models from an input data set. Examples include decision tree classifiers, rule-based classifiers, neural networks, support vector machines, and naïve Bayes classifiers. Each technique employs a **learning algorithm** to identify a model that best fits the relationship between the attribute set and class label of the input data. The model generated by a learning algorithm should both fit the input data well and correctly predict the class labels of records it has never seen before. Therefore, a key objective of the learning algorithm is to build models with good generalization capability; i.e., models that accurately predict the class labels of previously unknown records.

Figure 4.3 shows a general approach for solving classification problems. First, a **training set** consisting of records whose class labels are known must

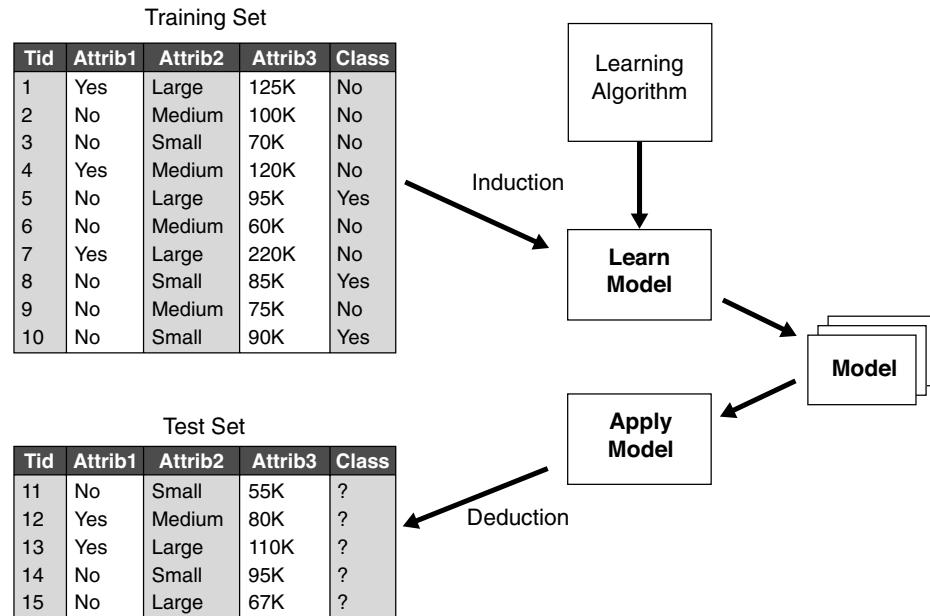


Figure 4.3. General approach for building a classification model.

Table 4.2. Confusion matrix for a 2-class problem.

		Predicted Class	
		<i>Class</i> = 1	<i>Class</i> = 0
Actual Class	<i>Class</i> = 1	f_{11}	f_{10}
	<i>Class</i> = 0	f_{01}	f_{00}

be provided. The training set is used to build a classification model, which is subsequently applied to the **test set**, which consists of records with unknown class labels.

Evaluation of the performance of a classification model is based on the counts of test records correctly and incorrectly predicted by the model. These counts are tabulated in a table known as a **confusion matrix**. Table 4.2 depicts the confusion matrix for a binary classification problem. Each entry f_{ij} in this table denotes the number of records from class i predicted to be of class j . For instance, f_{01} is the number of records from class 0 incorrectly predicted as class 1. Based on the entries in the confusion matrix, the total number of correct predictions made by the model is $(f_{11} + f_{00})$ and the total number of incorrect predictions is $(f_{10} + f_{01})$.

Although a confusion matrix provides the information needed to determine how well a classification model performs, summarizing this information with a single number would make it more convenient to compare the performance of different models. This can be done using a **performance metric** such as **accuracy**, which is defined as follows:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} = \frac{f_{11} + f_{00}}{f_{11} + f_{10} + f_{01} + f_{00}}. \quad (4.1)$$

Equivalently, the performance of a model can be expressed in terms of its **error rate**, which is given by the following equation:

$$\text{Error rate} = \frac{\text{Number of wrong predictions}}{\text{Total number of predictions}} = \frac{f_{10} + f_{01}}{f_{11} + f_{10} + f_{01} + f_{00}}. \quad (4.2)$$

Most classification algorithms seek models that attain the highest accuracy, or equivalently, the lowest error rate when applied to the test set. We will revisit the topic of model evaluation in Section 4.5.

4.3 Decision Tree Induction

This section introduces a **decision tree** classifier, which is a simple yet widely used classification technique.

4.3.1 How a Decision Tree Works

To illustrate how classification with a decision tree works, consider a simpler version of the vertebrate classification problem described in the previous section. Instead of classifying the vertebrates into five distinct groups of species, we assign them to two categories: mammals and non-mammals.

Suppose a new species is discovered by scientists. How can we tell whether it is a mammal or a non-mammal? One approach is to pose a series of questions about the characteristics of the species. The first question we may ask is whether the species is cold- or warm-blooded. If it is cold-blooded, then it is definitely not a mammal. Otherwise, it is either a bird or a mammal. In the latter case, we need to ask a follow-up question: Do the females of the species give birth to their young? Those that do give birth are definitely mammals, while those that do not are likely to be non-mammals (with the exception of egg-laying mammals such as the platypus and spiny anteater).

The previous example illustrates how we can solve a classification problem by asking a series of carefully crafted questions about the attributes of the test record. Each time we receive an answer, a follow-up question is asked until we reach a conclusion about the class label of the record. The series of questions and their possible answers can be organized in the form of a decision tree, which is a hierarchical structure consisting of nodes and directed edges. Figure 4.4 shows the decision tree for the mammal classification problem. The tree has three types of nodes:

- A **root node** that has no incoming edges and zero or more outgoing edges.
- **Internal nodes**, each of which has exactly one incoming edge and two or more outgoing edges.
- **Leaf or terminal nodes**, each of which has exactly one incoming edge and no outgoing edges.

In a decision tree, each leaf node is assigned a class label. The **non-terminal** nodes, which include the root and other internal nodes, contain attribute test conditions to separate records that have different characteristics. For example, the root node shown in Figure 4.4 uses the attribute **Body**

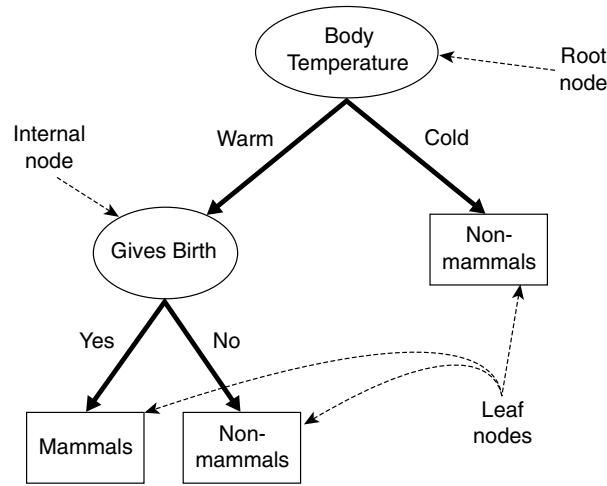


Figure 4.4. A decision tree for the mammal classification problem.

Temperature to separate warm-blooded from cold-blooded vertebrates. Since all cold-blooded vertebrates are non-mammals, a leaf node labeled Non-mammals is created as the right child of the root node. If the vertebrate is warm-blooded, a subsequent attribute, Gives Birth, is used to distinguish mammals from other warm-blooded creatures, which are mostly birds.

Classifying a test record is straightforward once a decision tree has been constructed. Starting from the root node, we apply the test condition to the record and follow the appropriate branch based on the outcome of the test. This will lead us either to another internal node, for which a new test condition is applied, or to a leaf node. The class label associated with the leaf node is then assigned to the record. As an illustration, Figure 4.5 traces the path in the decision tree that is used to predict the class label of a flamingo. The path terminates at a leaf node labeled Non-mammals.

4.3.2 How to Build a Decision Tree

In principle, there are exponentially many decision trees that can be constructed from a given set of attributes. While some of the trees are more accurate than others, finding the optimal tree is computationally infeasible because of the exponential size of the search space. Nevertheless, efficient algorithms have been developed to induce a reasonably accurate, albeit suboptimal, decision tree in a reasonable amount of time. These algorithms usually employ a greedy strategy that grows a decision tree by making a series of locally op-

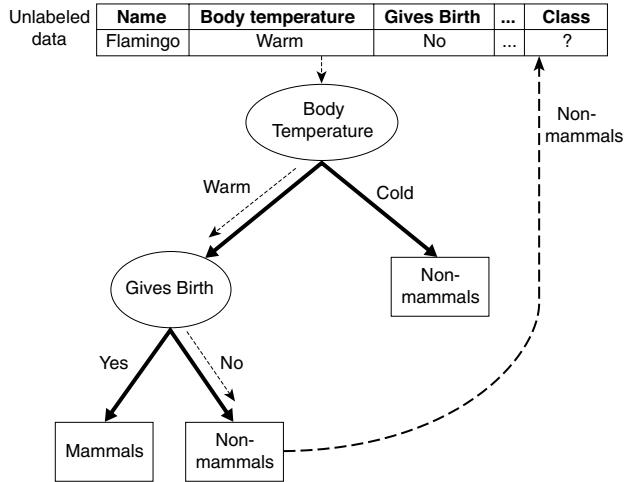


Figure 4.5. Classifying an unlabeled vertebrate. The dashed lines represent the outcomes of applying various attribute test conditions on the unlabeled vertebrate. The vertebrate is eventually assigned to the Non-mammal class.

timum decisions about which attribute to use for partitioning the data. One such algorithm is **Hunt's algorithm**, which is the basis of many existing decision tree induction algorithms, including ID3, C4.5, and CART. This section presents a high-level discussion of Hunt's algorithm and illustrates some of its design issues.

Hunt's Algorithm

In Hunt's algorithm, a decision tree is grown in a recursive fashion by partitioning the training records into successively purer subsets. Let D_t be the set of training records that are associated with node t and $y = \{y_1, y_2, \dots, y_c\}$ be the class labels. The following is a recursive definition of Hunt's algorithm.

Step 1: If all the records in D_t belong to the same class y_t , then t is a leaf node labeled as y_t .

Step 2: If D_t contains records that belong to more than one class, an **attribute test condition** is selected to partition the records into smaller subsets. A child node is created for each outcome of the test condition and the records in D_t are distributed to the children based on the outcomes. The algorithm is then recursively applied to each child node.

		binary	categorical	continuous	class
Tid	Home Owner	Marital Status	Annual Income	Defaulted Borrower	
1	Yes	Single	125K	No	
2	No	Married	100K	No	
3	No	Single	70K	No	
4	Yes	Married	120K	No	
5	No	Divorced	95K	Yes	
6	No	Married	60K	No	
7	Yes	Divorced	220K	No	
8	No	Single	85K	Yes	
9	No	Married	75K	No	
10	No	Single	90K	Yes	

Figure 4.6. Training set for predicting borrowers who will default on loan payments.

To illustrate how the algorithm works, consider the problem of predicting whether a loan applicant will repay her loan obligations or become delinquent, subsequently defaulting on her loan. A training set for this problem can be constructed by examining the records of previous borrowers. In the example shown in Figure 4.6, each record contains the personal information of a borrower along with a class label indicating whether the borrower has defaulted on loan payments.

The initial tree for the classification problem contains a single node with class label **Defaulted = No** (see Figure 4.7(a)), which means that most of the borrowers successfully repaid their loans. The tree, however, needs to be refined since the root node contains records from both classes. The records are subsequently divided into smaller subsets based on the outcomes of the **Home Owner** test condition, as shown in Figure 4.7(b). The justification for choosing this attribute test condition will be discussed later. For now, we will assume that this is the best criterion for splitting the data at this point. Hunt's algorithm is then applied recursively to each child of the root node. From the training set given in Figure 4.6, notice that all borrowers who are home owners successfully repaid their loans. The left child of the root is therefore a leaf node labeled **Defaulted = No** (see Figure 4.7(b)). For the right child, we need to continue applying the recursive step of Hunt's algorithm until all the records belong to the same class. The trees resulting from each recursive step are shown in Figures 4.7(c) and (d).

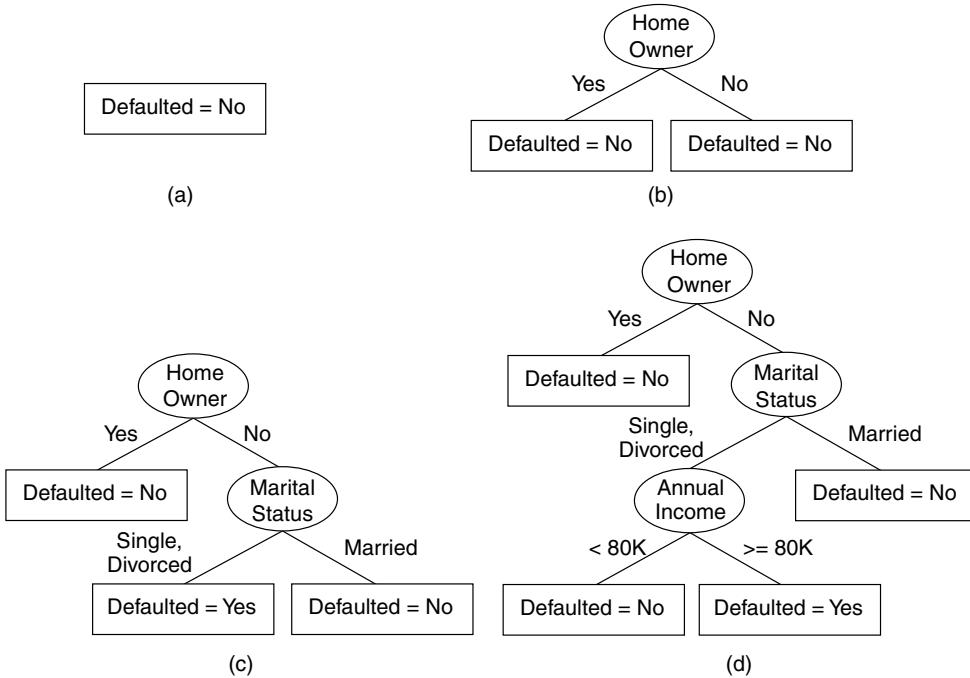


Figure 4.7. Hunt's algorithm for inducing decision trees.

Hunt's algorithm will work if every combination of attribute values is present in the training data and each combination has a unique class label. These assumptions are too stringent for use in most practical situations. Additional conditions are needed to handle the following cases:

1. It is possible for some of the child nodes created in Step 2 to be empty; i.e., there are no records associated with these nodes. This can happen if none of the training records have the combination of attribute values associated with such nodes. In this case the node is declared a leaf node with the same class label as the majority class of training records associated with its parent node.
2. In Step 2, if all the records associated with D_t have identical attribute values (except for the class label), then it is not possible to split these records any further. In this case, the node is declared a leaf node with the same class label as the majority class of training records associated with this node.

Design Issues of Decision Tree Induction

A learning algorithm for inducing decision trees must address the following two issues.

1. **How should the training records be split?** Each recursive step of the tree-growing process must select an attribute test condition to divide the records into smaller subsets. To implement this step, the algorithm must provide a method for specifying the test condition for different attribute types as well as an objective measure for evaluating the goodness of each test condition.
2. **How should the splitting procedure stop?** A stopping condition is needed to terminate the tree-growing process. A possible strategy is to continue expanding a node until either all the records belong to the same class or all the records have identical attribute values. Although both conditions are sufficient to stop any decision tree induction algorithm, other criteria can be imposed to allow the tree-growing procedure to terminate earlier. The advantages of early termination will be discussed later in Section 4.4.5.

4.3.3 Methods for Expressing Attribute Test Conditions

Decision tree induction algorithms must provide a method for expressing an attribute test condition and its corresponding outcomes for different attribute types.

Binary Attributes The test condition for a binary attribute generates two potential outcomes, as shown in Figure 4.8.

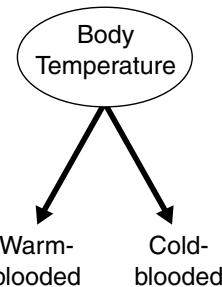


Figure 4.8. Test condition for binary attributes.

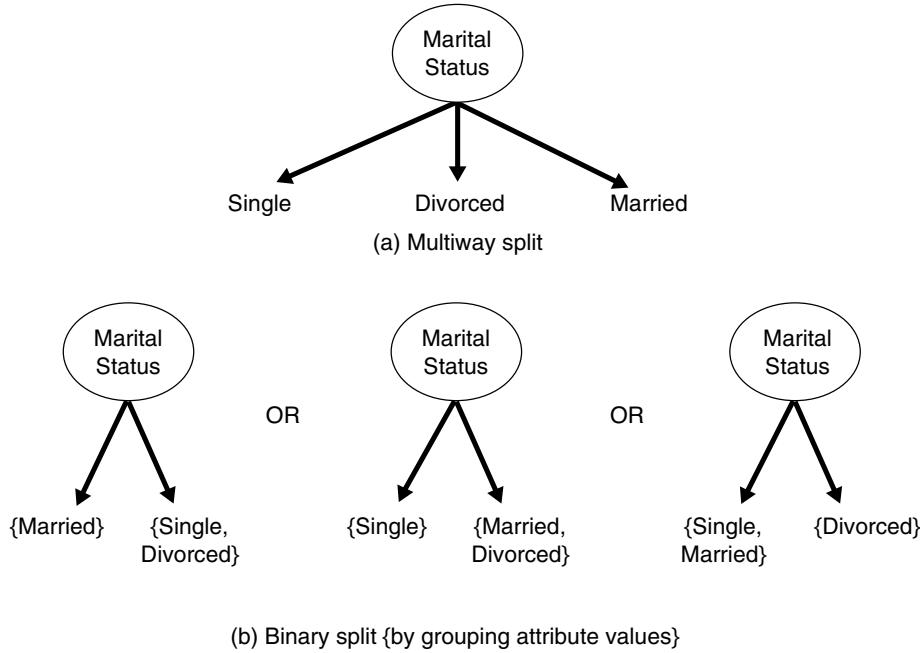


Figure 4.9. Test conditions for nominal attributes.

Nominal Attributes Since a nominal attribute can have many values, its test condition can be expressed in two ways, as shown in Figure 4.9. For a multiway split (Figure 4.9(a)), the number of outcomes depends on the number of distinct values for the corresponding attribute. For example, if an attribute such as marital status has three distinct values—single, married, or divorced—its test condition will produce a three-way split. On the other hand, some decision tree algorithms, such as CART, produce only binary splits by considering all $2^{k-1} - 1$ ways of creating a binary partition of k attribute values. Figure 4.9(b) illustrates three different ways of grouping the attribute values for marital status into two subsets.

Ordinal Attributes Ordinal attributes can also produce binary or multiway splits. Ordinal attribute values can be grouped as long as the grouping does not violate the order property of the attribute values. Figure 4.10 illustrates various ways of splitting training records based on the **Shirt Size** attribute. The groupings shown in Figures 4.10(a) and (b) preserve the order among the attribute values, whereas the grouping shown in Figure 4.10(c) violates this property because it combines the attribute values **Small** and **Large** into

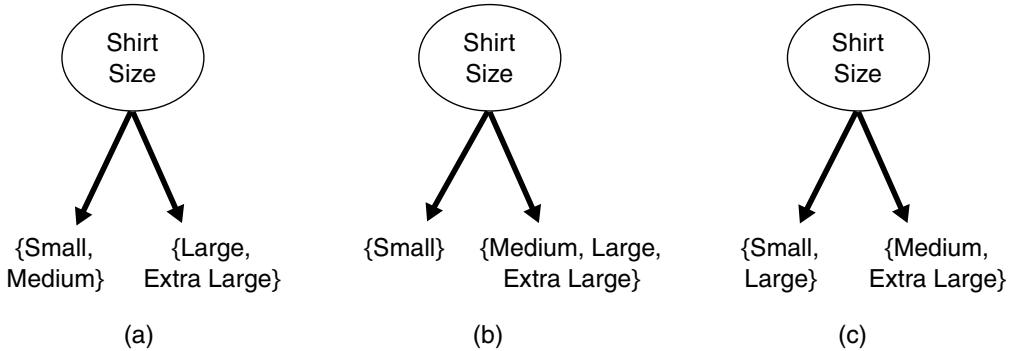


Figure 4.10. Different ways of grouping ordinal attribute values.

the same partition while **Medium** and **Extra Large** are combined into another partition.

Continuous Attributes For continuous attributes, the test condition can be expressed as a comparison test ($A < v$) or ($A \geq v$) with binary outcomes, or a range query with outcomes of the form $v_i \leq A < v_{i+1}$, for $i = 1, \dots, k$. The difference between these approaches is shown in Figure 4.11. For the binary case, the decision tree algorithm must consider all possible split positions v , and it selects the one that produces the best partition. For the multiway split, the algorithm must consider all possible ranges of continuous values. One approach is to apply the discretization strategies described in Section 2.3.6 on page 57. After discretization, a new ordinal value will be assigned to each discretized interval. Adjacent intervals can also be aggregated into wider ranges as long as the order property is preserved.

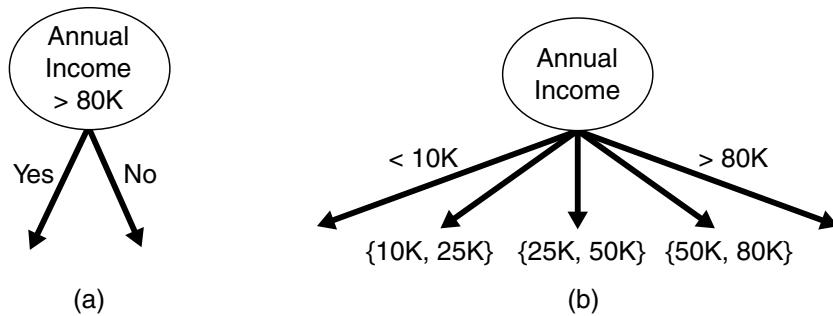


Figure 4.11. Test condition for continuous attributes.

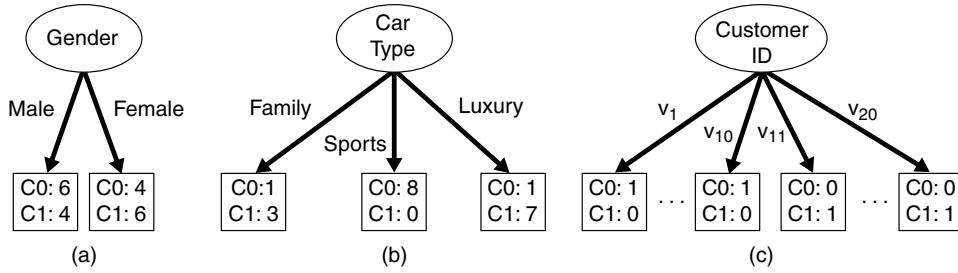


Figure 4.12. Multiway versus binary splits.

4.3.4 Measures for Selecting the Best Split

There are many measures that can be used to determine the best way to split the records. These measures are defined in terms of the class distribution of the records before and after splitting.

Let $p(i|t)$ denote the fraction of records belonging to class i at a given node t . We sometimes omit the reference to node t and express the fraction as p_i . In a two-class problem, the class distribution at any node can be written as (p_0, p_1) , where $p_1 = 1 - p_0$. To illustrate, consider the test conditions shown in Figure 4.12. The class distribution before splitting is $(0.5, 0.5)$ because there are an equal number of records from each class. If we split the data using the **Gender** attribute, then the class distributions of the child nodes are $(0.6, 0.4)$ and $(0.4, 0.6)$, respectively. Although the classes are no longer evenly distributed, the child nodes still contain records from both classes. Splitting on the second attribute, **Car Type**, will result in purer partitions.

The measures developed for selecting the best split are often based on the degree of impurity of the child nodes. The smaller the degree of impurity, the more skewed the class distribution. For example, a node with class distribution $(0, 1)$ has zero impurity, whereas a node with uniform class distribution $(0.5, 0.5)$ has the highest impurity. Examples of impurity measures include

$$\text{Entropy}(t) = - \sum_{i=0}^{c-1} p(i|t) \log_2 p(i|t), \quad (4.3)$$

$$\text{Gini}(t) = 1 - \sum_{i=0}^{c-1} [p(i|t)]^2, \quad (4.4)$$

$$\text{Classification error}(t) = 1 - \max_i [p(i|t)], \quad (4.5)$$

where c is the number of classes and $0 \log_2 0 = 0$ in entropy calculations.

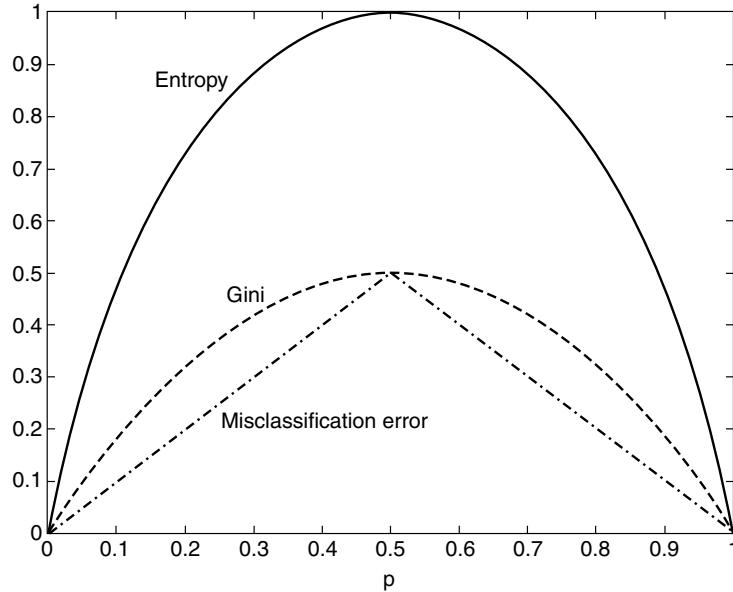


Figure 4.13. Comparison among the impurity measures for binary classification problems.

Figure 4.13 compares the values of the impurity measures for binary classification problems. p refers to the fraction of records that belong to one of the two classes. Observe that all three measures attain their maximum value when the class distribution is uniform (i.e., when $p = 0.5$). The minimum values for the measures are attained when all the records belong to the same class (i.e., when p equals 0 or 1). We next provide several examples of computing the different impurity measures.

Node N_1	Count
Class=0	0
Class=1	6

$$\begin{aligned} \text{Gini} &= 1 - (0/6)^2 - (6/6)^2 = 0 \\ \text{Entropy} &= -(0/6) \log_2(0/6) - (6/6) \log_2(6/6) = 0 \\ \text{Error} &= 1 - \max[0/6, 6/6] = 0 \end{aligned}$$

Node N_2	Count
Class=0	1
Class=1	5

$$\begin{aligned} \text{Gini} &= 1 - (1/6)^2 - (5/6)^2 = 0.278 \\ \text{Entropy} &= -(1/6) \log_2(1/6) - (5/6) \log_2(5/6) = 0.650 \\ \text{Error} &= 1 - \max[1/6, 5/6] = 0.167 \end{aligned}$$

Node N_3	Count
Class=0	3
Class=1	3

$$\begin{aligned} \text{Gini} &= 1 - (3/6)^2 - (3/6)^2 = 0.5 \\ \text{Entropy} &= -(3/6) \log_2(3/6) - (3/6) \log_2(3/6) = 1 \\ \text{Error} &= 1 - \max[3/6, 3/6] = 0.5 \end{aligned}$$

The preceding examples, along with Figure 4.13, illustrate the consistency among different impurity measures. Based on these calculations, node N_1 has the lowest impurity value, followed by N_2 and N_3 . Despite their consistency, the attribute chosen as the test condition may vary depending on the choice of impurity measure, as will be shown in Exercise 3 on page 198.

To determine how well a test condition performs, we need to compare the degree of impurity of the parent node (before splitting) with the degree of impurity of the child nodes (after splitting). The larger their difference, the better the test condition. The gain, Δ , is a criterion that can be used to determine the goodness of a split:

$$\Delta = I(\text{parent}) - \sum_{j=1}^k \frac{N(v_j)}{N} I(v_j), \quad (4.6)$$

where $I(\cdot)$ is the impurity measure of a given node, N is the total number of records at the parent node, k is the number of attribute values, and $N(v_j)$ is the number of records associated with the child node, v_j . Decision tree induction algorithms often choose a test condition that maximizes the gain Δ . Since $I(\text{parent})$ is the same for all test conditions, maximizing the gain is equivalent to minimizing the weighted average impurity measures of the child nodes. Finally, when entropy is used as the impurity measure in Equation 4.6, the difference in entropy is known as the **information gain**, Δ_{info} .

Splitting of Binary Attributes

Consider the diagram shown in Figure 4.14. Suppose there are two ways to split the data into smaller subsets. Before splitting, the Gini index is 0.5 since there are an equal number of records from both classes. If attribute A is chosen to split the data, the Gini index for node N_1 is 0.4898, and for node N_2 , it is 0.480. The weighted average of the Gini index for the descendent nodes is $(7/12) \times 0.4898 + (5/12) \times 0.480 = 0.486$. Similarly, we can show that the weighted average of the Gini index for attribute B is 0.375. Since the subsets for attribute B have a smaller Gini index, it is preferred over attribute A .

Splitting of Nominal Attributes

As previously noted, a nominal attribute can produce either binary or multi-way splits, as shown in Figure 4.15. The computation of the Gini index for a binary split is similar to that shown for determining binary attributes. For the first binary grouping of the **Car Type** attribute, the Gini index of {Sports,

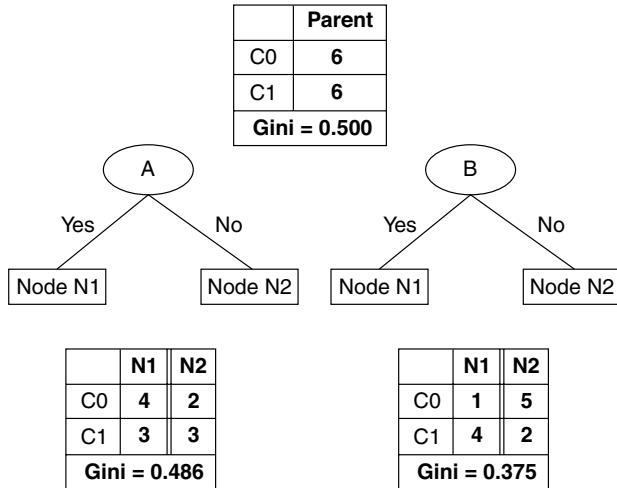


Figure 4.14. Splitting binary attributes.

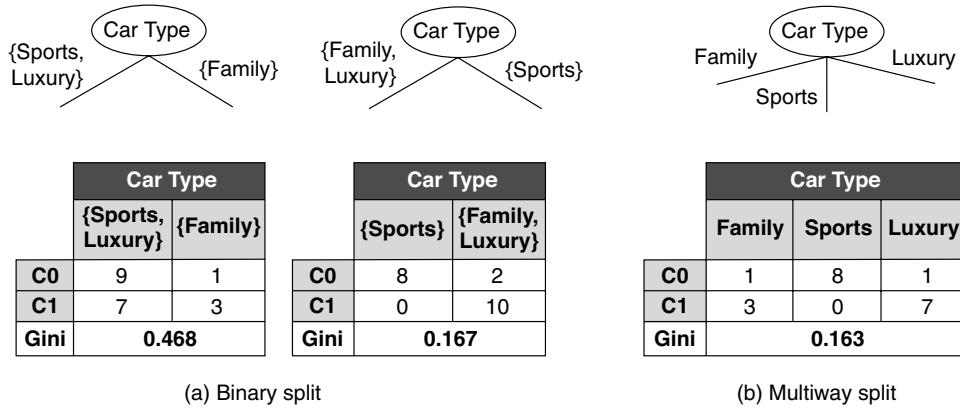


Figure 4.15. Splitting nominal attributes.

$\{Sports, Luxury\}$ is 0.4922 and the Gini index of $\{Family\}$ is 0.3750. The weighted average Gini index for the grouping is equal to

$$16/20 \times 0.4922 + 4/20 \times 0.3750 = 0.468.$$

Similarly, for the second binary grouping of $\{Sports\}$ and $\{Family, Luxury\}$, the weighted average Gini index is 0.167. The second grouping has a lower Gini index because its corresponding subsets are much purer.

Class	No	No	No	Yes	Yes	Yes	No	No	No	No	
Annual Income											
Sorted Values →	60	70	75	85	90	95	100	120	125	220	
Split Positions →	55	65	72	80	87	92	97	110	122	172	230
	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >
Yes	0 3	0 3	0 3	0 3	1 2	2 1	3 0	3 0	3 0	3 0	3 0
No	0 7	1 6	2 5	3 4	3 4	3 4	3 4	4 3	5 2	6 1	7 0
Gini	0.420	0.400	0.375	0.343	0.417	0.400	0.300	0.343	0.375	0.400	0.420

Figure 4.16. Splitting continuous attributes.

For the multiway split, the Gini index is computed for every attribute value. Since $\text{Gini}(\{\text{Family}\}) = 0.375$, $\text{Gini}(\{\text{Sports}\}) = 0$, and $\text{Gini}(\{\text{Luxury}\}) = 0.219$, the overall Gini index for the multiway split is equal to

$$4/20 \times 0.375 + 8/20 \times 0 + 8/20 \times 0.219 = 0.163.$$

The multiway split has a smaller Gini index compared to both two-way splits. This result is not surprising because the two-way split actually merges some of the outcomes of a multiway split, and thus, results in less pure subsets.

Splitting of Continuous Attributes

Consider the example shown in Figure 4.16, in which the test condition **Annual Income** $\leq v$ is used to split the training records for the loan default classification problem. A brute-force method for finding v is to consider every value of the attribute in the N records as a candidate split position. For each candidate v , the data set is scanned once to count the number of records with annual income less than or greater than v . We then compute the Gini index for each candidate and choose the one that gives the lowest value. This approach is computationally expensive because it requires $O(N)$ operations to compute the Gini index at each candidate split position. Since there are N candidates, the overall complexity of this task is $O(N^2)$. To reduce the complexity, the training records are sorted based on their annual income, a computation that requires $O(N \log N)$ time. Candidate split positions are identified by taking the midpoints between two adjacent sorted values: 55, 65, 72, and so on. However, unlike the brute-force approach, we do not have to examine all N records when evaluating the Gini index of a candidate split position.

For the first candidate, $v = 55$, none of the records has annual income less than \$55K. As a result, the Gini index for the descendent node with **Annual**

`Income < $55K` is zero. On the other hand, the number of records with annual income greater than or equal to \$55K is 3 (for class `Yes`) and 7 (for class `No`), respectively. Thus, the Gini index for this node is 0.420. The overall Gini index for this candidate split position is equal to $0 \times 0 + 1 \times 0.420 = 0.420$.

For the second candidate, $v = 65$, we can determine its class distribution by updating the distribution of the previous candidate. More specifically, the new distribution is obtained by examining the class label of the record with the lowest annual income (i.e., \$60K). Since the class label for this record is `No`, the count for class `No` is increased from 0 to 1 (for `Annual Income ≤ $65K`) and is decreased from 7 to 6 (for `Annual Income > $65K`). The distribution for class `Yes` remains unchanged. The new weighted-average Gini index for this candidate split position is 0.400.

This procedure is repeated until the Gini index values for all candidates are computed, as shown in Figure 4.16. The best split position corresponds to the one that produces the smallest Gini index, i.e., $v = 97$. This procedure is less expensive because it requires a constant amount of time to update the class distribution at each candidate split position. It can be further optimized by considering only candidate split positions located between two adjacent records with different class labels. For example, because the first three sorted records (with annual incomes \$60K, \$70K, and \$75K) have identical class labels, the best split position should not reside between \$60K and \$75K. Therefore, the candidate split positions at $v = \$55K, \$65K, \$72K, \$87K, \$92K, \$110K, \$122K, \$172K$, and $\$230K$ are ignored because they are located between two adjacent records with the same class labels. This approach allows us to reduce the number of candidate split positions from 11 to 2.

Gain Ratio

Impurity measures such as entropy and Gini index tend to favor attributes that have a large number of distinct values. Figure 4.12 shows three alternative test conditions for partitioning the data set given in Exercise 2 on page 198. Comparing the first test condition, `Gender`, with the second, `Car Type`, it is easy to see that `Car Type` seems to provide a better way of splitting the data since it produces purer descendant nodes. However, if we compare both conditions with `Customer ID`, the latter appears to produce purer partitions. Yet `Customer ID` is not a predictive attribute because its value is unique for each record. Even in a less extreme situation, a test condition that results in a large number of outcomes may not be desirable because the number of records associated with each partition is too small to enable us to make any reliable predictions.

There are two strategies for overcoming this problem. The first strategy is to restrict the test conditions to binary splits only. This strategy is employed by decision tree algorithms such as CART. Another strategy is to modify the splitting criterion to take into account the number of outcomes produced by the attribute test condition. For example, in the C4.5 decision tree algorithm, a splitting criterion known as **gain ratio** is used to determine the goodness of a split. This criterion is defined as follows:

$$\text{Gain ratio} = \frac{\Delta_{\text{info}}}{\text{Split Info}}. \quad (4.7)$$

Here, $\text{Split Info} = -\sum_{i=1}^k P(v_i) \log_2 P(v_i)$ and k is the total number of splits. For example, if each attribute value has the same number of records, then $\forall i : P(v_i) = 1/k$ and the split information would be equal to $\log_2 k$. This example suggests that if an attribute produces a large number of splits, its split information will also be large, which in turn reduces its gain ratio.

4.3.5 Algorithm for Decision Tree Induction

A skeleton decision tree induction algorithm called **TreeGrowth** is shown in Algorithm 4.1. The input to this algorithm consists of the training records E and the attribute set F . The algorithm works by recursively selecting the best attribute to split the data (Step 7) and expanding the leaf nodes of the

Algorithm 4.1 A skeleton decision tree induction algorithm.

```

TreeGrowth ( $E, F$ )
1: if stopping_cond( $E, F$ ) = true then
2:    $leaf = \text{createNode}()$ .
3:    $leaf.label = \text{Classify}(E)$ .
4:   return  $leaf$ .
5: else
6:    $root = \text{createNode}()$ .
7:    $root.test\_cond = \text{find\_best\_split}(E, F)$ .
8:   let  $V = \{v | v \text{ is a possible outcome of } root.test\_cond\}$ .
9:   for each  $v \in V$  do
10:     $E_v = \{e | root.test\_cond(e) = v \text{ and } e \in E\}$ .
11:     $child = \text{TreeGrowth}(E_v, F)$ .
12:    add  $child$  as descendent of  $root$  and label the edge  $(root \rightarrow child)$  as  $v$ .
13:   end for
14: end if
15: return  $root$ .
```

tree (Steps 11 and 12) until the stopping criterion is met (Step 1). The details of this algorithm are explained below:

1. The `createNode()` function extends the decision tree by creating a new node. A node in the decision tree has either a test condition, denoted as `node.test_cond`, or a class label, denoted as `node.label`.
2. The `find_best_split()` function determines which attribute should be selected as the test condition for splitting the training records. As previously noted, the choice of test condition depends on which impurity measure is used to determine the goodness of a split. Some widely used measures include entropy, the Gini index, and the χ^2 statistic.
3. The `Classify()` function determines the class label to be assigned to a leaf node. For each leaf node t , let $p(i|t)$ denote the fraction of training records from class i associated with the node t . In most cases, the leaf node is assigned to the class that has the majority number of training records:

$$leaf.label = \operatorname{argmax}_i p(i|t), \quad (4.8)$$

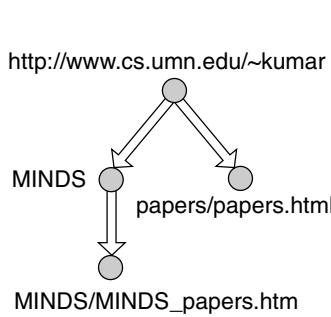
where the `argmax` operator returns the argument i that maximizes the expression $p(i|t)$. Besides providing the information needed to determine the class label of a leaf node, the fraction $p(i|t)$ can also be used to estimate the probability that a record assigned to the leaf node t belongs to class i . Sections 5.7.2 and 5.7.3 describe how such probability estimates can be used to determine the performance of a decision tree under different cost functions.

4. The `stopping_cond()` function is used to terminate the tree-growing process by testing whether all the records have either the same class label or the same attribute values. Another way to terminate the recursive function is to test whether the number of records have fallen below some minimum threshold.

After building the decision tree, a **tree-pruning** step can be performed to reduce the size of the decision tree. Decision trees that are too large are susceptible to a phenomenon known as **overfitting**. Pruning helps by trimming the branches of the initial tree in a way that improves the generalization capability of the decision tree. The issues of overfitting and tree pruning are discussed in more detail in Section 4.4.

Session	IP Address	Timestamp	Request Method	Requested Web Page	Protocol	Status	Number of Bytes	Referrer	User Agent
1	160.11.11.11	08/Aug/2004 10:15:21	GET	http://www.cs.umn.edu/~kumar	HTTP/1.1	200	6424		Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
1	160.11.11.11	08/Aug/2004 10:15:34	GET	http://www.cs.umn.edu/~kumar/MINDS	HTTP/1.1	200	41378	http://www.cs.umn.edu/~kumar	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
1	160.11.11.11	08/Aug/2004 10:15:41	GET	http://www.cs.umn.edu/~kumar/MINDS/MINDS_papers.htm	HTTP/1.1	200	1018516	http://www.cs.umn.edu/~kumar/MINDS	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
1	160.11.11.11	08/Aug/2004 10:16:11	GET	http://www.cs.umn.edu/~kumar/papers/papers.html	HTTP/1.1	200	7463	http://www.cs.umn.edu/~kumar	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
2	35.9.2.2	08/Aug/2004 10:16:15	GET	http://www.cs.umn.edu/~steinbac	HTTP/1.0	200	3149		Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7) Gecko/20040616

(a) Example of a Web server log.



(b) Graph of a Web session.

Attribute Name	Description
totalPages	Total number of pages retrieved in a Web session
ImagePages	Total number of image pages retrieved in a Web session
TotalTime	Total amount of time spent by Web site visitor
RepeatedAccess	The same page requested more than once in a Web session
ErrorRequest	Errors in requesting for Web pages
GET	Percentage of requests made using GET method
POST	Percentage of requests made using POST method
HEAD	Percentage of requests made using HEAD method
Breadth	Breadth of Web traversal
Depth	Depth of Web traversal
MultilP	Session with multiple IP addresses
MultiAgent	Session with multiple user agents

(c) Derived attributes for Web robot detection.

Figure 4.17. Input data for Web robot detection.

4.3.6 An Example: Web Robot Detection

Web usage mining is the task of applying data mining techniques to extract useful patterns from Web access logs. These patterns can reveal interesting characteristics of site visitors; e.g., people who repeatedly visit a Web site and view the same product description page are more likely to buy the product if certain incentives such as rebates or free shipping are offered.

In Web usage mining, it is important to distinguish accesses made by human users from those due to Web robots. A Web robot (also known as a Web crawler) is a software program that automatically locates and retrieves information from the Internet by following the hyperlinks embedded in Web pages. These programs are deployed by search engine portals to gather the documents necessary for indexing the Web. Web robot accesses must be discarded before applying Web mining techniques to analyze human browsing behavior.

This section describes how a decision tree classifier can be used to distinguish between accesses by human users and those by Web robots. The input data was obtained from a Web server log, a sample of which is shown in Figure 4.17(a). Each line corresponds to a single page request made by a Web client (a user or a Web robot). The fields recorded in the Web log include the IP address of the client, timestamp of the request, Web address of the requested document, size of the document, and the client's identity (via the user agent field). A Web session is a sequence of requests made by a client during a single visit to a Web site. Each Web session can be modeled as a directed graph, in which the nodes correspond to Web pages and the edges correspond to hyperlinks connecting one Web page to another. Figure 4.17(b) shows a graphical representation of the first Web session given in the Web server log.

To classify the Web sessions, features are constructed to describe the characteristics of each session. Figure 4.17(c) shows some of the features used for the Web robot detection task. Among the notable features include the **depth** and **breadth** of the traversal. Depth determines the maximum distance of a requested page, where distance is measured in terms of the number of hyperlinks away from the entry point of the Web site. For example, the home page <http://www.cs.umn.edu/~kumar> is assumed to be at depth 0, whereas http://www.cs.umn.edu/kumar/MINDS/MINDS_papers.htm is located at depth 2. Based on the Web graph shown in Figure 4.17(b), the **depth** attribute for the first session is equal to two. The **breadth** attribute measures the width of the corresponding Web graph. For example, the **breadth** of the Web session shown in Figure 4.17(b) is equal to two.

The data set for classification contains 2916 records, with equal numbers of sessions due to Web robots (class 1) and human users (class 0). 10% of the data were reserved for training while the remaining 90% were used for testing. The induced decision tree model is shown in Figure 4.18. The tree has an error rate equal to 3.8% on the training set and 5.3% on the test set.

The model suggests that Web robots can be distinguished from human users in the following way:

1. Accesses by Web robots tend to be broad but shallow, whereas accesses by human users tend to be more focused (narrow but deep).
2. Unlike human users, Web robots seldom retrieve the image pages associated with a Web document.
3. Sessions due to Web robots tend to be long and contain a large number of requested pages.

```

Decision Tree:
depth = 1:
| breadth> 7 : class 1
| breadth<= 7:
| | breadth <= 3:
| | | ImagePages> 0.375: class 0
| | | ImagePages<= 0.375:
| | | | totalPages<= 6: class 1
| | | | totalPages> 6:
| | | | | breadth <= 1: class 1
| | | | | breadth > 1: class 0
| | | width > 3:
| | | | MultilP = 0:
| | | | | ImagePages<= 0.1333: class 1
| | | | | ImagePages> 0.1333:
| | | | | breadth <= 6: class 0
| | | | | breadth > 6: class 1
| | | | | MultilP = 1:
| | | | | TotalTime <= 361: class 0
| | | | | TotalTime > 361: class 1
depth> 1:
| MultiAgent = 0:
| | depth > 2: class 0
| | depth < 2:
| | | MultilP = 1: class 0
| | | MultilP = 0:
| | | | breadth <= 6: class 0
| | | | breadth > 6:
| | | | | RepeatedAccess <= 0.322: class 0
| | | | | RepeatedAccess > 0.322: class 1
| | MultiAgent = 1:
| | | totalPages <= 81: class 0
| | | totalPages > 81: class 1

```

Figure 4.18. Decision tree model for Web robot detection.

