

INTRODUCTION TO DATA MINING

PANG-NING TAN

Michigan State University

MICHAEL STEINBACH

University of Minnesota

VIPIN KUMAR

University of Minnesota
and Army High Performance
Computing Research Center



Boston San Francisco New York
London Toronto Sydney Tokyo Singapore Madrid
Mexico City Munich Paris Cape Town Hong Kong Montreal

Contents

Preface	vii
1 Introduction	1
1.1 What Is Data Mining?	2
1.2 Motivating Challenges	4
1.3 The Origins of Data Mining	6
1.4 Data Mining Tasks	7
1.5 Scope and Organization of the Book	11
1.6 Bibliographic Notes	13
1.7 Exercises	16
2 Data	19
2.1 Types of Data	22
2.1.1 Attributes and Measurement	23
2.1.2 Types of Data Sets	29
2.2 Data Quality	36
2.2.1 Measurement and Data Collection Issues	37
2.2.2 Issues Related to Applications	43
2.3 Data Preprocessing	44
2.3.1 Aggregation	45
2.3.2 Sampling	47
2.3.3 Dimensionality Reduction	50
2.3.4 Feature Subset Selection	52
2.3.5 Feature Creation	55
2.3.6 Discretization and Binarization	57
2.3.7 Variable Transformation	63
2.4 Measures of Similarity and Dissimilarity	65
2.4.1 Basics	66
2.4.2 Similarity and Dissimilarity between Simple Attributes	67
2.4.3 Dissimilarities between Data Objects	69
2.4.4 Similarities between Data Objects	72

xiv Contents

2.4.5 Examples of Proximity Measures	73
2.4.6 Issues in Proximity Calculation	80
2.4.7 Selecting the Right Proximity Measure	83
2.5 Bibliographic Notes	84
2.6 Exercises	88
3 Exploring Data	97
3.1 The Iris Data Set	98
3.2 Summary Statistics	98
3.2.1 Frequencies and the Mode	99
3.2.2 Percentiles	100
3.2.3 Measures of Location: Mean and Median	101
3.2.4 Measures of Spread: Range and Variance	102
3.2.5 Multivariate Summary Statistics	104
3.2.6 Other Ways to Summarize the Data	105
3.3 Visualization	105
3.3.1 Motivations for Visualization	105
3.3.2 General Concepts	106
3.3.3 Techniques	110
3.3.4 Visualizing Higher-Dimensional Data	124
3.3.5 Do's and Don'ts	130
3.4 OLAP and Multidimensional Data Analysis	131
3.4.1 Representing Iris Data as a Multidimensional Array	131
3.4.2 Multidimensional Data: The General Case	133
3.4.3 Analyzing Multidimensional Data	135
3.4.4 Final Comments on Multidimensional Data Analysis	139
3.5 Bibliographic Notes	139
3.6 Exercises	141
4 Classification:	
Basic Concepts, Decision Trees, and Model Evaluation	145
4.1 Preliminaries	146
4.2 General Approach to Solving a Classification Problem	148
4.3 Decision Tree Induction	150
4.3.1 How a Decision Tree Works	150
4.3.2 How to Build a Decision Tree	151
4.3.3 Methods for Expressing Attribute Test Conditions	155
4.3.4 Measures for Selecting the Best Split	158
4.3.5 Algorithm for Decision Tree Induction	164
4.3.6 An Example: Web Robot Detection	166

Contents xv

4.3.7 Characteristics of Decision Tree Induction	168
4.4 Model Overfitting	172
4.4.1 Overfitting Due to Presence of Noise	175
4.4.2 Overfitting Due to Lack of Representative Samples	177
4.4.3 Overfitting and the Multiple Comparison Procedure	178
4.4.4 Estimation of Generalization Errors	179
4.4.5 Handling Overfitting in Decision Tree Induction	184
4.5 Evaluating the Performance of a Classifier	186
4.5.1 Holdout Method	186
4.5.2 Random Subsampling	187
4.5.3 Cross-Validation	187
4.5.4 Bootstrap	188
4.6 Methods for Comparing Classifiers	188
4.6.1 Estimating a Confidence Interval for Accuracy	189
4.6.2 Comparing the Performance of Two Models	191
4.6.3 Comparing the Performance of Two Classifiers	192
4.7 Bibliographic Notes	193
4.8 Exercises	198
5 Classification: Alternative Techniques	207
5.1 Rule-Based Classifier	207
5.1.1 How a Rule-Based Classifier Works	209
5.1.2 Rule-Ordering Schemes	211
5.1.3 How to Build a Rule-Based Classifier	212
5.1.4 Direct Methods for Rule Extraction	213
5.1.5 Indirect Methods for Rule Extraction	221
5.1.6 Characteristics of Rule-Based Classifiers	223
5.2 Nearest-Neighbor classifiers	223
5.2.1 Algorithm	225
5.2.2 Characteristics of Nearest-Neighbor Classifiers	226
5.3 Bayesian Classifiers	227
5.3.1 Bayes Theorem	228
5.3.2 Using the Bayes Theorem for Classification	229
5.3.3 Naïve Bayes Classifier	231
5.3.4 Bayes Error Rate	238
5.3.5 Bayesian Belief Networks	240
5.4 Artificial Neural Network (ANN)	246
5.4.1 Perceptron	247
5.4.2 Multilayer Artificial Neural Network	251
5.4.3 Characteristics of ANN	255

xvi **Contents**

5.5 Support Vector Machine (SVM)	256
5.5.1 Maximum Margin Hyperplanes	256
5.5.2 Linear SVM: Separable Case	259
5.5.3 Linear SVM: Nonseparable Case	266
5.5.4 Nonlinear SVM	270
5.5.5 Characteristics of SVM	276
5.6 Ensemble Methods	276
5.6.1 Rationale for Ensemble Method	277
5.6.2 Methods for Constructing an Ensemble Classifier	278
5.6.3 Bias-Variance Decomposition	281
5.6.4 Bagging	283
5.6.5 Boosting	285
5.6.6 Random Forests	290
5.6.7 Empirical Comparison among Ensemble Methods	294
5.7 Class Imbalance Problem	294
5.7.1 Alternative Metrics	295
5.7.2 The Receiver Operating Characteristic Curve	298
5.7.3 Cost-Sensitive Learning	302
5.7.4 Sampling-Based Approaches	305
5.8 Multiclass Problem	306
5.9 Bibliographic Notes	309
5.10 Exercises	315
6 Association Analysis: Basic Concepts and Algorithms	327
6.1 Problem Definition	328
6.2 Frequent Itemset Generation	332
6.2.1 The <i>Apriori</i> Principle	333
6.2.2 Frequent Itemset Generation in the <i>Apriori</i> Algorithm .	335
6.2.3 Candidate Generation and Pruning	338
6.2.4 Support Counting	342
6.2.5 Computational Complexity	345
6.3 Rule Generation	349
6.3.1 Confidence-Based Pruning	350
6.3.2 Rule Generation in <i>Apriori</i> Algorithm	350
6.3.3 An Example: Congressional Voting Records	352
6.4 Compact Representation of Frequent Itemsets	353
6.4.1 Maximal Frequent Itemsets	354
6.4.2 Closed Frequent Itemsets	355
6.5 Alternative Methods for Generating Frequent Itemsets	359
6.6 FP-Growth Algorithm	363

Contents xvii

6.6.1	FP-Tree Representation	363
6.6.2	Frequent Itemset Generation in FP-Growth Algorithm	366
6.7	Evaluation of Association Patterns	370
6.7.1	Objective Measures of Interestingness	371
6.7.2	Measures beyond Pairs of Binary Variables	382
6.7.3	Simpson's Paradox	384
6.8	Effect of Skewed Support Distribution	386
6.9	Bibliographic Notes	390
6.10	Exercises	404
7	Association Analysis: Advanced Concepts	415
7.1	Handling Categorical Attributes	415
7.2	Handling Continuous Attributes	418
7.2.1	Discretization-Based Methods	418
7.2.2	Statistics-Based Methods	422
7.2.3	Non-discretization Methods	424
7.3	Handling a Concept Hierarchy	426
7.4	Sequential Patterns	429
7.4.1	Problem Formulation	429
7.4.2	Sequential Pattern Discovery	431
7.4.3	Timing Constraints	436
7.4.4	Alternative Counting Schemes	439
7.5	Subgraph Patterns	442
7.5.1	Graphs and Subgraphs	443
7.5.2	Frequent Subgraph Mining	444
7.5.3	<i>Apriori</i> -like Method	447
7.5.4	Candidate Generation	448
7.5.5	Candidate Pruning	453
7.5.6	Support Counting	457
7.6	Infrequent Patterns	457
7.6.1	Negative Patterns	458
7.6.2	Negatively Correlated Patterns	458
7.6.3	Comparisons among Infrequent Patterns, Negative Patterns, and Negatively Correlated Patterns	460
7.6.4	Techniques for Mining Interesting Infrequent Patterns	461
7.6.5	Techniques Based on Mining Negative Patterns	463
7.6.6	Techniques Based on Support Expectation	465
7.7	Bibliographic Notes	469
7.8	Exercises	473

xviii Contents

8	Cluster Analysis: Basic Concepts and Algorithms	487
8.1	Overview	490
8.1.1	What Is Cluster Analysis?	490
8.1.2	Different Types of Clusterings	491
8.1.3	Different Types of Clusters	493
8.2	K-means	496
8.2.1	The Basic K-means Algorithm	497
8.2.2	K-means: Additional Issues	506
8.2.3	Bisecting K-means	508
8.2.4	K-means and Different Types of Clusters	510
8.2.5	Strengths and Weaknesses	510
8.2.6	K-means as an Optimization Problem	513
8.3	Agglomerative Hierarchical Clustering	515
8.3.1	Basic Agglomerative Hierarchical Clustering Algorithm .	516
8.3.2	Specific Techniques	518
8.3.3	The Lance-Williams Formula for Cluster Proximity .	524
8.3.4	Key Issues in Hierarchical Clustering	524
8.3.5	Strengths and Weaknesses	526
8.4	DBSCAN	526
8.4.1	Traditional Density: Center-Based Approach	527
8.4.2	The DBSCAN Algorithm	528
8.4.3	Strengths and Weaknesses	530
8.5	Cluster Evaluation	532
8.5.1	Overview	533
8.5.2	Unsupervised Cluster Evaluation Using Cohesion and Separation	536
8.5.3	Unsupervised Cluster Evaluation Using the Proximity Matrix	542
8.5.4	Unsupervised Evaluation of Hierarchical Clustering . .	544
8.5.5	Determining the Correct Number of Clusters	546
8.5.6	Clustering Tendency	547
8.5.7	Supervised Measures of Cluster Validity	548
8.5.8	Assessing the Significance of Cluster Validity Measures .	553
8.6	Bibliographic Notes	555
8.7	Exercises	559
9	Cluster Analysis: Additional Issues and Algorithms	569
9.1	Characteristics of Data, Clusters, and Clustering Algorithms .	570
9.1.1	Example: Comparing K-means and DBSCAN	570
9.1.2	Data Characteristics	571

Contents **xix**

9.1.3 Cluster Characteristics	573
9.1.4 General Characteristics of Clustering Algorithms	575
9.2 Prototype-Based Clustering	577
9.2.1 Fuzzy Clustering	577
9.2.2 Clustering Using Mixture Models	583
9.2.3 Self-Organizing Maps (SOM)	594
9.3 Density-Based Clustering	600
9.3.1 Grid-Based Clustering	601
9.3.2 Subspace Clustering	604
9.3.3 DENCLUE: A Kernel-Based Scheme for Density-Based Clustering	608
9.4 Graph-Based Clustering	612
9.4.1 Sparsification	613
9.4.2 Minimum Spanning Tree (MST) Clustering	614
9.4.3 OPOSSUM: Optimal Partitioning of Sparse Similarities Using METIS	616
9.4.4 Chameleon: Hierarchical Clustering with Dynamic Modeling	616
9.4.5 Shared Nearest Neighbor Similarity	622
9.4.6 The Jarvis-Patrick Clustering Algorithm	625
9.4.7 SNN Density	627
9.4.8 SNN Density-Based Clustering	629
9.5 Scalable Clustering Algorithms	630
9.5.1 Scalability: General Issues and Approaches	630
9.5.2 BIRCH	633
9.5.3 CURE	635
9.6 Which Clustering Algorithm?	639
9.7 Bibliographic Notes	643
9.8 Exercises	647
10 Anomaly Detection	651
10.1 Preliminaries	653
10.1.1 Causes of Anomalies	653
10.1.2 Approaches to Anomaly Detection	654
10.1.3 The Use of Class Labels	655
10.1.4 Issues	656
10.2 Statistical Approaches	658
10.2.1 Detecting Outliers in a Univariate Normal Distribution	659
10.2.2 Outliers in a Multivariate Normal Distribution	661
10.2.3 A Mixture Model Approach for Anomaly Detection	662

xx **Contents**

10.2.4 Strengths and Weaknesses	665
10.3 Proximity-Based Outlier Detection	666
10.3.1 Strengths and Weaknesses	666
10.4 Density-Based Outlier Detection	668
10.4.1 Detection of Outliers Using Relative Density	669
10.4.2 Strengths and Weaknesses	670
10.5 Clustering-Based Techniques	671
10.5.1 Assessing the Extent to Which an Object Belongs to a Cluster	672
10.5.2 Impact of Outliers on the Initial Clustering	674
10.5.3 The Number of Clusters to Use	674
10.5.4 Strengths and Weaknesses	674
10.6 Bibliographic Notes	675
10.7 Exercises	680

Appendix A Linear Algebra **685**

A.1 Vectors	685
A.1.1 Definition	685
A.1.2 Vector Addition and Multiplication by a Scalar	685
A.1.3 Vector Spaces	687
A.1.4 The Dot Product, Orthogonality, and Orthogonal Projections	688
A.1.5 Vectors and Data Analysis	690
A.2 Matrices	691
A.2.1 Matrices: Definitions	691
A.2.2 Matrices: Addition and Multiplication by a Scalar	692
A.2.3 Matrices: Multiplication	693
A.2.4 Linear Transformations and Inverse Matrices	695
A.2.5 Eigenvalue and Singular Value Decomposition	697
A.2.6 Matrices and Data Analysis	699
A.3 Bibliographic Notes	700

Appendix B Dimensionality Reduction **701**

B.1 PCA and SVD	701
B.1.1 Principal Components Analysis (PCA)	701
B.1.2 SVD	706
B.2 Other Dimensionality Reduction Techniques	708
B.2.1 Factor Analysis	708
B.2.2 Locally Linear Embedding (LLE)	710
B.2.3 Multidimensional Scaling, FastMap, and ISOMAP	712

Contents xxi

B.2.4 Common Issues	715
B.3 Bibliographic Notes	716
Appendix C Probability and Statistics	719
C.1 Probability	719
C.1.1 Expected Values	722
C.2 Statistics	723
C.2.1 Point Estimation	724
C.2.2 Central Limit Theorem	724
C.2.3 Interval Estimation	725
C.3 Hypothesis Testing	726
Appendix D Regression	729
D.1 Preliminaries	729
D.2 Simple Linear Regression	730
D.2.1 Least Square Method	731
D.2.2 Analyzing Regression Errors	733
D.2.3 Analyzing Goodness of Fit	735
D.3 Multivariate Linear Regression	736
D.4 Alternative Least-Square Regression Methods	737
Appendix E Optimization	739
E.1 Unconstrained Optimization	739
E.1.1 Numerical Methods	742
E.2 Constrained Optimization	746
E.2.1 Equality Constraints	746
E.2.2 Inequality Constraints	747
Author Index	750
Subject Index	758
Copyright Permissions	769

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.

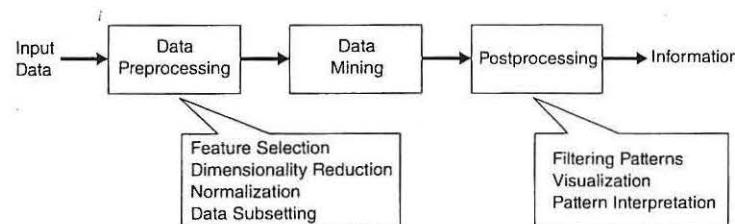


Figure 1.1. The process of knowledge discovery in databases (KDD).

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.

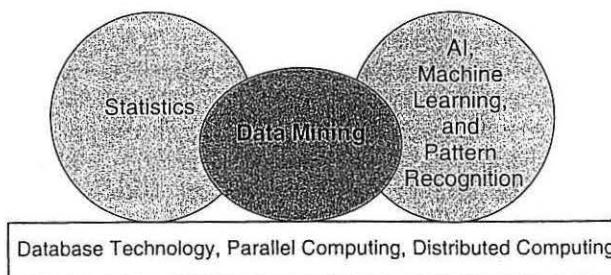


Figure 1.2. Data mining as a confluence of many disciplines.

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.

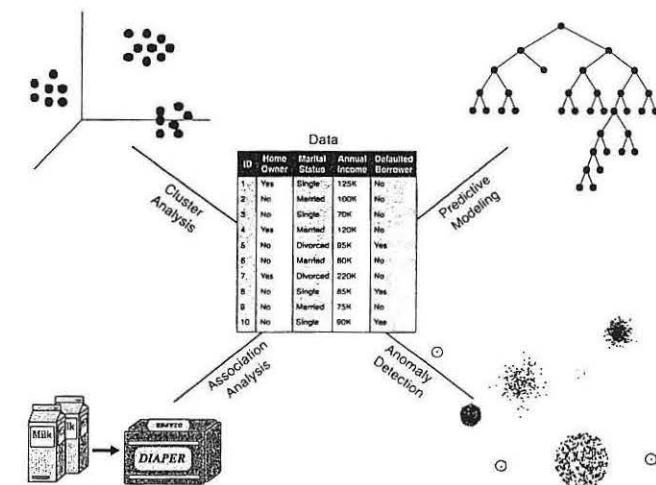


Figure 1.3. Four of the core data mining tasks.

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. ■

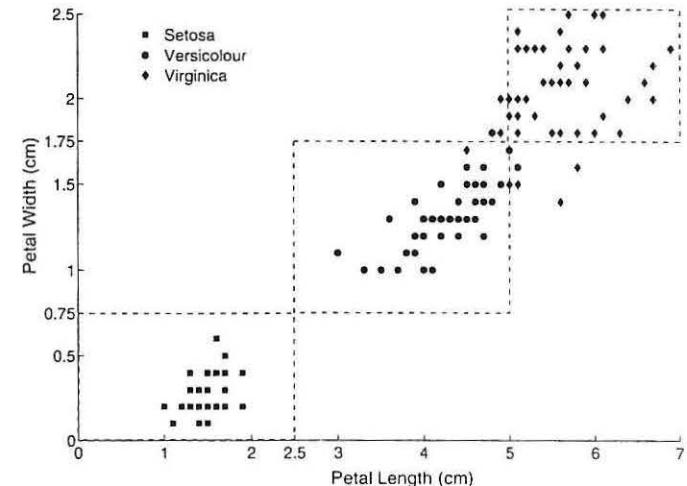


Figure 1.4. Petal width versus petal length for 150 Iris flowers.

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 classifi-

fers. 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.

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*.

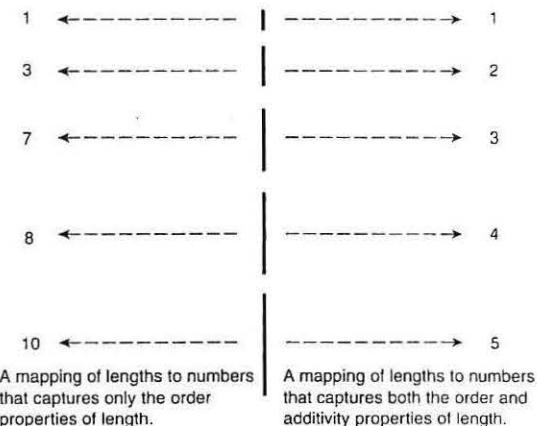


Figure 2.1. The measurement of the length of line segments on two different scales of measurement.

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	mode, entropy, contingency correlation, χ^2 test
	Ordinal	The values of an ordinal attribute provide enough information to order objects. ($<$, $>$)	hardness of minerals, {good, better, best}, grades, street numbers	median, percentiles, rank correlation, run tests, sign tests
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	mean, standard deviation, Pearson's correlation, t and F tests
	Ratio	For ratio variables, both differences and ratios are meaningful. (*, /)	temperature in Kelvin, monetary quantities, counts, age, mass, length, electrical current	geometric mean, harmonic mean, percent variation

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,

Tid	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.

TID	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.

Document	cat	coch	day	dl	ball	score	game	win	lost	imprint	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,

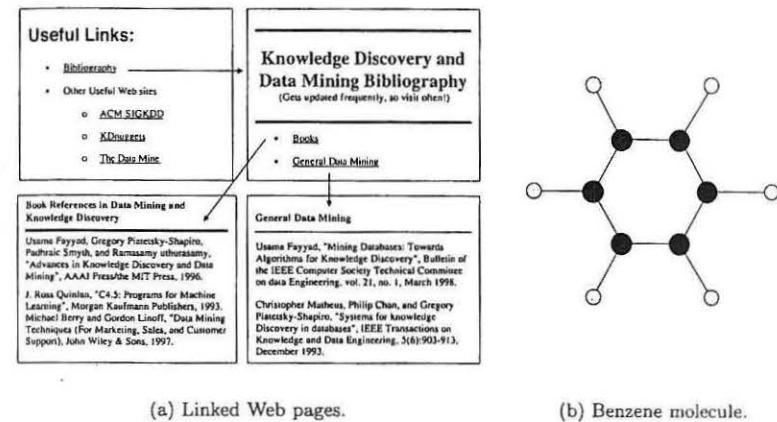
then a document can be represented as a term vector, where each term is a component (attribute) of the vector and the value of each component is the number of times the corresponding term occurs in the document. This representation of a collection of documents is often called a **document-term matrix**. 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,

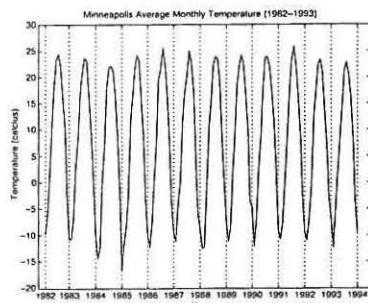
Time	Customer	Items Purchased
t1	C1	A, B
t2	C3	A, C
t3	C1	C, D
t4	C2	A, D
t5	C2	E
t6	C1	A, E

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)

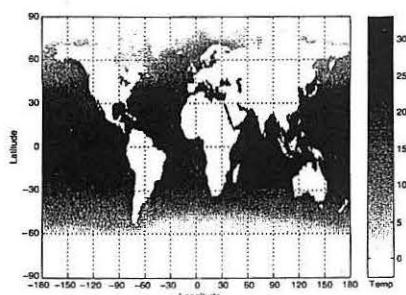
(a) Sequential transaction data.

GGTTCCGCCTTCAGCCCCGCGCC
CGCAGGGCCCAGCCCCGCGCCGTC
GAGAAGGGCCCGCCTGGCGGGCG
GGGGGAGGCAGGGCCGCCGAGC
CCAACCGAGTCCGACCAGGTGCC
CCCTCTGCTCGGCCTAGACCTGA
GCTCATTAGGCAGCGGACAG
GCCAAAGTAGAACACCGAAGCGC
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

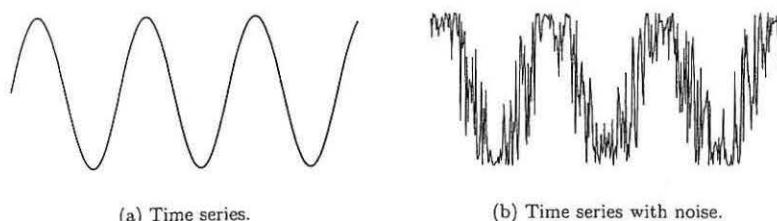


Figure 2.5. Noise in a time series context.

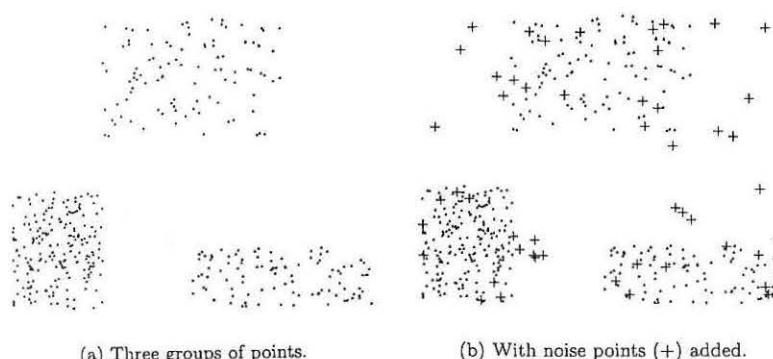


Figure 2.6. Noise in a spatial context.

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

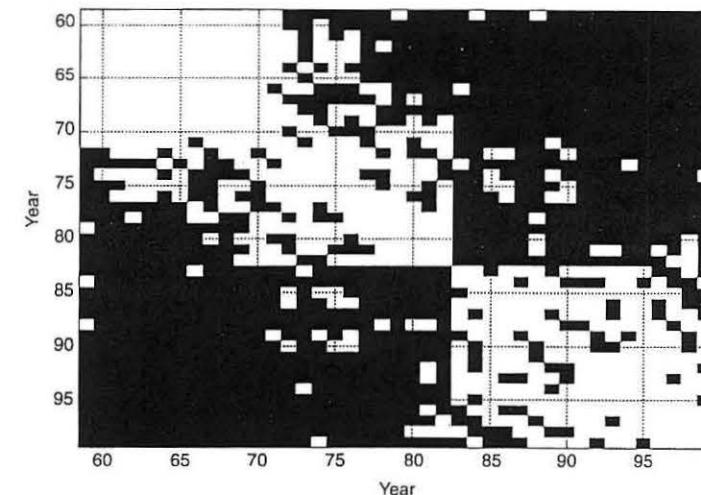


Figure 2.7. Correlation of SST data between pairs of years. White areas indicate positive correlation. Black areas indicate negative correlation.

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

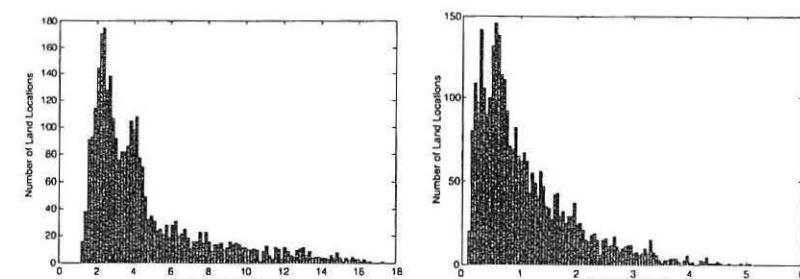
Table 2.4. Data set containing information about customer purchases.

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. ■

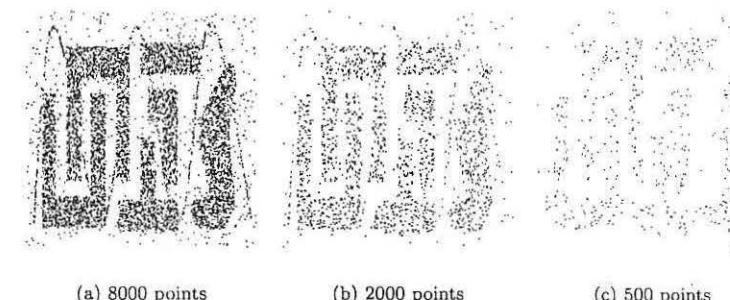


Figure 2.9. Example of the loss of structure with sampling.

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. ■

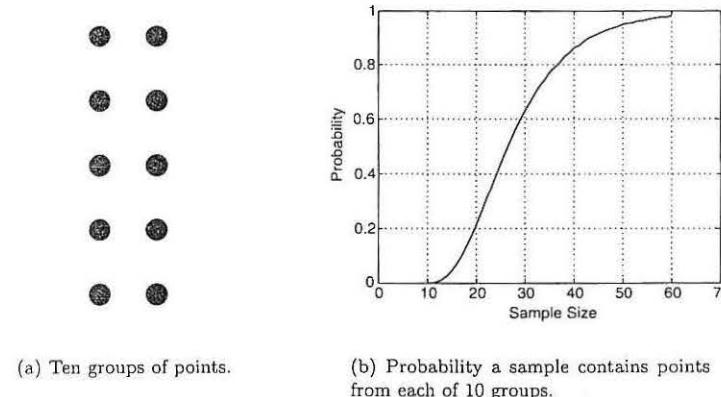


Figure 2.10. Finding representative points from 10 groups.

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

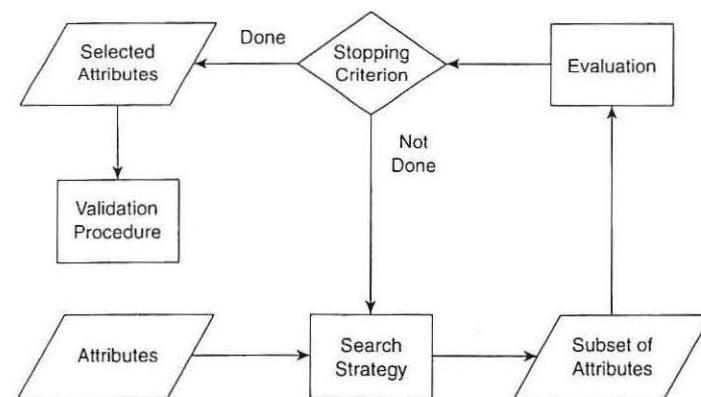


Figure 2.11. Flowchart of a feature subset selection process.

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.

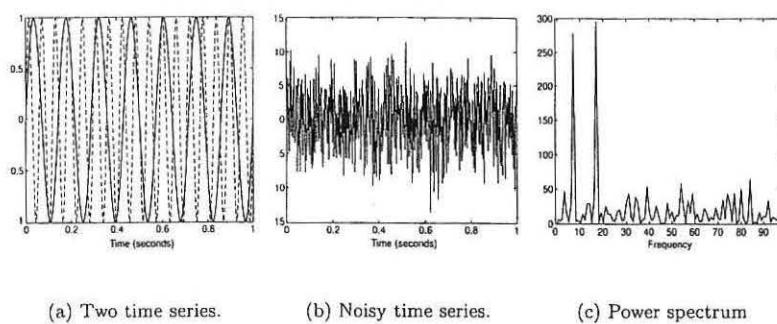


Figure 2.12. Application of the Fourier transform to identify the underlying frequencies in time series data.

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

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

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.,

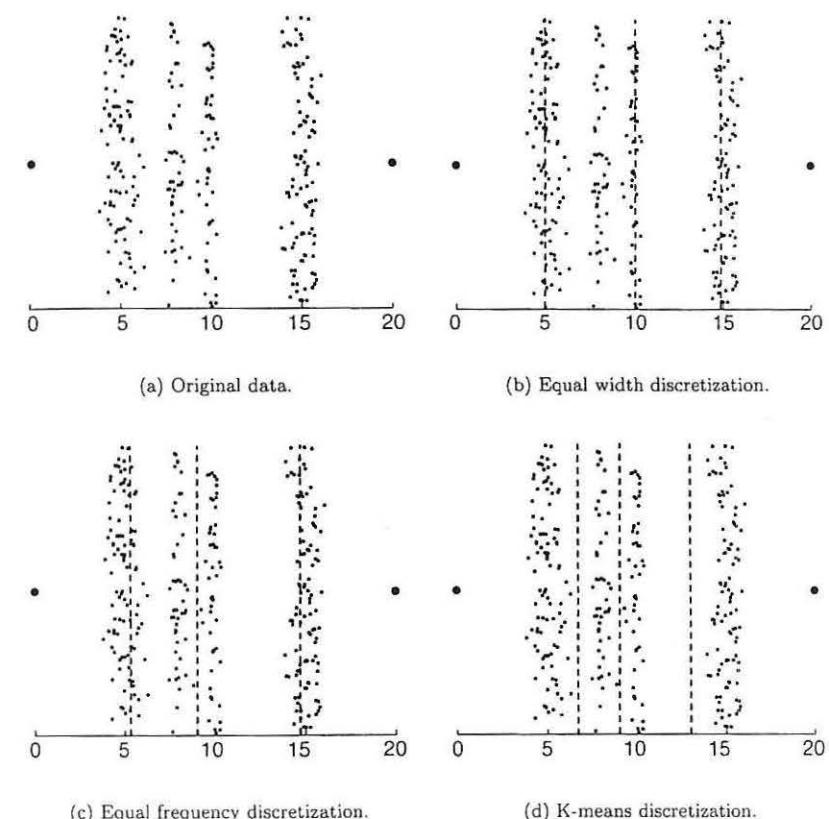


Figure 2.13. Different discretization techniques.

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

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

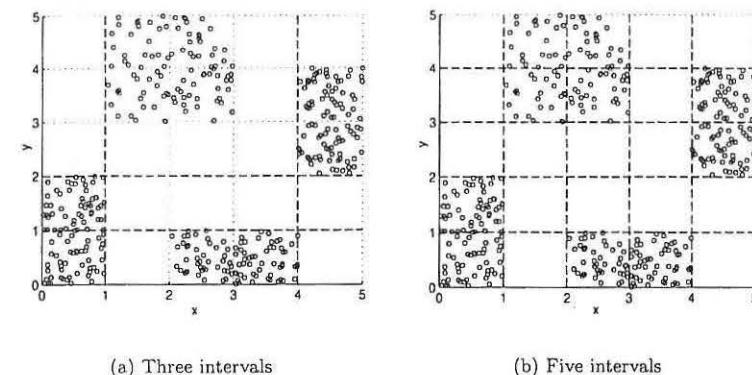


Figure 2.14. Discretizing x and y attributes for four groups (classes) of points.

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{x} , \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 $\{\frac{1}{2}, \frac{1}{3}\}$ 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 $s' = (s - \min_s)/(\max_s - \min_s)$, where \max_s and \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 $d' = (d - \min_d)/(\max_d - \min_d)$.

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, x and y , in one-, two-, three-, or higher-dimensional space, is given by the following familiar formula:

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

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

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

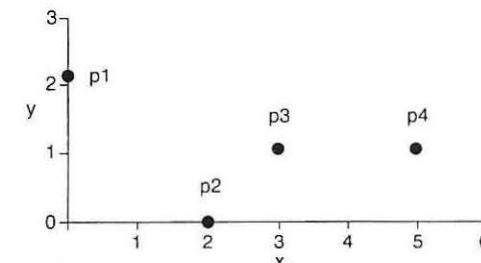


Figure 2.15. Four two-dimensional points.

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 x and 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.

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

- $f_{01} = 2$ the number of attributes where x was 0 and y was 1
- $f_{10} = 1$ the number of attributes where x was 1 and y was 0
- $f_{00} = 7$ the number of attributes where x was 0 and y was 0
- $f_{11} = 0$ the number of attributes where x was 1 and 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 x and y are two document vectors, then

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

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

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:

$$\begin{aligned} x &= (3, 2, 0, 5, 0, 0, 2, 0, 0) \\ y &= (1, 0, 0, 0, 0, 0, 1, 0, 2) \end{aligned}$$

$$\begin{aligned} x \cdot y &= 3 * 1 + 2 * 0 + 0 * 0 + 5 * 0 + 0 * 0 + 0 * 0 + 0 * 0 + 2 * 1 + 0 * 0 + 0 * 2 = 5 \\ \|x\| &= \sqrt{3 * 3 + 2 * 2 + 0 * 0 + 5 * 5 + 0 * 0 + 0 * 0 + 0 * 0 + 2 * 2 + 0 * 0 + 0 * 0} = 6.48 \\ \|y\| &= \sqrt{1 * 1 + 0 * 0 + 0 * 0 + 0 * 0 + 0 * 0 + 0 * 0 + 1 * 1 + 0 * 0 + 2 * 2} = 2.24 \\ \cos(x, y) &= 0.31 \end{aligned}$$

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

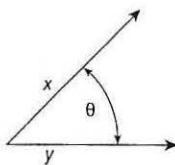


Figure 2.16. Geometric illustration of the cosine measure.

Equation 2.7 can be written as Equation 2.8.

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

where $x' = x/\|x\|$ and $y' = y/\|y\|$. Dividing x and 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:

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

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, x and y , is defined by the following equation:

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

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

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

$$\text{standard_deviation}(x) = s_x = \sqrt{\frac{1}{n-1} \sum_{k=1}^n (x_k - \bar{x})^2}$$

$$\text{standard_deviation}(y) = s_y = \sqrt{\frac{1}{n-1} \sum_{k=1}^n (y_k - \bar{y})^2}$$

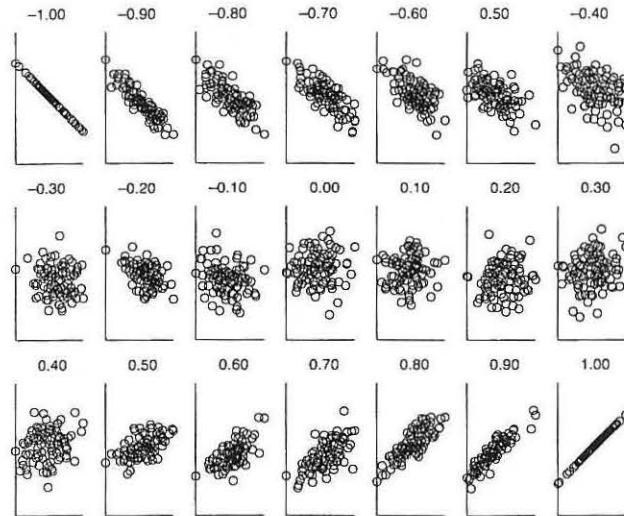
$$\bar{x} = \frac{1}{n} \sum_{k=1}^n x_k \text{ is the mean of } x$$

$$\bar{y} = \frac{1}{n} \sum_{k=1}^n y_k \text{ is the mean of } y$$

Example 2.19 (Perfect Correlation). Correlation is always in the range -1 to 1 . A correlation of 1 (-1) means that x and 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 x and y indicate cases where the correlation is -1 and $+1$, respectively. In the first case, the means of x and y were chosen to be 0 , for simplicity.

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

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

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 .

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

Example 2.21 (Visualizing Correlation). It is also easy to judge the correlation between two data objects x and y by plotting pairs of corresponding attribute values. Figure 2.17 shows a number of these plots when x and y have 30 attributes and the values of these attributes are randomly generated (with a normal distribution) so that the correlation of x and 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 x , while its y coordinate is the value of the same attribute for y .

If we transform x and 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 x and y be two points, where y is regarded as the original point and x is some distortion or approximation of it. For example, x may be a point that was generated, for example, by adding random noise to y . The goal is to measure the resulting distortion or loss that results if y is approximated by x . Of course, the more similar x and 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) $D(x, y)$ generated by that function is given by the following equation:

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

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

$D(x, y)$ can be written as $D(x, y) = \phi(x) - L(x)$, where $L(x) = \phi(y) + \langle \nabla \phi(y), (x - y) \rangle$ and represents the equation of a plane that is tangent to the function ϕ at y . Using calculus terminology, $L(x)$ is the linearization of ϕ around the point 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 $x = 3$. ■

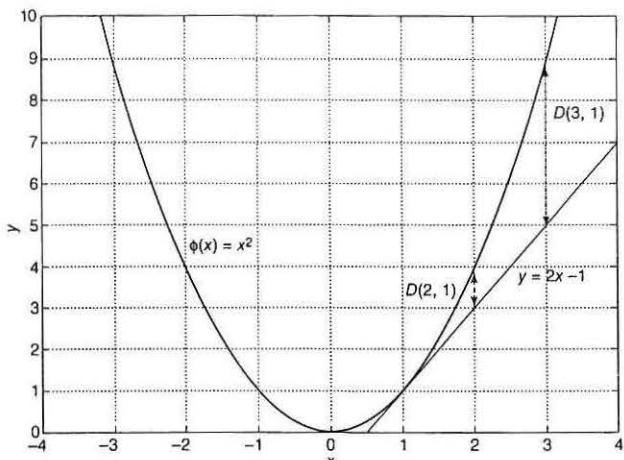


Figure 2.18. Illustration of Bregman divergence.

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.

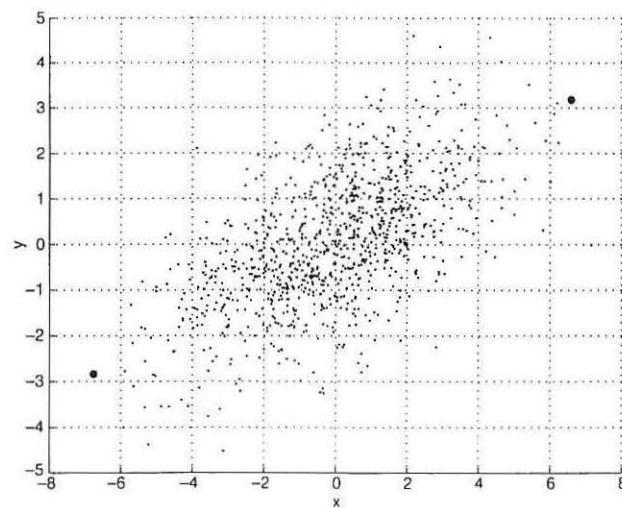


Figure 2.19. Set of two-dimensional points. The Mahalanobis distance between the two points represented by large dots is 6; their Euclidean distance is 14.7.

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, x and 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, $s_k(x, y)$, in the range [0, 1].
 - 2: Define an indicator variable, δ_k , for the k^{th} attribute as follows:
- $$\delta_k = \begin{cases} 0 & \text{if the } k^{\text{th}} \text{ attribute is an asymmetric attribute and} \\ & \text{both objects have a value of 0, or if one of the objects} \\ & \text{has a missing value for the } k^{\text{th}} \text{ attribute} \\ 1 & \text{otherwise} \end{cases}$$
- 3: Compute the overall similarity between the two objects using the following formula:

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

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

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

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

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

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 Int'l 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 Int'l 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 Int'l 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.

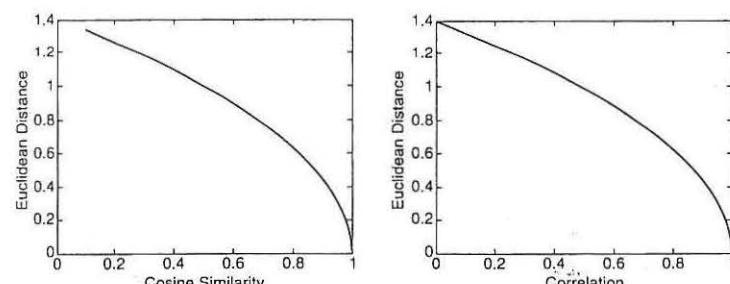
```

1: for  $i = 1$  to number of data objects do
2:   Find the distances of the  $i^{\text{th}}$  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.)
- We randomly select $n * m_i / m$ elements from each group.
 - 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.
- What is the effect of this transformation if a term occurs in one document? In every document?
 - 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 .
- What is the corresponding interval (a, b) in terms of x ?
 - Give an equation that relates y to x .
18. This exercise compares and contrasts some similarity and distance measures.
- 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.
- $x = 0101010001$
 $y = 0100011000$
- 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.)

- 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.)
- 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.)
- For the following vectors, x and y , calculate the indicated similarity or distance measures.
 - $x = (1, 1, 1, 1)$, $y = (2, 2, 2, 2)$ cosine, correlation, Euclidean
 - $x = (0, 1, 0, 1)$, $y = (1, 0, 1, 0)$ cosine, correlation, Euclidean, Jaccard
 - $x = (0, -1, 0, 1)$, $y = (1, 0, -1, 0)$ cosine, correlation, Euclidean
 - $x = (1, 1, 0, 1, 0, 1)$, $y = (1, 1, 1, 0, 0, 1)$ cosine, correlation, Jaccard
 - $x = (2, -1, 0, 2, 0, -3)$, $y = (-1, 1, -1, 0, 0, -1)$ cosine, correlation
- Here, we further explore the cosine and correlation measures.
 - What is the range of values that are possible for the cosine measure?
 - If two objects have a cosine measure of 1, are they identical? Explain.
 - 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.)
 - 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?
 - 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?
 - Derive the mathematical relationship between cosine similarity and Euclidean distance when each data object has an L2 length of 1.
 - Derive the mathematical relationship between correlation and Euclidean distance when each data point has 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 x . (It does not matter if $x \in S$.)

- (a) If the goal is to find all points within a specified distance ϵ of point y , $y \neq x$, explain how you could use the triangle inequality and the already calculated distances to x to potentially reduce the number of distance calculations necessary? Hint: The triangle inequality, $d(x, z) \leq d(x, y) + d(y, z)$, can be rewritten as $d(x, y) \geq d(x, z) - d(y, z)$.
 - (b) In general, how would the distance between x and 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, x and y , that satisfies the metric axioms given on page 70. Specifically, $d(x, y) = 1 - J(x, y)$.
27. Show that the distance measure defined as the angle between two data vectors, x and y , satisfies the metric axioms given on page 70. Specifically, $d(x, y) = \arccos(\cos(x, y))$.
28. Explain why computing the proximity between two attributes is often simpler than computing the similarity between two objects.

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.

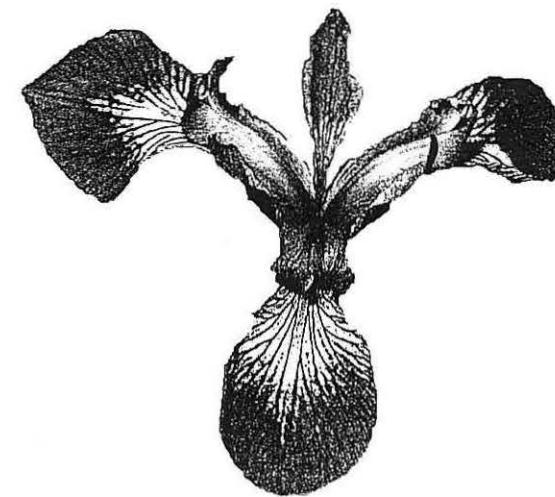


Figure 3.1. Picture of Iris Virginica. Robert H. Mohlenbrock @ USDA-NRCS PLANTS Database/USDA NRCS. 1995. Northeast wetland flora: Field office guide to plant species. Northeast National Technical Center, Chester, PA. Background removed.

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 *{freshman, sophomore, junior, 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 50th 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{x} , is given by

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

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** 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** 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 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

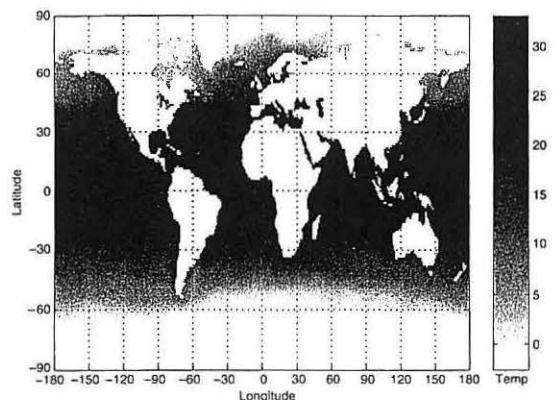


Figure 3.2. Sea Surface Temperature (SST) for July, 1982.

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. ■

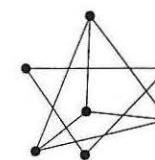
Table 3.5. A table of nine objects (rows) with six binary attributes (columns).

	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

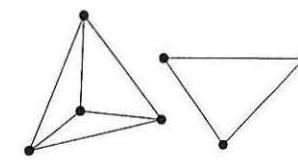
Table 3.6. A table of nine objects (rows) with six binary attributes (columns) permuted so that the relationships of the rows and columns are clear.

	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 49 49 49 49 49 49 49 50
50 50 50 50 50 50 50 50 51 51 51 51 51 51 51 51 52 52 52 52 53
54 54 54 54 54 54 55 55 55 55 55 55 55 56 56 56 56 56 57 57 57
57 57 57 57 58 58 58 58 58 58 59 59 59 60 60 60 60 60 61 61 61
61 61 61 62 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 : 000000000111111122234444455555566666777777888888999
6 : 00000111112222333333334444445555667777778889999
7 : 0122234677779

```

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

```

4 : 3444
4 : 56666778888899999
5 : 0000000001111111222344444
5 : 55555566666677777888888999
6 : 000001111122223333333344444444
6 : 555556677777778889999
7 : 0122234
7 : 677779

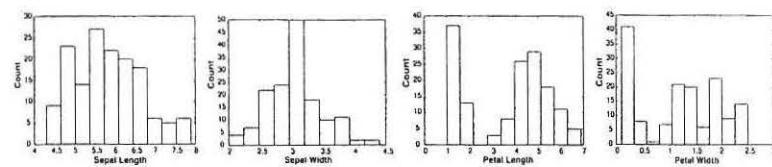
```

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.

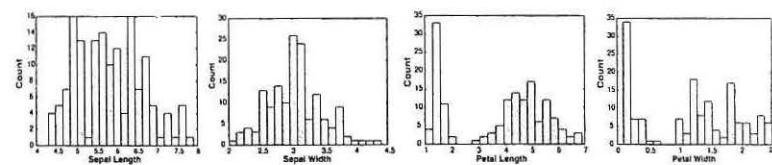
Example 3.8. Figure 3.7 shows histograms (with 10 bins) for sepal length, sepal width, petal length, and petal width. Since the shape of a histogram can depend on the number of bins, histograms for the same data, but with 20 bins, are shown in Figure 3.8.

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.

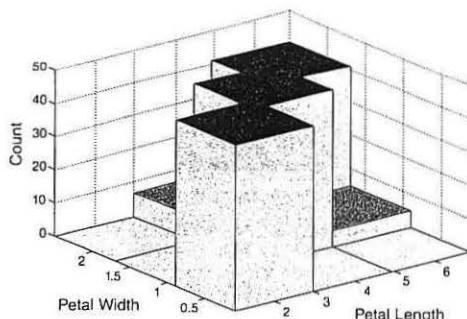


Figure 3.9. Two-dimensional histogram of petal length and width in the Iris data set.

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

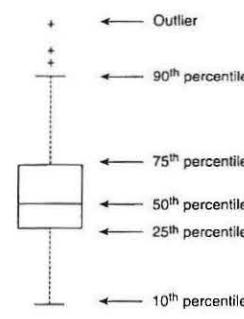


Figure 3.10. Description of box plot for sepal length.

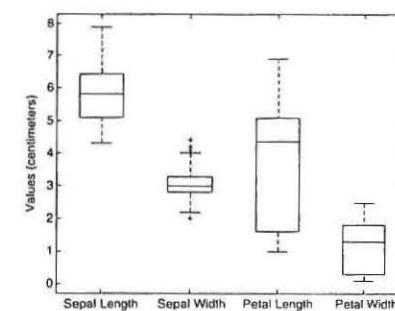
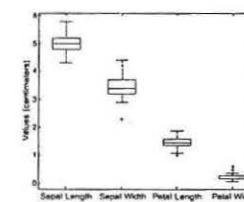
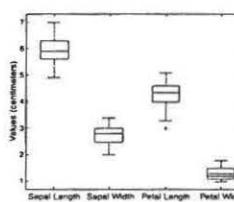


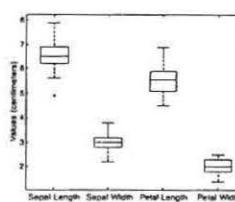
Figure 3.11. Box plot for Iris attributes.



(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

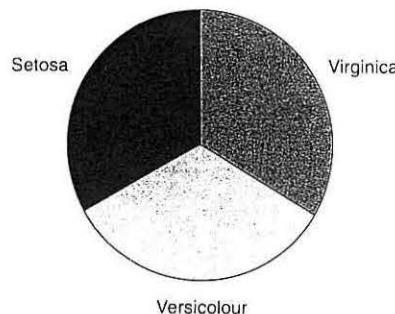


Figure 3.13. Distribution of the types of Iris flowers.

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.

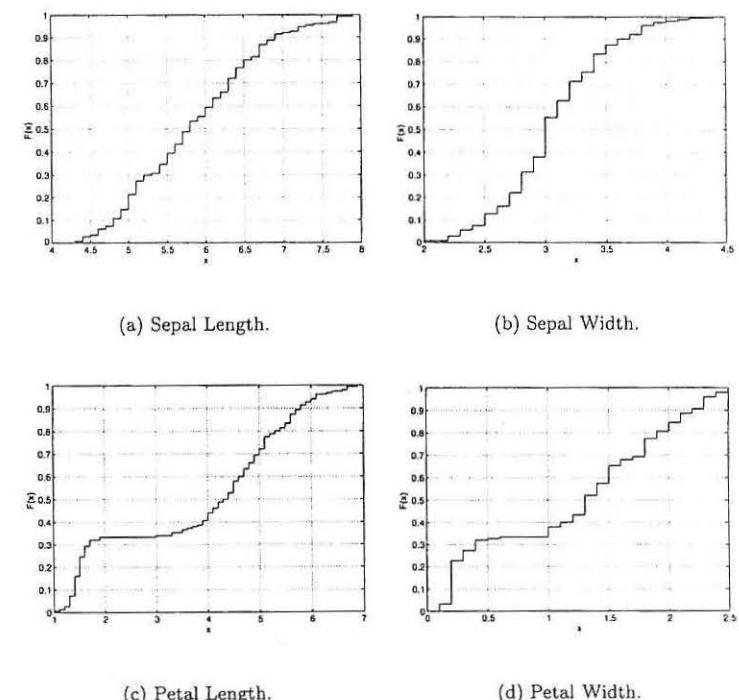


Figure 3.14. Empirical CDFs of four Iris attributes.

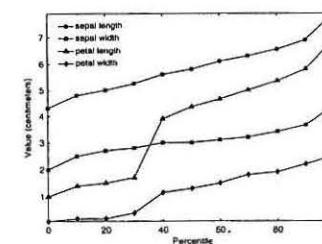


Figure 3.15. Percentile plots for sepal length, sepal width, petal length, and petal width.

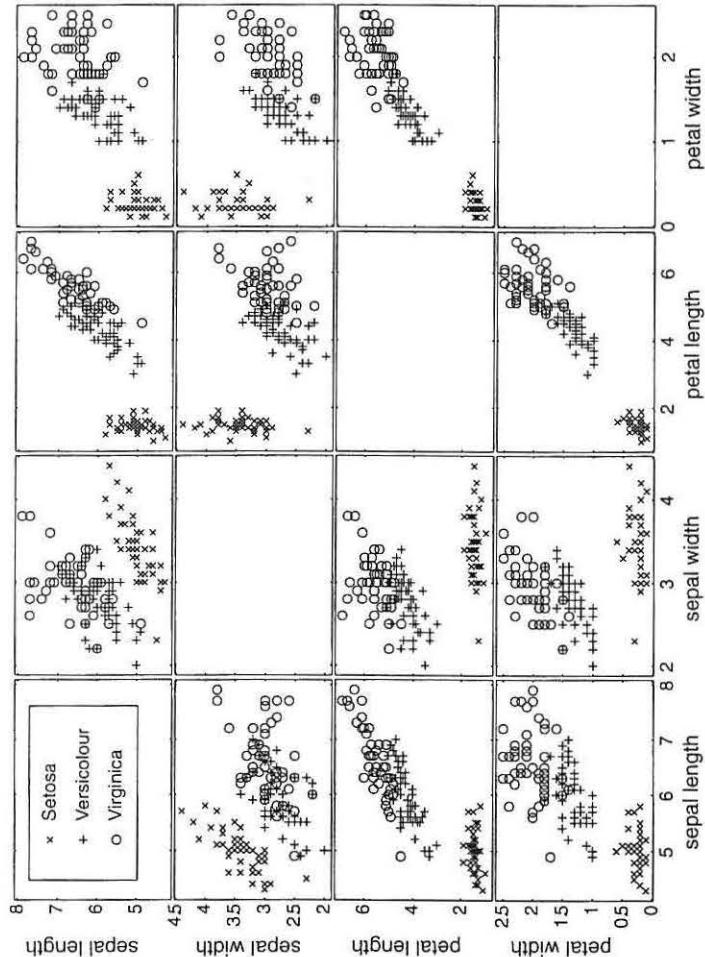


Figure 3.16. Matrix of scatter plots for the Iris data set.

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

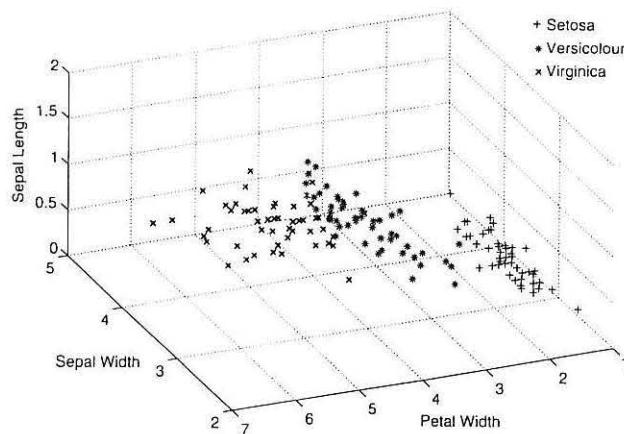


Figure 3.17. Three-dimensional scatter plot of sepal width, sepal length, and petal width.

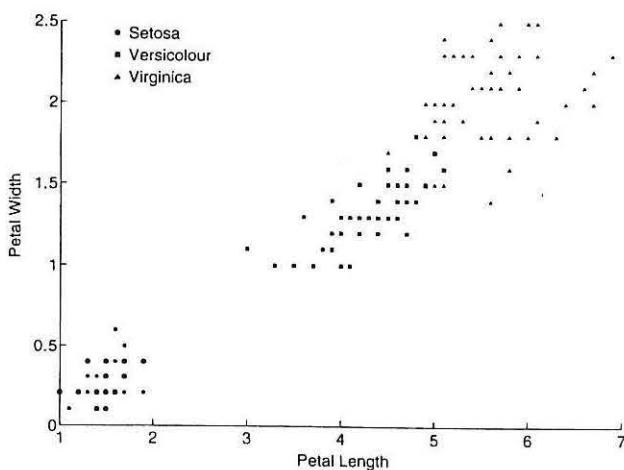


Figure 3.18. Scatter plot of petal length versus petal width, with the size of the marker indicating sepal width.

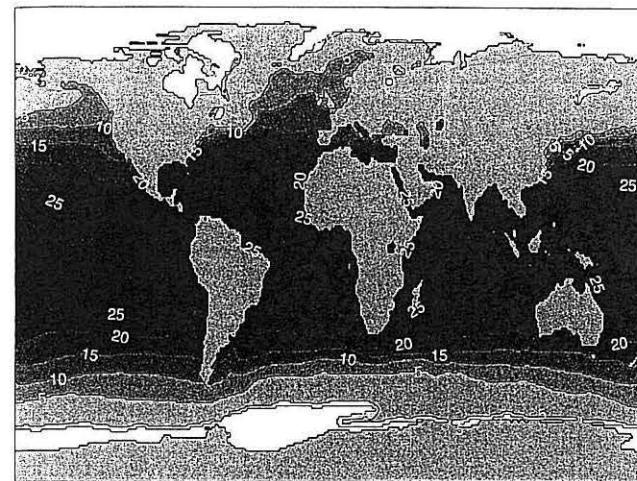


Figure 3.19. Contour plot of SST for December 1998.

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

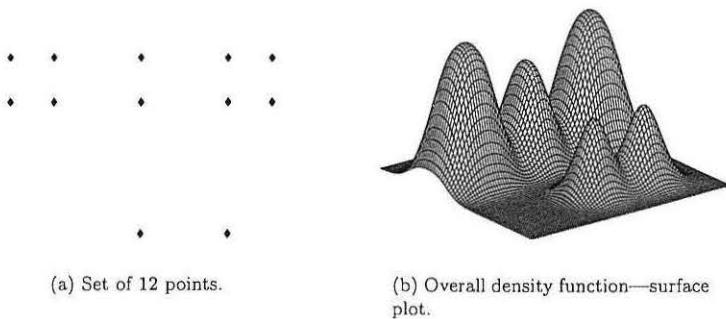


Figure 3.20. Density of a set of 12 points.

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

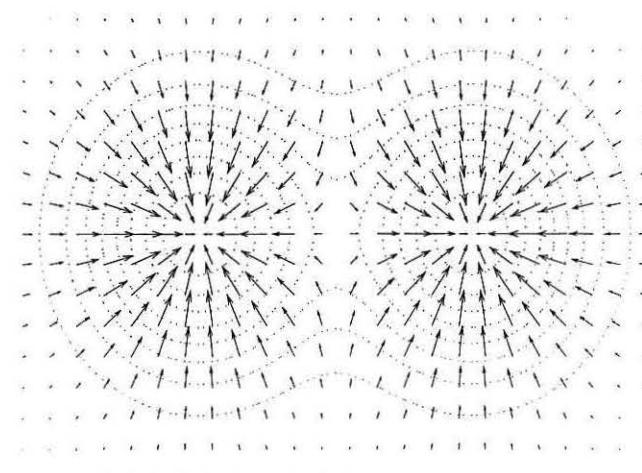


Figure 3.21. Vector plot of the gradient (change) in density for the bottom two density peaks of Figure 3.20.

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.

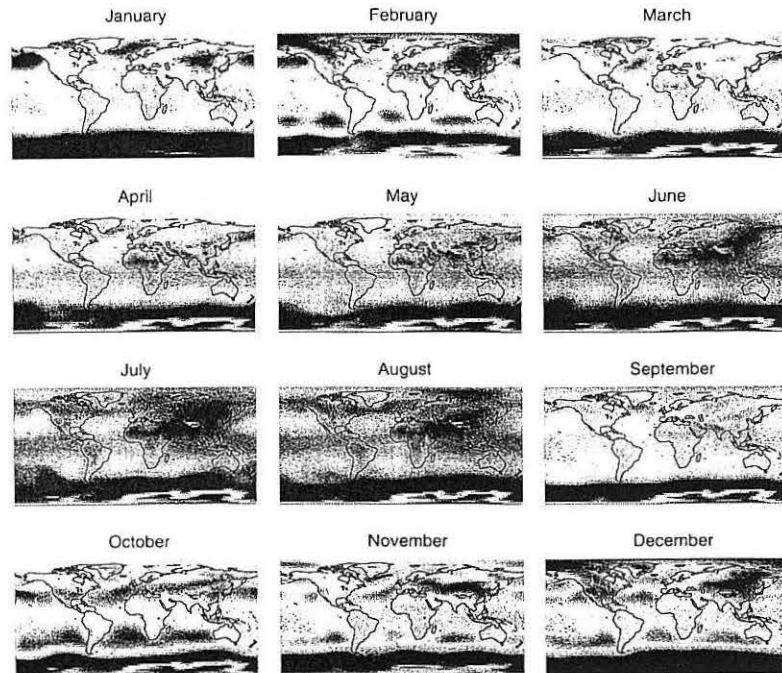


Figure 3.22. Monthly plots of sea level pressure over the 12 months of 1982.

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.

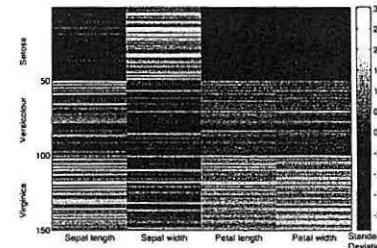


Figure 3.23. Plot of the Iris data matrix where columns have been standardized to have a mean of 0 and standard deviation of 1.

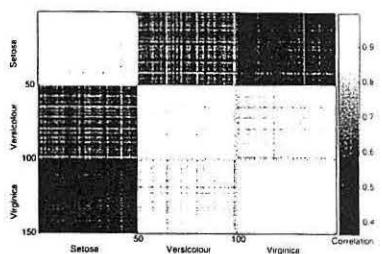


Figure 3.24. Plot of the Iris correlation matrix.

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

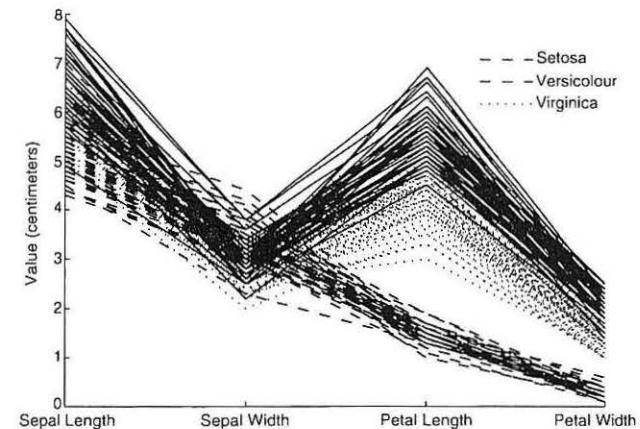


Figure 3.25. A parallel coordinates plot of the four Iris attributes.

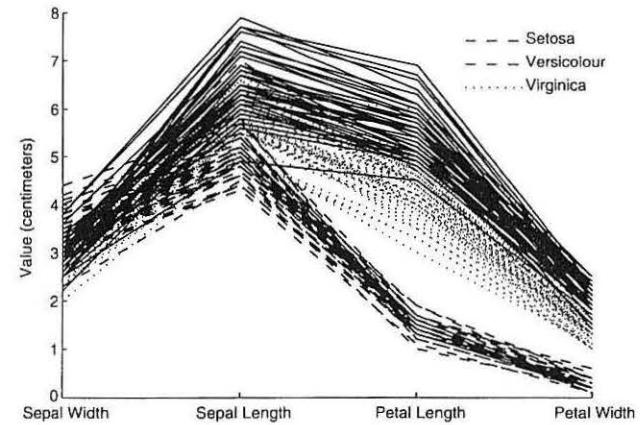


Figure 3.26. A parallel coordinates plot of the four Iris attributes with the attributes reordered to emphasize similarities and dissimilarities of groups.

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].

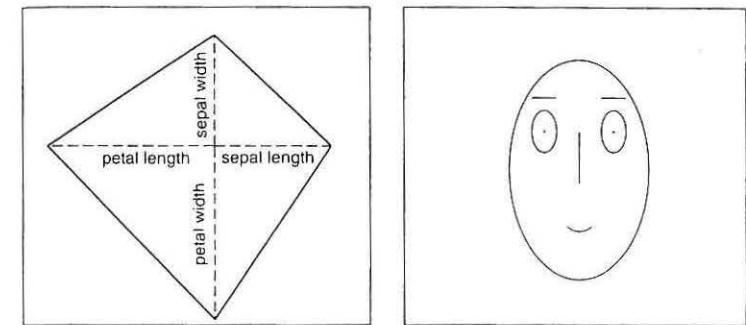
An object is mapped onto this star-shaped set of axes using the following process: Each attribute value of the object is converted to a fraction that represents its distance between the minimum and maximum values of the attribute. This fraction is mapped to a point on the axis corresponding to this attribute. 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.

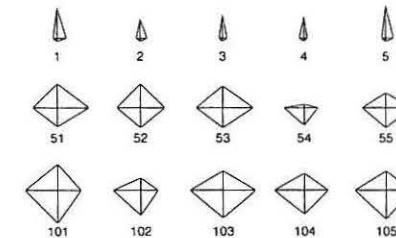
Figure 3.27. Star coordinates graph and Chernoff face of the 150th flower of the Iris data set.

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

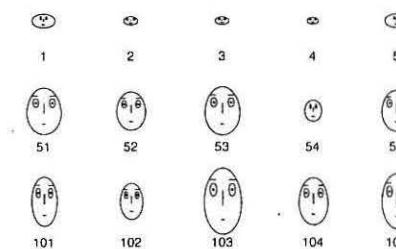


Figure 3.29. A plot of 15 Iris flowers using Chernoff faces.

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

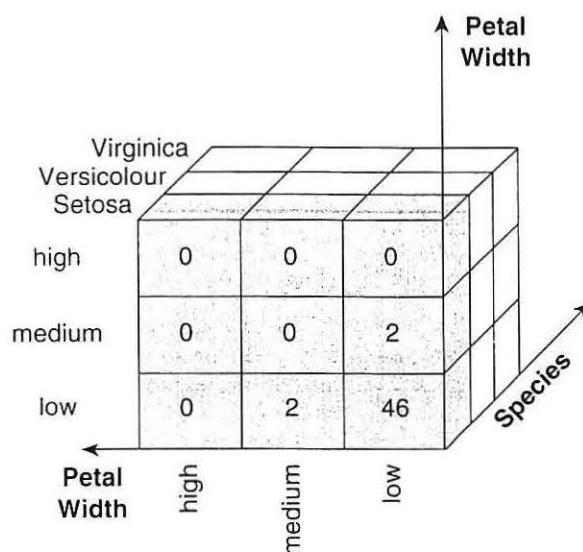


Figure 3.30. A multidimensional data representation for the Iris data set.

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

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

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

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

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

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

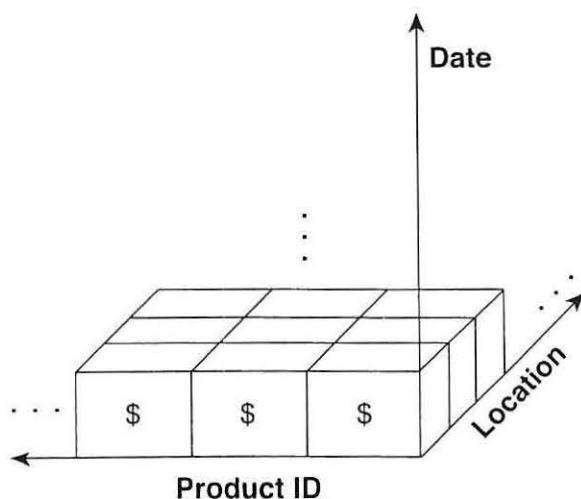


Figure 3.31. Multidimensional data representation for sales data.

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

product ID	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.

product ID	date			total
	Jan 1, 2004	Jan 2, 2004	... Dec 31, 2004	
1	\$1,001	\$987	...	\$370,000
27	\$10,265	\$10,225	...	\$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. Reichart, 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>, 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>, 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>, 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.

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.

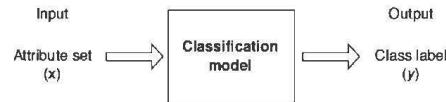


Figure 4.2. Classification as the task of mapping an input attribute set x into its class label y .

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 (x, y) , where 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 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	warm-blooded	hair	yes	no	yes	yes	yes	mammal
bat	warm-blooded	feathers	no	no	yes	yes	yes	bird
pigeon	warm-blooded	fur	yes	no	no	yes	no	mammal
cat	warm-blooded	feathers	yes	yes	no	no	no	fish
leopard	cold-blooded	fur	yes	yes	no	no	no	reptile
shark	cold-blooded	quills	no	semi	no	yes	no	bird
turtle	cold-blooded	quills	yes	no	no	yes	no	mammal
penguin	warm-blooded	quills	no	semi	no	yes	no	fish
porcupine	warm-blooded	quills	yes	no	no	yes	yes	amphibian
eel	cold-blooded	quills	no	yes	no	no	yes	reptile
salamander	cold-blooded	none	no	semi	no	yes	yes	bird

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

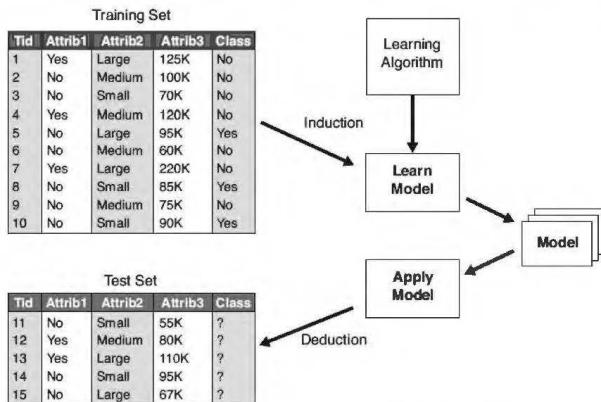


Figure 4.3. General approach for building a classification model.

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

		Predicted Class	
		Class = 1	Class = 0
Actual Class	Class = 1	f_{11}	f_{10}
	Class = 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**

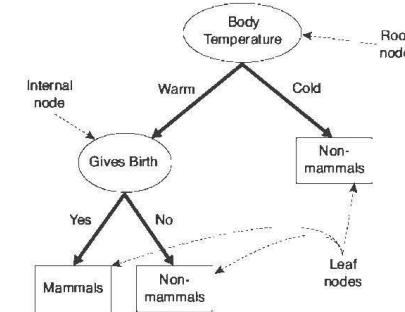


Figure 4.4. A decision tree for the mammal classification problem.

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-

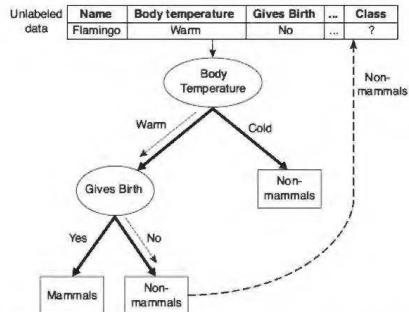


Figure 4.5. Classifying an unlabeled vertebrate. The dashed lines represent the outcomes of applying various attribute test conditions on the unlabeled vertebrate. The vertebrate is eventually assigned to the Non-mammal class.

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.

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).

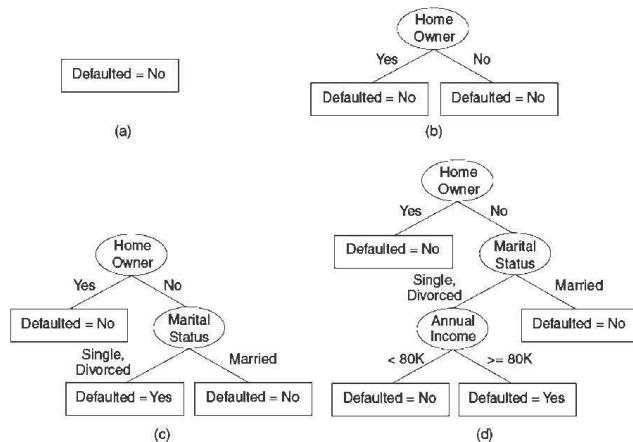


Figure 4.7. Hunt's algorithm for inducing decision trees.

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.

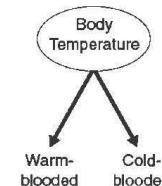


Figure 4.8. Test condition for binary attributes.

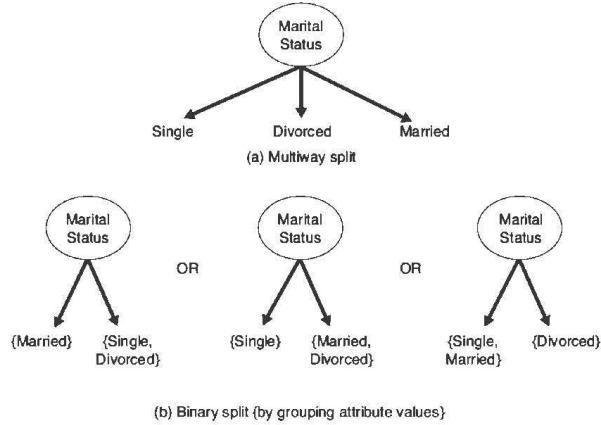


Figure 4.9. Test conditions for nominal attributes.

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

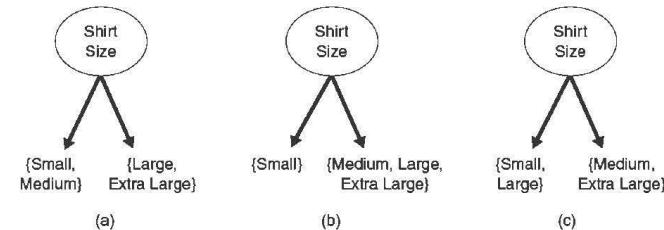


Figure 4.10. Different ways of grouping ordinal attribute values

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.

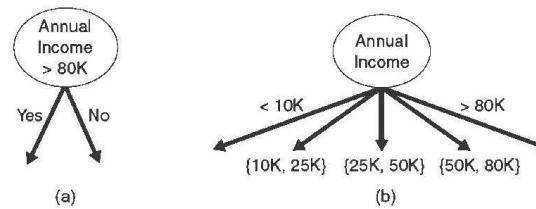


Figure 4.11. Test condition for continuous attributes

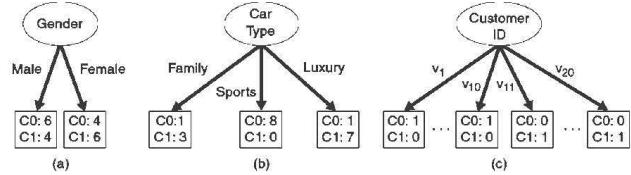


Figure 4.12. Multiway versus binary splits.

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.

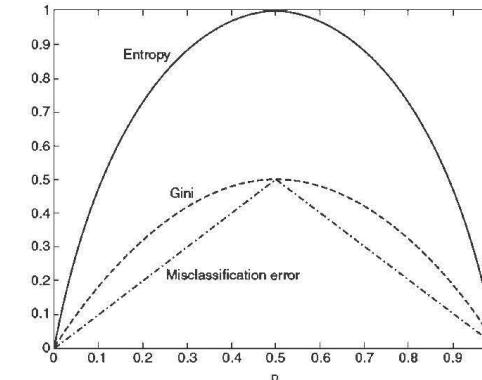


Figure 4.13. Comparison among the impurity measures for binary classification problems.

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

Node N_2	Count
Class=0	1
Class=1	5

Node N_3	Count
Class=0	3
Class=1	3

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

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

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

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 N1 is 0.4898, and for node N2, it is 0.480. The weighted average of the Gini index for the descendant 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,

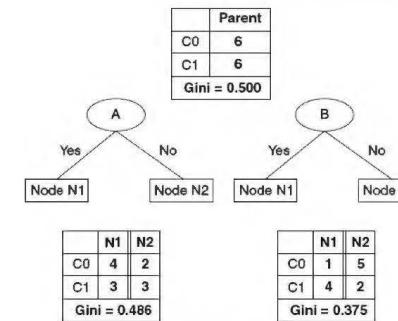


Figure 4.14. Splitting binary attributes

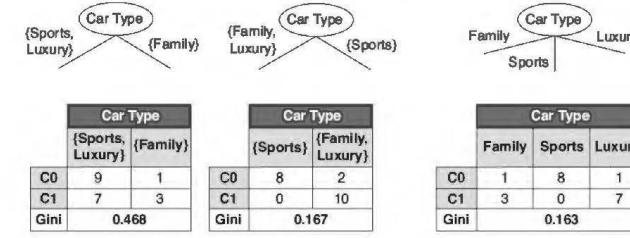


Figure 4.15. Splitting nominal attributes

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	1	2	2	1	3	0	3	0	3	0	3	0		
No	0	7	1	6	2	5	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* $\leq \$65K$) 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 descendent 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` = `createNode`().
- 3: `leaf.label` = `Classify`(E).
- 4: return `leaf`.
- 5: else
- 6: `root` = `createNode`().
- 7: `root.test_cond` = `find_best_split`(E, F).
- 8: let V = { v | v is a possible outcome of root.test_cond }.
- 9: for each $v \in V$ do
- 10: E_v = { e | $\text{root.test_cond}(e) = v$ and $e \in E$ }.
- 11: `child` = `TreeGrowth`(E_v, F).
- 12: add `child` as descendent of `root` and label the edge ($\text{root} \rightarrow \text{child}$) 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:

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

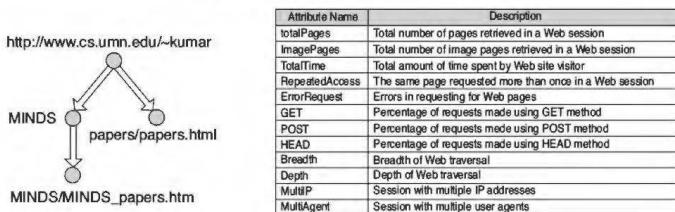
where the `argmax` operator returns the argument i that maximizes the expression $p(i|t)$. Besides providing the information needed to determine the class label of a leaf node, the fraction $p(i|t)$ can also be used to estimate the probability that a record assigned to the leaf node t belongs to class 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.

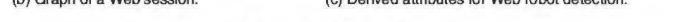
After building the decision tree, a tree-pruning step can be performed to reduce the size of the decision tree. Decision trees that are too large are susceptible to a phenomenon known as overfitting. Pruning helps by trimming the branches of the initial tree in a way that improves the generalization 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.htm	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.



(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:
| | | MultiIP = 0:
| | | | ImagePages<= 0.1333: class 1
| | | | | ImagePages> 0.1333:
| | | | | | breadth <= 6: class 0
| | | | | | breadth > 6: class 1
| | | | | MultiIP = 1:
| | | | | | TotalTime <= 361: class 0
| | | | | | TotalTime > 361: class 1
| | | depth>1:
| | | | MultiAgent = 0:
| | | | | depth > 2: class 0
| | | | | depth < 2:
| | | | | | MultiIP = 1: class 0
| | | | | | MultiIP = 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.

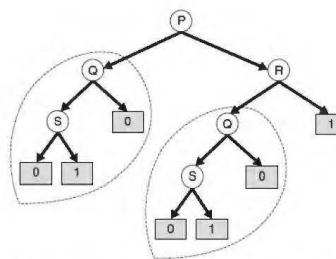


Figure 4.19. Tree replication problem. The same subtree can appear at different branches.

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

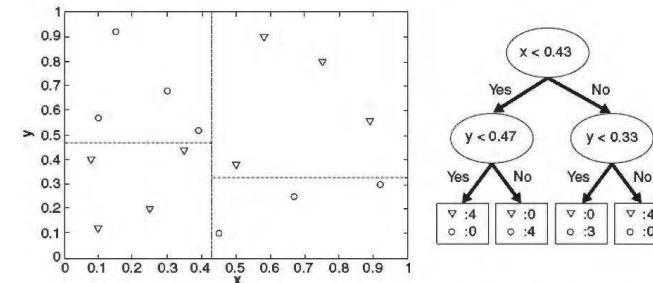


Figure 4.20. Example of a decision tree and its decision boundaries for a two-dimensional data set.

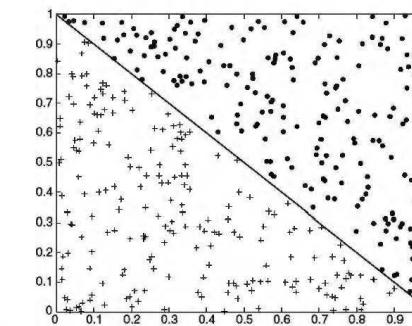


Figure 4.21. Example of data set that cannot be partitioned optimally using test conditions involving single attributes.

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

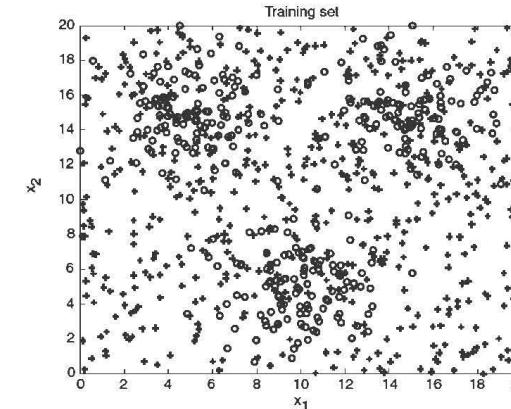


Figure 4.22. Example of a data set with binary classes.

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 o and class +, respectively. The data points for the o 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 o 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(h) shows the training and test error rates of the decision tree.

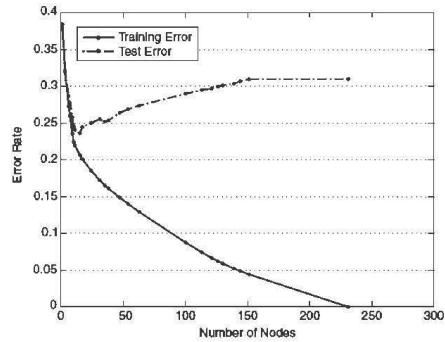


Figure 4.23. Training and test error rates.

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 numbers 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.

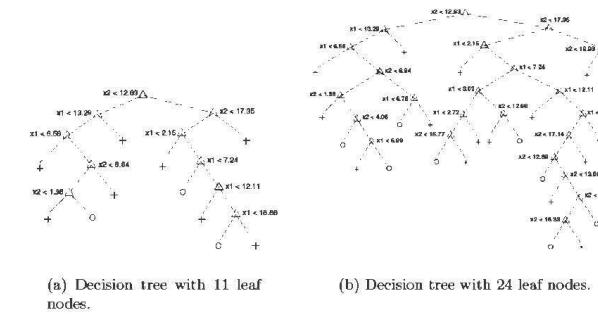


Figure 4.24. Decision trees with different model complexities.

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

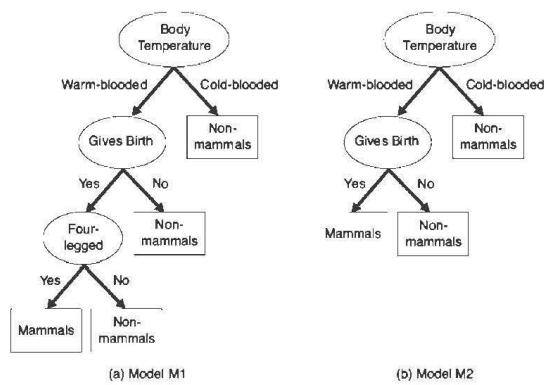


Figure 4.25. Decision tree induced from the data set shown in Table 4.3.

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.

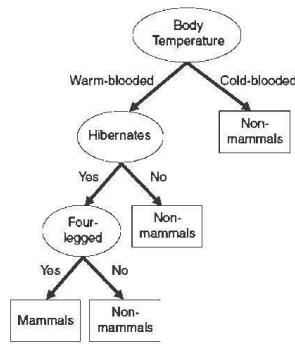


Figure 4.26. Decision tree induced from the data set shown in Table 4.5.