4. Web robots are more likely to make repeated requests for the same document since the Web pages retrieved by human users are often cached by the browser.

4.3.7 Characteristics of Decision Tree Induction

The following is a summary of the important characteristics of decision tree induction algorithms.

1. Decision tree induction is a nonparametric approach for building classification models. In other words, it does not require any prior assumptions regarding the type of probability distributions satisfied by the class and other attributes (unlike some of the techniques described in Chapter 5).

2. Finding an optimal decision tree is an NP-complete problem. Many decision tree algorithms employ a heuristic-based approach to guide their search in the vast hypothesis space. For example, the algorithm presented in Section 4.3.5 uses a greedy, top-down, recursive partitioning strategy for growing a decision tree.
3. Techniques developed for constructing decision trees are computationally inexpensive, making it possible to quickly construct models even when the training set size is very large. Furthermore, once a decision tree has been built, classifying a test record is extremely fast, with a worst-case complexity of $O(w)$, where w is the maximum depth of the tree.
4. Decision trees, especially smaller-sized trees, are relatively easy to interpret. The accuracies of the trees are also comparable to other classification techniques for many simple data sets.
5. Decision trees provide an expressive representation for learning discrete-valued functions. However, they do not generalize well to certain types of Boolean problems. One notable example is the parity function, whose value is 0 (1) when there is an odd (even) number of Boolean attributes with the value *True*. Accurate modeling of such a function requires a full decision tree with 2^d nodes, where d is the number of Boolean attributes (see Exercise 1 on page 198).
6. Decision tree algorithms are quite robust to the presence of noise, especially when methods for avoiding overfitting, as described in Section 4.4, are employed.
7. The presence of redundant attributes does not adversely affect the accuracy of decision trees. An attribute is redundant if it is strongly correlated with another attribute in the data. One of the two redundant attributes will not be used for splitting once the other attribute has been chosen. However, if the data set contains many irrelevant attributes, i.e., attributes that are not useful for the classification task, then some of the irrelevant attributes may be accidentally chosen during the tree-growing process, which results in a decision tree that is larger than necessary. Feature selection techniques can help to improve the accuracy of decision trees by eliminating the irrelevant attributes during preprocessing. We will investigate the issue of too many irrelevant attributes in Section 4.4.3.

8. Since most decision tree algorithms employ a top-down, recursive partitioning approach, the number of records becomes smaller as we traverse down the tree. At the leaf nodes, the number of records may be too small to make a statistically significant decision about the class representation of the nodes. This is known as the **data fragmentation** problem. One possible solution is to disallow further splitting when the number of records falls below a certain threshold.
9. A subtree can be replicated multiple times in a decision tree, as illustrated in Figure 4.19. This makes the decision tree more complex than necessary and perhaps more difficult to interpret. Such a situation can arise from decision tree implementations that rely on a single attribute test condition at each internal node. Since most of the decision tree algorithms use a divide-and-conquer partitioning strategy, the same test condition can be applied to different parts of the attribute space, thus leading to the subtree replication problem.

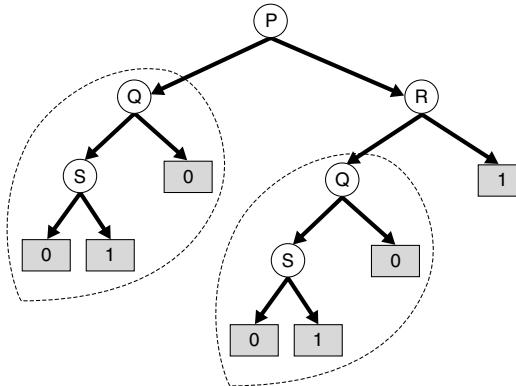


Figure 4.19. Tree replication problem. The same subtree can appear at different branches.

10. The test conditions described so far in this chapter involve using only a single attribute at a time. As a consequence, the tree-growing procedure can be viewed as the process of partitioning the attribute space into disjoint regions until each region contains records of the same class (see Figure 4.20). The border between two neighboring regions of different classes is known as a **decision boundary**. Since the test condition involves only a single attribute, the decision boundaries are rectilinear; i.e., parallel to the “coordinate axes.” This limits the expressiveness of the

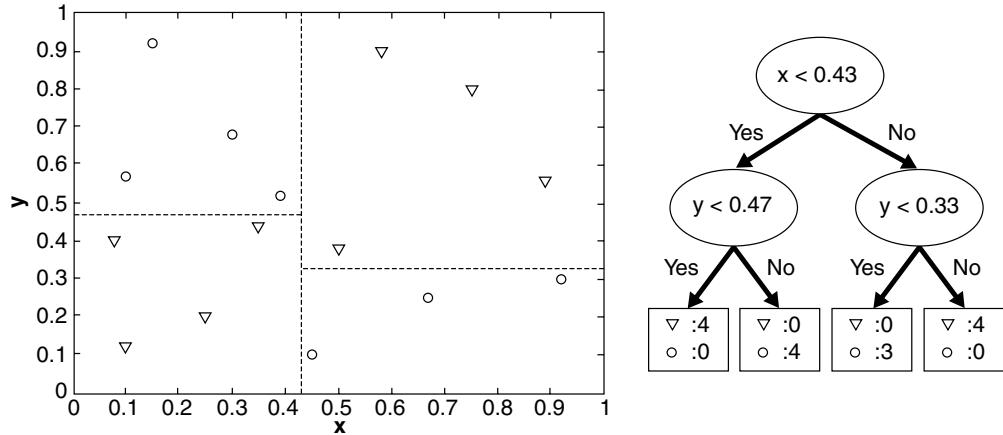


Figure 4.20. Example of a decision tree and its decision boundaries for a two-dimensional data set.

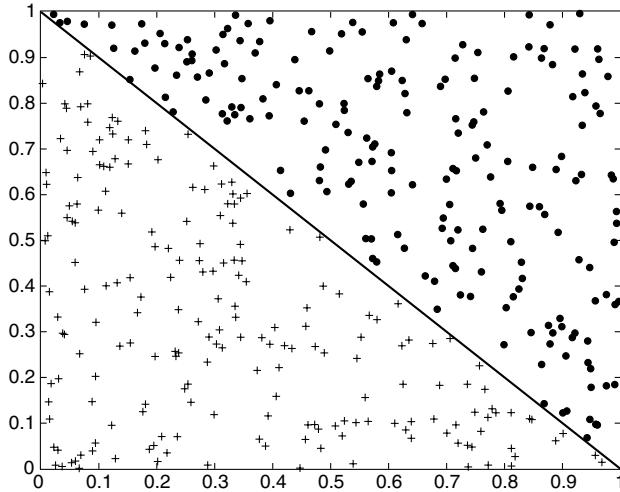


Figure 4.21. Example of data set that cannot be partitioned optimally using test conditions involving single attributes.

decision tree representation for modeling complex relationships among continuous attributes. Figure 4.21 illustrates a data set that cannot be classified effectively by a decision tree algorithm that uses test conditions involving only a single attribute at a time.

An **oblique decision tree** can be used to overcome this limitation because it allows test conditions that involve more than one attribute. The data set given in Figure 4.21 can be easily represented by an oblique decision tree containing a single node with test condition

$$x + y < 1.$$

Although such techniques are more expressive and can produce more compact trees, finding the optimal test condition for a given node can be computationally expensive.

Constructive induction provides another way to partition the data into homogeneous, nonrectangular regions (see Section 2.3.5 on page 57). This approach creates composite attributes representing an arithmetic or logical combination of the existing attributes. The new attributes provide a better discrimination of the classes and are augmented to the data set prior to decision tree induction. Unlike the oblique decision tree approach, constructive induction is less expensive because it identifies all the relevant combinations of attributes once, prior to constructing the decision tree. In contrast, an oblique decision tree must determine the right attribute combination dynamically, every time an internal node is expanded. However, constructive induction can introduce attribute redundancy in the data since the new attribute is a combination of several existing attributes.

11. Studies have shown that the choice of impurity measure has little effect on the performance of decision tree induction algorithms. This is because many impurity measures are quite consistent with each other, as shown in Figure 4.13 on page 159. Indeed, the strategy used to prune the tree has a greater impact on the final tree than the choice of impurity measure.

4.4 Model Overfitting

The errors committed by a classification model are generally divided into two types: **training errors** and **generalization errors**. Training error, also known as **resubstitution error** or **apparent error**, is the number of misclassification errors committed on training records, whereas generalization error is the expected error of the model on previously unseen records.

Recall from Section 4.2 that a good classification model must not only fit the training data well, it must also accurately classify records it has never

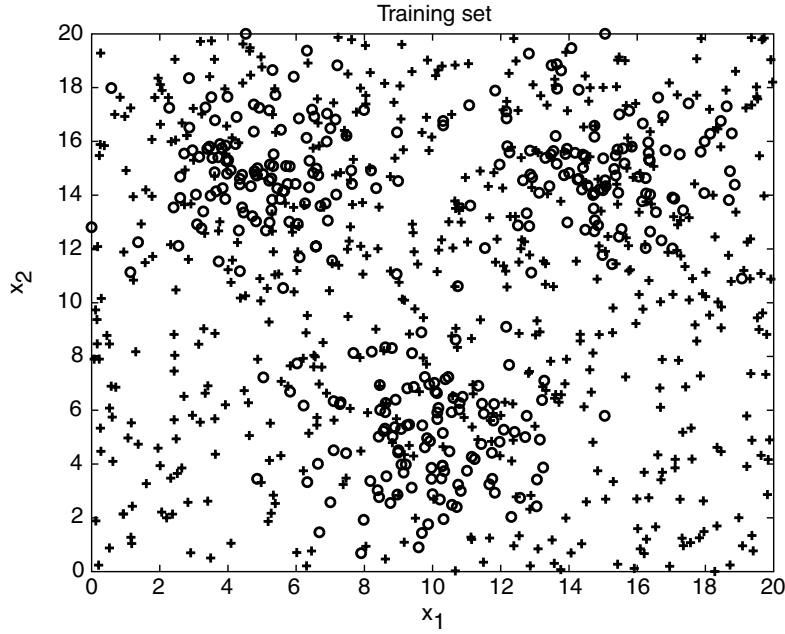


Figure 4.22. Example of a data set with binary classes.

seen before. In other words, a good model must have low training error as well as low generalization error. This is important because a model that fits the training data too well can have a poorer generalization error than a model with a higher training error. Such a situation is known as model overfitting.

Overfitting Example in Two-Dimensional Data For a more concrete example of the overfitting problem, consider the two-dimensional data set shown in Figure 4.22. The data set contains data points that belong to two different classes, denoted as class \circ and class $+$, respectively. The data points for the \circ class are generated from a mixture of three Gaussian distributions, while a uniform distribution is used to generate the data points for the $+$ class. There are altogether 1200 points belonging to the \circ class and 1800 points belonging to the $+$ class. 30% of the points are chosen for training, while the remaining 70% are used for testing. A decision tree classifier that uses the Gini index as its impurity measure is then applied to the training set. To investigate the effect of overfitting, different levels of pruning are applied to the initial, fully-grown tree. Figure 4.23(b) shows the training and test error rates of the decision tree.

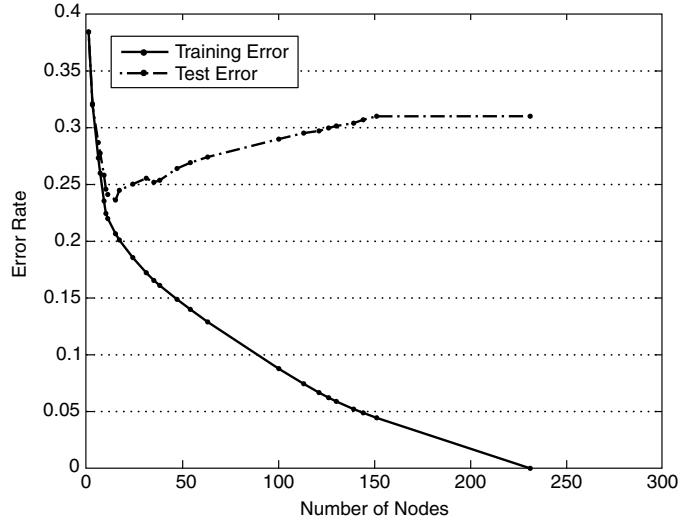
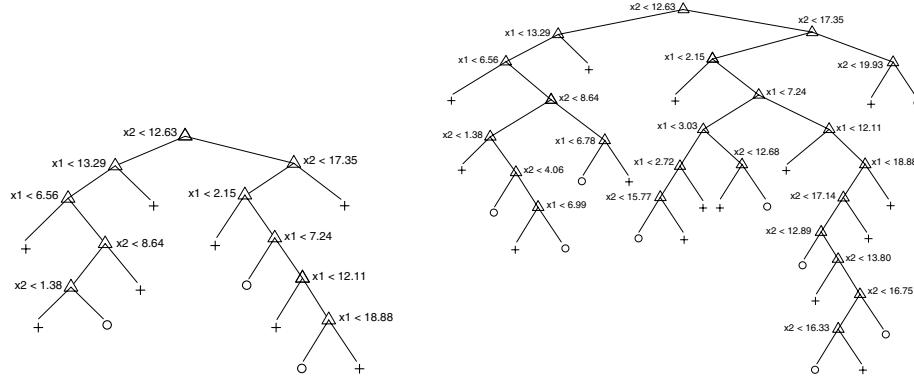


Figure 4.23. Training and test error rates.

Notice that the training and test error rates of the model are large when the size of the tree is very small. This situation is known as **model underfitting**. Underfitting occurs because the model has yet to learn the true structure of the data. As a result, it performs poorly on both the training and the test sets. As the number of nodes in the decision tree increases, the tree will have fewer training and test errors. However, once the tree becomes too large, its test error rate begins to increase even though its training error rate continues to decrease. This phenomenon is known as **model overfitting**.

To understand the overfitting phenomenon, note that the training error of a model can be reduced by increasing the model complexity. For example, the leaf nodes of the tree can be expanded until it perfectly fits the training data. Although the training error for such a complex tree is zero, the test error can be large because the tree may contain nodes that accidentally fit some of the noise points in the training data. Such nodes can degrade the performance of the tree because they do not generalize well to the test examples. Figure 4.24 shows the structure of two decision trees with different number of nodes. The tree that contains the smaller number of nodes has a higher training error rate, but a lower test error rate compared to the more complex tree.

Overfitting and underfitting are two pathologies that are related to the model complexity. The remainder of this section examines some of the potential causes of model overfitting.



(a) Decision tree with 11 leaf nodes.

(b) Decision tree with 24 leaf nodes.

Figure 4.24. Decision trees with different model complexities.

4.4.1 Overfitting Due to Presence of Noise

Consider the training and test sets shown in Tables 4.3 and 4.4 for the mammal classification problem. Two of the ten training records are mislabeled: bats and whales are classified as non-mammals instead of mammals.

A decision tree that perfectly fits the training data is shown in Figure 4.25(a). Although the training error for the tree is zero, its error rate on

Table 4.3. An example training set for classifying mammals. Class labels with asterisk symbols represent mislabeled records.

Name	Body Temperature	Gives Birth	Four-legged	Hibernates	Class Label
porcupine	warm-blooded	yes	yes	yes	yes
cat	warm-blooded	yes	yes	no	yes
bat	warm-blooded	yes	no	yes	no*
whale	warm-blooded	yes	no	no	no*
salamander	cold-blooded	no	yes	yes	no
komodo dragon	cold-blooded	no	yes	no	no
python	cold-blooded	no	no	yes	no
salmon	cold-blooded	no	no	no	no
eagle	warm-blooded	no	no	no	no
guppy	cold-blooded	yes	no	no	no

Table 4.4. An example test set for classifying mammals.

Name	Body Temperature	Gives Birth	Four-legged	Hibernates	Class Label
human	warm-blooded	yes	no	no	yes
pigeon	warm-blooded	no	no	no	no
elephant	warm-blooded	yes	yes	no	yes
leopard shark	cold-blooded	yes	no	no	no
turtle	cold-blooded	no	yes	no	no
penguin	cold-blooded	no	no	no	no
eel	cold-blooded	no	no	no	no
dolphin	warm-blooded	yes	no	no	yes
spiny anteater	warm-blooded	no	yes	yes	yes
gila monster	cold-blooded	no	yes	yes	no

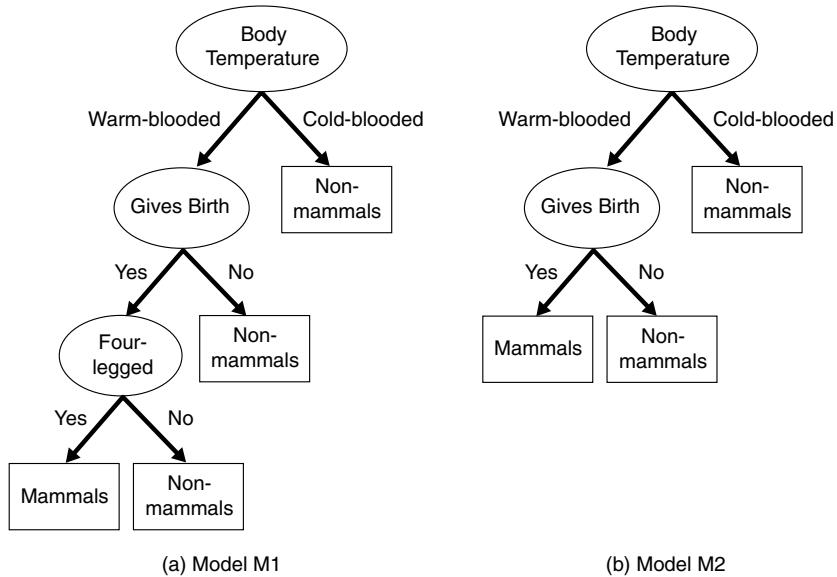


Figure 4.25. Decision tree induced from the data set shown in Table 4.3.

the test set is 30%. Both humans and dolphins were misclassified as non-mammals because their attribute values for **Body Temperature**, **Gives Birth**, and **Four-legged** are identical to the mislabeled records in the training set. Spiny anteaters, on the other hand, represent an exceptional case in which the class label of a test record contradicts the class labels of other similar records in the training set. Errors due to exceptional cases are often unavoidable and establish the minimum error rate achievable by any classifier.

In contrast, the decision tree M_2 shown in Figure 4.25(b) has a lower test error rate (10%) even though its training error rate is somewhat higher (20%). It is evident that the first decision tree, M_1 , has overfitted the training data because there is a simpler model with lower error rate on the test set. The **Four-legged** attribute test condition in model M_1 is spurious because it fits the mislabeled training records, which leads to the misclassification of records in the test set.

4.4.2 Overfitting Due to Lack of Representative Samples

Models that make their classification decisions based on a small number of training records are also susceptible to overfitting. Such models can be generated because of lack of representative samples in the training data and learning algorithms that continue to refine their models even when few training records are available. We illustrate these effects in the example below.

Consider the five training records shown in Table 4.5. All of these training records are labeled correctly and the corresponding decision tree is depicted in Figure 4.26. Although its training error is zero, its error rate on the test set is 30%.

Table 4.5. An example training set for classifying mammals.

Name	Body Temperature	Gives Birth	Four-legged	Hibernates	Class Label
salamander	cold-blooded	no	yes	yes	no
guppy	cold-blooded	yes	no	no	no
eagle	warm-blooded	no	no	no	no
poorwill	warm-blooded	no	no	yes	no
platypus	warm-blooded	no	yes	yes	yes

Humans, elephants, and dolphins are misclassified because the decision tree classifies all warm-blooded vertebrates that do not hibernate as non-mammals. The tree arrives at this classification decision because there is only one training record, which is an eagle, with such characteristics. This example clearly demonstrates the danger of making wrong predictions when there are not enough representative examples at the leaf nodes of a decision tree.

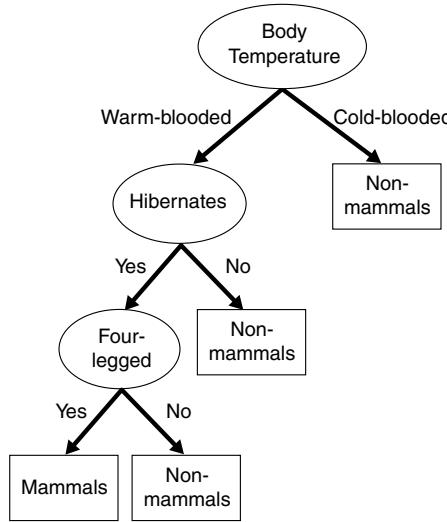


Figure 4.26. Decision tree induced from the data set shown in Table 4.5.

4.4.3 Overfitting and the Multiple Comparison Procedure

Model overfitting may arise in learning algorithms that employ a methodology known as multiple comparison procedure. To understand multiple comparison procedure, consider the task of predicting whether the stock market will rise or fall in the next ten trading days. If a stock analyst simply makes random guesses, the probability that her prediction is correct on any trading day is 0.5. However, the probability that she will predict correctly at least eight out of the ten times is

$$\frac{\binom{10}{8} + \binom{10}{9} + \binom{10}{10}}{2^{10}} = 0.0547,$$

which seems quite unlikely.

Suppose we are interested in choosing an investment advisor from a pool of fifty stock analysts. Our strategy is to select the analyst who makes the most correct predictions in the next ten trading days. The flaw in this strategy is that even if all the analysts had made their predictions in a random fashion, the probability that at least one of them makes at least eight correct predictions is

$$1 - (1 - 0.0547)^{50} = 0.9399,$$

which is very high. Although each analyst has a low probability of predicting at least eight times correctly, putting them together, we have a high probability of finding an analyst who can do so. Furthermore, there is no guarantee in the

future that such an analyst will continue to make accurate predictions through random guessing.

How does the multiple comparison procedure relate to model overfitting? Many learning algorithms explore a set of independent alternatives, $\{\gamma_i\}$, and then choose an alternative, γ_{\max} , that maximizes a given criterion function. The algorithm will add γ_{\max} to the current model in order to improve its overall performance. This procedure is repeated until no further improvement is observed. As an example, during decision tree growing, multiple tests are performed to determine which attribute can best split the training data. The attribute that leads to the best split is chosen to extend the tree as long as the observed improvement is statistically significant.

Let T_0 be the initial decision tree and T_x be the new tree after inserting an internal node for attribute x . In principle, x can be added to the tree if the observed gain, $\Delta(T_0, T_x)$, is greater than some predefined threshold α . If there is only one attribute test condition to be evaluated, then we can avoid inserting spurious nodes by choosing a large enough value of α . However, in practice, more than one test condition is available and the decision tree algorithm must choose the best attribute x_{\max} from a set of candidates, $\{x_1, x_2, \dots, x_k\}$, to partition the data. In this situation, the algorithm is actually using a multiple comparison procedure to decide whether a decision tree should be extended. More specifically, it is testing for $\Delta(T_0, T_{x_{\max}}) > \alpha$ instead of $\Delta(T_0, T_x) > \alpha$. As the number of alternatives, k , increases, so does our chance of finding $\Delta(T_0, T_{x_{\max}}) > \alpha$. Unless the gain function Δ or threshold α is modified to account for k , the algorithm may inadvertently add spurious nodes to the model, which leads to model overfitting.

This effect becomes more pronounced when the number of training records from which x_{\max} is chosen is small, because the variance of $\Delta(T_0, T_{x_{\max}})$ is high when fewer examples are available for training. As a result, the probability of finding $\Delta(T_0, T_{x_{\max}}) > \alpha$ increases when there are very few training records. This often happens when the decision tree grows deeper, which in turn reduces the number of records covered by the nodes and increases the likelihood of adding unnecessary nodes into the tree. Failure to compensate for the large number of alternatives or the small number of training records will therefore lead to model overfitting.

4.4.4 Estimation of Generalization Errors

Although the primary reason for overfitting is still a subject of debate, it is generally agreed that the complexity of a model has an impact on model overfitting, as was illustrated in Figure 4.23. The question is, how do we

determine the right model complexity? The ideal complexity is that of a model that produces the lowest generalization error. The problem is that the learning algorithm has access only to the training set during model building (see Figure 4.3). It has no knowledge of the test set, and thus, does not know how well the tree will perform on records it has never seen before. The best it can do is to estimate the generalization error of the induced tree. This section presents several methods for doing the estimation.

Using Resubstitution Estimate

The resubstitution estimate approach assumes that the training set is a good representation of the overall data. Consequently, the training error, otherwise known as resubstitution error, can be used to provide an optimistic estimate for the generalization error. Under this assumption, a decision tree induction algorithm simply selects the model that produces the lowest training error rate as its final model. However, the training error is usually a poor estimate of generalization error.

Example 4.1. Consider the binary decision trees shown in Figure 4.27. Assume that both trees are generated from the same training data and both make their classification decisions at each leaf node according to the majority class. Note that the left tree, T_L , is more complex because it expands some of the leaf nodes in the right tree, T_R . The training error rate for the left tree is $e(T_L) = 4/24 = 0.167$, while the training error rate for the right tree is

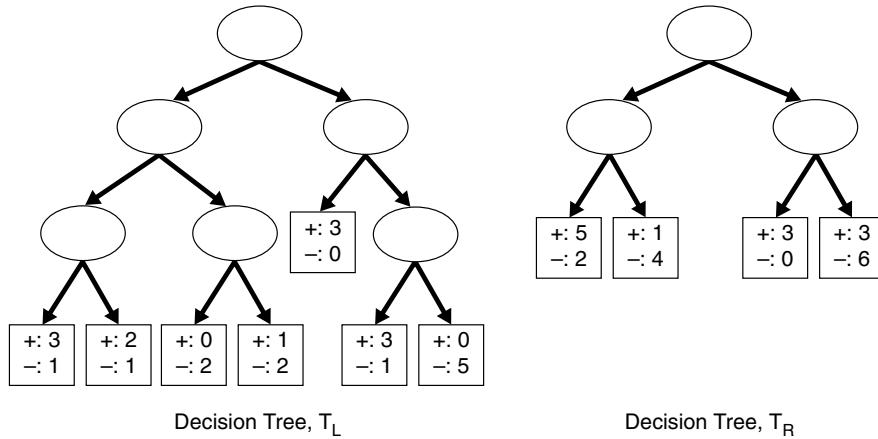


Figure 4.27. Example of two decision trees generated from the same training data.

$e(T_R) = 6/24 = 0.25$. Based on their resubstitution estimate, the left tree is considered better than the right tree. ■

Incorporating Model Complexity

As previously noted, the chance for model overfitting increases as the model becomes more complex. For this reason, we should prefer simpler models, a strategy that agrees with a well-known principle known as **Occam's razor** or the **principle of parsimony**:

Definition 4.2. Occam's Razor: Given two models with the same generalization errors, the simpler model is preferred over the more complex model.

Occam's razor is intuitive because the additional components in a complex model stand a greater chance of being fitted purely by chance. In the words of Einstein, “Everything should be made as simple as possible, but not simpler.” Next, we present two methods for incorporating model complexity into the evaluation of classification models.

Pessimistic Error Estimate The first approach explicitly computes generalization error as the sum of training error and a penalty term for model complexity. The resulting generalization error can be considered its pessimistic error estimate. For instance, let $n(t)$ be the number of training records classified by node t and $e(t)$ be the number of misclassified records. The pessimistic error estimate of a decision tree T , $e_g(T)$, can be computed as follows:

$$e_g(T) = \frac{\sum_{i=1}^k [e(t_i) + \Omega(t_i)]}{\sum_{i=1}^k n(t_i)} = \frac{e(T) + \Omega(T)}{N_t},$$

where k is the number of leaf nodes, $e(T)$ is the overall training error of the decision tree, N_t is the number of training records, and $\Omega(t_i)$ is the penalty term associated with each node t_i .

Example 4.2. Consider the binary decision trees shown in Figure 4.27. If the penalty term is equal to 0.5, then the pessimistic error estimate for the left tree is

$$e_g(T_L) = \frac{4 + 7 \times 0.5}{24} = \frac{7.5}{24} = 0.3125$$

and the pessimistic error estimate for the right tree is

$$e_g(T_R) = \frac{6 + 4 \times 0.5}{24} = \frac{8}{24} = 0.3333.$$

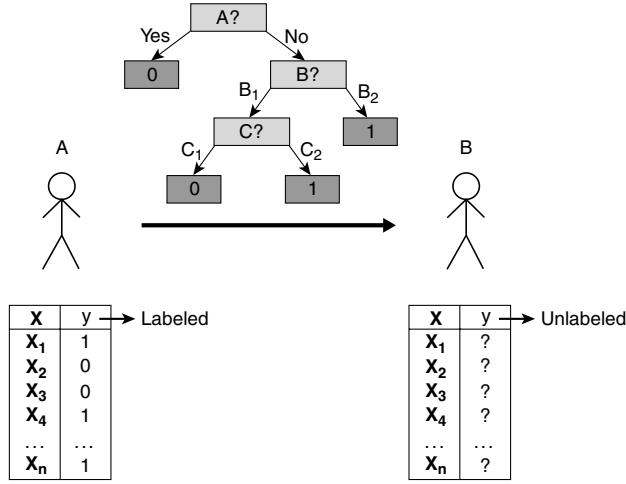


Figure 4.28. The minimum description length (MDL) principle.

Thus, the left tree has a better pessimistic error rate than the right tree. For binary trees, a penalty term of 0.5 means a node should always be expanded into its two child nodes as long as it improves the classification of at least one training record because expanding a node, which is equivalent to adding 0.5 to the overall error, is less costly than committing one training error.

If $\Omega(t) = 1$ for all the nodes t , the pessimistic error estimate for the left tree is $e_g(T_L) = 11/24 = 0.458$, while the pessimistic error estimate for the right tree is $e_g(T_R) = 10/24 = 0.417$. The right tree therefore has a better pessimistic error rate than the left tree. Thus, a node should not be expanded into its child nodes unless it reduces the misclassification error for more than one training record. ■

Minimum Description Length Principle Another way to incorporate model complexity is based on an information-theoretic approach known as the minimum description length or MDL principle. To illustrate this principle, consider the example shown in Figure 4.28. In this example, both A and B are given a set of records with known attribute values \mathbf{x} . In addition, person A knows the exact class label for each record, while person B knows none of this information. B can obtain the classification of each record by requesting that A transmits the class labels sequentially. Such a message would require $\Theta(n)$ bits of information, where n is the total number of records.

Alternatively, A may decide to build a classification model that summarizes the relationship between \mathbf{x} and y . The model can be encoded in a compact

form before being transmitted to B. If the model is 100% accurate, then the cost of transmission is equivalent to the cost of encoding the model. Otherwise, A must also transmit information about which record is classified incorrectly by the model. Thus, the overall cost of transmission is

$$Cost(model, data) = Cost(model) + Cost(data|model), \quad (4.9)$$

where the first term on the right-hand side is the cost of encoding the model, while the second term represents the cost of encoding the mislabeled records. According to the MDL principle, we should seek a model that minimizes the overall cost function. An example showing how to compute the total description length of a decision tree is given by Exercise 9 on page 202.

Estimating Statistical Bounds

The generalization error can also be estimated as a statistical correction to the training error. Since generalization error tends to be larger than training error, the statistical correction is usually computed as an upper bound to the training error, taking into account the number of training records that reach a particular leaf node. For instance, in the C4.5 decision tree algorithm, the number of errors committed by each leaf node is assumed to follow a binomial distribution. To compute its generalization error, we must determine the upper bound limit to the observed training error, as illustrated in the next example.

Example 4.3. Consider the left-most branch of the binary decision trees shown in Figure 4.27. Observe that the left-most leaf node of T_R has been expanded into two child nodes in T_L . Before splitting, the error rate of the node is $2/7 = 0.286$. By approximating a binomial distribution with a normal distribution, the following upper bound of the error rate e can be derived:

$$e_{upper}(N, e, \alpha) = \frac{e + \frac{z_{\alpha/2}^2}{2N} + z_{\alpha/2} \sqrt{\frac{e(1-e)}{N} + \frac{z_{\alpha/2}^2}{4N^2}}}{1 + \frac{z_{\alpha/2}^2}{N}}, \quad (4.10)$$

where α is the confidence level, $z_{\alpha/2}$ is the standardized value from a standard normal distribution, and N is the total number of training records used to compute e . By replacing $\alpha = 25\%$, $N = 7$, and $e = 2/7$, the upper bound for the error rate is $e_{upper}(7, 2/7, 0.25) = 0.503$, which corresponds to $7 \times 0.503 = 3.521$ errors. If we expand the node into its child nodes as shown in T_L , the training error rates for the child nodes are $1/4 = 0.250$ and $1/3 = 0.333$,

respectively. Using Equation 4.10, the upper bounds of these error rates are $e_{upper}(4, 1/4, 0.25) = 0.537$ and $e_{upper}(3, 1/3, 0.25) = 0.650$, respectively. The overall training error of the child nodes is $4 \times 0.537 + 3 \times 0.650 = 4.098$, which is larger than the estimated error for the corresponding node in T_R . ■

Using a Validation Set

In this approach, instead of using the training set to estimate the generalization error, the original training data is divided into two smaller subsets. One of the subsets is used for training, while the other, known as the validation set, is used for estimating the generalization error. Typically, two-thirds of the training set is reserved for model building, while the remaining one-third is used for error estimation.

This approach is typically used with classification techniques that can be parameterized to obtain models with different levels of complexity. The complexity of the best model can be estimated by adjusting the parameter of the learning algorithm (e.g., the pruning level of a decision tree) until the empirical model produced by the learning algorithm attains the lowest error rate on the validation set. Although this approach provides a better way for estimating how well the model performs on previously unseen records, less data is available for training.

4.4.5 Handling Overfitting in Decision Tree Induction

In the previous section, we described several methods for estimating the generalization error of a classification model. Having a reliable estimate of generalization error allows the learning algorithm to search for an accurate model without overfitting the training data. This section presents two strategies for avoiding model overfitting in the context of decision tree induction.

Prepruning (Early Stopping Rule) In this approach, the tree-growing algorithm is halted before generating a fully grown tree that perfectly fits the entire training data. To do this, a more restrictive stopping condition must be used; e.g., stop expanding a leaf node when the observed gain in impurity measure (or improvement in the estimated generalization error) falls below a certain threshold. The advantage of this approach is that it avoids generating overly complex subtrees that overfit the training data. Nevertheless, it is difficult to choose the right threshold for early termination. Too high of a threshold will result in underfitted models, while a threshold that is set too low may not be sufficient to overcome the model overfitting problem. Furthermore,

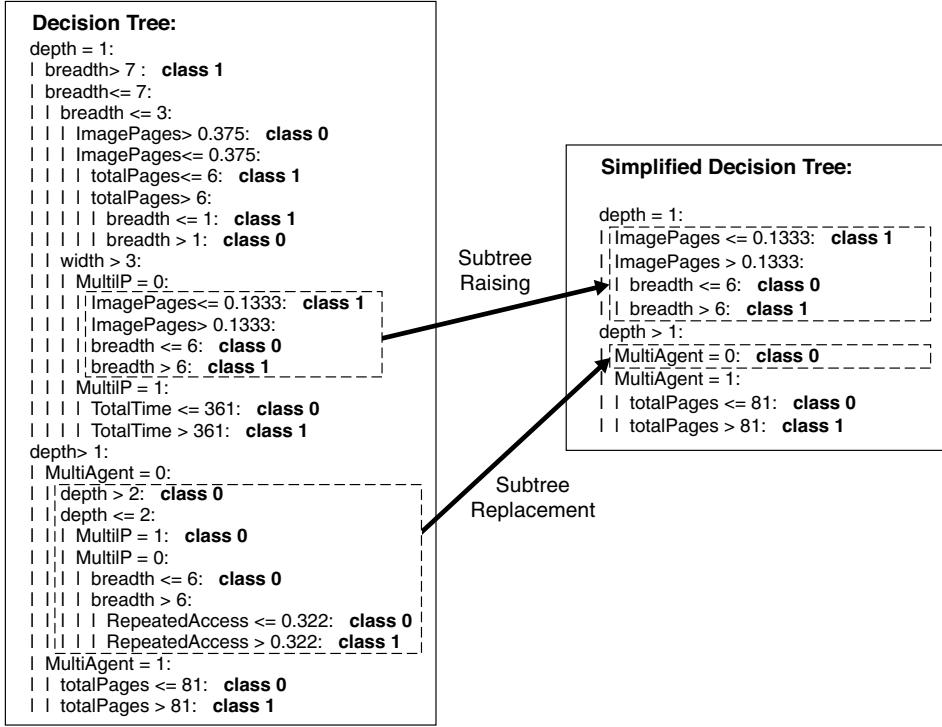


Figure 4.29. Post-pruning of the decision tree for Web robot detection.

even if no significant gain is obtained using one of the existing attribute test conditions, subsequent splitting may result in better subtrees.

Post-pruning In this approach, the decision tree is initially grown to its maximum size. This is followed by a tree-pruning step, which proceeds to trim the fully grown tree in a bottom-up fashion. Trimming can be done by replacing a subtree with (1) a new leaf node whose class label is determined from the majority class of records affiliated with the subtree, or (2) the most frequently used branch of the subtree. The tree-pruning step terminates when no further improvement is observed. Post-pruning tends to give better results than prepruning because it makes pruning decisions based on a fully grown tree, unlike prepruning, which can suffer from premature termination of the tree-growing process. However, for post-pruning, the additional computations needed to grow the full tree may be wasted when the subtree is pruned.

Figure 4.29 illustrates the simplified decision tree model for the Web robot detection example given in Section 4.3.6. Notice that the subtrees rooted at

`depth = 1` have been replaced by one of the branches involving the attribute `ImagePages`. This approach is also known as **subtree raising**. The `depth > 1` and `MultiAgent = 0` subtree has been replaced by a leaf node assigned to class 0. This approach is known as **subtree replacement**. The subtree for `depth > 1` and `MultiAgent = 1` remains intact.

4.5 Evaluating the Performance of a Classifier

Section 4.4.4 described several methods for estimating the generalization error of a model during training. The estimated error helps the learning algorithm to do **model selection**; i.e., to find a model of the right complexity that is not susceptible to overfitting. Once the model has been constructed, it can be applied to the test set to predict the class labels of previously unseen records.

It is often useful to measure the performance of the model on the test set because such a measure provides an unbiased estimate of its generalization error. The accuracy or error rate computed from the test set can also be used to compare the relative performance of different classifiers on the same domain. However, in order to do this, the class labels of the test records must be known. This section reviews some of the methods commonly used to evaluate the performance of a classifier.

4.5.1 Holdout Method

In the holdout method, the original data with labeled examples is partitioned into two disjoint sets, called the training and the test sets, respectively. A classification model is then induced from the training set and its performance is evaluated on the test set. The proportion of data reserved for training and for testing is typically at the discretion of the analysts (e.g., 50-50 or two-thirds for training and one-third for testing). The accuracy of the classifier can be estimated based on the accuracy of the induced model on the test set.

The holdout method has several well-known limitations. First, fewer labeled examples are available for training because some of the records are withheld for testing. As a result, the induced model may not be as good as when all the labeled examples are used for training. Second, the model may be highly dependent on the composition of the training and test sets. The smaller the training set size, the larger the variance of the model. On the other hand, if the training set is too large, then the estimated accuracy computed from the smaller test set is less reliable. Such an estimate is said to have a wide confidence interval. Finally, the training and test sets are no longer independent

of each other. Because the training and test sets are subsets of the original data, a class that is overrepresented in one subset will be underrepresented in the other, and vice versa.

4.5.2 Random Subsampling

The holdout method can be repeated several times to improve the estimation of a classifier's performance. This approach is known as random subsampling. Let acc_i be the model accuracy during the i^{th} iteration. The overall accuracy is given by $acc_{\text{sub}} = \sum_{i=1}^k acc_i/k$. Random subsampling still encounters some of the problems associated with the holdout method because it does not utilize as much data as possible for training. It also has no control over the number of times each record is used for testing and training. Consequently, some records might be used for training more often than others.

4.5.3 Cross-Validation

An alternative to random subsampling is cross-validation. In this approach, each record is used the same number of times for training and exactly once for testing. To illustrate this method, suppose we partition the data into two equal-sized subsets. First, we choose one of the subsets for training and the other for testing. We then swap the roles of the subsets so that the previous training set becomes the test set and vice versa. This approach is called a two-fold cross-validation. The total error is obtained by summing up the errors for both runs. In this example, each record is used exactly once for training and once for testing. The k -fold cross-validation method generalizes this approach by segmenting the data into k equal-sized partitions. During each run, one of the partitions is chosen for testing, while the rest of them are used for training. This procedure is repeated k times so that each partition is used for testing exactly once. Again, the total error is found by summing up the errors for all k runs. A special case of the k -fold cross-validation method sets $k = N$, the size of the data set. In this so-called **leave-one-out** approach, each test set contains only one record. This approach has the advantage of utilizing as much data as possible for training. In addition, the test sets are mutually exclusive and they effectively cover the entire data set. The drawback of this approach is that it is computationally expensive to repeat the procedure N times. Furthermore, since each test set contains only one record, the variance of the estimated performance metric tends to be high.

4.5.4 Bootstrap

The methods presented so far assume that the training records are sampled without replacement. As a result, there are no duplicate records in the training and test sets. In the bootstrap approach, the training records are sampled with replacement; i.e., a record already chosen for training is put back into the original pool of records so that it is equally likely to be redrawn. If the original data has N records, it can be shown that, on average, a bootstrap sample of size N contains about 63.2% of the records in the original data. This approximation follows from the fact that the probability a record is chosen by a bootstrap sample is $1 - (1 - 1/N)^N$. When N is sufficiently large, the probability asymptotically approaches $1 - e^{-1} = 0.632$. Records that are not included in the bootstrap sample become part of the test set. The model induced from the training set is then applied to the test set to obtain an estimate of the accuracy of the bootstrap sample, ϵ_i . The sampling procedure is then repeated b times to generate b bootstrap samples.

There are several variations to the bootstrap sampling approach in terms of how the overall accuracy of the classifier is computed. One of the more widely used approaches is the **.632 bootstrap**, which computes the overall accuracy by combining the accuracies of each bootstrap sample (ϵ_i) with the accuracy computed from a training set that contains all the labeled examples in the original data (acc_s):

$$\text{Accuracy, } acc_{boot} = \frac{1}{b} \sum_{i=1}^b (0.632 \times \epsilon_i + 0.368 \times acc_s). \quad (4.11)$$

4.6 Methods for Comparing Classifiers

It is often useful to compare the performance of different classifiers to determine which classifier works better on a given data set. However, depending on the size of the data, the observed difference in accuracy between two classifiers may not be statistically significant. This section examines some of the statistical tests available to compare the performance of different models and classifiers.

For illustrative purposes, consider a pair of classification models, M_A and M_B . Suppose M_A achieves 85% accuracy when evaluated on a test set containing 30 records, while M_B achieves 75% accuracy on a different test set containing 5000 records. Based on this information, is M_A a better model than M_B ?

The preceding example raises two key questions regarding the statistical significance of the performance metrics:

1. Although M_A has a higher accuracy than M_B , it was tested on a smaller test set. How much confidence can we place on the accuracy for M_A ?
2. Is it possible to explain the difference in accuracy as a result of variations in the composition of the test sets?

The first question relates to the issue of estimating the confidence interval of a given model accuracy. The second question relates to the issue of testing the statistical significance of the observed deviation. These issues are investigated in the remainder of this section.

4.6.1 Estimating a Confidence Interval for Accuracy

To determine the confidence interval, we need to establish the probability distribution that governs the accuracy measure. This section describes an approach for deriving the confidence interval by modeling the classification task as a binomial experiment. Following is a list of characteristics of a binomial experiment:

1. The experiment consists of N independent trials, where each trial has two possible outcomes: success or failure.
2. The probability of success, p , in each trial is constant.

An example of a binomial experiment is counting the number of heads that turn up when a coin is flipped N times. If X is the number of successes observed in N trials, then the probability that X takes a particular value is given by a binomial distribution with mean Np and variance $Np(1 - p)$:

$$P(X = v) = \binom{N}{p} p^v (1 - p)^{N-v}.$$

For example, if the coin is fair ($p = 0.5$) and is flipped fifty times, then the probability that the head shows up 20 times is

$$P(X = 20) = \binom{50}{20} 0.5^{20} (1 - 0.5)^{30} = 0.0419.$$

If the experiment is repeated many times, then the average number of heads expected to show up is $50 \times 0.5 = 25$, while its variance is $50 \times 0.5 \times 0.5 = 12.5$.

The task of predicting the class labels of test records can also be considered as a binomial experiment. Given a test set that contains N records, let X be the number of records correctly predicted by a model and p be the true accuracy of the model. By modeling the prediction task as a binomial experiment, X has a binomial distribution with mean Np and variance $Np(1 - p)$. It can be shown that the empirical accuracy, $acc = X/N$, also has a binomial distribution with mean p and variance $p(1 - p)/N$ (see Exercise 12). Although the binomial distribution can be used to estimate the confidence interval for acc , it is often approximated by a normal distribution when N is sufficiently large. Based on the normal distribution, the following confidence interval for acc can be derived:

$$P\left(-Z_{\alpha/2} \leq \frac{acc - p}{\sqrt{p(1 - p)/N}} \leq Z_{1-\alpha/2}\right) = 1 - \alpha, \quad (4.12)$$

where $Z_{\alpha/2}$ and $Z_{1-\alpha/2}$ are the upper and lower bounds obtained from a standard normal distribution at confidence level $(1 - \alpha)$. Since a standard normal distribution is symmetric around $Z = 0$, it follows that $Z_{\alpha/2} = Z_{1-\alpha/2}$. Rearranging this inequality leads to the following confidence interval for p :

$$\frac{2 \times N \times acc + Z_{\alpha/2}^2 \pm Z_{\alpha/2} \sqrt{Z_{\alpha/2}^2 + 4Nacc - 4Nacc^2}}{2(N + Z_{\alpha/2}^2)}. \quad (4.13)$$

The following table shows the values of $Z_{\alpha/2}$ at different confidence levels:

$1 - \alpha$	0.99	0.98	0.95	0.9	0.8	0.7	0.5
$Z_{\alpha/2}$	2.58	2.33	1.96	1.65	1.28	1.04	0.67

Example 4.4. Consider a model that has an accuracy of 80% when evaluated on 100 test records. What is the confidence interval for its true accuracy at a 95% confidence level? The confidence level of 95% corresponds to $Z_{\alpha/2} = 1.96$ according to the table given above. Inserting this term into Equation 4.13 yields a confidence interval between 71.1% and 86.7%. The following table shows the confidence interval when the number of records, N , increases:

N	20	50	100	500	1000	5000
Confidence Interval	0.584	0.670	0.711	0.763	0.774	0.789
	- 0.919	- 0.888	- 0.867	- 0.833	- 0.824	- 0.811

Note that the confidence interval becomes tighter when N increases. ■

4.6.2 Comparing the Performance of Two Models

Consider a pair of models, M_1 and M_2 , that are evaluated on two independent test sets, D_1 and D_2 . Let n_1 denote the number of records in D_1 and n_2 denote the number of records in D_2 . In addition, suppose the error rate for M_1 on D_1 is e_1 and the error rate for M_2 on D_2 is e_2 . Our goal is to test whether the observed difference between e_1 and e_2 is statistically significant.

Assuming that n_1 and n_2 are sufficiently large, the error rates e_1 and e_2 can be approximated using normal distributions. If the observed difference in the error rate is denoted as $d = e_1 - e_2$, then d is also normally distributed with mean d_t , its true difference, and variance, σ_d^2 . The variance of d can be computed as follows:

$$\sigma_d^2 \approx \hat{\sigma}_d^2 = \frac{e_1(1 - e_1)}{n_1} + \frac{e_2(1 - e_2)}{n_2}, \quad (4.14)$$

where $e_1(1 - e_1)/n_1$ and $e_2(1 - e_2)/n_2$ are the variances of the error rates. Finally, at the $(1 - \alpha)\%$ confidence level, it can be shown that the confidence interval for the true difference d_t is given by the following equation:

$$d_t = d \pm z_{\alpha/2} \hat{\sigma}_d. \quad (4.15)$$

Example 4.5. Consider the problem described at the beginning of this section. Model M_A has an error rate of $e_1 = 0.15$ when applied to $N_1 = 30$ test records, while model M_B has an error rate of $e_2 = 0.25$ when applied to $N_2 = 5000$ test records. The observed difference in their error rates is $d = |0.15 - 0.25| = 0.1$. In this example, we are performing a two-sided test to check whether $d_t = 0$ or $d_t \neq 0$. The estimated variance of the observed difference in error rates can be computed as follows:

$$\hat{\sigma}_d^2 = \frac{0.15(1 - 0.15)}{30} + \frac{0.25(1 - 0.25)}{5000} = 0.0043$$

or $\hat{\sigma}_d = 0.0655$. Inserting this value into Equation 4.15, we obtain the following confidence interval for d_t at 95% confidence level:

$$d_t = 0.1 \pm 1.96 \times 0.0655 = 0.1 \pm 0.128.$$

As the interval spans the value zero, we can conclude that the observed difference is not statistically significant at a 95% confidence level. ■

At what confidence level can we reject the hypothesis that $d_t = 0$? To do this, we need to determine the value of $Z_{\alpha/2}$ such that the confidence interval for d_t does not span the value zero. We can reverse the preceding computation and look for the value $Z_{\alpha/2}$ such that $d > Z_{\alpha/2}\hat{\sigma}_d$. Replacing the values of d and $\hat{\sigma}_d$ gives $Z_{\alpha/2} < 1.527$. This value first occurs when $(1 - \alpha) \lesssim 0.936$ (for a two-sided test). The result suggests that the null hypothesis can be rejected at confidence level of 93.6% or lower.

4.6.3 Comparing the Performance of Two Classifiers

Suppose we want to compare the performance of two classifiers using the k -fold cross-validation approach. Initially, the data set D is divided into k equal-sized partitions. We then apply each classifier to construct a model from $k - 1$ of the partitions and test it on the remaining partition. This step is repeated k times, each time using a different partition as the test set.

Let M_{ij} denote the model induced by classification technique L_i during the j^{th} iteration. Note that each pair of models M_{1j} and M_{2j} are tested on the same partition j . Let e_{1j} and e_{2j} be their respective error rates. The difference between their error rates during the j^{th} fold can be written as $d_j = e_{1j} - e_{2j}$. If k is sufficiently large, then d_j is normally distributed with mean d_t^{cv} , which is the true difference in their error rates, and variance σ^{cv} . Unlike the previous approach, the overall variance in the observed differences is estimated using the following formula:

$$\hat{\sigma}_{d^{cv}}^2 = \frac{\sum_{j=1}^k (d_j - \bar{d})^2}{k(k-1)}, \quad (4.16)$$

where \bar{d} is the average difference. For this approach, we need to use a t -distribution to compute the confidence interval for d_t^{cv} :

$$d_t^{cv} = \bar{d} \pm t_{(1-\alpha), k-1} \hat{\sigma}_{d^{cv}}.$$

The coefficient $t_{(1-\alpha), k-1}$ is obtained from a probability table with two input parameters, its confidence level $(1 - \alpha)$ and the number of degrees of freedom, $k - 1$. The probability table for the t -distribution is shown in Table 4.6.

Example 4.6. Suppose the estimated difference in the accuracy of models generated by two classification techniques has a mean equal to 0.05 and a standard deviation equal to 0.002. If the accuracy is estimated using a 30-fold cross-validation approach, then at a 95% confidence level, the true accuracy difference is

$$d_t^{cv} = 0.05 \pm 2.04 \times 0.002. \quad (4.17)$$

Table 4.6. Probability table for t -distribution.

$k - 1$	(1 - α)				
	0.99	0.98	0.95	0.9	0.8
1	3.08	6.31	12.7	31.8	63.7
2	1.89	2.92	4.30	6.96	9.92
4	1.53	2.13	2.78	3.75	4.60
9	1.38	1.83	2.26	2.82	3.25
14	1.34	1.76	2.14	2.62	2.98
19	1.33	1.73	2.09	2.54	2.86
24	1.32	1.71	2.06	2.49	2.80
29	1.31	1.70	2.04	2.46	2.76

Since the confidence interval does not span the value zero, the observed difference between the techniques is statistically significant. ■

4.7 Bibliographic Notes

Early classification systems were developed to organize a large collection of objects. For example, the Dewey Decimal and Library of Congress classification systems were designed to catalog and index the vast number of library books. The categories are typically identified in a manual fashion, with the help of domain experts.

Automated classification has been a subject of intensive research for many years. The study of classification in classical statistics is sometimes known as **discriminant analysis**, where the objective is to predict the group membership of an object based on a set of predictor variables. A well-known classical method is Fisher's linear discriminant analysis [117], which seeks to find a linear projection of the data that produces the greatest discrimination between objects that belong to different classes.

Many pattern recognition problems also require the discrimination of objects from different classes. Examples include speech recognition, handwritten character identification, and image classification. Readers who are interested in the application of classification techniques for pattern recognition can refer to the survey articles by Jain et al. [122] and Kulkarni et al. [128] or classic pattern recognition books by Bishop [107], Duda et al. [114], and Fukunaga [118]. The subject of classification is also a major research topic in the fields of neural networks, statistical learning, and machine learning. An in-depth treat-

ment of various classification techniques is given in the books by Cherkassky and Mulier [112], Hastie et al. [120], Michie et al. [133], and Mitchell [136].

An overview of decision tree induction algorithms can be found in the survey articles by Buntine [110], Moret [137], Murthy [138], and Safavian et al. [147]. Examples of some well-known decision tree algorithms include CART [108], ID3 [143], C4.5 [145], and CHAID [125]. Both ID3 and C4.5 employ the entropy measure as their splitting function. An in-depth discussion of the C4.5 decision tree algorithm is given by Quinlan [145]. Besides explaining the methodology for decision tree growing and tree pruning, Quinlan [145] also described how the algorithm can be modified to handle data sets with missing values. The CART algorithm was developed by Breiman et al. [108] and uses the Gini index as its splitting function. CHAID [125] uses the statistical χ^2 test to determine the best split during the tree-growing process.

The decision tree algorithm presented in this chapter assumes that the splitting condition is specified one attribute at a time. An oblique decision tree can use multiple attributes to form the attribute test condition in the internal nodes [121, 152]. Breiman et al. [108] provide an option for using linear combinations of attributes in their CART implementation. Other approaches for inducing oblique decision trees were proposed by Heath et al. [121], Murthy et al. [139], Cantú-Paz and Kamath [111], and Utgoff and Brodley [152]. Although oblique decision trees help to improve the expressiveness of a decision tree representation, learning the appropriate test condition at each node is computationally challenging. Another way to improve the expressiveness of a decision tree without using oblique decision trees is to apply a method known as **constructive induction** [132]. This method simplifies the task of learning complex splitting functions by creating compound features from the original attributes.

Besides the top-down approach, other strategies for growing a decision tree include the bottom-up approach by Landeweerd et al. [130] and Pattipati and Alexandridis [142], as well as the bidirectional approach by Kim and Landgrebe [126]. Schuermann and Doster [150] and Wang and Suen [154] proposed using a **soft splitting criterion** to address the data fragmentation problem. In this approach, each record is assigned to different branches of the decision tree with different probabilities.

Model overfitting is an important issue that must be addressed to ensure that a decision tree classifier performs equally well on previously unknown records. The model overfitting problem has been investigated by many authors including Breiman et al. [108], Schaffer [148], Mingers [135], and Jensen and Cohen [123]. While the presence of noise is often regarded as one of the

primary reasons for overfitting [135, 140], Jensen and Cohen [123] argued that overfitting is the result of using incorrect hypothesis tests in a multiple comparison procedure.

Schapire [149] defined generalization error as “the probability of misclassifying a new example” and test error as “the fraction of mistakes on a newly sampled test set.” Generalization error can therefore be considered as the expected test error of a classifier. Generalization error may sometimes refer to the true error [136] of a model, i.e., its expected error for randomly drawn data points from the same population distribution where the training set is sampled. These definitions are in fact equivalent if both the training and test sets are gathered from the same population distribution, which is often the case in many data mining and machine learning applications.

The Occam’s razor principle is often attributed to the philosopher William of Occam. Domingos [113] cautioned against the pitfall of misinterpreting Occam’s razor as comparing models with similar training errors, instead of generalization errors. A survey on decision tree-pruning methods to avoid overfitting is given by Breslow and Aha [109] and Esposito et al. [116]. Some of the typical pruning methods include reduced error pruning [144], pessimistic error pruning [144], minimum error pruning [141], critical value pruning [134], cost-complexity pruning [108], and error-based pruning [145]. Quinlan and Rivest proposed using the minimum description length principle for decision tree pruning in [146].

Kohavi [127] had performed an extensive empirical study to compare the performance metrics obtained using different estimation methods such as random subsampling, bootstrapping, and k -fold cross-validation. Their results suggest that the best estimation method is based on the ten-fold stratified cross-validation. Efron and Tibshirani [115] provided a theoretical and empirical comparison between cross-validation and a bootstrap method known as the 632+ rule.

Current techniques such as C4.5 require that the entire training data set fit into main memory. There has been considerable effort to develop parallel and scalable versions of decision tree induction algorithms. Some of the proposed algorithms include SLIQ by Mehta et al. [131], SPRINT by Shafer et al. [151], CMP by Wang and Zaniolo [153], CLOUDS by Alsabti et al. [106], RainForest by Gehrke et al. [119], and ScalParC by Joshi et al. [124]. A general survey of parallel algorithms for data mining is available in [129].

Bibliography

- [106] K. Alsabti, S. Ranka, and V. Singh. CLOUDS: A Decision Tree Classifier for Large Datasets. In *Proc. of the 4th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 2–8, New York, NY, August 1998.
- [107] C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford, U.K., 1995.
- [108] L. Breiman, J. H. Friedman, R. Olshen, and C. J. Stone. *Classification and Regression Trees*. Chapman & Hall, New York, 1984.
- [109] L. A. Breslow and D. W. Aha. Simplifying Decision Trees: A Survey. *Knowledge Engineering Review*, 12(1):1–40, 1997.
- [110] W. Buntine. Learning classification trees. In *Artificial Intelligence Frontiers in Statistics*, pages 182–201. Chapman & Hall, London, 1993.
- [111] E. Cantú-Paz and C. Kamath. Using evolutionary algorithms to induce oblique decision trees. In *Proc. of the Genetic and Evolutionary Computation Conf.*, pages 1053–1060, San Francisco, CA, 2000.
- [112] V. Cherkassky and F. Mulier. *Learning from Data: Concepts, Theory, and Methods*. Wiley Interscience, 1998.
- [113] P. Domingos. The Role of Occam’s Razor in Knowledge Discovery. *Data Mining and Knowledge Discovery*, 3(4):409–425, 1999.
- [114] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. John Wiley & Sons, Inc., New York, 2nd edition, 2001.
- [115] B. Efron and R. Tibshirani. Cross-validation and the Bootstrap: Estimating the Error Rate of a Prediction Rule. Technical report, Stanford University, 1995.
- [116] F. Esposito, D. Malerba, and G. Semeraro. A Comparative Analysis of Methods for Pruning Decision Trees. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 19(5):476–491, May 1997.
- [117] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7:179–188, 1936.
- [118] K. Fukunaga. *Introduction to Statistical Pattern Recognition*. Academic Press, New York, 1990.
- [119] J. Gehrke, R. Ramakrishnan, and V. Ganti. RainForest—A Framework for Fast Decision Tree Construction of Large Datasets. *Data Mining and Knowledge Discovery*, 4(2/3):127–162, 2000.
- [120] T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, Prediction*. Springer, New York, 2001.
- [121] D. Heath, S. Kasif, and S. Salzberg. Induction of Oblique Decision Trees. In *Proc. of the 13th Intl. Joint Conf. on Artificial Intelligence*, pages 1002–1007, Chambery, France, August 1993.
- [122] A. K. Jain, R. P. W. Duin, and J. Mao. Statistical Pattern Recognition: A Review. *IEEE Tran. Patt. Anal. and Mach. Intellig.*, 22(1):4–37, 2000.
- [123] D. Jensen and P. R. Cohen. Multiple Comparisons in Induction Algorithms. *Machine Learning*, 38(3):309–338, March 2000.
- [124] M. V. Joshi, G. Karypis, and V. Kumar. ScalParC: A New Scalable and Efficient Parallel Classification Algorithm for Mining Large Datasets. In *Proc. of 12th Intl. Parallel Processing Symp. (IPPS/SPDP)*, pages 573–579, Orlando, FL, April 1998.
- [125] G. V. Kass. An Exploratory Technique for Investigating Large Quantities of Categorical Data. *Applied Statistics*, 29:119–127, 1980.

- [126] B. Kim and D. Landgrebe. Hierarchical decision classifiers in high-dimensional and large class data. *IEEE Trans. on Geoscience and Remote Sensing*, 29(4):518–528, 1991.
- [127] R. Kohavi. A Study on Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In *Proc. of the 15th Intl. Joint Conf. on Artificial Intelligence*, pages 1137–1145, Montreal, Canada, August 1995.
- [128] S. R. Kulkarni, G. Lugosi, and S. S. Venkatesh. Learning Pattern Classification—A Survey. *IEEE Tran. Inf. Theory*, 44(6):2178–2206, 1998.
- [129] V. Kumar, M. V. Joshi, E.-H. Han, P. N. Tan, and M. Steinbach. High Performance Data Mining. In *High Performance Computing for Computational Science (VECPAR 2002)*, pages 111–125. Springer, 2002.
- [130] G. Landeweerd, T. Timmers, E. Gersema, M. Bins, and M. Halic. Binary tree versus single level tree classification of white blood cells. *Pattern Recognition*, 16:571–577, 1983.
- [131] M. Mehta, R. Agrawal, and J. Rissanen. SLIQ: A Fast Scalable Classifier for Data Mining. In *Proc. of the 5th Intl. Conf. on Extending Database Technology*, pages 18–32, Avignon, France, March 1996.
- [132] R. S. Michalski. A theory and methodology of inductive learning. *Artificial Intelligence*, 20:111–116, 1983.
- [133] D. Michie, D. J. Spiegelhalter, and C. C. Taylor. *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, Upper Saddle River, NJ, 1994.
- [134] J. Mingers. Expert Systems—Rule Induction with Statistical Data. *J Operational Research Society*, 38:39–47, 1987.
- [135] J. Mingers. An empirical comparison of pruning methods for decision tree induction. *Machine Learning*, 4:227–243, 1989.
- [136] T. Mitchell. *Machine Learning*. McGraw-Hill, Boston, MA, 1997.
- [137] B. M. E. Moret. Decision Trees and Diagrams. *Computing Surveys*, 14(4):593–623, 1982.
- [138] S. K. Murthy. Automatic Construction of Decision Trees from Data: A Multi-Disciplinary Survey. *Data Mining and Knowledge Discovery*, 2(4):345–389, 1998.
- [139] S. K. Murthy, S. Kasif, and S. Salzberg. A system for induction of oblique decision trees. *J of Artificial Intelligence Research*, 2:1–33, 1994.
- [140] T. Niblett. Constructing decision trees in noisy domains. In *Proc. of the 2nd European Working Session on Learning*, pages 67–78, Bled, Yugoslavia, May 1987.
- [141] T. Niblett and I. Bratko. Learning Decision Rules in Noisy Domains. In *Research and Development in Expert Systems III*, Cambridge, 1986. Cambridge University Press.
- [142] K. R. Pattipati and M. G. Alexandridis. Application of heuristic search and information theory to sequential fault diagnosis. *IEEE Trans. on Systems, Man, and Cybernetics*, 20(4):872–887, 1990.
- [143] J. R. Quinlan. Discovering rules by induction from large collection of examples. In D. Michie, editor, *Expert Systems in the Micro Electronic Age*. Edinburgh University Press, Edinburgh, UK, 1979.
- [144] J. R. Quinlan. Simplifying Decision Trees. *Intl. J. Man-Machine Studies*, 27:221–234, 1987.
- [145] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan-Kaufmann Publishers, San Mateo, CA, 1993.
- [146] J. R. Quinlan and R. L. Rivest. Inferring Decision Trees Using the Minimum Description Length Principle. *Information and Computation*, 80(3):227–248, 1989.

- [147] S. R. Safavian and D. Landgrebe. A Survey of Decision Tree Classifier Methodology. *IEEE Trans. Systems, Man and Cybernetics*, 22:660–674, May/June 1998.
- [148] C. Schaffer. Overfitting avoidance as bias. *Machine Learning*, 10:153–178, 1993.
- [149] R. E. Schapire. The Boosting Approach to Machine Learning: An Overview. In *MSRI Workshop on Nonlinear Estimation and Classification*, 2002.
- [150] J. Schuermann and W. Doster. A decision-theoretic approach in hierarchical classifier design. *Pattern Recognition*, 17:359–369, 1984.
- [151] J. C. Shafer, R. Agrawal, and M. Mehta. SPRINT: A Scalable Parallel Classifier for Data Mining. In *Proc. of the 22nd VLDB Conf.*, pages 544–555, Bombay, India, September 1996.
- [152] P. E. Utgoff and C. E. Brodley. An incremental method for finding multivariate splits for decision trees. In *Proc. of the 7th Intl. Conf. on Machine Learning*, pages 58–65, Austin, TX, June 1990.
- [153] H. Wang and C. Zaniolo. CMP: A Fast Decision Tree Classifier Using Multivariate Predictions. In *Proc. of the 16th Intl. Conf. on Data Engineering*, pages 449–460, San Diego, CA, March 2000.
- [154] Q. R. Wang and C. Y. Suen. Large tree classifier with heuristic search and global training. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 9(1):91–102, 1987.

4.8 Exercises

1. Draw the full decision tree for the parity function of four Boolean attributes, *A*, *B*, *C*, and *D*. Is it possible to simplify the tree?
2. Consider the training examples shown in Table 4.7 for a binary classification problem.
 - (a) Compute the Gini index for the overall collection of training examples.
 - (b) Compute the Gini index for the **Customer ID** attribute.
 - (c) Compute the Gini index for the **Gender** attribute.
 - (d) Compute the Gini index for the **Car Type** attribute using multiway split.
 - (e) Compute the Gini index for the **Shirt Size** attribute using multiway split.
 - (f) Which attribute is better, **Gender**, **Car Type**, or **Shirt Size**?
 - (g) Explain why **Customer ID** should not be used as the attribute test condition even though it has the lowest Gini.
3. Consider the training examples shown in Table 4.8 for a binary classification problem.
 - (a) What is the entropy of this collection of training examples with respect to the positive class?

Table 4.7. Data set for Exercise 2.

Customer ID	Gender	Car Type	Shirt Size	Class
1	M	Family	Small	C0
2	M	Sports	Medium	C0
3	M	Sports	Medium	C0
4	M	Sports	Large	C0
5	M	Sports	Extra Large	C0
6	M	Sports	Extra Large	C0
7	F	Sports	Small	C0
8	F	Sports	Small	C0
9	F	Sports	Medium	C0
10	F	Luxury	Large	C0
11	M	Family	Large	C1
12	M	Family	Extra Large	C1
13	M	Family	Medium	C1
14	M	Luxury	Extra Large	C1
15	F	Luxury	Small	C1
16	F	Luxury	Small	C1
17	F	Luxury	Medium	C1
18	F	Luxury	Medium	C1
19	F	Luxury	Medium	C1
20	F	Luxury	Large	C1

Table 4.8. Data set for Exercise 3.

Instance	a_1	a_2	a_3	Target Class
1	T	T	1.0	+
2	T	T	6.0	+
3	T	F	5.0	-
4	F	F	4.0	+
5	F	T	7.0	-
6	F	T	3.0	-
7	F	F	8.0	-
8	T	F	7.0	+
9	F	T	5.0	-

- (b) What are the information gains of a_1 and a_2 relative to these training examples?
- (c) For a_3 , which is a continuous attribute, compute the information gain for every possible split.

- (d) What is the best split (among a_1 , a_2 , and a_3) according to the information gain?
- (e) What is the best split (between a_1 and a_2) according to the classification error rate?
- (f) What is the best split (between a_1 and a_2) according to the Gini index?
4. Show that the entropy of a node never increases after splitting it into smaller successor nodes.
5. Consider the following data set for a binary class problem.

A	B	Class Label
T	F	+
T	T	+
T	T	+
T	F	-
T	T	+
F	F	-
F	F	-
F	F	-
T	T	-
T	F	-

- (a) Calculate the information gain when splitting on A and B . Which attribute would the decision tree induction algorithm choose?
- (b) Calculate the gain in the Gini index when splitting on A and B . Which attribute would the decision tree induction algorithm choose?
- (c) Figure 4.13 shows that entropy and the Gini index are both monotonously increasing on the range $[0, 0.5]$ and they are both monotonously decreasing on the range $[0.5, 1]$. Is it possible that information gain and the gain in the Gini index favor different attributes? Explain.
6. Consider the following set of training examples.

X	Y	Z	No. of Class C1 Examples	No. of Class C2 Examples
0	0	0	5	40
0	0	1	0	15
0	1	0	10	5
0	1	1	45	0
1	0	0	10	5
1	0	1	25	0
1	1	0	5	20
1	1	1	0	15

- (a) Compute a two-level decision tree using the greedy approach described in this chapter. Use the classification error rate as the criterion for splitting. What is the overall error rate of the induced tree?
- (b) Repeat part (a) using X as the first splitting attribute and then choose the best remaining attribute for splitting at each of the two successor nodes. What is the error rate of the induced tree?
- (c) Compare the results of parts (a) and (b). Comment on the suitability of the greedy heuristic used for splitting attribute selection.
7. The following table summarizes a data set with three attributes A , B , C and two class labels $+$, $-$. Build a two-level decision tree.
- | A | B | C | Number of Instances | |
|---|---|---|---------------------|----|
| | | | + | - |
| T | T | T | 5 | 0 |
| F | T | T | 0 | 20 |
| T | F | T | 20 | 0 |
| F | F | T | 0 | 5 |
| T | T | F | 0 | 0 |
| F | T | F | 25 | 0 |
| T | F | F | 0 | 0 |
| F | F | F | 0 | 25 |
- (a) According to the classification error rate, which attribute would be chosen as the first splitting attribute? For each attribute, show the contingency table and the gains in classification error rate.
- (b) Repeat for the two children of the root node.
- (c) How many instances are misclassified by the resulting decision tree?
- (d) Repeat parts (a), (b), and (c) using C as the splitting attribute.
- (e) Use the results in parts (c) and (d) to conclude about the greedy nature of the decision tree induction algorithm.
8. Consider the decision tree shown in Figure 4.30.
- (a) Compute the generalization error rate of the tree using the optimistic approach.
- (b) Compute the generalization error rate of the tree using the pessimistic approach. (For simplicity, use the strategy of adding a factor of 0.5 to each leaf node.)
- (c) Compute the generalization error rate of the tree using the validation set shown above. This approach is known as **reduced error pruning**.

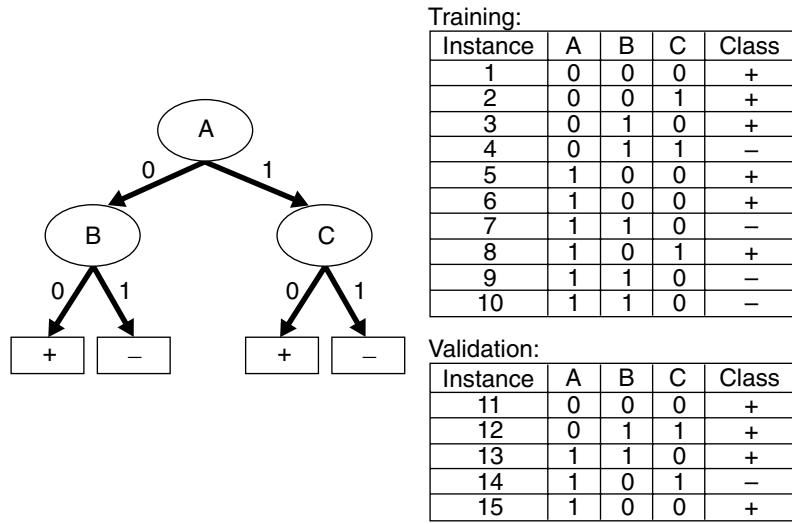


Figure 4.30. Decision tree and data sets for Exercise 8.

9. Consider the decision trees shown in Figure 4.31. Assume they are generated from a data set that contains 16 binary attributes and 3 classes, C_1 , C_2 , and C_3 .

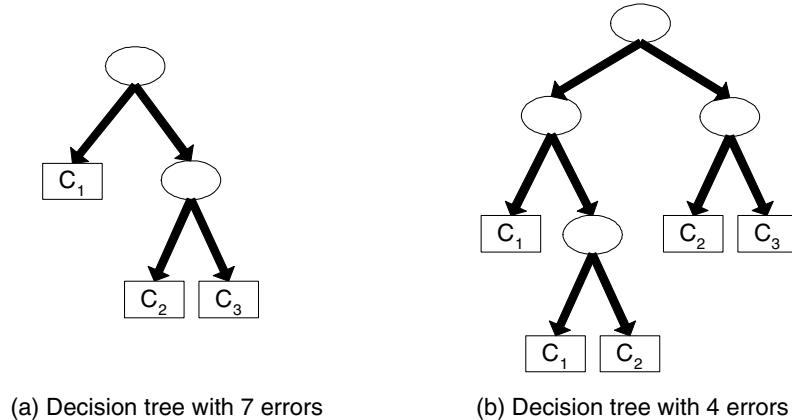


Figure 4.31. Decision trees for Exercise 9.

Compute the total description length of each decision tree according to the minimum description length principle.

- The total description length of a tree is given by:

$$Cost(tree, data) = Cost(tree) + Cost(data|tree).$$

- Each internal node of the tree is encoded by the ID of the splitting attribute. If there are m attributes, the cost of encoding each attribute is $\log_2 m$ bits.
- Each leaf is encoded using the ID of the class it is associated with. If there are k classes, the cost of encoding a class is $\log_2 k$ bits.
- $Cost(tree)$ is the cost of encoding all the nodes in the tree. To simplify the computation, you can assume that the total cost of the tree is obtained by adding up the costs of encoding each internal node and each leaf node.
- $Cost(data|tree)$ is encoded using the classification errors the tree commits on the training set. Each error is encoded by $\log_2 n$ bits, where n is the total number of training instances.

Which decision tree is better, according to the MDL principle?

10. While the .632 bootstrap approach is useful for obtaining a reliable estimate of model accuracy, it has a known limitation [127]. Consider a two-class problem, where there are equal number of positive and negative examples in the data. Suppose the class labels for the examples are generated randomly. The classifier used is an unpruned decision tree (i.e., a perfect memorizer). Determine the accuracy of the classifier using each of the following methods.
 - (a) The holdout method, where two-thirds of the data are used for training and the remaining one-third are used for testing.
 - (b) Ten-fold cross-validation.
 - (c) The .632 bootstrap method.
 - (d) From the results in parts (a), (b), and (c), which method provides a more reliable evaluation of the classifier's accuracy?
11. Consider the following approach for testing whether a classifier A beats another classifier B. Let N be the size of a given data set, p_A be the accuracy of classifier A, p_B be the accuracy of classifier B, and $p = (p_A + p_B)/2$ be the average accuracy for both classifiers. To test whether classifier A is significantly better than B, the following Z-statistic is used:

$$Z = \frac{p_A - p_B}{\sqrt{\frac{2p(1-p)}{N}}}.$$

Classifier A is assumed to be better than classifier B if $Z > 1.96$.

Table 4.9 compares the accuracies of three different classifiers, decision tree classifiers, naïve Bayes classifiers, and support vector machines, on various data sets. (The latter two classifiers are described in Chapter 5.)

Table 4.9. Comparing the accuracy of various classification methods.

Data Set	Size (N)	Decision Tree (%)	naïve Bayes (%)	Support vector machine (%)
Anneal	898	92.09	79.62	87.19
Australia	690	85.51	76.81	84.78
Auto	205	81.95	58.05	70.73
Breast	699	95.14	95.99	96.42
Cleve	303	76.24	83.50	84.49
Credit	690	85.80	77.54	85.07
Diabetes	768	72.40	75.91	76.82
German	1000	70.90	74.70	74.40
Glass	214	67.29	48.59	59.81
Heart	270	80.00	84.07	83.70
Hepatitis	155	81.94	83.23	87.10
Horse	368	85.33	78.80	82.61
Ionosphere	351	89.17	82.34	88.89
Iris	150	94.67	95.33	96.00
Labor	57	78.95	94.74	92.98
Led7	3200	73.34	73.16	73.56
Lymphography	148	77.03	83.11	86.49
Pima	768	74.35	76.04	76.95
Sonar	208	78.85	69.71	76.92
Tic-tac-toe	958	83.72	70.04	98.33
Vehicle	846	71.04	45.04	74.94
Wine	178	94.38	96.63	98.88
Zoo	101	93.07	93.07	96.04

Summarize the performance of the classifiers given in Table 4.9 using the following 3×3 table:

win-loss-draw	Decision tree	Naïve Bayes	Support vector machine
Decision tree	0 - 0 - 23		
Naïve Bayes		0 - 0 - 23	
Support vector machine			0 - 0 - 23

Each cell in the table contains the number of wins, losses, and draws when comparing the classifier in a given row to the classifier in a given column.

12. Let X be a binomial random variable with mean Np and variance $Np(1 - p)$. Show that the ratio X/N also has a binomial distribution with mean p and variance $p(1 - p)/N$.

5

Classification: Alternative Techniques

The previous chapter described a simple, yet quite effective, classification technique known as decision tree induction. Issues such as model overfitting and classifier evaluation were also discussed in great detail. This chapter presents alternative techniques for building classification models—from simple techniques such as rule-based and nearest-neighbor classifiers to more advanced techniques such as support vector machines and ensemble methods. Other key issues such as the class imbalance and multiclass problems are also discussed at the end of the chapter.

5.1 Rule-Based Classifier

A rule-based classifier is a technique for classifying records using a collection of “if . . . then . . .” rules. Table 5.1 shows an example of a model generated by a rule-based classifier for the vertebrate classification problem. The rules for the model are represented in a disjunctive normal form, $R = (r_1 \vee r_2 \vee \dots \vee r_k)$, where R is known as the **rule set** and r_i ’s are the classification rules or disjuncts.

Table 5.1. Example of a rule set for the vertebrate classification problem.

$r_1:$	(Gives Birth = no) \wedge (Aerial Creature = yes) \longrightarrow Birds
$r_2:$	(Gives Birth = no) \wedge (Aquatic Creature = yes) \longrightarrow Fishes
$r_3:$	(Gives Birth = yes) \wedge (Body Temperature = warm-blooded) \longrightarrow Mammals
$r_4:$	(Gives Birth = no) \wedge (Aerial Creature = no) \longrightarrow Reptiles
$r_5:$	(Aquatic Creature = semi) \longrightarrow Amphibians

Each classification rule can be expressed in the following way:

$$r_i : (Condition_i) \longrightarrow y_i. \quad (5.1)$$

The left-hand side of the rule is called the **rule antecedent** or **precondition**. It contains a conjunction of attribute tests:

$$Condition_i = (A_1 \text{ op } v_1) \wedge (A_2 \text{ op } v_2) \wedge \dots (A_k \text{ op } v_k), \quad (5.2)$$

where (A_j, v_j) is an attribute-value pair and op is a logical operator chosen from the set $\{=, \neq, <, >, \leq, \geq\}$. Each attribute test $(A_j \text{ op } v_j)$ is known as a conjunct. The right-hand side of the rule is called the **rule consequent**, which contains the predicted class y_i .

A rule r covers a record x if the precondition of r matches the attributes of x . r is also said to be fired or triggered whenever it covers a given record. For an illustration, consider the rule r_1 given in Table 5.1 and the following attributes for two vertebrates: hawk and grizzly bear.

Name	Body Temperature	Skin Cover	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hibernates
hawk	warm-blooded	feather	no	no	yes	yes	no
grizzly bear	warm-blooded	fur	yes	no	no	yes	yes

r_1 covers the first vertebrate because its precondition is satisfied by the hawk's attributes. The rule does not cover the second vertebrate because grizzly bears give birth to their young and cannot fly, thus violating the precondition of r_1 .

The quality of a classification rule can be evaluated using measures such as coverage and accuracy. Given a data set D and a classification rule $r : A \longrightarrow y$, the coverage of the rule is defined as the fraction of records in D that trigger the rule r . On the other hand, its accuracy or confidence factor is defined as the fraction of records triggered by r whose class labels are equal to y . The formal definitions of these measures are

$$\begin{aligned} \text{Coverage}(r) &= \frac{|A|}{|D|} \\ \text{Accuracy}(r) &= \frac{|A \cap y|}{|A|}, \end{aligned} \quad (5.3)$$

where $|A|$ is the number of records that satisfy the rule antecedent, $|A \cap y|$ is the number of records that satisfy both the antecedent and consequent, and $|D|$ is the total number of records.

Table 5.2. The vertebrate data set.

Name	Body Temperature	Skin Cover	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hibernates	Class Label
human	warm-blooded	hair	yes	no	no	yes	no	Mammals
python	cold-blooded	scales	no	no	no	no	yes	Reptiles
salmon	cold-blooded	scales	no	yes	no	no	no	Fishes
whale	warm-blooded	hair	yes	yes	no	no	no	Mammals
frog	cold-blooded	none	no	semi	no	yes	yes	Amphibians
komodo	cold-blooded	scales	no	no	no	yes	no	Reptiles
dragon								
bat	warm-blooded	hair	yes	no	yes	yes	yes	Mammals
pigeon	warm-blooded	feathers	no	no	yes	yes	no	Birds
cat	warm-blooded	fur	yes	no	no	yes	no	Mammals
guppy	cold-blooded	scales	yes	yes	no	no	no	Fishes
alligator	cold-blooded	scales	no	semi	no	yes	no	Reptiles
penguin	warm-blooded	feathers	no	semi	no	yes	no	Birds
porcupine	warm-blooded	quills	yes	no	no	yes	yes	Mammals
eel	cold-blooded	scales	no	yes	no	no	no	Fishes
salamander	cold-blooded	none	no	semi	no	yes	yes	Amphibians

Example 5.1. Consider the data set shown in Table 5.2. The rule

$$(\text{Gives Birth} = \text{yes}) \wedge (\text{Body Temperature} = \text{warm-blooded}) \longrightarrow \text{Mammals}$$

has a coverage of 33% since five of the fifteen records support the rule antecedent. The rule accuracy is 100% because all five vertebrates covered by the rule are mammals. ■

5.1.1 How a Rule-Based Classifier Works

A rule-based classifier classifies a test record based on the rule triggered by the record. To illustrate how a rule-based classifier works, consider the rule set shown in Table 5.1 and the following vertebrates:

Name	Body Temperature	Skin Cover	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hibernates
lemur	warm-blooded	fur	yes	no	no	yes	yes
turtle	cold-blooded	scales	no	semi	no	yes	no
dogfish shark	cold-blooded	scales	yes	yes	no	no	no

- The first vertebrate, which is a lemur, is warm-blooded and gives birth to its young. It triggers the rule r_3 , and thus, is classified as a mammal.

- The second vertebrate, which is a turtle, triggers the rules r_4 and r_5 . Since the classes predicted by the rules are contradictory (reptiles versus amphibians), their conflicting classes must be resolved.
- None of the rules are applicable to a dogfish shark. In this case, we need to ensure that the classifier can still make a reliable prediction even though a test record is not covered by any rule.

The previous example illustrates two important properties of the rule set generated by a rule-based classifier.

Mutually Exclusive Rules The rules in a rule set R are mutually exclusive if no two rules in R are triggered by the same record. This property ensures that every record is covered by at most one rule in R . An example of a mutually exclusive rule set is shown in Table 5.3.

Exhaustive Rules A rule set R has exhaustive coverage if there is a rule for each combination of attribute values. This property ensures that every record is covered by at least one rule in R . Assuming that **Body Temperature** and **Gives Birth** are binary variables, the rule set shown in Table 5.3 has exhaustive coverage.

Table 5.3. Example of a mutually exclusive and exhaustive rule set.

$r_1:$ (Body Temperature = cold-blooded) \longrightarrow Non-mammals
$r_2:$ (Body Temperature = warm-blooded) \wedge (Gives Birth = yes) \longrightarrow Mammals
$r_3:$ (Body Temperature = warm-blooded) \wedge (Gives Birth = no) \longrightarrow Non-mammals

Together, these properties ensure that every record is covered by exactly one rule. Unfortunately, many rule-based classifiers, including the one shown in Table 5.1, do not have such properties. If the rule set is not exhaustive, then a default rule, $r_d : () \longrightarrow y_d$, must be added to cover the remaining cases. A default rule has an empty antecedent and is triggered when all other rules have failed. y_d is known as the default class and is typically assigned to the majority class of training records not covered by the existing rules.

If the rule set is not mutually exclusive, then a record can be covered by several rules, some of which may predict conflicting classes. There are two ways to overcome this problem.

Ordered Rules In this approach, the rules in a rule set are ordered in decreasing order of their priority, which can be defined in many ways (e.g., based on accuracy, coverage, total description length, or the order in which the rules are generated). An ordered rule set is also known as a **decision list**. When a test record is presented, it is classified by the highest-ranked rule that covers the record. This avoids the problem of having conflicting classes predicted by multiple classification rules.

Unordered Rules This approach allows a test record to trigger multiple classification rules and considers the consequent of each rule as a vote for a particular class. The votes are then tallied to determine the class label of the test record. The record is usually assigned to the class that receives the highest number of votes. In some cases, the vote may be weighted by the rule's accuracy. Using unordered rules to build a rule-based classifier has both advantages and disadvantages. Unordered rules are less susceptible to errors caused by the wrong rule being selected to classify a test record (unlike classifiers based on ordered rules, which are sensitive to the choice of rule-ordering criteria). Model building is also less expensive because the rules do not have to be kept in sorted order. Nevertheless, classifying a test record can be quite an expensive task because the attributes of the test record must be compared against the precondition of every rule in the rule set.

In the remainder of this section, we will focus on rule-based classifiers that use ordered rules.

5.1.2 Rule-Ordering Schemes

Rule ordering can be implemented on a rule-by-rule basis or on a class-by-class basis. The difference between these schemes is illustrated in Figure 5.1.

Rule-Based Ordering Scheme This approach orders the individual rules by some rule quality measure. This ordering scheme ensures that every test record is classified by the “best” rule covering it. A potential drawback of this scheme is that lower-ranked rules are much harder to interpret because they assume the negation of the rules preceding them. For example, the fourth rule shown in Figure 5.1 for rule-based ordering,

$$\text{Aquatic Creature} = \text{semi} \longrightarrow \text{Amphibians},$$

has the following interpretation: If the vertebrate does not have any feathers or cannot fly, and is cold-blooded and semi-aquatic, then it is an amphibian.

Rule-Based Ordering	Class-Based Ordering
(Skin Cover=feathers, Aerial Creature=yes) ==> Birds	(Skin Cover=feathers, Aerial Creature=yes) ==> Birds
(Body temperature=warm-blooded, Gives Birth=yes) ==> Mammals	(Body temperature=warm-blooded, Gives Birth=no) ==> Birds
(Body temperature=warm-blooded, Gives Birth=no) ==> Birds	(Body temperature=warm-blooded, Gives Birth=yes) ==> Mammals
(Aquatic Creature=semi)) ==> Amphibians	(Aquatic Creature=semi)) ==> Amphibians
(Skin Cover=scales, Aquatic Creature=no) ==> Reptiles	(Skin Cover=none) ==> Amphibians
(Skin Cover=scales, Aquatic Creature=yes) ==> Fishes	(Skin Cover=scales, Aquatic Creature=no) ==> Reptiles
(Skin Cover=none) ==> Amphibians	(Skin Cover=scales, Aquatic Creature=yes) ==> Fishes

Figure 5.1. Comparison between rule-based and class-based ordering schemes.

The additional conditions (that the vertebrate does not have any feathers or cannot fly, and is cold-blooded) are due to the fact that the vertebrate does not satisfy the first three rules. If the number of rules is large, interpreting the meaning of the rules residing near the bottom of the list can be a cumbersome task.

Class-Based Ordering Scheme In this approach, rules that belong to the same class appear together in the rule set R . The rules are then collectively sorted on the basis of their class information. The relative ordering among the rules from the same class is not important; as long as one of the rules fires, the class will be assigned to the test record. This makes rule interpretation slightly easier. However, it is possible for a high-quality rule to be overlooked in favor of an inferior rule that happens to predict the higher-ranked class.

Since most of the well-known rule-based classifiers (such as C4.5rules and RIPPER) employ the class-based ordering scheme, the discussion in the remainder of this section focuses mainly on this type of ordering scheme.

5.1.3 How to Build a Rule-Based Classifier

To build a rule-based classifier, we need to extract a set of rules that identifies key relationships between the attributes of a data set and the class label.

There are two broad classes of methods for extracting classification rules: (1) direct methods, which extract classification rules directly from data, and (2) indirect methods, which extract classification rules from other classification models, such as decision trees and neural networks.

Direct methods partition the attribute space into smaller subspaces so that all the records that belong to a subspace can be classified using a single classification rule. Indirect methods use the classification rules to provide a succinct description of more complex classification models. Detailed discussions of these methods are presented in Sections 5.1.4 and 5.1.5, respectively.

5.1.4 Direct Methods for Rule Extraction

The **sequential covering** algorithm is often used to extract rules directly from data. Rules are grown in a greedy fashion based on a certain evaluation measure. The algorithm extracts the rules one class at a time for data sets that contain more than two classes. For the vertebrate classification problem, the sequential covering algorithm may generate rules for classifying birds first, followed by rules for classifying mammals, amphibians, reptiles, and finally, fishes (see Figure 5.1). The criterion for deciding which class should be generated first depends on a number of factors, such as the class prevalence (i.e., fraction of training records that belong to a particular class) or the cost of misclassifying records from a given class.

A summary of the sequential covering algorithm is given in Algorithm 5.1. The algorithm starts with an empty decision list, R . The Learn-One-Rule function is then used to extract the best rule for class y that covers the current set of training records. During rule extraction, all training records for class y are considered to be positive examples, while those that belong to

Algorithm 5.1 Sequential covering algorithm.

- 1: Let E be the training records and A be the set of attribute-value pairs, $\{(A_j, v_j)\}$.
 - 2: Let Y_o be an ordered set of classes $\{y_1, y_2, \dots, y_k\}$.
 - 3: Let $R = \{\}$ be the initial rule list.
 - 4: **for** each class $y \in Y_o - \{y_k\}$ **do**
 - 5: **while** stopping condition is not met **do**
 - 6: $r \leftarrow \text{Learn-One-Rule}(E, A, y)$.
 - 7: Remove training records from E that are covered by r .
 - 8: Add r to the bottom of the rule list: $R \longrightarrow R \vee r$.
 - 9: **end while**
 - 10: **end for**
 - 11: Insert the default rule, $\{\} \longrightarrow y_k$, to the bottom of the rule list R .
-

other classes are considered to be negative examples. A rule is desirable if it covers most of the positive examples and none (or very few) of the negative examples. Once such a rule is found, the training records covered by the rule are eliminated. The new rule is added to the bottom of the decision list R . This procedure is repeated until the stopping criterion is met. The algorithm then proceeds to generate rules for the next class.

Figure 5.2 demonstrates how the sequential covering algorithm works for a data set that contains a collection of positive and negative examples. The rule $R1$, whose coverage is shown in Figure 5.2(b), is extracted first because it covers the largest fraction of positive examples. All the training records covered by $R1$ are subsequently removed and the algorithm proceeds to look for the next best rule, which is $R2$.