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

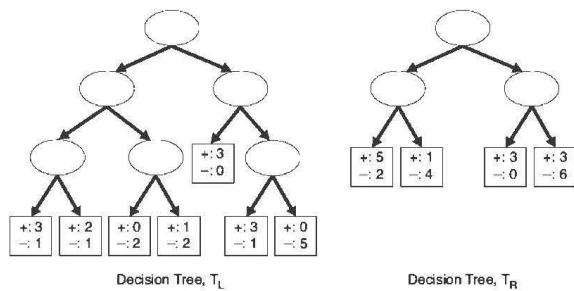


Figure 4.27. Example of two decision trees generated from the same training data.

$e(T_R) = 6/24 = 0.25$. 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

$$e_g(T_R) = \frac{6 + 4 \times 0.5}{24} = \frac{8}{24} = 0.3333.$$

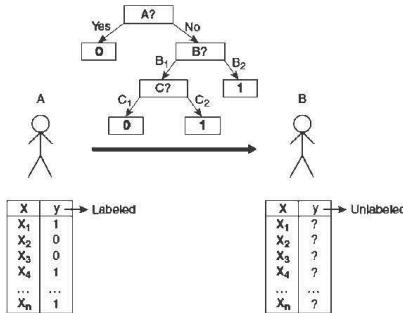


Figure 4.28. The minimum description length (MDL) principle.

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 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 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

$$\text{Cost(model, data)} = \text{Cost(model)} + \text{Cost(data|model)}, \quad (4.9)$$

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 $\epsilon_{upper}(4, 1/4, 0.25) = 0.537$ and $\epsilon_{upper}(3, 1/3, 0.25) = 0.650$, 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,

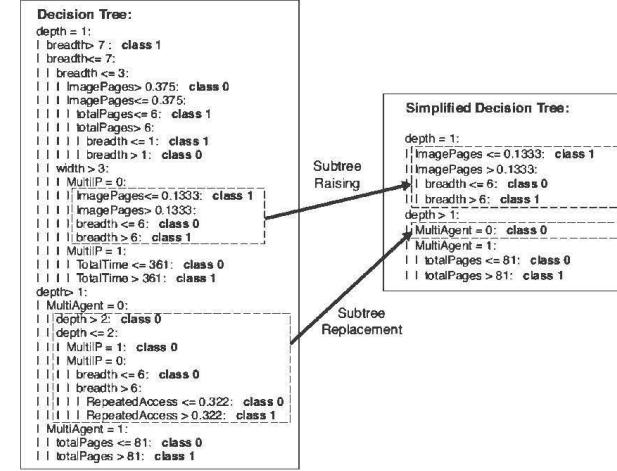


Figure 4.29. Post-pruning of the decision tree for Web robot detection.

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 $acc_{sub} = \sum_{i=1}^k acc_i/k$. 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):

$$\text{Accuracy, } acc_{\text{boot}} = \frac{1}{b} \sum_{i=1}^b (0.632 \times \epsilon_i + 0.368 \times acc_s). \quad (4.11)$$

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	0.584	0.670	0.711	0.763	0.774	0.789
Interval	-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:

$$\sigma_d^2 \simeq \hat{\sigma}_d^2 = \frac{e_1(1 - e_1)}{n_1} + \frac{e_2(1 - e_2)}{n_2}, \quad (4.14)$$

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 $\sigma_{d^{cv}}$. 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 Langrebe [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. Sabberg. 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 Trans. Patt. Anal. and Mach. Intell.*, 22(1):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. Lingosi, 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. Schuemann 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 Int'l. 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 Int'l. 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.

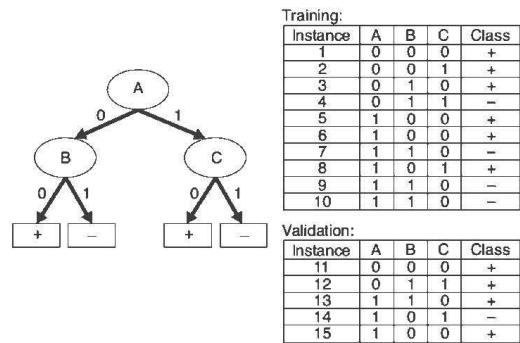


Figure 4.30. Decision tree and data sets for Exercise 8.

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 .

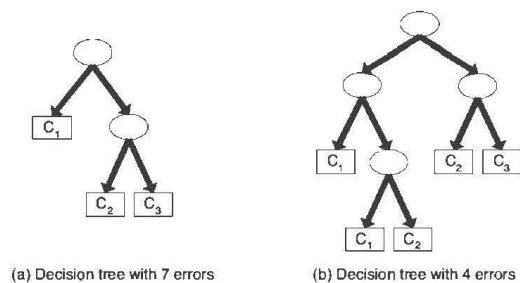


Figure 4.31. Decision trees for Exercise 9.

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:

$$\text{Cost(tree,data)} = \text{Cost(tree)} + \text{Cost(data|tree}).$$

- 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.
- The holdout method, where two-thirds of the data are used for training and the remaining one-third are used for testing.
 - Ten-fold cross-validation.
 - The .632 bootstrap method.
 - 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	80.17	82.34	88.89
Iris	150	94.07	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 $p(1-p)/N$.

Association Analysis: Basic Concepts and Algorithms

Many business enterprises accumulate large quantities of data from their day-to-day operations. For example, huge amounts of customer purchase data are collected daily at the checkout counters of grocery stores. Table 6.1 illustrates an example of such data, commonly known as market basket transactions. Each row in this table corresponds to a transaction, which contains a unique identifier labeled *TID* and a set of items bought by a given customer. Retailers are interested in analyzing the data to learn about the purchasing behavior of their customers. Such valuable information can be used to support a variety of business-related applications such as marketing promotions, inventory management, and customer relationship management.

This chapter presents a methodology known as association analysis, which is useful for discovering interesting relationships hidden in large data sets. The uncovered relationships can be represented in the form of associa-

Table 6.1. An example of market basket transactions.

<i>TID</i>	Items
1	{Bread, Milk}
2	{Bread, Diapers, Beer, Eggs}
3	{Milk, Diapers, Beer, Cola}
4	{Bread, Milk, Diapers, Beer}
5	{Bread, Milk, Diapers, Cola}

tion rules or sets of frequent items. For example, the following rule can be extracted from the data set shown in Table 6.1:

$$\{\text{Diapers}\} \longrightarrow \{\text{Beer}\}.$$

The rule suggests that a strong relationship exists between the sale of diapers and beer because many customers who buy diapers also buy beer. Retailers can use this type of rules to help them identify new opportunities for cross-selling their products to the customers.

Besides market basket data, association analysis is also applicable to other application domains such as bioinformatics, medical diagnosis, Web mining, and scientific data analysis. In the analysis of Earth science data, for example, the association patterns may reveal interesting connections among the ocean, land, and atmospheric processes. Such information may help Earth scientists develop a better understanding of how the different elements of the Earth system interact with each other. Even though the techniques presented here are generally applicable to a wider variety of data sets, for illustrative purposes, our discussion will focus mainly on market basket data.

There are two key issues that need to be addressed when applying association analysis to market basket data. First, discovering patterns from a large transaction data set can be computationally expensive. Second, some of the discovered patterns are potentially spurious because they may happen simply by chance. The remainder of this chapter is organized around these two issues. The first part of the chapter is devoted to explaining the basic concepts of association analysis and the algorithms used to efficiently mine such patterns. The second part of the chapter deals with the issue of evaluating the discovered patterns in order to prevent the generation of spurious results.

6.1 Problem Definition

This section reviews the basic terminology used in association analysis and presents a formal description of the task.

Binary Representation Market basket data can be represented in a binary format as shown in Table 6.2, where each row corresponds to a transaction and each column corresponds to an item. An item can be treated as a binary variable whose value is one if the item is present in a transaction and zero otherwise. Because the presence of an item in a transaction is often considered more important than its absence, an item is an asymmetric binary variable.

Table 6.2. A binary 0/1 representation of market basket data.

TID	Bread	Milk	Diapers	Beer	Eggs	Cola
1	1	1	0	0	0	0
2	1	0	1	1	1	0
3	0	1	1	1	0	1
4	1	1	1	1	0	0
5	1	1	1	0	0	1

This representation is perhaps a very simplistic view of real market basket data because it ignores certain important aspects of the data such as the quantity of items sold or the price paid to purchase them. Methods for handling such non-binary data will be explained in Chapter 7.

Itemset and Support Count Let $I = \{i_1, i_2, \dots, i_d\}$ be the set of all items in a market basket data and $T = \{t_1, t_2, \dots, t_N\}$ be the set of all transactions. Each transaction t_i contains a subset of items chosen from I . In association analysis, a collection of zero or more items is termed an itemset. If an itemset contains k items, it is called a k -itemset. For instance, $\{\text{Beer}, \text{Diapers}, \text{Milk}\}$ is an example of a 3-itemset. The null (or empty) set is an itemset that does not contain any items.

The transaction width is defined as the number of items present in a transaction. A transaction t_j is said to contain an itemset X if X is a subset of t_j . For example, the second transaction shown in Table 6.2 contains the itemset $\{\text{Bread}, \text{Diapers}\}$ but not $\{\text{Bread}, \text{Milk}\}$. An important property of an itemset is its support count, which refers to the number of transactions that contain a particular itemset. Mathematically, the support count, $\sigma(X)$, for an itemset X can be stated as follows:

$$\sigma(X) = |\{t_i | X \subseteq t_i, t_i \in T\}|,$$

where the symbol $|\cdot|$ denotes the number of elements in a set. In the data set shown in Table 6.2, the support count for $\{\text{Beer}, \text{Diapers}, \text{Milk}\}$ is equal to two because there are only two transactions that contain all three items.

Association Rule An association rule is an implication expression of the form $X \longrightarrow Y$, where X and Y are disjoint itemsets, i.e., $X \cap Y = \emptyset$. The strength of an association rule can be measured in terms of its support and confidence. Support determines how often a rule is applicable to a given

data set, while confidence determines how frequently items in Y appear in transactions that contain X . The formal definitions of these metrics are

$$\text{Support, } s(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{N}; \quad (6.1)$$

$$\text{Confidence, } c(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{\sigma(X)}. \quad (6.2)$$

Example 6.1. Consider the rule $\{\text{Milk, Diapers}\} \rightarrow \{\text{Beer}\}$. Since the support count for $\{\text{Milk, Diapers, Beer}\}$ is 2 and the total number of transactions is 5, the rule's support is $2/5 = 0.4$. The rule's confidence is obtained by dividing the support count for $\{\text{Milk, Diapers, Beer}\}$ by the support count for $\{\text{Milk, Diapers}\}$. Since there are 3 transactions that contain milk and diapers, the confidence for this rule is $2/3 = 0.67$. ■

Why Use Support and Confidence? Support is an important measure because a rule that has very low support may occur simply by chance. A low support rule is also likely to be uninteresting from a business perspective because it may not be profitable to promote items that customers seldom buy together (with the exception of the situation described in Section 6.8). For these reasons, support is often used to eliminate uninteresting rules. As will be shown in Section 6.2.1, support also has a desirable property that can be exploited for the efficient discovery of association rules.

Confidence, on the other hand, measures the reliability of the inference made by a rule. For a given rule $X \rightarrow Y$, the higher the confidence, the more likely it is for Y to be present in transactions that contain X . Confidence also provides an estimate of the conditional probability of Y given X .

Association analysis results should be interpreted with caution. The inference made by an association rule does not necessarily imply causality. Instead, it suggests a strong co-occurrence relationship between items in the antecedent and consequent of the rule. Causality, on the other hand, requires knowledge about the causal and effect attributes in the data and typically involves relationships occurring over time (e.g., ozone depletion leads to global warming).

Formulation of Association Rule Mining Problem The association rule mining problem can be formally stated as follows:

Definition 6.1 (Association Rule Discovery). Given a set of transactions T , find all the rules having support $\geq \text{minsup}$ and confidence $\geq \text{minconf}$, where minsup and minconf are the corresponding support and confidence thresholds.

A brute-force approach for mining association rules is to compute the support and confidence for every possible rule. This approach is prohibitively expensive because there are exponentially many rules that can be extracted from a data set. More specifically, the total number of possible rules extracted from a data set that contains d items is

$$R = 3^d - 2^{d+1} + 1. \quad (6.3)$$

The proof for this equation is left as an exercise to the readers (see Exercise 5 on page 405). Even for the small data set shown in Table 6.1, this approach requires us to compute the support and confidence for $3^6 - 2^7 + 1 = 602$ rules. More than 80% of the rules are discarded after applying $\text{minsup} = 20\%$ and $\text{minconf} = 50\%$, thus making most of the computations become wasted. To avoid performing needless computations, it would be useful to prune the rules early without having to compute their support and confidence values.

An initial step toward improving the performance of association rule mining algorithms is to decouple the support and confidence requirements. From Equation 6.2, notice that the support of a rule $X \rightarrow Y$ depends only on the support of its corresponding itemset, $X \cup Y$. For example, the following rules have identical support because they involve items from the same itemset, $\{\text{Beer, Diapers, Milk}\}$:

$$\begin{aligned} \{\text{Beer, Diapers}\} &\rightarrow \{\text{Milk}\}, \quad \{\text{Beer, Milk}\} \rightarrow \{\text{Diapers}\}, \\ \{\text{Diapers, Milk}\} &\rightarrow \{\text{Beer}\}, \quad \{\text{Beer}\} \rightarrow \{\text{Diapers, Milk}\}, \\ \{\text{Milk}\} &\rightarrow \{\text{Beer, Diapers}\}, \quad \{\text{Diapers}\} \rightarrow \{\text{Beer, Milk}\}. \end{aligned}$$

If the itemset is infrequent, then all six candidate rules can be pruned immediately without our having to compute their confidence values.

Therefore, a common strategy adopted by many association rule mining algorithms is to decompose the problem into two major subtasks:

1. **Frequent Itemset Generation**, whose objective is to find all the itemsets that satisfy the minsup threshold. These itemsets are called frequent itemsets.
2. **Rule Generation**, whose objective is to extract all the high-confidence rules from the frequent itemsets found in the previous step. These rules are called strong rules.

The computational requirements for frequent itemset generation are generally more expensive than those of rule generation. Efficient techniques for generating frequent itemsets and association rules are discussed in Sections 6.2 and 6.3, respectively.

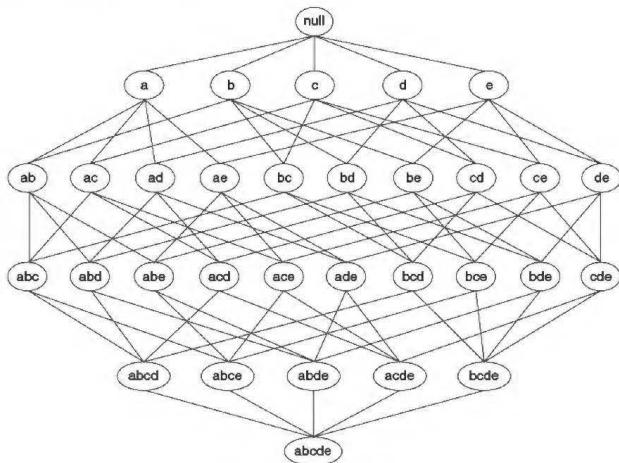


Figure 6.1. An itemset lattice.

6.2 Frequent Itemset Generation

A lattice structure can be used to enumerate the list of all possible itemsets. Figure 6.1 shows an itemset lattice for $I = \{a, b, c, d, e\}$. In general, a data set that contains k items can potentially generate up to $2^k - 1$ frequent itemsets, excluding the null set. Because k can be very large in many practical applications, the search space of itemsets that need to be explored is exponentially large.

A brute-force approach for finding frequent itemsets is to determine the support count for every candidate itemset in the lattice structure. To do this, we need to compare each candidate against every transaction, an operation that is shown in Figure 6.2. If the candidate is contained in a transaction, its support count will be incremented. For example, the support for $\{\text{Bread}, \text{Milk}\}$ is incremented three times because the itemset is contained in transactions 1, 4, and 5. Such an approach can be very expensive because it requires $O(NMw)$ comparisons, where N is the number of transactions, $M = 2^k - 1$ is the number of candidate itemsets, and w is the maximum transaction width.

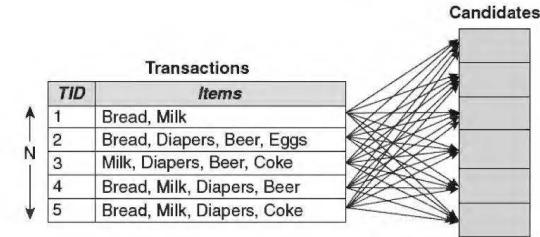


Figure 6.2. Counting the support of candidate itemsets

There are several ways to reduce the computational complexity of frequent itemset generation.

1. **Reduce the number of candidate itemsets (M).** The *Apriori* principle, described in the next section, is an effective way to eliminate some of the candidate itemsets without counting their support values.
 2. **Reduce the number of comparisons.** Instead of matching each candidate itemset against every transaction, we can reduce the number of comparisons by using more advanced data structures, either to store the candidate itemsets or to compress the data set. We will discuss these strategies in Sections 6.2.4 and 6.6.

6.2.1 The *A priori* Principle

This section describes how the support measure helps to reduce the number of candidate itemsets explored during frequent itemset generation. The use of support for pruning candidate itemsets is guided by the following principle.

Theorem 6.1 (Apriori Principle). If an itemset is frequent, then all of its subsets must also be frequent.

To illustrate the idea behind the *Apriori* principle, consider the itemset lattice shown in Figure 6.3. Suppose $\{c, d, e\}$ is a frequent itemset. Clearly, any transaction that contains $\{c, d, e\}$ must also contain its subsets, $\{c, d\}$, $\{c, e\}$, $\{d, e\}$, $\{c\}$, $\{d\}$, and $\{e\}$. As a result, if $\{c, d, e\}$ is frequent, then all subsets of $\{c, d, e\}$ (i.e., the shaded itemsets in this figure) must also be frequent.

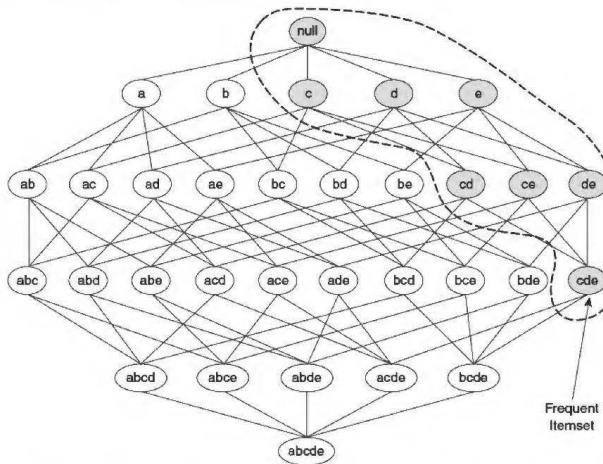


Figure 6.3. An illustration of the *Apriori* principle. If $\{c, d, e\}$ is frequent, then all subsets of this itemset are frequent.

Conversely, if an itemset such as $\{a, b\}$ is infrequent, then all of its supersets must be infrequent too. As illustrated in Figure 6.4, the entire subgraph containing the supersets of $\{a, b\}$ can be pruned immediately once $\{a, b\}$ is found to be infrequent. This strategy of trimming the exponential search space based on the support measure is known as **support-based pruning**. Such a pruning strategy is made possible by a key property of the support measure, namely, that the support for an itemset never exceeds the support for its subsets. This property is also known as the **anti-monotone** property of the support measure.

Definition 6.2 (Monotonicity Property). Let I be a set of items, and $J = 2^I$ be the power set of I . A measure f is monotone (or upward closed) if

$$\forall X, Y \in J : (X \subseteq Y) \rightarrow f(X) \leq f(Y),$$

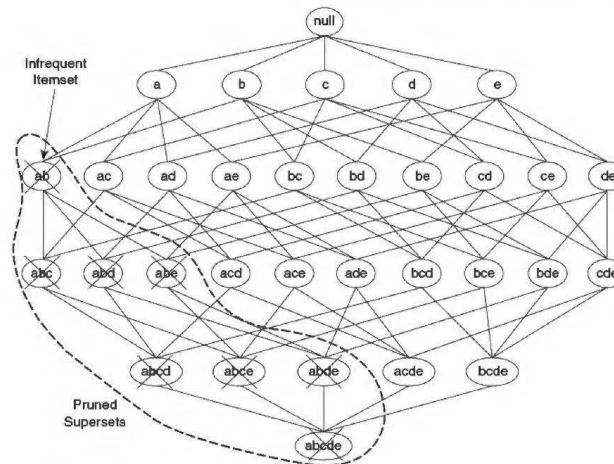


Figure 6.4. An illustration of support-based pruning. If $\{a, b\}$ is infrequent, then all supersets of $\{a, b\}$ are infrequent.

which means that if X is a subset of Y , then $f(X)$ must not exceed $f(Y)$. On the other hand, f is anti-monotone (or downward closed) if

$$\forall X, Y \in J : (X \subseteq Y) \rightarrow f(Y) \leq f(X),$$

which means that if X is a subset of Y , then $f(Y)$ must not exceed $f(X)$.

Any measure that possesses an anti-monotone property can be incorporated directly into the mining algorithm to effectively prune the exponential search space of candidate itemsets, as will be shown in the next section.

6.2.2 Frequent Itemset Generation in the *Apriori* Algorithm

Apriori is the first association rule mining algorithm that pioneered the use of support-based pruning to systematically control the exponential growth of candidate itemsets. Figure 6.5 provides a high-level illustration of the frequent itemset generation part of the *Apriori* algorithm for the transactions shown in

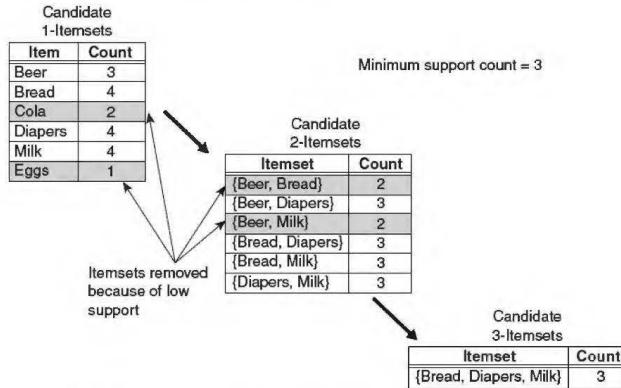
Figure 6.5. Illustration of frequent itemset generation using the *Apriori* algorithm.

Table 6.1. We assume that the support threshold is 60%, which is equivalent to a minimum support count equal to 3.

Initially, every item is considered as a candidate 1-itemset. After counting their supports, the candidate itemsets $\{\text{Cola}\}$ and $\{\text{Eggs}\}$ are discarded because they appear in fewer than three transactions. In the next iteration, candidate 2-itemsets are generated using only the frequent 1-itemsets because the *Apriori* principle ensures that all supersets of the infrequent 1-itemsets must be infrequent. Because there are only four frequent 1-itemsets, the number of candidate 2-itemsets generated by the algorithm is $\binom{4}{2} = 6$. Two of these six candidates, $\{\text{Beer, Bread}\}$ and $\{\text{Beer, Milk}\}$, are subsequently found to be infrequent after computing their support values. The remaining four candidates are frequent, and thus will be used to generate candidate 3-itemsets. Without support-based pruning, there are $\binom{6}{3} = 20$ candidate 3-itemsets that can be formed using the six items given in this example. With the *Apriori* principle, we only need to keep candidate 3-itemsets whose subsets are frequent. The only candidate that has this property is $\{\text{Bread, Diapers, Milk}\}$.

The effectiveness of the *Apriori* pruning strategy can be shown by counting the number of candidate itemsets generated. A brute-force strategy of

enumerating all itemsets (up to size 3) as candidates will produce

$$\binom{6}{1} + \binom{6}{2} + \binom{6}{3} = 6 + 15 + 20 = 41$$

candidates. With the *Apriori* principle, this number decreases to

$$\binom{6}{1} + \binom{4}{2} + 1 = 6 + 6 + 1 = 13$$

candidates, which represents a 68% reduction in the number of candidate itemsets even in this simple example.

The pseudocode for the frequent itemset generation part of the *Apriori* algorithm is shown in Algorithm 6.1. Let C_k denote the set of candidate k -itemsets and F_k denote the set of frequent k -itemsets:

- The algorithm initially makes a single pass over the data set to determine the support of each item. Upon completion of this step, the set of all frequent 1-itemsets, F_1 , will be known (steps 1 and 2).
- Next, the algorithm will iteratively generate new candidate k -itemsets using the frequent $(k - 1)$ -itemsets found in the previous iteration (step 5). Candidate generation is implemented using a function called *apriori-gen*, which is described in Section 6.2.3.

Algorithm 6.1 Frequent itemset generation of the *Apriori* algorithm.

```

1:  $k = 1$ .
2:  $F_k = \{ i \mid i \in I \wedge \sigma(\{i\}) \geq N \times \text{minsup} \}$ . {Find all frequent 1-itemsets}
3: repeat
4:    $k = k + 1$ .
5:    $C_k = \text{apriori-gen}(F_{k-1})$ . {Generate candidate itemsets}
6:   for each transaction  $t \in T$  do
7:      $C_t = \text{subset}(C_k, t)$ . {Identify all candidates that belong to  $t$ }
8:     for each candidate itemset  $c \in C_t$  do
9:        $\sigma(c) = \sigma(c) + 1$ . {Increment support count}
10:    end for
11:  end for
12:   $F_k = \{ c \mid c \in C_k \wedge \sigma(c) \geq N \times \text{minsup} \}$ . {Extract the frequent  $k$ -itemsets}
13: until  $F_k = \emptyset$ 
14: Result =  $\bigcup F_k$ .

```

- To count the support of the candidates, the algorithm needs to make an additional pass over the data set (steps 6–10). The subset function is used to determine all the candidate itemsets in C_k that are contained in each transaction t . The implementation of this function is described in Section 6.2.4.
- After counting their supports, the algorithm eliminates all candidate itemsets whose support counts are less than $minsup$ (step 12).
- The algorithm terminates when there are no new frequent itemsets generated, i.e., $F_k = \emptyset$ (step 13).

The frequent itemset generation part of the *Apriori* algorithm has two important characteristics. First, it is a level-wise algorithm; i.e., it traverses the itemset lattice one level at a time, from frequent 1-itemsets to the maximum size of frequent itemsets. Second, it employs a generate-and-test strategy for finding frequent itemsets. At each iteration, new candidate itemsets are generated from the frequent itemsets found in the previous iteration. The support for each candidate is then counted and tested against the $minsup$ threshold. The total number of iterations needed by the algorithm is $k_{\max} + 1$, where k_{\max} is the maximum size of the frequent itemsets.

6.2.3 Candidate Generation and Pruning

The *apriori-gen* function shown in Step 5 of Algorithm 6.1 generates candidate itemsets by performing the following two operations:

1. **Candidate Generation.** This operation generates new candidate k -itemsets based on the frequent $(k - 1)$ -itemsets found in the previous iteration.
2. **Candidate Pruning.** This operation eliminates some of the candidate k -itemsets using the support-based pruning strategy.

To illustrate the candidate pruning operation, consider a candidate k -itemset, $X = \{i_1, i_2, \dots, i_k\}$. The algorithm must determine whether all of its proper subsets, $X - \{i_j\}$ ($\forall j = 1, 2, \dots, k$), are frequent. If one of them is infrequent, then X is immediately pruned. This approach can effectively reduce the number of candidate itemsets considered during support counting. The complexity of this operation is $O(k)$ for each candidate k -itemset. However, as will be shown later, we do not have to examine all k subsets of a given candidate itemset. If m of the k subsets were used to generate a candidate, we only need to check the remaining $k - m$ subsets during candidate pruning.

In principle, there are many ways to generate candidate itemsets. The following is a list of requirements for an effective candidate generation procedure:

1. It should avoid generating too many unnecessary candidates. A candidate itemset is unnecessary if at least one of its subsets is infrequent. Such a candidate is guaranteed to be infrequent according to the anti-monotone property of support.
2. It must ensure that the candidate set is complete, i.e., no frequent itemsets are left out by the candidate generation procedure. To ensure completeness, the set of candidate itemsets must subsume the set of all frequent itemsets, i.e., $\forall k : F_k \subseteq C_k$.
3. It should not generate the same candidate itemset more than once. For example, the candidate itemset $\{a, b, c, d\}$ can be generated in many ways—by merging $\{a, b, c\}$ with $\{d\}$, $\{b, d\}$ with $\{a, c\}$, $\{c\}$ with $\{a, b, d\}$, etc. Generation of duplicate candidates leads to wasted computations and thus should be avoided for efficiency reasons.

Next, we will briefly describe several candidate generation procedures, including the one used by the *apriori-gen* function.

Brute-Force Method The brute-force method considers every k -itemset as a potential candidate and then applies the candidate pruning step to remove any unnecessary candidates (see Figure 6.6). The number of candidate itemsets generated at level k is equal to $\binom{d}{k}$, where d is the total number of items. Although candidate generation is rather trivial, candidate pruning becomes extremely expensive because a large number of itemsets must be examined. Given that the amount of computations needed for each candidate is $O(k)$, the overall complexity of this method is $O(\sum_{k=1}^d k \times \binom{d}{k}) = O(d \cdot 2^{d-1})$.

$F_{k-1} \times F_1$ Method An alternative method for candidate generation is to extend each frequent $(k - 1)$ -itemset with other frequent items. Figure 6.7 illustrates how a frequent 2-itemset such as $\{\text{Beer}, \text{Diapers}\}$ can be augmented with a frequent item such as Bread to produce a candidate 3-itemset $\{\text{Beer}, \text{Diapers}, \text{Bread}\}$. This method will produce $O(|F_{k-1}| \times |F_1|)$ candidate k -itemsets, where $|F_j|$ is the number of frequent j -itemsets. The overall complexity of this step is $O(\sum_k k |F_{k-1}| |F_1|)$.

The procedure is complete because every frequent k -itemset is composed of a frequent $(k - 1)$ -itemset and a frequent 1-itemset. Therefore, all frequent k -itemsets are part of the candidate k -itemsets generated by this procedure.

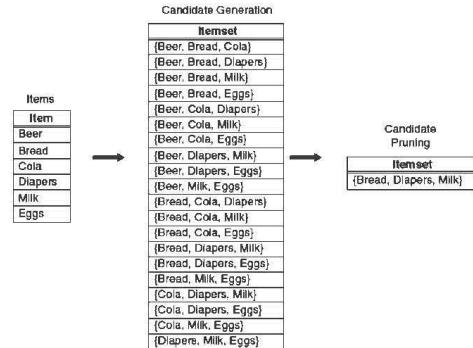
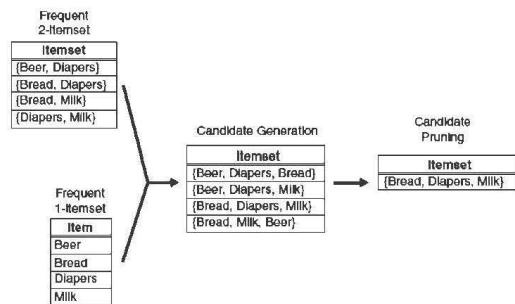


Figure 6.6. A brute-force method for generating candidate 3-itemsets.

Figure 6.7. Generating and pruning candidate k -itemsets by merging a frequent $(k - 1)$ -itemset with a frequent item. Note that some of the candidates are unnecessary because their subsets are infrequent.

This approach, however, does not prevent the same candidate itemset from being generated more than once. For instance, {Bread, Diapers, Milk} can be generated by merging {Bread, Diapers} with {Milk}, {Bread, Milk} with {Diapers}, or {Diapers, Milk} with {Bread}. One way to avoid generating

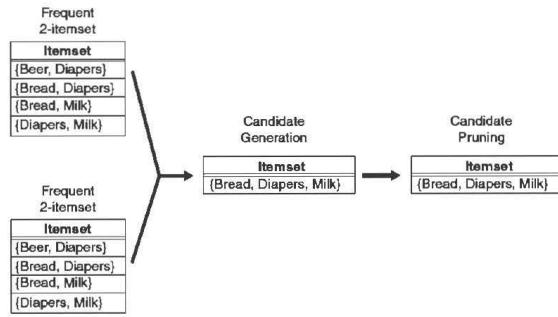
duplicate candidates is by ensuring that the items in each frequent itemset are kept sorted in their lexicographic order. Each frequent $(k - 1)$ -itemset X is then extended with frequent items that are lexicographically larger than the items in X . For example, the itemset {Bread, Diapers} can be augmented with {Milk} since Milk is lexicographically larger than Bread and Diapers. However, we should not augment {Diapers, Milk} with {Bread} nor {Bread, Milk} with {Diapers} because they violate the lexicographic ordering condition.

While this procedure is a substantial improvement over the brute-force method, it can still produce a large number of unnecessary candidates. For example, the candidate itemset obtained by merging {Beer, Diapers} with {Milk} is unnecessary because one of its subsets, {Beer, Milk}, is infrequent. There are several heuristics available to reduce the number of unnecessary candidates. For example, note that, for every candidate k -itemset that survives the pruning step, every item in the candidate must be contained in at least $k - 1$ of the frequent $(k - 1)$ -itemsets. Otherwise, the candidate is guaranteed to be infrequent. For example, {Beer, Diapers, Milk} is a viable candidate 3-itemset only if every item in the candidate, including Beer, is contained in at least two frequent 2-itemsets. Since there is only one frequent 2-itemset containing Beer, all candidate itemsets involving Beer must be infrequent.

$F_{k-1} \times F_{k-1}$ Method The candidate generation procedure in the apriori-gen function merges a pair of frequent $(k - 1)$ -itemsets only if their first $k - 2$ items are identical. Let $A = \{a_1, a_2, \dots, a_{k-1}\}$ and $B = \{b_1, b_2, \dots, b_{k-1}\}$ be a pair of frequent $(k - 1)$ -itemsets. A and B are merged if they satisfy the following conditions:

$$a_i = b_i \text{ (for } i = 1, 2, \dots, k - 2\text{) and } a_{k-1} \neq b_{k-1}.$$

In Figure 6.8, the frequent itemsets {Bread, Diapers} and {Bread, Milk} are merged to form a candidate 3-itemset {Bread, Diapers, Milk}. The algorithm does not have to merge {Beer, Diapers} with {Diapers, Milk} because the first item in both itemsets is different. Indeed, if {Beer, Diapers, Milk} is a viable candidate, it would have been obtained by merging {Beer, Diapers} with {Beer, Milk} instead. This example illustrates both the completeness of the candidate generation procedure and the advantages of using lexicographic ordering to prevent duplicate candidates. However, because each candidate is obtained by merging a pair of frequent $(k - 1)$ -itemsets, an additional candidate pruning step is needed to ensure that the remaining $k - 2$ subsets of the candidate are frequent.

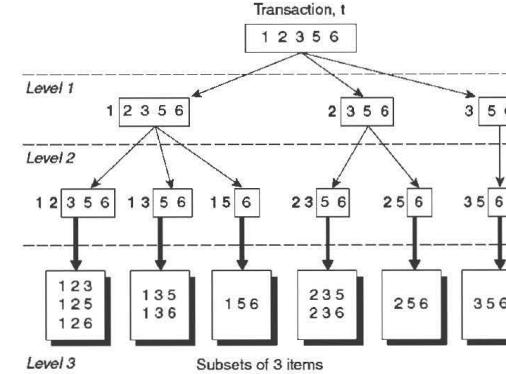
Figure 6.8. Generating and pruning candidate k -itemsets by merging pairs of frequent $(k-1)$ -itemsets.

6.2.4 Support Counting

Support counting is the process of determining the frequency of occurrence for every candidate itemset that survives the candidate pruning step of the apriori-gen function. Support counting is implemented in steps 6 through 11 of Algorithm 6.1. One approach for doing this is to compare each transaction against every candidate itemset (see Figure 6.2) and to update the support counts of candidates contained in the transaction. This approach is computationally expensive, especially when the numbers of transactions and candidate itemsets are large.

An alternative approach is to enumerate the itemsets contained in each transaction and use them to update the support counts of their respective candidate itemsets. To illustrate, consider a transaction t that contains five items, $\{1, 2, 3, 5, 6\}$. There are $\binom{5}{3} = 10$ itemsets of size 3 contained in this transaction. Some of the itemsets may correspond to the candidate 3-itemsets under investigation, in which case, their support counts are incremented. Other subsets of t that do not correspond to any candidates can be ignored.

Figure 6.9 shows a systematic way for enumerating the 3-itemsets contained in t . Assuming that each itemset keeps its items in increasing lexicographic order, an itemset can be enumerated by specifying the smallest item first, followed by the larger items. For instance, given $t = \{1, 2, 3, 5, 6\}$, all the 3-itemsets contained in t must begin with item 1, 2, or 3. It is not possible to construct a 3-itemset that begins with items 5 or 6 because there are only two

Figure 6.9. Enumerating subsets of three items from a transaction t .

items in t whose labels are greater than or equal to 5. The number of ways to specify the first item of a 3-itemset contained in t is illustrated by the Level 1 prefix structures depicted in Figure 6.9. For instance, $1 | 2 3 5 6$ represents a 3-itemset that begins with item 1, followed by two more items chosen from the set $\{2, 3, 5, 6\}$.

After fixing the first item, the prefix structures at Level 2 represent the number of ways to select the second item. For example, $1 | 2 | 3 5 6$ corresponds to itemsets that begin with prefix $\{1, 2\}$ and are followed by items 3, 5, or 6. Finally, the prefix structures at Level 3 represent the complete set of 3-itemsets contained in t . For example, the 3-itemsets that begin with prefix $\{1, 2\}$ are $\{1, 2, 3\}$, $\{1, 2, 5\}$, and $\{1, 2, 6\}$, while those that begin with prefix $\{2, 3\}$ are $\{2, 3, 5\}$ and $\{2, 3, 6\}$.

The prefix structures shown in Figure 6.9 demonstrate how itemsets contained in a transaction can be systematically enumerated, i.e., by specifying their items one by one, from the leftmost item to the rightmost item. We still have to determine whether each enumerated 3-itemset corresponds to an existing candidate itemset. If it matches one of the candidates, then the support count of the corresponding candidate is incremented. In the next section, we illustrate how this matching operation can be performed efficiently using a hash tree structure.

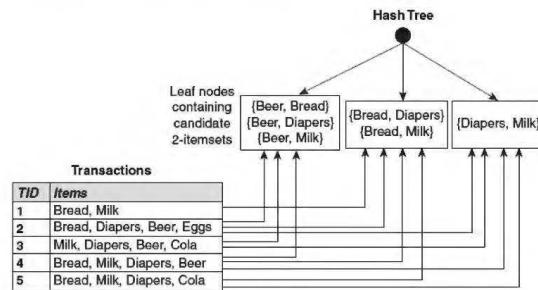


Figure 6.10. Counting the support of itemsets using hash structure.

Support Counting Using a Hash Tree

In the *Apriori* algorithm, candidate itemsets are partitioned into different buckets and stored in a hash tree. During support counting, itemsets contained in each transaction are also hashed into their appropriate buckets. That way, instead of comparing each itemset in the transaction with every candidate itemset, it is matched only against candidate itemsets that belong to the same bucket, as shown in Figure 6.10.

Figure 6.11 shows an example of a hash tree structure. Each internal node of the tree uses the following hash function, $h(p) = p \bmod 3$, to determine which branch of the current node should be followed next. For example, items 1, 4, and 7 are hashed to the same branch (i.e., the leftmost branch) because they have the same remainder after dividing the number by 3. All candidate itemsets are stored at the leaf nodes of the hash tree. The hash tree shown in Figure 6.11 contains 15 candidate 3-itemsets, distributed across 9 leaf nodes.

Consider a transaction, $t = \{1, 2, 3, 5, 6\}$. To update the support counts of the candidate itemsets, the hash tree must be traversed in such a way that all the leaf nodes containing candidate 3-itemsets belonging to t must be visited at least once. Recall that the 3-itemsets contained in t must begin with items 1, 2, or 3, as indicated by the Level 1 prefix structures shown in Figure 6.9. Therefore, at the root node of the hash tree, the items 1, 2, and 3 of the transaction are hashed separately. Item 1 is hashed to the left child of the root node, item 2 is hashed to the middle child, and item 3 is hashed to the right child. At the next level of the tree, the transaction is hashed on the second

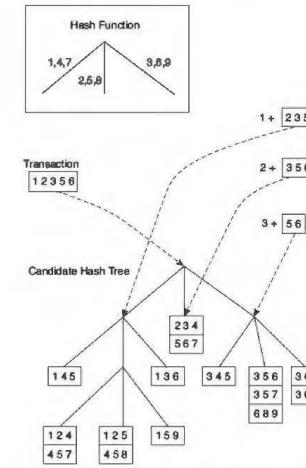


Figure 6.11. Hashing a transaction at the root node of a hash tree.

item listed in the Level 2 structures shown in Figure 6.9. For example, after hashing on item 1 at the root node, items 2, 3, and 5 of the transaction are hashed. Items 2 and 5 are hashed to the middle child, while item 3 is hashed to the right child, as shown in Figure 6.12. This process continues until the leaf nodes of the hash tree are reached. The candidate itemsets stored at the visited leaf nodes are compared against the transaction. If a candidate is a subset of the transaction, its support count is incremented. In this example, 5 out of the 9 leaf nodes are visited and 9 out of the 15 itemsets are compared against the transaction.

6.2.5 Computational Complexity

The computational complexity of the *Apriori* algorithm can be affected by the following factors.

Support Threshold Lowering the support threshold often results in more itemsets being declared as frequent. This has an adverse effect on the com-

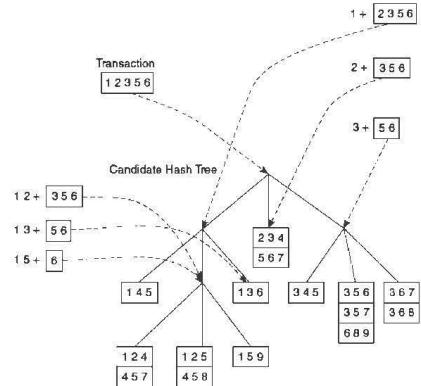


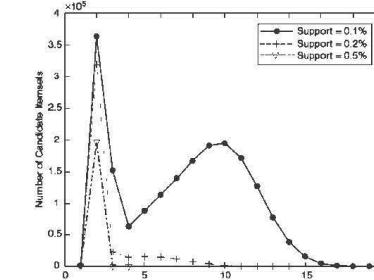
Figure 6.12. Subset operation on the leftmost subtree of the root of a candidate hash tree.

computational complexity of the algorithm because more candidate itemsets must be generated and counted, as shown in Figure 6.13. The maximum size of frequent itemsets also tends to increase with lower support thresholds. As the maximum size of the frequent itemsets increases, the algorithm will need to make more passes over the data set.

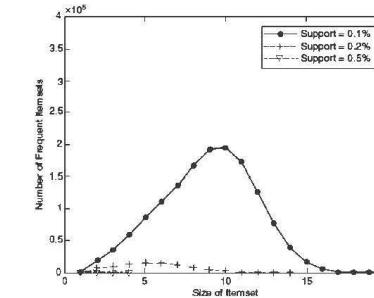
Number of Items (Dimensionality) As the number of items increases, more space will be needed to store the support counts of items. If the number of frequent items also grows with the dimensionality of the data, the computation and I/O costs will increase because of the larger number of candidate itemsets generated by the algorithm.

Number of Transactions Since the *Apriori* algorithm makes repeated passes over the data set, its run time increases with a larger number of transactions.

Average Transaction Width For dense data sets, the average transaction width can be very large. This affects the complexity of the *Apriori* algorithm in two ways. First, the maximum size of frequent itemsets tends to increase as the



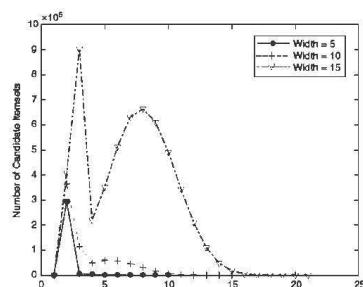
(a) Number of candidate itemsets.



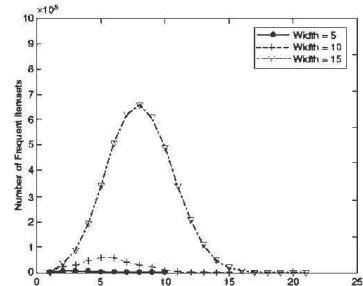
(b) Number of frequent itemsets.

Figure 6.13. Effect of support threshold on the number of candidate and frequent itemsets.

average transaction width increases. As a result, more candidate itemsets must be examined during candidate generation and support counting, as illustrated in Figure 6.14. Second, as the transaction width increases, more itemsets



(a) Number of candidate itemsets.



(b) Number of Frequent Itemsets.

Figure 6.14. Effect of average transaction width on the number of candidate and frequent itemsets.

are contained in the transaction. This will increase the number of hash tree traversals performed during support counting.

A detailed analysis of the time complexity for the *Apriori* algorithm is presented next.

Generation of frequent 1-itemsets For each transaction, we need to update the support count for every item present in the transaction. Assuming that w is the average transaction width, this operation requires $O(Nw)$ time, where N is the total number of transactions.

Candidate generation To generate candidate k -itemsets, pairs of frequent $(k-1)$ -itemsets are merged to determine whether they have at least $k-2$ items in common. Each merging operation requires at most $k-2$ equality comparisons. In the best-case scenario, every merging step produces a viable candidate k -itemset. In the worst-case scenario, the algorithm must merge every pair of frequent $(k-1)$ -itemsets found in the previous iteration. Therefore, the overall cost of merging frequent itemsets is

$$\sum_{k=2}^w (k-2)|C_k| < \text{Cost of merging} < \sum_{k=2}^w (k-2)|F_{k-1}|^2.$$

A hash tree is also constructed during candidate generation to store the candidate itemsets. Because the maximum depth of the tree is k , the cost for populating the hash tree with candidate itemsets is $O(\sum_{k=2}^w k|C_k|)$. During candidate pruning, we need to verify that the $k-2$ subsets of every candidate k -itemset are frequent. Since the cost for looking up a candidate in a hash tree is $O(k)$, the candidate pruning step requires $O(\sum_{k=2}^w k(k-2)|C_k|)$ time.

Support counting Each transaction of length $|t|$ produces $\binom{|t|}{k}$ itemsets of size k . This is also the effective number of hash tree traversals performed for each transaction. The cost for support counting is $O(N \sum_k \binom{w}{k} \alpha_k)$, where w is the maximum transaction width and α_k is the cost for updating the support count of a candidate k -itemset in the hash tree.

6.3 Rule Generation

This section describes how to extract association rules efficiently from a given frequent itemset. Each frequent k -itemset, Y , can produce up to $2^k - 2$ association rules, ignoring rules that have empty antecedents or consequents ($\emptyset \rightarrow Y$ or $Y \rightarrow \emptyset$). An association rule can be extracted by partitioning the itemset Y into two non-empty subsets, X and $Y - X$, such that $X \rightarrow Y - X$ satisfies the confidence threshold. Note that all such rules must have already met the support threshold because they are generated from a frequent itemset.

Example 6.2. Let $X = \{1, 2, 3\}$ be a frequent itemset. There are six candidate association rules that can be generated from X : $\{1, 2\} \rightarrow \{3\}$, $\{1, 3\} \rightarrow \{2\}$, $\{2, 3\} \rightarrow \{1\}$, $\{1\} \rightarrow \{2, 3\}$, $\{2\} \rightarrow \{1, 3\}$, and $\{3\} \rightarrow \{1, 2\}$. As each of their support is identical to the support for X , the rules must satisfy the support threshold. ■

Computing the confidence of an association rule does not require additional scans of the transaction data set. Consider the rule $\{1, 2\} \rightarrow \{3\}$, which is generated from the frequent itemset $X = \{1, 2, 3\}$. The confidence for this rule is $\sigma(\{1, 2, 3\})/\sigma(\{1, 2\})$. Because $\{1, 2, 3\}$ is frequent, the anti-monotone property of support ensures that $\{1, 2\}$ must be frequent, too. Since the support counts for both itemsets were already found during frequent itemset generation, there is no need to read the entire data set again.

6.3.1 Confidence-Based Pruning

Unlike the support measure, confidence does not have any monotone property. For example, the confidence for $X \rightarrow Y$ can be larger, smaller, or equal to the confidence for another rule $\tilde{X} \rightarrow \tilde{Y}$, where $\tilde{X} \subseteq X$ and $\tilde{Y} \subseteq Y$ (see Exercise 3 on page 405). Nevertheless, if we compare rules generated from the same frequent itemset Y , the following theorem holds for the confidence measure.

Theorem 6.2. If a rule $X \rightarrow Y - X$ does not satisfy the confidence threshold, then any rule $X' \rightarrow Y - X'$, where X' is a subset of X , must not satisfy the confidence threshold as well.

To prove this theorem, consider the following two rules: $X' \rightarrow Y - X'$ and $X \rightarrow Y - X$, where $X' \subset X$. The confidence of the rules are $\sigma(Y)/\sigma(X')$ and $\sigma(Y)/\sigma(X)$, respectively. Since X' is a subset of X , $\sigma(X') \geq \sigma(X)$. Therefore, the former rule cannot have a higher confidence than the latter rule.

6.3.2 Rule Generation in *Apriori* Algorithm

The *Apriori* algorithm uses a level-wise approach for generating association rules, where each level corresponds to the number of items that belong to the rule consequent. Initially, all the high-confidence rules that have only one item in the rule consequent are extracted. These rules are then used to generate new candidate rules. For example, if $\{acd\} \rightarrow \{b\}$ and $\{abd\} \rightarrow \{c\}$ are high-confidence rules, then the candidate rule $\{ad\} \rightarrow \{bc\}$ is generated by merging the consequents of both rules. Figure 6.15 shows a lattice structure for the association rules generated from the frequent itemset $\{a, b, c, d\}$. If any node in the lattice has low confidence, then according to Theorem 6.2, the

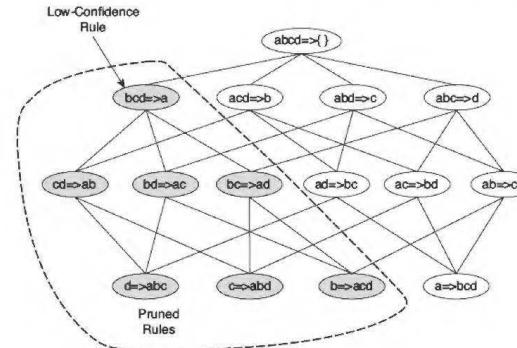


Figure 6.15. Pruning of association rules using the confidence measure.

entire subgraph spanned by the node can be pruned immediately. Suppose the confidence for $\{bcd\} \rightarrow \{a\}$ is low. All the rules containing item a in its consequent, including $\{cd\} \rightarrow \{ab\}$, $\{bd\} \rightarrow \{ac\}$, $\{bc\} \rightarrow \{ad\}$, and $\{d\} \rightarrow \{abc\}$ can be discarded.

A pseudocode for the rule generation step is shown in Algorithms 6.2 and 6.3. Note the similarity between the `ap-genrules` procedure given in Algorithm 6.3 and the frequent itemset generation procedure given in Algorithm 6.1. The only difference is that, in rule generation, we do not have to make additional passes over the data set to compute the confidence of the candidate rules. Instead, we determine the confidence of each rule by using the support counts computed during frequent itemset generation.

Algorithm 6.2 Rule generation of the *Apriori* algorithm.

```

1: for each frequent  $k$ -itemset  $f_k$ ,  $k \geq 2$  do
2:    $H_1 = \{i \mid i \in f_k\}$  {1-item consequents of the rule.}
3:   call ap-genrules( $f_k, H_1$ )
4: end for

```

Algorithm 6.3 Procedure ap-genrules(f_k, H_m).

```

1:  $k = |f_k|$  {size of frequent itemset.}
2:  $m = |H_m|$  {size of rule consequent.}
3: If  $k > m + 1$  then
4:    $H_{m+1} = \text{apriori-gen}(H_m)$ .
5:   for each  $h_{m+1} \in H_{m+1}$  do
6:      $\text{conf} = \sigma(f_k)/\sigma(f_k - h_{m+1})$ .
7:     if  $\text{conf} \geq \text{minconf}$  then
8:       output the rule  $(f_k - h_{m+1}) \rightarrow h_{m+1}$ .
9:     else
10:      delete  $h_{m+1}$  from  $H_{m+1}$ .
11:    end if
12:  end for
13:  call ap-genrules( $f_k, H_{m+1}$ )
14: end if

```

6.3.3 An Example: Congressional Voting Records

This section demonstrates the results of applying association analysis to the voting records of members of the United States House of Representatives. The data is obtained from the 1984 Congressional Voting Records Database, which is available at the UCI machine learning data repository. Each transaction contains information about the party affiliation for a representative along with his or her voting record on 16 key issues. There are 435 transactions and 34 items in the data set. The set of items are listed in Table 6.3.

The *Apriori* algorithm is then applied to the data set with $\text{minsup} = 30\%$ and $\text{minconf} = 90\%$. Some of the high-confidence rules extracted by the algorithm are shown in Table 6.4. The first two rules suggest that most of the members who voted yes for aid to El Salvador and no for budget resolution and MX missile are Republicans; while those who voted no for aid to El Salvador and yes for budget resolution and MX missile are Democrats. These high-confidence rules show the key issues that divide members from both political parties. If minconf is reduced, we may find rules that contain issues that cut across the party lines. For example, with $\text{minconf} = 40\%$, the rules suggest that corporation cutbacks is an issue that receives almost equal number of votes from both parties—52.3% of the members who voted no are Republicans, while the remaining 47.7% of them who voted no are Democrats.

Table 6.3. List of binary attributes from the 1984 United States Congressional Voting Records. Source: The UCI machine learning repository.

1. Republican	18. aid to Nicaragua = no
2. Democrat	19. MX-missile = yes
3. handicapped-infants = yes	20. MX-missile = no
4. handicapped-infants = no	21. immigration = yes
5. water project cost sharing = yes	22. immigration = no
6. water project cost sharing = no	23. synfuel corporation cutback = yes
7. budget-resolution = yes	24. synfuel corporation cutback = no
8. budget-resolution = no	25. education spending = yes
9. physician fee freeze = yes	26. education spending = no
10. physician fee freeze = no	27. right-to-sue = yes
11. aid to El Salvador = yes	28. right-to-sue = no
12. aid to El Salvador = no	29. crime = yes
13. religious groups in schools = yes	30. crime = no
14. religious groups in schools = no	31. duty-free-exports = yes
15. anti-satellite test ban = yes	32. duty-free-exports = no
16. anti-satellite test ban = no	33. export administration act = yes
17. aid to Nicaragua = yes	34. export administration act = no

Table 6.4. Association rules extracted from the 1984 United States Congressional Voting Records.

Association Rule	Confidence
{budget resolution = no, MX-missile=no, aid to El Salvador = yes } → {Republican}	91.0%
{budget resolution = yes, MX-missile=yes, aid to El Salvador = no } → {Democrat}	97.5%
{crime = yes, right-to-sue = yes, physician fee freeze = yes} → {Republican}	93.5%
{crime = no, right-to-sue = no, physician fee freeze = no} → {Democrat}	100%

6.4 Compact Representation of Frequent Itemsets

In practice, the number of frequent itemsets produced from a transaction data set can be very large. It is useful to identify a small representative set of itemsets from which all other frequent itemsets can be derived. Two such representations are presented in this section in the form of maximal and closed frequent itemsets.

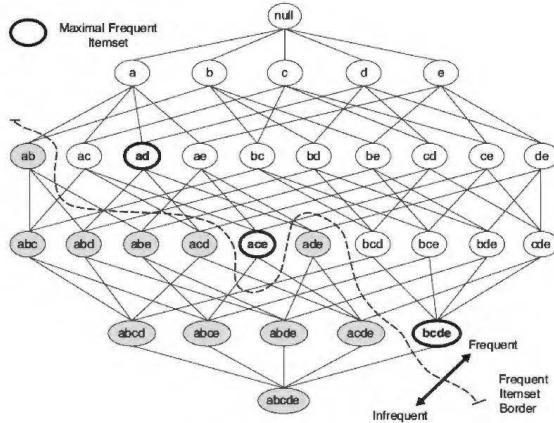


Figure 6.16. Maximal frequent itemset.

6.4.1 Maximal Frequent Itemsets

Definition 6.3 (Maximal Frequent Itemset). A maximal frequent itemset is defined as a frequent itemset for which none of its immediate supersets are frequent.

To illustrate this concept, consider the itemset lattice shown in Figure 6.16. The itemsets in the lattice are divided into two groups: those that are frequent and those that are infrequent. A frequent itemset border, which is represented by a dashed line, is also illustrated in the diagram. Every itemset located above the border is frequent, while those located below the border (the shaded nodes) are infrequent. Among the itemsets residing near the border, $\{a, d\}$, $\{a, c, e\}$, and $\{b, c, d, e\}$ are considered to be maximal frequent itemsets because their immediate supersets are infrequent. An itemset such as $\{a, d\}$ is maximal frequent because all of its immediate supersets, $\{a, b, d\}$, $\{a, c, d\}$, and $\{a, d, e\}$, are infrequent. In contrast, $\{a, c\}$ is non-maximal because one of its immediate supersets, $\{a, c, e\}$, is frequent.

Maximal frequent itemsets effectively provide a compact representation of frequent itemsets. In other words, they form the smallest set of itemsets from

which all frequent itemsets can be derived. For example, the frequent itemsets shown in Figure 6.16 can be divided into two groups:

- Frequent itemsets that begin with item a and that may contain items c , d , or e . This group includes itemsets such as $\{a\}$, $\{a, c\}$, $\{a, d\}$, $\{a, e\}$, and $\{a, c, e\}$.
- Frequent itemsets that begin with items b , c , d , or e . This group includes itemsets such as $\{b\}$, $\{b, c\}$, $\{c, d\}$, $\{b, c, d, e\}$, etc.

Frequent itemsets that belong in the first group are subsets of either $\{a, c, e\}$ or $\{a, d\}$, while those that belong in the second group are subsets of $\{b, c, d, e\}$. Hence, the maximal frequent itemsets $\{a, c, e\}$, $\{a, d\}$, and $\{b, c, d, e\}$ provide a compact representation of the frequent itemsets shown in Figure 6.16.

Maximal frequent itemsets provide a valuable representation for data sets that can produce very long, frequent itemsets, as there are exponentially many frequent itemsets in such data. Nevertheless, this approach is practical only if an efficient algorithm exists to explicitly find the maximal frequent itemsets without having to enumerate all their subsets. We briefly describe one such approach in Section 6.5.

Despite providing a compact representation, maximal frequent itemsets do not contain the support information of their subsets. For example, the support of the maximal frequent itemsets $\{a, c, e\}$, $\{a, d\}$, and $\{b, c, d, e\}$ do not provide any hint about the support of their subsets. An additional pass over the data set is therefore needed to determine the support counts of the non-maximal frequent itemsets. In some cases, it might be desirable to have a minimal representation of frequent itemsets that preserves the support information. We illustrate such a representation in the next section.

6.4.2 Closed Frequent Itemsets

Closed itemsets provide a minimal representation of itemsets without losing their support information. A formal definition of a closed itemset is presented below.

Definition 6.4 (Closed Itemset). An itemset X is closed if none of its immediate supersets has exactly the same support count as X .

Put another way, X is not closed if at least one of its immediate supersets has the same support count as X . Examples of closed itemsets are shown in Figure 6.17. To better illustrate the support count of each itemset, we have associated each node (itemset) in the lattice with a list of its corresponding

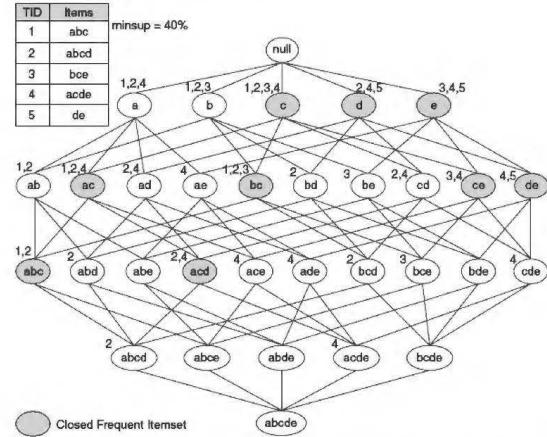


Figure 6.17. An example of the closed frequent itemsets (with minimum support count equal to 40%).

transaction IDs. For example, since the node $\{b, c\}$ is associated with transaction IDs 1, 2, and 3, its support count is equal to three. From the transactions given in this diagram, notice that every transaction that contains b also contains c . Consequently, the support for $\{b\}$ is identical to $\{b, c\}$ and $\{b\}$ should not be considered a closed itemset. Similarly, since c occurs in every transaction that contains both a and d , the itemset $\{a, d\}$ is not closed. On the other hand, $\{b, c\}$ is a closed itemset because it does not have the same support count as any of its supersets.

Definition 6.5 (Closed Frequent Itemset). An itemset is a closed frequent itemset if it is closed and its support is greater than or equal to $minsup$.

In the previous example, assuming that the support threshold is 40%, $\{b, c\}$ is a closed frequent itemset because its support is 60%. The rest of the closed frequent itemsets are indicated by the shaded nodes.

Algorithms are available to explicitly extract closed frequent itemsets from a given data set. Interested readers may refer to the bibliographic notes at the end of this chapter for further discussions of these algorithms. We can use the closed frequent itemsets to determine the support counts for the non-closed

Algorithm 6.4 Support counting using closed frequent itemsets.

```

1: Let  $C$  denote the set of closed frequent itemsets
2: Let  $k_{max}$  denote the maximum size of closed frequent itemsets
3:  $F_{k_{max}} = \{f | f \in C, |f| = k_{max}\}$  {Find all frequent itemsets of size  $k_{max}$ }
4: for  $k = k_{max} - 1$  downto 1 do
5:    $F_k = \{f | f \subset F_{k+1}, |f| = k\}$  {Find all frequent itemsets of size  $k$ }
6:   for each  $f \in F_k$  do
7:     if  $f \notin C$  then
8:        $f.support = \max\{f'.support | f' \in F_{k+1}, f \subset f'\}$ 
9:     end if
10:   end for
11: end for

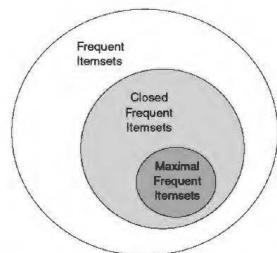
```

frequent itemsets. For example, consider the frequent itemset $\{a, d\}$ shown in Figure 6.17. Because the itemset is not closed, its support count must be identical to one of its immediate supersets. The key is to determine which superset (among $\{a, b, d\}$, $\{a, c, d\}$, or $\{a, d, e\}$) has exactly the same support count as $\{a, d\}$. The *Apriori* principle states that any transaction that contains the superset of $\{a, d\}$ must also contain $\{a, d\}$. However, any transaction that contains $\{a, d\}$ does not have to contain the supersets of $\{a, d\}$. For this reason, the support for $\{a, d\}$ must be equal to the largest support among its supersets. Since $\{a, c, d\}$ has a larger support than both $\{a, b, d\}$ and $\{a, d, e\}$, the support for $\{a, d\}$ must be identical to the support for $\{a, c, d\}$. Using this methodology, an algorithm can be developed to compute the support for the non-closed frequent itemsets. The pseudocode for this algorithm is shown in Algorithm 6.4. The algorithm proceeds in a specific-to-general fashion, i.e., from the largest to the smallest frequent itemsets. This is because, in order to find the support for a non-closed frequent itemset, the support for all of its supersets must be known.

To illustrate the advantage of using closed frequent itemsets, consider the data set shown in Table 6.5, which contains ten transactions and fifteen items. The items can be divided into three groups: (1) Group A , which contains items a_1 through a_5 ; (2) Group B , which contains items b_1 through b_5 ; and (3) Group C , which contains items c_1 through c_5 . Note that items within each group are perfectly associated with each other and they do not appear with items from another group. Assuming the support threshold is 20%, the total number of frequent itemsets is $3 \times (2^5 - 1) = 93$. However, there are only three closed frequent itemsets in the data: $\{\{a_1, a_2, a_3, a_4, a_5\}, \{b_1, b_2, b_3, b_4, b_5\}, \text{ and } \{c_1, c_2, c_3, c_4, c_5\}\}$. It is often sufficient to present only the closed frequent itemsets to the analysts instead of the entire set of frequent itemsets.

Table 6.5. A transaction data set for mining closed itemsets.

TID	a_1	a_2	a_3	a_4	a_5	b_1	b_2	b_3	b_4	b_5	c_1	c_2	c_3	c_4	c_5
1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
2	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
3	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0
5	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0
6	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
8	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
9	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
10	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1

**Figure 6.18.** Relationships among frequent, maximal frequent, and closed frequent itemsets.

Closed frequent itemsets are useful for removing some of the redundant association rules. An association rule $X \rightarrow Y$ is redundant if there exists another rule $X' \rightarrow Y'$, where X is a subset of X' and Y is a subset of Y' , such that the support and confidence for both rules are identical. In the example shown in Figure 6.17, $\{b\}$ is not a closed frequent itemset while $\{b, c\}$ is closed. The association rule $\{b\} \rightarrow \{d, e\}$ is therefore redundant because it has the same support and confidence as $\{b, c\} \rightarrow \{d, e\}$. Such redundant rules are not generated if closed frequent itemsets are used for rule generation.

Finally, note that all maximal frequent itemsets are closed because none of the maximal frequent itemsets can have the same support count as their immediate supersets. The relationships among frequent, maximal frequent, and closed frequent itemsets are shown in Figure 6.18.

6.5 Alternative Methods for Generating Frequent Itemsets

Apriori is one of the earliest algorithms to have successfully addressed the combinatorial explosion of frequent itemset generation. It achieves this by applying the *Apriori* principle to prune the exponential search space. Despite its significant performance improvement, the algorithm still incurs considerable I/O overhead since it requires making several passes over the transaction data set. In addition, as noted in Section 6.2.5, the performance of the *Apriori* algorithm may degrade significantly for dense data sets because of the increasing width of transactions. Several alternative methods have been developed to overcome these limitations and improve upon the efficiency of the *Apriori* algorithm. The following is a high-level description of these methods.

Traversal of Itemset Lattice A search for frequent itemsets can be conceptually viewed as a traversal on the itemset lattice shown in Figure 6.1. The search strategy employed by an algorithm dictates how the lattice structure is traversed during the frequent itemset generation process. Some search strategies are better than others, depending on the configuration of frequent itemsets in the lattice. An overview of these strategies is presented next.

- **General-to-Specific versus Specific-to-General:** The *Apriori* algorithm uses a general-to-specific search strategy, where pairs of frequent $(k-1)$ -itemsets are merged to obtain candidate k -itemsets. This general-to-specific search strategy is effective, provided the maximum length of a frequent itemset is not too long. The configuration of frequent itemsets that works best with this strategy is shown in Figure 6.19(a), where the darker nodes represent infrequent itemsets. Alternatively, a specific-to-general search strategy looks for more specific frequent itemsets first, before finding the more general frequent itemsets. This strategy is useful to discover maximal frequent itemsets in dense transactions, where the frequent itemset border is located near the bottom of the lattice, as shown in Figure 6.19(b). The *Apriori* principle can be applied to prune all subsets of maximal frequent itemsets. Specifically, if a candidate k -itemset is maximal frequent, we do not have to examine any of its subsets of size $k-1$. However, if the candidate k -itemset is infrequent, we need to check all of its $k-1$ subsets in the next iteration. Another approach is to combine both general-to-specific and specific-to-general search strategies. This bidirectional approach requires more space to

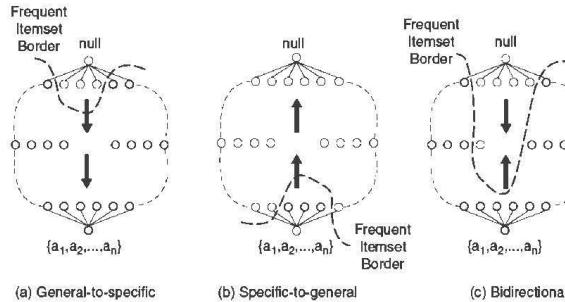


Figure 6.19. General-to-specific, specific-to-general, and bidirectional search.

store the candidate itemsets, but it can help to rapidly identify the frequent itemset border, given the configuration shown in Figure 6.19(c).

- Equivalence Classes:** Another way to envision the traversal is to first partition the lattice into disjoint groups of nodes (or equivalence classes). A frequent itemset generation algorithm searches for frequent itemsets within a particular equivalence class first before moving to another equivalence class. As an example, the level-wise strategy used in the *Apriori* algorithm can be considered to be partitioning the lattice on the basis of itemset sizes; i.e., the algorithm discovers all frequent 1-itemsets first before proceeding to larger-sized itemsets. Equivalence classes can also be defined according to the prefix or suffix labels of an itemset. In this case, two itemsets belong to the same equivalence class if they share a common prefix or suffix of length k . In the prefix-based approach, the algorithm can search for frequent itemsets starting with the prefix a before looking for those starting with prefixes b , c , and so on. Both prefix-based and suffix-based equivalence classes can be demonstrated using the tree-like structure shown in Figure 6.20.
- Breadth-First versus Depth-First:** The *Apriori* algorithm traverses the lattice in a breadth-first manner, as shown in Figure 6.21(a). It first discovers all the frequent 1-itemsets, followed by the frequent 2-itemsets, and so on, until no new frequent itemsets are generated. The itemset

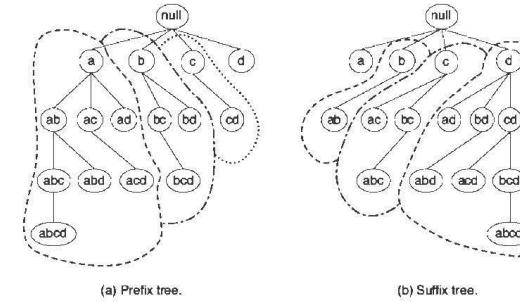


Figure 6.20. Equivalence classes based on the prefix and suffix labels of itemsets.

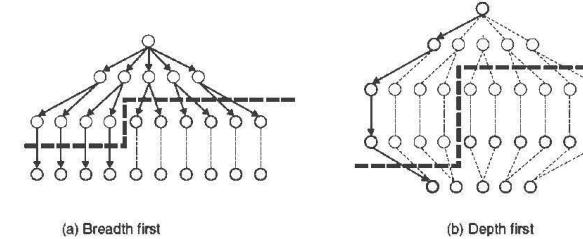


Figure 6.21. Breadth-first and depth-first traversals.

lattice can also be traversed in a depth-first manner, as shown in Figures 6.21(b) and 6.22. The algorithm can start from, say, node a in Figure 6.22, and count its support to determine whether it is frequent. If so, the algorithm progressively expands the next level of nodes, i.e., ab , abc , and so on, until an infrequent node is reached, say, $abcd$. It then backtracks to another branch, say, $abce$, and continues the search from there.

The depth-first approach is often used by algorithms designed to find maximal frequent itemsets. This approach allows the frequent itemset border to be detected more quickly than using a breadth-first approach. Once a maximal frequent itemset is found, substantial pruning can be

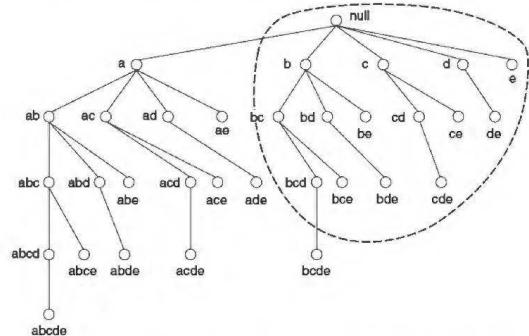


Figure 6.22. Generating candidate itemsets using the depth-first approach.

performed on its subsets. For example, if the node *bcd e* shown in Figure 6.22 is maximal frequent, then the algorithm does not have to visit the subtrees rooted at *bd*, *be*, *c*, *d*, and *e* because they will not contain any maximal frequent itemsets. However, if *abc* is maximal frequent, only the nodes such as *ac* and *bc* are not maximal frequent (but the subtrees of *ac* and *bc* may still contain maximal frequent itemsets). The depth-first approach also allows a different kind of pruning based on the support of itemsets. For example, suppose the support for $\{a, b, c\}$ is identical to the support for $\{a, b\}$. The subtrees rooted at *abd* and *abe* can be skipped because they are guaranteed not to have any maximal frequent itemsets. The proof of this is left as an exercise to the readers.

Representation of Transaction Data Set There are many ways to represent a transaction data set. The choice of representation can affect the I/O costs incurred when computing the support of candidate itemsets. Figure 6.23 shows two different ways of representing market basket transactions. The representation on the left is called a **horizontal data layout**, which is adopted by many association rule mining algorithms, including *Apriori*. Another possibility is to store the list of transaction identifiers (TID-list) associated with each item. Such a representation is known as the **vertical data layout**. The support for each candidate itemset is obtained by intersecting the TID-lists of its subset items. The length of the TID-lists shrinks as we progress to larger

Horizontal Data Layout		Vertical Data Layout				
TID	Items	a	b	c	d	e
1	a,b,e	1	1	2	2	1
2	b,c,d	4	2	3	4	3
3	c,e	5	5	4	5	6
4	a,c,d	6	7	8	9	
5	a,b,c,d	7	8	9		
6	a,e	8	10			
7	a,b					
8	a,b,c					
9	a,c,d					
10	b					

Figure 6.23. Horizontal and vertical data format.

sized itemsets. However, one problem with this approach is that the initial set of TID-lists may be too large to fit into main memory, thus requiring more sophisticated techniques to compress the TID-lists. We describe another effective approach to represent the data in the next section.

6.6 FP-Growth Algorithm

This section presents an alternative algorithm called **FP-growth** that takes a radically different approach to discovering frequent itemsets. The algorithm does not subscribe to the generate-and-test paradigm of *Apriori*. Instead, it encodes the data set using a compact data structure called an **FP-tree** and extracts frequent itemsets directly from this structure. The details of this approach are presented next.

6.6.1 FP-Tree Representation

An FP-tree is a compressed representation of the input data. It is constructed by reading the data set one transaction at a time and mapping each transaction onto a path in the FP-tree. As different transactions can have several items in common, their paths may overlap. The more the paths overlap with one another, the more compression we can achieve using the FP-tree structure. If the size of the FP-tree is small enough to fit into main memory, this will allow us to extract frequent itemsets directly from the structure in memory instead of making repeated passes over the data stored on disk.

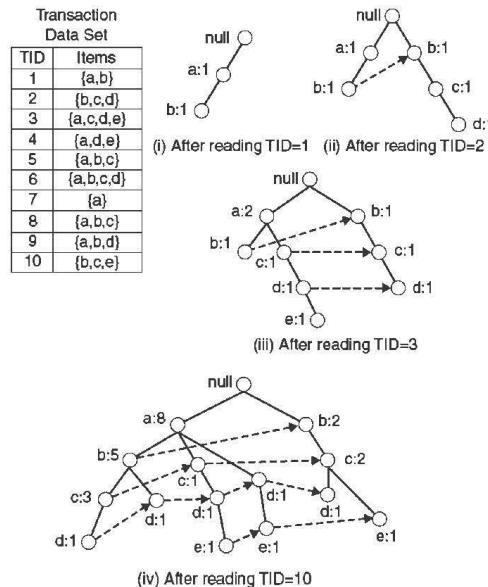


Figure 6.24. Construction of an FP-tree.

Figure 6.24 shows a data set that contains ten transactions and five items. The structures of the FP-tree after reading the first three transactions are also depicted in the diagram. Each node in the tree contains the label of an item along with a counter that shows the number of transactions mapped onto the given path. Initially, the FP-tree contains only the root node represented by the null symbol. The FP-tree is subsequently extended in the following way:

1. The data set is scanned once to determine the support count of each item. Infrequent items are discarded, while the frequent items are sorted in decreasing support counts. For the data set shown in Figure 6.24, *a* is the most frequent item, followed by *b*, *c*, *d*, and *e*.

2. The algorithm makes a second pass over the data to construct the FP-tree. After reading the first transaction, $\{a, b\}$, the nodes labeled as *a* and *b* are created. A path is then formed from $\text{null} \rightarrow a \rightarrow b$ to encode the transaction. Every node along the path has a frequency count of 1.
3. After reading the second transaction, $\{b, c, d\}$, a new set of nodes is created for items *b*, *c*, and *d*. A path is then formed to represent the transaction by connecting the nodes $\text{null} \rightarrow b \rightarrow c \rightarrow d$. Every node along this path also has a frequency count equal to one. Although the first two transactions have an item in common, which is *b*, their paths are disjoint because the transactions do not share a common prefix.
4. The third transaction, $\{a, c, d, e\}$, shares a common prefix item (which is *a*) with the first transaction. As a result, the path for the third transaction, $\text{null} \rightarrow a \rightarrow c \rightarrow d \rightarrow e$, overlaps with the path for the first transaction, $\text{null} \rightarrow a \rightarrow b$. Because of their overlapping path, the frequency count for node *a* is incremented to two, while the frequency counts for the newly created nodes, *c*, *d*, and *e*, are equal to one.
5. This process continues until every transaction has been mapped onto one of the paths given in the FP-tree. The resulting FP-tree after reading all the transactions is shown at the bottom of Figure 6.24.

The size of an FP-tree is typically smaller than the size of the uncompressed data because many transactions in market basket data often share a few items in common. In the best-case scenario, where all the transactions have the same set of items, the FP-tree contains only a single branch of nodes. The worst-case scenario happens when every transaction has a unique set of items. As none of the transactions have any items in common, the size of the FP-tree is effectively the same as the size of the original data. However, the physical storage requirement for the FP-tree is higher because it requires additional space to store pointers between nodes and counters for each item.

The size of an FP-tree also depends on how the items are ordered. If the ordering scheme in the preceding example is reversed, i.e., from lowest to highest support item, the resulting FP-tree is shown in Figure 6.25. The tree appears to be denser because the branching factor at the root node has increased from 2 to 5 and the number of nodes containing the high support items such as *a* and *b* has increased from 3 to 12. Nevertheless, ordering by decreasing support counts does not always lead to the smallest tree. For example, suppose we augment the data set given in Figure 6.24 with 100 transactions that contain $\{e\}$, 80 transactions that contain $\{d\}$, 60 transactions

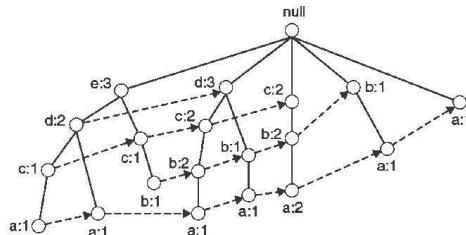


Figure 6.25. An FP-tree representation for the data set shown in Figure 6.24 with a different item ordering scheme.

that contain $\{c\}$, and 40 transactions that contain $\{b\}$. Item e is now most frequent, followed by d , c , b , and a . With the augmented transactions, ordering by decreasing support counts will result in an FP-tree similar to Figure 6.25, while a scheme based on increasing support counts produces a smaller FP-tree similar to Figure 6.24(iv).

An FP-tree also contains a list of pointers connecting between nodes that have the same items. These pointers, represented as dashed lines in Figures 6.24 and 6.25, help to facilitate the rapid access of individual items in the tree. We explain how to use the FP-tree and its corresponding pointers for frequent itemset generation in the next section.

6.6.2 Frequent Itemset Generation in FP-Growth Algorithm

FP-growth is an algorithm that generates frequent itemsets from an FP-tree by exploring the tree in a bottom-up fashion. Given the example tree shown in Figure 6.24, the algorithm looks for frequent itemsets ending in e first, followed by d , c , b , and finally a . This bottom-up strategy for finding frequent itemsets ending with a particular item is equivalent to the suffix-based approach described in Section 6.5. Since every transaction is mapped onto a path in the FP-tree, we can derive the frequent itemsets ending with a particular item, say, e , by examining only the paths containing node e . These paths can be accessed rapidly using the pointers associated with node e . The extracted paths are shown in Figure 6.26(a). The details on how to process the paths to obtain frequent itemsets will be explained later.

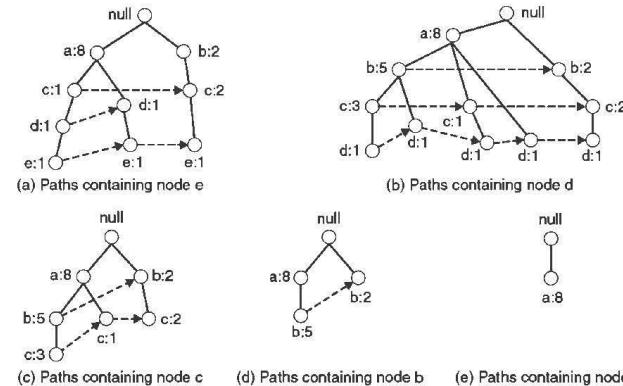


Figure 6.26. Decomposing the frequent itemset generation problem into multiple subproblems, where each subproblem involves finding frequent itemsets ending in e , d , c , b , and a .

Table 6.6. The list of frequent itemsets ordered by their corresponding suffixes.

Suffix	Frequent Itemsets
e	$\{e\}$, $\{d,e\}$, $\{a,d,e\}$, $\{c,e\}$, $\{a,e\}$
d	$\{d\}$, $\{c,d\}$, $\{b,c,d\}$, $\{a,c,d\}$, $\{b,d\}$, $\{a,b,d\}$, $\{a,d\}$
c	$\{c\}$, $\{b,c\}$, $\{a,b,c\}$, $\{a,c\}$
b	$\{b\}$, $\{a,b\}$
a	$\{a\}$

After finding the frequent itemsets ending in e , the algorithm proceeds to look for frequent itemsets ending in d by processing the paths associated with node d . The corresponding paths are shown in Figure 6.26(b). This process continues until all the paths associated with nodes c , b , and finally a , are processed. The paths for these items are shown in Figures 6.26(c), (d), and (e), while their corresponding frequent itemsets are summarized in Table 6.6.

FP-growth finds all the frequent itemsets ending with a particular suffix by employing a divide-and-conquer strategy to split the problem into smaller subproblems. For example, suppose we are interested in finding all frequent

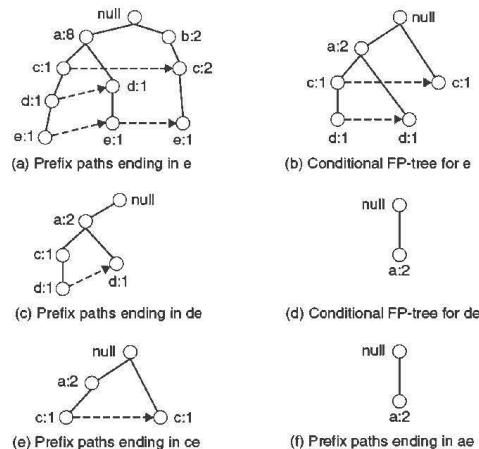


Figure 6.27. Example of applying the FP-growth algorithm to find frequent itemsets ending in e .

itemsets ending in e . To do this, we must first check whether the itemset $\{e\}$ itself is frequent. If it is frequent, we consider the subproblem of finding frequent itemsets ending in de , followed by ce , be , and ae . In turn, each of these subproblems are further decomposed into smaller subproblems. By merging the solutions obtained from the subproblems, all the frequent itemsets ending in e can be found. This divide-and-conquer approach is the key strategy employed by the FP-growth algorithm.

For a more concrete example on how to solve the subproblems, consider the task of finding frequent itemsets ending with e .

1. The first step is to gather all the paths containing node e . These initial paths are called prefix paths and are shown in Figure 6.27(a).
2. From the prefix paths shown in Figure 6.27(a), the support count for e is obtained by adding the support counts associated with node e . Assuming that the minimum support count is 2, $\{e\}$ is declared a frequent itemset because its support count is 3.

3. Because $\{e\}$ is frequent, the algorithm has to solve the subproblems of finding frequent itemsets ending in de , ce , be , and ae . Before solving these subproblems, it must first convert the prefix paths into a conditional FP-tree, which is structurally similar to an FP-tree, except it is used to find frequent itemsets ending with a particular suffix. A conditional FP-tree is obtained in the following way:

- (a) First, the support counts along the prefix paths must be updated because some of the counts include transactions that do not contain item e . For example, the rightmost path shown in Figure 6.27(a), $null \rightarrow b:2 \rightarrow c:2 \rightarrow e:1$, includes a transaction $\{b, c\}$ that does not contain item e . The counts along the prefix path must therefore be adjusted to 1 to reflect the actual number of transactions containing $\{b, c, e\}$.
- (b) The prefix paths are truncated by removing the nodes for e . These nodes can be removed because the support counts along the prefix paths have been updated to reflect only transactions that contain e and the subproblems of finding frequent itemsets ending in de , ce , be , and ae no longer need information about node e .
- (c) After updating the support counts along the prefix paths, some of the items may no longer be frequent. For example, the node b appears only once and has a support count equal to 1, which means that there is only one transaction that contains both b and e . Item b can be safely ignored from subsequent analysis because all itemsets ending in be must be infrequent.

The conditional FP-tree for e is shown in Figure 6.27(b). The tree looks different than the original prefix paths because the frequency counts have been updated and the nodes b and c have been eliminated.