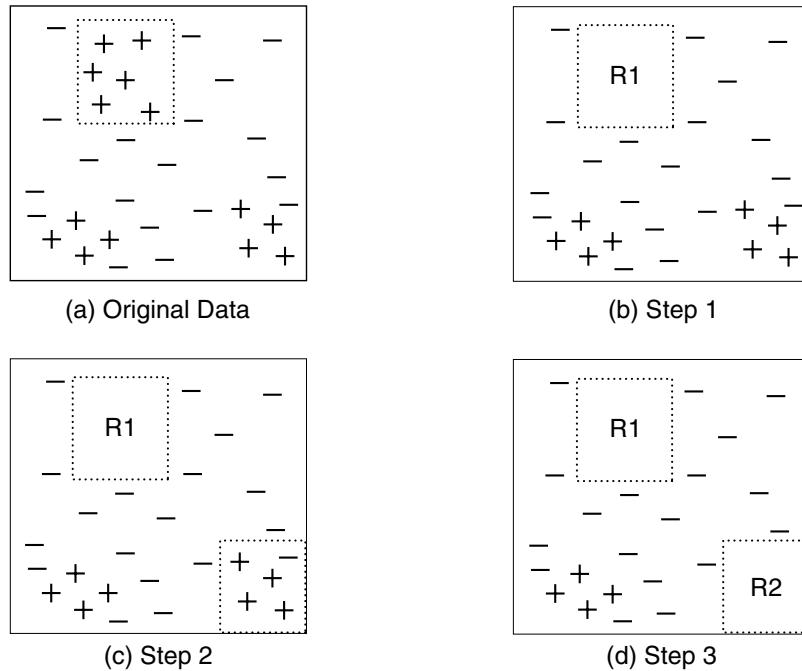


Figure 5.2. An example of the sequential covering algorithm.

Learn-One-Rule Function

The objective of the Learn-One-Rule function is to extract a classification rule that covers many of the positive examples and none (or very few) of the negative examples in the training set. However, finding an optimal rule is computationally expensive given the exponential size of the search space. The Learn-One-Rule function addresses the exponential search problem by growing the rules in a greedy fashion. It generates an initial rule r and keeps refining the rule until a certain stopping criterion is met. The rule is then pruned to improve its generalization error.

Rule-Growing Strategy There are two common strategies for growing a classification rule: general-to-specific or specific-to-general. Under the general-to-specific strategy, an initial rule $r : \{\} \rightarrow y$ is created, where the left-hand side is an empty set and the right-hand side contains the target class. The rule has poor quality because it covers all the examples in the training set. New

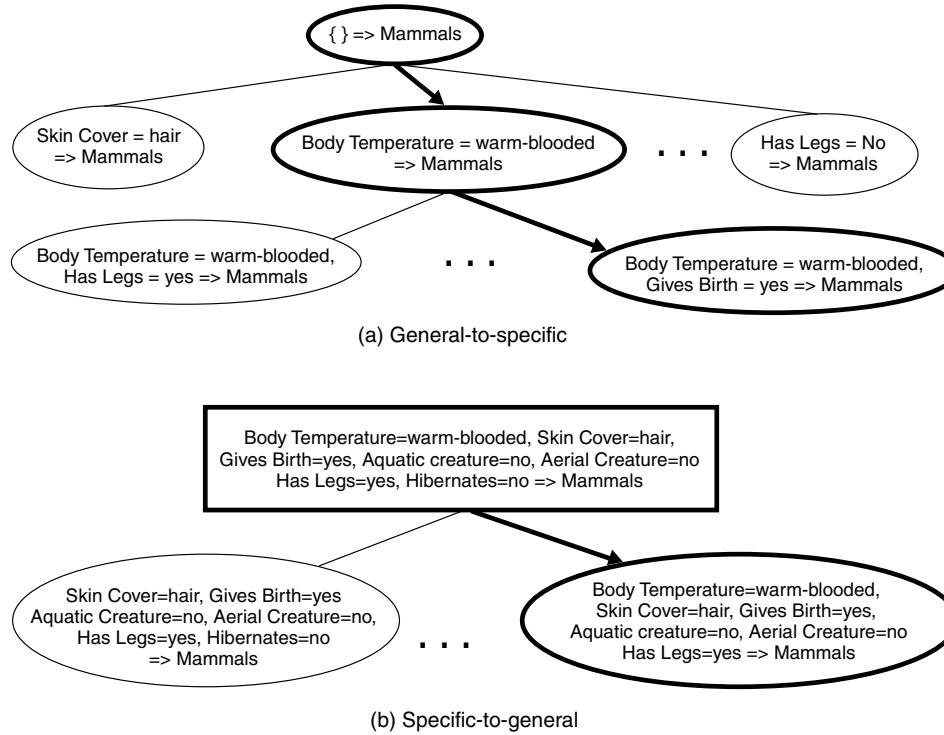


Figure 5.3. General-to-specific and specific-to-general rule-growing strategies.

conjuncts are subsequently added to improve the rule's quality. Figure 5.3(a) shows the general-to-specific rule-growing strategy for the vertebrate classification problem. The conjunct **Body Temperature=warm-blooded** is initially chosen to form the rule antecedent. The algorithm then explores all the possible candidates and greedily chooses the next conjunct, **Gives Birth=yes**, to be added into the rule antecedent. This process continues until the stopping criterion is met (e.g., when the added conjunct does not improve the quality of the rule).

For the specific-to-general strategy, one of the positive examples is randomly chosen as the initial seed for the rule-growing process. During the refinement step, the rule is generalized by removing one of its conjuncts so that it can cover more positive examples. Figure 5.3(b) shows the specific-to-general approach for the vertebrate classification problem. Suppose a positive example for mammals is chosen as the initial seed. The initial rule contains the same conjuncts as the attribute values of the seed. To improve its coverage, the rule is generalized by removing the conjunct **Hibernate=no**. The refinement step is repeated until the stopping criterion is met, e.g., when the rule starts covering negative examples.

The previous approaches may produce suboptimal rules because the rules are grown in a greedy fashion. To avoid this problem, a beam search may be used, where k of the best candidate rules are maintained by the algorithm. Each candidate rule is then grown separately by adding (or removing) a conjunct from its antecedent. The quality of the candidates are evaluated and the k best candidates are chosen for the next iteration.

Rule Evaluation An evaluation metric is needed to determine which conjunct should be added (or removed) during the rule-growing process. Accuracy is an obvious choice because it explicitly measures the fraction of training examples classified correctly by the rule. However, a potential limitation of accuracy is that it does not take into account the rule's coverage. For example, consider a training set that contains 60 positive examples and 100 negative examples. Suppose we are given the following two candidate rules:

- Rule r_1 : covers 50 positive examples and 5 negative examples,
- Rule r_2 : covers 2 positive examples and no negative examples.

The accuracies for r_1 and r_2 are 90.9% and 100%, respectively. However, r_1 is the better rule despite its lower accuracy. The high accuracy for r_2 is potentially spurious because the coverage of the rule is too low.

The following approaches can be used to handle this problem.

1. A statistical test can be used to prune rules that have poor coverage. For example, we may compute the following likelihood ratio statistic:

$$R = 2 \sum_{i=1}^k f_i \log(f_i/e_i),$$

where k is the number of classes, f_i is the observed frequency of class i examples that are covered by the rule, and e_i is the expected frequency of a rule that makes random predictions. Note that R has a chi-square distribution with $k - 1$ degrees of freedom. A large R value suggests that the number of correct predictions made by the rule is significantly larger than that expected by random guessing. For example, since r_1 covers 55 examples, the expected frequency for the positive class is $e_+ = 55 \times 60/160 = 20.625$, while the expected frequency for the negative class is $e_- = 55 \times 100/160 = 34.375$. Thus, the likelihood ratio for r_1 is

$$R(r_1) = 2 \times [50 \times \log_2(50/20.625) + 5 \times \log_2(5/34.375)] = 99.9.$$

Similarly, the expected frequencies for r_2 are $e_+ = 2 \times 60/160 = 0.75$ and $e_- = 2 \times 100/160 = 1.25$. The likelihood ratio statistic for r_2 is

$$R(r_2) = 2 \times [2 \times \log_2(2/0.75) + 0 \times \log_2(0/1.25)] = 5.66.$$

This statistic therefore suggests that r_1 is a better rule than r_2 .

2. An evaluation metric that takes into account the rule coverage can be used. Consider the following evaluation metrics:

$$\text{Laplace} = \frac{f_+ + 1}{n + k}, \quad (5.4)$$

$$\text{m-estimate} = \frac{f_+ + kp_+}{n + k}, \quad (5.5)$$

where n is the number of examples covered by the rule, f_+ is the number of positive examples covered by the rule, k is the total number of classes, and p_+ is the prior probability for the positive class. Note that the m-estimate is equivalent to the Laplace measure by choosing $p_+ = 1/k$. Depending on the rule coverage, these measures capture the trade-off

between rule accuracy and the prior probability of the positive class. If the rule does not cover any training example, then the Laplace measure reduces to $1/k$, which is the prior probability of the positive class assuming a uniform class distribution. The m-estimate also reduces to the prior probability (p_+) when $n = 0$. However, if the rule coverage is large, then both measures asymptotically approach the rule accuracy, f_+/n . Going back to the previous example, the Laplace measure for r_1 is $51/57 = 89.47\%$, which is quite close to its accuracy. Conversely, the Laplace measure for r_2 (75%) is significantly lower than its accuracy because r_2 has a much lower coverage.

3. An evaluation metric that takes into account the support count of the rule can be used. One such metric is the **FOIL's information gain**. The support count of a rule corresponds to the number of positive examples covered by the rule. Suppose the rule $r : A \longrightarrow +$ covers p_0 positive examples and n_0 negative examples. After adding a new conjunct B , the extended rule $r' : A \wedge B \longrightarrow +$ covers p_1 positive examples and n_1 negative examples. Given this information, the FOIL's information gain of the extended rule is defined as follows:

$$\text{FOIL's information gain} = p_1 \times \left(\log_2 \frac{p_1}{p_1 + n_1} - \log_2 \frac{p_0}{p_0 + n_0} \right). \quad (5.6)$$

Since the measure is proportional to p_1 and $p_1/(p_1 + n_1)$, it prefers rules that have high support count and accuracy. The FOIL's information gains for rules r_1 and r_2 given in the preceding example are 43.12 and 2, respectively. Therefore, r_1 is a better rule than r_2 .

Rule Pruning The rules generated by the Learn-One-Rule function can be pruned to improve their generalization errors. To determine whether pruning is necessary, we may apply the methods described in Section 4.4 on page 172 to estimate the generalization error of a rule. For example, if the error on validation set decreases after pruning, we should keep the simplified rule. Another approach is to compare the pessimistic error of the rule before and after pruning (see Section 4.4.4 on page 179). The simplified rule is retained in place of the original rule if the pessimistic error improves after pruning.

Rationale for Sequential Covering

After a rule is extracted, the sequential covering algorithm must eliminate all the positive and negative examples covered by the rule. The rationale for doing this is given in the next example.

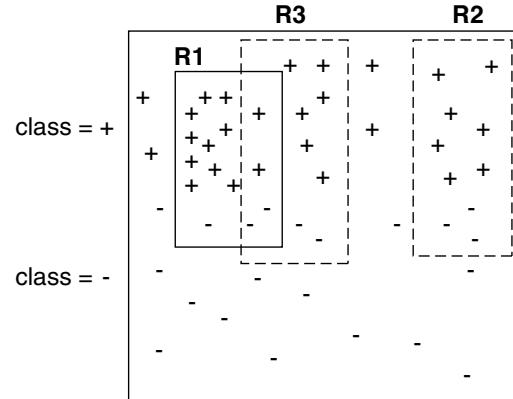


Figure 5.4. Elimination of training records by the sequential covering algorithm. $R1$, $R2$, and $R3$ represent regions covered by three different rules.

Figure 5.4 shows three possible rules, $R1$, $R2$, and $R3$, extracted from a data set that contains 29 positive examples and 21 negative examples. The accuracies of $R1$, $R2$, and $R3$ are 12/15 (80%), 7/10 (70%), and 8/12 (66.7%), respectively. $R1$ is generated first because it has the highest accuracy. After generating $R1$, it is clear that the positive examples covered by the rule must be removed so that the next rule generated by the algorithm is different than $R1$. Next, suppose the algorithm is given the choice of generating either $R2$ or $R3$. Even though $R2$ has higher accuracy than $R3$, $R1$ and $R3$ together cover 18 positive examples and 5 negative examples (resulting in an overall accuracy of 78.3%), whereas $R1$ and $R2$ together cover 19 positive examples and 6 negative examples (resulting in an overall accuracy of 76%). The incremental impact of $R2$ or $R3$ on accuracy is more evident when the positive and negative examples covered by $R1$ are removed before computing their accuracies. In particular, if positive examples covered by $R1$ are not removed, then we may overestimate the effective accuracy of $R3$, and if negative examples are not removed, then we may underestimate the accuracy of $R3$. In the latter case, we might end up preferring $R2$ over $R3$ even though half of the false positive errors committed by $R3$ have already been accounted for by the preceding rule, $R1$.

RIPPER Algorithm

To illustrate the direct method, we consider a widely used rule induction algorithm called RIPPER. This algorithm scales almost linearly with the number of training examples and is particularly suited for building models from data sets with imbalanced class distributions. RIPPER also works well with noisy data sets because it uses a validation set to prevent model overfitting.

For two-class problems, RIPPER chooses the majority class as its default class and learns the rules for detecting the minority class. For multiclass problems, the classes are ordered according to their frequencies. Let (y_1, y_2, \dots, y_c) be the ordered classes, where y_1 is the least frequent class and y_c is the most frequent class. During the first iteration, instances that belong to y_1 are labeled as positive examples, while those that belong to other classes are labeled as negative examples. The sequential covering method is used to generate rules that discriminate between the positive and negative examples. Next, RIPPER extracts rules that distinguish y_2 from other remaining classes. This process is repeated until we are left with y_c , which is designated as the default class.

Rule Growing RIPPER employs a general-to-specific strategy to grow a rule and the FOIL's information gain measure to choose the best conjunct to be added into the rule antecedent. It stops adding conjuncts when the rule starts covering negative examples. The new rule is then pruned based on its performance on the validation set. The following metric is computed to determine whether pruning is needed: $(p-n)/(p+n)$, where p (n) is the number of positive (negative) examples in the validation set covered by the rule. This metric is monotonically related to the rule's accuracy on the validation set. If the metric improves after pruning, then the conjunct is removed. Pruning is done starting from the last conjunct added to the rule. For example, given a rule $ABCD \rightarrow y$, RIPPER checks whether D should be pruned first, followed by CD , BCD , etc. While the original rule covers only positive examples, the pruned rule may cover some of the negative examples in the training set.

Building the Rule Set After generating a rule, all the positive and negative examples covered by the rule are eliminated. The rule is then added into the rule set as long as it does not violate the stopping condition, which is based on the minimum description length principle. If the new rule increases the total description length of the rule set by at least d bits, then RIPPER stops adding rules into its rule set (by default, d is chosen to be 64 bits). Another stopping condition used by RIPPER is that the error rate of the rule on the validation set must not exceed 50%.

RIPPER also performs additional optimization steps to determine whether some of the existing rules in the rule set can be replaced by better alternative rules. Readers who are interested in the details of the optimization method may refer to the reference cited at the end of this chapter.

5.1.5 Indirect Methods for Rule Extraction

This section presents a method for generating a rule set from a decision tree. In principle, every path from the root node to the leaf node of a decision tree can be expressed as a classification rule. The test conditions encountered along the path form the conjuncts of the rule antecedent, while the class label at the leaf node is assigned to the rule consequent. Figure 5.5 shows an example of a rule set generated from a decision tree. Notice that the rule set is exhaustive and contains mutually exclusive rules. However, some of the rules can be simplified as shown in the next example.

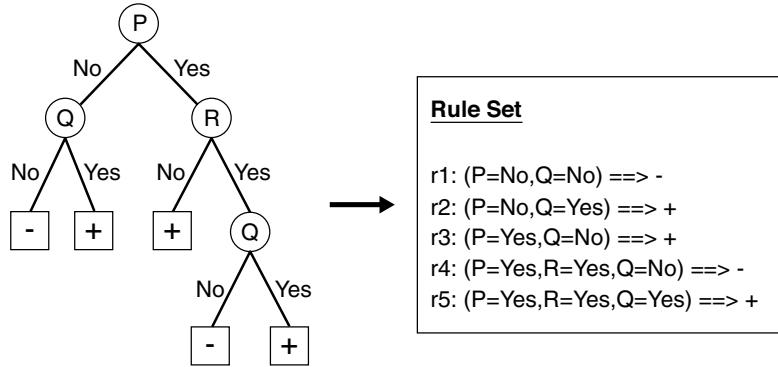


Figure 5.5. Converting a decision tree into classification rules.

Example 5.2. Consider the following three rules from Figure 5.5:

- $r2 : (P = \text{No}) \wedge (Q = \text{Yes}) \longrightarrow +$
- $r3 : (P = \text{Yes}) \wedge (R = \text{No}) \longrightarrow +$
- $r5 : (P = \text{Yes}) \wedge (R = \text{Yes}) \wedge (Q = \text{Yes}) \longrightarrow +$

Observe that the rule set always predicts a positive class when the value of Q is Yes. Therefore, we may simplify the rules as follows:

- $r2' : (Q = \text{Yes}) \longrightarrow +$
- $r3 : (P = \text{Yes}) \wedge (R = \text{No}) \longrightarrow +.$

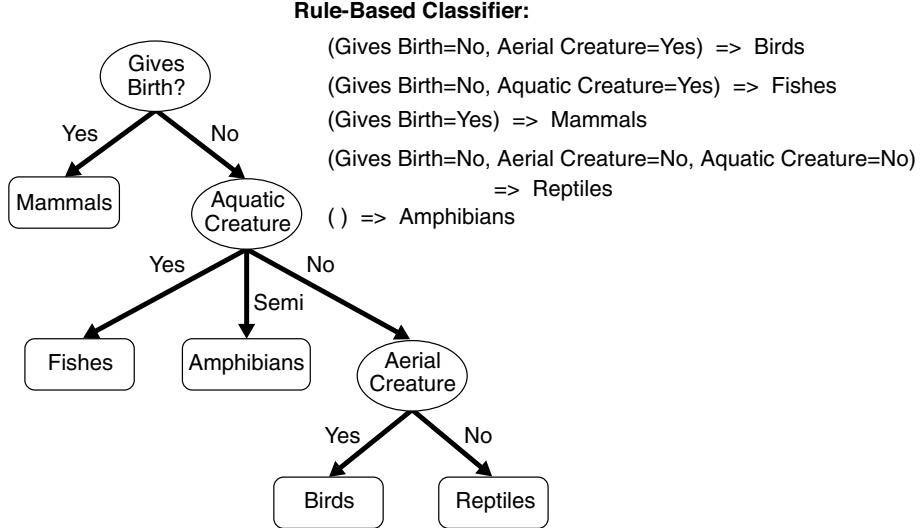


Figure 5.6. Classification rules extracted from a decision tree for the vertebrate classification problem.

r_3 is retained to cover the remaining instances of the positive class. Although the rules obtained after simplification are no longer mutually exclusive, they are less complex and are easier to interpret. ■

In the following, we describe an approach used by the C4.5rules algorithm to generate a rule set from a decision tree. Figure 5.6 shows the decision tree and resulting classification rules obtained for the data set given in Table 5.2.

Rule Generation Classification rules are extracted for every path from the root to one of the leaf nodes in the decision tree. Given a classification rule $r : A \rightarrow y$, we consider a simplified rule, $r' : A' \rightarrow y$, where A' is obtained by removing one of the conjuncts in A . The simplified rule with the lowest pessimistic error rate is retained provided its error rate is less than that of the original rule. The rule-pruning step is repeated until the pessimistic error of the rule cannot be improved further. Because some of the rules may become identical after pruning, the duplicate rules must be discarded.

Rule Ordering After generating the rule set, C4.5rules uses the class-based ordering scheme to order the extracted rules. Rules that predict the same class are grouped together into the same subset. The total description length for each subset is computed, and the classes are arranged in increasing order of their total description length. The class that has the smallest description

length is given the highest priority because it is expected to contain the best set of rules. The total description length for a class is given by $L_{\text{exception}} + g \times L_{\text{model}}$, where $L_{\text{exception}}$ is the number of bits needed to encode the misclassified examples, L_{model} is the number of bits needed to encode the model, and g is a tuning parameter whose default value is 0.5. The tuning parameter depends on the number of redundant attributes present in the model. The value of the tuning parameter is small if the model contains many redundant attributes.

5.1.6 Characteristics of Rule-Based Classifiers

A rule-based classifier has the following characteristics:

- The expressiveness of a rule set is almost equivalent to that of a decision tree because a decision tree can be represented by a set of mutually exclusive and exhaustive rules. Both rule-based and decision tree classifiers create rectilinear partitions of the attribute space and assign a class to each partition. Nevertheless, if the rule-based classifier allows multiple rules to be triggered for a given record, then a more complex decision boundary can be constructed.
- Rule-based classifiers are generally used to produce descriptive models that are easier to interpret, but gives comparable performance to the decision tree classifier.
- The class-based ordering approach adopted by many rule-based classifiers (such as RIPPER) is well suited for handling data sets with imbalanced class distributions.

5.2 Nearest-Neighbor classifiers

The classification framework shown in Figure 4.3 involves a two-step process: (1) an inductive step for constructing a classification model from data, and (2) a deductive step for applying the model to test examples. Decision tree and rule-based classifiers are examples of **eager learners** because they are designed to learn a model that maps the input attributes to the class label as soon as the training data becomes available. An opposite strategy would be to delay the process of modeling the training data until it is needed to classify the test examples. Techniques that employ this strategy are known as **lazy learners**. An example of a lazy learner is the **Rote classifier**, which memorizes the entire training data and performs classification only if the attributes of a test instance match one of the training examples exactly. An obvious drawback of

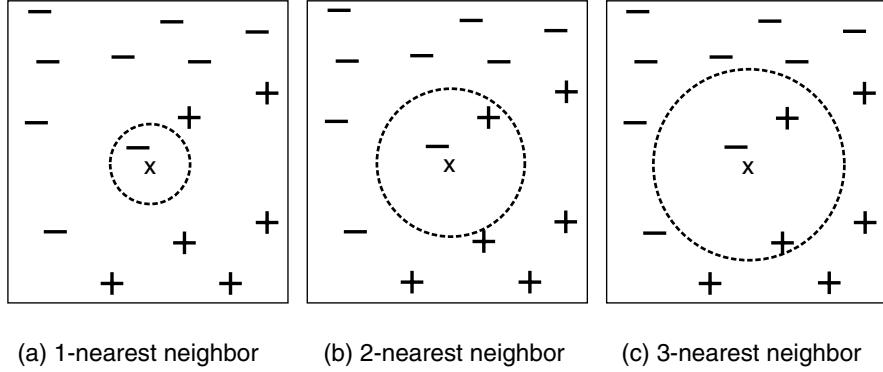


Figure 5.7. The 1-, 2-, and 3-nearest neighbors of an instance.

this approach is that some test records may not be classified because they do not match any training example.

One way to make this approach more flexible is to find all the training examples that are relatively similar to the attributes of the test example. These examples, which are known as **nearest neighbors**, can be used to determine the class label of the test example. The justification for using nearest neighbors is best exemplified by the following saying: “*If it walks like a duck, quacks like a duck, and looks like a duck, then it’s probably a duck.*” A nearest-neighbor classifier represents each example as a data point in a d -dimensional space, where d is the number of attributes. Given a test example, we compute its proximity to the rest of the data points in the training set, using one of the proximity measures described in Section 2.4 on page 65. The k -nearest neighbors of a given example z refer to the k points that are closest to z .

Figure 5.7 illustrates the 1-, 2-, and 3-nearest neighbors of a data point located at the center of each circle. The data point is classified based on the class labels of its neighbors. In the case where the neighbors have more than one label, the data point is assigned to the majority class of its nearest neighbors. In Figure 5.7(a), the 1-nearest neighbor of the data point is a negative example. Therefore the data point is assigned to the negative class. If the number of nearest neighbors is three, as shown in Figure 5.7(c), then the neighborhood contains two positive examples and one negative example. Using the majority voting scheme, the data point is assigned to the positive class. In the case where there is a tie between the classes (see Figure 5.7(b)), we may randomly choose one of them to classify the data point.

The preceding discussion underscores the importance of choosing the right value for k . If k is too small, then the nearest-neighbor classifier may be

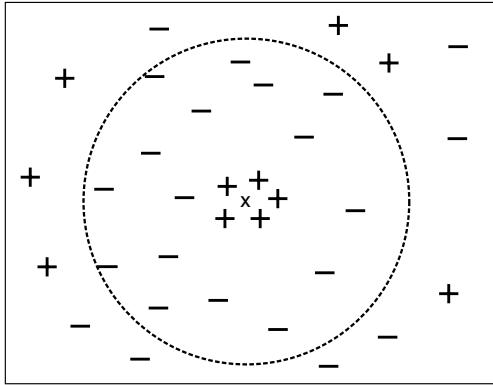


Figure 5.8. k -nearest neighbor classification with large k .

susceptible to overfitting because of noise in the training data. On the other hand, if k is too large, the nearest-neighbor classifier may misclassify the test instance because its list of nearest neighbors may include data points that are located far away from its neighborhood (see Figure 5.8).

5.2.1 Algorithm

A high-level summary of the nearest-neighbor classification method is given in Algorithm 5.2. The algorithm computes the distance (or similarity) between each test example $z = (\mathbf{x}', y')$ and all the training examples $(\mathbf{x}, y) \in D$ to determine its nearest-neighbor list, D_z . Such computation can be costly if the number of training examples is large. However, efficient indexing techniques are available to reduce the amount of computations needed to find the nearest neighbors of a test example.

Algorithm 5.2 The k -nearest neighbor classification algorithm.

- 1: Let k be the number of nearest neighbors and D be the set of training examples.
 - 2: **for** each test example $z = (\mathbf{x}', y')$ **do**
 - 3: Compute $d(\mathbf{x}', \mathbf{x})$, the distance between z and every example, $(\mathbf{x}, y) \in D$.
 - 4: Select $D_z \subseteq D$, the set of k closest training examples to z .
 - 5: $y' = \operatorname{argmax}_v \sum_{(\mathbf{x}_i, y_i) \in D_z} I(v = y_i)$
 - 6: **end for**
-

Once the nearest-neighbor list is obtained, the test example is classified based on the majority class of its nearest neighbors:

$$\text{Majority Voting: } y' = \operatorname{argmax}_v \sum_{(\mathbf{x}_i, y_i) \in D_z} I(v = y_i), \quad (5.7)$$

where v is a class label, y_i is the class label for one of the nearest neighbors, and $I(\cdot)$ is an indicator function that returns the value 1 if its argument is true and 0 otherwise.

In the majority voting approach, every neighbor has the same impact on the classification. This makes the algorithm sensitive to the choice of k , as shown in Figure 5.7. One way to reduce the impact of k is to weight the influence of each nearest neighbor \mathbf{x}_i according to its distance: $w_i = 1/d(\mathbf{x}', \mathbf{x}_i)^2$. As a result, training examples that are located far away from z have a weaker impact on the classification compared to those that are located close to z . Using the distance-weighted voting scheme, the class label can be determined as follows:

$$\text{Distance-Weighted Voting: } y' = \operatorname{argmax}_v \sum_{(\mathbf{x}_i, y_i) \in D_z} w_i \times I(v = y_i). \quad (5.8)$$

5.2.2 Characteristics of Nearest-Neighbor Classifiers

The characteristics of the nearest-neighbor classifier are summarized below:

- Nearest-neighbor classification is part of a more general technique known as instance-based learning, which uses specific training instances to make predictions without having to maintain an abstraction (or model) derived from data. Instance-based learning algorithms require a proximity measure to determine the similarity or distance between instances and a classification function that returns the predicted class of a test instance based on its proximity to other instances.
- Lazy learners such as nearest-neighbor classifiers do not require model building. However, classifying a test example can be quite expensive because we need to compute the proximity values individually between the test and training examples. In contrast, eager learners often spend the bulk of their computing resources for model building. Once a model has been built, classifying a test example is extremely fast.
- Nearest-neighbor classifiers make their predictions based on local information, whereas decision tree and rule-based classifiers attempt to find

a global model that fits the entire input space. Because the classification decisions are made locally, nearest-neighbor classifiers (with small values of k) are quite susceptible to noise.

- Nearest-neighbor classifiers can produce arbitrarily shaped decision boundaries. Such boundaries provide a more flexible model representation compared to decision tree and rule-based classifiers that are often constrained to rectilinear decision boundaries. The decision boundaries of nearest-neighbor classifiers also have high variability because they depend on the composition of training examples. Increasing the number of nearest neighbors may reduce such variability.
- Nearest-neighbor classifiers can produce wrong predictions unless the appropriate proximity measure and data preprocessing steps are taken. For example, suppose we want to classify a group of people based on attributes such as height (measured in meters) and weight (measured in pounds). The height attribute has a low variability, ranging from 1.5 m to 1.85 m, whereas the weight attribute may vary from 90 lb. to 250 lb. If the scale of the attributes are not taken into consideration, the proximity measure may be dominated by differences in the weights of a person.

5.3 Bayesian Classifiers

In many applications the relationship between the attribute set and the class variable is non-deterministic. In other words, the class label of a test record cannot be predicted with certainty even though its attribute set is identical to some of the training examples. This situation may arise because of noisy data or the presence of certain confounding factors that affect classification but are not included in the analysis. For example, consider the task of predicting whether a person is at risk for heart disease based on the person's diet and workout frequency. Although most people who eat healthily and exercise regularly have less chance of developing heart disease, they may still do so because of other factors such as heredity, excessive smoking, and alcohol abuse. Determining whether a person's diet is healthy or the workout frequency is sufficient is also subject to interpretation, which in turn may introduce uncertainties into the learning problem.

This section presents an approach for modeling probabilistic relationships between the attribute set and the class variable. The section begins with an introduction to the **Bayes theorem**, a statistical principle for combining prior

knowledge of the classes with new evidence gathered from data. The use of the Bayes theorem for solving classification problems will be explained, followed by a description of two implementations of Bayesian classifiers: naïve Bayes and the Bayesian belief network.

5.3.1 Bayes Theorem

Consider a football game between two rival teams: Team 0 and Team 1. Suppose Team 0 wins 65% of the time and Team 1 wins the remaining matches. Among the games won by Team 0, only 30% of them come from playing on Team 1's football field. On the other hand, 75% of the victories for Team 1 are obtained while playing at home. If Team 1 is to host the next match between the two teams, which team will most likely emerge as the winner?

This question can be answered by using the well-known Bayes theorem. For completeness, we begin with some basic definitions from probability theory. Readers who are unfamiliar with concepts in probability may refer to Appendix C for a brief review of this topic.

Let X and Y be a pair of random variables. Their joint probability, $P(X = x, Y = y)$, refers to the probability that variable X will take on the value x and variable Y will take on the value y . A conditional probability is the probability that a random variable will take on a particular value given that the outcome for another random variable is known. For example, the conditional probability $P(Y = y|X = x)$ refers to the probability that the variable Y will take on the value y , given that the variable X is observed to have the value x . The joint and conditional probabilities for X and Y are related in the following way:

$$P(X, Y) = P(Y|X) \times P(X) = P(X|Y) \times P(Y). \quad (5.9)$$

Rearranging the last two expressions in Equation 5.9 leads to the following formula, known as the Bayes theorem:

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}. \quad (5.10)$$

The Bayes theorem can be used to solve the prediction problem stated at the beginning of this section. For notational convenience, let X be the random variable that represents the team hosting the match and Y be the random variable that represents the winner of the match. Both X and Y can

take on values from the set $\{0, 1\}$. We can summarize the information given in the problem as follows:

Probability Team 0 wins is $P(Y = 0) = 0.65$.

Probability Team 1 wins is $P(Y = 1) = 1 - P(Y = 0) = 0.35$.

Probability Team 1 hosted the match it won is $P(X = 1|Y = 1) = 0.75$.

Probability Team 1 hosted the match won by Team 0 is $P(X = 1|Y = 0) = 0.3$.

Our objective is to compute $P(Y = 1|X = 1)$, which is the conditional probability that Team 1 wins the next match it will be hosting, and compares it against $P(Y = 0|X = 1)$. Using the Bayes theorem, we obtain

$$\begin{aligned} P(Y = 1|X = 1) &= \frac{P(X = 1|Y = 1) \times P(Y = 1)}{P(X = 1)} \\ &= \frac{P(X = 1|Y = 1) \times P(Y = 1)}{P(X = 1, Y = 1) + P(X = 1, Y = 0)} \\ &= \frac{P(X = 1|Y = 1) \times P(Y = 1)}{P(X = 1|Y = 1)P(Y = 1) + P(X = 1|Y = 0)P(Y = 0)} \\ &= \frac{0.75 \times 0.35}{0.75 \times 0.35 + 0.3 \times 0.65} \\ &= 0.5738, \end{aligned}$$

where the law of total probability (see Equation C.5 on page 722) was applied in the second line. Furthermore, $P(Y = 0|X = 1) = 1 - P(Y = 1|X = 1) = 0.4262$. Since $P(Y = 1|X = 1) > P(Y = 0|X = 1)$, Team 1 has a better chance than Team 0 of winning the next match.

5.3.2 Using the Bayes Theorem for Classification

Before describing how the Bayes theorem can be used for classification, let us formalize the classification problem from a statistical perspective. Let \mathbf{X} denote the attribute set and Y denote the class variable. If the class variable has a non-deterministic relationship with the attributes, then we can treat \mathbf{X} and Y as random variables and capture their relationship probabilistically using $P(Y|\mathbf{X})$. This conditional probability is also known as the **posterior probability** for Y , as opposed to its **prior probability**, $P(Y)$.

During the training phase, we need to learn the posterior probabilities $P(Y|\mathbf{X})$ for every combination of \mathbf{X} and Y based on information gathered from the training data. By knowing these probabilities, a test record \mathbf{X}' can be classified by finding the class Y' that maximizes the posterior probability,

$P(Y'|\mathbf{X}')$. To illustrate this approach, consider the task of predicting whether a loan borrower will default on their payments. Figure 5.9 shows a training set with the following attributes: **Home Owner**, **Marital Status**, and **Annual Income**. Loan borrowers who defaulted on their payments are classified as **Yes**, while those who repaid their loans are classified as **No**.

Tid	Home Owner	Marital Status	Annual Income	Defaulted Borrower
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Figure 5.9. Training set for predicting the loan default problem.

Suppose we are given a test record with the following attribute set: $\mathbf{X} = (\text{Home Owner} = \text{No}, \text{Marital Status} = \text{Married}, \text{Annual Income} = \$120\text{K})$. To classify the record, we need to compute the posterior probabilities $P(\text{Yes}|\mathbf{X})$ and $P(\text{No}|\mathbf{X})$ based on information available in the training data. If $P(\text{Yes}|\mathbf{X}) > P(\text{No}|\mathbf{X})$, then the record is classified as **Yes**; otherwise, it is classified as **No**.

Estimating the posterior probabilities accurately for every possible combination of class label and attribute value is a difficult problem because it requires a very large training set, even for a moderate number of attributes. The Bayes theorem is useful because it allows us to express the posterior probability in terms of the prior probability $P(Y)$, the **class-conditional** probability $P(\mathbf{X}|Y)$, and the evidence, $P(\mathbf{X})$:

$$P(Y|\mathbf{X}) = \frac{P(\mathbf{X}|Y) \times P(Y)}{P(\mathbf{X})}. \quad (5.11)$$

When comparing the posterior probabilities for different values of Y , the denominator term, $P(\mathbf{X})$, is always constant, and thus, can be ignored. The

prior probability $P(Y)$ can be easily estimated from the training set by computing the fraction of training records that belong to each class. To estimate the class-conditional probabilities $P(\mathbf{X}|Y)$, we present two implementations of Bayesian classification methods: the naïve Bayes classifier and the Bayesian belief network. These implementations are described in Sections 5.3.3 and 5.3.5, respectively.

5.3.3 Naïve Bayes Classifier

A naïve Bayes classifier estimates the class-conditional probability by assuming that the attributes are conditionally independent, given the class label y . The conditional independence assumption can be formally stated as follows:

$$P(\mathbf{X}|Y = y) = \prod_{i=1}^d P(X_i|Y = y), \quad (5.12)$$

where each attribute set $\mathbf{X} = \{X_1, X_2, \dots, X_d\}$ consists of d attributes.

Conditional Independence

Before delving into the details of how a naïve Bayes classifier works, let us examine the notion of conditional independence. Let \mathbf{X} , \mathbf{Y} , and \mathbf{Z} denote three sets of random variables. The variables in \mathbf{X} are said to be conditionally independent of \mathbf{Y} , given \mathbf{Z} , if the following condition holds:

$$P(\mathbf{X}|\mathbf{Y}, \mathbf{Z}) = P(\mathbf{X}|\mathbf{Z}). \quad (5.13)$$

An example of conditional independence is the relationship between a person's arm length and his or her reading skills. One might observe that people with longer arms tend to have higher levels of reading skills. This relationship can be explained by the presence of a confounding factor, which is age. A young child tends to have short arms and lacks the reading skills of an adult. If the age of a person is fixed, then the observed relationship between arm length and reading skills disappears. Thus, we can conclude that arm length and reading skills are conditionally independent when the age variable is fixed.

The conditional independence between \mathbf{X} and \mathbf{Y} can also be written into a form that looks similar to Equation 5.12:

$$\begin{aligned}
 P(\mathbf{X}, \mathbf{Y}|\mathbf{Z}) &= \frac{P(\mathbf{X}, \mathbf{Y}, \mathbf{Z})}{P(\mathbf{Z})} \\
 &= \frac{P(\mathbf{X}, \mathbf{Y}, \mathbf{Z})}{P(\mathbf{Y}, \mathbf{Z})} \times \frac{P(\mathbf{Y}, \mathbf{Z})}{P(\mathbf{Z})} \\
 &= P(\mathbf{X}|\mathbf{Y}, \mathbf{Z}) \times P(\mathbf{Y}|\mathbf{Z}) \\
 &= P(\mathbf{X}|\mathbf{Z}) \times P(\mathbf{Y}|\mathbf{Z}),
 \end{aligned} \tag{5.14}$$

where Equation 5.13 was used to obtain the last line of Equation 5.14.

How a Naïve Bayes Classifier Works

With the conditional independence assumption, instead of computing the class-conditional probability for every combination of \mathbf{X} , we only have to estimate the conditional probability of each X_i , given Y . The latter approach is more practical because it does not require a very large training set to obtain a good estimate of the probability.

To classify a test record, the naïve Bayes classifier computes the posterior probability for each class Y :

$$P(Y|\mathbf{X}) = \frac{P(Y) \prod_{i=1}^d P(X_i|Y)}{P(\mathbf{X})}. \tag{5.15}$$

Since $P(\mathbf{X})$ is fixed for every Y , it is sufficient to choose the class that maximizes the numerator term, $P(Y) \prod_{i=1}^d P(X_i|Y)$. In the next two subsections, we describe several approaches for estimating the conditional probabilities $P(X_i|Y)$ for categorical and continuous attributes.

Estimating Conditional Probabilities for Categorical Attributes

For a categorical attribute X_i , the conditional probability $P(X_i = x_i|Y = y)$ is estimated according to the fraction of training instances in class y that take on a particular attribute value x_i . For example, in the training set given in Figure 5.9, three out of the seven people who repaid their loans also own a home. As a result, the conditional probability for $P(\text{Home Owner}=\text{Yes}|\text{No})$ is equal to 3/7. Similarly, the conditional probability for defaulted borrowers who are single is given by $P(\text{Marital Status} = \text{Single}|\text{Yes}) = 2/3$.

Estimating Conditional Probabilities for Continuous Attributes

There are two ways to estimate the class-conditional probabilities for continuous attributes in naïve Bayes classifiers:

1. We can discretize each continuous attribute and then replace the continuous attribute value with its corresponding discrete interval. This approach transforms the continuous attributes into ordinal attributes. The conditional probability $P(X_i|Y = y)$ is estimated by computing the fraction of training records belonging to class y that falls within the corresponding interval for X_i . The estimation error depends on the discretization strategy (as described in Section 2.3.6 on page 57), as well as the number of discrete intervals. If the number of intervals is too large, there are too few training records in each interval to provide a reliable estimate for $P(X_i|Y)$. On the other hand, if the number of intervals is too small, then some intervals may aggregate records from different classes and we may miss the correct decision boundary.
2. We can assume a certain form of probability distribution for the continuous variable and estimate the parameters of the distribution using the training data. A Gaussian distribution is usually chosen to represent the class-conditional probability for continuous attributes. The distribution is characterized by two parameters, its mean, μ , and variance, σ^2 . For each class y_j , the class-conditional probability for attribute X_i is

$$P(X_i = x_i|Y = y_j) = \frac{1}{\sqrt{2\pi}\sigma_{ij}} \exp^{-\frac{(x_i - \mu_{ij})^2}{2\sigma_{ij}^2}}. \quad (5.16)$$

The parameter μ_{ij} can be estimated based on the sample mean of X_i (\bar{x}) for all training records that belong to the class y_j . Similarly, σ_{ij}^2 can be estimated from the sample variance (s^2) of such training records. For example, consider the annual income attribute shown in Figure 5.9. The sample mean and variance for this attribute with respect to the class No are

$$\begin{aligned} \bar{x} &= \frac{125 + 100 + 70 + \dots + 75}{7} = 110 \\ s^2 &= \frac{(125 - 110)^2 + (100 - 110)^2 + \dots + (75 - 110)^2}{7(6)} = 2975 \\ s &= \sqrt{2975} = 54.54. \end{aligned}$$

Given a test record with taxable income equal to \$120K, we can compute its class-conditional probability as follows:

$$P(\text{Income}=120|\text{No}) = \frac{1}{\sqrt{2\pi}(54.54)} \exp^{-\frac{(120-110)^2}{2\times 2975}} = 0.0072.$$

Note that the preceding interpretation of class-conditional probability is somewhat misleading. The right-hand side of Equation 5.16 corresponds to a **probability density function**, $f(X_i; \mu_{ij}, \sigma_{ij})$. Since the function is continuous, the probability that the random variable X_i takes a particular value is zero. Instead, we should compute the conditional probability that X_i lies within some interval, x_i and $x_i + \epsilon$, where ϵ is a small constant:

$$\begin{aligned} P(x_i \leq X_i \leq x_i + \epsilon | Y = y_j) &= \int_{x_i}^{x_i + \epsilon} f(X_i; \mu_{ij}, \sigma_{ij}) dX_i \\ &\approx f(x_i; \mu_{ij}, \sigma_{ij}) \times \epsilon. \end{aligned} \quad (5.17)$$

Since ϵ appears as a constant multiplicative factor for each class, it cancels out when we normalize the posterior probability for $P(Y|\mathbf{X})$. Therefore, we can still apply Equation 5.16 to approximate the class-conditional probability $P(X_i|Y)$.

Example of the Naïve Bayes Classifier

Consider the data set shown in Figure 5.10(a). We can compute the class-conditional probability for each categorical attribute, along with the sample mean and variance for the continuous attribute using the methodology described in the previous subsections. These probabilities are summarized in Figure 5.10(b).

To predict the class label of a test record $\mathbf{X} = (\text{Home Owner}=\text{No}, \text{Marital Status} = \text{Married}, \text{Income} = \$120K)$, we need to compute the posterior probabilities $P(\text{No}|\mathbf{X})$ and $P(\text{Yes}|\mathbf{X})$. Recall from our earlier discussion that these posterior probabilities can be estimated by computing the product between the prior probability $P(Y)$ and the class-conditional probabilities $\prod_i P(X_i|Y)$, which corresponds to the numerator of the right-hand side term in Equation 5.15.

The prior probabilities of each class can be estimated by calculating the fraction of training records that belong to each class. Since there are three records that belong to the class **Yes** and seven records that belong to the class

Tid	Home Owner	Marital Status	Annual Income	Defaulted Borrower
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

(a)

$P(\text{Home Owner}=\text{Yes}|\text{No}) = 3/7$
 $P(\text{Home Owner}=\text{No}|\text{No}) = 4/7$
 $P(\text{Home Owner}=\text{Yes}|\text{Yes}) = 0$
 $P(\text{Home Owner}=\text{No}|\text{Yes}) = 1$
 $P(\text{Marital Status}=\text{Single}|\text{No}) = 2/7$
 $P(\text{Marital Status}=\text{Divorced}|\text{No}) = 1/7$
 $P(\text{Marital Status}=\text{Married}|\text{No}) = 4/7$
 $P(\text{Marital Status}=\text{Single}|\text{Yes}) = 2/3$
 $P(\text{Marital Status}=\text{Divorced}|\text{Yes}) = 1/3$
 $P(\text{Marital Status}=\text{Married}|\text{Yes}) = 0$

For Annual Income:
If class=No: sample mean=110
sample variance=2975
If class=Yes: sample mean=90
sample variance=25

(b)

Figure 5.10. The naïve Bayes classifier for the loan classification problem.

No, $P(\text{Yes}) = 0.3$ and $P(\text{No}) = 0.7$. Using the information provided in Figure 5.10(b), the class-conditional probabilities can be computed as follows:

$$\begin{aligned}
P(\mathbf{X}|\text{No}) &= P(\text{Home Owner} = \text{No}|\text{No}) \times P(\text{Status} = \text{Married}|\text{No}) \\
&\quad \times P(\text{Annual Income} = \$120\text{K}|\text{No}) \\
&= 4/7 \times 4/7 \times 0.0072 = 0.0024.
\end{aligned}$$

$$\begin{aligned}
P(\mathbf{X}|\text{Yes}) &= P(\text{Home Owner} = \text{No}|\text{Yes}) \times P(\text{Status} = \text{Married}|\text{Yes}) \\
&\quad \times P(\text{Annual Income} = \$120\text{K}|\text{Yes}) \\
&= 1 \times 0 \times 1.2 \times 10^{-9} = 0.
\end{aligned}$$

Putting them together, the posterior probability for class No is $P(\text{No}|\mathbf{X}) = \alpha \times 7/10 \times 0.0024 = 0.0016\alpha$, where $\alpha = 1/P(\mathbf{X})$ is a constant term. Using a similar approach, we can show that the posterior probability for class Yes is zero because its class-conditional probability is zero. Since $P(\text{No}|\mathbf{X}) > P(\text{Yes}|\mathbf{X})$, the record is classified as No.

M-estimate of Conditional Probability

The preceding example illustrates a potential problem with estimating posterior probabilities from training data. If the class-conditional probability for one of the attributes is zero, then the overall posterior probability for the class vanishes. This approach of estimating class-conditional probabilities using simple fractions may seem too brittle, especially when there are few training examples available and the number of attributes is large.