4. FP-growth uses the conditional FP-tree for e to solve the subproblems of finding frequent itemsets ending in de , ce , and ae . To find the frequent itemsets ending in de , the prefix paths for d are gathered from the conditional FP-tree for e (Figure 6.27(c)). By adding the frequency counts associated with node d , we obtain the support count for $\{d, e\}$. Since the support count is equal to 2, $\{d, e\}$ is declared a frequent itemset. Next, the algorithm constructs the conditional FP-tree for de using the approach described in step 3. After updating the support counts and removing the infrequent item c , the conditional FP-tree for de is shown in Figure 6.27(d). Since the conditional FP-tree contains only one item,

a, whose support is equal to *minsup*, the algorithm extracts the frequent itemset $\{a, d, e\}$ and moves on to the next subproblem, which is to generate frequent itemsets ending in *ce*. After processing the prefix paths for *c*, only $\{c, e\}$ is found to be frequent. The algorithm proceeds to solve the next subproblem and found $\{a, e\}$ to be the only frequent itemset remaining.

This example illustrates the divide-and-conquer approach used in the FP-growth algorithm. At each recursive step, a conditional FP-tree is constructed by updating the frequency counts along the prefix paths and removing all infrequent items. Because the subproblems are disjoint, FP-growth will not generate any duplicate itemsets. In addition, the counts associated with the nodes allow the algorithm to perform support counting while generating the common suffix itemsets.

FP-growth is an interesting algorithm because it illustrates how a compact representation of the transaction data set helps to efficiently generate frequent itemsets. In addition, for certain transaction data sets, FP-growth outperforms the standard *Apriori* algorithm by several orders of magnitude. The run-time performance of FP-growth depends on the compaction factor of the data set. If the resulting conditional FP-trees are very bushy (in the worst case, a full prefix tree), then the performance of the algorithm degrades significantly because it has to generate a large number of subproblems and merge the results returned by each subproblem.

6.7 Evaluation of Association Patterns

Association analysis algorithms have the potential to generate a large number of patterns. For example, although the data set shown in Table 6.1 contains only six items, it can produce up to hundreds of association rules at certain support and confidence thresholds. As the size and dimensionality of real commercial databases can be very large, we could easily end up with thousands or even millions of patterns, many of which might not be interesting. Sifting through the patterns to identify the most interesting ones is not a trivial task because “one person’s trash might be another person’s treasure.” It is therefore important to establish a set of well-accepted criteria for evaluating the quality of association patterns.

The first set of criteria can be established through statistical arguments. Patterns that involve a set of mutually independent items or cover very few transactions are considered uninteresting because they may capture spurious relationships in the data. Such patterns can be eliminated by applying an

objective interestingness measure that uses statistics derived from data to determine whether a pattern is interesting. Examples of objective interestingness measures include support, confidence, and correlation.

The second set of criteria can be established through subjective arguments. A pattern is considered subjectively uninteresting unless it reveals unexpected information about the data or provides useful knowledge that can lead to profitable actions. For example, the rule $\{\text{Butter}\} \rightarrow \{\text{Bread}\}$ may not be interesting, despite having high support and confidence values, because the relationship represented by the rule may seem rather obvious. On the other hand, the rule $\{\text{Diapers}\} \rightarrow \{\text{Beer}\}$ is interesting because the relationship is quite unexpected and may suggest a new cross-selling opportunity for retailers. Incorporating subjective knowledge into pattern evaluation is a difficult task because it requires a considerable amount of prior information from the domain experts.

The following are some of the approaches for incorporating subjective knowledge into the pattern discovery task.

Visualization This approach requires a user-friendly environment to keep the human user in the loop. It also allows the domain experts to interact with the data mining system by interpreting and verifying the discovered patterns.

Template-based approach This approach allows the users to constrain the type of patterns extracted by the mining algorithm. Instead of reporting all the extracted rules, only rules that satisfy a user-specified template are returned to the users.

Subjective interestingness measure A subjective measure can be defined based on domain information such as concept hierarchy (to be discussed in Section 7.3) or profit margin of items. The measure can then be used to filter patterns that are obvious and non-actionable.

Readers interested in subjective interestingness measures may refer to resources listed in the bibliography at the end of this chapter.

6.7.1 Objective Measures of Interestingness

An objective measure is a data-driven approach for evaluating the quality of association patterns. It is domain-independent and requires minimal input from the users, other than to specify a threshold for filtering low-quality patterns. An objective measure is usually computed based on the frequency

Table 6.7. A 2-way contingency table for variables A and B .

	B	\bar{B}	
A	f_{11}	f_{10}	f_{1+}
\bar{A}	f_{01}	f_{00}	f_{0+}
	f_{+1}	f_{+0}	N

counts tabulated in a contingency table. Table 6.7 shows an example of a contingency table for a pair of binary variables, A and B . We use the notation \bar{A} (\bar{B}) to indicate that A (B) is absent from a transaction. Each entry f_{ij} in this 2×2 table denotes a frequency count. For example, f_{11} is the number of times A and B appear together in the same transaction, while f_{01} is the number of transactions that contain B but not A . The row sum f_{1+} represents the support count for A , while the column sum f_{+1} represents the support count for B . Finally, even though our discussion focuses mainly on asymmetric binary variables, note that contingency tables are also applicable to other attribute types such as symmetric binary, nominal, and ordinal variables.

Limitations of the Support-Confidence Framework Existing association rule mining formulation relies on the support and confidence measures to eliminate uninteresting patterns. The drawback of support was previously described in Section 6.8, in which many potentially interesting patterns involving low support items might be eliminated by the support threshold. The drawback of confidence is more subtle and is best demonstrated with the following example.

Example 6.3. Suppose we are interested in analyzing the relationship between people who drink tea and coffee. We may gather information about the beverage preferences among a group of people and summarize their responses into a table such as the one shown in Table 6.8.

Table 6.8. Beverage preferences among a group of 1000 people.

	<i>Coffee</i>	$\bar{\text{Coffee}}$	
<i>Tea</i>	150	50	200
$\bar{\text{Tea}}$	650	150	800
	800	200	1000

The information given in this table can be used to evaluate the association rule $\{\text{Tea}\} \rightarrow \{\text{Coffee}\}$. At first glance, it may appear that people who drink tea also tend to drink coffee because the rule's support (15%) and confidence (75%) values are reasonably high. This argument would have been acceptable except that the fraction of people who drink coffee, regardless of whether they drink tea, is 80%, while the fraction of tea drinkers who drink coffee is only 75%. Thus knowing that a person is a tea drinker actually decreases her probability of being a coffee drinker from 80% to 75%. The rule $\{\text{Tea}\} \rightarrow \{\text{Coffee}\}$ is therefore misleading despite its high confidence value. ■

The pitfall of confidence can be traced to the fact that the measure ignores the support of the itemset in the rule consequent. Indeed, if the support of coffee drinkers is taken into account, we would not be surprised to find that many of the people who drink tea also drink coffee. What is more surprising is that the fraction of tea drinkers who drink coffee is actually less than the overall fraction of people who drink coffee, which points to an inverse relationship between tea drinkers and coffee drinkers.

Because of the limitations in the support-confidence framework, various objective measures have been used to evaluate the quality of association patterns. Below, we provide a brief description of these measures and explain some of their strengths and limitations.

Interest Factor The tea-coffee example shows that high-confidence rules can sometimes be misleading because the confidence measure ignores the support of the itemset appearing in the rule consequent. One way to address this problem is by applying a metric known as lift:

$$\text{Lift} = \frac{c(A \rightarrow B)}{s(B)}, \quad (6.4)$$

which computes the ratio between the rule's confidence and the support of the itemset in the rule consequent. For binary variables, lift is equivalent to another objective measure called interest factor, which is defined as follows:

$$I(A, B) = \frac{s(A, B)}{s(A) \times s(B)} = \frac{Nf_{11}}{f_{1+}f_{+1}}. \quad (6.5)$$

Interest factor compares the frequency of a pattern against a baseline frequency computed under the statistical independence assumption. The baseline frequency for a pair of mutually independent variables is

$$\frac{f_{11}}{N} = \frac{f_{1+}}{N} \times \frac{f_{+1}}{N}, \quad \text{or equivalently, } f_{11} = \frac{f_{1+}f_{+1}}{N}. \quad (6.6)$$

Table 6.9. Contingency tables for the word pairs ($\{p,q\}$ and $\{r,s\}$).

	p	\bar{p}	
q	880	50	930
\bar{q}	50	20	70
	930	70	1000

	r	\bar{r}	
s	20	50	70
\bar{s}	50	880	930
	70	930	1000

This equation follows from the standard approach of using simple fractions as estimates for probabilities. The fraction f_{11}/N is an estimate for the joint probability $P(A, B)$, while f_{1+}/N and f_{+1}/N are the estimates for $P(A)$ and $P(B)$, respectively. If A and B are statistically independent, then $P(A, B) = P(A) \times P(B)$, thus leading to the formula shown in Equation 6.6. Using Equations 6.5 and 6.6, we can interpret the measure as follows:

$$I(A, B) = \begin{cases} = 1, & \text{if } A \text{ and } B \text{ are independent;} \\ > 1, & \text{if } A \text{ and } B \text{ are positively correlated;} \\ < 1, & \text{if } A \text{ and } B \text{ are negatively correlated.} \end{cases} \quad (6.7)$$

For the tea-coffee example shown in Table 6.8, $I = \frac{0.15}{0.2 \times 0.8} = 0.9375$, thus suggesting a slight negative correlation between tea drinkers and coffee drinkers.

Limitations of Interest Factor We illustrate the limitation of interest factor with an example from the text mining domain. In the text domain, it is reasonable to assume that the association between a pair of words depends on the number of documents that contain both words. For example, because of their stronger association, we expect the words `data` and `mining` to appear together more frequently than the words `compiler` and `mining` in a collection of computer science articles.

Table 6.9 shows the frequency of occurrences between two pairs of words, $\{p, q\}$ and $\{r, s\}$. Using the formula given in Equation 6.5, the interest factor for $\{p, q\}$ is 1.02 and for $\{r, s\}$ is 4.08. These results are somewhat troubling for the following reasons. Although p and q appear together in 88% of the documents, their interest factor is close to 1, which is the value when p and q are statistically independent. On the other hand, the interest factor for $\{r, s\}$ is higher than $\{p, q\}$ even though r and s seldom appear together in the same document. Confidence is perhaps the better choice in this situation because it considers the association between p and q (94.6%) to be much stronger than that between r and s (28.6%).

Correlation Analysis Correlation analysis is a statistical-based technique for analyzing relationships between a pair of variables. For continuous variables, correlation is defined using Pearson's correlation coefficient (see Equation 2.10 on page 77). For binary variables, correlation can be measured using the ϕ -coefficient, which is defined as

$$\phi = \frac{f_{11}f_{00} - f_{01}f_{10}}{\sqrt{f_{1+}f_{+1}f_{0+}f_{++}}}. \quad (6.8)$$

The value of correlation ranges from -1 (perfect negative correlation) to $+1$ (perfect positive correlation). If the variables are statistically independent, then $\phi = 0$. For example, the correlation between the tea and coffee drinkers given in Table 6.8 is -0.0625 .

Limitations of Correlation Analysis The drawback of using correlation can be seen from the word association example given in Table 6.9. Although the words p and q appear together more often than r and s , their ϕ -coefficients are identical, i.e., $\phi(p, q) = \phi(r, s) = 0.232$. This is because the ϕ -coefficient gives equal importance to both co-presence and co-absence of items in a transaction. It is therefore more suitable for analyzing symmetric binary variables. Another limitation of this measure is that it does not remain invariant when there are proportional changes to the sample size. This issue will be discussed in greater detail when we describe the properties of objective measures on page 377.

IS Measure IS is an alternative measure that has been proposed for handling asymmetric binary variables. The measure is defined as follows:

$$IS(A, B) = \sqrt{I(A, B) \times s(A, B)} = \frac{s(A, B)}{\sqrt{s(A)s(B)}}. \quad (6.9)$$

Note that IS is large when the interest factor and support of the pattern are large. For example, the value of IS for the word pairs $\{p, q\}$ and $\{r, s\}$ shown in Table 6.9 are 0.946 and 0.286, respectively. Contrary to the results given by interest factor and the ϕ -coefficient, the IS measure suggests that the association between $\{p, q\}$ is stronger than $\{r, s\}$, which agrees with what we expect from word associations in documents.

It is possible to show that IS is mathematically equivalent to the cosine measure for binary variables (see Equation 2.7 on page 75). In this regard, we

Table 6.10. Example of a contingency table for items p and q .

	q	\bar{q}	
p	800	100	900
\bar{p}	100	0	100
	900	100	1000

consider A and B as a pair of bit vectors, $A \bullet B = s(A, B)$ the dot product between the vectors, and $|A| = \sqrt{s(A)}$ the magnitude of vector A . Therefore:

$$IS(A, B) = \frac{s(A, B)}{\sqrt{s(A) \times s(B)}} = \frac{A \bullet B}{|A| \times |B|} = \text{cosine}(A, B). \quad (6.10)$$

The IS measure can also be expressed as the geometric mean between the confidence of association rules extracted from a pair of binary variables:

$$IS(A, B) = \sqrt{\frac{s(A, B)}{s(A)} \times \frac{s(A, B)}{s(B)}} = \sqrt{c(A \rightarrow B) \times c(B \rightarrow A)}. \quad (6.11)$$

Because the geometric mean between any two numbers is always closer to the smaller number, the IS value of an itemset $\{p, q\}$ is low whenever one of its rules, $p \rightarrow q$ or $q \rightarrow p$, has low confidence.

Limitations of IS Measure The IS value for a pair of independent itemsets, A and B , is

$$IS_{\text{indep}}(A, B) = \frac{s(A, B)}{\sqrt{s(A) \times s(B)}} = \frac{s(A) \times s(B)}{\sqrt{s(A) \times s(B)}} = \sqrt{s(A) \times s(B)}.$$

Since the value depends on $s(A)$ and $s(B)$, IS shares a similar problem as the confidence measure—that the value of the measure can be quite large, even for uncorrelated and negatively correlated patterns. For example, despite the large IS value between items p and q given in Table 6.10 (0.889), it is still less than the expected value when the items are statistically independent ($IS_{\text{indep}} = 0.9$).

Alternative Objective Interestingness Measures

Besides the measures we have described so far, there are other alternative measures proposed for analyzing relationships between pairs of binary variables. These measures can be divided into two categories, symmetric and asymmetric measures. A measure M is symmetric if $M(A \rightarrow B) = M(B \rightarrow A)$. For example, interest factor is a symmetric measure because its value is identical for the rules $A \rightarrow B$ and $B \rightarrow A$. In contrast, confidence is an asymmetric measure since the confidence for $A \rightarrow B$ and $B \rightarrow A$ may not be the same. Symmetric measures are generally used for evaluating itemsets, while asymmetric measures are more suitable for analyzing association rules. Tables 6.11 and 6.12 provide the definitions for some of these measures in terms of the frequency counts of a 2×2 contingency table.

Consistency among Objective Measures

Given the wide variety of measures available, it is reasonable to question whether the measures can produce similar ordering results when applied to a set of association patterns. If the measures are consistent, then we can choose any one of them as our evaluation metric. Otherwise, it is important to understand what their differences are in order to determine which measure is more suitable for analyzing certain types of patterns.

Table 6.11. Examples of symmetric objective measures for the itemset $\{A, B\}$.

Measure (Symbol)	Definition
Correlation (ϕ)	$\frac{Nf_{11}-f_{1+}f_{+1}}{\sqrt{f_{1+}f_{+1}f_{0+}f_{++}}}$
Odds ratio (α)	$(f_{10}f_{01})/(f_{10}f_{01})$
Kappa (κ)	$\frac{Nf_{11}+Nf_{00}-f_{1+}f_{+1}-f_{0+}f_{++}}{N^2-f_{1+}f_{+1}-f_{0+}f_{++}}$
Interest (I)	$(Nf_{11})/(f_{1+}f_{+1})$
Cosine (IS)	$(f_{11})/(\sqrt{f_{1+}f_{+1}})$
Piatetsky-Shapiro (PS)	$\frac{f_{11}-f_{1+}f_{+1}}{N^2}$
Collective strength (S)	$\frac{f_{11}+f_{00}}{f_{1+}f_{+1}+f_{0+}f_{++}} \times \frac{N-f_{1+}f_{+1}-f_{0+}f_{++}}{N-f_{11}-f_{00}}$
Jaccard (ζ)	$f_{11}/(f_{1+}+f_{+1}-f_{11})$
All-confidence (h)	$\min[\frac{f_{11}}{f_{1+}}, \frac{f_{11}}{f_{+1}}]$

Table 6.12. Examples of asymmetric objective measures for the rule $A \rightarrow B$.

Measure (Symbol)	Definition
Goodman-Kruskal (λ)	$(\sum_j \max_k f_{jk} - \max_k f_{+k}) / (N - \max_k f_{+k})$
Mutual Information (M)	$(\sum_i \sum_j \frac{f_{ij}}{N} \log \frac{Nf_{ij}}{f_{+i}f_{+j}}) / (-\sum_i \frac{f_{i+}}{N} \log \frac{f_{i+}}{N})$
J-Measure (J)	$\frac{f_{11}}{N} \log \frac{Nf_{11}}{f_{1+}f_{+1}} + \frac{f_{10}}{N} \log \frac{Nf_{10}}{f_{1+}f_{+0}}$
Gini index (G)	$\frac{f_{1+}}{N} \times (\frac{f_{11}}{f_{1+}})^2 + (\frac{f_{10}}{f_{1+}})^2 - (\frac{f_{+1}}{N})^2$ + $\frac{f_{0+}}{N} \times [(\frac{f_{01}}{f_{0+}})^2 + (\frac{f_{00}}{f_{0+}})^2] - (\frac{f_{+0}}{N})^2$
Laplace (L)	$(f_{11} + 1) / (f_{1+} + 2)$
Conviction (V)	$(f_{1+}f_{+0}) / (Nf_{10})$
Certainty factor (F)	$(\frac{f_{11}}{f_{1+}} - \frac{f_{+1}}{N}) / (1 - \frac{f_{+1}}{N})$
Added Value (AV)	$\frac{f_{11}}{f_{1+}} - \frac{f_{+1}}{N}$

Table 6.13. Example of contingency tables.

Example	f_{11}	f_{10}	f_{01}	f_{00}
E_1	8123	83	424	1370
E_2	8330	2	622	1046
E_3	3954	3080	5	2961
E_4	2886	1363	1320	4431
E_5	1500	2000	500	6000
E_6	4000	2000	1000	3000
E_7	9481	298	127	94
E_8	4000	2000	2000	2000
E_9	7450	2483	4	63
E_{10}	61	2483	4	7452

Suppose the symmetric and asymmetric measures are applied to rank the ten contingency tables shown in Table 6.13. These contingency tables are chosen to illustrate the differences among the existing measures. The ordering produced by these measures are shown in Tables 6.14 and 6.15, respectively (with 1 as the most interesting and 10 as the least interesting table). Although some of the measures appear to be consistent with each other, there are certain measures that produce quite different ordering results. For example, the rankings given by the ϕ -coefficient agree with those provided by κ and collective strength, but are somewhat different than the rankings produced by interest

Table 6.14. Rankings of contingency tables using the symmetric measures given in Table 6.11.

	ϕ	α	κ	I	IS	PS	S	ζ	h
E_1	1	3	1	6	2	2	1	2	2
E_2	2	1	2	7	3	5	2	3	3
E_3	3	2	4	4	5	1	3	6	8
E_4	4	8	3	3	7	3	4	7	5
E_5	5	7	6	2	9	6	6	9	9
E_6	6	9	5	5	6	4	5	5	7
E_7	7	6	7	9	1	8	7	1	1
E_8	8	10	8	8	8	7	8	8	7
E_9	9	4	9	10	4	9	9	4	4
E_{10}	10	5	10	1	10	10	10	10	10

Table 6.15. Rankings of contingency tables using the asymmetric measures given in Table 6.12.

	λ	M	J	G	L	V	F	AV
E_1	1	1	1	1	4	2	2	5
E_2	2	2	2	3	5	1	1	6
E_3	5	3	5	2	2	6	6	4
E_4	4	6	3	4	9	3	3	1
E_5	9	7	4	6	8	5	5	2
E_6	3	8	6	5	7	4	4	3
E_7	7	5	9	8	3	7	7	9
E_8	8	9	7	7	10	8	8	7
E_9	6	4	10	9	1	9	9	10
E_{10}	10	10	8	10	6	10	10	8

factor and odds ratio. Furthermore, a contingency table such as E_{10} is ranked lowest according to the ϕ -coefficient, but highest according to interest factor.

Properties of Objective Measures

The results shown in Table 6.14 suggest that a significant number of the measures provide conflicting information about the quality of a pattern. To understand their differences, we need to examine the properties of these measures.

Inversion Property Consider the bit vectors shown in Figure 6.28. The 0/1 bit in each column vector indicates whether a transaction (row) contains a particular item (column). For example, the vector A indicates that item a

A	B	C	D	E	F
1	0	0	1	0	0
0	0	1	1	1	1
0	0	1	1	1	0
0	0	1	1	1	0
0	1	1	0	1	1
0	0	1	1	1	0
0	0	1	1	1	0
0	0	1	1	1	0
1	0	0	1	0	0

Figure 6.28. Effect of the inversion operation. The vectors C and E are inversions of vector A, while the vector D is an inversion of vectors B and F.

belongs to the first and last transactions, whereas the vector B indicates that item b is contained only in the fifth transaction. The vectors C and E are in fact related to the vector A—their bits have been inverted from 0's (absence) to 1's (presence), and vice versa. Similarly, D is related to vectors B and F by inverting their bits. The process of flipping a bit vector is called inversion. If a measure is invariant under the inversion operation, then its value for the vector pair (C, D) should be identical to its value for (A, B). The inversion property of a measure can be tested as follows.

Definition 6.6 (Inversion Property). An objective measure M is invariant under the inversion operation if its value remains the same when exchanging the frequency counts f_{11} with f_{00} and f_{10} with f_{01} .

Among the measures that remain invariant under this operation include the ϕ -coefficient, odds ratio, κ , and collective strength. These measures may not be suitable for analyzing asymmetric binary data. For example, the ϕ -coefficient between C and D is identical to the ϕ -coefficient between A and B, even though items c and d appear together more frequently than a and b. Furthermore, the ϕ -coefficient between C and D is less than that between E and F even though items e and f appear together only once! We had previously raised this issue when discussing the limitations of the ϕ -coefficient on page 375. For asymmetric binary data, measures that do not remain invariant under the inversion operation are preferred. Some of the non-invariant measures include interest factor, IS , PS , and the Jaccard coefficient.

Null Addition Property Suppose we are interested in analyzing the relationship between a pair of words, such as `data` and `mining`, in a set of documents. If a collection of articles about ice fishing is added to the data set, should the association between `data` and `mining` be affected? This process of adding unrelated data (in this case, documents) to a given data set is known as the null addition operation.

Definition 6.7 (Null Addition Property). An objective measure M is invariant under the null addition operation if it is not affected by increasing f_{00} , while all other frequencies in the contingency table stay the same.

For applications such as document analysis or market basket analysis, the measure is expected to remain invariant under the null addition operation. Otherwise, the relationship between words may disappear simply by adding enough documents that do not contain both words! Examples of measures that satisfy this property include cosine (IS) and Jaccard (ξ) measures, while those that violate this property include interest factor, PS , odds ratio, and the ϕ -coefficient.

Scaling Property Table 6.16 shows the contingency tables for gender and the grades achieved by students enrolled in a particular course in 1993 and 2004. The data in these tables showed that the number of male students has doubled since 1993, while the number of female students has increased by a factor of 3. However, the male students in 2004 are not performing any better than those in 1993 because the ratio of male students who achieve a high grade to those who achieve a low grade is still the same, i.e., 3:4. Similarly, the female students in 2004 are performing no better than those in 1993. The association between grade and gender is expected to remain unchanged despite changes in the sampling distribution.

Table 6.16. The grade-gender example.

		Male	Female		Male	Female		
		High	Low		High	Low		
High	Male	30	20	50	Male	60	60	120
	Female	40	10	50	Female	80	30	110
	Total	70	30	100	Total	140	90	230

(a) Sample data from 1993.

(b) Sample data from 2004.

Table 6.17. Properties of symmetric measures.

Symbol	Measure	Inversion	Null Addition	Scaling
ϕ	ϕ -coefficient	Yes	No	No
α	odds ratio	Yes	No	Yes
κ	Cohen's	Yes	No	No
I	Interest	No	No	No
IS	Cosine	No	Yes	No
PS	Platetsky-Shapiro's	Yes	No	No
S	Collective strength	Yes	No	No
ζ	Jaccard	No	Yes	No
h	All-confidence	No	No	No
s	Support	No	No	No

Definition 6.8 (Scaling Invariance Property). An objective measure M is invariant under the row/column scaling operation if $M(T) = M(T')$, where T is a contingency table with frequency counts $[f_{11}; f_{10}; f_{01}; f_{00}]$, T' is a contingency table with scaled frequency counts $[k_1 k_3 f_{11}; k_2 k_3 f_{10}; k_1 k_4 f_{01}; k_2 k_4 f_{00}]$, and k_1, k_2, k_3, k_4 are positive constants.

From Table 6.17, notice that only the odds ratio (α) is invariant under the row and column scaling operations. All other measures such as the ϕ -coefficient, κ , IS , interest factor, and collective strength (S) change their values when the rows and columns of the contingency table are rescaled. Although we do not discuss the properties of asymmetric measures (such as confidence, J-measure, Gini index, and conviction), it is clear that such measures do not preserve their values under inversion and row/column scaling operations, but are invariant under the null addition operation.

6.7.2 Measures beyond Pairs of Binary Variables

The measures shown in Tables 6.11 and 6.12 are defined for pairs of binary variables (e.g., 2-itemsets or association rules). However, many of them, such as support and all-confidence, are also applicable to larger-sized itemsets. Other measures, such as interest factor, IS , PS , and Jaccard coefficient, can be extended to more than two variables using the frequency tables tabulated in a multidimensional contingency table. An example of a three-dimensional contingency table for a , b , and c is shown in Table 6.18. Each entry f_{ijk} in this table represents the number of transactions that contain a particular combination of items a , b , and c . For example, f_{101} is the number of transactions that contain a and c , but not b . On the other hand, a marginal frequency

Table 6.18. Example of a three-dimensional contingency table.

c	b	\bar{b}		\bar{c}	b	\bar{b}	
a	f_{111}	f_{101}	f_{1+1}	a	f_{110}	f_{100}	f_{1+0}
\bar{a}	f_{011}	f_{001}	f_{0+1}	\bar{a}	f_{010}	f_{000}	f_{0+0}
	f_{+11}	f_{+01}	f_{++1}		f_{+10}	f_{+00}	f_{++0}

such as f_{1+1} is the number of transactions that contain a and c , irrespective of whether b is present in the transaction.

Given a k -itemset $\{i_1, i_2, \dots, i_k\}$, the condition for statistical independence can be stated as follows:

$$f_{i_1 i_2 \dots i_k} = \frac{f_{i_1+ \dots +} \times f_{+i_2+ \dots +} \times \dots \times f_{++ \dots i_k}}{N^{k-1}}. \quad (6.12)$$

With this definition, we can extend objective measures such as interest factor and PS , which are based on deviations from statistical independence, to more than two variables:

$$\begin{aligned} I &= \frac{N^{k-1} \times f_{i_1 i_2 \dots i_k}}{f_{i_1+ \dots +} \times f_{+i_2+ \dots +} \times \dots \times f_{++ \dots i_k}} \\ PS &= \frac{f_{i_1 i_2 \dots i_k}}{N} - \frac{f_{i_1+ \dots +} \times f_{+i_2+ \dots +} \times \dots \times f_{++ \dots i_k}}{N^k} \end{aligned}$$

Another approach is to define the objective measure as the maximum, minimum, or average value for the associations between pairs of items in a pattern. For example, given a k -itemset $X = \{i_1, i_2, \dots, i_k\}$, we may define the ϕ -coefficient for X as the average ϕ -coefficient between every pair of items (i_p, i_q) in X . However, because the measure considers only pairwise associations, it may not capture all the underlying relationships within a pattern.

Analysis of multidimensional contingency tables is more complicated because of the presence of partial associations in the data. For example, some associations may appear or disappear when conditioned upon the value of certain variables. This problem is known as Simpson's paradox and is described in the next section. More sophisticated statistical techniques are available to analyze such relationships, e.g., loglinear models, but these techniques are beyond the scope of this book.

Table 6.19. A two-way contingency table between the sale of high-definition television and exercise machine.

Buy HDTV	Buy Exercise Machine		
	Yes	No	
Yes	99	81	180
No	54	66	120
	153	147	300

Table 6.20. Example of a three-way contingency table.

Customer Group	Buy HDTV	Buy Exercise Machine		Total
		Yes	No	
College Students	Yes	1	9	10
	No	4	30	34
Working Adult	Yes	98	72	170
	No	50	36	86

6.7.3 Simpson's Paradox

It is important to exercise caution when interpreting the association between variables because the observed relationship may be influenced by the presence of other confounding factors, i.e., hidden variables that are not included in the analysis. In some cases, the hidden variables may cause the observed relationship between a pair of variables to disappear or reverse its direction, a phenomenon that is known as Simpson's paradox. We illustrate the nature of this paradox with the following example.

Consider the relationship between the sale of high-definition television (HDTV) and exercise machine, as shown in Table 6.19. The rule $\{\text{HDTV}=\text{Yes}\} \rightarrow \{\text{Exercise machine}=\text{Yes}\}$ has a confidence of $99/180 = 55\%$ and the rule $\{\text{HDTV}=\text{No}\} \rightarrow \{\text{Exercise machine}=\text{Yes}\}$ has a confidence of $54/120 = 45\%$. Together, these rules suggest that customers who buy high-definition televisions are more likely to buy exercise machines than those who do not buy high-definition televisions.

However, a deeper analysis reveals that the sales of these items depend on whether the customer is a college student or a working adult. Table 6.20 summarizes the relationship between the sale of HDTVs and exercise machines among college students and working adults. Notice that the support counts given in the table for college students and working adults sum up to the frequencies shown in Table 6.19. Furthermore, there are more working adults

than college students who buy these items. For college students:

$$c(\{\text{HDTV}=\text{Yes}\} \rightarrow \{\text{Exercise machine}=\text{Yes}\}) = 1/10 = 10\%, \\ c(\{\text{HDTV}=\text{No}\} \rightarrow \{\text{Exercise machine}=\text{Yes}\}) = 4/34 = 11.8\%,$$

while for working adults:

$$c(\{\text{HDTV}=\text{Yes}\} \rightarrow \{\text{Exercise machine}=\text{Yes}\}) = 98/170 = 57.7\%, \\ c(\{\text{HDTV}=\text{No}\} \rightarrow \{\text{Exercise machine}=\text{Yes}\}) = 50/86 = 58.1\%.$$

The rules suggest that, for each group, customers who do not buy high-definition televisions are more likely to buy exercise machines, which contradict the previous conclusion when data from the two customer groups are pooled together. Even if alternative measures such as correlation, odds ratio, or interest are applied, we still find that the sale of HDTV and exercise machine is positively correlated in the combined data but is negatively correlated in the stratified data (see Exercise 20 on page 414). The reversal in the direction of association is known as Simpson's paradox.

The paradox can be explained in the following way. Notice that most customers who buy HDTVs are working adults. Working adults are also the largest group of customers who buy exercise machines. Because nearly 85% of the customers are working adults, the observed relationship between HDTV and exercise machine turns out to be stronger in the combined data than what it would have been if the data is stratified. This can also be illustrated mathematically as follows. Suppose

$$a/b < c/d \text{ and } p/q < r/s,$$

where a/b and p/q may represent the confidence of the rule $A \rightarrow B$ in two different strata, while c/d and r/s may represent the confidence of the rule $\bar{A} \rightarrow \bar{B}$ in the two strata. When the data is pooled together, the confidence values of the rules in the combined data are $(a+p)/(b+q)$ and $(c+r)/(d+s)$, respectively. Simpson's paradox occurs when

$$\frac{a+p}{b+q} > \frac{c+r}{d+s},$$

thus leading to the wrong conclusion about the relationship between the variables. The lesson here is that proper stratification is needed to avoid generating spurious patterns resulting from Simpson's paradox. For example, market

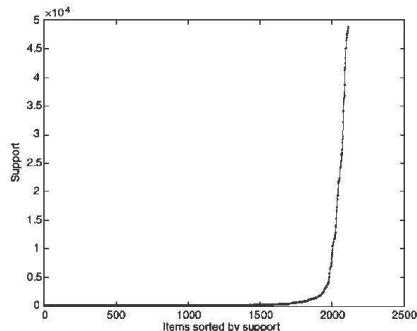


Figure 6.29. Support distribution of items in the census data set.

basket data from a major supermarket chain should be stratified according to store locations, while medical records from various patients should be stratified according to confounding factors such as age and gender.

6.8 Effect of Skewed Support Distribution

The performances of many association analysis algorithms are influenced by properties of their input data. For example, the computational complexity of the *Apriori* algorithm depends on properties such as the number of items in the data and average transaction width. This section examines another important property that has significant influence on the performance of association analysis algorithms as well as the quality of extracted patterns. More specifically, we focus on data sets with skewed support distributions, where most of the items have relatively low to moderate frequencies, but a small number of them have very high frequencies.

An example of a real data set that exhibits such a distribution is shown in Figure 6.29. The data, taken from the PUMS (Public Use Microdata Sample) census data, contains 49,046 records and 2113 asymmetric binary variables. We shall treat the asymmetric binary variables as items and records as transactions in the remainder of this section. While more than 80% of the items have support less than 1%, a handful of them have support greater than 90%.

Table 6.21. Grouping the items in the census data set based on their support values.

Group	G_1	G_2	G_3
Support	< 1%	1% – 90%	> 90%
Number of Items	1735	358	20

To illustrate the effect of skewed support distribution on frequent itemset mining, we divide the items into three groups, G_1 , G_2 , and G_3 , according to their support levels. The number of items that belong to each group is shown in Table 6.21.

Choosing the right support threshold for mining this data set can be quite tricky. If we set the threshold too high (e.g., 20%), then we may miss many interesting patterns involving the low support items from G_1 . In market basket analysis, such low support items may correspond to expensive products (such as jewelry) that are seldom bought by customers, but whose patterns are still interesting to retailers. Conversely, when the threshold is set too low, it becomes difficult to find the association patterns due to the following reasons. First, the computational and memory requirements of existing association analysis algorithms increase considerably with low support thresholds. Second, the number of extracted patterns also increases substantially with low support thresholds. Third, we may extract many spurious patterns that relate a high-frequency item such as milk to a low-frequency item such as caviar. Such patterns, which are called cross-support patterns, are likely to be spurious because their correlations tend to be weak. For example, at a support threshold equal to 0.05%, there are 18,847 frequent pairs involving items from G_1 and G_3 . Out of these, 93% of them are cross-support patterns; i.e., the patterns contain items from both G_1 and G_3 . The maximum correlation obtained from the cross-support patterns is 0.029, which is much lower than the maximum correlation obtained from frequent patterns involving items from the same group (which is as high as 1.0). Similar statement can be made about many other interestingness measures discussed in the previous section. This example shows that a large number of weakly correlated cross-support patterns can be generated when the support threshold is sufficiently low. Before presenting a methodology for eliminating such patterns, we formally define the concept of cross-support patterns.

Definition 6.9 (Cross-Support Pattern). A cross-support pattern is an itemset $X = \{i_1, i_2, \dots, i_k\}$ whose support ratio

$$r(X) = \frac{\min [s(i_1), s(i_2), \dots, s(i_k)]}{\max [s(i_1), s(i_2), \dots, s(i_k)]}, \quad (6.13)$$

is less than a user-specified threshold h_c .

Example 6.4. Suppose the support for milk is 70%, while the support for sugar is 10% and caviar is 0.04%. Given $h_c = 0.01$, the frequent itemset {milk, sugar, caviar} is a cross-support pattern because its support ratio is

$$r = \frac{\min [0.7, 0.1, 0.0004]}{\max [0.7, 0.1, 0.0004]} = \frac{0.0004}{0.7} = 0.00058 < 0.01.$$

■

Existing measures such as support and confidence may not be sufficient to eliminate cross-support patterns, as illustrated by the data set shown in Figure 6.30. Assuming that $h_c = 0.3$, the itemsets $\{p, q\}$, $\{p, r\}$, and $\{p, q, r\}$ are cross-support patterns because their support ratios, which are equal to 0.2, are less than the threshold h_c . Although we can apply a high support threshold, say, 20%, to eliminate the cross-support patterns, this may come at the expense of discarding other interesting patterns such as the strongly correlated itemset, $\{q, r\}$ that has support equal to 16.7%.

Confidence pruning also does not help because the confidence of the rules extracted from cross-support patterns can be very high. For example, the confidence for $\{q\} \rightarrow \{p\}$ is 80% even though $\{p, q\}$ is a cross-support pattern. The fact that the cross-support pattern can produce a high-confidence rule should not come as a surprise because one of its items (p) appears very frequently in the data. Therefore, p is expected to appear in many of the transactions that contain q . Meanwhile, the rule $\{q\} \rightarrow \{r\}$ also has high confidence even though $\{q, r\}$ is not a cross-support pattern. This example demonstrates the difficulty of using the confidence measure to distinguish between rules extracted from cross-support and non-cross-support patterns.

Returning to the previous example, notice that the rule $\{p\} \rightarrow \{q\}$ has very low confidence because most of the transactions that contain p do not contain q . In contrast, the rule $\{r\} \rightarrow \{q\}$, which is derived from the pattern $\{q, r\}$, has very high confidence. This observation suggests that cross-support patterns can be detected by examining the lowest confidence rule that can be extracted from a given itemset. The proof of this statement can be understood as follows.

p	q	r
0	1	1
1	1	1
1	1	1
1	1	1
1	0	0
1	0	0
1	0	0
1	0	0
1	0	0
1	0	0
1	0	0
1	0	0
1	0	0
1	0	0
1	0	0
1	0	0
0	0	0
0	0	0
0	0	0

Figure 6.30. A transaction data set containing three items, p , q , and r , where p is a high support item and q and r are low support items.

1. Recall the following anti-monotone property of confidence:

$$\text{conf}(\{i_1 i_2\} \rightarrow \{i_3, i_4, \dots, i_k\}) \leq \text{conf}(\{i_1 i_2 i_3\} \rightarrow \{i_4, i_5, \dots, i_k\}).$$

This property suggests that confidence never increases as we shift more items from the left- to the right-hand side of an association rule. Because of this property, the lowest confidence rule extracted from a frequent itemset contains only one item on its left-hand side. We denote the set of all rules with only one item on its left-hand side as R_1 .

2. Given a frequent itemset $\{i_1, i_2, \dots, i_k\}$, the rule

$$\{i_j\} \rightarrow \{i_1, i_2, \dots, i_{j-1}, i_{j+1}, \dots, i_k\}$$

has the lowest confidence in R_1 if $s(i_j) = \max [s(i_1), s(i_2), \dots, s(i_k)]$. This follows directly from the definition of confidence as the ratio between the rule's support and the support of the rule antecedent.

3. Summarizing the previous points, the lowest confidence attainable from a frequent itemset $\{i_1, i_2, \dots, i_k\}$ is

$$\frac{s(\{i_1, i_2, \dots, i_k\})}{\max [s(i_1), s(i_2), \dots, s(i_k)]}.$$

This expression is also known as the h-confidence or all-confidence measure. Because of the anti-monotone property of support, the numerator of the h-confidence measure is bounded by the minimum support of any item that appears in the frequent itemset. In other words, the h-confidence of an itemset $X = \{i_1, i_2, \dots, i_k\}$ must not exceed the following expression:

$$\text{h-confidence}(X) \leq \frac{\min [s(i_1), s(i_2), \dots, s(i_k)]}{\max [s(i_1), s(i_2), \dots, s(i_k)]}.$$

Note the equivalence between the upper bound of h-confidence and the support ratio (r) given in Equation 6.13. Because the support ratio for a cross-support pattern is always less than h_c , the h-confidence of the pattern is also guaranteed to be less than h_c .

Therefore, cross-support patterns can be eliminated by ensuring that the h-confidence values for the patterns exceed h_c . As a final note, it is worth mentioning that the advantages of using h-confidence go beyond eliminating cross-support patterns. The measure is also anti-monotone, i.e.,

$$\text{h-confidence}(\{i_1, i_2, \dots, i_k\}) \geq \text{h-confidence}(\{i_1, i_2, \dots, i_{k+1}\}),$$

and thus can be incorporated directly into the mining algorithm. Furthermore, h-confidence ensures that the items contained in an itemset are strongly associated with each other. For example, suppose the h-confidence of an itemset X is 80%. If one of the items in X is present in a transaction, there is at least an 80% chance that the rest of the items in X also belong to the same transaction. Such strongly associated patterns are called hyperclique patterns.

6.9 Bibliographic Notes

The association rule mining task was first introduced by Agrawal et al. in [228, 229] to discover interesting relationships among items in market basket

transactions. Since its inception, extensive studies have been conducted to address the various conceptual, implementation, and application issues pertaining to the association analysis task. A summary of the various research activities in this area is shown in Figure 6.31.

Conceptual Issues

Research in conceptual issues is focused primarily on (1) developing a framework to describe the theoretical underpinnings of association analysis, (2) extending the formulation to handle new types of patterns, and (3) extending the formulation to incorporate attribute types beyond asymmetric binary data.

Following the pioneering work by Agrawal et al., there has been a vast amount of research on developing a theory for the association analysis problem. In [254], Gunopoulos et al. showed a relation between the problem of finding maximal frequent itemsets and the hypergraph transversal problem. An upper bound on the complexity of association analysis task was also derived. Zaki et al. [334, 336] and Pasquier et al. [294] have applied formal concept analysis to study the frequent itemset generation problem. The work by Zaki et al. have subsequently led them to introduce the notion of closed frequent itemsets [336]. Friedman et al. have studied the association analysis problem in the context of bump hunting in multidimensional space [252]. More specifically, they consider frequent itemset generation as the task of finding high probability density regions in multidimensional space.

Over the years, new types of patterns have been defined, such as profile association rules [225], cyclic association rules [290], fuzzy association rules [273], exception rules [316], negative association rules [238, 304], weighted association rules [240, 300], dependence rules [308], peculiar rules [340], inter-transaction association rules [250, 323], and partial classification rules [231, 285]. Other types of patterns include closed itemsets [294, 336], maximal itemsets [234], hyperclique patterns [330], support envelopes [314], emerging patterns [246], and contrast sets [233]. Association analysis has also been successfully applied to sequential [230, 312], spatial [266], and graph-based [268, 274, 293, 331, 335] data. The concept of cross-support pattern was first introduced by Hui et al. in [330]. An efficient algorithm (called Hyperclique Miner) that automatically eliminates cross-support patterns was also proposed by the authors.

Substantial research has been conducted to extend the original association rule formulation to nominal [311], ordinal [281], interval [284], and ratio [253, 255, 311, 325, 339] attributes. One of the key issues is how to define the support measure for these attributes. A methodology was proposed by Steinbach et

al. [315] to extend the traditional notion of support to more general patterns and attribute types.

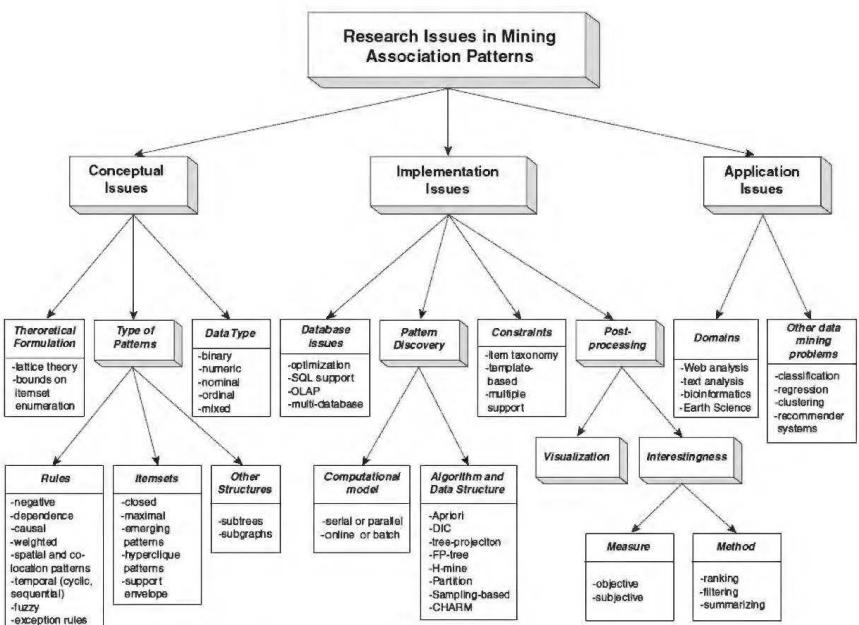


Figure 6.31. A summary of the various research activities in association analysis.

Implementation Issues

Research activities in this area revolve around (1) integrating the mining capability into existing database technology, (2) developing efficient and scalable mining algorithms, (3) handling user-specified or domain-specific constraints, and (4) post-processing the extracted patterns.

There are several advantages to integrating association analysis into existing database technology. First, it can make use of the indexing and query processing capabilities of the database system. Second, it can also exploit the DBMS support for scalability, check-pointing, and parallelization [301]. The SETM algorithm developed by Houtsma et al. [265] was one of the earliest algorithms to support association rule discovery via SQL queries. Since then, numerous methods have been developed to provide capabilities for mining association rules in database systems. For example, the DMQL [258] and M-SQL [267] query languages extend the basic SQL with new operators for mining association rules. The Mine Rule operator [283] is an expressive SQL operator that can handle both clustered attributes and item hierarchies. Tsur et al. [322] developed a generate-and-test approach called *query flocks* for mining association rules. A distributed OLAP-based infrastructure was developed by Chen et al. [241] for mining multilevel association rules.

Dunkel and Soparkar [248] investigated the time and storage complexity of the *Apriori* algorithm. The FP-growth algorithm was developed by Han et al. in [259]. Other algorithms for mining frequent itemsets include the DHP (dynamic hashing and pruning) algorithm proposed by Park et al. [292] and the Partition algorithm developed by Savasere et al. [303]. A sampling-based frequent itemset generation algorithm was proposed by Toivonen [320]. The algorithm requires only a single pass over the data, but it can produce more candidate itemsets than necessary. The Dynamic Itemset Counting (DIC) algorithm [239] makes only 1.5 passes over the data and generates less candidate itemsets than the sampling-based algorithm. Other notable algorithms include the tree-projection algorithm [223] and H-Mine [295]. Survey articles on frequent itemset generation algorithms can be found in [226, 262]. A repository of data sets and algorithms is available at the Frequent Itemset Mining Implementations (FIMI) repository (<http://fimi.cs.helsinki.fi>). Parallel algorithms for mining association patterns have been developed by various authors [224, 256, 287, 306, 337]. A survey of such algorithms can be found in [333]. Online and incremental versions of association rule mining algorithms had also been proposed by Hidher [260] and Chennng et al. [242].

Srikant et al. [313] have considered the problem of mining association rules in the presence of boolean constraints such as the following:

$$(Cookies \wedge Milk) \vee (\text{descendents}(Cookies) \wedge \neg\text{ancestors}(\text{Wheat Bread}))$$

Given such a constraint, the algorithm looks for rules that contain both cookies and milk, or rules that contain the descendant items of cookies but not ancestor items of wheat bread. Singh et al. [310] and Ng et al. [288] had also developed alternative techniques for constrained-based association rule mining. Constraints can also be imposed on the support for different itemsets. This problem was investigated by Wang et al. [324], Liu et al. in [279], and Seno et al. [305].

One potential problem with association analysis is the large number of patterns that can be generated by current algorithms. To overcome this problem, methods to rank, summarize, and filter patterns have been developed. Toivonen et al. [321] proposed the idea of eliminating redundant rules using structural rule covers and to group the remaining rules using clustering. Liu et al. [280] applied the statistical chi-square test to prune spurious patterns and summarized the remaining patterns using a subset of the patterns called direction setting rules. The use of objective measures to filter patterns has been investigated by many authors, including Brin et al. [238], Bayardo and Agrawal [235], Aggarwal and Yu [227], and DuMouchel and Pregibon [247]. The properties for many of these measures were analyzed by Piatetsky-Shapiro [297], Kamher and Singhal [270], Hilderman and Hamilton [261], and Tan et al. [318]. The grade-gender example used to highlight the importance of the row and column scaling invariance property was heavily influenced by the discussion given in [286] by Mosteller. Meanwhile, the tea-coffee example illustrating the limitation of confidence was motivated by an example given in [238] by Brin et al. Because of the limitation of confidence, Brin et al. [238] had proposed the idea of using interest factor as a measure of interestingness. The all-confidence measure was proposed by Omiecinski [289]. Xiong et al. [330] introduced the cross-support property and showed that the all-confidence measure can be used to eliminate cross-support patterns. A key difficulty in using alternative objective measures besides support is their lack of a monotonicity property, which makes it difficult to incorporate the measures directly into the mining algorithms. Xiong et al. [328] have proposed an efficient method for mining correlations by introducing an upper bound function to the ϕ -coefficient. Although the measure is non-monotone, it has an upper bound expression that can be exploited for the efficient mining of strongly correlated itempairs.

Fabris and Freitas [249] have proposed a method for discovering interesting associations by detecting the occurrences of Simpson's paradox [309]. Megiddo and Srikant [282] described an approach for validating the extracted

patterns using hypothesis testing methods. A resampling-based technique was also developed to avoid generating spurious patterns because of the multiple comparison problem. Bolton et al. [237] have applied the Benjamini-Hochberg [236] and Bonferroni correction methods to adjust the p-values of discovered patterns in market basket data. Alternative methods for handling the multiple comparison problem were suggested by Webb [326] and Zhang et al. [338].

Application of subjective measures to association analysis has been investigated by many authors. Silberschatz and Tuzhilin [307] presented two principles in which a rule can be considered interesting from a subjective point of view. The concept of unexpected condition rules was introduced by Liu et al. in [277]. Cooley et al. [243] analyzed the idea of combining soft belief sets using the Dempster-Shafer theory and applied this approach to identify contradictory and novel association patterns in Web data. Alternative approaches include using Bayesian networks [269] and neighborhood-based information [245] to identify subjectively interesting patterns.

Visualization also helps the user to quickly grasp the underlying structure of the discovered patterns. Many commercial data mining tools display the complete set of rules (which satisfy both support and confidence threshold criteria) as a two-dimensional plot, with each axis corresponding to the antecedent or consequent itemsets of the rule. Hofmann et al. [263] proposed using Mosaic plots and Double Decker plots to visualize association rules. This approach can visualize not only a particular rule, but also the overall contingency table between itemsets in the antecedent and consequent parts of the rule. Nevertheless, this technique assumes that the rule consequent consists of only a single attribute.

Application Issues

Association analysis has been applied to a variety of application domains such as Web mining [296, 317], document analysis [264], telecommunication alarm diagnosis [271], network intrusion detection [232, 244, 275], and bioinformatics [302, 327]. Applications of association and correlation pattern analysis to Earth Science studies have been investigated in [298, 299, 319].

Association patterns have also been applied to other learning problems such as classification [276, 278], regression [291], and clustering [257, 329, 332]. A comparison between classification and association rule mining was made by Freitas in his position paper [251]. The use of association patterns for clustering has been studied by many authors including Han et al. [257], Kosters et al. [272], Yang et al. [332] and Xiong et al. [329].

Bibliography

- [223] R. C. Agarwal, C. C. Aggarwal, and V. V. V. Prasad. A Tree Projection Algorithm for Generation of Frequent Itemsets. *Journal of Parallel and Distributed Computing (Special Issue on High Performance Data Mining)*, 61(3):350–371, 2001.
- [224] R. C. Agarwal and J. C. Shafer. Parallel Mining of Association Rules. *IEEE Transactions on Knowledge and Data Engineering*, 8(6):962–969, March 1996.
- [225] C. C. Aggarwal, Z. Sun, and P. S. Yu. Online Generation of Profile Association Rules. In *Proc. of the 4th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 129–133, New York, NY, August 1996.
- [226] C. C. Aggarwal and P. S. Yu. Mining Large Itemsets for Association Rules. *Data Engineering Bulletin*, 21(1):23–31, March 1998.
- [227] C. C. Aggarwal and P. S. Yu. Mining Associations with the Collective Strength Approach. *IEEE Trans. on Knowledge and Data Engineering*, 13(6):863–873, January/February 2001.
- [228] R. Agrawal, T. Imielinski, and A. Swami. Database mining: A performance perspective. *IEEE Transactions on Knowledge and Data Engineering*, 5:914–925, 1993.
- [229] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proc. ACM SIGMOD Intl. Conf. Management of Data*, pages 207–216, Washington, DC, 1993.
- [230] R. Agrawal and R. Srikant. Mining Sequential Patterns. In *Proc. of Intl. Conf. on Data Engineering*, pages 3–14, Taipei, Taiwan, 1995.
- [231] K. Ali, S. Manganaris, and R. Srikant. Partial Classification using Association Rules. In *Proc. of the 3rd Intl. Conf. on Knowledge Discovery and Data Mining*, pages 115–118, Newport Beach, CA, August 1997.
- [232] D. Barberá, J. Couto, S. Jajodia, and N. Wu. ADAM: A Testbed for Exploring the Use of Data Mining in Intrusion Detection. *SIGMOD Record*, 30(4):15–24, 2001.
- [233] S. D. Bay and M. Pazzani. Detecting Group Differences: Mining Contrast Sets. *Data Mining and Knowledge Discovery*, 5(3):213–246, 2001.
- [234] R. Bayardo. Efficiently Mining Long Patterns from Databases. In *Proc. of 1998 ACM-SIGMOD Intl. Conf. on Management of Data*, pages 85–93, Seattle, WA, June 1998.
- [235] R. Bayardo and R. Agrawal. Mining the Most Interesting Rules. In *Proc. of the 5th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 145–153, San Diego, CA, August 1999.
- [236] Y. Benjamini and Y. Hochberg. Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing. *Journal Royal Statistical Society B*, 57(1):289–300, 1995.
- [237] R. J. Bolton, D. J. Hand, and N. M. Adams. Determining Hit Rate in Pattern Search. In *Proc. of the ESF Exploratory Workshop on Pattern Detection and Discovery in Data Mining*, pages 36–48, London, UK, September 2002.
- [238] S. Brin, R. Motwani, and C. Silverstein. Beyond market baskets: Generalizing association rules to correlations. In *Proc. ACM SIGMOD Intl. Conf. Management of Data*, pages 265–276, Tucson, AZ, 1997.
- [239] S. Brin, R. Motwani, J. Ullman, and S. Tsur. Dynamic Itemset Counting and Implication Rules for market basket data. In *Proc. of 1997 ACM-SIGMOD Intl. Conf. on Management of Data*, pages 255–264, Tucson, AZ, June 1997.
- [240] C. H. Cai, A. Fu, C. H. Cheng, and W. W. Wong. Mining Association Rules with Weighted Items. In *Proc. of IEEE Intl. Database Engineering and Applications Symp.*, pages 68–77, Cardiff, Wales, 1998.

- [241] Q. Chen, U. Dayal, and M. Hsu. A Distributed OLAP infrastructure for E-Commerce. In *Proc. of the 4th IFCIS Intl. Conf. on Cooperative Information Systems*, pages 209–220, Edinburgh, Scotland, 1999.
- [242] D. C. Cheung, S. D. Lee, and B. Kao. A General Incremental Technique for Maintaining Discovered Association Rules. In *Proc. of the 5th Intl. Conf. on Database Systems for Advanced Applications*, pages 185–194, Melbourne, Australia, 1997.
- [243] R. Cooley, P. N. Tan, and J. Srivastava. Discovery of Interesting Usage Patterns from Web Data. In M. Spiliopoulou and B. Masand, editors, *Advances in Web Usage Analysis and User Profiling*, volume 1836, pages 163–182. Lecture Notes in Computer Science, 2000.
- [244] P. Dokas, L. Ertöz, V. Kumar, A. Lazarevic, J. Srivastava, and P. N. Tan. Data Mining for Network Intrusion Detection. In *Proc. NSF Workshop on Next Generation Data Mining*, Baltimore, MD, 2002.
- [245] G. Dong and J. Li. Interestingness of discovered association rules in terms of neighborhood-based unexpectedness. In *Proc. of the 2nd Pacific-Asia Conf. on Knowledge Discovery and Data Mining*, pages 72–86, Melbourne, Australia, April 1998.
- [246] G. Dong and J. Li. Efficient Mining of Emerging Patterns: Discovering Trends and Differences. In *Proc. of the 5th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 43–52, San Diego, CA, August 1999.
- [247] W. DuMouchel and D. Pregibon. Empirical Bayes Screening for Multi-Item Associations. In *Proc. of the 7th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 67–76, San Francisco, CA, August 2001.
- [248] B. Dunkel and N. Soparkar. Data Organization and Access for Efficient Data Mining. In *Proc. of the 15th Intl. Conf. on Data Engineering*, pages 522–529, Sydney, Australia, March 1999.
- [249] C. C. Fabris and A. A. Freitas. Discovering surprising patterns by detecting occurrences of Simpson's paradox. In *Proc. of the 19th SGES Intl. Conf. on Knowledge-Based Systems and Applied Artificial Intelligence*, pages 148–160, Cambridge, UK, December 1999.
- [250] L. Feng, H. J. Lu, J. X. Yu, and J. Han. Mining inter-transaction associations with templates. In *Proc. of the 8th Intl. Conf. on Information and Knowledge Management*, pages 225–233, Kansas City, Missouri, Nov 1999.
- [251] A. A. Freitas. Understanding the crucial differences between classification and discovery of association rules—a position paper. *SIGKDD Explorations*, 2(1):65–69, 2000.
- [252] J. H. Friedman and N. I. Fisher. Bump hunting in high-dimensional data. *Statistics and Computing*, 9(2):123–143, April 1999.
- [253] T. Fukuda, Y. Morimoto, S. Morishita, and T. Tokuyama. Mining Optimized Association Rules for Numeric Attributes. In *Proc. of the 15th Symp. on Principles of Database Systems*, pages 182–191, Montreal, Canada, June 1996.
- [254] D. Gunopulos, R. Khardon, H. Mannila, and H. Toivonen. Data Mining, Hypergraph Transversals, and Machine Learning. In *Proc. of the 16th Symp. on Principles of Database Systems*, pages 209–216, Tucson, AZ, May 1997.
- [255] E.-H. Han, G. Karypis, and V. Kumar. Min-Apriori: An Algorithm for Finding Association Rules in Data with Continuous Attributes. <http://www.cs.umn.edu/~han/>, 1997.
- [256] E.-H. Han, G. Karypis, and V. Kumar. Scalable Parallel Data Mining for Association Rules. In *Proc. of 1997 ACM-SIGMOD Intl. Conf. on Management of Data*, pages 277–288, Tucson, AZ, May 1997.
- [257] E.-H. Han, G. Karypis, V. Kumar, and B. Mobasher. Clustering Based on Association Rule Hypergraphs. In *Proc. of the 1997 ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, Tucson, AZ, 1997.
- [258] J. Han, Y. Fu, K. Koperski, W. Wang, and O. R. Zaiane. DMQL: A data mining query language for relational databases. In *Proc. of the 1996 ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, Montreal, Canada, June 1996.
- [259] J. Han, J. Pei, and Y. Yin. Mining Frequent Patterns without Candidate Generation. In *Proc. ACM-SIGMOD Int. Conf. on Management of Data (SIGMOD'00)*, pages 1–12, Dallas, TX, May 2000.
- [260] C. Hidber. Online Association Rule Mining. In *Proc. of 1999 ACM-SIGMOD Intl. Conf. on Management of Data*, pages 145–156, Philadelphia, PA, 1999.
- [261] R. J. Hilderman and H. J. Hamilton. *Knowledge Discovery and Measures of Interest*. Kluwer Academic Publishers, 2001.
- [262] J. Hipp, U. Guntzer, and G. Nakhaeizadeh. Algorithms for Association Rule Mining—A General Survey. *SigKDD Explorations*, 2(1):58–64, June 2000.
- [263] H. Hofmann, A. P. J. M. Siebes, and A. F. X. Wilhelm. Visualizing Association Rules with Interactive Mosaic Plots. In *Proc. of the 6th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 227–235, Boston, MA, August 2000.
- [264] J. D. Holt and S. M. Chung. Efficient Mining of Association Rules in Text Databases. In *Proc. of the 8th Intl. Conf. on Information and Knowledge Management*, pages 234–242, Kansas City, Missouri, 1999.
- [265] M. Houtsma and A. Swami. Set-oriented Mining for Association Rules in Relational Databases. In *Proc. of the 11th Intl. Conf. on Data Engineering*, pages 25–33, Taipei, Taiwan, 1995.
- [266] Y. Huang, S. Shekhar, and H. Xiong. Discovering Co-location Patterns from Spatial Datasets: A General Approach. *IEEE Trans. on Knowledge and Data Engineering*, 16(12):1472–1485, December 2004.
- [267] T. Imielinski, A. Virmani, and A. Abdullahevi. DataMine: Application Programming Interface and Query Language for Database Mining. In *Proc. of the 2nd Intl. Conf. on Knowledge Discovery and Data Mining*, pages 256–262, Portland, Oregon, 1996.
- [268] A. Inokuchi, T. Washio, and H. Motoda. An Apriori-based Algorithm for Mining Frequent Substructures from Graph Data. In *Proc. of the 4th European Conf. of Principles and Practice of Knowledge Discovery in Databases*, pages 13–23, Lyon, France, 2000.
- [269] S. Jaroszewicz and D. Simovici. Interestingness of Frequent Itemsets Using Bayesian Networks as Background Knowledge. In *Proc. of the 10th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 178–186, Seattle, WA, August 2004.
- [270] M. Kamber and R. Shinghal. Evaluating the Interestingness of Characteristic Rules. In *Proc. of the 2nd Intl. Conf. on Knowledge Discovery and Data Mining*, pages 263–266, Portland, Oregon, 1996.
- [271] M. Klemettinen. *A Knowledge Discovery Methodology for Telecommunication Network Alarm Databases*. PhD thesis, University of Helsinki, 1999.
- [272] W. A. Kosters, E. Marchiori, and A. Oerlemans. Mining Clusters with Association Rules. In *The 3rd Symp. on Intelligent Data Analysis (IDA99)*, pages 39–50, Amsterdam, August 1999.
- [273] C. M. Kuok, A. Fu, and M. H. Wong. Mining Fuzzy Association Rules in Databases. *ACM SIGMOD Record*, 27(1):41–46, March 1998.

- [274] M. Kuramochi and G. Karypis. Frequent Subgraph Discovery. In *Proc. of the 2001 IEEE Intl. Conf. on Data Mining*, pages 313–320, San Jose, CA, November 2001.
- [275] W. Lee, S. J. Stolfo, and K. W. Mok. Adaptive Intrusion Detection: A Data Mining Approach. *Artificial Intelligence Review*, 14(6):533–567, 2000.
- [276] W. Li, J. Han, and J. Pei. CMAR: Accurate and Efficient Classification Based on Multiple Class-association Rules. In *Proc. of the 2001 IEEE Intl. Conf. on Data Mining*, pages 369–376, San Jose, CA, 2001.
- [277] B. Liu, W. Hsu, and S. Chen. Using General Impressions to Analyze Discovered Classification Rules. In *Proc. of the 3rd Intl. Conf. on Knowledge Discovery and Data Mining*, pages 31–36, Newport Beach, CA, August 1997.
- [278] B. Liu, W. Hsu, and Y. Ma. Integrating Classification and Association Rule Mining. In *Proc. of the 4th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 80–86, New York, NY, August 1998.
- [279] B. Liu, W. Hsu, and Y. Ma. Mining association rules with multiple minimum supports. In *Proc. of the 5th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 125–134, San Diego, CA, August 1999.
- [280] B. Liu, W. Hsu, and Y. Ma. Pruning and Summarizing the Discovered Associations. In *Proc. of the 5th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 125–134, San Diego, CA, August 1999.
- [281] A. Marcus, J. I. Maletic, and K.-I. Lin. Ordinal association rules for error identification in data sets. In *Proc. of the 10th Intl. Conf. on Information and Knowledge Management*, pages 589–591, Atlanta, GA, October 2001.
- [282] N. Megiddo and R. Srikant. Discovering Predictive Association Rules. In *Proc. of the 4th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 274–278, New York, August 1998.
- [283] R. Meo, G. Psaila, and S. Ceri. A New SQL-like Operator for Mining Association Rules. In *Proc. of the 22nd VLDB Conf.*, pages 122–133, Bombay, India, 1996.
- [284] R. J. Miller and Y. Yang. Association Rules over Interval Data. In *Proc. of 1997 ACM-SIGMOD Intl. Conf. on Management of Data*, pages 452–461, Tucson, AZ, May 1997.
- [285] Y. Morimoto, T. Fukuda, H. Matsuzawa, T. Tokuyama, and K. Yoda. Algorithms for mining association rules for binary segmentations of huge categorical databases. In *Proc. of the 24th VLDB Conf.*, pages 380–391, New York, August 1998.
- [286] F. Mosteller. Association and Estimation in Contingency Tables. *Journal of the American Statistical Association*, 63:1–28, 1968.
- [287] A. Mueller. Past sequential and parallel algorithms for association rule mining: A comparison. Technical Report CS-TR-3515, University of Maryland, August 1995.
- [288] R. T. Ng, L. V. S. Lakshmanan, J. Han, and A. Pang. Exploratory Mining and Pruning Optimizations of Constrained Association Rules. In *Proc. of 1998 ACM-SIGMOD Intl. Conf. on Management of Data*, pages 13–24, Seattle, WA, June 1998.
- [289] E. Omiecinski. Alternative Interest Measures for Mining Associations in Databases. *IEEE Trans. on Knowledge and Data Engineering*, 15(1):57–69, January/February 2003.
- [290] B. Ozden, S. Ramaswamy, and A. Silberschatz. Cyclic Association Rules. In *Proc. of the 14th Intl. Conf. on Data Eng.*, pages 412–421, Orlando, FL, February 1998.
- [291] A. Ozgur, P. N. Tan, and V. Kumar. RBA: An Integrated Framework for Regression based on Association Rules. In *Proc. of the SIAM Intl. Conf. on Data Mining*, pages 210–221, Orlando, FL, April 2004.
- [292] J. S. Park, M.-S. Chen, and P. S. Yu. An effective hash-based algorithm for mining association rules. *SIGMOD Record*, 25(2):175–186, 1995.
- [293] S. Parthasarathy and M. Coatey. Efficient Discovery of Common Substructures in Macromolecules. In *Proc. of the 2002 IEEE Intl. Conf. on Data Mining*, pages 362–369, Maebashi City, Japan, December 2002.
- [294] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In *Proc. of the 7th Intl. Conf. on Database Theory (ICDT'99)*, pages 398–416, Jerusalem, Israel, January 1999.
- [295] J. Pei, J. Han, H. J. Lu, S. Nishio, and S. Tang. H-Mine: Hyper-Structure Mining of Frequent Patterns in Large Databases. In *Proc. of the 2001 IEEE Intl. Conf. on Data Mining*, pages 441–448, San Jose, CA, November 2001.
- [296] J. Pei, J. Han, B. Mortazavi-Asl, and H. Zhu. Mining Access Patterns Efficiently from Web Logs. In *Proc. of the 4th Pacific-Asia Conf. on Knowledge Discovery and Data Mining*, pages 396–407, Kyoto, Japan, April 2000.
- [297] G. Piatetsky-Shapiro. Discovery, Analysis and Presentation of Strong Rules. In G. Piatetsky-Shapiro and W. Frawley, editors, *Knowledge Discovery in Databases*, pages 229–248. MIT Press, Cambridge, MA, 1991.
- [298] C. Potter, S. Klooster, M. Steinbach, P. N. Tan, V. Kumar, S. Shekhar, and C. Carvalho. Understanding Global Teleconnections of Climate to Regional Model Estimates of Amazon Ecosystem Carbon Fluxes. *Global Change Biology*, 10(5):693–703, 2004.
- [299] C. Potter, S. Klooster, M. Steinbach, P. N. Tan, V. Kumar, S. Shekhar, R. Myneni, and R. Nemani. Global Teleconnections of Ocean Climate to Terrestrial Carbon Flux. *J. Geophysical Research*, 108(D17), 2003.
- [300] G. D. Ramkumar, S. Ranka, and S. Tsui. Weighted Association Rules: Model and Algorithm. <http://www.cs.ucla.edu/~czdemo/tsuir/>, 1997.
- [301] S. Sarawagi, S. Thomas, and R. Agrawal. Integrating Mining with Relational Database Systems: Alternatives and Implications. In *Proc. of 1998 ACM-SIGMOD Intl. Conf. on Management of Data*, pages 343–354, Seattle, WA, 1998.
- [302] K. Satou, G. Shibayama, T. Ono, Y. Yamamura, E. Furukchi, S. Kuwara, and T. Takagi. Finding Association Rules on Heterogeneous Genome Data. In *Proc. of the Pacific Symp. on Biocomputing*, pages 397–408, Hawaii, January 1997.
- [303] A. Savasere, E. Omiecinski, and S. Navathe. An efficient algorithm for mining association rules in large databases. In *Proc. of the 21st Int. Conf. on Very Large Databases (VLDB'95)*, pages 432–444, Zurich, Switzerland, September 1995.
- [304] A. Savasere, E. Omiecinski, and S. Navathe. Mining for Strong Negative Associations in a Large Database of Customer Transactions. In *Proc. of the 14th Intl. Conf. on Data Engineering*, pages 494–502, Orlando, Florida, February 1998.
- [305] M. Seno and G. Karypis. LPMiner: An Algorithm for Finding Frequent Itemsets Using Length-Decreasing Support Constraint. In *Proc. of the 2001 IEEE Intl. Conf. on Data Mining*, pages 505–512, San Jose, CA, November 2001.
- [306] T. Shintani and M. Kitsuregawa. Hash based parallel algorithms for mining association rules. In *Proc. of the 4th Intl. Conf. on Parallel and Distributed Info. Systems*, pages 19–30, Miami Beach, FL, December 1996.
- [307] A. Silberschatz and A. Tuzhilin. What makes patterns interesting in knowledge discovery systems. *IEEE Trans. on Knowledge and Data Engineering*, 8(6):970–974, 1996.
- [308] C. Silverstein, S. Brin, and R. Motwani. Beyond market baskets: Generalizing association rules to dependence rules. *Data Mining and Knowledge Discovery*, 2(1):39–68, 1998.

- [309] E.-H. Simpson. The Interpretation of Interaction in Contingency Tables. *Journal of the Royal Statistical Society, B*(13):238–241, 1951.
- [310] L. Singh, B. Chen, R. Haight, and P. Scheuermann. An Algorithm for Constrained Association Rule Mining in Semi-structured Data. In *Proc. of the 3rd Pacific-Asia Conf. on Knowledge Discovery and Data Mining*, pages 148–158, Beijing, China, April 1999.
- [311] 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, Canada, 1996.
- [312] R. Srikant and R. Agrawal. Mining Sequential Patterns: Generalizations and Performance Improvements. In *Proc. of the 5th Intl Conf. on Extending Database Technology (EDBT'96)*, pages 18–32, Avignon, France, 1996.
- [313] R. Srikant, Q. Vu, and R. Agrawal. Mining Association Rules with Item Constraints. In *Proc. of the 3rd Intl. Conf. on Knowledge Discovery and Data Mining*, pages 67–73, Newport Beach, CA, August 1997.
- [314] M. Steinbach, P. N. Tan, and V. Kumar. Support Envelopes: A Technique for Exploring the Structure of Association Patterns. In *Proc. of the 10th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 296–305, Seattle, WA, August 2004.
- [315] M. Steinbach, P. N. Tan, H. Xiong, and V. Kumar. Extending the Notion of Support. In *Proc. of the 10th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 689–694, Seattle, WA, August 2004.
- [316] E. Suzuki. Autonomous Discovery of Reliable Exception Rules. In *Proc. of the 3rd Intl. Conf. on Knowledge Discovery and Data Mining*, pages 259–262, Newport Beach, CA, August 1997.
- [317] P. N. Tan and V. Kumar. Mining Association Patterns in Web Usage Data. In *Proc. of the Intl. Conf. on Advances in Infrastructure for e-Business, e-Education, e-Science and e-Medicine on the Internet*, L’Aquila, Italy, January 2002.
- [318] P. N. Tan, V. Kumar, and J. Srivastava. Selecting the Right Interestingness Measure for Association Patterns. In *Proc. of the 8th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 32–41, Edmonton, Canada, July 2002.
- [319] P. N. Tan, M. Steinbach, V. Kumar, S. Klooster, C. Potter, and A. Torregrosa. Finding Spatio-Temporal Patterns in Earth Science Data. In *KDD 2001 Workshop on Temporal Data Mining*, San Francisco, CA, 2001.
- [320] H. Toivonen. Sampling Large Databases for Association Rules. In *Proc. of the 22nd VLDB Conf.*, pages 134–145, Boinbay, India, 1996.
- [321] H. Toivonen, M. Klemettinen, P. Ronkainen, K. Hatonen, and H. Mannila. Pruning and Grouping Discovered Association Rules. In *ECML-95 Workshop on Statistics, Machine Learning and Knowledge Discovery in Databases*, pages 47–52, Heraklion, Greece, April 1995.
- [322] S. Tsur, J. Ullman, S. Abiteboul, C. Clifton, R. Motwani, S. Nestorov, and A. Rosenthal. Query Flocks: A Generalization of Association Rule Mining. In *Proc. of 1998 ACM-SIGMOD Intl. Conf. on Management of Data*, pages 1–12, Seattle, WA, June 1998.
- [323] A. Tung, H. J. Lu, J. Han, and L. Feng. Breaking the Barrier of Transactions: Mining Inter-Transaction Association Rules. In *Proc. of the 5th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 297–301, San Diego, CA, August 1999.
- [324] K. Wang, Y. He, and J. Han. Mining Frequent Itemsets Using Support Constraints. In *Proc. of the 26th VLDB Conf.*, pages 43–52, Cairo, Egypt, September 2000.
- [325] K. Wang, S. H. Tay, and B. Liu. Interestingness-Based Interval Merger for Numeric Association Rules. In *Proc. of the 4th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 121–128, New York, NY, August 1998.
- [326] G. I. Webb. Preliminary investigations into statistically valid exploratory rule discovery. In *Proc. of the Australasian Data Mining Workshop (AusDM03)*, Canberra, Australia, December 2003.
- [327] H. Xiong, X. He, C. Ding, Y. Zhang, V. Kumar, and S. R. Holbrook. Identification of Functional Modules in Protein Complexes via Hyperclique Pattern Discovery. In *Proc. of the Pacific Symposium on Biocomputing (PSB 2005)*, Maui, January 2005.
- [328] H. Xiong, S. Shekhar, P. N. Tan, and V. Kumar. Exploiting a Support-based Upper Bound of Pearson’s Correlation Coefficient for Efficiently Identifying Strongly Correlated Pairs. In *Proc. of the 10th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 334–343, Seattle, WA, August 2004.
- [329] H. Xiong, M. Steinbach, P. N. Tan, and V. Kumar. HICAP: Hierarchical Clustering with Pattern Preservation. In *Proc. of the SIAM Intl. Conf. on Data Mining*, pages 279–290, Orlando, FL, April 2004.
- [330] H. Xiong, P. N. Tan, and V. Kumar. Mining Strong Affinity Association Patterns in Data Sets with Skewed Support Distribution. In *Proc. of the 2003 IEEE Intl. Conf. on Data Mining*, pages 387–394, Melbourne, FL, 2003.
- [331] X. Yan and J. Han. gSpan: Graph-based Substructure Pattern Mining. In *Proc. of the 2002 IEEE Intl. Conf. on Data Mining*, pages 721–724, Maebashi City, Japan, December 2002.
- [332] C. Yang, U. M. Fayyad, and P. S. Bradley. Efficient discovery of error-tolerant frequent itemsets in high dimensions. In *Proc. of the 7th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 194–203, San Francisco, CA, August 2001.
- [333] M. J. Zaki. Parallel and Distributed Association Mining: A Survey. *IEEE Concurrency, special issue on Parallel Mechanisms for Data Mining*, 7(4):14–25, December 1999.
- [334] M. J. Zaki. Generating Non-Redundant Association Rules. In *Proc. of the 6th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 34–43, Boston, MA, August 2000.
- [335] M. J. Zaki. Efficiently mining frequent trees in a forest. In *Proc. of the 8th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 71–80, Edmonton, Canada, July 2002.
- [336] M. J. Zaki and M. Orihara. Theoretical foundations of association rules. In *Proc. of the 1998 ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, Seattle, WA, June 1998.
- [337] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New Algorithms for Fast Discovery of Association Rules. In *Proc. of the 3rd Intl. Conf. on Knowledge Discovery and Data Mining*, pages 283–286, Newport Beach, CA, August 1997.
- [338] H. Zhang, B. Padmanabhan, and A. Tuzbilin. On the Discovery of Significant Statistical Quantitative Rules. In *Proc. of the 10th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 374–383, Seattle, WA, August 2004.
- [339] Z. Zhang, Y. Lu, and B. Zhang. An Effective Partitioning-Combining Algorithm for Discovering Quantitative Association Rules. In *Proc. of the 1st Pacific-Asia Conf. on Knowledge Discovery and Data Mining*, Singapore, 1997.
- [340] N. Zhong, Y. Y. Yao, and S. Ohsuga. Peculiarity Oriented Multi-database Mining. In *Proc. of the 3rd European Conf. of Principles and Practice of Knowledge Discovery in Databases*, pages 136–146, Prague, Czech Republic, 1999.

6.10 Exercises

- For each of the following questions, provide an example of an association rule from the market basket domain that satisfies the following conditions. Also, describe whether such rules are subjectively interesting.
 - A rule that has high support and high confidence.
 - A rule that has reasonably high support but low confidence.
 - A rule that has low support and low confidence.
 - A rule that has low support and high confidence.
- Consider the data set shown in Table 6.22.

Table 6.22. Example of market basket transactions.

Customer ID	Transaction ID	Items Bought
1	0001	{a, d, e}
1	0024	{a, b, c, e}
2	0012	{a, b, d, e}
2	0031	{a, c, d, e}
3	0015	{b, c, e}
3	0022	{b, d, e}
4	0029	{c, d}
4	0040	{a, b, c}
5	0033	{a, d, e}
5	0038	{a, b, e}

- Compute the support for itemsets $\{e\}$, $\{b, d\}$, and $\{b, d, e\}$ by treating each transaction ID as a market basket.
- Use the results in part (a) to compute the confidence for the association rules $\{b, d\} \rightarrow \{e\}$ and $\{e\} \rightarrow \{b, d\}$. Is confidence a symmetric measure?
- Repeat part (a) by treating each customer ID as a market basket. Each item should be treated as a binary variable (1 if an item appears in at least one transaction bought by the customer, and 0 otherwise.)
- Use the results in part (c) to compute the confidence for the association rules $\{b, d\} \rightarrow \{e\}$ and $\{e\} \rightarrow \{b, d\}$.
- Suppose s_1 and c_1 are the support and confidence values of an association rule r when treating each transaction ID as a market basket. Also, let s_2 and c_2 be the support and confidence values of r when treating each customer ID as a market basket. Discuss whether there are any relationships between s_1 and s_2 or c_1 and c_2 .

- (a) What is the confidence for the rules $\emptyset \rightarrow A$ and $A \rightarrow \emptyset$?
 - Let c_1 , c_2 , and c_3 be the confidence values of the rules $\{p\} \rightarrow \{q\}$, $\{p\} \rightarrow \{q, r\}$, and $\{p, r\} \rightarrow \{q\}$, respectively. If we assume that c_1 , c_2 , and c_3 have different values, what are the possible relationships that may exist among c_1 , c_2 , and c_3 ? Which rule has the lowest confidence?
 - Repeat the analysis in part (b) assuming that the rules have identical support. Which rule has the highest confidence?
 - Transitivity: Suppose the confidence of the rules $A \rightarrow B$ and $B \rightarrow C$ are larger than some threshold, $minconf$. Is it possible that $A \rightarrow C$ has a confidence less than $minconf$?
- For each of the following measures, determine whether it is monotone, anti-monotone, or non-monotone (i.e., neither monotone nor anti-monotone).

Example: Support, $s = \frac{\sigma(X)}{|T|}$ is anti-monotone because $s(X) \geq s(Y)$ whenever $X \subset Y$.

- A characteristic rule is a rule of the form $\{p\} \rightarrow \{q_1, q_2, \dots, q_n\}$, where the rule antecedent contains only a single item. An itemset of size k can produce up to k characteristic rules. Let ζ be the minimum confidence of all characteristic rules generated from a given itemset:
- $$\zeta(\{p_1, p_2, \dots, p_k\}) = \min [c(\{p_1\} \rightarrow \{p_2, p_3, \dots, p_k\}), \dots, c(\{p_k\} \rightarrow \{p_1, p_2, \dots, p_{k-1}\})]$$
- Is ζ monotone, anti-monotone, or non-monotone?
- A discriminant rule is a rule of the form $\{p_1, p_2, \dots, p_n\} \rightarrow \{q\}$, where the rule consequent contains only a single item. An itemset of size k can produce up to k discriminant rules. Let η be the minimum confidence of all discriminant rules generated from a given itemset:

$$\eta(\{p_1, p_2, \dots, p_k\}) = \min [c(\{p_2, p_3, \dots, p_k\} \rightarrow \{p_1\}), \dots, c(\{p_1, p_2, \dots, p_{k-1}\} \rightarrow \{p_k\})]$$

- Is η monotone, anti-monotone, or non-monotone?
- Repeat the analysis in parts (a) and (b) by replacing the min function with a max function.

- Prove Equation 6.3. (Hint: First, count the number of ways to create an itemset that forms the left hand side of the rule. Next, for each size k itemset selected for the left-hand side, count the number of ways to choose the remaining $d - k$ items to form the right-hand side of the rule.)

Table 6.23. Market basket transactions.

Transaction ID	Items Bought
1	{Milk, Beer, Diapers}
2	{Bread, Butter, Milk}
3	{Milk, Diapers, Cookies}
4	{Bread, Butter, Cookies}
5	{Beer, Cookies, Diapers}
6	{Milk, Diapers, Bread, Butter}
7	{Bread, Butter, Diapers}
8	{Beer, Diapers}
9	{Milk, Diapers, Bread, Butter}
10	{Beer, Cookies}

6. Consider the market basket transactions shown in Table 6.23.

- (a) What is the maximum number of association rules that can be extracted from this data (including rules that have zero support)?
- (b) What is the maximum size of frequent itemsets that can be extracted (assuming $minsup > 0$)?
- (c) Write an expression for the maximum number of size-3 itemsets that can be derived from this data set.
- (d) Find an itemset (of size 2 or larger) that has the largest support.
- (e) Find a pair of items, a and b , such that the rules $\{a\} \rightarrow \{b\}$ and $\{b\} \rightarrow \{a\}$ have the same confidence.

7. Consider the following set of frequent 3-itemsets:

$$\{1, 2, 3\}, \{1, 2, 4\}, \{1, 2, 5\}, \{1, 3, 4\}, \{1, 3, 5\}, \{2, 3, 4\}, \{2, 3, 5\}, \{3, 4, 5\}.$$

Assume that there are only five items in the data set.

- (a) List all candidate 4-itemsets obtained by a candidate generation procedure using the $F_{k-1} \times F_1$ merging strategy.
 - (b) List all candidate 4-itemsets obtained by the candidate generation procedure in *Apriori*.
 - (c) List all candidate 4-itemsets that survive the candidate pruning step of the *Apriori* algorithm.
8. The *Apriori* algorithm uses a generate-and-count strategy for deriving frequent itemsets. Candidate itemsets of size $k+1$ are created by joining a pair of frequent itemsets of size k (this is known as the candidate generation step). A candidate is discarded if any one of its subsets is found to be infrequent during the candidate pruning step. Suppose the *Apriori* algorithm is applied to the

Table 6.24. Example of market basket transactions.

Transaction ID	Items Bought
1	{a, b, d, e}
2	{b, c, d}
3	{a, b, d, e}
4	{a, c, d, e}
5	{b, c, d, e}
6	{b, d, e}
7	{c, d}
8	{a, b, c}
9	{a, d, e}
10	{b, d}

data set shown in Table 6.24 with $minsup = 30\%$, i.e., any itemset occurring in less than 3 transactions is considered to be infrequent.

- (a) Draw an itemset lattice representing the data set given in Table 6.24. Label each node in the lattice with the following letter(s):
 - N: If the itemset is not considered to be a candidate itemset by the *Apriori* algorithm. There are two reasons for an itemset not to be considered as a candidate itemset: (1) it is not generated at all during the candidate generation step, or (2) it is generated during the candidate generation step but is subsequently removed during the candidate pruning step because one of its subsets is found to be infrequent.
 - F: If the candidate itemset is found to be frequent by the *Apriori* algorithm.
 - I: If the candidate itemset is found to be infrequent after support counting.
 - (b) What is the percentage of frequent itemsets (with respect to all itemsets in the lattice)?
 - (c) What is the pruning ratio of the *Apriori* algorithm on this data set? (Pruning ratio is defined as the percentage of itemsets not considered to be a candidate because (1) they are not generated during candidate generation or (2) they are pruned during the candidate pruning step.)
 - (d) What is the false alarm rate (i.e., percentage of candidate itemsets that are found to be infrequent after performing support counting)?
9. The *Apriori* algorithm uses a hash tree data structure to efficiently count the support of candidate itemsets. Consider the hash tree for candidate 3-itemsets shown in Figure 6.32.

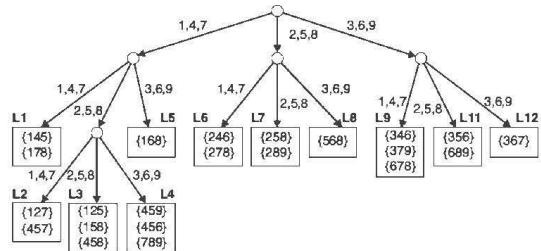


Figure 6.32. An example of a hash tree structure.