In a more extreme case, if the training examples do not cover many of the attribute values, we may not be able to classify some of the test records. For example, if $P(\text{Marital Status} = \text{Divorced}|\text{No})$ is zero instead of $1/7$, then a record with attribute set $\mathbf{X} = (\text{Home Owner} = \text{Yes}, \text{Marital Status} = \text{Divorced}, \text{Income} = \$120\text{K})$ has the following class-conditional probabilities:

$$P(\mathbf{X}|\text{No}) = 3/7 \times 0 \times 0.0072 = 0.$$

$$P(\mathbf{X}|\text{Yes}) = 0 \times 1/3 \times 1.2 \times 10^{-9} = 0.$$

The naïve Bayes classifier will not be able to classify the record. This problem can be addressed by using the m-estimate approach for estimating the conditional probabilities:

$$P(x_i|y_j) = \frac{n_c + mp}{n + m}, \quad (5.18)$$

where n is the total number of instances from class y_j , n_c is the number of training examples from class y_j that take on the value x_i , m is a parameter known as the equivalent sample size, and p is a user-specified parameter. If there is no training set available (i.e., $n = 0$), then $P(x_i|y_j) = p$. Therefore p can be regarded as the prior probability of observing the attribute value x_i among records with class y_j . The equivalent sample size determines the tradeoff between the prior probability p and the observed probability n_c/n .

In the example given in the previous section, the conditional probability $P(\text{Status} = \text{Married}|\text{Yes}) = 0$ because none of the training records for the class has the particular attribute value. Using the m-estimate approach with $m = 3$ and $p = 1/3$, the conditional probability is no longer zero:

$$P(\text{Marital Status} = \text{Married}|\text{Yes}) = (0 + 3 \times 1/3)/(3 + 3) = 1/6.$$

If we assume $p = 1/3$ for all attributes of class **Yes** and $p = 2/3$ for all attributes of class **No**, then

$$\begin{aligned} P(\mathbf{X}|\text{No}) &= P(\text{Home Owner} = \text{No}|\text{No}) \times P(\text{Status} = \text{Married}|\text{No}) \\ &\quad \times P(\text{Annual Income} = \$120\text{K}|\text{No}) \\ &= 6/10 \times 6/10 \times 0.0072 = 0.0026. \end{aligned}$$

$$\begin{aligned} P(\mathbf{X}|\text{Yes}) &= P(\text{Home Owner} = \text{No}|\text{Yes}) \times P(\text{Status} = \text{Married}|\text{Yes}) \\ &\quad \times P(\text{Annual Income} = \$120\text{K}|\text{Yes}) \\ &= 4/6 \times 1/6 \times 1.2 \times 10^{-9} = 1.3 \times 10^{-10}. \end{aligned}$$

The posterior probability for class **No** is $P(\text{No}|\mathbf{X}) = \alpha \times 7/10 \times 0.0026 = 0.0018\alpha$, while the posterior probability for class **Yes** is $P(\text{Yes}|\mathbf{X}) = \alpha \times 3/10 \times 1.3 \times 10^{-10} = 4.0 \times 10^{-11}\alpha$. Although the classification decision has not changed, the m-estimate approach generally provides a more robust way for estimating probabilities when the number of training examples is small.

Characteristics of Naïve Bayes Classifiers

Naïve Bayes classifiers generally have the following characteristics:

- They are robust to isolated noise points because such points are averaged out when estimating conditional probabilities from data. Naïve Bayes classifiers can also handle missing values by ignoring the example during model building and classification.
- They are robust to irrelevant attributes. If X_i is an irrelevant attribute, then $P(X_i|Y)$ becomes almost uniformly distributed. The class-conditional probability for X_i has no impact on the overall computation of the posterior probability.
- Correlated attributes can degrade the performance of naïve Bayes classifiers because the conditional independence assumption no longer holds for such attributes. For example, consider the following probabilities:

$$\begin{aligned} P(A = 0|Y = 0) &= 0.4, & P(A = 1|Y = 0) &= 0.6, \\ P(A = 0|Y = 1) &= 0.6, & P(A = 1|Y = 1) &= 0.4, \end{aligned}$$

where A is a binary attribute and Y is a binary class variable. Suppose there is another binary attribute B that is perfectly correlated with A

when $Y = 0$, but is independent of A when $Y = 1$. For simplicity, assume that the class-conditional probabilities for B are the same as for A . Given a record with attributes $A = 0, B = 0$, we can compute its posterior probabilities as follows:

$$P(Y = 0|A = 0, B = 0) = \frac{P(A = 0|Y = 0)P(B = 0|Y = 0)P(Y = 0)}{P(A = 0, B = 0)}$$

$$= \frac{0.16 \times P(Y = 0)}{P(A = 0, B = 0)}.$$

$$P(Y = 1|A = 0, B = 0) = \frac{P(A = 0|Y = 1)P(B = 0|Y = 1)P(Y = 1)}{P(A = 0, B = 0)}$$

$$= \frac{0.36 \times P(Y = 1)}{P(A = 0, B = 0)}.$$

If $P(Y = 0) = P(Y = 1)$, then the naive Bayes classifier would assign the record to class 1. However, the truth is,

$$P(A = 0, B = 0|Y = 0) = P(A = 0|Y = 0) = 0.4,$$

because A and B are perfectly correlated when $Y = 0$. As a result, the posterior probability for $Y = 0$ is

$$P(Y = 0|A = 0, B = 0) = \frac{P(A = 0, B = 0|Y = 0)P(Y = 0)}{P(A = 0, B = 0)}$$

$$= \frac{0.4 \times P(Y = 0)}{P(A = 0, B = 0)},$$

which is larger than that for $Y = 1$. The record should have been classified as class 0.

5.3.4 Bayes Error Rate

Suppose we know the true probability distribution that governs $P(\mathbf{X}|Y)$. The Bayesian classification method allows us to determine the ideal decision boundary for the classification task, as illustrated in the following example.

Example 5.3. Consider the task of identifying alligators and crocodiles based on their respective lengths. The average length of an adult crocodile is about 15 feet, while the average length of an adult alligator is about 12 feet. Assuming

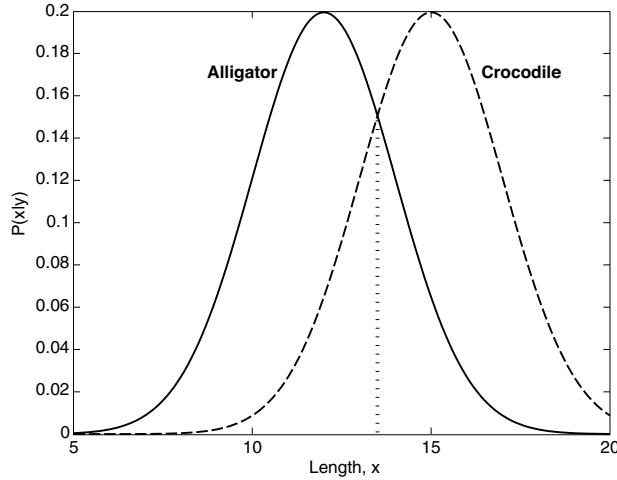


Figure 5.11. Comparing the likelihood functions of a crocodile and an alligator.

that their length x follows a Gaussian distribution with a standard deviation equal to 2 feet, we can express their class-conditional probabilities as follows:

$$P(X|\text{Crocodile}) = \frac{1}{\sqrt{2\pi} \cdot 2} \exp \left[-\frac{1}{2} \left(\frac{X - 15}{2} \right)^2 \right] \quad (5.19)$$

$$P(X|\text{Alligator}) = \frac{1}{\sqrt{2\pi} \cdot 2} \exp \left[-\frac{1}{2} \left(\frac{X - 12}{2} \right)^2 \right] \quad (5.20)$$

Figure 5.11 shows a comparison between the class-conditional probabilities for a crocodile and an alligator. Assuming that their prior probabilities are the same, the ideal decision boundary is located at some length \hat{x} such that

$$P(X = \hat{x}|\text{Crocodile}) = P(X = \hat{x}|\text{Alligator}).$$

Using Equations 5.19 and 5.20, we obtain

$$\left(\frac{\hat{x} - 15}{2} \right)^2 = \left(\frac{\hat{x} - 12}{2} \right)^2,$$

which can be solved to yield $\hat{x} = 13.5$. The decision boundary for this example is located halfway between the two means. ■

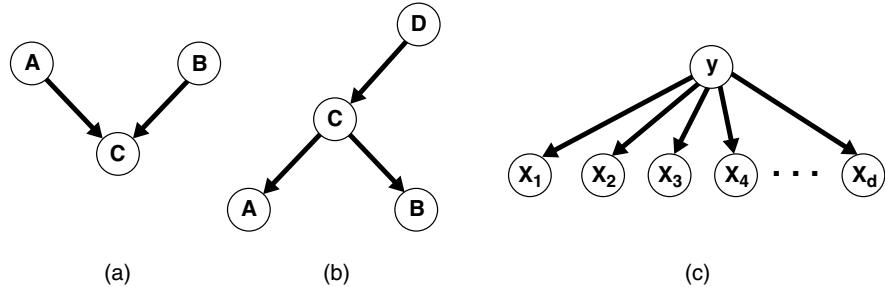


Figure 5.12. Representing probabilistic relationships using directed acyclic graphs.

When the prior probabilities are different, the decision boundary shifts toward the class with lower prior probability (see Exercise 10 on page 319). Furthermore, the minimum error rate attainable by any classifier on the given data can also be computed. The ideal decision boundary in the preceding example classifies all creatures whose lengths are less than \hat{x} as alligators and those whose lengths are greater than \hat{x} as crocodiles. The error rate of the classifier is given by the sum of the area under the posterior probability curve for crocodiles (from length 0 to \hat{x}) and the area under the posterior probability curve for alligators (from \hat{x} to ∞):

$$\text{Error} = \int_0^{\hat{x}} P(\text{Crocodile}|X)dX + \int_{\hat{x}}^{\infty} P(\text{Alligator}|X)dX.$$

The total error rate is known as the **Bayes error rate**.

5.3.5 Bayesian Belief Networks

The conditional independence assumption made by naïve Bayes classifiers may seem too rigid, especially for classification problems in which the attributes are somewhat correlated. This section presents a more flexible approach for modeling the class-conditional probabilities $P(\mathbf{X}|Y)$. Instead of requiring all the attributes to be conditionally independent given the class, this approach allows us to specify which pair of attributes are conditionally independent. We begin with a discussion on how to represent and build such a probabilistic model, followed by an example of how to make inferences from the model.

Model Representation

A Bayesian belief network (BBN), or simply, Bayesian network, provides a graphical representation of the probabilistic relationships among a set of random variables. There are two key elements of a Bayesian network:

1. A directed acyclic graph (dag) encoding the dependence relationships among a set of variables.
2. A probability table associating each node to its immediate parent nodes.

Consider three random variables, A , B , and C , in which A and B are independent variables and each has a direct influence on a third variable, C . The relationships among the variables can be summarized into the directed acyclic graph shown in Figure 5.12(a). Each node in the graph represents a variable, and each arc asserts the dependence relationship between the pair of variables. If there is a directed arc from X to Y , then X is the **parent** of Y and Y is the **child** of X . Furthermore, if there is a directed path in the network from X to Z , then X is an **ancestor** of Z , while Z is a **descendant** of X . For example, in the diagram shown in Figure 5.12(b), A is a descendant of D and D is an ancestor of B . Both B and D are also non-descendants of A . An important property of the Bayesian network can be stated as follows:

Property 1 (Conditional Independence). *A node in a Bayesian network is conditionally independent of its non-descendants, if its parents are known.*

In the diagram shown in Figure 5.12(b), A is conditionally independent of both B and D given C because the nodes for B and D are non-descendants of node A . The conditional independence assumption made by a naïve Bayes classifier can also be represented using a Bayesian network, as shown in Figure 5.12(c), where y is the target class and $\{X_1, X_2, \dots, X_d\}$ is the attribute set.

Besides the conditional independence conditions imposed by the network topology, each node is also associated with a probability table.

1. If a node X does not have any parents, then the table contains only the prior probability $P(X)$.
2. If a node X has only one parent, Y , then the table contains the conditional probability $P(X|Y)$.
3. If a node X has multiple parents, $\{Y_1, Y_2, \dots, Y_k\}$, then the table contains the conditional probability $P(X|Y_1, Y_2, \dots, Y_k)$.

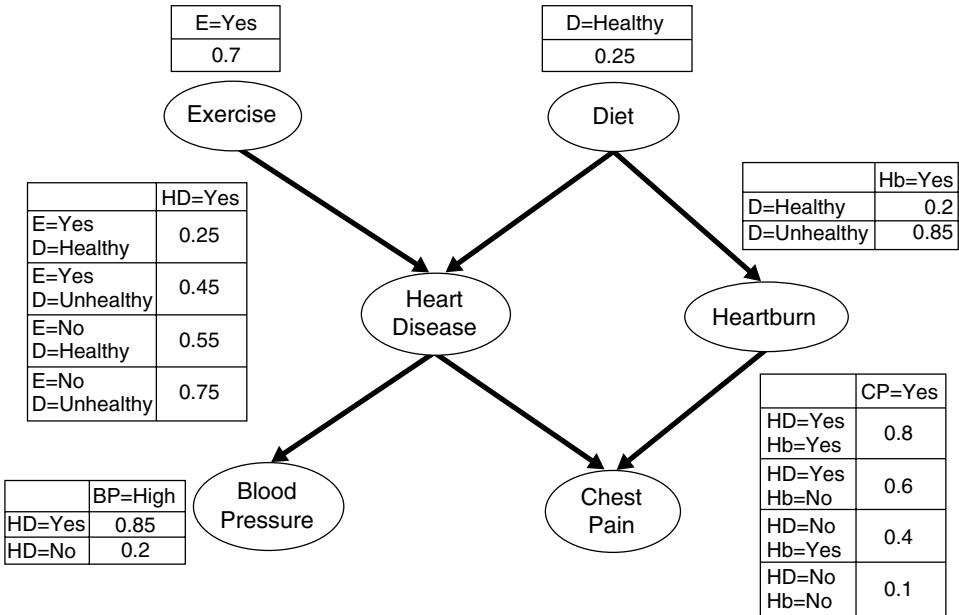


Figure 5.13. A Bayesian belief network for detecting heart disease and heartburn in patients.

Figure 5.13 shows an example of a Bayesian network for modeling patients with heart disease or heartburn problems. Each variable in the diagram is assumed to be binary-valued. The parent nodes for heart disease (HD) correspond to risk factors that may affect the disease, such as exercise (E) and diet (D). The child nodes for heart disease correspond to symptoms of the disease, such as chest pain (CP) and high blood pressure (BP). For example, the diagram shows that heartburn (Hb) may result from an unhealthy diet and may lead to chest pain.

The nodes associated with the risk factors contain only the prior probabilities, whereas the nodes for heart disease, heartburn, and their corresponding symptoms contain the conditional probabilities. To save space, some of the probabilities have been omitted from the diagram. The omitted probabilities can be recovered by noting that $P(X = \bar{x}) = 1 - P(X = x)$ and $P(X = \bar{x}|Y) = 1 - P(X = x|Y)$, where \bar{x} denotes the opposite outcome of x . For example, the conditional probability

$$\begin{aligned}
 & P(\text{Heart Disease} = \text{No} | \text{Exercise} = \text{No}, \text{Diet} = \text{Healthy}) \\
 &= 1 - P(\text{Heart Disease} = \text{Yes} | \text{Exercise} = \text{No}, \text{Diet} = \text{Healthy}) \\
 &= 1 - 0.55 = 0.45.
 \end{aligned}$$

Model Building

Model building in Bayesian networks involves two steps: (1) creating the structure of the network, and (2) estimating the probability values in the tables associated with each node. The network topology can be obtained by encoding the subjective knowledge of domain experts. Algorithm 5.3 presents a systematic procedure for inducing the topology of a Bayesian network.

Algorithm 5.3 Algorithm for generating the topology of a Bayesian network.

- 1: Let $T = (X_1, X_2, \dots, X_d)$ denote a total order of the variables.
 - 2: **for** $j = 1$ to d **do**
 - 3: Let $X_{T(j)}$ denote the j^{th} highest order variable in T .
 - 4: Let $\pi(X_{T(j)}) = \{X_{T(1)}, X_{T(2)}, \dots, X_{T(j-1)}\}$ denote the set of variables preceding $X_{T(j)}$.
 - 5: Remove the variables from $\pi(X_{T(j)})$ that do not affect X_j (using prior knowledge).
 - 6: Create an arc between $X_{T(j)}$ and the remaining variables in $\pi(X_{T(j)})$.
 - 7: **end for**
-

Example 5.4. Consider the variables shown in Figure 5.13. After performing Step 1, let us assume that the variables are ordered in the following way: (E, D, HD, Hb, CP, BP) . From Steps 2 to 7, starting with variable D , we obtain the following conditional probabilities:

- $P(D|E)$ is simplified to $P(D)$.
- $P(HD|E, D)$ cannot be simplified.
- $P(Hb|HD, E, D)$ is simplified to $P(Hb|D)$.
- $P(CP|Hb, HD, E, D)$ is simplified to $P(CP|Hb, HD)$.
- $P(BP|CP, Hb, HD, E, D)$ is simplified to $P(BP|HD)$.

Based on these conditional probabilities, we can create arcs between the nodes (E, HD) , (D, HD) , (D, Hb) , (HD, CP) , (Hb, CP) , and (HD, BP) . These arcs result in the network structure shown in Figure 5.13. ■

Algorithm 5.3 guarantees a topology that does not contain any cycles. The proof for this is quite straightforward. If a cycle exists, then there must be at least one arc connecting the lower-ordered nodes to the higher-ordered nodes, and at least another arc connecting the higher-ordered nodes to the lower-ordered nodes. Since Algorithm 5.3 prevents any arc from connecting the

lower-ordered nodes to the higher-ordered nodes, there cannot be any cycles in the topology.

Nevertheless, the network topology may change if we apply a different ordering scheme to the variables. Some topology may be inferior because it produces many arcs connecting between different pairs of nodes. In principle, we may have to examine all $d!$ possible orderings to determine the most appropriate topology, a task that can be computationally expensive. An alternative approach is to divide the variables into causal and effect variables, and then draw the arcs from each causal variable to its corresponding effect variables. This approach eases the task of building the Bayesian network structure.

Once the right topology has been found, the probability table associated with each node is determined. Estimating such probabilities is fairly straightforward and is similar to the approach used by naïve Bayes classifiers.

Example of Inferencing Using BBN

Suppose we are interested in using the BBN shown in Figure 5.13 to diagnose whether a person has heart disease. The following cases illustrate how the diagnosis can be made under different scenarios.

Case 1: No Prior Information

Without any prior information, we can determine whether the person is likely to have heart disease by computing the prior probabilities $P(\text{HD} = \text{Yes})$ and $P(\text{HD} = \text{No})$. To simplify the notation, let $\alpha \in \{\text{Yes}, \text{No}\}$ denote the binary values of **Exercise** and $\beta \in \{\text{Healthy}, \text{Unhealthy}\}$ denote the binary values of **Diet**.

$$\begin{aligned}
P(\text{HD} = \text{Yes}) &= \sum_{\alpha} \sum_{\beta} P(\text{HD} = \text{Yes} | E = \alpha, D = \beta) P(E = \alpha, D = \beta) \\
&= \sum_{\alpha} \sum_{\beta} P(\text{HD} = \text{Yes} | E = \alpha, D = \beta) P(E = \alpha) P(D = \beta) \\
&= 0.25 \times 0.7 \times 0.25 + 0.45 \times 0.7 \times 0.75 + 0.55 \times 0.3 \times 0.25 \\
&\quad + 0.75 \times 0.3 \times 0.75 \\
&= 0.49.
\end{aligned}$$

Since $P(\text{HD} = \text{no}) = 1 - P(\text{HD} = \text{yes}) = 0.51$, the person has a slightly higher chance of not getting the disease.

Case 2: High Blood Pressure

If the person has high blood pressure, we can make a diagnosis about heart disease by comparing the posterior probabilities, $P(\text{HD} = \text{Yes}|\text{BP} = \text{High})$ against $P(\text{HD} = \text{No}|\text{BP} = \text{High})$. To do this, we must compute $P(\text{BP} = \text{High})$:

$$\begin{aligned} P(\text{BP} = \text{High}) &= \sum_{\gamma} P(\text{BP} = \text{High}|\text{HD} = \gamma)P(\text{HD} = \gamma) \\ &= 0.85 \times 0.49 + 0.2 \times 0.51 = 0.5185. \end{aligned}$$

where $\gamma \in \{\text{Yes}, \text{No}\}$. Therefore, the posterior probability the person has heart disease is

$$\begin{aligned} P(\text{HD} = \text{Yes}|\text{BP} = \text{High}) &= \frac{P(\text{BP} = \text{High}|\text{HD} = \text{Yes})P(\text{HD} = \text{Yes})}{P(\text{BP} = \text{High})} \\ &= \frac{0.85 \times 0.49}{0.5185} = 0.8033. \end{aligned}$$

Similarly, $P(\text{HD} = \text{No}|\text{BP} = \text{High}) = 1 - 0.8033 = 0.1967$. Therefore, when a person has high blood pressure, it increases the risk of heart disease.

Case 3: High Blood Pressure, Healthy Diet, and Regular Exercise

Suppose we are told that the person exercises regularly and eats a healthy diet. How does the new information affect our diagnosis? With the new information, the posterior probability that the person has heart disease is

$$\begin{aligned} &P(\text{HD} = \text{Yes}|\text{BP} = \text{High}, D = \text{Healthy}, E = \text{Yes}) \\ &= \left[\frac{P(\text{BP} = \text{High}|\text{HD} = \text{Yes}, D = \text{Healthy}, E = \text{Yes})}{P(\text{BP} = \text{High}|D = \text{Healthy}, E = \text{Yes})} \right] \\ &\quad \times P(\text{HD} = \text{Yes}|D = \text{Healthy}, E = \text{Yes}) \\ &= \frac{P(\text{BP} = \text{High}|\text{HD} = \text{Yes})P(\text{HD} = \text{Yes}|D = \text{Healthy}, E = \text{Yes})}{\sum_{\gamma} P(\text{BP} = \text{High}|\text{HD} = \gamma)P(\text{HD} = \gamma|D = \text{Healthy}, E = \text{Yes})} \\ &= \frac{0.85 \times 0.25}{0.85 \times 0.25 + 0.2 \times 0.75} \\ &= 0.5862, \end{aligned}$$

while the probability that the person does not have heart disease is

$$P(\text{HD} = \text{No}|\text{BP} = \text{High}, D = \text{Healthy}, E = \text{Yes}) = 1 - 0.5862 = 0.4138.$$

The model therefore suggests that eating healthily and exercising regularly may reduce a person's risk of getting heart disease.

Characteristics of BBN

Following are some of the general characteristics of the BBN method:

1. BBN provides an approach for capturing the prior knowledge of a particular domain using a graphical model. The network can also be used to encode causal dependencies among variables.
2. Constructing the network can be time consuming and requires a large amount of effort. However, once the structure of the network has been determined, adding a new variable is quite straightforward.
3. Bayesian networks are well suited to dealing with incomplete data. Instances with missing attributes can be handled by summing or integrating the probabilities over all possible values of the attribute.
4. Because the data is combined probabilistically with prior knowledge, the method is quite robust to model overfitting.

5.4 Artificial Neural Network (ANN)

The study of artificial neural networks (ANN) was inspired by attempts to simulate biological neural systems. The human brain consists primarily of nerve cells called **neurons**, linked together with other neurons via strands of fiber called **axons**. Axons are used to transmit nerve impulses from one neuron to another whenever the neurons are stimulated. A neuron is connected to the axons of other neurons via **dendrites**, which are extensions from the cell body of the neuron. The contact point between a dendrite and an axon is called a **synapse**. Neurologists have discovered that the human brain learns by changing the strength of the synaptic connection between neurons upon repeated stimulation by the same impulse.

Analogous to human brain structure, an ANN is composed of an interconnected assembly of nodes and directed links. In this section, we will examine a family of ANN models, starting with the simplest model called **perceptron**, and show how the models can be trained to solve classification problems.

5.4.1 Perceptron

Consider the diagram shown in Figure 5.14. The table on the left shows a data set containing three boolean variables (x_1, x_2, x_3) and an output variable, y , that takes on the value -1 if at least two of the three inputs are zero, and $+1$ if at least two of the inputs are greater than zero.

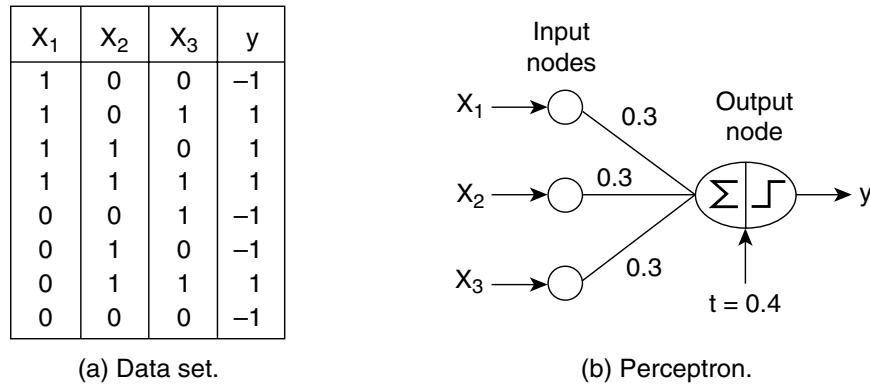


Figure 5.14. Modeling a boolean function using a perceptron.

Figure 5.14(b) illustrates a simple neural network architecture known as a perceptron. The perceptron consists of two types of nodes: input nodes, which are used to represent the input attributes, and an output node, which is used to represent the model output. The nodes in a neural network architecture are commonly known as neurons or units. In a perceptron, each input node is connected via a weighted link to the output node. The weighted link is used to emulate the strength of synaptic connection between neurons. As in biological neural systems, training a perceptron model amounts to adapting the weights of the links until they fit the input-output relationships of the underlying data.

A perceptron computes its output value, \hat{y} , by performing a weighted sum on its inputs, subtracting a bias factor t from the sum, and then examining the sign of the result. The model shown in Figure 5.14(b) has three input nodes, each of which has an identical weight of 0.3 to the output node and a bias factor of $t = 0.4$. The output computed by the model is

$$\hat{y} = \begin{cases} 1, & \text{if } 0.3x_1 + 0.3x_2 + 0.3x_3 - 0.4 > 0; \\ -1, & \text{if } 0.3x_1 + 0.3x_2 + 0.3x_3 - 0.4 < 0. \end{cases} \quad (5.21)$$

For example, if $x_1 = 1, x_2 = 1, x_3 = 0$, then $\hat{y} = +1$ because $0.3x_1 + 0.3x_2 + 0.3x_3 - 0.4$ is positive. On the other hand, if $x_1 = 0, x_2 = 1, x_3 = 0$, then $\hat{y} = -1$ because the weighted sum subtracted by the bias factor is negative.

Note the difference between the input and output nodes of a perceptron. An input node simply transmits the value it receives to the outgoing link without performing any transformation. The output node, on the other hand, is a mathematical device that computes the weighted sum of its inputs, subtracts the bias term, and then produces an output that depends on the sign of the resulting sum. More specifically, the output of a perceptron model can be expressed mathematically as follows:

$$\hat{y} = \text{sign}(w_d x_d + w_{d-1} x_{d-1} + \dots + w_2 x_2 + w_1 x_1 - t), \quad (5.22)$$

where w_1, w_2, \dots, w_d are the weights of the input links and x_1, x_2, \dots, x_d are the input attribute values. The sign function, which acts as an **activation function** for the output neuron, outputs a value $+1$ if its argument is positive and -1 if its argument is negative. The perceptron model can be written in a more compact form as follows:

$$\hat{y} = \text{sign}[w_d x_d + w_{d-1} x_{d-1} + \dots + w_1 x_1 + w_0 x_0] = \text{sign}(\mathbf{w} \cdot \mathbf{x}), \quad (5.23)$$

where $w_0 = -t$, $x_0 = 1$, and $\mathbf{w} \cdot \mathbf{x}$ is the dot product between the weight vector \mathbf{w} and the input attribute vector \mathbf{x} .

Learning Perceptron Model

During the training phase of a perceptron model, the weight parameters \mathbf{w} are adjusted until the outputs of the perceptron become consistent with the true outputs of training examples. A summary of the perceptron learning algorithm is given in Algorithm 5.4.

The key computation for this algorithm is the weight update formula given in Step 7 of the algorithm:

$$w_j^{(k+1)} = w_j^{(k)} + \lambda(y_i - \hat{y}_i^{(k)})x_{ij}, \quad (5.24)$$

where $w^{(k)}$ is the weight parameter associated with the i^{th} input link after the k^{th} iteration, λ is a parameter known as the **learning rate**, and x_{ij} is the value of the j^{th} attribute of the training example \mathbf{x}_i . The justification for the weight update formula is rather intuitive. Equation 5.24 shows that the new weight $w^{(k+1)}$ is a combination of the old weight $w^{(k)}$ and a term proportional

Algorithm 5.4 Perceptron learning algorithm.

```
1: Let  $D = \{(\mathbf{x}_i, y_i) \mid i = 1, 2, \dots, N\}$  be the set of training examples.  
2: Initialize the weight vector with random values,  $\mathbf{w}^{(0)}$   
3: repeat  
4:   for each training example  $(\mathbf{x}_i, y_i) \in D$  do  
5:     Compute the predicted output  $\hat{y}_i^{(k)}$   
6:     for each weight  $w_j$  do  
7:       Update the weight,  $w_j^{(k+1)} = w_j^{(k)} + \lambda(y_i - \hat{y}_i^{(k)})x_{ij}$ .  
8:     end for  
9:   end for  
10:  until stopping condition is met

---


```

to the prediction error, $(y - \hat{y})$. If the prediction is correct, then the weight remains unchanged. Otherwise, it is modified in the following ways:

- If $y = +1$ and $\hat{y} = -1$, then the prediction error is $(y - \hat{y}) = 2$. To compensate for the error, we need to increase the value of the predicted output by increasing the weights of all links with positive inputs and decreasing the weights of all links with negative inputs.
- If $y_i = -1$ and $\hat{y} = +1$, then $(y - \hat{y}) = -2$. To compensate for the error, we need to decrease the value of the predicted output by decreasing the weights of all links with positive inputs and increasing the weights of all links with negative inputs.

In the weight update formula, links that contribute the most to the error term are the ones that require the largest adjustment. However, the weights should not be changed too drastically because the error term is computed only for the current training example. Otherwise, the adjustments made in earlier iterations will be undone. The learning rate λ , a parameter whose value is between 0 and 1, can be used to control the amount of adjustments made in each iteration. If λ is close to 0, then the new weight is mostly influenced by the value of the old weight. On the other hand, if λ is close to 1, then the new weight is sensitive to the amount of adjustment performed in the current iteration. In some cases, an adaptive λ value can be used; initially, λ is moderately large during the first few iterations and then gradually decreases in subsequent iterations.

The perceptron model shown in Equation 5.23 is linear in its parameters \mathbf{w} and attributes \mathbf{x} . Because of this, the decision boundary of a perceptron, which is obtained by setting $\hat{y} = 0$, is a linear hyperplane that separates the data into two classes, -1 and $+1$. Figure 5.15 shows the decision boundary

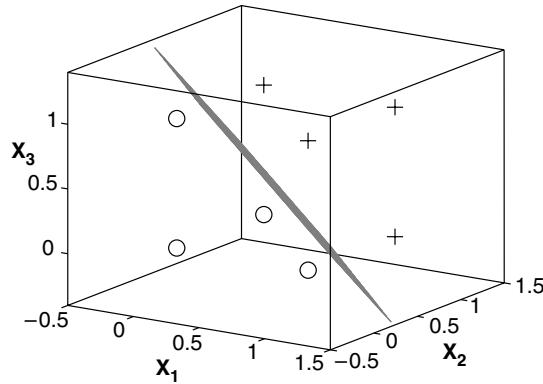


Figure 5.15. Perceptron decision boundary for the data given in Figure 5.14.

obtained by applying the perceptron learning algorithm to the data set given in Figure 5.14. The perceptron learning algorithm is guaranteed to converge to an optimal solution (as long as the learning rate is sufficiently small) for linearly separable classification problems. If the problem is not linearly separable, the algorithm fails to converge. Figure 5.16 shows an example of nonlinearly separable data given by the XOR function. Perceptron cannot find the right solution for this data because there is no linear hyperplane that can perfectly separate the training instances.

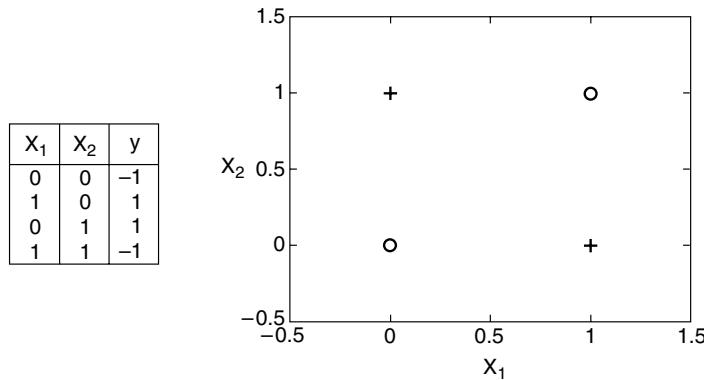


Figure 5.16. XOR classification problem. No linear hyperplane can separate the two classes.

5.4.2 Multilayer Artificial Neural Network

An artificial neural network has a more complex structure than that of a perceptron model. The additional complexities may arise in a number of ways:

1. The network may contain several intermediary layers between its input and output layers. Such intermediary layers are called **hidden layers** and the nodes embedded in these layers are called **hidden nodes**. The resulting structure is known as a multilayer neural network (see Figure 5.17). In a **feed-forward** neural network, the nodes in one layer

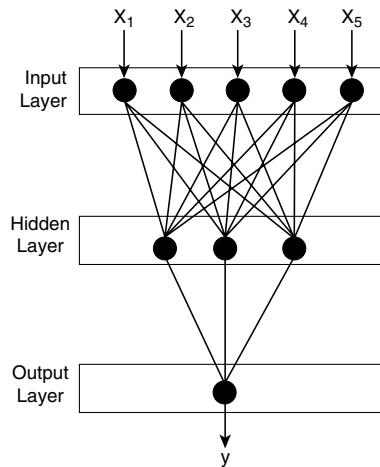


Figure 5.17. Example of a multilayer feed-forward artificial neural network (ANN).

are connected only to the nodes in the next layer. The perceptron is a single-layer, feed-forward neural network because it has only one layer of nodes—the output layer—that performs complex mathematical operations. In a **recurrent** neural network, the links may connect nodes within the same layer or nodes from one layer to the previous layers.

2. The network may use types of activation functions other than the sign function. Examples of other activation functions include linear, sigmoid (logistic), and hyperbolic tangent functions, as shown in Figure 5.18. These activation functions allow the hidden and output nodes to produce output values that are nonlinear in their input parameters.

These additional complexities allow multilayer neural networks to model more complex relationships between the input and output variables. For ex-

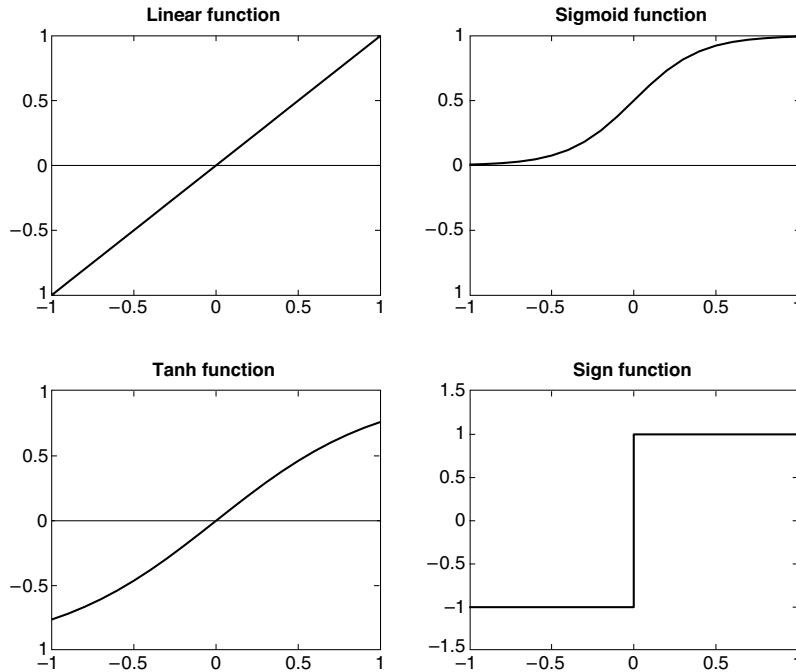


Figure 5.18. Types of activation functions in artificial neural networks.

ample, consider the XOR problem described in the previous section. The instances can be classified using two hyperplanes that partition the input space into their respective classes, as shown in Figure 5.19(a). Because a perceptron can create only one hyperplane, it cannot find the optimal solution. This problem can be addressed using a two-layer, feed-forward neural network, as shown in Figure 5.19(b). Intuitively, we can think of each hidden node as a perceptron that tries to construct one of the two hyperplanes, while the output node simply combines the results of the perceptrons to yield the decision boundary shown in Figure 5.19(a).

To learn the weights of an ANN model, we need an efficient algorithm that converges to the right solution when a sufficient amount of training data is provided. One approach is to treat each hidden node or output node in the network as an independent perceptron unit and to apply the same weight update formula as Equation 5.24. Obviously, this approach will not work because we lack *a priori* knowledge about the true outputs of the hidden nodes. This makes it difficult to determine the error term, $(y - \hat{y})$, associated

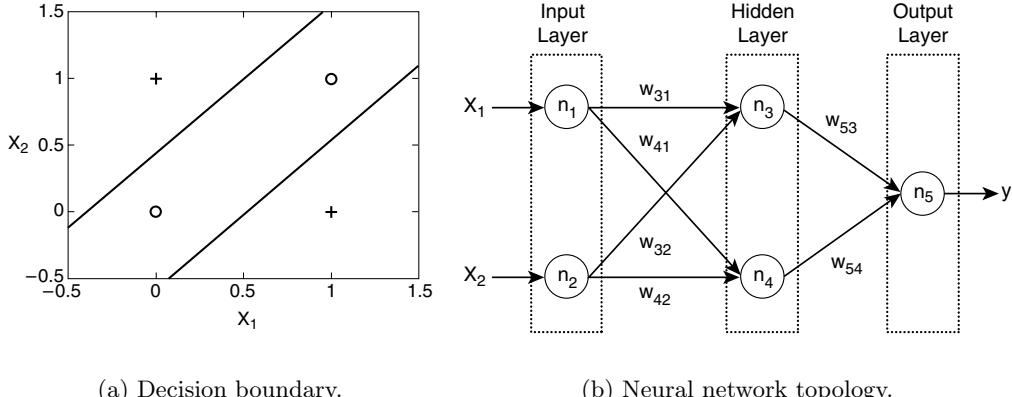


Figure 5.19. A two-layer, feed-forward neural network for the XOR problem.

with each hidden node. A methodology for learning the weights of a neural network based on the gradient descent approach is presented next.

Learning the ANN Model

The goal of the ANN learning algorithm is to determine a set of weights \mathbf{w} that minimize the total sum of squared errors:

$$E(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^N (y_i - \hat{y}_i)^2. \quad (5.25)$$

Note that the sum of squared errors depends on \mathbf{w} because the predicted class \hat{y} is a function of the weights assigned to the hidden and output nodes. Figure 5.20 shows an example of the error surface as a function of its two parameters, w_1 and w_2 . This type of error surface is typically encountered when \hat{y}_i is a linear function of its parameters, \mathbf{w} . If we replace $\hat{y} = \mathbf{w} \cdot \mathbf{x}$ into Equation 5.25, then the error function becomes quadratic in its parameters and a global minimum solution can be easily found.

In most cases, the output of an ANN is a nonlinear function of its parameters because of the choice of its activation functions (e.g., sigmoid or tanh function). As a result, it is no longer straightforward to derive a solution for \mathbf{w} that is guaranteed to be globally optimal. Greedy algorithms such as those based on the gradient descent method have been developed to efficiently solve the optimization problem. The weight update formula used by the gradient

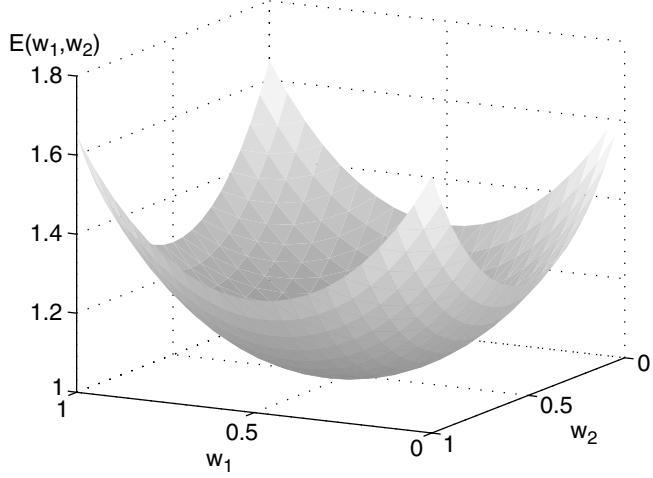


Figure 5.20. Error surface $E(w_1, w_2)$ for a two-parameter model.

descent method can be written as follows:

$$w_j \leftarrow w_j - \lambda \frac{\partial E(\mathbf{w})}{\partial w_j}, \quad (5.26)$$

where λ is the learning rate. The second term states that the weight should be increased in a direction that reduces the overall error term. However, because the error function is nonlinear, it is possible that the gradient descent method may get trapped in a local minimum.

The gradient descent method can be used to learn the weights of the output and hidden nodes of a neural network. For hidden nodes, the computation is not trivial because it is difficult to assess their error term, $\partial E / \partial w_j$, without knowing what their output values should be. A technique known as **back-propagation** has been developed to address this problem. There are two phases in each iteration of the algorithm: the forward phase and the backward phase. During the forward phase, the weights obtained from the previous iteration are used to compute the output value of each neuron in the network. The computation progresses in the forward direction; i.e., outputs of the neurons at level k are computed prior to computing the outputs at level $k + 1$. During the backward phase, the weight update formula is applied in the reverse direction. In other words, the weights at level $k + 1$ are updated before the weights at level k are updated. This back-propagation approach allows us to use the errors for neurons at layer $k + 1$ to estimate the errors for neurons at layer k .

Design Issues in ANN Learning

Before we train a neural network to learn a classification task, the following design issues must be considered.

1. The number of nodes in the input layer should be determined. Assign an input node to each numerical or binary input variable. If the input variable is categorical, we could either create one node for each categorical value or encode the k -ary variable using $\lceil \log_2 k \rceil$ input nodes.
2. The number of nodes in the output layer should be established. For a two-class problem, it is sufficient to use a single output node. For a k -class problem, there are k output nodes.
3. The network topology (e.g., the number of hidden layers and hidden nodes, and feed-forward or recurrent network architecture) must be selected. Note that the target function representation depends on the weights of the links, the number of hidden nodes and hidden layers, biases in the nodes, and type of activation function. Finding the right topology is not an easy task. One way to do this is to start from a fully connected network with a sufficiently large number of nodes and hidden layers, and then repeat the model-building procedure with a smaller number of nodes. This approach can be very time consuming. Alternatively, instead of repeating the model-building procedure, we could remove some of the nodes and repeat the model evaluation procedure to select the right model complexity.
4. The weights and biases need to be initialized. Random assignments are usually acceptable.
5. Training examples with missing values should be removed or replaced with most likely values.

5.4.3 Characteristics of ANN

Following is a summary of the general characteristics of an artificial neural network:

1. Multilayer neural networks with at least one hidden layer are **universal approximators**; i.e., they can be used to approximate any target functions. Since an ANN has a very expressive hypothesis space, it is important to choose the appropriate network topology for a given problem to avoid model overfitting.

2. ANN can handle redundant features because the weights are automatically learned during the training step. The weights for redundant features tend to be very small.
3. Neural networks are quite sensitive to the presence of noise in the training data. One approach to handling noise is to use a validation set to determine the generalization error of the model. Another approach is to decrease the weight by some factor at each iteration.
4. The gradient descent method used for learning the weights of an ANN often converges to some local minimum. One way to escape from the local minimum is to add a momentum term to the weight update formula.
5. Training an ANN is a time consuming process, especially when the number of hidden nodes is large. Nevertheless, test examples can be classified rapidly.

5.5 Support Vector Machine (SVM)

A classification technique that has received considerable attention is support vector machine (SVM). This technique has its roots in statistical learning theory and has shown promising empirical results in many practical applications, from handwritten digit recognition to text categorization. SVM also works very well with high-dimensional data and avoids the curse of dimensionality problem. Another unique aspect of this approach is that it represents the decision boundary using a subset of the training examples, known as the **support vectors**.

To illustrate the basic idea behind SVM, we first introduce the concept of a **maximal margin hyperplane** and explain the rationale of choosing such a hyperplane. We then describe how a linear SVM can be trained to explicitly look for this type of hyperplane in linearly separable data. We conclude by showing how the SVM methodology can be extended to non-linearly separable data.

5.5.1 Maximum Margin Hyperplanes

Figure 5.21 shows a plot of a data set containing examples that belong to two different classes, represented as squares and circles. The data set is also linearly separable; i.e., we can find a hyperplane such that all the squares reside on one side of the hyperplane and all the circles reside on the other

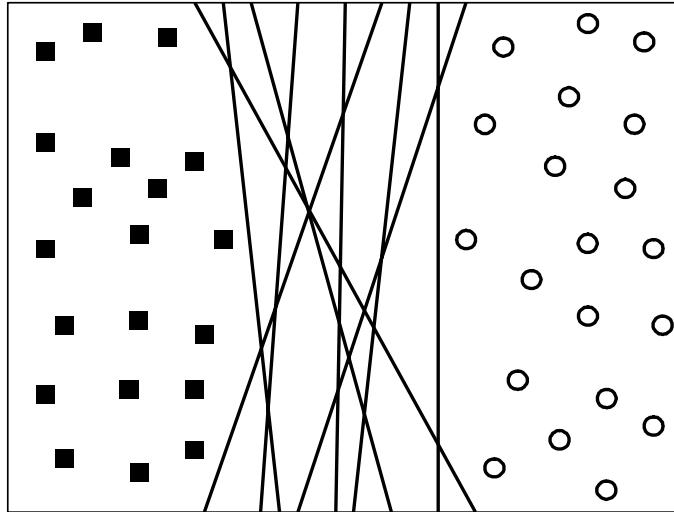


Figure 5.21. Possible decision boundaries for a linearly separable data set.

side. However, as shown in Figure 5.21, there are infinitely many such hyperplanes possible. Although their training errors are zero, there is no guarantee that the hyperplanes will perform equally well on previously unseen examples. The classifier must choose one of these hyperplanes to represent its decision boundary, based on how well they are expected to perform on test examples.

To get a clearer picture of how the different choices of hyperplanes affect the generalization errors, consider the two decision boundaries, B_1 and B_2 , shown in Figure 5.22. Both decision boundaries can separate the training examples into their respective classes without committing any misclassification errors. Each decision boundary B_i is associated with a pair of hyperplanes, denoted as b_{i1} and b_{i2} , respectively. b_{i1} is obtained by moving a parallel hyperplane away from the decision boundary until it touches the closest square(s), whereas b_{i2} is obtained by moving the hyperplane until it touches the closest circle(s). The distance between these two hyperplanes is known as the margin of the classifier. From the diagram shown in Figure 5.22, notice that the margin for B_1 is considerably larger than that for B_2 . In this example, B_1 turns out to be the maximum margin hyperplane of the training instances.

Rationale for Maximum Margin

Decision boundaries with large margins tend to have better generalization errors than those with small margins. Intuitively, if the margin is small, then

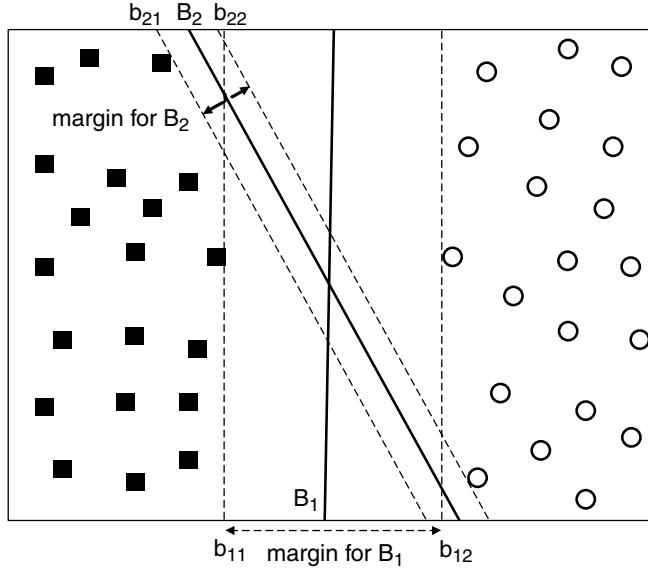


Figure 5.22. Margin of a decision boundary.

any slight perturbations to the decision boundary can have quite a significant impact on its classification. Classifiers that produce decision boundaries with small margins are therefore more susceptible to model overfitting and tend to generalize poorly on previously unseen examples.

A more formal explanation relating the margin of a linear classifier to its generalization error is given by a statistical learning principle known as **structural risk minimization** (SRM). This principle provides an upper bound to the generalization error of a classifier (R) in terms of its training error (R_e), the number of training examples (N), and the model complexity, otherwise known as its **capacity** (h). More specifically, with a probability of $1 - \eta$, the generalization error of the classifier can be at worst

$$R \leq R_e + \varphi\left(\frac{h}{N}, \frac{\log(\eta)}{N}\right), \quad (5.27)$$

where φ is a monotone increasing function of the capacity h . The preceding inequality may seem quite familiar to the readers because it resembles the equation given in Section 4.4.4 (on page 179) for the minimum description length (MDL) principle. In this regard, SRM is another way to express generalization error as a tradeoff between training error and model complexity.

The capacity of a linear model is inversely related to its margin. Models with small margins have higher capacities because they are more flexible and can fit many training sets, unlike models with large margins. However, according to the SRM principle, as the capacity increases, the generalization error bound will also increase. Therefore, it is desirable to design linear classifiers that maximize the margins of their decision boundaries in order to ensure that their worst-case generalization errors are minimized. One such classifier is the **linear SVM**, which is explained in the next section.

5.5.2 Linear SVM: Separable Case

A linear SVM is a classifier that searches for a hyperplane with the largest margin, which is why it is often known as a **maximal margin classifier**. To understand how SVM learns such a boundary, we begin with some preliminary discussion about the decision boundary and margin of a linear classifier.

Linear Decision Boundary

Consider a binary classification problem consisting of N training examples. Each example is denoted by a tuple (\mathbf{x}_i, y_i) ($i = 1, 2, \dots, N$), where $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{id})^T$ corresponds to the attribute set for the i^{th} example. By convention, let $y_i \in \{-1, 1\}$ denote its class label. The decision boundary of a linear classifier can be written in the following form:

$$\mathbf{w} \cdot \mathbf{x} + b = 0, \quad (5.28)$$

where \mathbf{w} and b are parameters of the model.

Figure 5.23 shows a two-dimensional training set consisting of squares and circles. A decision boundary that bisects the training examples into their respective classes is illustrated with a solid line. Any example located along the decision boundary must satisfy Equation 5.28. For example, if \mathbf{x}_a and \mathbf{x}_b are two points located on the decision boundary, then

$$\begin{aligned} \mathbf{w} \cdot \mathbf{x}_a + b &= 0, \\ \mathbf{w} \cdot \mathbf{x}_b + b &= 0. \end{aligned}$$

Subtracting the two equations will yield the following:

$$\mathbf{w} \cdot (\mathbf{x}_b - \mathbf{x}_a) = 0,$$

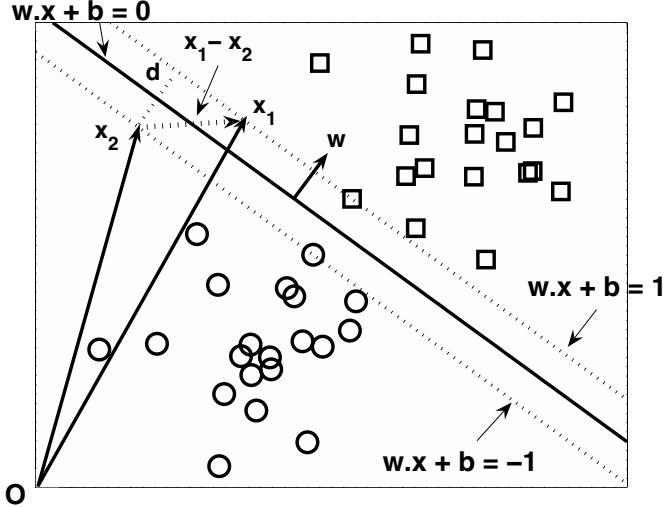


Figure 5.23. Decision boundary and margin of SVM.

where $\mathbf{x}_b - \mathbf{x}_a$ is a vector parallel to the decision boundary and is directed from \mathbf{x}_a to \mathbf{x}_b . Since the dot product is zero, the direction for \mathbf{w} must be perpendicular to the decision boundary, as shown in Figure 5.23.

For any square \mathbf{x}_s located above the decision boundary, we can show that

$$\mathbf{w} \cdot \mathbf{x}_s + b = k, \quad (5.29)$$

where $k > 0$. Similarly, for any circle \mathbf{x}_c located below the decision boundary, we can show that

$$\mathbf{w} \cdot \mathbf{x}_c + b = k', \quad (5.30)$$

where $k' < 0$. If we label all the squares as class +1 and all the circles as class -1, then we can predict the class label y for any test example \mathbf{z} in the following way:

$$y = \begin{cases} 1, & \text{if } \mathbf{w} \cdot \mathbf{z} + b > 0; \\ -1, & \text{if } \mathbf{w} \cdot \mathbf{z} + b < 0. \end{cases} \quad (5.31)$$

Margin of a Linear Classifier

Consider the square and the circle that are closest to the decision boundary. Since the square is located above the decision boundary, it must satisfy Equation 5.29 for some positive value k , whereas the circle must satisfy Equation

5.30 for some negative value k' . We can rescale the parameters \mathbf{w} and b of the decision boundary so that the two parallel hyperplanes b_{i1} and b_{i2} can be expressed as follows:

$$b_{i1} : \mathbf{w} \cdot \mathbf{x} + b = 1, \quad (5.32)$$

$$b_{i2} : \mathbf{w} \cdot \mathbf{x} + b = -1. \quad (5.33)$$

The margin of the decision boundary is given by the distance between these two hyperplanes. To compute the margin, let \mathbf{x}_1 be a data point located on b_{i1} and \mathbf{x}_2 be a data point on b_{i2} , as shown in Figure 5.23. Upon substituting these points into Equations 5.32 and 5.33, the margin d can be computed by subtracting the second equation from the first equation:

$$\begin{aligned} \mathbf{w} \cdot (\mathbf{x}_1 - \mathbf{x}_2) &= 2 \\ \|\mathbf{w}\| \times d &= 2 \\ \therefore d &= \frac{2}{\|\mathbf{w}\|}. \end{aligned} \quad (5.34)$$

Learning a Linear SVM Model

The training phase of SVM involves estimating the parameters \mathbf{w} and b of the decision boundary from the training data. The parameters must be chosen in such a way that the following two conditions are met:

$$\begin{aligned} \mathbf{w} \cdot \mathbf{x}_i + b &\geq 1 \text{ if } y_i = 1, \\ \mathbf{w} \cdot \mathbf{x}_i + b &\leq -1 \text{ if } y_i = -1. \end{aligned} \quad (5.35)$$

These conditions impose the requirements that all training instances from class $y = 1$ (i.e., the squares) must be located on or above the hyperplane $\mathbf{w} \cdot \mathbf{x} + b = 1$, while those instances from class $y = -1$ (i.e., the circles) must be located on or below the hyperplane $\mathbf{w} \cdot \mathbf{x} + b = -1$. Both inequalities can be summarized in a more compact form as follows:

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1, \quad i = 1, 2, \dots, N. \quad (5.36)$$

Although the preceding conditions are also applicable to any linear classifiers (including perceptrons), SVM imposes an additional requirement that the margin of its decision boundary must be maximal. Maximizing the margin, however, is equivalent to minimizing the following objective function:

$$f(\mathbf{w}) = \frac{\|\mathbf{w}\|^2}{2}. \quad (5.37)$$

Definition 5.1 (Linear SVM: Separable Case). The learning task in SVM can be formalized as the following constrained optimization problem:

$$\begin{aligned} & \min_{\mathbf{w}} \frac{\|\mathbf{w}\|^2}{2} \\ \text{subject to } & y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1, \quad i = 1, 2, \dots, N. \end{aligned}$$

Since the objective function is quadratic and the constraints are linear in the parameters \mathbf{w} and b , this is known as a **convex** optimization problem, which can be solved using the standard **Lagrange multiplier** method. Following is a brief sketch of the main ideas for solving the optimization problem. A more detailed discussion is given in Appendix E.

First, we must rewrite the objective function in a form that takes into account the constraints imposed on its solutions. The new objective function is known as the Lagrangian for the optimization problem:

$$L_P = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^N \lambda_i (y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1), \quad (5.38)$$

where the parameters λ_i are called the Lagrange multipliers. The first term in the Lagrangian is the same as the original objective function, while the second term captures the inequality constraints. To understand why the objective function must be modified, consider the original objective function given in Equation 5.37. It is easy to show that the function is minimized when $\mathbf{w} = \mathbf{0}$, a null vector whose components are all zeros. Such a solution, however, violates the constraints given in Definition 5.1 because there is no feasible solution for b . The solutions for \mathbf{w} and b are infeasible if they violate the inequality constraints; i.e., if $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 < 0$. The Lagrangian given in Equation 5.38 incorporates this constraint by subtracting the term from its original objective function. Assuming that $\lambda_i \geq 0$, it is clear that any infeasible solution may only increase the value of the Lagrangian.

To minimize the Lagrangian, we must take the derivative of L_P with respect to \mathbf{w} and b and set them to zero:

$$\frac{\partial L_p}{\partial \mathbf{w}} = 0 \implies \mathbf{w} = \sum_{i=1}^N \lambda_i y_i \mathbf{x}_i, \quad (5.39)$$

$$\frac{\partial L_p}{\partial b} = 0 \implies \sum_{i=1}^N \lambda_i y_i = 0. \quad (5.40)$$

Because the Lagrange multipliers are unknown, we still cannot solve for \mathbf{w} and b . If Definition 5.1 contains only equality instead of inequality constraints, then we can use the N equations from equality constraints along with Equations 5.39 and 5.40 to find the feasible solutions for \mathbf{w} , b , and λ_i . Note that the Lagrange multipliers for equality constraints are free parameters that can take any values.

One way to handle the inequality constraints is to transform them into a set of equality constraints. This is possible as long as the Lagrange multipliers are restricted to be non-negative. Such transformation leads to the following constraints on the Lagrange multipliers, which are known as the Karush-Kuhn-Tucker (KKT) conditions:

$$\lambda_i \geq 0, \quad (5.41)$$

$$\lambda_i [y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1] = 0. \quad (5.42)$$

At first glance, it may seem that there are as many Lagrange multipliers as there are training instances. It turns out that many of the Lagrange multipliers become zero after applying the constraint given in Equation 5.42. The constraint states that the Lagrange multiplier λ_i must be zero unless the training instance \mathbf{x}_i satisfies the equation $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) = 1$. Such training instance, with $\lambda_i > 0$, lies along the hyperplanes b_{i1} or b_{i2} and is known as a support vector. Training instances that do not reside along these hyperplanes have $\lambda_i = 0$. Equations 5.39 and 5.42 also suggest that the parameters \mathbf{w} and b , which define the decision boundary, depend only on the support vectors.

Solving the preceding optimization problem is still quite a daunting task because it involves a large number of parameters: \mathbf{w} , b , and λ_i . The problem can be simplified by transforming the Lagrangian into a function of the Lagrange multipliers only (this is known as the dual problem). To do this, we first substitute Equations 5.39 and 5.40 into Equation 5.38. This will lead to the following dual formulation of the optimization problem:

$$L_D = \sum_{i=1}^N \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j. \quad (5.43)$$

The key differences between the dual and primary Lagrangians are as follows:

1. The dual Lagrangian involves only the Lagrange multipliers and the training data, while the primary Lagrangian involves the Lagrange multipliers as well as parameters of the decision boundary. Nevertheless, the solutions for both optimization problems are equivalent.

2. The quadratic term in Equation 5.43 has a negative sign, which means that the original minimization problem involving the primary Lagrangian, L_P , has turned into a maximization problem involving the dual Lagrangian, L_D .

For large data sets, the dual optimization problem can be solved using numerical techniques such as quadratic programming, a topic that is beyond the scope of this book. Once the λ_i 's are found, we can use Equations 5.39 and 5.42 to obtain the feasible solutions for \mathbf{w} and b . The decision boundary can be expressed as follows:

$$\left(\sum_{i=1}^N \lambda_i y_i \mathbf{x}_i \cdot \mathbf{x} \right) + b = 0. \quad (5.44)$$

b is obtained by solving Equation 5.42 for the support vectors. Because the λ_i 's are calculated numerically and can have numerical errors, the value computed for b may not be unique. Instead it depends on the support vector used in Equation 5.42. In practice, the average value for b is chosen to be the parameter of the decision boundary.

Example 5.5. Consider the two-dimensional data set shown in Figure 5.24, which contains eight training instances. Using quadratic programming, we can solve the optimization problem stated in Equation 5.43 to obtain the Lagrange multiplier λ_i for each training instance. The Lagrange multipliers are depicted in the last column of the table. Notice that only the first two instances have non-zero Lagrange multipliers. These instances correspond to the support vectors for this data set.

Let $\mathbf{w} = (w_1, w_2)$ and b denote the parameters of the decision boundary. Using Equation 5.39, we can solve for w_1 and w_2 in the following way:

$$\begin{aligned} w_1 &= \sum_i \lambda_i y_i x_{i1} = 65.5621 \times 1 \times 0.3858 + 65.5621 \times -1 \times 0.4871 = -6.64. \\ w_2 &= \sum_i \lambda_i y_i x_{i2} = 65.5621 \times 1 \times 0.4687 + 65.5621 \times -1 \times 0.611 = -9.32. \end{aligned}$$

The bias term b can be computed using Equation 5.42 for each support vector:

$$\begin{aligned} b^{(1)} &= 1 - \mathbf{w} \cdot \mathbf{x}_1 = 1 - (-6.64)(0.3858) - (-9.32)(0.4687) = 7.9300. \\ b^{(2)} &= -1 - \mathbf{w} \cdot \mathbf{x}_2 = -1 - (-6.64)(0.4871) - (-9.32)(0.611) = 7.9289. \end{aligned}$$

Averaging these values, we obtain $b = 7.93$. The decision boundary corresponding to these parameters is shown in Figure 5.24. ■

x_1	x_2	y	Lagrange Multiplier
0.3858	0.4687	1	65.5261
0.4871	0.611	-1	65.5261
0.9218	0.4103	-1	0
0.7382	0.8936	-1	0
0.1763	0.0579	1	0
0.4057	0.3529	1	0
0.9355	0.8132	-1	0
0.2146	0.0099	1	0

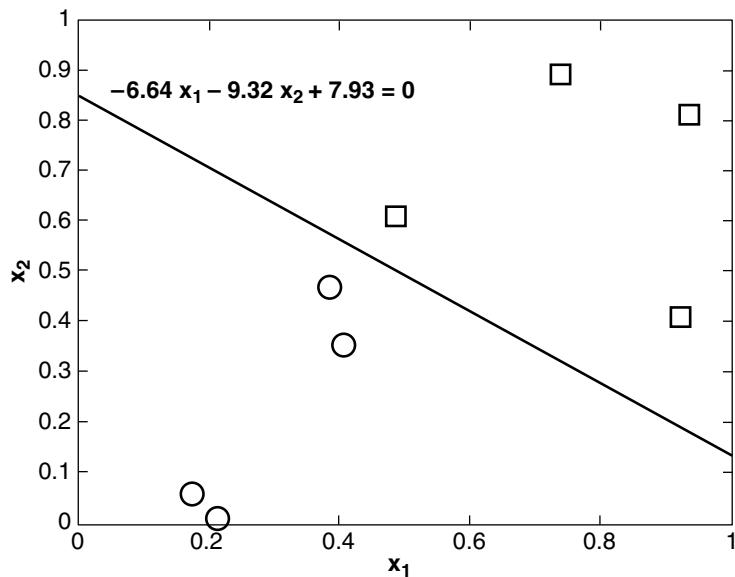


Figure 5.24. Example of a linearly separable data set.

Once the parameters of the decision boundary are found, a test instance \mathbf{z} is classified as follows:

$$f(\mathbf{z}) = \text{sign}(\mathbf{w} \cdot \mathbf{z} + b) = \text{sign}\left(\sum_{i=1}^N \lambda_i y_i \mathbf{x}_i \cdot \mathbf{z} + b\right).$$

If $f(\mathbf{z}) = 1$, then the test instance is classified as a positive class; otherwise, it is classified as a negative class.

5.5.3 Linear SVM: Nonseparable Case

Figure 5.25 shows a data set that is similar to Figure 5.22, except it has two new examples, P and Q . Although the decision boundary B_1 misclassifies the new examples, while B_2 classifies them correctly, this does not mean that B_2 is a better decision boundary than B_1 because the new examples may correspond to noise in the training data. B_1 should still be preferred over B_2 because it has a wider margin, and thus, is less susceptible to overfitting. However, the SVM formulation presented in the previous section constructs only decision boundaries that are mistake-free. This section examines how the formulation can be modified to learn a decision boundary that is tolerable to small training errors using a method known as the **soft margin** approach. More importantly, the method presented in this section allows SVM to construct a linear decision boundary even in situations where the classes are not linearly separable. To do this, the learning algorithm in SVM must consider the trade-off between the width of the margin and the number of training errors committed by the linear decision boundary.

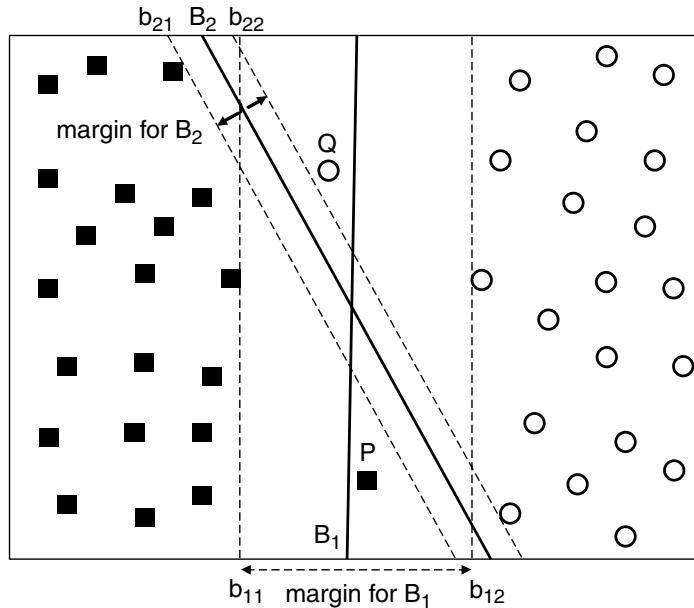


Figure 5.25. Decision boundary of SVM for the nonseparable case.

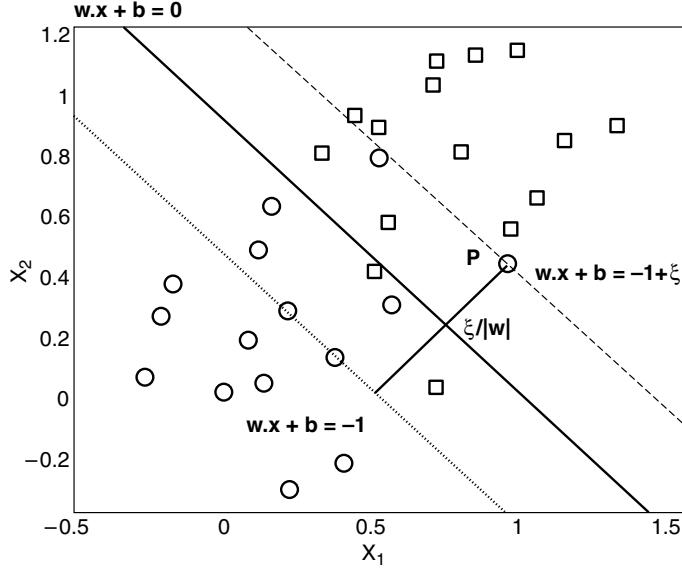


Figure 5.26. Slack variables for nonseparable data.

While the original objective function given in Equation 5.37 is still applicable, the decision boundary B_1 no longer satisfies all the constraints given in Equation 5.36. The inequality constraints must therefore be relaxed to accommodate the nonlinearly separable data. This can be done by introducing positive-valued **slack variables** (ξ) into the constraints of the optimization problem, as shown in the following equations:

$$\begin{aligned} \mathbf{w} \cdot \mathbf{x}_i + b &\geq 1 - \xi_i & \text{if } y_i = 1, \\ \mathbf{w} \cdot \mathbf{x}_i + b &\leq -1 + \xi_i & \text{if } y_i = -1, \end{aligned} \quad (5.45)$$

where $\forall i : \xi_i > 0$.

To interpret the meaning of the slack variables ξ_i , consider the diagram shown in Figure 5.26. The circle \mathbf{P} is one of the instances that violates the constraints given in Equation 5.35. Let $\mathbf{w} \cdot \mathbf{x} + b = -1 + \xi$ denote a line that is parallel to the decision boundary and passes through the point \mathbf{P} . It can be shown that the distance between this line and the hyperplane $\mathbf{w} \cdot \mathbf{x} + b = -1$ is $\xi/\|\mathbf{w}\|$. Thus, ξ provides an estimate of the error of the decision boundary on the training example \mathbf{P} .

In principle, we can apply the same objective function as before and impose the conditions given in Equation 5.45 to find the decision boundary. However,

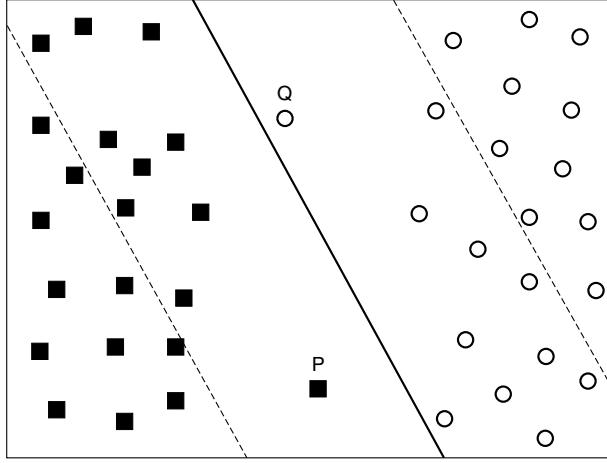


Figure 5.27. A decision boundary that has a wide margin but large training error.

since there are no constraints on the number of mistakes the decision boundary can make, the learning algorithm may find a decision boundary with a very wide margin but misclassifies many of the training examples, as shown in Figure 5.27. To avoid this problem, the objective function must be modified to penalize a decision boundary with large values of slack variables. The modified objective function is given by the following equation:

$$f(\mathbf{w}) = \frac{\|\mathbf{w}\|^2}{2} + C \left(\sum_{i=1}^N \xi_i \right)^k,$$

where C and k are user-specified parameters representing the penalty of misclassifying the training instances. For the remainder of this section, we assume $k = 1$ to simplify the problem. The parameter C can be chosen based on the model's performance on the validation set.

It follows that the Lagrangian for this constrained optimization problem can be written as follows:

$$L_P = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N \xi_i - \sum_{i=1}^N \lambda_i \{y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 + \xi_i\} - \sum_{i=1}^N \mu_i \xi_i, \quad (5.46)$$

where the first two terms are the objective function to be minimized, the third term represents the inequality constraints associated with the slack variables,

and the last term is the result of the non-negativity requirements on the values of ξ_i 's. Furthermore, the inequality constraints can be transformed into equality constraints using the following KKT conditions:

$$\xi_i \geq 0, \quad \lambda_i \geq 0, \quad \mu_i \geq 0, \quad (5.47)$$

$$\lambda_i \{y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 + \xi_i\} = 0, \quad (5.48)$$

$$\mu_i \xi_i = 0. \quad (5.49)$$

Note that the Lagrange multiplier λ_i given in Equation 5.48 is non-vanishing only if the training instance resides along the lines $\mathbf{w} \cdot \mathbf{x}_i + b = \pm 1$ or has $\xi_i > 0$. On the other hand, the Lagrange multipliers μ_i given in Equation 5.49 are zero for any training instances that are misclassified (i.e., having $\xi_i > 0$).

Setting the first-order derivative of L with respect to \mathbf{w} , b , and ξ_i to zero would result in the following equations:

$$\frac{\partial L}{\partial w_j} = w_j - \sum_{i=1}^N \lambda_i y_i x_{ij} = 0 \implies w_j = \sum_{i=1}^N \lambda_i y_i x_{ij}. \quad (5.50)$$

$$\frac{\partial L}{\partial b} = - \sum_{i=1}^N \lambda_i y_i = 0 \implies \sum_{i=1}^N \lambda_i y_i = 0. \quad (5.51)$$

$$\frac{\partial L}{\partial \xi_i} = C - \lambda_i - \mu_i = 0 \implies \lambda_i + \mu_i = C. \quad (5.52)$$

Substituting Equations 5.50, 5.51, and 5.52 into the Lagrangian will produce the following dual Lagrangian:

$$\begin{aligned} L_D &= \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j + C \sum_i \xi_i \\ &\quad - \sum_i \lambda_i \{y_i (\sum_j \lambda_j y_j \mathbf{x}_i \cdot \mathbf{x}_j + b) - 1 + \xi_i\} \\ &\quad - \sum_i (C - \lambda_i) \xi_i \\ &= \sum_{i=1}^N \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j, \end{aligned} \quad (5.53)$$

which turns out to be identical to the dual Lagrangian for linearly separable data (see Equation 5.40 on page 262). Nevertheless, the constraints imposed

on the Lagrange multipliers λ_i 's are slightly different those in the linearly separable case. In the linearly separable case, the Lagrange multipliers must be non-negative, i.e., $\lambda_i \geq 0$. On the other hand, Equation 5.52 suggests that λ_i should not exceed C (since both μ_i and λ_i are non-negative). Therefore, the Lagrange multipliers for nonlinearly separable data are restricted to $0 \leq \lambda_i \leq C$.

The dual problem can then be solved numerically using quadratic programming techniques to obtain the Lagrange multipliers λ_i . These multipliers can be replaced into Equation 5.50 and the KKT conditions to obtain the parameters of the decision boundary.

5.5.4 Nonlinear SVM

The SVM formulations described in the previous sections construct a linear decision boundary to separate the training examples into their respective classes. This section presents a methodology for applying SVM to data sets that have nonlinear decision boundaries. The trick here is to transform the data from its original coordinate space in \mathbf{x} into a new space $\Phi(\mathbf{x})$ so that a linear decision boundary can be used to separate the instances in the transformed space. After doing the transformation, we can apply the methodology presented in the previous sections to find a linear decision boundary in the transformed space.

Attribute Transformation

To illustrate how attribute transformation can lead to a linear decision boundary, Figure 5.28(a) shows an example of a two-dimensional data set consisting of squares (classified as $y = 1$) and circles (classified as $y = -1$). The data set is generated in such a way that all the circles are clustered near the center of the diagram and all the squares are distributed farther away from the center. Instances of the data set can be classified using the following equation:

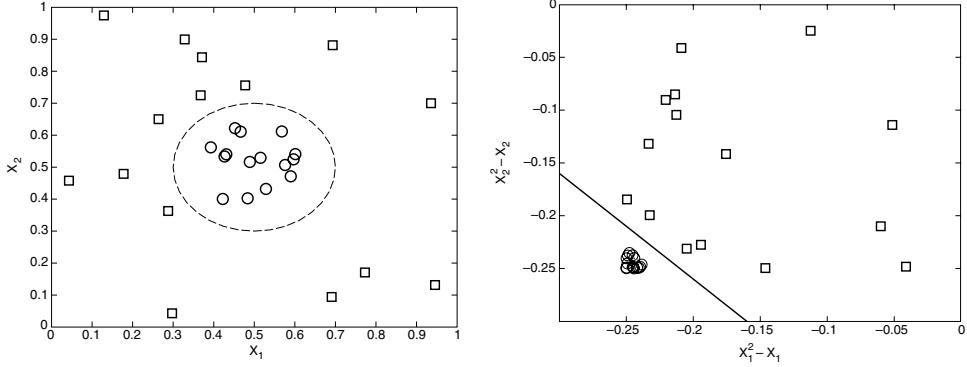
$$y(x_1, x_2) = \begin{cases} 1 & \text{if } \sqrt{(x_1 - 0.5)^2 + (x_2 - 0.5)^2} > 0.2, \\ -1 & \text{otherwise.} \end{cases} \quad (5.54)$$

The decision boundary for the data can therefore be written as follows:

$$\sqrt{(x_1 - 0.5)^2 + (x_2 - 0.5)^2} = 0.2,$$

which can be further simplified into the following quadratic equation:

$$x_1^2 - x_1 + x_2^2 - x_2 = -0.46.$$



(a) Decision boundary in the original two-dimensional space.

(b) Decision boundary in the transformed space.

Figure 5.28. Classifying data with a nonlinear decision boundary.

A nonlinear transformation Φ is needed to map the data from its original feature space into a new space where the decision boundary becomes linear. Suppose we choose the following transformation:

$$\Phi : (x_1, x_2) \longrightarrow (x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, 1). \quad (5.55)$$

In the transformed space, we can find the parameters $\mathbf{w} = (w_0, w_1, \dots, w_4)$ such that:

$$w_4x_1^2 + w_3x_2^2 + w_2\sqrt{2}x_1 + w_1\sqrt{2}x_2 + w_0 = 0.$$

For illustration purposes, let us plot the graph of $x_2^2 - x_2$ versus $x_1^2 - x_1$ for the previously given instances. Figure 5.28(b) shows that in the transformed space, all the circles are located in the lower right-hand side of the diagram. A linear decision boundary can therefore be constructed to separate the instances into their respective classes.

One potential problem with this approach is that it may suffer from the curse of dimensionality problem often associated with high-dimensional data. We will show how nonlinear SVM avoids this problem (using a method known as the kernel trick) later in this section.

Learning a Nonlinear SVM Model

Although the attribute transformation approach seems promising, it raises several implementation issues. First, it is not clear what type of mapping

function should be used to ensure that a linear decision boundary can be constructed in the transformed space. One possibility is to transform the data into an infinite dimensional space, but such a high-dimensional space may not be that easy to work with. Second, even if the appropriate mapping function is known, solving the constrained optimization problem in the high-dimensional feature space is a computationally expensive task.

To illustrate these issues and examine the ways they can be addressed, let us assume that there is a suitable function, $\Phi(\mathbf{x})$, to transform a given data set. After the transformation, we need to construct a linear decision boundary that will separate the instances into their respective classes. The linear decision boundary in the transformed space has the following form: $\mathbf{w} \cdot \Phi(\mathbf{x}) + b = 0$.

Definition 5.2 (Nonlinear SVM). The learning task for a nonlinear SVM can be formalized as the following optimization problem:

$$\begin{aligned} & \min_{\mathbf{w}} \frac{\|\mathbf{w}\|^2}{2} \\ \text{subject to } & y_i(\mathbf{w} \cdot \Phi(\mathbf{x}_i) + b) \geq 1, \quad i = 1, 2, \dots, N. \end{aligned}$$

Note the similarity between the learning task of a nonlinear SVM to that of a linear SVM (see Definition 5.1 on page 262). The main difference is that, instead of using the original attributes \mathbf{x} , the learning task is performed on the transformed attributes $\Phi(\mathbf{x})$. Following the approach taken in Sections 5.5.2 and 5.5.3 for linear SVM, we may derive the following dual Lagrangian for the constrained optimization problem:

$$L_D = \sum_{i=1}^n \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j) \quad (5.56)$$

Once the λ_i 's are found using quadratic programming techniques, the parameters \mathbf{w} and b can be derived using the following equations:

$$\mathbf{w} = \sum_i \lambda_i y_i \Phi(\mathbf{x}_i) \quad (5.57)$$

$$\lambda_i \{y_i (\sum_j \lambda_j y_j \Phi(\mathbf{x}_j) \cdot \Phi(\mathbf{x}_i) + b) - 1\} = 0, \quad (5.58)$$

which are analogous to Equations 5.39 and 5.40 for linear SVM. Finally, a test instance z can be classified using the following equation:

$$f(\mathbf{z}) = \text{sign}(\mathbf{w} \cdot \Phi(\mathbf{z}) + b) = \text{sign}\left(\sum_{i=1}^n \lambda_i y_i \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{z}) + b\right). \quad (5.59)$$

Except for Equation 5.57, note that the rest of the computations (Equations 5.58 and 5.59) involve calculating the dot product (i.e., similarity) between pairs of vectors in the transformed space, $\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$. Such computation can be quite cumbersome and may suffer from the curse of dimensionality problem. A breakthrough solution to this problem comes in the form of a method known as the **kernel trick**.

Kernel Trick

The dot product is often regarded as a measure of similarity between two input vectors. For example, the cosine similarity described in Section 2.4.5 on page 73 can be defined as the dot product between two vectors that are normalized to unit length. Analogously, the dot product $\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$ can also be regarded as a measure of similarity between two instances, \mathbf{x}_i and \mathbf{x}_j , in the transformed space.

The kernel trick is a method for computing similarity in the transformed space using the original attribute set. Consider the mapping function Φ given in Equation 5.55. The dot product between two input vectors \mathbf{u} and \mathbf{v} in the transformed space can be written as follows:

$$\begin{aligned} \Phi(\mathbf{u}) \cdot \Phi(\mathbf{v}) &= (u_1^2, u_2^2, \sqrt{2}u_1, \sqrt{2}u_2, 1) \cdot (v_1^2, v_2^2, \sqrt{2}v_1, \sqrt{2}v_2, 1) \\ &= u_1^2 v_1^2 + u_2^2 v_2^2 + 2u_1 v_1 + 2u_2 v_2 + 1 \\ &= (\mathbf{u} \cdot \mathbf{v} + 1)^2. \end{aligned} \quad (5.60)$$

This analysis shows that the dot product in the transformed space can be expressed in terms of a similarity function in the original space:

$$K(\mathbf{u}, \mathbf{v}) = \Phi(\mathbf{u}) \cdot \Phi(\mathbf{v}) = (\mathbf{u} \cdot \mathbf{v} + 1)^2. \quad (5.61)$$

The similarity function, K , which is computed in the original attribute space, is known as the **kernel function**. The kernel trick helps to address some of the concerns about how to implement nonlinear SVM. First, we do not have to know the exact form of the mapping function Φ because the kernel

functions used in nonlinear SVM must satisfy a mathematical principle known as **Mercer's theorem**. This principle ensures that the kernel functions can always be expressed as the dot product between two input vectors in some high-dimensional space. The transformed space of the SVM kernels is called a **reproducing kernel Hilbert space** (RKHS). Second, computing the dot products using kernel functions is considerably cheaper than using the transformed attribute set $\Phi(\mathbf{x})$. Third, since the computations are performed in the original space, issues associated with the curse of dimensionality problem can be avoided.

Figure 5.29 shows the nonlinear decision boundary obtained by SVM using the polynomial kernel function given in Equation 5.61. A test instance \mathbf{z} is classified according to the following equation:

$$\begin{aligned} f(\mathbf{z}) &= \text{sign}\left(\sum_{i=1}^n \lambda_i y_i \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{z}) + b\right) \\ &= \text{sign}\left(\sum_{i=1}^n \lambda_i y_i K(\mathbf{x}_i, \mathbf{z}) + b\right) \\ &= \text{sign}\left(\sum_{i=1}^n \lambda_i y_i (\mathbf{x}_i \cdot \mathbf{z} + 1)^2 + b\right), \end{aligned} \quad (5.62)$$

where b is the parameter obtained using Equation 5.58. The decision boundary obtained by nonlinear SVM is quite close to the true decision boundary shown in Figure 5.28(a).

Mercer's Theorem

The main requirement for the kernel function used in nonlinear SVM is that there must exist a corresponding transformation such that the kernel function computed for a pair of vectors is equivalent to the dot product between the vectors in the transformed space. This requirement can be formally stated in the form of Mercer's theorem.

Theorem 5.1 (Mercer's Theorem). *A kernel function K can be expressed as*

$$K(\mathbf{u}, \mathbf{v}) = \Phi(\mathbf{u}) \cdot \Phi(\mathbf{v})$$

if and only if, for any function $g(x)$ such that $\int g(\mathbf{x})^2 d\mathbf{x}$ is finite, then

$$\int K(\mathbf{x}, \mathbf{y}) g(\mathbf{x}) g(\mathbf{y}) d\mathbf{x} d\mathbf{y} \geq 0.$$

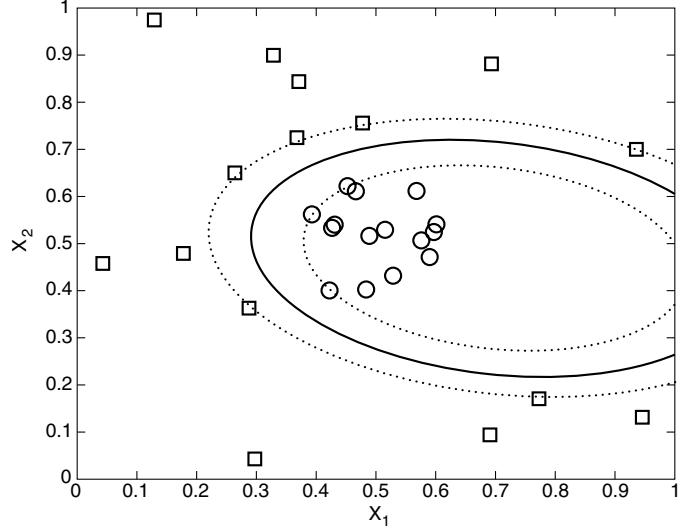


Figure 5.29. Decision boundary produced by a nonlinear SVM with polynomial kernel.

Kernel functions that satisfy Theorem 5.1 are called positive definite kernel functions. Examples of such functions are listed below:

$$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y} + 1)^p \quad (5.63)$$

$$K(\mathbf{x}, \mathbf{y}) = e^{-\|\mathbf{x}-\mathbf{y}\|^2/(2\sigma^2)} \quad (5.64)$$

$$K(\mathbf{x}, \mathbf{y}) = \tanh(k\mathbf{x} \cdot \mathbf{y} - \delta) \quad (5.65)$$

Example 5.6. Consider the polynomial kernel function given in Equation 5.63. Let $g(x)$ be a function that has a finite L_2 norm, i.e., $\int g(\mathbf{x})^2 d\mathbf{x} < \infty$.

$$\begin{aligned} & \int (\mathbf{x} \cdot \mathbf{y} + 1)^p g(\mathbf{x}) g(\mathbf{y}) d\mathbf{x} d\mathbf{y} \\ &= \int \sum_{i=0}^p \binom{p}{i} (\mathbf{x} \cdot \mathbf{y})^i g(\mathbf{x}) g(\mathbf{y}) d\mathbf{x} d\mathbf{y} \\ &= \sum_{i=0}^p \binom{p}{i} \int \sum_{\alpha_1, \alpha_2, \dots} \binom{i}{\alpha_1 \alpha_2 \dots} \left[(x_1 y_1)^{\alpha_1} (x_2 y_2)^{\alpha_2} (x_3 y_3)^{\alpha_3} \dots \right] \\ & \quad g(x_1, x_2, \dots) g(y_1, y_2, \dots) dx_1 dx_2 \dots dy_1 dy_2 \dots \end{aligned}$$

$$= \sum_{i=0}^p \sum_{\alpha_1, \alpha_2, \dots} \binom{p}{i} \binom{i}{\alpha_1 \alpha_2 \dots} \left[\int x_1^{\alpha_1} x_2^{\alpha_2} \dots g(x_1, x_2, \dots) dx_1 dx_2 \dots \right]^2.$$

Because the result of the integration is non-negative, the polynomial kernel function therefore satisfies Mercer's theorem. ■

5.5.5 Characteristics of SVM

SVM has many desirable qualities that make it one of the most widely used classification algorithms. Following is a summary of the general characteristics of SVM:

1. The SVM learning problem can be formulated as a convex optimization problem, in which efficient algorithms are available to find the global minimum of the objective function. Other classification methods, such as rule-based classifiers and artificial neural networks, employ a greedy-based strategy to search the hypothesis space. Such methods tend to find only locally optimum solutions.
2. SVM performs capacity control by maximizing the margin of the decision boundary. Nevertheless, the user must still provide other parameters such as the type of kernel function to use and the cost function C for introducing each slack variable.
3. SVM can be applied to categorical data by introducing dummy variables for each categorical attribute value present in the data. For example, if **Marital Status** has three values {**Single**, **Married**, **Divorced**}, we can introduce a binary variable for each of the attribute values.
4. The SVM formulation presented in this chapter is for binary class problems. Some of the methods available to extend SVM to multiclass problems are presented in Section 5.8.

5.6 Ensemble Methods

The classification techniques we have seen so far in this chapter, with the exception of the nearest-neighbor method, predict the class labels of unknown examples using a single classifier induced from training data. This section presents techniques for improving classification accuracy by aggregating the predictions of multiple classifiers. These techniques are known as the **ensemble** or **classifier combination** methods. An ensemble method constructs a

set of **base classifiers** from training data and performs classification by taking a vote on the predictions made by each base classifier. This section explains why ensemble methods tend to perform better than any single classifier and presents techniques for constructing the classifier ensemble.

5.6.1 Rationale for Ensemble Method

The following example illustrates how an ensemble method can improve a classifier's performance.

Example 5.7. Consider an ensemble of twenty-five binary classifiers, each of which has an error rate of $\epsilon = 0.35$. The ensemble classifier predicts the class label of a test example by taking a majority vote on the predictions made by the base classifiers. If the base classifiers are identical, then the ensemble will misclassify the same examples predicted incorrectly by the base classifiers. Thus, the error rate of the ensemble remains 0.35. On the other hand, if the base classifiers are independent—i.e., their errors are uncorrelated—then the ensemble makes a wrong prediction only if more than half of the base classifiers predict incorrectly. In this case, the error rate of the ensemble classifier is

$$e_{\text{ensemble}} = \sum_{i=13}^{25} \binom{25}{i} \epsilon^i (1 - \epsilon)^{25-i} = 0.06, \quad (5.66)$$

which is considerably lower than the error rate of the base classifiers. ■

Figure 5.30 shows the error rate of an ensemble of twenty-five binary classifiers (e_{ensemble}) for different base classifier error rates (ϵ). The diagonal line represents the case in which the base classifiers are identical, while the solid line represents the case in which the base classifiers are independent. Observe that the ensemble classifier performs worse than the base classifiers when ϵ is larger than 0.5.

The preceding example illustrates two necessary conditions for an ensemble classifier to perform better than a single classifier: (1) the base classifiers should be independent of each other, and (2) the base classifiers should do better than a classifier that performs random guessing. In practice, it is difficult to ensure total independence among the base classifiers. Nevertheless, improvements in classification accuracies have been observed in ensemble methods in which the base classifiers are slightly correlated.

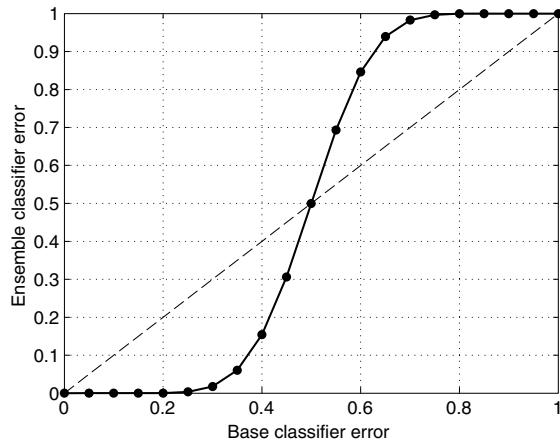


Figure 5.30. Comparison between errors of base classifiers and errors of the ensemble classifier.