- (a) Given a transaction that contains items $\{1, 3, 4, 5, 8\}$, which of the hash tree leaf nodes will be visited when finding the candidates of the transaction?
- (b) Use the visited leaf nodes in part (b) to determine the candidate itemsets that are contained in the transaction $\{1, 3, 4, 5, 8\}$.
10. Consider the following set of candidate 3-itemsets:

$$\{1, 2, 3\}, \{1, 2, 6\}, \{1, 3, 4\}, \{2, 3, 4\}, \{2, 4, 5\}, \{3, 4, 6\}, \{4, 5, 6\}$$

- (a) Construct a hash tree for the above candidate 3-itemsets. Assume the tree uses a hash function where all odd-numbered items are hashed to the left child of a node, while the even-numbered items are hashed to the right child. A candidate k -itemset is inserted into the tree by hashing on each successive item in the candidate and then following the appropriate branch of the tree according to the hash value. Once a leaf node is reached, the candidate is inserted based on one of the following conditions:

Condition 1: If the depth of the leaf node is equal to k (the root is assumed to be at depth 0), then the candidate is inserted regardless of the number of itemsets already stored at the node.

Condition 2: If the depth of the leaf node is less than k , then the candidate can be inserted as long as the number of itemsets stored at the node is less than $maxsize$. Assume $maxsize = 2$ for this question.

Condition 3: If the depth of the leaf node is less than k and the number of itemsets stored at the node is equal to $maxsize$, then the leaf node is converted into an internal node. New leaf nodes are created as children of the old leaf node. Candidate itemsets previously stored

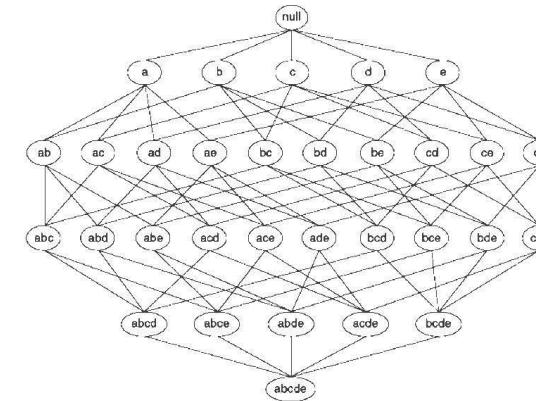


Figure 6.33. An itemset lattice

in the old leaf node are distributed to the children based on their hash values. The new candidate is also hashed to its appropriate leaf node.

- (b) How many leaf nodes are there in the candidate hash tree? How many internal nodes are there?
- (c) Consider a transaction that contains the following items: $\{1, 2, 3, 5, 6\}$. Using the hash tree constructed in part (a), which leaf nodes will be checked against the transaction? What are the candidate 3-itemsets contained in the transaction?

11. Given the lattice structure shown in Figure 6.33 and the transactions given in Table 6.24, label each node with the following letter(s):

- M if the node is a maximal frequent itemset,
- C if it is a closed frequent itemset,
- N if it is frequent but neither maximal nor closed, and
- I if it is infrequent.

Assume that the support threshold is equal to 30%.

12. The original association rule mining formulation uses the support and confidence measures to prune uninteresting rules.

- (a) Draw a contingency table for each of the following rules using the transactions shown in Table 6.25.

Table 6.25. Example of market basket transactions.

Transaction ID	Items Bought
1	{a, b, d, e}
2	{b, c, d}
3	{a, b, d, e}
4	{a, c, d, e}
5	{b, c, d, e}
6	{b, d, e}
7	{c, d}
8	{a, b, c}
9	{a, d, e}
10	{b, d}

Rules: {b} \rightarrow {c}, {a} \rightarrow {d}, {b} \rightarrow {d}, {e} \rightarrow {c}, {c} \rightarrow {a}.

- (b) Use the contingency tables in part (a) to compute and rank the rules in decreasing order according to the following measures.
- Support.
 - Confidence.
 - Interest($X \rightarrow Y$) = $\frac{P(X,Y)}{P(X)}P(Y)$.
 - IS($X \rightarrow Y$) = $\frac{P(X,Y)}{\sqrt{P(X)P(Y)}}$.
 - Klosgen($X \rightarrow Y$) = $\sqrt{P(X,Y) \times (P(Y|X) - P(Y))}$, where $P(Y|X) = \frac{P(X,Y)}{P(X)}$.
 - Odds ratio($X \rightarrow Y$) = $\frac{P(X,Y)P(X,Y)}{P(X,Y)P(X,Y)}$.
13. Given the rankings you had obtained in Exercise 12, compute the correlation between the rankings of confidence and the other five measures. Which measure is most highly correlated with confidence? Which measure is least correlated with confidence?
14. Answer the following questions using the data sets shown in Figure 6.34. Note that each data set contains 1000 items and 10,000 transactions. Dark cells indicate the presence of items and white cells indicate the absence of items. We will apply the *Apriori* algorithm to extract frequent itemsets with *minsup* = 10% (i.e., itemsets must be contained in at least 1000 transactions)?
- (a) Which data set(s) will produce the most number of frequent itemsets?

- (b) Which data set(s) will produce the fewest number of frequent itemsets?
(c) Which data set(s) will produce the longest frequent itemset?
(d) Which data set(s) will produce frequent itemsets with highest maximum support?
(e) Which data set(s) will produce frequent itemsets containing items with wide-varying support levels (i.e., items with mixed support, ranging from less than 20% to more than 70%).

15. (a) Prove that the ϕ coefficient is equal to 1 if and only if $f_{11} = f_{1+} = f_{+1}$.
(b) Show that if A and B are independent, then $P(A, B) \times P(A, \bar{B}) = P(A, B) \times P(\bar{A}, B)$.
(c) Show that Yule's Q and Y coefficients

$$Q = \frac{[f_{11}f_{00} - f_{10}f_{01}]}{[f_{11}f_{00} + f_{10}f_{01}]}$$

$$Y = \frac{[\sqrt{f_{11}f_{00}} - \sqrt{f_{10}f_{01}}]}{[\sqrt{f_{11}f_{00}} + \sqrt{f_{10}f_{01}}]}$$

are normalized versions of the odds ratio.

- (d) Write a simplified expression for the value of each measure shown in Tables 6.11 and 6.12 when the variables are statistically independent.
16. Consider the interestingness measure, $M = \frac{P(B|A) - P(B)}{1 - P(B)}$, for an association rule $A \rightarrow B$.
- What is the range of this measure? When does the measure attain its maximum and minimum values?
 - How does M behave when $P(A, B)$ is increased while $P(A)$ and $P(B)$ remain unchanged?
 - How does M behave when $P(A)$ is increased while $P(A, B)$ and $P(B)$ remain unchanged?
 - How does M behave when $P(B)$ is increased while $P(A, B)$ and $P(A)$ remain unchanged?
 - Is the measure symmetric under variable permutation?
 - What is the value of the measure when A and B are statistically independent?
 - Is the measure null-invariant?
 - Does the measure remain invariant under row or column scaling operations?
 - How does the measure behave under the inversion operation?

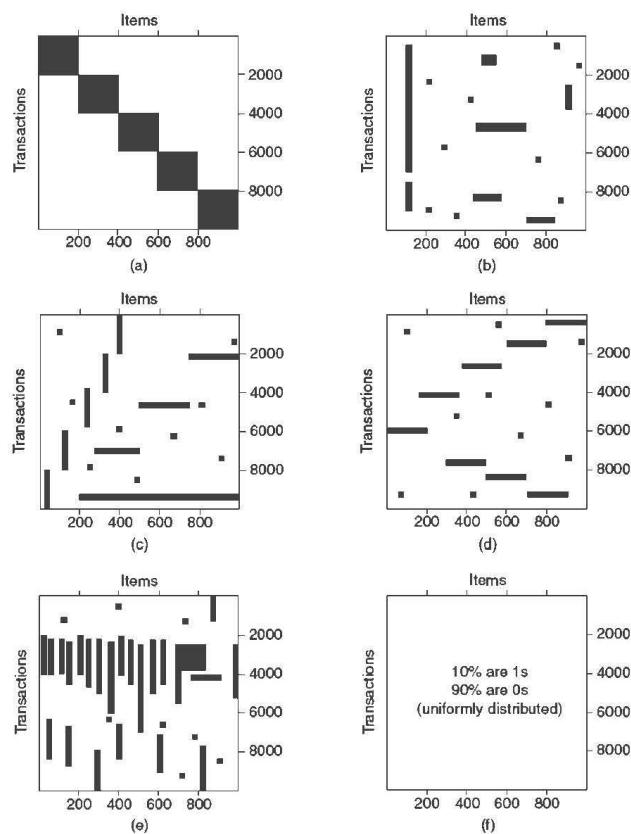


Figure 6.34. Figures for Exercise 14.

17. Suppose we have market basket data consisting of 100 transactions and 20 items. If the support for item a is 25%, the support for item b is 90% and the support for itemset $\{a, b\}$ is 20%. Let the support and confidence thresholds be 10% and 60%, respectively.

- (a) Compute the confidence of the association rule $\{a\} \rightarrow \{b\}$. Is the rule interesting according to the confidence measure?
- (b) Compute the interest measure for the association pattern $\{a, b\}$. Describe the nature of the relationship between item a and item b in terms of the interest measure.
- (c) What conclusions can you draw from the results of parts (a) and (b)?
- (d) Prove that if the confidence of the rule $\{a\} \rightarrow \{b\}$ is less than the support of $\{b\}$, then:
 - i. $c(\{a\} \rightarrow \{b\}) > c(\{\bar{a}\} \rightarrow \{\bar{b}\})$,
 - ii. $c(\{a\} \rightarrow \{b\}) > s(\{b\})$,
 where $c(\cdot)$ denote the rule confidence and $s(\cdot)$ denote the support of an itemset.

18. Table 6.26 shows a $2 \times 2 \times 2$ contingency table for the binary variables A and B at different values of the control variable C .

Table 6.26. A Contingency Table.

		A	
		1	0
C = 0	B	1	0
	0	15	30
C = 1	B	1	5
	0	0	15

- (a) Compute the ϕ coefficient for A and B when $C = 0$, $C = 1$, and $C = 0$ or 1. Note that $\phi(\{A, B\}) = \frac{P(A, B) - P(A)P(B)}{\sqrt{P(A)P(B)(1-P(A))(1-P(B))}}$.
- (b) What conclusions can you draw from the above result?

19. Consider the contingency tables shown in Table 6.27.

- (a) For table I, compute support, the interest measure, and the ϕ correlation coefficient for the association pattern $\{A, B\}$. Also, compute the confidence of rules $A \rightarrow B$ and $B \rightarrow A$.

Table 6.27. Contingency tables for Exercise 19.

	B	\bar{B}		B	\bar{B}
A	9	1	A	89	1
\bar{A}	1	89	\bar{A}	1	9

(a) Table I. (b) Table II.

- (b) For table II, compute support, the interest measure, and the ϕ correlation coefficient for the association pattern $\{A, B\}$. Also, compute the confidence of rules $A \rightarrow B$ and $B \rightarrow A$.
- (c) What conclusions can you draw from the results of (a) and (b)?
20. Consider the relationship between customers who buy high-definition televisions and exercise machines as shown in Tables 6.19 and 6.20.
- (a) Compute the odds ratios for both tables.
 - (b) Compute the ϕ -coefficient for both tables.
 - (c) Compute the interest factor for both tables.

For each of the measures given above, describe how the direction of association changes when data is pooled together instead of being stratified.

Cluster Analysis: Basic Concepts and Algorithms

Cluster analysis divides data into groups (clusters) that are meaningful, useful, or both. If meaningful groups are the goal, then the clusters should capture the natural structure of the data. In some cases, however, cluster analysis is only a useful starting point for other purposes, such as data summarization. Whether for understanding or utility, cluster analysis has long played an important role in a wide variety of fields: psychology and other social sciences, biology, statistics, pattern recognition, information retrieval, machine learning, and data mining.

There have been many applications of cluster analysis to practical problems. We provide some specific examples, organized by whether the purpose of the clustering is understanding or utility.

Clustering for Understanding Classes, or conceptually meaningful groups of objects that share common characteristics, play an important role in how people analyze and describe the world. Indeed, human beings are skilled at dividing objects into groups (clustering) and assigning particular objects to these groups (classification). For example, even relatively young children can quickly label the objects in a photograph as buildings, vehicles, people, animals, plants, etc. In the context of understanding data, clusters are potential classes and cluster analysis is the study of techniques for automatically finding classes. The following are some examples:

- **Biology.** Biologists have spent many years creating a taxonomy (hierarchical classification) of all living things: kingdom, phylum, class, order, family, genus, and species. Thus, it is perhaps not surprising that much of the early work in cluster analysis sought to create a discipline of mathematical taxonomy that could automatically find such classification structures. More recently, biologists have applied clustering to analyze the large amounts of genetic information that are now available. For example, clustering has been used to find groups of genes that have similar functions.
- **Information Retrieval.** The World Wide Web consists of billions of Web pages, and the results of a query to a search engine can return thousands of pages. Clustering can be used to group these search results into a small number of clusters, each of which captures a particular aspect of the query. For instance, a query of “movie” might return Web pages grouped into categories such as reviews, trailers, stars, and theaters. Each category (cluster) can be broken into subcategories (sub-clusters), producing a hierarchical structure that further assists a user’s exploration of the query results.
- **Climate.** Understanding the Earth’s climate requires finding patterns in the atmosphere and ocean. To that end, cluster analysis has been applied to find patterns in the atmospheric pressure of polar regions and areas of the ocean that have a significant impact on land climate.
- **Psychology and Medicine.** An illness or condition frequently has a number of variations, and cluster analysis can be used to identify these different subcategories. For example, clustering has been used to identify different types of depression. Cluster analysis can also be used to detect patterns in the spatial or temporal distribution of a disease.
- **Business.** Businesses collect large amounts of information on current and potential customers. Clustering can be used to segment customers into a small number of groups for additional analysis and marketing activities.

Clustering for Utility Cluster analysis provides an abstraction from individual data objects to the clusters in which those data objects reside. Additionally, some clustering techniques characterize each cluster in terms of a cluster prototype; i.e., a data object that is representative of the other objects in the cluster. These cluster prototypes can be used as the basis for a

number of data analysis or data processing techniques. Therefore, in the context of utility, cluster analysis is the study of techniques for finding the most representative cluster prototypes.

- **Summarization.** Many data analysis techniques, such as regression or PCA, have a time or space complexity of $O(m^2)$ or higher (where m is the number of objects), and thus, are not practical for large data sets. However, instead of applying the algorithm to the entire data set, it can be applied to a reduced data set consisting only of cluster prototypes. Depending on the type of analysis, the number of prototypes, and the accuracy with which the prototypes represent the data, the results can be comparable to those that would have been obtained if all the data could have been used.
- **Compression.** Cluster prototypes can also be used for data compression. In particular, a table is created that consists of the prototypes for each cluster; i.e., each prototype is assigned an integer value that is its position (index) in the table. Each object is represented by the index of the prototype associated with its cluster. This type of compression is known as **vector quantization** and is often applied to image, sound, and video data, where (1) many of the data objects are highly similar to one another, (2) some loss of information is acceptable, and (3) a substantial reduction in the data size is desired.
- **Efficiently Finding Nearest Neighbors.** Finding nearest neighbors can require computing the pairwise distance between all points. Often clusters and their cluster prototypes can be found much more efficiently. If objects are relatively close to the prototype of their cluster, then we can use the prototypes to reduce the number of distance computations that are necessary to find the nearest neighbors of an object. Intuitively, if two cluster prototypes are far apart, then the objects in the corresponding clusters cannot be nearest neighbors of each other. Consequently, to find an object’s nearest neighbors it is only necessary to compute the distance to objects in nearby clusters, where the nearness of two clusters is measured by the distance between their prototypes. This idea is made more precise in Exercise 25 on page 94.

This chapter provides an introduction to cluster analysis. We begin with a high-level overview of clustering, including a discussion of the various approaches to dividing objects into sets of clusters and the different types of clusters. We then describe three specific clustering techniques that represent

broad categories of algorithms and illustrate a variety of concepts: K-means, agglomerative hierarchical clustering, and DBSCAN. The final section of this chapter is devoted to cluster validity—methods for evaluating the goodness of the clusters produced by a clustering algorithm. More advanced clustering concepts and algorithms will be discussed in Chapter 9. Whenever possible, we discuss the strengths and weaknesses of different schemes. In addition, the bibliographic notes provide references to relevant books and papers that explore cluster analysis in greater depth.

8.1 Overview

Before discussing specific clustering techniques, we provide some necessary background. First, we further define cluster analysis, illustrating why it is difficult and explaining its relationship to other techniques that group data. Then we explore two important topics: (1) different ways to group a set of objects into a set of clusters, and (2) types of clusters.

8.1.1 What Is Cluster Analysis?

Cluster analysis groups data objects based only on information found in the data that describes the objects and their relationships. The goal is that the objects within a group be similar (or related) to one another and different from (or unrelated to) the objects in other groups. The greater the similarity (or homogeneity) within a group and the greater the difference between groups, the better or more distinct the clustering.

In many applications, the notion of a cluster is not well defined. To better understand the difficulty of deciding what constitutes a cluster, consider Figure 8.1, which shows twenty points and three different ways of dividing them into clusters. The shapes of the markers indicate cluster membership. Figures 8.1(b) and 8.1(d) divide the data into two and six parts, respectively. However, the apparent division of each of the two larger clusters into three subclusters may simply be an artifact of the human visual system. Also, it may not be unreasonable to say that the points form four clusters, as shown in Figure 8.1(c). This figure illustrates that the definition of a cluster is imprecise and that the best definition depends on the nature of data and the desired results.

Cluster analysis is related to other techniques that are used to divide data objects into groups. For instance, clustering can be regarded as a form of classification in that it creates a labeling of objects with class (cluster) labels. However, it derives these labels only from the data. In contrast, classification

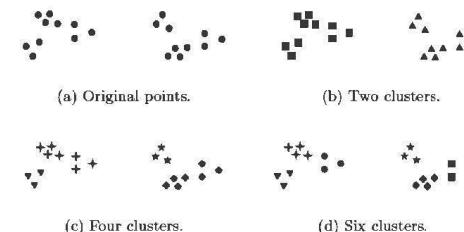


Figure 8.1. Different ways of clustering the same set of points.

in the sense of Chapter 4 is supervised classification; i.e., new, unlabeled objects are assigned a class label using a model developed from objects with known class labels. For this reason, cluster analysis is sometimes referred to as unsupervised classification. When the term classification is used without any qualification within data mining, it typically refers to supervised classification.

Also, while the terms segmentation and partitioning are sometimes used as synonyms for clustering, these terms are frequently used for approaches outside the traditional bounds of cluster analysis. For example, the term partitioning is often used in connection with techniques that divide graphs into subgraphs and that are not strongly connected to clustering. Segmentation often refers to the division of data into groups using simple techniques; e.g., an image can be split into segments based only on pixel intensity and color, or people can be divided into groups based on their income. Nonetheless, some work in graph partitioning and in image and market segmentation is related to cluster analysis.

8.1.2 Different Types of Clusterings

An entire collection of clusters is commonly referred to as a clustering, and in this section, we distinguish various types of clusterings: hierarchical (nested) versus partitional (unnested), exclusive versus overlapping versus fuzzy, and complete versus partial.

Hierarchical versus Partitional The most commonly discussed distinction among different types of clusterings is whether the set of clusters is nested

or unnested, or in more traditional terminology, hierarchical or partitional. A **partitional clustering** is simply a division of the set of data objects into non-overlapping subsets (clusters) such that each data object is in exactly one subset. Taken individually, each collection of clusters in Figures 8.1 (b–d) is a partitional clustering.

If we permit clusters to have subclusters, then we obtain a **hierarchical clustering**, which is a set of nested clusters that are organized as a tree. Each node (cluster) in the tree (except for the leaf nodes) is the union of its children (subclusters), and the root of the tree is the cluster containing all the objects. Often, but not always, the leaves of the tree are singleton clusters of individual data objects. If we allow clusters to be nested, then one interpretation of Figure 8.1(a) is that it has two subclusters (Figure 8.1(b)), each of which, in turn, has three subclusters (Figure 8.1(d)). The clusters shown in Figures 8.1 (a–d), when taken in that order, also form a hierarchical (nested) clustering with, respectively, 1, 2, 4, and 6 clusters on each level. Finally, note that a hierarchical clustering can be viewed as a sequence of partitional clusterings and a partitional clustering can be obtained by taking any member of that sequence; i.e., by cutting the hierarchical tree at a particular level.

Exclusive versus Overlapping versus Fuzzy The clusterings shown in Figure 8.1 are all exclusive, as they assign each object to a single cluster. There are many situations in which a point could reasonably be placed in more than one cluster, and these situations are better addressed by non-exclusive clustering. In the most general sense, an **overlapping** or **non-exclusive** clustering is used to reflect the fact that an object can *simultaneously* belong to more than one group (class). For instance, a person at a university can be both an enrolled student and an employee of the university. A non-exclusive clustering is also often used when, for example, an object is “between” two or more clusters and could reasonably be assigned to any of these clusters. Imagine a point halfway between two of the clusters of Figure 8.1. Rather than make a somewhat arbitrary assignment of the object to a single cluster, it is placed in all of the “equally good” clusters.

In a **fuzzy clustering**, every object belongs to every cluster with a membership weight that is between 0 (absolutely doesn’t belong) and 1 (absolutely belongs). In other words, clusters are treated as fuzzy sets. (Mathematically, a fuzzy set is one in which an object belongs to any set with a weight that is between 0 and 1. In fuzzy clustering, we often impose the additional constraint that the sum of the weights for each object must equal 1.) Similarly, probabilistic clustering techniques compute the probability with which each

point belongs to each cluster, and these probabilities must also sum to 1. Because the membership weights or probabilities for any object sum to 1, a fuzzy or probabilistic clustering does not address true multiclass situations, such as the case of a student employee, where an object belongs to multiple classes. Instead, these approaches are most appropriate for avoiding the arbitrariness of assigning an object to only one cluster when it may be close to several. In practice, a fuzzy or probabilistic clustering is often converted to an exclusive clustering by assigning each object to the cluster in which its membership weight or probability is highest.

Complete versus Partial A **complete clustering** assigns every object to a cluster, whereas a **partial clustering** does not. The motivation for a partial clustering is that some objects in a data set may not belong to well-defined groups. Many times objects in the data set may represent noise, outliers, or “uninteresting background.” For example, some newspaper stories may share a common theme, such as global warming, while other stories are more generic or one-of-a-kind. Thus, to find the important topics in last month’s stories, we may want to search only for clusters of documents that are tightly related by a common theme. In other cases, a complete clustering of the objects is desired. For example, an application that uses clustering to organize documents for browsing needs to guarantee that all documents can be browsed.

8.1.3 Different Types of Clusters

Clustering aims to find useful groups of objects (clusters), where usefulness is defined by the goals of the data analysis. Not surprisingly, there are several different notions of a cluster that prove useful in practice. In order to visually illustrate the differences among these types of clusters, we use two-dimensional points, as shown in Figure 8.2, as our data objects. We stress, however, that the types of clusters described here are equally valid for other kinds of data.

Well-Separated A **cluster** is a set of objects in which each object is closer (or more similar) to every other object in the cluster than to any object not in the cluster. Sometimes a threshold is used to specify that all the objects in a cluster must be sufficiently close (or similar) to one another. This idealistic definition of a cluster is satisfied only when the data contains natural clusters that are quite far from each other. Figure 8.2(a) gives an example of well-separated clusters that consists of two groups of points in a two-dimensional space. The distance between any two points in different groups is larger than

the distance between any two points within a group. Well-separated clusters do not need to be globular, but can have any shape.

Prototype-Based A cluster is a set of objects in which each object is closer (more similar) to the prototype that defines the cluster than to the prototype of any other cluster. For data with continuous attributes, the prototype of a cluster is often a centroid, i.e., the average (mean) of all the points in the cluster. When a centroid is not meaningful, such as when the data has categorical attributes, the prototype is often a medoid, i.e., the most representative point of a cluster. For many types of data, the prototype can be regarded as the most central point, and in such instances, we commonly refer to prototype-based clusters as center-based clusters. Not surprisingly, such clusters tend to be globular. Figure 8.2(b) shows an example of center-based clusters.

Graph-Based If the data is represented as a graph, where the nodes are objects and the links represent connections among objects (see Section 2.1.2), then a cluster can be defined as a connected component; i.e., a group of objects that are connected to one another, but that have no connection to objects outside the group. An important example of graph-based clusters are contiguity-based clusters, where two objects are connected only if they are within a specified distance of each other. This implies that each object in a contiguity-based cluster is closer to some other object in the cluster than to any point in a different cluster. Figure 8.2(c) shows an example of such clusters for two-dimensional points. This definition of a cluster is useful when clusters are irregular or intertwined, but can have trouble when noise is present since, as illustrated by the two spherical clusters of Figure 8.2(c), a small bridge of points can merge two distinct clusters.

Other types of graph-based clusters are also possible. One such approach (Section 8.3.2) defines a cluster as a clique; i.e., a set of nodes in a graph that are completely connected to each other. Specifically, if we add connections between objects in the order of their distance from one another, a cluster is formed when a set of objects forms a clique. Like prototype-based clusters, such clusters tend to be globular.

Density-Based A cluster is a dense region of objects that is surrounded by a region of low density. Figure 8.2(d) shows some density-based clusters for data created by adding noise to the data of Figure 8.2(c). The two circular clusters are not merged, as in Figure 8.2(c), because the bridge between them fades into the noise. Likewise, the curve that is present in Figure 8.2(c) also

fades into the noise and does not form a cluster in Figure 8.2(d). A density-based definition of a cluster is often employed when the clusters are irregular or intertwined, and when noise and outliers are present. By contrast, a contiguity-based definition of a cluster would not work well for the data of Figure 8.2(d) since the noise would tend to form bridges between clusters.

Shared-Property (Conceptual Clusters) More generally, we can define a cluster as a set of objects that share some property. This definition encompasses all the previous definitions of a cluster; e.g., objects in a center-based cluster share the property that they are all closest to the same centroid or medoid. However, the shared-property approach also includes new types of clusters. Consider the clusters shown in Figure 8.2(e). A triangular area (cluster) is adjacent to a rectangular one, and there are two intertwined circles (clusters). In both cases, a clustering algorithm would need a very specific concept of a cluster to successfully detect these clusters. The process of finding such clusters is called conceptual clustering. However, too sophisticated a notion of a cluster would take us into the area of pattern recognition, and thus, we only consider simpler types of clusters in this book.

Road Map

In this chapter, we use the following three simple, but important techniques to introduce many of the concepts involved in cluster analysis.

- **K-means.** This is a prototype-based, partitional clustering technique that attempts to find a user-specified number of clusters (K), which are represented by their centroids.
- **Agglomerative Hierarchical Clustering.** This clustering approach refers to a collection of closely related clustering techniques that produce a hierarchical clustering by starting with each point as a singleton cluster and then repeatedly merging the two closest clusters until a single, all-encompassing cluster remains. Some of these techniques have a natural interpretation in terms of graph-based clustering, while others have an interpretation in terms of a prototype-based approach.
- **DBSCAN.** This is a density-based clustering algorithm that produces a partitional clustering, in which the number of clusters is automatically determined by the algorithm. Points in low-density regions are classified as noise and omitted; thus, DBSCAN does not produce a complete clustering.

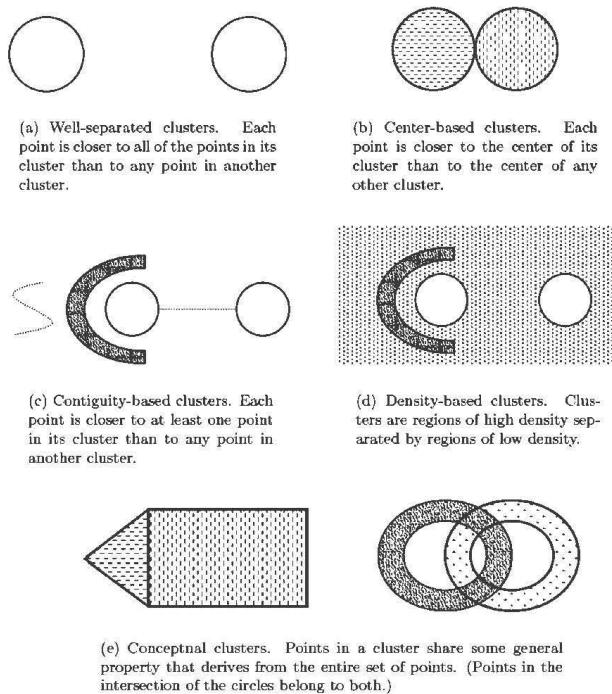


Figure 8.2. Different types of clusters as illustrated by sets of two-dimensional points.

8.2 K-means

Prototype-based clustering techniques create a one-level partitioning of the data objects. There are a number of such techniques, but two of the most prominent are K-means and K-medoid. K-means defines a prototype in terms of a centroid, which is usually the mean of a group of points, and is typically

applied to objects in a continuous n -dimensional space. K-medoid defines a prototype in terms of a medoid, which is the most representative point for a group of points, and can be applied to a wide range of data since it requires only a proximity measure for a pair of objects. While a centroid almost never corresponds to an actual data point, a medoid, by its definition, must be an actual data point. In this section, we will focus solely on K-means, which is one of the oldest and most widely used clustering algorithms.

8.2.1 The Basic K-means Algorithm

The K-means clustering technique is simple, and we begin with a description of the basic algorithm. We first choose K initial centroids, where K is a user-specified parameter, namely, the number of clusters desired. Each point is then assigned to the closest centroid, and each collection of points assigned to a centroid is a cluster. The centroid of each cluster is then updated based on the points assigned to the cluster. We repeat the assignment and update steps until no point changes clusters, or equivalently, until the centroids remain the same.

K-means is formally described by Algorithm 8.1. The operation of K-means is illustrated in Figure 8.3, which shows how, starting from three centroids, the final clusters are found in four assignment-update steps. In these and other figures displaying K-means clustering, each subfigure shows (1) the centroids at the start of the iteration and (2) the assignment of the points to those centroids. The centroids are indicated by the “+” symbol; all points belonging to the same cluster have the same marker shape.

Algorithm 8.1 Basic K-means algorithm.

- 1: Select K points as initial centroids.
 - 2: repeat
 - 3: Form K clusters by assigning each point to its closest centroid.
 - 4: Recompute the centroid of each cluster.
 - 5: until Centroids do not change.
-

In the first step, shown in Figure 8.3(a), points are assigned to the initial centroids, which are all in the larger group of points. For this example, we use the mean as the centroid. After points are assigned to a centroid, the centroid is then updated. Again, the figure for each step shows the centroid at the beginning of the step and the assignment of points to those centroids. In the second step, points are assigned to the updated centroids, and the centroids

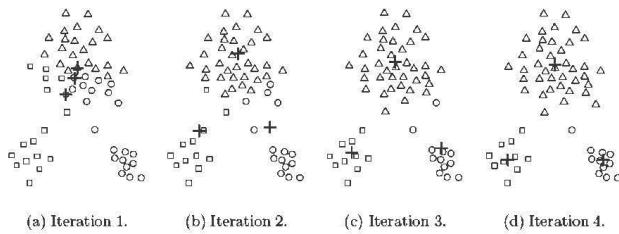


Figure 8.3. Using the K-means algorithm to find three clusters in sample data.

are updated again. In steps 2, 3, and 4, which are shown in Figures 8.3 (b), (c), and (d), respectively, two of the centroids move to the two small groups of points at the bottom of the figures. When the K-means algorithm terminates in Figure 8.3(d), because no more changes occur, the centroids have identified the natural groupings of points.

For some combinations of proximity functions and types of centroids, K-means always converges to a solution; i.e., K-means reaches a state in which no points are shifting from one cluster to another, and hence, the centroids don't change. Because most of the convergence occurs in the early steps, however, the condition on line 5 of Algorithm 8.1 is often replaced by a weaker condition, e.g., repeat until only 1% of the points change clusters.

We consider each of the steps in the basic K-means algorithm in more detail and then provide an analysis of the algorithm's space and time complexity.

Assigning Points to the Closest Centroid

To assign a point to the closest centroid, we need a proximity measure that quantifies the notion of "closest" for the specific data under consideration. Euclidean (L_2) distance is often used for data points in Euclidean space, while cosine similarity is more appropriate for documents. However, there may be several types of proximity measures that are appropriate for a given type of data. For example, Manhattan (L_1) distance can be used for Euclidean data, while the Jaccard measure is often employed for documents.

Usually, the similarity measures used for K-means are relatively simple since the algorithm repeatedly calculates the similarity of each point to each centroid. In some cases, however, such as when the data is in low-dimensional

Table 8.1. Table of notation.

Symbol	Description
x	An object.
C_i	The i^{th} cluster.
c_i	The centroid of cluster C_i .
c	The centroid of all points.
m_i	The number of objects in the i^{th} cluster.
m	The number of objects in the data set.
K	The number of clusters.

Euclidean space, it is possible to avoid computing many of the similarities, thus significantly speeding up the K-means algorithm. Bisecting K-means (described in Section 8.2.3) is another approach that speeds up K-means by reducing the number of similarities computed.

Centroids and Objective Functions

Step 4 of the K-means algorithm was stated rather generally as "recompute the centroid of each cluster," since the centroid can vary, depending on the proximity measure for the data and the goal of the clustering. The goal of the clustering is typically expressed by an objective function that depends on the proximities of the points to one another or to the cluster centroids; e.g., minimize the squared distance of each point to its closest centroid. We illustrate this with two examples. However, the key point is this: once we have specified a proximity measure and an objective function, the centroid that we should choose can often be determined mathematically. We provide mathematical details in Section 8.2.6, and provide a non-mathematical discussion of this observation here.

Data in Euclidean Space Consider data whose proximity measure is Euclidean distance. For our objective function, which measures the quality of a clustering, we use the sum of the squared error (SSE), which is also known as scatter. In other words, we calculate the error of each data point, i.e., its Euclidean distance to the closest centroid, and then compute the total sum of the squared errors. Given two different sets of clusters that are produced by two different runs of K-means, we prefer the one with the smallest squared error since this means that the prototypes (centroids) of this clustering are a better representation of the points in their cluster. Using the notation in Table 8.1, the SSE is formally defined as follows:

$$\text{SSE} = \sum_{i=1}^K \sum_{x \in C_i} \text{dist}(c_i, x)^2 \quad (8.1)$$

where dist is the standard Euclidean (L_2) distance between two objects in Euclidean space.

Given these assumptions, it can be shown (see Section 8.2.6) that the centroid that minimizes the SSE of the cluster is the mean. Using the notation in Table 8.1, the centroid (mean) of the i^{th} cluster is defined by Equation 8.2.

$$c_i = \frac{1}{m_i} \sum_{x \in C_i} x \quad (8.2)$$

To illustrate, the centroid of a cluster containing the three two-dimensional points, (1,1), (2,3), and (6,2), is $((1+2+6)/3, ((1+3+2)/3)) = (3, 2)$.

Steps 3 and 4 of the K-means algorithm directly attempt to minimize the SSE (or more generally, the objective function). Step 3 forms clusters by assigning points to their nearest centroid, which minimizes the SSE for the given set of centroids. Step 4 recomputes the centroids so as to further minimize the SSE. However, the actions of K-means in Steps 3 and 4 are only guaranteed to find a local minimum with respect to the SSE since they are based on optimizing the SSE for specific choices of the centroids and clusters, rather than for all possible choices. We will later see an example in which this leads to a suboptimal clustering.

Document Data To illustrate that K-means is not restricted to data in Euclidean space, we consider document data and the cosine similarity measure. Here we assume that the document data is represented as a document-term matrix as described on page 31. Our objective is to maximize the similarity of the documents in a cluster to the cluster centroid; this quantity is known as the cohesion of the cluster. For this objective it can be shown that the cluster centroid is, as for Euclidean data, the mean. The analogous quantity to the total SSE is the total cohesion, which is given by Equation 8.3.

$$\text{Total Cohesion} = \sum_{i=1}^K \sum_{x \in C_i} \text{cosine}(x, c_i) \quad (8.3)$$

The General Case There are a number of choices for the proximity function, centroid, and objective function that can be used in the basic K-means

Table 8.2. K-means: Common choices for proximity, centroids, and objective functions.

Proximity Function	Centroid	Objective Function
Manhattan (L_1)	median	Minimize sum of the L_1 distance of an object to its cluster centroid
Squared Euclidean (L_2^2)	mean	Minimize sum of the squared L_2 distance of an object to its cluster centroid
cosine	mean	Maximize sum of the cosine similarity of an object to its cluster centroid
Bregman divergence	mean	Minimize sum of the Bregman divergence of an object to its cluster centroid

algorithm and that are guaranteed to converge. Table 8.2 shows some possible choices, including the two that we have just discussed. Notice that for Manhattan (L_1) distance and the objective of minimizing the sum of the distances, the appropriate centroid is the median of the points in a cluster.

The last entry in the table, Bregman divergence (Section 2.4.5), is actually a class of proximity measures that includes the squared Euclidean distance, L_2^2 , the Mahalanobis distance, and cosine similarity. The importance of Bregman divergence functions is that any such function can be used as the basis of a K-means style clustering algorithm with the mean as the centroid. Specifically, if we use a Bregman divergence as our proximity function, then the resulting clustering algorithm has the usual properties of K-means with respect to convergence, local minima, etc. Furthermore, the properties of such a clustering algorithm can be developed for all possible Bregman divergences. Indeed, K-means algorithms that use cosine similarity or squared Euclidean distance are particular instances of a general clustering algorithm based on Bregman divergences.

For the rest of our K-means discussion, we use two-dimensional data since it is easy to explain K-means and its properties for this type of data. But, as suggested by the last few paragraphs, K-means is a very general clustering algorithm and can be used with a wide variety of data types, such as documents and time series.

Choosing Initial Centroids

When random initialization of centroids is used, different runs of K-means typically produce different total SSEs. We illustrate this with the set of two-dimensional points shown in Figure 8.3, which has three natural clusters of points. Figure 8.4(a) shows a clustering solution that is the global minimum of

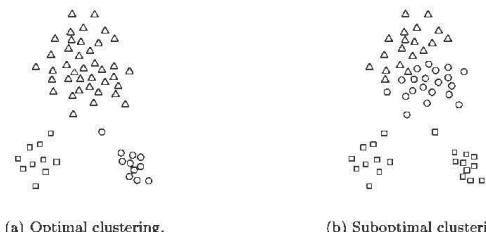


Figure 8.4. Three optimal and non-optimal clusters.

the SSE for three clusters, while Figure 8.4(b) shows a suboptimal clustering that is only a local minimum.

Choosing the proper initial centroids is the key step of the basic K-means procedure. A common approach is to choose the initial centroids randomly, but the resulting clusters are often poor.

Example 8.1 (Poor Initial Centroids). Randomly selected initial centroids may be poor. We provide an example of this using the same data set used in Figures 8.3 and 8.4. Figures 8.3 and 8.5 show the clusters that result from two particular choices of initial centroids. (For both figures, the positions of the cluster centroids in the various iterations are indicated by crosses.) In Figure 8.3, even though all the initial centroids are from one natural cluster, the minimum SSE clustering is still found. In Figure 8.5, however, even though the initial centroids seem to be better distributed, we obtain a suboptimal clustering, with higher squared error. ■

Example 8.2 (Limits of Random Initialization). One technique that is commonly used to address the problem of choosing initial centroids is to perform multiple runs, each with a different set of randomly chosen initial centroids, and then select the set of clusters with the minimum SSE. While simple, this strategy may not work very well, depending on the data set and the number of clusters sought. We demonstrate this using the sample data set shown in Figure 8.6(a). The data consists of two pairs of clusters, where the clusters in each (top-bottom) pair are closer to each other than to the clusters in the other pair. Figure 8.6 (b-d) shows that if we start with two initial centroids per pair of clusters, then even when both centroids are in a single

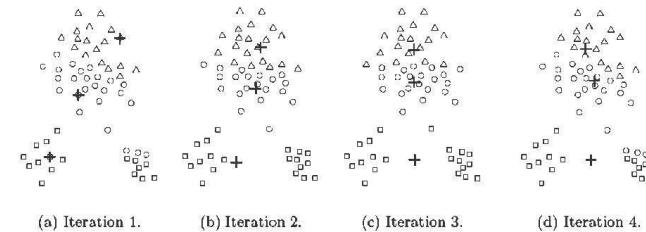


Figure 8.5. Poor starting centroids for K-means.

cluster, the centroids will redistribute themselves so that the “true” clusters are found. However, Figure 8.7 shows that if a pair of clusters has only one initial centroid and the other pair has three, then two of the true clusters will be combined and one true cluster will be split.

Note that an optimal clustering will be obtained as long as two initial centroids fall anywhere in a pair of clusters, since the centroids will redistribute themselves, one to each cluster. Unfortunately, as the number of clusters becomes larger, it is increasingly likely that at least one pair of clusters will have only one initial centroid. (See Exercise 4 on page 559.) In this case, because the pairs of clusters are farther apart than clusters within a pair, the K-means algorithm will not redistribute the centroids between pairs of clusters, and thus, only a local minimum will be achieved. ■

Because of the problems with using randomly selected initial centroids, which even repeated runs may not overcome, other techniques are often employed for initialization. One effective approach is to take a sample of points and cluster them using a hierarchical clustering technique. K clusters are extracted from the hierarchical clustering, and the centroids of those clusters are used as the initial centroids. This approach often works well, but is practical only if (1) the sample is relatively small, e.g., a few hundred to a few thousand (hierarchical clustering is expensive), and (2) K is relatively small compared to the sample size.

The following procedure is another approach to selecting initial centroids. Select the first point at random or take the centroid of all points. Then, for each successive initial centroid, select the point that is farthest from any of the initial centroids already selected. In this way, we obtain a set of initial

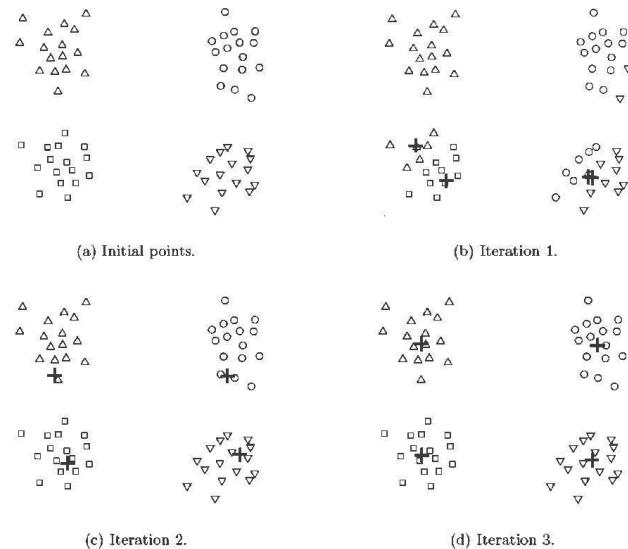


Figure 8.6. Two pairs of clusters with a pair of initial centroids within each pair of clusters.

centroids that is guaranteed to be not only randomly selected but also well separated. Unfortunately, such an approach can select outliers, rather than points in dense regions (clusters). Also, it is expensive to compute the farthest point from the current set of initial centroids. To overcome these problems, this approach is often applied to a sample of the points. Since outliers are rare, they tend not to show up in a random sample. In contrast, points from every dense region are likely to be included unless the sample size is very small. Also, the computation involved in finding the initial centroids is greatly reduced because the sample size is typically much smaller than the number of points.