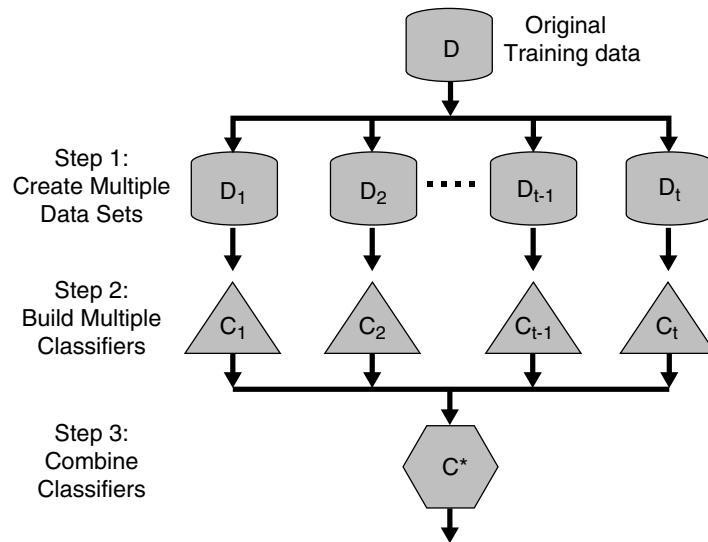


Figure 5.31. A logical view of the ensemble learning method.

5.6.2 Methods for Constructing an Ensemble Classifier

A logical view of the ensemble method is presented in Figure 5.31. The basic idea is to construct multiple classifiers from the original data and then aggregate their predictions when classifying unknown examples. The ensemble of classifiers can be constructed in many ways:

1. **By manipulating the training set.** In this approach, multiple training sets are created by resampling the original data according to some sampling distribution. The sampling distribution determines how likely it is that an example will be selected for training, and it may vary from one trial to another. A classifier is then built from each training set using a particular learning algorithm. **Bagging** and **boosting** are two examples of ensemble methods that manipulate their training sets. These methods are described in further detail in Sections 5.6.4 and 5.6.5.
2. **By manipulating the input features.** In this approach, a subset of input features is chosen to form each training set. The subset can be either chosen randomly or based on the recommendation of domain experts. Some studies have shown that this approach works very well with data sets that contain highly redundant features. **Random forest**, which is described in Section 5.6.6, is an ensemble method that manipulates its input features and uses decision trees as its base classifiers.
3. **By manipulating the class labels.** This method can be used when the number of classes is sufficiently large. The training data is transformed into a binary class problem by randomly partitioning the class labels into two disjoint subsets, A_0 and A_1 . Training examples whose class label belongs to the subset A_0 are assigned to class 0, while those that belong to the subset A_1 are assigned to class 1. The relabeled examples are then used to train a base classifier. By repeating the class-relabeling and model-building steps multiple times, an ensemble of base classifiers is obtained. When a test example is presented, each base classifier C_i is used to predict its class label. If the test example is predicted as class 0, then all the classes that belong to A_0 will receive a vote. Conversely, if it is predicted to be class 1, then all the classes that belong to A_1 will receive a vote. The votes are tallied and the class that receives the highest vote is assigned to the test example. An example of this approach is the **error-correcting output coding** method described on page 307.
4. **By manipulating the learning algorithm.** Many learning algorithms can be manipulated in such a way that applying the algorithm several times on the same training data may result in different models. For example, an artificial neural network can produce different models by changing its network topology or the initial weights of the links between neurons. Similarly, an ensemble of decision trees can be constructed by injecting randomness into the tree-growing procedure. For

example, instead of choosing the best splitting attribute at each node, we can randomly choose one of the top k attributes for splitting.

The first three approaches are generic methods that are applicable to any classifiers, whereas the fourth approach depends on the type of classifier used. The base classifiers for most of these approaches can be generated sequentially (one after another) or in parallel (all at once). Algorithm 5.5 shows the steps needed to build an ensemble classifier in a sequential manner. The first step is to create a training set from the original data D . Depending on the type of ensemble method used, the training sets are either identical to or slight modifications of D . The size of the training set is often kept the same as the original data, but the distribution of examples may not be identical; i.e., some examples may appear multiple times in the training set, while others may not appear even once. A base classifier C_i is then constructed from each training set D_i . Ensemble methods work better with **unstable classifiers**, i.e., base classifiers that are sensitive to minor perturbations in the training set. Examples of unstable classifiers include decision trees, rule-based classifiers, and artificial neural networks. As will be discussed in Section 5.6.3, the variability among training examples is one of the primary sources of errors in a classifier. By aggregating the base classifiers built from different training sets, this may help to reduce such types of errors.

Finally, a test example \mathbf{x} is classified by combining the predictions made by the base classifiers $C_i(\mathbf{x})$:

$$C^*(\mathbf{x}) = \text{Vote}(C_1(\mathbf{x}), C_2(\mathbf{x}), \dots, C_k(\mathbf{x})).$$

The class can be obtained by taking a majority vote on the individual predictions or by weighting each prediction with the accuracy of the base classifier.

Algorithm 5.5 General procedure for ensemble method.

- 1: Let D denote the original training data, k denote the number of base classifiers, and T be the test data.
 - 2: **for** $i = 1$ to k **do**
 - 3: Create training set, D_i from D .
 - 4: Build a base classifier C_i from D_i .
 - 5: **end for**
 - 6: **for** each test record $x \in T$ **do**
 - 7: $C^*(x) = \text{Vote}(C_1(x), C_2(x), \dots, C_k(x))$
 - 8: **end for**
-

5.6.3 Bias-Variance Decomposition

Bias-variance decomposition is a formal method for analyzing the prediction error of a predictive model. The following example gives an intuitive explanation for this method.

Figure 5.32 shows the trajectories of a projectile launched at a particular angle. Suppose the projectile hits the floor surface at some location x , at a distance d away from the target position t . Depending on the force applied to the projectile, the observed distance may vary from one trial to another. The observed distance can be decomposed into several components. The first component, which is known as **bias**, measures the average distance between the target position and the location where the projectile hits the floor. The amount of bias depends on the angle of the projectile launcher. The second component, which is known as **variance**, measures the deviation between x and the average position \bar{x} where the projectile hits the floor. The variance can be explained as a result of changes in the amount of force applied to the projectile. Finally, if the target is not stationary, then the observed distance is also affected by changes in the location of the target. This is considered the **noise** component associated with variability in the target position. Putting these components together, the average distance can be expressed as:

$$d_{f,\theta}(y, t) = \text{Bias}_\theta + \text{Variance}_f + \text{Noise}_t, \quad (5.67)$$

where f refers to the amount of force applied and θ is the angle of the launcher.

The task of predicting the class label of a given example can be analyzed using the same approach. For a given classifier, some predictions may turn out to be correct, while others may be completely off the mark. We can decompose the expected error of a classifier as a sum of the three terms given in Equation 5.67, where expected error is the probability that the classifier misclassifies a

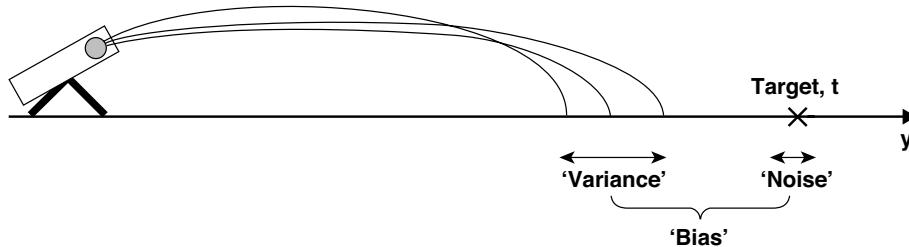


Figure 5.32. Bias-variance decomposition.

given example. The remainder of this section examines the meaning of bias, variance, and noise in the context of classification.

A classifier is usually trained to minimize its training error. However, to be useful, the classifier must be able to make an informed guess about the class labels of examples it has never seen before. This requires the classifier to generalize its decision boundary to regions where there are no training examples available—a decision that depends on the design choice of the classifier. For example, a key design issue in decision tree induction is the amount of pruning needed to obtain a tree with low expected error. Figure 5.33 shows two decision trees, T_1 and T_2 , that are generated from the same training data, but have different complexities. T_2 is obtained by pruning T_1 until a tree with maximum depth of two is obtained. T_1 , on the other hand, performs very little pruning on its decision tree. These design choices will introduce a bias into the classifier that is analogous to the bias of the projectile launcher described in the previous example. In general, the stronger the assumptions made by a classifier about the nature of its decision boundary, the larger the classifier’s bias will be. T_2 therefore has a larger bias because it makes stronger assumptions about its decision boundary (which is reflected by the size of the tree) compared to T_1 . Other design choices that may introduce a bias into a classifier include the network topology of an artificial neural network and the number of neighbors considered by a nearest-neighbor classifier.

The expected error of a classifier is also affected by variability in the training data because different compositions of the training set may lead to different decision boundaries. This is analogous to the variance in x when different amounts of force are applied to the projectile. The last component of the expected error is associated with the intrinsic noise in the target class. The target class for some domains can be non-deterministic; i.e., instances with the same attribute values can have different class labels. Such errors are unavoidable even when the true decision boundary is known.

The amount of bias and variance contributing to the expected error depend on the type of classifier used. Figure 5.34 compares the decision boundaries produced by a decision tree and a 1-nearest neighbor classifier. For each classifier, we plot the decision boundary obtained by “averaging” the models induced from 100 training sets, each containing 100 examples. The true decision boundary from which the data is generated is also plotted using a dashed line. The difference between the true decision boundary and the “averaged” decision boundary reflects the bias of the classifier. After averaging the models, observe that the difference between the true decision boundary and the decision boundary produced by the 1-nearest neighbor classifier is smaller than

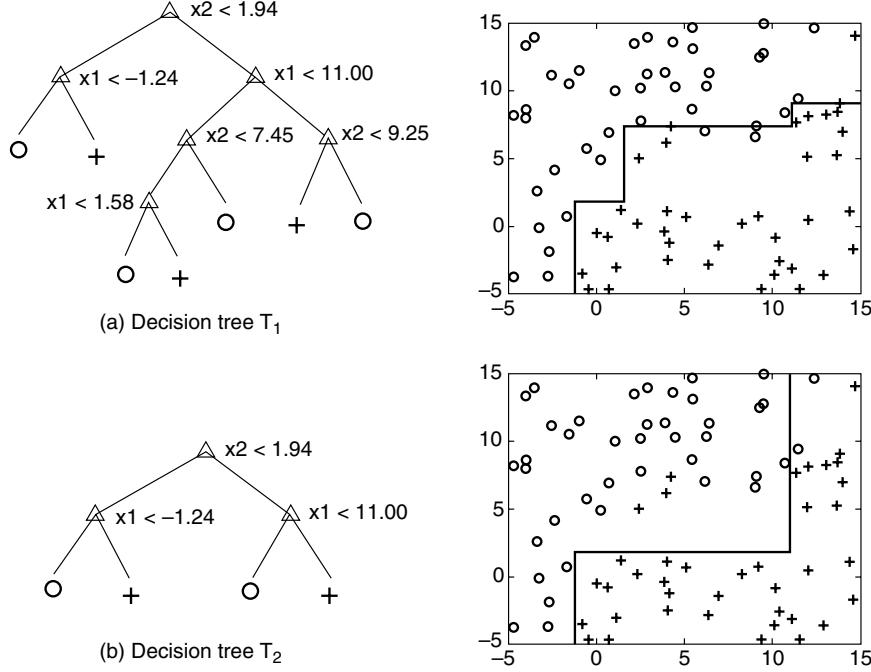


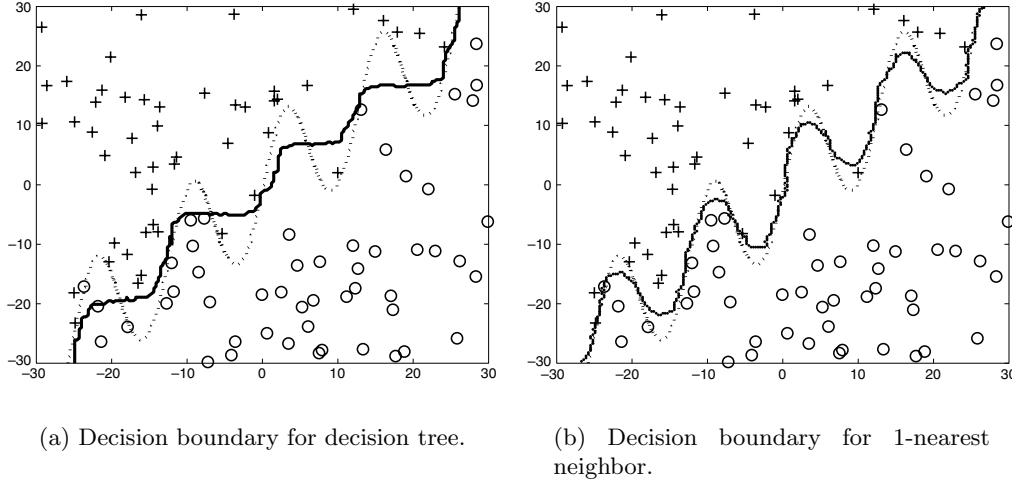
Figure 5.33. Two decision trees with different complexities induced from the same training data.

the observed difference for a decision tree classifier. This result suggests that the bias of a 1-nearest neighbor classifier is lower than the bias of a decision tree classifier.

On the other hand, the 1-nearest neighbor classifier is more sensitive to the composition of its training examples. If we examine the models induced from different training sets, there is more variability in the decision boundary of a 1-nearest neighbor classifier than a decision tree classifier. Therefore, the decision boundary of a decision tree classifier has a lower variance than the 1-nearest neighbor classifier.

5.6.4 Bagging

Bagging, which is also known as bootstrap aggregating, is a technique that repeatedly samples (with replacement) from a data set according to a uniform probability distribution. Each bootstrap sample has the same size as the original data. Because the sampling is done with replacement, some instances may appear several times in the same training set, while others may be omitted from the training set. On average, a bootstrap sample D_i contains approxi-



(a) Decision boundary for decision tree. (b) Decision boundary for 1-nearest neighbor.

Figure 5.34. Bias of decision tree and 1-nearest neighbor classifiers.

Algorithm 5.6 Bagging algorithm.

- 1: Let k be the number of bootstrap samples.
- 2: **for** $i = 1$ to k **do**
- 3: Create a bootstrap sample of size N , D_i .
- 4: Train a base classifier C_i on the bootstrap sample D_i .
- 5: **end for**
- 6: $C^*(x) = \operatorname{argmax}_y \sum_i \delta(C_i(x) = y)$.
 $\{\delta(\cdot) = 1 \text{ if its argument is true and } 0 \text{ otherwise}\}$.

mately 63% of the original training data because each sample has a probability $1 - (1 - 1/N)^N$ of being selected in each D_i . If N is sufficiently large, this probability converges to $1 - 1/e \simeq 0.632$. The basic procedure for bagging is summarized in Algorithm 5.6. After training the k classifiers, a test instance is assigned to the class that receives the highest number of votes.

To illustrate how bagging works, consider the data set shown in Table 5.4. Let x denote a one-dimensional attribute and y denote the class label. Suppose we apply a classifier that induces only one-level binary decision trees, with a test condition $x \leq k$, where k is a split point chosen to minimize the entropy of the leaf nodes. Such a tree is also known as a **decision stump**.

Without bagging, the best decision stump we can produce splits the records at either $x \leq 0.35$ or $x \leq 0.75$. Either way, the accuracy of the tree is at

Table 5.4. Example of data set used to construct an ensemble of bagging classifiers.

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	1	1	1	-1	-1	-1	-1	1	1	1

most 70%. Suppose we apply the bagging procedure on the data set using ten bootstrap samples. The examples chosen for training in each bagging round are shown in Figure 5.35. On the right-hand side of each table, we also illustrate the decision boundary produced by the classifier.

We classify the entire data set given in Table 5.4 by taking a majority vote among the predictions made by each base classifier. The results of the predictions are shown in Figure 5.36. Since the class labels are either -1 or $+1$, taking the majority vote is equivalent to summing up the predicted values of y and examining the sign of the resulting sum (refer to the second to last row in Figure 5.36). Notice that the ensemble classifier perfectly classifies all ten examples in the original data.

The preceding example illustrates another advantage of using ensemble methods in terms of enhancing the representation of the target function. Even though each base classifier is a decision stump, combining the classifiers can lead to a decision tree of depth 2.

Bagging improves generalization error by reducing the variance of the base classifiers. The performance of bagging depends on the stability of the base classifier. If a base classifier is unstable, bagging helps to reduce the errors associated with random fluctuations in the training data. If a base classifier is stable, i.e., robust to minor perturbations in the training set, then the error of the ensemble is primarily caused by bias in the base classifier. In this situation, bagging may not be able to improve the performance of the base classifiers significantly. It may even degrade the classifier's performance because the effective size of each training set is about 37% smaller than the original data.

Finally, since every sample has an equal probability of being selected, bagging does not focus on any particular instance of the training data. It is therefore less susceptible to model overfitting when applied to noisy data.

5.6.5 Boosting

Boosting is an iterative procedure used to adaptively change the distribution of training examples so that the base classifiers will focus on examples that are hard to classify. Unlike bagging, boosting assigns a weight to each training

Bagging Round 1:

x	0.1	0.2	0.2	0.3	0.4	0.4	0.5	0.6	0.9	0.9
y	1	1	1	1	-1	-1	-1	-1	1	1

$x \leq 0.35 \Rightarrow y = 1$
 $x > 0.35 \Rightarrow y = -1$

Bagging Round 2:

x	0.1	0.2	0.3	0.4	0.5	0.8	0.9	1	1	1
y	1	1	1	-1	-1	1	1	1	1	1

$x \leq 0.65 \Rightarrow y = 1$
 $x > 0.65 \Rightarrow y = -1$

Bagging Round 3:

x	0.1	0.2	0.3	0.4	0.4	0.5	0.7	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

$x \leq 0.35 \Rightarrow y = 1$
 $x > 0.35 \Rightarrow y = -1$

Bagging Round 4:

x	0.1	0.1	0.2	0.4	0.4	0.5	0.5	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

$x \leq 0.3 \Rightarrow y = 1$
 $x > 0.3 \Rightarrow y = -1$

Bagging Round 5:

x	0.1	0.1	0.2	0.5	0.6	0.6	0.6	1	1	1
y	1	1	1	-1	-1	-1	-1	1	1	1

$x \leq 0.35 \Rightarrow y = 1$
 $x > 0.35 \Rightarrow y = -1$

Bagging Round 6:

x	0.2	0.4	0.5	0.6	0.7	0.7	0.7	0.8	0.9	1
y	1	-1	-1	-1	-1	-1	-1	1	1	1

$x \leq 0.75 \Rightarrow y = -1$
 $x > 0.75 \Rightarrow y = 1$

Bagging Round 7:

x	0.1	0.4	0.4	0.6	0.7	0.8	0.9	0.9	0.9	1
y	1	-1	-1	-1	-1	1	1	1	1	1

$x \leq 0.75 \Rightarrow y = -1$
 $x > 0.75 \Rightarrow y = 1$

Bagging Round 8:

x	0.1	0.2	0.5	0.5	0.5	0.7	0.7	0.8	0.9	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

$x \leq 0.75 \Rightarrow y = -1$
 $x > 0.75 \Rightarrow y = 1$

Bagging Round 9:

x	0.1	0.3	0.4	0.4	0.6	0.7	0.7	0.8	1	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

$x \leq 0.75 \Rightarrow y = -1$
 $x > 0.75 \Rightarrow y = 1$

Bagging Round 10:

x	0.1	0.1	0.1	0.1	0.3	0.3	0.8	0.8	0.9	0.9
y	1	1	1	1	1	1	1	1	1	1

$x \leq 0.05 \Rightarrow y = -1$
 $x > 0.05 \Rightarrow y = 1$

Figure 5.35. Example of bagging.

example and may adaptively change the weight at the end of each boosting round. The weights assigned to the training examples can be used in the following ways:

1. They can be used as a sampling distribution to draw a set of bootstrap samples from the original data.
2. They can be used by the base classifier to learn a model that is biased toward higher-weight examples.

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	1	1	1	-1	-1	-1	-1	-1	-1	-1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
4	1	1	1	-1	-1	-1	-1	-1	-1	-1
5	1	1	1	-1	-1	-1	-1	-1	-1	-1
6	-1	-1	-1	-1	-1	-1	-1	1	1	1
7	-1	-1	-1	-1	-1	-1	-1	1	1	1
8	-1	-1	-1	-1	-1	-1	-1	1	1	1
9	-1	-1	-1	-1	-1	-1	-1	1	1	1
10	1	1	1	1	1	1	1	1	1	1
Sum	2	2	2	-6	-6	-6	-6	2	2	2
Sign	1	1	1	-1	-1	-1	-1	1	1	1
True Class	1	1	1	-1	-1	-1	-1	1	1	1

Figure 5.36. Example of combining classifiers constructed using the bagging approach.

This section describes an algorithm that uses weights of examples to determine the sampling distribution of its training set. Initially, the examples are assigned equal weights, $1/N$, so that they are equally likely to be chosen for training. A sample is drawn according to the sampling distribution of the training examples to obtain a new training set. Next, a classifier is induced from the training set and used to classify all the examples in the original data. The weights of the training examples are updated at the end of each boosting round. Examples that are classified incorrectly will have their weights increased, while those that are classified correctly will have their weights decreased. This forces the classifier to focus on examples that are difficult to classify in subsequent iterations.

The following table shows the examples chosen during each boosting round.

Boosting (Round 1):	7	3	2	8	7	9	4	10	6	3
Boosting (Round 2):	5	4	9	4	2	5	1	7	4	2
Boosting (Round 3):	4	4	8	10	4	5	4	6	3	4

Initially, all the examples are assigned the same weights. However, some examples may be chosen more than once, e.g., examples 3 and 7, because the sampling is done with replacement. A classifier built from the data is then used to classify all the examples. Suppose example 4 is difficult to classify. The weight for this example will be increased in future iterations as it gets misclassified repeatedly. Meanwhile, examples that were not chosen in the pre-

vious round, e.g., examples 1 and 5, also have a better chance of being selected in the next round since their predictions in the previous round were likely to be wrong. As the boosting rounds proceed, examples that are the hardest to classify tend to become even more prevalent. The final ensemble is obtained by aggregating the base classifiers obtained from each boosting round.

Over the years, several implementations of the boosting algorithm have been developed. These algorithms differ in terms of (1) how the weights of the training examples are updated at the end of each boosting round, and (2) how the predictions made by each classifier are combined. An implementation called AdaBoost is explored in the next section.

AdaBoost

Let $\{(\mathbf{x}_j, y_j) \mid j = 1, 2, \dots, N\}$ denote a set of N training examples. In the AdaBoost algorithm, the importance of a base classifier C_i depends on its error rate, which is defined as

$$\epsilon_i = \frac{1}{N} \left[\sum_{j=1}^N w_j I(C_i(\mathbf{x}_j) \neq y_j) \right], \quad (5.68)$$

where $I(p) = 1$ if the predicate p is true, and 0 otherwise. The importance of a classifier C_i is given by the following parameter,

$$\alpha_i = \frac{1}{2} \ln \left(\frac{1 - \epsilon_i}{\epsilon_i} \right).$$

Note that α_i has a large positive value if the error rate is close to 0 and a large negative value if the error rate is close to 1, as shown in Figure 5.37.

The α_i parameter is also used to update the weight of the training examples. To illustrate, let $w_i^{(j)}$ denote the weight assigned to example (\mathbf{x}_i, y_i) during the j^{th} boosting round. The weight update mechanism for AdaBoost is given by the equation:

$$w_i^{(j+1)} = \frac{w_i^{(j)}}{Z_j} \times \begin{cases} \exp^{-\alpha_j} & \text{if } C_j(\mathbf{x}_i) = y_i \\ \exp^{\alpha_j} & \text{if } C_j(\mathbf{x}_i) \neq y_i \end{cases}, \quad (5.69)$$

where Z_j is the normalization factor used to ensure that $\sum_i w_i^{(j+1)} = 1$. The weight update formula given in Equation 5.69 increases the weights of incorrectly classified examples and decreases the weights of those classified correctly.

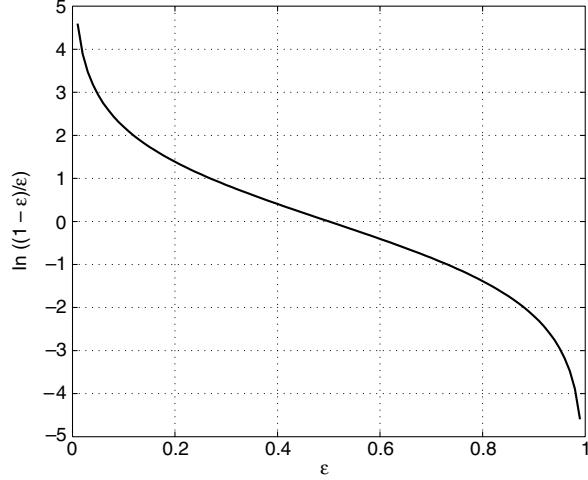


Figure 5.37. Plot of α as a function of training error ϵ .

Instead of using a majority voting scheme, the prediction made by each classifier C_j is weighted according to α_j . This approach allows AdaBoost to penalize models that have poor accuracy, e.g., those generated at the earlier boosting rounds. In addition, if any intermediate rounds produce an error rate higher than 50%, the weights are reverted back to their original uniform values, $w_i = 1/N$, and the resampling procedure is repeated. The AdaBoost algorithm is summarized in Algorithm 5.7.

Let us examine how the boosting approach works on the data set shown in Table 5.4. Initially, all the examples have identical weights. After three boosting rounds, the examples chosen for training are shown in Figure 5.38(a). The weights for each example are updated at the end of each boosting round using Equation 5.69.

Without boosting, the accuracy of the decision stump is, at best, 70%. With AdaBoost, the results of the predictions are given in Figure 5.39(b). The final prediction of the ensemble classifier is obtained by taking a weighted average of the predictions made by each base classifier, which is shown in the last row of Figure 5.39(b). Notice that AdaBoost perfectly classifies all the examples in the training data.

An important analytical result of boosting shows that the training error of the ensemble is bounded by the following expression:

$$e_{\text{ensemble}} \leq \prod_i \left[\sqrt{\epsilon_i(1 - \epsilon_i)} \right], \quad (5.70)$$

Algorithm 5.7 AdaBoost algorithm.

- 1: $\mathbf{w} = \{w_j = 1/N \mid j = 1, 2, \dots, N\}$. {Initialize the weights for all N examples.}
- 2: Let k be the number of boosting rounds.
- 3: **for** $i = 1$ to k **do**
- 4: Create training set D_i by sampling (with replacement) from D according to \mathbf{w} .
- 5: Train a base classifier C_i on D_i .
- 6: Apply C_i to all examples in the original training set, D .
- 7: $\epsilon_i = \frac{1}{N} [\sum_j w_j \delta(C_i(x_j) \neq y_j)]$ {Calculate the weighted error.}
- 8: **if** $\epsilon_i > 0.5$ **then**
- 9: $\mathbf{w} = \{w_j = 1/N \mid j = 1, 2, \dots, N\}$. {Reset the weights for all N examples.}
- 10: Go back to Step 4.
- 11: **end if**
- 12: $\alpha_i = \frac{1}{2} \ln \frac{1-\epsilon_i}{\epsilon_i}$.
- 13: Update the weight of each example according to Equation 5.69.
- 14: **end for**
- 15: $C^*(\mathbf{x}) = \operatorname{argmax}_y \sum_{j=1}^T \alpha_j \delta(C_j(\mathbf{x}) = y))$.

where ϵ_i is the error rate of each base classifier i . If the error rate of the base classifier is less than 50%, we can write $\epsilon_i = 0.5 - \gamma_i$, where γ_i measures how much better the classifier is than random guessing. The bound on the training error of the ensemble becomes

$$e_{\text{ensemble}} \leq \prod_i \sqrt{1 - 4\gamma_i^2} \leq \exp \left(-2 \sum_i \gamma_i^2 \right). \quad (5.71)$$

If $\gamma_i < \gamma^*$ for all i 's, then the training error of the ensemble decreases exponentially, which leads to the fast convergence of the algorithm. Nevertheless, because of its tendency to focus on training examples that are wrongly classified, the boosting technique can be quite susceptible to overfitting.

5.6.6 Random Forests

Random forest is a class of ensemble methods specifically designed for decision tree classifiers. It combines the predictions made by multiple decision trees, where each tree is generated based on the values of an independent set of random vectors, as shown in Figure 5.40. The random vectors are generated from a fixed probability distribution, unlike the adaptive approach used in AdaBoost, where the probability distribution is varied to focus on examples that are hard to classify. Bagging using decision trees is a special case of random forests, where randomness is injected into the model-building process

Boosting Round 1:

x	0.1	0.4	0.5	0.6	0.6	0.7	0.7	0.7	0.8	1
y	1	-1	-1	-1	-1	-1	-1	-1	1	1

Boosting Round 2:

x	0.1	0.1	0.2	0.2	0.2	0.2	0.3	0.3	0.3	0.3
y	1	1	1	1	1	1	1	1	1	1

Boosting Round 3:

x	0.2	0.2	0.4	0.4	0.4	0.4	0.5	0.6	0.6	0.7
y	1	1	-1	-1	-1	-1	-1	-1	-1	-1

(a) Training records chosen during boosting

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
2	0.311	0.311	0.311	0.01	0.01	0.01	0.01	0.01	0.01	0.01
3	0.029	0.029	0.029	0.228	0.228	0.228	0.228	0.009	0.009	0.009

(b) Weights of training records

Figure 5.38. Example of boosting.

by randomly choosing N samples, with replacement, from the original training set. Bagging also uses the same uniform probability distribution to generate its bootstrapped samples throughout the entire model-building process.

It was theoretically proven that the upper bound for generalization error of random forests converges to the following expression, when the number of trees is sufficiently large.

$$\text{Generalization error} \leq \frac{\bar{\rho}(1-s^2)}{s^2}, \quad (5.72)$$

where $\bar{\rho}$ is the average correlation among the trees and s is a quantity that measures the “strength” of the tree classifiers. The strength of a set of classifiers refers to the average performance of the classifiers, where performance is measured probabilistically in terms of the classifier’s margin:

$$\text{margin, } M(\mathbf{X}, Y) = P(\hat{Y}_\theta = Y) - \max_{Z \neq Y} P(\hat{Y}_\theta = Z), \quad (5.73)$$

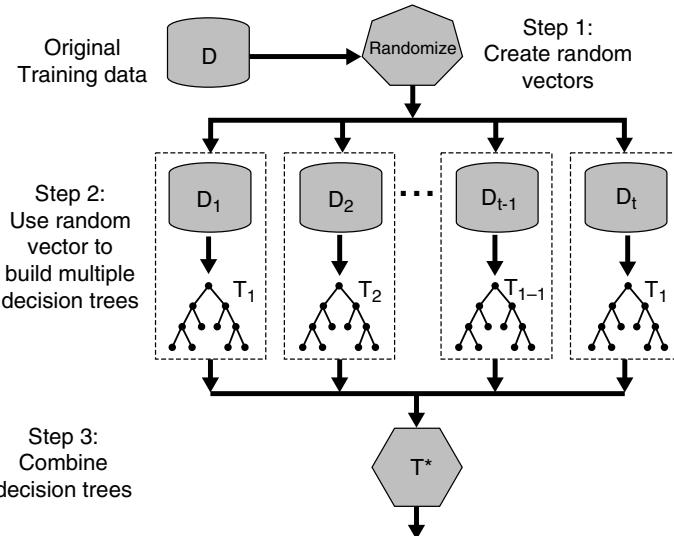
where \hat{Y}_θ is the predicted class of \mathbf{X} according to a classifier built from some random vector θ . The higher the margin is, the more likely it is that the

Round	Split Point	Left Class	Right Class	α
1	0.75	-1	1	1.738
2	0.05	1	1	2.7784
3	0.3	1	-1	4.1195

(a)

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	-1	-1	-1	-1	-1	-1	-1	1	1	1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
Sum	5.16	5.16	5.16	-3.08	-3.08	-3.08	-3.08	0.397	0.397	0.397
Sign	1	1	1	-1	-1	-1	-1	1	1	1

(b)

Figure 5.39. Example of combining classifiers constructed using the AdaBoost approach.**Figure 5.40.** Random forests.

classifier correctly predicts a given example \mathbf{X} . Equation 5.72 is quite intuitive; as the trees become more correlated or the strength of the ensemble decreases, the generalization error bound tends to increase. Randomization helps to reduce the correlation among decision trees so that the generalization error of the ensemble can be improved.

Each decision tree uses a random vector that is generated from some fixed probability distribution. A random vector can be incorporated into the tree-growing process in many ways. The first approach is to randomly select F input features to split at each node of the decision tree. As a result, instead of examining all the available features, the decision to split a node is determined from these selected F features. The tree is then grown to its entirety without any pruning. This may help reduce the bias present in the resulting tree. Once the trees have been constructed, the predictions are combined using a majority voting scheme. This approach is known as Forest-RI, where RI refers to random input selection. To increase randomness, bagging can also be used to generate bootstrap samples for Forest-RI. The strength and correlation of random forests may depend on the size of F . If F is sufficiently small, then the trees tend to become less correlated. On the other hand, the strength of the tree classifier tends to improve with a larger number of features, F . As a tradeoff, the number of features is commonly chosen to be $F = \log_2 d + 1$, where d is the number of input features. Since only a subset of the features needs to be examined at each node, this approach helps to significantly reduce the runtime of the algorithm.

If the number of original features d is too small, then it is difficult to choose an independent set of random features for building the decision trees. One way to increase the feature space is to create linear combinations of the input features. Specifically, at each node, a new feature is generated by randomly selecting L of the input features. The input features are linearly combined using coefficients generated from a uniform distribution in the range of $[-1, 1]$. At each node, F of such randomly combined new features are generated, and the best of them is subsequently selected to split the node. This approach is known as Forest-RC.

A third approach for generating the random trees is to randomly select one of the F best splits at each node of the decision tree. This approach may potentially generate trees that are more correlated than Forest-RI and Forest-RC, unless F is sufficiently large. It also does not have the runtime savings of Forest-RI and Forest-RC because the algorithm must examine all the splitting features at each node of the decision tree.

It has been shown empirically that the classification accuracies of random forests are quite comparable to the AdaBoost algorithm. It is also more robust to noise and runs much faster than the AdaBoost algorithm. The classification accuracies of various ensemble algorithms are compared in the next section.

Table 5.5. Comparing the accuracy of a decision tree classifier against three ensemble methods.

Data Set	Number of (Attributes, Classes, Records)	Decision Tree (%)	Bagging (%)	Boosting (%)	RF (%)
Anneal	(39, 6, 898)	92.09	94.43	95.43	95.43
Australia	(15, 2, 690)	85.51	87.10	85.22	85.80
Auto	(26, 7, 205)	81.95	85.37	85.37	84.39
Breast	(11, 2, 699)	95.14	96.42	97.28	96.14
Cleve	(14, 2, 303)	76.24	81.52	82.18	82.18
Credit	(16, 2, 690)	85.8	86.23	86.09	85.8
Diabetes	(9, 2, 768)	72.40	76.30	73.18	75.13
German	(21, 2, 1000)	70.90	73.40	73.00	74.5
Glass	(10, 7, 214)	67.29	76.17	77.57	78.04
Heart	(14, 2, 270)	80.00	81.48	80.74	83.33
Hepatitis	(20, 2, 155)	81.94	81.29	83.87	83.23
Horse	(23, 2, 368)	85.33	85.87	81.25	85.33
Ionosphere	(35, 2, 351)	89.17	92.02	93.73	93.45
Iris	(5, 3, 150)	94.67	94.67	94.00	93.33
Labor	(17, 2, 57)	78.95	84.21	89.47	84.21
Led7	(8, 10, 3200)	73.34	73.66	73.34	73.06
Lymphography	(19, 4, 148)	77.03	79.05	85.14	82.43
Pima	(9, 2, 768)	74.35	76.69	73.44	77.60
Sonar	(61, 2, 208)	78.85	78.85	84.62	85.58
Tic-tac-toe	(10, 2, 958)	83.72	93.84	98.54	95.82
Vehicle	(19, 4, 846)	71.04	74.11	78.25	74.94
Waveform	(22, 3, 5000)	76.44	83.30	83.90	84.04
Wine	(14, 3, 178)	94.38	96.07	97.75	97.75
Zoo	(17, 7, 101)	93.07	93.07	95.05	97.03

5.6.7 Empirical Comparison among Ensemble Methods

Table 5.5 shows the empirical results obtained when comparing the performance of a decision tree classifier against bagging, boosting, and random forest. The base classifiers used in each ensemble method consist of fifty decision trees. The classification accuracies reported in this table are obtained from ten-fold cross-validation. Notice that the ensemble classifiers generally outperform a single decision tree classifier on many of the data sets.

5.7 Class Imbalance Problem

Data sets with imbalanced class distributions are quite common in many real applications. For example, an automated inspection system that monitors products that come off a manufacturing assembly line may find that the num-

ber of defective products is significantly fewer than that of non-defective products. Similarly, in credit card fraud detection, fraudulent transactions are outnumbered by legitimate transactions. In both of these examples, there is a disproportionate number of instances that belong to different classes. The degree of imbalance varies from one application to another—a manufacturing plant operating under the six sigma principle may discover four defects in a million products shipped to their customers, while the amount of credit card fraud may be of the order of 1 in 100. Despite their infrequent occurrences, a correct classification of the rare class in these applications often has greater value than a correct classification of the majority class. However, because the class distribution is imbalanced, this presents a number of problems to existing classification algorithms.

The accuracy measure, which is used extensively to compare the performance of classifiers, may not be well suited for evaluating models derived from imbalanced data sets. For example, if 1% of the credit card transactions are fraudulent, then a model that predicts every transaction as legitimate has an accuracy of 99% even though it fails to detect any of the fraudulent activities. Additionally, measures that are used to guide the learning algorithm (e.g., information gain for decision tree induction) may need to be modified to focus on the rare class.

Detecting instances of the rare class is akin to finding a needle in a haystack. Because their instances occur infrequently, models that describe the rare class tend to be highly specialized. For example, in a rule-based classifier, the rules extracted for the rare class typically involve a large number of attributes and cannot be easily simplified into more general rules with broader coverage (unlike the rules for the majority class). Such models are also susceptible to the presence of noise in training data. As a result, many of the existing classification algorithms may not effectively detect instances of the rare class.

This section presents some of the methods developed for handling the class imbalance problem. First, alternative metrics besides accuracy are introduced, along with a graphical method called ROC analysis. We then describe how cost-sensitive learning and sampling-based methods may be used to improve the detection of rare classes.

5.7.1 Alternative Metrics

Since the accuracy measure treats every class as equally important, it may not be suitable for analyzing imbalanced data sets, where the rare class is considered more interesting than the majority class. For binary classification, the rare class is often denoted as the positive class, while the majority class is

Table 5.6. A confusion matrix for a binary classification problem in which the classes are not equally important.

		Predicted Class	
		+	-
Actual Class	+	f_{++} (TP)	f_{+-} (FN)
	-	f_{-+} (FP)	f_{--} (TN)

denoted as the negative class. A confusion matrix that summarizes the number of instances predicted correctly or incorrectly by a classification model is shown in Table 5.6.

The following terminology is often used when referring to the counts tabulated in a confusion matrix:

- True positive (TP) or f_{++} , which corresponds to the number of positive examples correctly predicted by the classification model.
- False negative (FN) or f_{+-} , which corresponds to the number of positive examples wrongly predicted as negative by the classification model.
- False positive (FP) or f_{-+} , which corresponds to the number of negative examples wrongly predicted as positive by the classification model.
- True negative (TN) or f_{--} , which corresponds to the number of negative examples correctly predicted by the classification model.

The counts in a confusion matrix can also be expressed in terms of percentages. The **true positive rate (TPR)** or **sensitivity** is defined as the fraction of positive examples predicted correctly by the model, i.e.,

$$TPR = TP/(TP + FN).$$

Similarly, the **true negative rate (TNR)** or **specificity** is defined as the fraction of negative examples predicted correctly by the model, i.e.,

$$TNR = TN/(TN + FP).$$

Finally, the **false positive rate (FPR)** is the fraction of negative examples predicted as a positive class, i.e.,

$$FPR = FP/(TN + FP),$$

while the **false negative rate** (FNR) is the fraction of positive examples predicted as a negative class, i.e.,

$$FNR = FN/(TP + FN).$$

Recall and **precision** are two widely used metrics employed in applications where successful detection of one of the classes is considered more significant than detection of the other classes. A formal definition of these metrics is given below.

$$\text{Precision, } p = \frac{TP}{TP + FP} \quad (5.74)$$

$$\text{Recall, } r = \frac{TP}{TP + FN} \quad (5.75)$$

Precision determines the fraction of records that actually turns out to be positive in the group the classifier has declared as a positive class. The higher the precision is, the lower the number of false positive errors committed by the classifier. Recall measures the fraction of positive examples correctly predicted by the classifier. Classifiers with large recall have very few positive examples misclassified as the negative class. In fact, the value of recall is equivalent to the true positive rate.

It is often possible to construct baseline models that maximize one metric but not the other. For example, a model that declares every record to be the positive class will have a perfect recall, but very poor precision. Conversely, a model that assigns a positive class to every test record that matches one of the positive records in the training set has very high precision, but low recall. Building a model that maximizes both precision and recall is the key challenge of classification algorithms.

Precision and recall can be summarized into another metric known as the F_1 measure.

$$F_1 = \frac{2rp}{r + p} = \frac{2 \times TP}{2 \times TP + FP + FN} \quad (5.76)$$

In principle, F_1 represents a harmonic mean between recall and precision, i.e.,

$$F_1 = \frac{2}{\frac{1}{r} + \frac{1}{p}}.$$

The harmonic mean of two numbers x and y tends to be closer to the smaller of the two numbers. Hence, a high value of F_1 -measure ensures that both

precision and recall are reasonably high. A comparison among harmonic, geometric, and arithmetic means is given in the next example.

Example 5.8. Consider two positive numbers $a = 1$ and $b = 5$. Their arithmetic mean is $\mu_a = (a + b)/2 = 3$ and their geometric mean is $\mu_g = \sqrt{ab} = 2.236$. Their harmonic mean is $\mu_h = (2 \times 1 \times 5)/6 = 1.667$, which is closer to the smaller value between a and b than the arithmetic and geometric means. ■

More generally, the F_β measure can be used to examine the tradeoff between recall and precision:

$$F_\beta = \frac{(\beta^2 + 1)rp}{r + \beta^2 p} = \frac{(\beta^2 + 1) \times TP}{(\beta^2 + 1)TP + \beta^2 FP + FN}. \quad (5.77)$$

Both precision and recall are special cases of F_β by setting $\beta = 0$ and $\beta = \infty$, respectively. Low values of β make F_β closer to precision, and high values make it closer to recall.

A more general metric that captures F_β as well as accuracy is the weighted accuracy measure, which is defined by the following equation:

$$\text{Weighted accuracy} = \frac{w_1 TP + w_4 TN}{w_1 TP + w_2 FP + w_3 FN + w_4 TN}. \quad (5.78)$$

The relationship between weighted accuracy and other performance metrics is summarized in the following table:

Measure	w_1	w_2	w_3	w_4
Recall	1	1	0	0
Precision	1	0	1	0
F_β	$\beta^2 + 1$	β^2	1	0
Accuracy	1	1	1	1

5.7.2 The Receiver Operating Characteristic Curve

A receiver operating characteristic (ROC) curve is a graphical approach for displaying the tradeoff between true positive rate and false positive rate of a classifier. In an ROC curve, the true positive rate (TPR) is plotted along the y axis and the false positive rate (FPR) is shown on the x axis. Each point along the curve corresponds to one of the models induced by the classifier. Figure 5.41 shows the ROC curves for a pair of classifiers, M_1 and M_2 .

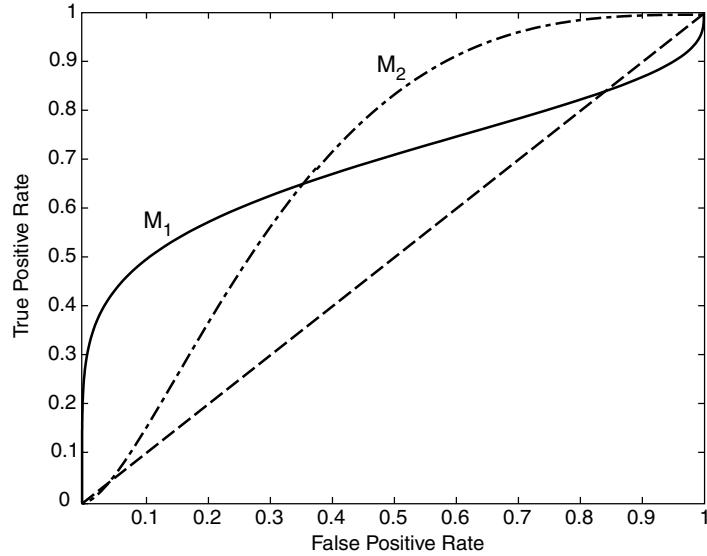


Figure 5.41. ROC curves for two different classifiers.

There are several critical points along an ROC curve that have well-known interpretations:

- (TPR=0, FPR=0): Model predicts every instance to be a negative class.
- (TPR=1, FPR=1): Model predicts every instance to be a positive class.
- (TPR=1, FPR=0): The ideal model.

A good classification model should be located as close as possible to the upper left corner of the diagram, while a model that makes random guesses should reside along the main diagonal, connecting the points ($TPR = 0, FPR = 0$) and ($TPR = 1, FPR = 1$). Random guessing means that a record is classified as a positive class with a fixed probability p , irrespective of its attribute set. For example, consider a data set that contains n_+ positive instances and n_- negative instances. The random classifier is expected to correctly classify pn_+ of the positive instances and to misclassify pn_- of the negative instances. Therefore, the TPR of the classifier is $(pn_+)/n_+ = p$, while its FPR is $(pn_-)/p = p$. Since the TPR and FPR are identical, the ROC curve for a random classifier always reside along the main diagonal.

An ROC curve is useful for comparing the relative performance among different classifiers. In Figure 5.41, M_1 is better than M_2 when FPR is less

than 0.36, while M_2 is superior when FPR is greater than 0.36. Clearly, neither of these two classifiers dominates the other.

The area under the ROC curve (AUC) provides another approach for evaluating which model is better on average. If the model is perfect, then its area under the ROC curve would equal 1. If the model simply performs random guessing, then its area under the ROC curve would equal 0.5. A model that is strictly better than another would have a larger area under the ROC curve.

Generating an ROC curve

To draw an ROC curve, the classifier should be able to produce a continuous-valued output that can be used to rank its predictions, from the most likely record to be classified as a positive class to the least likely record. These outputs may correspond to the posterior probabilities generated by a Bayesian classifier or the numeric-valued outputs produced by an artificial neural network. The following procedure can then be used to generate an ROC curve:

1. Assuming that the continuous-valued outputs are defined for the positive class, sort the test records in increasing order of their output values.
2. Select the lowest ranked test record (i.e., the record with lowest output value). Assign the selected record and those ranked above it to the positive class. This approach is equivalent to classifying all the test records as positive class. Because all the positive examples are classified correctly and the negative examples are misclassified, $TPR = FPR = 1$.
3. Select the next test record from the sorted list. Classify the selected record and those ranked above it as positive, while those ranked below it as negative. Update the counts of TP and FP by examining the actual class label of the previously selected record. If the previously selected record is a positive class, the TP count is decremented and the FP count remains the same as before. If the previously selected record is a negative class, the FP count is decremented and TP count remains the same as before.
4. Repeat Step 3 and update the TP and FP counts accordingly until the highest ranked test record is selected.
5. Plot the TPR against FPR of the classifier.

Figure 5.42 shows an example of how to compute the ROC curve. There are five positive examples and five negative examples in the test set. The class

Class	+	-	+	-	-	-	+	-	+	+	+	
	0.25	0.43	0.53	0.76	0.85	0.85	0.85	0.87	0.93	0.95	1.00	
TP	5	4	4	3	3	3	3	2	2	1	0	
FP	5	5	4	4	3	2	1	1	0	0	0	
TN	0	0	1	1	2	3	4	4	5	5	5	
FN	0	1	1	2	2	2	2	3	3	4	5	
TPR	1	0.8	0.8	0.6	0.6	0.6	0.6	0.4	0.4	0.2	0	
FPR	1	1	0.8	0.8	0.6	0.4	0.2	0.2	0	0	0	

Figure 5.42. Constructing an ROC curve.

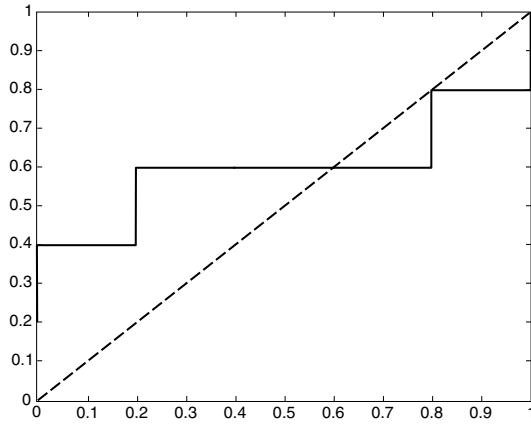


Figure 5.43. ROC curve for the data shown in Figure 5.42.

labels of the test records are shown in the first row of the table. The second row corresponds to the sorted output values for each record. For example, they may correspond to the posterior probabilities $P(+|\mathbf{x})$ generated by a naïve Bayes classifier. The next six rows contain the counts of TP , FP , TN , and FN , along with their corresponding TPR and FPR . The table is then filled from left to right. Initially, all the records are predicted to be positive. Thus, $TP = FP = 5$ and $TPR = FPR = 1$. Next, we assign the test record with the lowest output value as the negative class. Because the selected record is actually a positive example, the TP count reduces from 5 to 4 and the FP count is the same as before. The FPR and TPR are updated accordingly. This process is repeated until we reach the end of the list, where $TPR = 0$ and $FPR = 0$. The ROC curve for this example is shown in Figure 5.43.

5.7.3 Cost-Sensitive Learning

A cost matrix encodes the penalty of classifying records from one class as another. Let $C(i, j)$ denote the cost of predicting a record from class i as class j . With this notation, $C(+, -)$ is the cost of committing a false negative error, while $C(-, +)$ is the cost of generating a false alarm. A negative entry in the cost matrix represents the reward for making correct classification. Given a collection of N test records, the overall cost of a model M is

$$\begin{aligned} C_t(M) = & \quad TP \times C(+, +) + FP \times C(-, +) + FN \times C(+, -) \\ & + TN \times C(-, -). \end{aligned} \quad (5.79)$$

Under the 0/1 cost matrix, i.e., $C(+, +) = C(-, -) = 0$ and $C(+, -) = C(-, +) = 1$, it can be shown that the overall cost is equivalent to the number of misclassification errors.

$$C_t(M) = 0 \times (TP + TN) + 1 \times (FP + FN) = N \times Err, \quad (5.80)$$

where Err is the error rate of the classifier.

Example 5.9. Consider the cost matrix shown in Table 5.7: The cost of committing a false negative error is a hundred times larger than the cost of committing a false alarm. In other words, failure to detect any positive example is just as bad as committing a hundred false alarms. Given the classification models with the confusion matrices shown in Table 5.8, the total cost for each model is

$$\begin{aligned} C_t(M_1) &= 150 \times (-1) + 60 \times 1 + 40 \times 100 = 3910, \\ C_t(M_2) &= 250 \times (-1) + 5 \times 1 + 45 \times 100 = 4255. \end{aligned}$$

Table 5.7. Cost matrix for Example 5.9.

		Predicted Class	
		Class = +	Class = -
Actual	Class = +	-1	100
	Class = -	1	0

Table 5.8. Confusion matrix for two classification models.

Model M_1		Predicted Class		Model M_2		Predicted Class	
		Class +	Class -			Class +	Class -
Actual Class	Class +	150	40	Actual Class	Class +	250	45
	Class -	60	250		Class -	5	200

Notice that despite improving both of its true positive and false positive counts, model M_2 is still inferior since the improvement comes at the expense of increasing the more costly false negative errors. A standard accuracy measure would have preferred model M_2 over M_1 . ■

A cost-sensitive classification technique takes the cost matrix into consideration during model building and generates a model that has the lowest cost. For example, if false negative errors are the most costly, the learning algorithm will try to reduce these errors by extending its decision boundary toward the negative class, as shown in Figure 5.44. In this way, the generated model can cover more positive examples, although at the expense of generating additional false alarms.

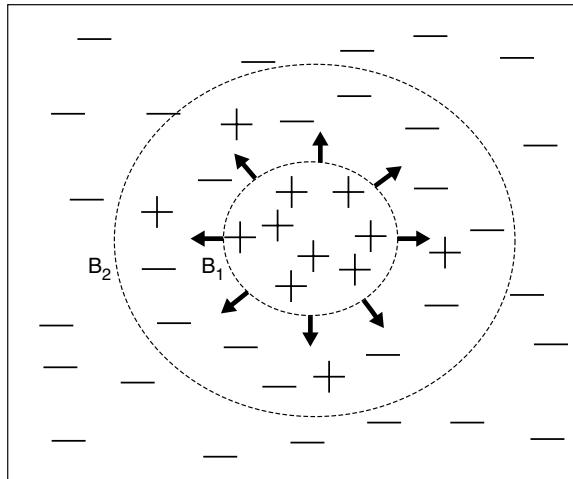


Figure 5.44. Modifying the decision boundary (from B_1 to B_2) to reduce the false negative errors of a classifier.

There are various ways to incorporate cost information into classification algorithms. For example, in the context of decision tree induction, the cost

information can be used to: (1) choose the best attribute to use for splitting the data, (2) determine whether a subtree should be pruned, (3) manipulate the weights of the training records so that the learning algorithm converges to a decision tree that has the lowest cost, and (4) modify the decision rule at each leaf node. To illustrate the last approach, let $p(i|t)$ denote the fraction of training records from class i that belong to the leaf node t . A typical decision rule for a binary classification problem assigns the positive class to node t if the following condition holds.

$$\begin{aligned} p(+|t) &> p(-|t) \\ \implies p(+|t) &> (1 - p(+|t)) \\ \implies 2p(+|t) &> 1 \\ \implies p(+|t) &> 0.5. \end{aligned} \tag{5.81}$$

The preceding decision rule suggests that the class label of a leaf node depends on the majority class of the training records that reach the particular node. Note that this rule assumes that the misclassification costs are identical for both positive and negative examples. This decision rule is equivalent to the expression given in Equation 4.8 on page 165.

Instead of taking a majority vote, a cost-sensitive algorithm assigns the class label i to node t if it minimizes the following expression:

$$C(i|t) = \sum_j p(j|t)C(j, i). \tag{5.82}$$

In the case where $C(+, +) = C(-, -) = 0$, a leaf node t is assigned to the positive class if:

$$\begin{aligned} p(+|t)C(+, -) &> p(-|t)C(-, +) \\ \implies p(+|t)C(+, -) &> (1 - p(+|t))C(-, +) \\ \implies p(+|t) &> \frac{C(-, +)}{C(-, +) + C(+, -)}. \end{aligned} \tag{5.83}$$

This expression suggests that we can modify the threshold of the decision rule from 0.5 to $C(-, +)/(C(-, +) + C(+, -))$ to obtain a cost-sensitive classifier. If $C(-, +) < C(+, -)$, then the threshold will be less than 0.5. This result makes sense because the cost of making a false negative error is more expensive than that for generating a false alarm. Lowering the threshold will expand the decision boundary toward the negative class, as shown in Figure 5.44.

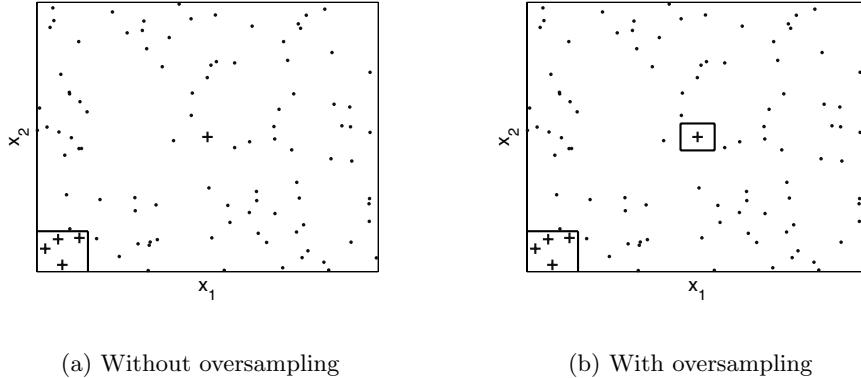


Figure 5.45. Illustrating the effect of oversampling of the rare class.

5.7.4 Sampling-Based Approaches

Sampling is another widely used approach for handling the class imbalance problem. The idea of sampling is to modify the distribution of instances so that the rare class is well represented in the training set. Some of the available techniques for sampling include undersampling, oversampling, and a hybrid of both approaches. To illustrate these techniques, consider a data set that contains 100 positive examples and 1000 negative examples.

In the case of undersampling, a random sample of 100 negative examples is chosen to form the training set along with all the positive examples. One potential problem with this approach is that some of the useful negative examples may not be chosen for training, therefore, resulting in a less than optimal model. A potential method to overcome this problem is to perform undersampling multiple times and to induce multiple classifiers similar to the ensemble learning approach. Focused undersampling methods may also be used, where the sampling procedure makes an informed choice with regard to the negative examples that should be eliminated, e.g., those located far away from the decision boundary.

Oversampling replicates the positive examples until the training set has an equal number of positive and negative examples. Figure 5.45 illustrates the effect of oversampling on the construction of a decision boundary using a classifier such as a decision tree. Without oversampling, only the positive examples at the bottom right-hand side of Figure 5.45(a) are classified correctly. The positive example in the middle of the diagram is misclassified because there

are not enough examples to justify the creation of a new decision boundary to separate the positive and negative instances. Oversampling provides the additional examples needed to ensure that the decision boundary surrounding the positive example is not pruned, as illustrated in Figure 5.45(b).

However, for noisy data, oversampling may cause model overfitting because some of the noise examples may be replicated many times. In principle, oversampling does not add any new information into the training set. Replication of positive examples only prevents the learning algorithm from pruning certain parts of the model that describe regions that contain very few training examples (i.e., the small disjuncts). The additional positive examples also tend to increase the computation time for model building.

The hybrid approach uses a combination of undersampling the majority class and oversampling the rare class to achieve uniform class distribution. Undersampling can be performed using random or focused subsampling. Oversampling, on the other hand, can be done by replicating the existing positive examples or generating new positive examples in the neighborhood of the existing positive examples. In the latter approach, we must first determine the k -nearest neighbors for each existing positive example. A new positive example is then generated at some random point along the line segment that joins the positive example to one of its k -nearest neighbors. This process is repeated until the desired number of positive examples is reached. Unlike the data replication approach, the new examples allow us to extend the decision boundary for the positive class outward, similar to the approach shown in Figure 5.44. Nevertheless, this approach may still be quite susceptible to model overfitting.

5.8 Multiclass Problem

Some of the classification techniques described in this chapter, such as support vector machines and AdaBoost, are originally designed for binary classification problems. Yet there are many real-world problems, such as character recognition, face identification, and text classification, where the input data is divided into more than two categories. This section presents several approaches for extending the binary classifiers to handle multiclass problems. To illustrate these approaches, let $Y = \{y_1, y_2, \dots, y_K\}$ be the set of classes of the input data.

The first approach decomposes the multiclass problem into K binary problems. For each class $y_i \in Y$, a binary problem is created where all instances that belong to y_i are considered positive examples, while the remaining in-

stances are considered negative examples. A binary classifier is then constructed to separate instances of class y_i from the rest of the classes. This is known as the one-against-rest (1-r) approach.

The second approach, which is known as the one-against-one (1-1) approach, constructs $K(K - 1)/2$ binary classifiers, where each classifier is used to distinguish between a pair of classes, (y_i, y_j) . Instances that do not belong to either y_i or y_j are ignored when constructing the binary classifier for (y_i, y_j) . In both 1-r and 1-1 approaches, a test instance is classified by combining the predictions made by the binary classifiers. A voting scheme is typically employed to combine the predictions, where the class that receives the highest number of votes is assigned to the test instance. In the 1-r approach, if an instance is classified as negative, then all classes except for the positive class receive a vote. This approach, however, may lead to ties among the different classes. Another possibility is to transform the outputs of the binary classifiers into probability estimates and then assign the test instance to the class that has the highest probability.

Example 5.10. Consider a multiclass problem where $Y = \{y_1, y_2, y_3, y_4\}$. Suppose a test instance is classified as $(+, -, -, -)$ according to the 1-r approach. In other words, it is classified as positive when y_1 is used as the positive class and negative when y_2 , y_3 , and y_4 are used as the positive class. Using a simple majority vote, notice that y_1 receives the highest number of votes, which is four, while the remaining classes receive only three votes. The test instance is therefore classified as y_1 .

Suppose the test instance is classified as follows using the 1-1 approach:

Binary pair of classes	$+ : y_1$	$+ : y_1$	$+ : y_1$	$+ : y_2$	$+ : y_2$	$+ : y_3$
Classification	+	+	-	+	-	+

The first two rows in this table correspond to the pair of classes (y_i, y_j) chosen to build the classifier and the last row represents the predicted class for the test instance. After combining the predictions, y_1 and y_4 each receive two votes, while y_2 and y_3 each receives only one vote. The test instance is therefore classified as either y_1 or y_4 , depending on the tie-breaking procedure. ■

Error-Correcting Output Coding

A potential problem with the previous two approaches is that they are sensitive to the binary classification errors. For the 1-r approach given in Example 5.10,

if at least one of the binary classifiers makes a mistake in its prediction, then the ensemble may end up declaring a tie between classes or making a wrong prediction. For example, suppose the test instance is classified as $(+, -, +, -)$ due to misclassification by the third classifier. In this case, it will be difficult to tell whether the instance should be classified as y_1 or y_3 , unless the probability associated with each class prediction is taken into account.

The error-correcting output coding (ECOC) method provides a more robust way for handling multiclass problems. The method is inspired by an information-theoretic approach for sending messages across noisy channels. The idea behind this approach is to add redundancy into the transmitted message by means of a codeword, so that the receiver may detect errors in the received message and perhaps recover the original message if the number of errors is small.

For multiclass learning, each class y_i is represented by a unique bit string of length n known as its codeword. We then train n binary classifiers to predict each bit of the codeword string. The predicted class of a test instance is given by the codeword whose Hamming distance is closest to the codeword produced by the binary classifiers. Recall that the Hamming distance between a pair of bit strings is given by the number of bits that differ.

Example 5.11. Consider a multiclass problem where $Y = \{y_1, y_2, y_3, y_4\}$. Suppose we encode the classes using the following 7-bit codewords:

Class	Codeword						
	1	1	1	1	1	1	1
y_1	1	1	1	1	1	1	1
y_2	0	0	0	0	1	1	1
y_3	0	0	1	1	0	0	1
y_4	0	1	0	1	0	1	0

Each bit of the codeword is used to train a binary classifier. If a test instance is classified as $(0,1,1,1,1,1,1)$ by the binary classifiers, then the Hamming distance between the codeword and y_1 is 1, while the Hamming distance to the remaining classes is 3. The test instance is therefore classified as y_1 . ■

An interesting property of an error-correcting code is that if the minimum Hamming distance between any pair of codewords is d , then any $\lfloor(d-1)/2\rfloor$ errors in the output code can be corrected using its nearest codeword. In Example 5.11, because the minimum Hamming distance between any pair of codewords is 4, the ensemble may tolerate errors made by one of the seven

binary classifiers. If there is more than one classifier that makes a mistake, then the ensemble may not be able to compensate for the error.

An important issue is how to design the appropriate set of codewords for different classes. From coding theory, a vast number of algorithms have been developed for generating n -bit codewords with bounded Hamming distance. However, the discussion of these algorithms is beyond the scope of this book. It is worthwhile mentioning that there is a significant difference between the design of error-correcting codes for communication tasks compared to those used for multiclass learning. For communication, the codewords should maximize the Hamming distance between the rows so that error correction can be performed. Multiclass learning, however, requires that the row-wise and column-wise distances of the codewords must be well separated. A larger column-wise distance ensures that the binary classifiers are mutually independent, which is an important requirement for ensemble learning methods.

5.9 Bibliographic Notes

Mitchell [208] provides an excellent coverage on many classification techniques from a machine learning perspective. Extensive coverage on classification can also be found in Duda et al. [180], Webb [219], Fukunaga [187], Bishop [159], Hastie et al. [192], Cherkassky and Mulier [167], Witten and Frank [221], Hand et al. [190], Han and Kamber [189], and Dunham [181].

Direct methods for rule-based classifiers typically employ the sequential covering scheme for inducing classification rules. Holte's 1R [195] is the simplest form of a rule-based classifier because its rule set contains only a single rule. Despite its simplicity, Holte found that for some data sets that exhibit a strong one-to-one relationship between the attributes and the class label, 1R performs just as well as other classifiers. Other examples of rule-based classifiers include IREP [184], RIPPER [170], CN2 [168, 169], AQ [207], RISE [176], and ITRULE [214]. Table 5.9 shows a comparison of the characteristics of four of these classifiers.

For rule-based classifiers, the rule antecedent can be generalized to include any propositional or first-order logical expression (e.g., Horn clauses). Readers who are interested in first-order logic rule-based classifiers may refer to references such as [208] or the vast literature on inductive logic programming [209]. Quinlan [211] proposed the C4.5rules algorithm for extracting classification rules from decision trees. An indirect method for extracting rules from artificial neural networks was given by Andrews et al. in [157].

Table 5.9. Comparison of various rule-based classifiers.

	RIPPER	CN2 (unordered)	CN2 (ordered)	AQR
Rule-growing strategy	General-to-specific	General-to-specific	General-to-specific	General-to-specific (seeded by a positive example)
Evaluation Metric	FOIL's Info gain	Laplace	Entropy and likelihood ratio	Number of true positives
Stopping condition for rule-growing	All examples belong to the same class	No performance gain	No performance gain	Rules cover only positive class
Rule Pruning	Reduced error pruning	None	None	None
Instance Elimination	Positive and negative	Positive only	Positive only	Positive and negative
Stopping condition for adding rules	Error > 50% or based on MDL	No performance gain	No performance gain	All positive examples are covered
Rule Set Pruning	Replace or modify rules	Statistical tests	None	None
Search strategy	Greedy	Beam search	Beam search	Beam search

Cover and Hart [172] presented an overview of the nearest-neighbor classification method from a Bayesian perspective. Aha provided both theoretical and empirical evaluations for instance-based methods in [155]. PEBLS, which was developed by Cost and Salzberg [171], is a nearest-neighbor classification algorithm that can handle data sets containing nominal attributes. Each training example in PEBLS is also assigned a weight factor that depends on the number of times the example helps make a correct prediction. Han et al. [188] developed a weight-adjusted nearest-neighbor algorithm, in which the feature weights are learned using a greedy, hill-climbing optimization algorithm.

Naïve Bayes classifiers have been investigated by many authors, including Langley et al. [203], Ramoni and Sebastiani [212], Lewis [204], and Domingos and Pazzani [178]. Although the independence assumption used in naïve Bayes classifiers may seem rather unrealistic, the method has worked surprisingly well for applications such as text classification. Bayesian belief networks provide a more flexible approach by allowing some of the attributes to be interdependent. An excellent tutorial on Bayesian belief networks is given by Heckerman in [194].

Vapnik [217, 218] had written two authoritative books on Support Vector Machines (SVM). Other useful resources on SVM and kernel methods include the books by Cristianini and Shawe-Taylor [173] and Schölkopf and Smola

[213]. There are several survey articles on SVM, including those written by Burges [164], Bennet et al. [158], Hearst [193], and Mangasarian [205].

A survey of ensemble methods in machine learning was given by Dietterich [174]. The bagging method was proposed by Breiman [161]. Freund and Schapire [186] developed the AdaBoost algorithm. Arcing, which stands for adaptive resampling and combining, is a variant of the boosting algorithm proposed by Breiman [162]. It uses the non-uniform weights assigned to training examples to resample the data for building an ensemble of training sets. Unlike AdaBoost, the votes of the base classifiers are not weighted when determining the class label of test examples. The random forest method was introduced by Breiman in [163].

Related work on mining rare and imbalanced data sets can be found in the survey papers written by Chawla et al. [166] and Weiss [220]. Sampling-based methods for mining imbalanced data sets have been investigated by many authors, such as Kubat and Matwin [202], Japkowitz [196], and Drummond and Holte [179]. Joshi et al. [199] discussed the limitations of boosting algorithms for rare class modeling. Other algorithms developed for mining rare classes include SMOTE [165], PNrule [198], and CREDOS [200].

Various alternative metrics that are well-suited for class imbalanced problems are available. The precision, recall, and F_1 -measure are widely used metrics in information retrieval [216]. ROC analysis was originally used in signal detection theory. Bradley [160] investigated the use of area under the ROC curve as a performance metric for machine learning algorithms. A method for comparing classifier performance using the convex hull of ROC curves was suggested by Provost and Fawcett in [210]. Ferri et al. [185] developed a methodology for performing ROC analysis on decision tree classifiers. They had also proposed a methodology for incorporating area under the ROC curve (AUC) as the splitting criterion during the tree-growing process. Joshi [197] examined the performance of these measures from the perspective of analyzing rare classes.

A vast amount of literature on cost-sensitive learning can be found in the online proceedings of the ICML'2000 Workshop on cost-sensitive learning. The properties of a cost matrix had been studied by Elkan in [182]. Margineantu and Dietterich [206] examined various methods for incorporating cost information into the C4.5 learning algorithm, including wrapper methods, class distribution-based methods, and loss-based methods. Other cost-sensitive learning methods that are algorithm-independent include AdaCost [183], MetaCost [177], and costing [222].

Extensive literature is also available on the subject of multiclass learning. This includes the works of Hastie and Tibshirani [191], Allwein et al. [156], Kong and Dietterich [201], and Tax and Duin [215]. The error-correcting output coding (ECOC) method was proposed by Dietterich and Bakiri [175]. They had also investigated techniques for designing codes that are suitable for solving multiclass problems.

Bibliography

- [155] D. W. Aha. *A study of instance-based algorithms for supervised learning tasks: mathematical, empirical, and psychological evaluations*. PhD thesis, University of California, Irvine, 1990.
- [156] E. L. Allwein, R. E. Schapire, and Y. Singer. Reducing Multiclass to Binary: A Unifying Approach to Margin Classifiers. *Journal of Machine Learning Research*, 1: 113–141, 2000.
- [157] R. Andrews, J. Diederich, and A. Tickle. A Survey and Critique of Techniques For Extracting Rules From Trained Artificial Neural Networks. *Knowledge Based Systems*, 8(6):373–389, 1995.
- [158] K. Bennett and C. Campbell. Support Vector Machines: Hype or Hallelujah. *SIGKDD Explorations*, 2(2):1–13, 2000.
- [159] C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford, U.K., 1995.
- [160] A. P. Bradley. The use of the area under the ROC curve in the Evaluation of Machine Learning Algorithms. *Pattern Recognition*, 30(7):1145–1149, 1997.
- [161] L. Breiman. Bagging Predictors. *Machine Learning*, 24(2):123–140, 1996.
- [162] L. Breiman. Bias, Variance, and Arcing Classifiers. Technical Report 486, University of California, Berkeley, CA, 1996.
- [163] L. Breiman. Random Forests. *Machine Learning*, 45(1):5–32, 2001.
- [164] C. J. C. Burges. A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.
- [165] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, 16:321–357, 2002.
- [166] N. V. Chawla, N. Japkowicz, and A. Kolcz. Editorial: Special Issue on Learning from Imbalanced Data Sets. *SIGKDD Explorations*, 6(1):1–6, 2004.
- [167] V. Cherkassky and F. Mulier. *Learning from Data: Concepts, Theory, and Methods*. Wiley Interscience, 1998.
- [168] P. Clark and R. Boswell. Rule Induction with CN2: Some Recent Improvements. In *Machine Learning: Proc. of the 5th European Conf. (EWSL-91)*, pages 151–163, 1991.
- [169] P. Clark and T. Niblett. The CN2 Induction Algorithm. *Machine Learning*, 3(4): 261–283, 1989.
- [170] W. W. Cohen. Fast Effective Rule Induction. In *Proc. of the 12th Intl. Conf. on Machine Learning*, pages 115–123, Tahoe City, CA, July 1995.
- [171] S. Cost and S. Salzberg. A Weighted Nearest Neighbor Algorithm for Learning with Symbolic Features. *Machine Learning*, 10:57–78, 1993.
- [172] T. M. Cover and P. E. Hart. Nearest Neighbor Pattern Classification. *Knowledge Based Systems*, 8(6):373–389, 1995.

- [173] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 2000.
- [174] T. G. Dietterich. Ensemble Methods in Machine Learning. In *First Intl. Workshop on Multiple Classifier Systems*, Cagliari, Italy, 2000.
- [175] T. G. Dietterich and G. Bakiri. Solving Multiclass Learning Problems via Error-Correcting Output Codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995.
- [176] P. Domingos. The RISE system: Conquering without separating. In *Proc. of the 6th IEEE Intl. Conf. on Tools with Artificial Intelligence*, pages 704–707, New Orleans, LA, 1994.
- [177] P. Domingos. MetaCost: A General Method for Making Classifiers Cost-Sensitive. In *Proc. of the 5th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 155–164, San Diego, CA, August 1999.
- [178] P. Domingos and M. Pazzani. On the Optimality of the Simple Bayesian Classifier under Zero-One Loss. *Machine Learning*, 29(2-3):103–130, 1997.
- [179] C. Drummond and R. C. Holte. C4.5, Class imbalance, and Cost sensitivity: Why under-sampling beats over-sampling. In *ICML'2004 Workshop on Learning from Imbalanced Data Sets II*, Washington, DC, August 2003.
- [180] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. John Wiley & Sons, Inc., New York, 2nd edition, 2001.
- [181] M. H. Dunham. *Data Mining: Introductory and Advanced Topics*. Prentice Hall, 2002.
- [182] C. Elkan. The Foundations of Cost-Sensitive Learning. In *Proc. of the 17th Intl. Joint Conf. on Artificial Intelligence*, pages 973–978, Seattle, WA, August 2001.
- [183] W. Fan, S. J. Stolfo, J. Zhang, and P. K. Chan. AdaCost: misclassification cost-sensitive boosting. In *Proc. of the 16th Intl. Conf. on Machine Learning*, pages 97–105, Bled, Slovenia, June 1999.
- [184] J. Fürnkranz and G. Widmer. Incremental reduced error pruning. In *Proc. of the 11th Intl. Conf. on Machine Learning*, pages 70–77, New Brunswick, NJ, July 1994.
- [185] C. Ferri, P. Flach, and J. Hernandez-Orallo. Learning Decision Trees Using the Area Under the ROC Curve. In *Proc. of the 19th Intl. Conf. on Machine Learning*, pages 139–146, Sydney, Australia, July 2002.
- [186] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.
- [187] K. Fukunaga. *Introduction to Statistical Pattern Recognition*. Academic Press, New York, 1990.
- [188] E.-H. Han, G. Karypis, and V. Kumar. Text Categorization Using Weight Adjusted k-Nearest Neighbor Classification. In *Proc. of the 5th Pacific-Asia Conf. on Knowledge Discovery and Data Mining*, Lyon, France, 2001.
- [189] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, San Francisco, 2001.
- [190] D. J. Hand, H. Mannila, and P. Smyth. *Principles of Data Mining*. MIT Press, 2001.
- [191] T. Hastie and R. Tibshirani. Classification by pairwise coupling. *Annals of Statistics*, 26(2):451–471, 1998.
- [192] T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, Prediction*. Springer, New York, 2001.
- [193] M. Hearst. Trends & Controversies: Support Vector Machines. *IEEE Intelligent Systems*, 13(4):18–28, 1998.

- [194] D. Heckerman. Bayesian Networks for Data Mining. *Data Mining and Knowledge Discovery*, 1(1):79–119, 1997.
- [195] R. C. Holte. Very Simple Classification Rules Perform Well on Most Commonly Used Data sets. *Machine Learning*, 11:63–91, 1993.
- [196] N. Japkowicz. The Class Imbalance Problem: Significance and Strategies. In *Proc. of the 2000 Intl. Conf. on Artificial Intelligence: Special Track on Inductive Learning*, volume 1, pages 111–117, Las Vegas, NV, June 2000.
- [197] M. V. Joshi. On Evaluating Performance of Classifiers for Rare Classes. In *Proc. of the 2002 IEEE Intl. Conf. on Data Mining*, Maebashi City, Japan, December 2002.
- [198] M. V. Joshi, R. C. Agarwal, and V. Kumar. Mining Needles in a Haystack: Classifying Rare Classes via Two-Phase Rule Induction. In *Proc. of 2001 ACM-SIGMOD Intl. Conf. on Management of Data*, pages 91–102, Santa Barbara, CA, June 2001.
- [199] M. V. Joshi, R. C. Agarwal, and V. Kumar. Predicting rare classes: can boosting make any weak learner strong? In *Proc. of the 8th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 297–306, Edmonton, Canada, July 2002.
- [200] M. V. Joshi and V. Kumar. CREDO: Classification Using Ripple Down Structure (A Case for Rare Classes). In *Proc. of the SIAM Intl. Conf. on Data Mining*, pages 321–332, Orlando, FL, April 2004.
- [201] E. B. Kong and T. G. Dietterich. Error-Correcting Output Coding Corrects Bias and Variance. In *Proc. of the 12th Intl. Conf. on Machine Learning*, pages 313–321, Tahoe City, CA, July 1995.
- [202] M. Kubat and S. Matwin. Addressing the Curse of Imbalanced Training Sets: One Sided Selection. In *Proc. of the 14th Intl. Conf. on Machine Learning*, pages 179–186, Nashville, TN, July 1997.
- [203] P. Langley, W. Iba, and K. Thompson. An analysis of Bayesian classifiers. In *Proc. of the 10th National Conf. on Artificial Intelligence*, pages 223–228, 1992.
- [204] D. D. Lewis. Naive Bayes at Forty: The Independence Assumption in Information Retrieval. In *Proc. of the 10th European Conf. on Machine Learning (ECML 1998)*, pages 4–15, 1998.
- [205] O. Mangasarian. Data Mining via Support Vector Machines. Technical Report Technical Report 01-05, Data Mining Institute, May 2001.
- [206] D. D. Margineantu and T. G. Dietterich. Learning Decision Trees for Loss Minimization in Multi-Class Problems. Technical Report 99-30-03, Oregon State University, 1999.
- [207] R. S. Michalski, I. Mozetic, J. Hong, and N. Lavrac. The Multi-Purpose Incremental Learning System AQ15 and Its Testing Application to Three Medical Domains. In *Proc. of 5th National Conf. on Artificial Intelligence*, Orlando, August 1986.
- [208] T. Mitchell. *Machine Learning*. McGraw-Hill, Boston, MA, 1997.
- [209] S. Muggleton. *Foundations of Inductive Logic Programming*. Prentice Hall, Englewood Cliffs, NJ, 1995.
- [210] F. J. Provost and T. Fawcett. Analysis and Visualization of Classifier Performance: Comparison under Imprecise Class and Cost Distributions. In *Proc. of the 3rd Intl. Conf. on Knowledge Discovery and Data Mining*, pages 43–48, Newport Beach, CA, August 1997.
- [211] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan-Kaufmann Publishers, San Mateo, CA, 1993.
- [212] M. Ramoni and P. Sebastiani. Robust Bayes classifiers. *Artificial Intelligence*, 125: 209–226, 2001.

- [213] B. Schölkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2001.
- [214] P. Smyth and R. M. Goodman. An Information Theoretic Approach to Rule Induction from Databases. *IEEE Trans. on Knowledge and Data Engineering*, 4(4):301–316, 1992.
- [215] D. M. J. Tax and R. P. W. Duin. Using Two-Class Classifiers for Multiclass Classification. In *Proc. of the 16th Intl. Conf. on Pattern Recognition (ICPR 2002)*, pages 124–127, Quebec, Canada, August 2002.
- [216] C. J. van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, Newton, MA, 1978.
- [217] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer Verlag, New York, 1995.
- [218] V. Vapnik. *Statistical Learning Theory*. John Wiley & Sons, New York, 1998.
- [219] A. R. Webb. *Statistical Pattern Recognition*. John Wiley & Sons, 2nd edition, 2002.
- [220] G. M. Weiss. Mining with Rarity: A Unifying Framework. *SIGKDD Explorations*, 6(1):7–19, 2004.
- [221] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 1999.
- [222] B. Zadrozny, J. C. Langford, and N. Abe. Cost-Sensitive Learning by Cost-Proportionate Example Weighting. In *Proc. of the 2003 IEEE Intl. Conf. on Data Mining*, pages 435–442, Melbourne, FL, August 2003.

5.10 Exercises

- Consider a binary classification problem with the following set of attributes and attribute values:
 - Air Conditioner = {Working, Broken}
 - Engine = {Good, Bad}
 - Mileage = {High, Medium, Low}
 - Rust = {Yes, No}

Suppose a rule-based classifier produces the following rule set:

Mileage = High —> Value = Low
Mileage = Low —> Value = High
Air Conditioner = Working, Engine = Good —> Value = High
Air Conditioner = Working, Engine = Bad —> Value = Low
Air Conditioner = Broken —> Value = Low

- Are the rules mutually exclusive?

- (b) Is the rule set exhaustive?
 - (c) Is ordering needed for this set of rules?
 - (d) Do you need a default class for the rule set?
2. The RIPPER algorithm (by Cohen [170]) is an extension of an earlier algorithm called IREP (by Fürnkranz and Widmer [184]). Both algorithms apply the **reduced-error pruning** method to determine whether a rule needs to be pruned. The reduced error pruning method uses a validation set to estimate the generalization error of a classifier. Consider the following pair of rules:

$$\begin{aligned} R_1: \quad & A \longrightarrow C \\ R_2: \quad & A \wedge B \longrightarrow C \end{aligned}$$

R_2 is obtained by adding a new conjunct, B , to the left-hand side of R_1 . For this question, you will be asked to determine whether R_2 is preferred over R_1 from the perspectives of rule-growing and rule-pruning. To determine whether a rule should be pruned, IREP computes the following measure:

$$v_{IREP} = \frac{p + (N - n)}{P + N},$$

where P is the total number of positive examples in the validation set, N is the total number of negative examples in the validation set, p is the number of positive examples in the validation set covered by the rule, and n is the number of negative examples in the validation set covered by the rule. v_{IREP} is actually similar to classification accuracy for the validation set. IREP favors rules that have higher values of v_{IREP} . On the other hand, RIPPER applies the following measure to determine whether a rule should be pruned:

$$v_{RIPPER} = \frac{p - n}{p + n}.$$

- (a) Suppose R_1 is covered by 350 positive examples and 150 negative examples, while R_2 is covered by 300 positive examples and 50 negative examples. Compute the FOIL's information gain for the rule R_2 with respect to R_1 .
- (b) Consider a validation set that contains 500 positive examples and 500 negative examples. For R_1 , suppose the number of positive examples covered by the rule is 200, and the number of negative examples covered by the rule is 50. For R_2 , suppose the number of positive examples covered by the rule is 100 and the number of negative examples is 5. Compute v_{IREP} for both rules. Which rule does IREP prefer?
- (c) Compute v_{RIPPER} for the previous problem. Which rule does RIPPER prefer?

3. C4.5rules is an implementation of an indirect method for generating rules from a decision tree. RIPPER is an implementation of a direct method for generating rules directly from data.
 - (a) Discuss the strengths and weaknesses of both methods.
 - (b) Consider a data set that has a large difference in the class size (i.e., some classes are much bigger than others). Which method (between C4.5rules and RIPPER) is better in terms of finding high accuracy rules for the small classes?
4. Consider a training set that contains 100 positive examples and 400 negative examples. For each of the following candidate rules,

$R_1: A \longrightarrow +$ (covers 4 positive and 1 negative examples),

$R_2: B \longrightarrow +$ (covers 30 positive and 10 negative examples),

$R_3: C \longrightarrow +$ (covers 100 positive and 90 negative examples),

determine which is the best and worst candidate rule according to:

- (a) Rule accuracy.
- (b) FOIL's information gain.
- (c) The likelihood ratio statistic.
- (d) The Laplace measure.
- (e) The m-estimate measure (with $k = 2$ and $p_+ = 0.2$).
5. Figure 5.4 illustrates the coverage of the classification rules R_1 , R_2 , and R_3 . Determine which is the best and worst rule according to:
 - (a) The likelihood ratio statistic.
 - (b) The Laplace measure.
 - (c) The m-estimate measure (with $k = 2$ and $p_+ = 0.58$).
 - (d) The rule accuracy after R_1 has been discovered, where none of the examples covered by R_1 are discarded.
 - (e) The rule accuracy after R_1 has been discovered, where only the positive examples covered by R_1 are discarded.
 - (f) The rule accuracy after R_1 has been discovered, where both positive and negative examples covered by R_1 are discarded.
6. (a) Suppose the fraction of undergraduate students who smoke is 15% and the fraction of graduate students who smoke is 23%. If one-fifth of the college students are graduate students and the rest are undergraduates, what is the probability that a student who smokes is a graduate student?

- (b) Given the information in part (a), is a randomly chosen college student more likely to be a graduate or undergraduate student?
- (c) Repeat part (b) assuming that the student is a smoker.
- (d) Suppose 30% of the graduate students live in a dorm but only 10% of the undergraduate students live in a dorm. If a student smokes and lives in the dorm, is he or she more likely to be a graduate or undergraduate student? You can assume independence between students who live in a dorm and those who smoke.
7. Consider the data set shown in Table 5.10

Table 5.10. Data set for Exercise 7.

Record	A	B	C	Class
1	0	0	0	+
2	0	0	1	-
3	0	1	1	-
4	0	1	1	-
5	0	0	1	+
6	1	0	1	+
7	1	0	1	-
8	1	0	1	-
9	1	1	1	+
10	1	0	1	+

- (a) Estimate the conditional probabilities for $P(A|+)$, $P(B|+)$, $P(C|+)$, $P(A|-)$, $P(B|-)$, and $P(C|-)$.
- (b) Use the estimate of conditional probabilities given in the previous question to predict the class label for a test sample ($A = 0, B = 1, C = 0$) using the naïve Bayes approach.
- (c) Estimate the conditional probabilities using the m-estimate approach, with $p = 1/2$ and $m = 4$.
- (d) Repeat part (b) using the conditional probabilities given in part (c).
- (e) Compare the two methods for estimating probabilities. Which method is better and why?

8. Consider the data set shown in Table 5.11.

- (a) Estimate the conditional probabilities for $P(A = 1|+)$, $P(B = 1|+)$, $P(C = 1|+)$, $P(A = 1|-)$, $P(B = 1|-)$, and $P(C = 1|-)$ using the same approach as in the previous problem.

Table 5.11. Data set for Exercise 8.

Instance	A	B	C	Class
1	0	0	1	–
2	1	0	1	+
3	0	1	0	–
4	1	0	0	–
5	1	0	1	+
6	0	0	1	+
7	1	1	0	–
8	0	0	0	–
9	0	1	0	+
10	1	1	1	+

- (b) Use the conditional probabilities in part (a) to predict the class label for a test sample ($A = 1, B = 1, C = 1$) using the naïve Bayes approach.
 - (c) Compare $P(A = 1)$, $P(B = 1)$, and $P(A = 1, B = 1)$. State the relationships between A and B .
 - (d) Repeat the analysis in part (c) using $P(A = 1)$, $P(B = 0)$, and $P(A = 1, B = 0)$.
 - (e) Compare $P(A = 1, B = 1|Class = +)$ against $P(A = 1|Class = +)$ and $P(B = 1|Class = +)$. Are the variables conditionally independent given the class?
9. (a) Explain how naïve Bayes performs on the data set shown in Figure 5.46.
(b) If each class is further divided such that there are four classes (A_1 , A_2 , B_1 , and B_2), will naïve Bayes perform better?
(c) How will a decision tree perform on this data set (for the two-class problem)? What if there are four classes?
10. Repeat the analysis shown in Example 5.3 for finding the location of a decision boundary using the following information:
- (a) The prior probabilities are $P(\text{Crocodile}) = 2 \times P(\text{Alligator})$.
 - (b) The prior probabilities are $P(\text{Alligator}) = 2 \times P(\text{Crocodile})$.
 - (c) The prior probabilities are the same, but their standard deviations are different; i.e., $\sigma(\text{Crocodile}) = 4$ and $\sigma(\text{Alligator}) = 2$.
11. Figure 5.47 illustrates the Bayesian belief network for the data set shown in Table 5.12. (Assume that all the attributes are binary).
- (a) Draw the probability table for each node in the network.

		Attributes		
		Distinguishing Attributes	Noise Attributes	
Records	A1			
	A2			
Records	B1			Class A
	B2			Class B

Figure 5.46. Data set for Exercise 9.

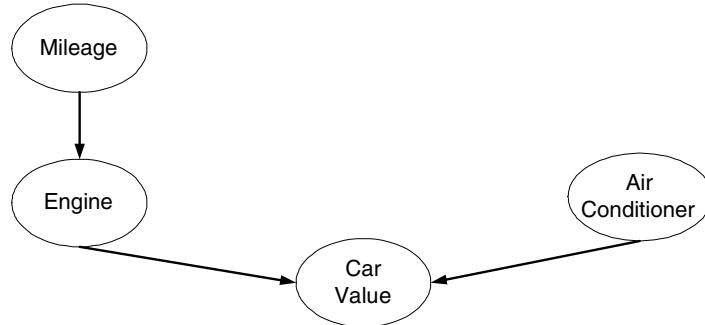


Figure 5.47. Bayesian belief network.

- (b) Use the Bayesian network to compute $P(\text{Engine} = \text{Bad}, \text{Air Conditioner} = \text{Broken})$.
12. Given the Bayesian network shown in Figure 5.48, compute the following probabilities:
- (a) $P(B = \text{good}, F = \text{empty}, G = \text{empty}, S = \text{yes})$.
 - (b) $P(B = \text{bad}, F = \text{empty}, G = \text{not empty}, S = \text{no})$.
 - (c) Given that the battery is bad, compute the probability that the car will start.
13. Consider the one-dimensional data set shown in Table 5.13.

Table 5.12. Data set for Exercise 11.

Mileage	Engine	Air Conditioner	Number of Records with Car Value=Hi	Number of Records with Car Value=Lo
Hi	Good	Working	3	4
Hi	Good	Broken	1	2
Hi	Bad	Working	1	5
Hi	Bad	Broken	0	4
Lo	Good	Working	9	0
Lo	Good	Broken	5	1
Lo	Bad	Working	1	2
Lo	Bad	Broken	0	2

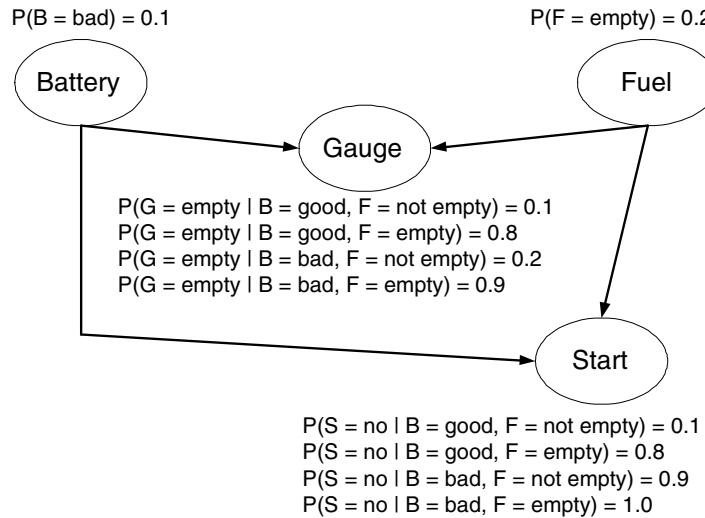


Figure 5.48. Bayesian belief network for Exercise 12.

- (a) Classify the data point $x = 5.0$ according to its 1-, 3-, 5-, and 9-nearest neighbors (using majority vote).
 - (b) Repeat the previous analysis using the distance-weighted voting approach described in Section 5.2.1.
14. The nearest-neighbor algorithm described in Section 5.2 can be extended to handle nominal attributes. A variant of the algorithm called PEBLS (Parallel Examplar-Based Learning System) by Cost and Salzberg [171] measures the distance between two values of a nominal attribute using the modified value difference metric (MVDM). Given a pair of nominal attribute values, V_1 and