Later on, we will discuss two other approaches that are useful for producing better-quality (lower SSE) clusterings: using a variant of K-means that

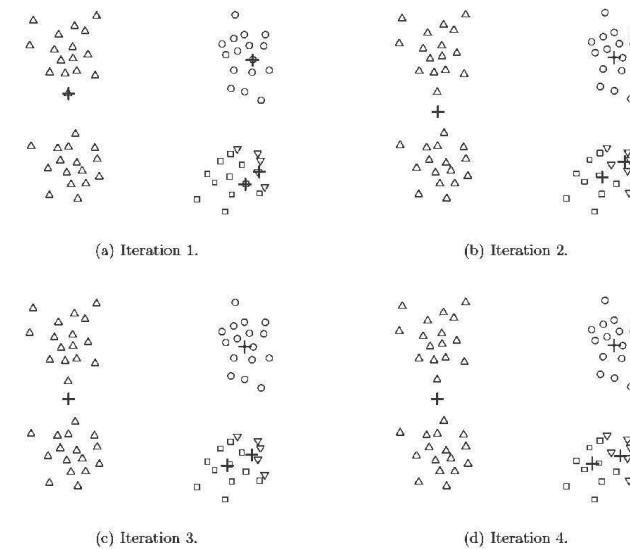


Figure 8.7. Two pairs of clusters with more or fewer than two initial centroids within a pair of clusters.

is less susceptible to initialization problems (bisection K-means) and using postprocessing to “fixup” the set of clusters produced.

Time and Space Complexity

The space requirements for K-means are modest because only the data points and centroids are stored. Specifically, the storage required is $O((m + K)n)$, where m is the number of points and n is the number of attributes. The time requirements for K-means are also modest—basically linear in the number of data points. In particular, the time required is $O(I * K * m * n)$, where I is the number of iterations required for convergence. As mentioned, I is often small and can usually be safely bounded, as most changes typically occur in the

first few iterations. Therefore, K-means is linear in m , the number of points, and is efficient as well as simple provided that K , the number of clusters, is significantly less than m .

8.2.2 K-means: Additional Issues

Handling Empty Clusters

One of the problems with the basic K-means algorithm given earlier is that empty clusters can be obtained if no points are allocated to a cluster during the assignment step. If this happens, then a strategy is needed to choose a replacement centroid, since otherwise, the squared error will be larger than necessary. One approach is to choose the point that is farthest away from any current centroid. If nothing else, this eliminates the point that currently contributes most to the total squared error. Another approach is to choose the replacement centroid from the cluster that has the highest SSE. This will typically split the cluster and reduce the overall SSE of the clustering. If there are several empty clusters, then this process can be repeated several times.

Outliers

When the squared error criterion is used, outliers can unduly influence the clusters that are found. In particular, when outliers are present, the resulting cluster centroids (prototypes) may not be as representative as they otherwise would be and thus, the SSE will be higher as well. Because of this, it is often useful to discover outliers and eliminate them beforehand. It is important, however, to appreciate that there are certain clustering applications for which outliers should not be eliminated. When clustering is used for data compression, every point must be clustered, and in some cases, such as financial analysis, apparent outliers, e.g., unusually profitable customers, can be the most interesting points.

An obvious issue is how to identify outliers. A number of techniques for identifying outliers will be discussed in Chapter 10. If we use approaches that remove outliers before clustering, we avoid clustering points that will not cluster well. Alternatively, outliers can also be identified in a postprocessing step. For instance, we can keep track of the SSE contributed by each point, and eliminate those points with unusually high contributions, especially over multiple runs. Also, we may want to eliminate small clusters since they frequently represent groups of outliers.

Reducing the SSE with Postprocessing

An obvious way to reduce the SSE is to find more clusters, i.e., to use a larger K . However, in many cases, we would like to improve the SSE, but don't want to increase the number of clusters. This is often possible because K-means typically converges to a local minimum. Various techniques are used to "fix up" the resulting clusters in order to produce a clustering that has lower SSE. The strategy is to focus on individual clusters since the total SSE is simply the sum of the SSE contributed by each cluster. (We will use the terminology *total SSE* and *cluster SSE*, respectively, to avoid any potential confusion.) We can change the total SSE by performing various operations on the clusters, such as splitting or merging clusters. One commonly used approach is to use alternate cluster splitting and merging phases. During a splitting phase, clusters are divided, while during a merging phase, clusters are combined. In this way, it is often possible to escape local SSE minima and still produce a clustering solution with the desired number of clusters. The following are some techniques used in the splitting and merging phases.

Two strategies that decrease the total SSE by increasing the number of clusters are the following:

Split a cluster: The cluster with the largest SSE is usually chosen, but we could also split the cluster with the largest standard deviation for one particular attribute.

Introduce a new cluster centroid: Often the point that is farthest from any cluster center is chosen. We can easily determine this if we keep track of the SSE contributed by each point. Another approach is to choose randomly from all points or from the points with the highest SSE.

Two strategies that decrease the number of clusters, while trying to minimize the increase in total SSE, are the following:

Disperse a cluster: This is accomplished by removing the centroid that corresponds to the cluster and reassigning the points to other clusters. Ideally, the cluster that is dispersed should be the one that increases the total SSE the least.

Merge two clusters: The clusters with the closest centroids are typically chosen, although another, perhaps better, approach is to merge the two clusters that result in the smallest increase in total SSE. These two merging strategies are the same ones that are used in the hierarchical

clustering techniques known as the centroid method and Ward's method, respectively. Both methods are discussed in Section 8.3.

Updating Centroids Incrementally

Instead of updating cluster centroids after all points have been assigned to a cluster, the centroids can be updated incrementally, after each assignment of a point to a cluster. Notice that this requires either zero or two updates to cluster centroids at each step, since a point either moves to a new cluster (two updates) or stays in its current cluster (zero updates). Using an incremental update strategy guarantees that empty clusters are not produced since all clusters start with a single point, and if a cluster ever has only one point, then that point will always be reassigned to the same cluster.

In addition, if incremental updating is used, the relative weight of the point being added may be adjusted; e.g., the weight of points is often decreased as the clustering proceeds. While this can result in better accuracy and faster convergence, it can be difficult to make a good choice for the relative weight, especially in a wide variety of situations. These update issues are similar to those involved in updating weights for artificial neural networks.

Yet another benefit of incremental updates has to do with using objectives other than "minimize SSE." Suppose that we are given an arbitrary objective function to measure the goodness of a set of clusters. When we process an individual point, we can compute the value of the objective function for each possible cluster assignment, and then choose the one that optimizes the objective. Specific examples of alternative objective functions are given in Section 8.5.2.

On the negative side, updating centroids incrementally introduces an order dependency. In other words, the clusters produced may depend on the order in which the points are processed. Although this can be addressed by randomizing the order in which the points are processed, the basic K-means approach of updating the centroids after all points have been assigned to clusters has no order dependency. Also, incremental updates are slightly more expensive. However, K-means converges rather quickly, and therefore, the number of points switching clusters quickly becomes relatively small.

8.2.3 Bisecting K-means

The bisecting K-means algorithm is a straightforward extension of the basic K-means algorithm that is based on a simple idea: to obtain K clusters, split the set of all points into two clusters, select one of these clusters to split, and

so on, until K clusters have been produced. The details of bisecting K-means are given by Algorithm 8.2.

Algorithm 8.2 Bisecting K-means algorithm.

```

1: Initialize the list of clusters to contain the cluster consisting of all points.
2: repeat
3:   Remove a cluster from the list of clusters.
4:   {Perform several "trial" bisections of the chosen cluster.}
5:   for  $i = 1$  to number of trials do
6:     Bisect the selected cluster using basic K-means.
7:   end for
8:   Select the two clusters from the bisection with the lowest total SSE.
9:   Add these two clusters to the list of clusters.
10: until Until the list of clusters contains  $K$  clusters.

```

There are a number of different ways to choose which cluster to split. We can choose the largest cluster at each step, choose the one with the largest SSE, or use a criterion based on both size and SSE. Different choices result in different clusters.

We often refine the resulting clusters by using their centroids as the initial centroids for the basic K-means algorithm. This is necessary because, although the K-means algorithm is guaranteed to find a clustering that represents a local minimum with respect to the SSE, in bisecting K-means we are using the K-means algorithm "locally," i.e., to bisect individual clusters. Therefore, the final set of clusters does not represent a clustering that is a local minimum with respect to the total SSE.

Example 8.3 (Bisecting K-means and Initialization). To illustrate that bisecting K-means is less susceptible to initialization problems, we show, in Figure 8.8, how bisecting K-means finds four clusters in the data set originally shown in Figure 8.6(a). In iteration 1, two pairs of clusters are found; in iteration 2, the rightmost pair of clusters is split; and in iteration 3, the leftmost pair of clusters is split. Bisecting K-means has less trouble with initialization because it performs several trial bisections and takes the one with the lowest SSE, and because there are only two centroids at each step. ■

Finally, by recording the sequence of clusterings produced as K-means bisects clusters, we can also use bisecting K-means to produce a hierarchical clustering.

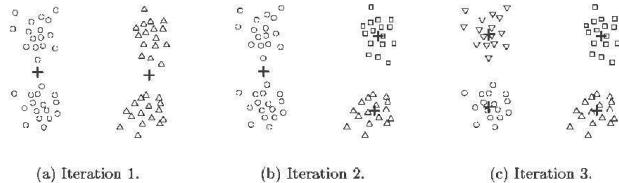


Figure 8.8. Bisecting K-means on the four clusters example.

8.2.4 K-means and Different Types of Clusters

K-means and its variations have a number of limitations with respect to finding different types of clusters. In particular, K-means has difficulty detecting the “natural” clusters, when clusters have non-spherical shapes or widely different sizes or densities. This is illustrated by Figures 8.9, 8.10, and 8.11. In Figure 8.9, K-means cannot find the three natural clusters because one of the clusters is much larger than the other two, and hence, the larger cluster is broken, while one of the smaller clusters is combined with a portion of the larger cluster. In Figure 8.10, K-means fails to find the three natural clusters because the two smaller clusters are much denser than the larger cluster. Finally, in Figure 8.11, K-means finds two clusters that mix portions of the two natural clusters because the shape of the natural clusters is not globular.

The difficulty in these three situations is that the K-means objective function is a mismatch for the kinds of clusters we are trying to find since it is minimized by globular clusters of equal size and density or by clusters that are well separated. However, these limitations can be overcome, in some sense, if the user is willing to accept a clustering that breaks the natural clusters into a number of subclusters. Figure 8.12 shows what happens to the three previous data sets if we find six clusters instead of two or three. Each smaller cluster is pure in the sense that it contains only points from one of the natural clusters.

8.2.5 Strengths and Weaknesses

K-means is simple and can be used for a wide variety of data types. It is also quite efficient, even though multiple runs are often performed. Some variants, including bisecting K-means, are even more efficient, and are less susceptible to initialization problems. K-means is not suitable for all types of data,

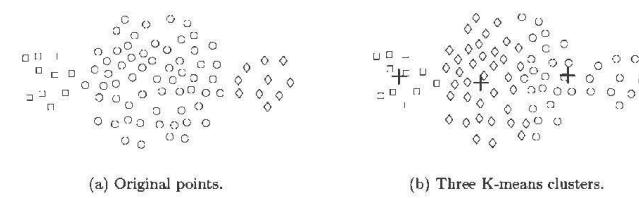


Figure 8.9. K-means with clusters of different size.

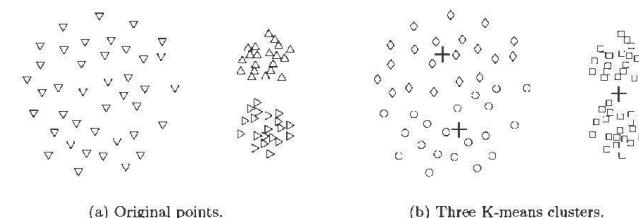


Figure 8.10. K-means with clusters of different density.

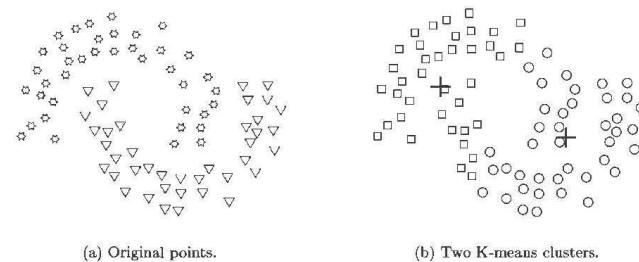


Figure 8.11. K-means with non-globular clusters.

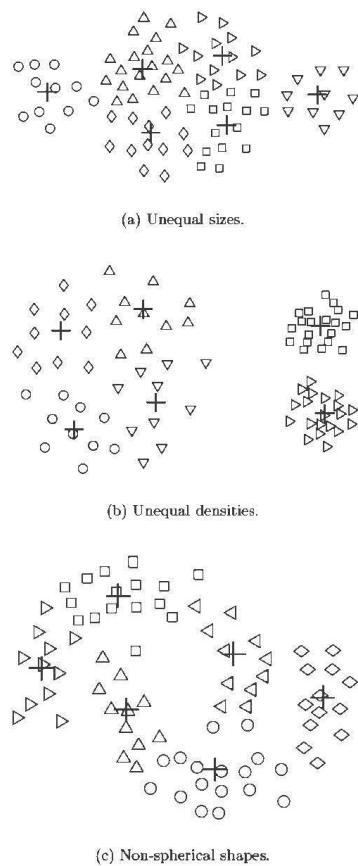


Figure 8.12. Using K-means to find clusters that are subclusters of the natural clusters.

however. It cannot handle non-globular clusters or clusters of different sizes and densities, although it can typically find pure subclusters if a large enough number of clusters is specified. K-means also has trouble clustering data that contains outliers. Outlier detection and removal can help significantly in such situations. Finally, K-means is restricted to data for which there is a notion of a center (centroid). A related technique, K-medoid clustering, does not have this restriction, but is more expensive.

8.2.6 K-means as an Optimization Problem

Here, we delve into the mathematics behind K-means. This section, which can be skipped without loss of continuity, requires knowledge of calculus through partial derivatives. Familiarity with optimization techniques, especially those based on gradient descent, may also be helpful.

As mentioned earlier, given an objective function such as “minimize SSE,” clustering can be treated as an optimization problem. One way to solve this problem—to find a global optimum—is to enumerate all possible ways of dividing the points into clusters and then choose the set of clusters that best satisfies the objective function, e.g., that minimizes the total SSE. Of course, this exhaustive strategy is computationally infeasible and as a result, a more practical approach is needed, even if such an approach finds solutions that are not guaranteed to be optimal. One technique, which is known as gradient descent, is based on picking an initial solution and then repeating the following two steps: compute the change to the solution that best optimizes the objective function and then update the solution.

We assume that the data is one-dimensional, i.e., $dist(x, y) = (x - y)^2$. This does not change anything essential, but greatly simplifies the notation.

Derivation of K-means as an Algorithm to Minimize the SSE

In this section, we show how the centroid for the K-means algorithm can be mathematically derived when the proximity function is Euclidean distance and the objective is to minimize the SSE. Specifically, we investigate how we can best update a cluster centroid so that the cluster SSE is minimized. In mathematical terms, we seek to minimize Equation 8.1, which we repeat here, specialized for one-dimensional data.

$$\text{SSE} = \sum_{i=1}^K \sum_{x \in C_i} (c_i - x)^2 \quad (8.4)$$

Here, C_i is the i^{th} cluster, x is a point in C_i , and c_i is the mean of the i^{th} cluster. See Table 8.1 for a complete list of notation.

We can solve for the k^{th} centroid c_k , which minimizes Equation 8.4, by differentiating the SSE, setting it equal to 0, and solving, as indicated below.

$$\begin{aligned}\frac{\partial}{\partial c_k} \text{SSE} &= \frac{\partial}{\partial c_k} \sum_{i=1}^K \sum_{x \in C_i} (c_i - x)^2 \\ &= \sum_{i=1}^K \sum_{x \in C_i} \frac{\partial}{\partial c_k} (c_i - x)^2 \\ &= \sum_{x \in C_k} 2 * (c_k - x_k) = 0\end{aligned}$$

$$\sum_{x \in C_k} 2 * (c_k - x_k) = 0 \Rightarrow m_k c_k = \sum_{x \in C_k} x_k \Rightarrow c_k = \frac{1}{m_k} \sum_{x \in C_k} x_k$$

Thus, as previously indicated, the best centroid for minimizing the SSE of a cluster is the mean of the points in the cluster.

Derivation of K-means for SAE

To demonstrate that the K-means algorithm can be applied to a variety of different objective functions, we consider how to partition the data into K clusters such that the sum of the Manhattan (L_1) distances of points from the center of their clusters is minimized. We are seeking to minimize the sum of the L_1 absolute errors (SAE) as given by the following equation, where dist_{L_1} is the L_1 distance. Again, for notational simplicity, we use one-dimensional data, i.e., $\text{dist}_{L_1} = |c_i - x|$.

$$\text{SAE} = \sum_{i=1}^K \sum_{x \in C_i} \text{dist}_{L_1}(c_i, x) \quad (8.5)$$

We can solve for the k^{th} centroid c_k , which minimizes Equation 8.5, by differentiating the SAE, setting it equal to 0, and solving.

$$\begin{aligned}\frac{\partial}{\partial c_k} \text{SAE} &= \frac{\partial}{\partial c_k} \sum_{i=1}^K \sum_{x \in C_i} |c_i - x| \\ &= \sum_{i=1}^K \sum_{x \in C_i} \frac{\partial}{\partial c_k} |c_i - x| \\ &= \sum_{x \in C_k} \frac{\partial}{\partial c_k} |c_k - x| = 0 \\ \sum_{x \in C_k} \frac{\partial}{\partial c_k} |c_k - x| = 0 &\Rightarrow \sum_{x \in C_k} \text{sign}(x - c_k) = 0\end{aligned}$$

If we solve for c_k , we find that $c_k = \text{median}\{x \in C_k\}$, the median of the points in the cluster. The median of a group of points is straightforward to compute and less susceptible to distortion by outliers.

8.3 Agglomerative Hierarchical Clustering

Hierarchical clustering techniques are a second important category of clustering methods. As with K-means, these approaches are relatively old compared to many clustering algorithms, but they still enjoy widespread use. There are two basic approaches for generating a hierarchical clustering:

Agglomerative: Start with the points as individual clusters and, at each step, merge the closest pair of clusters. This requires defining a notion of cluster proximity.

Divisive: Start with one, all-inclusive cluster and, at each step, split a cluster until only singleton clusters of individual points remain. In this case, we need to decide which cluster to split at each step and how to do the splitting.

Agglomerative hierarchical clustering techniques are by far the most common, and, in this section, we will focus exclusively on these methods. A divisive hierarchical clustering technique is described in Section 9.4.2.

A hierarchical clustering is often displayed graphically using a tree-like diagram called a dendrogram, which displays both the cluster-subcluster

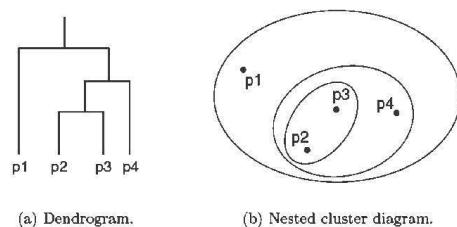


Figure 8.13. A hierarchical clustering of four points shown as a dendrogram and as nested clusters.

relationships and the order in which the clusters were merged (agglomerative view) or split (divisive view). For sets of two-dimensional points, such as those that we will use as examples, a hierarchical clustering can also be graphically represented using a nested cluster diagram. Figure 8.13 shows an example of these two types of figures for a set of four two-dimensional points. These points were clustered using the single-link technique that is described in Section 8.3.2.

8.3.1 Basic Agglomerative Hierarchical Clustering Algorithm

Many agglomerative hierarchical clustering techniques are variations on a single approach: starting with individual points as clusters, successively merge the two closest clusters until only one cluster remains. This approach is expressed more formally in Algorithm 8.3.

Algorithm 8.3 Basic agglomerative hierarchical clustering algorithm.

- 1: Compute the proximity matrix, if necessary.
 - 2: repeat
 - 3: Merge the closest two clusters.
 - 4: Update the proximity matrix to reflect the proximity between the new cluster and the original clusters.
 - 5: until Only one cluster remains.
-

Defining Proximity between Clusters

The key operation of Algorithm 8.3 is the computation of the proximity between two clusters, and it is the definition of cluster proximity that differentiates the various agglomerative hierarchical techniques that we will discuss. Cluster proximity is typically defined with a particular type of cluster in mind—see Section 8.1.2. For example, many agglomerative hierarchical clustering techniques, such as MIN, MAX, and Group Average, come from a graph-based view of clusters. MIN defines cluster proximity as the proximity between the closest two points that are in different clusters, or using graph terms, the shortest edge between two nodes in different subsets of nodes. This yields contiguity-based clusters as shown in Figure 8.2(c). Alternatively, MAX takes the proximity between the farthest two points in different clusters to be the cluster proximity, or using graph terms, the longest edge between two nodes in different subsets of nodes. (If our proximities are distances, then the names, MIN and MAX, are short and suggestive. For similarities, however, where higher values indicate closer points, the names seem reversed. For that reason, we usually prefer to use the alternative names, single link and complete link, respectively.) Another graph-based approach, the group average technique, defines cluster proximity to be the average pairwise proximities (average length of edges) of all pairs of points from different clusters. Figure 8.14 illustrates these three approaches.

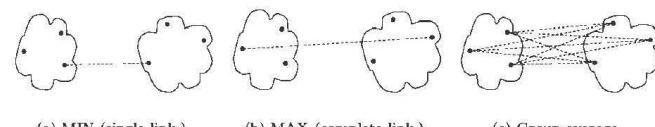


Figure 8.14. Graph-based definitions of cluster proximity

If, instead, we take a prototype-based view, in which each cluster is represented by a centroid, different definitions of cluster proximity are more natural. When using centroids, the cluster proximity is commonly defined as the proximity between cluster centroids. An alternative technique, Ward's method, also assumes that a cluster is represented by its centroid, but it measures the proximity between two clusters in terms of the increase in the SSE that re-

sults from merging the two clusters. Like K-means, Ward's method attempts to minimize the sum of the squared distances of points from their cluster centroids.

Time and Space Complexity

The basic agglomerative hierarchical clustering algorithm just presented uses a proximity matrix. This requires the storage of $\frac{1}{2}m^2$ proximities (assuming the proximity matrix is symmetric) where m is the number of data points. The space needed to keep track of the clusters is proportional to the number of clusters, which is $m - 1$, excluding singleton clusters. Hence, the total space complexity is $O(m^2)$.

The analysis of the basic agglomerative hierarchical clustering algorithm is also straightforward with respect to computational complexity. $O(m^2)$ time is required to compute the proximity matrix. After that step, there are $m - 1$ iterations involving steps 3 and 4 because there are m clusters at the start and two clusters are merged during each iteration. If performed as a linear search of the proximity matrix, then for the i^{th} iteration, step 3 requires $O((m - i + 1)^2)$ time, which is proportional to the current number of clusters squared. Step 4 only requires $O(m - i + 1)$ time to update the proximity matrix after the merger of two clusters. (A cluster merger affects only $O(m - i + 1)$ proximities for the techniques that we consider.) Without modification, this would yield a time complexity of $O(m^3)$. If the distances from each cluster to all other clusters are stored as a sorted list (or heap), it is possible to reduce the cost of finding the two closest clusters to $O(m - i + 1)$. However, because of the additional complexity of keeping data in sorted list or heap, the overall time required for a hierarchical clustering based on Algorithm 8.3 is $O(m^2 \log m)$.

The space and time complexity of hierarchical clustering severely limits the size of data sets that can be processed. We discuss scalability approaches for clustering algorithms, including hierarchical clustering techniques, in Section 9.5.

8.3.2 Specific Techniques

Sample Data

To illustrate the behavior of the various hierarchical clustering algorithms, we shall use sample data that consists of 6 two-dimensional points, which are shown in Figure 8.15. The x and y coordinates of the points and the Euclidean distances between them are shown in Tables 8.3 and 8.4, respectively.

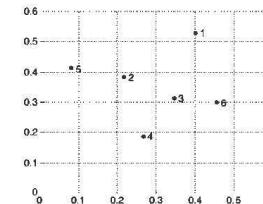


Figure 8.15. Set of 6 two-dimensional points.

Point	<i>x</i> Coordinate	<i>y</i> Coordinate
p1	0.40	0.53
p2	0.22	0.38
p3	0.35	0.32
p4	0.26	0.19
p5	0.08	0.41
p6	0.45	0.30

Table 8.3. *xy* coordinates of 6 points.

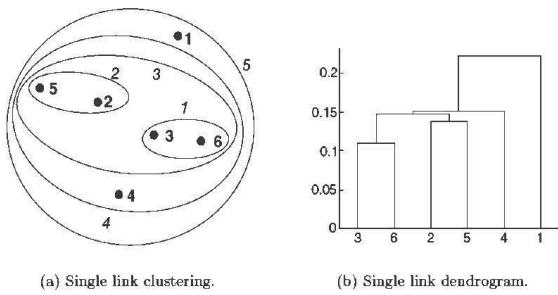
	p1	p2	p3	p4	p5	p6
p1	0.00	0.24	0.22	0.37	0.34	0.23
p2	0.24	0.00	0.15	0.20	0.14	0.25
p3	0.22	0.15	0.00	0.15	0.28	0.11
p4	0.37	0.20	0.15	0.00	0.29	0.22
p5	0.34	0.14	0.28	0.29	0.00	0.39
p6	0.23	0.25	0.11	0.22	0.39	0.00

Table 8.4. Euclidean distance matrix for 6 points.

Single Link or MIN

For the single link or MIN version of hierarchical clustering, the proximity of two clusters is defined as the minimum of the distance (maximum of the similarity) between any two points in the two different clusters. Using graph terminology, if you start with all points as singleton clusters and add links between points one at a time, shortest links first, then these single links combine the points into clusters. The single link technique is good at handling non-elliptical shapes, but is sensitive to noise and outliers.

Example 8.4 (Single Link). Figure 8.16 shows the result of applying the single link technique to our example data set of six points. Figure 8.16(a) shows the nested clusters as a sequence of nested ellipses, where the numbers associated with the ellipses indicate the order of the clustering. Figure 8.16(b) shows the same information, but as a dendrogram. The height at which two clusters are merged in the dendrogram reflects the distance of the two clusters. For instance, from Table 8.4, we see that the distance between points 3 and 6



(a) Single link clustering.

(b) Single link dendrogram.

Figure 8.16. Single link clustering of the six points shown in Figure 8.15.

is 0.11, and that is the height at which they are joined into one cluster in the dendrogram. As another example, the distance between clusters $\{3, 6\}$ and $\{2, 5\}$ is given by

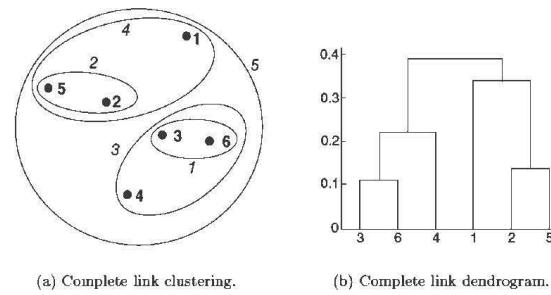
$$\begin{aligned} \text{dist}(\{3, 6\}, \{2, 5\}) &= \min(\text{dist}(3, 2), \text{dist}(6, 2), \text{dist}(3, 5), \text{dist}(6, 5)) \\ &= \min(0.15, 0.25, 0.28, 0.39) \\ &= 0.15. \end{aligned}$$

■

Complete Link or MAX or CLIQUE

For the complete link or MAX version of hierarchical clustering, the proximity of two clusters is defined as the maximum of the distance (minimum of the similarity) between any two points in the two different clusters. Using graph terminology, if you start with all points as singleton clusters and add links between points one at a time, shortest links first, then a group of points is not a cluster until all the points in it are completely linked, i.e., form a *clique*. Complete link is less susceptible to noise and outliers, but it can break large clusters and it favors globular shapes.

Example 8.5 (Complete Link). Figure 8.17 shows the results of applying MAX to the sample data set of six points. As with single link, points 3 and 6



(a) Complete link clustering.

(b) Complete link dendrogram.

Figure 8.17. Complete link clustering of the six points shown in Figure 8.15.

are merged first. However, $\{3, 6\}$ is merged with $\{4\}$, instead of $\{2, 5\}$ or $\{1\}$ because

$$\begin{aligned} \text{dist}(\{3, 6\}, \{4\}) &= \max(\text{dist}(3, 4), \text{dist}(6, 4)) \\ &= \max(0.15, 0.22) \\ &= 0.22. \\ \text{dist}(\{3, 6\}, \{2, 5\}) &= \max(\text{dist}(3, 2), \text{dist}(6, 2), \text{dist}(3, 5), \text{dist}(6, 5)) \\ &= \max(0.15, 0.25, 0.28, 0.39) \\ &= 0.39. \\ \text{dist}(\{3, 6\}, \{1\}) &= \max(\text{dist}(3, 1), \text{dist}(6, 1)) \\ &= \max(0.22, 0.23) \\ &= 0.23. \end{aligned}$$

■

Group Average

For the group average version of hierarchical clustering, the proximity of two clusters is defined as the average pairwise proximity among all pairs of points in the different clusters. This is an intermediate approach between the single and complete link approaches. Thus, for group average, the cluster proxim-

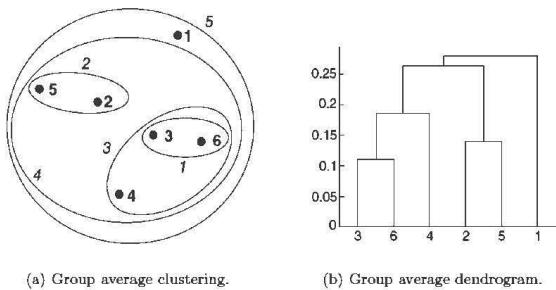


Figure 8.18. Group average clustering of the six points shown in Figure 8.15.

ity $\text{proximity}(C_i, C_j)$ of clusters C_i and C_j , which are of size m_i and m_j , respectively, is expressed by the following equation:

$$\text{proximity}(C_i, C_j) = \frac{\sum_{x \in C_i} \sum_{y \in C_j} \text{proximity}(x, y)}{m_i * m_j}. \quad (8.6)$$

Example 8.6 (Group Average). Figure 8.18 shows the results of applying the group average approach to the sample data set of six points. To illustrate how group average works, we calculate the distance between some clusters.

$$\begin{aligned} \text{dist}(\{3, 6, 4\}, \{1\}) &= (0.22 + 0.37 + 0.23)/(3 * 1) \\ &= 0.28 \\ \text{dist}(\{2, 5\}, \{1\}) &= (0.2357 + 0.3421)/(2 * 1) \\ &= 0.2889 \\ \text{dist}(\{3, 6, 4\}, \{2, 5\}) &= (0.15 + 0.28 + 0.25 + 0.39 + 0.20 + 0.29)/(6 * 2) \\ &= 0.26 \end{aligned}$$

Because $\text{dist}(\{3, 6, 4\}, \{2, 5\})$ is smaller than $\text{dist}(\{3, 6, 4\}, \{1\})$ and $\text{dist}(\{2, 5\}, \{1\})$, clusters $\{3, 6, 4\}$ and $\{2, 5\}$ are merged at the fourth stage. ■

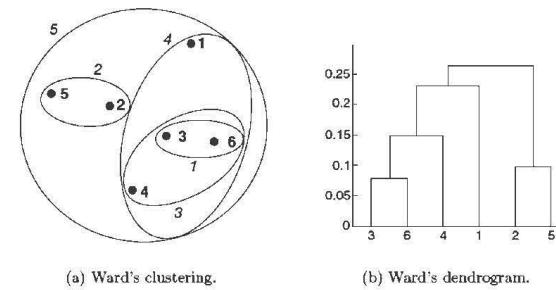


Figure 8.19. Ward's clustering of the six points shown in Figure 8.15.

Ward's Method and Centroid Methods

For Ward's method, the proximity between two clusters is defined as the increase in the squared error that results when two clusters are merged. Thus, this method uses the same objective function as K-means clustering. While it may seem that this feature makes Ward's method somewhat distinct from other hierarchical techniques, it can be shown mathematically that Ward's method is very similar to the group average method when the proximity between two points is taken to be the square of the distance between them.

Example 8.7 (Ward's Method). Figure 8.19 shows the results of applying Ward's method to the sample data set of six points. The clustering that is produced is different from those produced by single link, complete link, and group average. ■

Centroid methods calculate the proximity between two clusters by calculating the distance between the centroids of clusters. These techniques may seem similar to K-means, but as we have remarked, Ward's method is the correct hierarchical analog.

Centroid methods also have a characteristic—often considered bad—that is not possessed by the other hierarchical clustering techniques that we have discussed: the possibility of inversions. Specifically, two clusters that are merged may be more similar (less distant) than the pair of clusters that were merged in a previous step. For the other methods, the distance between

Table 8.5. Table of Lance-Williams coefficients for common hierarchical clustering approaches.

Clustering Method	α_A	α_B	β	γ
Single Link	1/2	1/2	0	-1/2
Complete Link	1/2	1/2	0	1/2
Group Average	$\frac{m_A}{m_A+m_B}$	$\frac{m_B}{m_A+m_B}$	0	0
Centroid	$\frac{m_A}{m_A+m_B}$	$\frac{m_B}{m_A+m_B}$	$\frac{-m_A m_B}{(m_A+m_B)^2}$	0
Ward's	$\frac{m_A+m_Q}{m_A+m_B+m_Q}$	$\frac{m_B+m_Q}{m_A+m_B+m_Q}$	$\frac{-m_Q}{m_A+m_B+m_Q}$	0

merged clusters monotonically increases (or is, at worst, non-increasing) as we proceed from singleton clusters to one all-inclusive cluster.

8.3.3 The Lance-Williams Formula for Cluster Proximity

Any of the cluster proximities that we have discussed in this section can be viewed as a choice of different parameters (in the Lance-Williams formula shown below in Equation 8.7) for the proximity between clusters Q and R , where R is formed by merging clusters A and B . In this equation, $p(\dots)$ is a proximity function, while m_A , m_B , and m_Q are the number of points in clusters A , B , and Q , respectively. In other words, after we merge clusters A and B to form cluster R , the proximity of the new cluster, R , to an existing cluster, Q , is a linear function of the proximities of Q with respect to the original clusters A and B . Table 8.5 shows the values of these coefficients for the techniques that we have discussed.

$$p(R, Q) = \alpha_A p(A, Q) + \alpha_B p(B, Q) + \beta p(A, B) + \gamma |p(A, Q) - p(B, Q)| \quad (8.7)$$

Any hierarchical clustering technique that can be expressed using the Lance-Williams formula does not need to keep the original data points. Instead, the proximity matrix is updated as clustering occurs. While a general formula is appealing, especially for implementation, it is easier to understand the different hierarchical methods by looking directly at the definition of cluster proximity that each method uses.

8.3.4 Key Issues in Hierarchical Clustering

Lack of a Global Objective Function

We previously mentioned that agglomerative hierarchical clustering cannot be viewed as globally optimizing an objective function. Instead, agglomerative hierarchical clustering techniques use various criteria to decide locally, at each

step, which clusters should be merged (or split for divisive approaches). This approach yields clustering algorithms that avoid the difficulty of attempting to solve a hard combinatorial optimization problem. (It can be shown that the general clustering problem for an objective function such as “minimize SSE” is computationally infeasible.) Furthermore, such approaches do not have problems with local minima or difficulties in choosing initial points. Of course, the time complexity of $O(m^2 \log m)$ and the space complexity of $O(m^2)$ are prohibitive in many cases.

Ability to Handle Different Cluster Sizes

One aspect of agglomerative hierarchical clustering that we have not yet discussed is how to treat the relative sizes of the pairs of clusters that are merged. (This discussion applies only to cluster proximity schemes that involve sums, such as centroid, Ward's, and group average.) There are two approaches: **weighted**, which treats all clusters equally, and **unweighted**, which takes the number of points in each cluster into account. Note that the terminology of weighted or unweighted refers to the data points, not the clusters. In other words, treating clusters of unequal size equally gives different weights to the points in different clusters, while taking the cluster size into account gives points in different clusters the same weight.

We will illustrate this using the group average technique discussed in Section 8.3.2, which is the unweighted version of the group average technique. In the clustering literature, the full name of this approach is the Unweighted Pair Group Method using Arithmetic averages (UPGMA). In Table 8.5, which gives the formula for updating cluster similarity, the coefficients for UPGMA involve the size of each of the clusters that were merged: $\alpha_A = \frac{m_A}{m_A+m_B}$, $\alpha_B = \frac{m_B}{m_A+m_B}$, $\beta = 0$, $\gamma = 0$. For the weighted version of group average—known as WPGMA—the coefficients are constants: $\alpha_A = 1/2$, $\alpha_B = 1/2$, $\beta = 0$, $\gamma = 0$. In general, unweighted approaches are preferred unless there is reason to believe that individual points should have different weights; e.g., perhaps classes of objects have been unevenly sampled.

Merging Decisions Are Final

Agglomerative hierarchical clustering algorithms tend to make good local decisions about combining two clusters since they can use information about the pairwise similarity of all points. However, once a decision is made to merge two clusters, it cannot be undone at a later time. This approach prevents a local optimization criterion from becoming a global optimization criterion.

For example, although the “minimize squared error” criterion from K-means is used in deciding which clusters to merge in Ward’s method, the clusters at each level do not represent local minima with respect to the total SSE. Indeed, the clusters are not even stable, in the sense that a point in one cluster may be closer to the centroid of some other cluster than it is to the centroid of its current cluster. Nonetheless, Ward’s method is often used as a robust method of initializing a K-means clustering, indicating that a local “minimize squared error” objective function does have a connection to a global “minimize squared error” objective function.

There are some techniques that attempt to overcome the limitation that merges are final. One approach attempts to fix up the hierarchical clustering by moving branches of the tree around so as to improve a global objective function. Another approach uses a partitional clustering technique such as K-means to create many small clusters, and then performs hierarchical clustering using these small clusters as the starting point.

8.3.5 Strengths and Weaknesses

The strengths and weakness of specific agglomerative hierarchical clustering algorithms were discussed above. More generally, such algorithms are typically used because the underlying application, e.g., creation of a taxonomy, requires a hierarchy. Also, there have been some studies that suggest that these algorithms can produce better-quality clusters. However, agglomerative hierarchical clustering algorithms are expensive in terms of their computational and storage requirements. The fact that all merges are final can also cause trouble for noisy, high-dimensional data, such as document data. In turn, these two problems can be addressed to some degree by first partially clustering the data using another technique, such as K-means.

8.4 DBSCAN

Density-based clustering locates regions of high density that are separated from one another by regions of low density. DBSCAN is a simple and effective density-based clustering algorithm that illustrates a number of important concepts that are important for any density-based clustering approach. In this section, we focus solely on DBSCAN after first considering the key notion of density. Other algorithms for finding density-based clusters are described in the next chapter.

8.4.1 Traditional Density: Center-Based Approach

Although there are not as many approaches for defining density as there are for defining similarity, there are several distinct methods. In this section we discuss the center-based approach on which DBSCAN is based. Other definitions of density will be presented in Chapter 9.

In the center-based approach, density is estimated for a particular point in the data set by counting the number of points within a specified radius, Eps , of that point. This includes the point itself. This technique is graphically illustrated by Figure 8.20. The number of points within a radius of Eps of point A is 7, including A itself.

This method is simple to implement, but the density of any point will depend on the specified radius. For instance, if the radius is large enough, then all points will have a density of m , the number of points in the data set. Likewise, if the radius is too small, then all points will have a density of 1. An approach for deciding on the appropriate radius for low-dimensional data is given in the next section in the context of our discussion of DBSCAN.

Classification of Points According to Center-Based Density

The center-based approach to density allows us to classify a point as being (1) in the interior of a dense region (a core point), (2) on the edge of a dense region (a border point), or (3) in a sparsely occupied region (a noise or background point). Figure 8.21 graphically illustrates the concepts of core, border, and noise points using a collection of two-dimensional points. The following text provides a more precise description.

Core points: These points are in the interior of a density-based cluster. A point is a core point if the number of points within a given neighborhood around the point as determined by the distance function and a user-specified distance parameter, Eps , exceeds a certain threshold, $MinPts$, which is also a user-specified parameter. In Figure 8.21, point A is a core point, for the indicated radius (Eps) if $MinPts \leq 7$.

Border points: A border point is not a core point, but falls within the neighborhood of a core point. In Figure 8.21, point B is a border point. A border point can fall within the neighborhoods of several core points.

Noise points: A noise point is any point that is neither a core point nor a border point. In Figure 8.21, point C is a noise point.

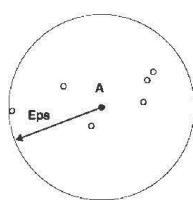


Figure 8.20. Center-based density.

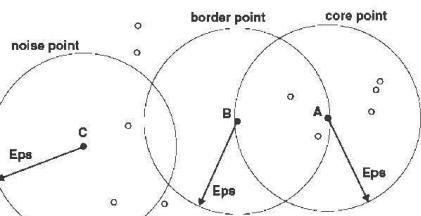


Figure 8.21. Core, border, and noise points.

8.4.2 The DBSCAN Algorithm

Given the previous definitions of core points, border points, and noise points, the DBSCAN algorithm can be informally described as follows. Any two core points that are close enough—within a distance Eps of one another—are put in the same cluster. Likewise, any border point that is close enough to a core point is put in the same cluster as the core point. (Ties may need to be resolved if a border point is close to core points from different clusters.) Noise points are discarded. The formal details are given in Algorithm 8.4. This algorithm uses the same concepts and finds the same clusters as the original DBSCAN, but is optimized for simplicity, not efficiency.

Algorithm 8.4 DBSCAN algorithm.

- 1: Label all points as core, border, or noise points.
 - 2: Eliminate noise points.
 - 3: Put an edge between all core points that are within Eps of each other.
 - 4: Make each group of connected core points into a separate cluster.
 - 5: Assign each border point to one of the clusters of its associated core points.
-

Time and Space Complexity

The basic time complexity of the DBSCAN algorithm is $O(m \times \text{time to find points in the } Eps\text{-neighborhood})$, where m is the number of points. In the worst case, this complexity is $O(m^2)$. However, in low-dimensional spaces, there are data structures, such as kd-trees, that allow efficient retrieval of all

points within a given distance of a specified point, and the time complexity can be as low as $O(m \log m)$. The space requirement of DBSCAN, even for high-dimensional data, is $O(m)$ because it is only necessary to keep a small amount of data for each point, i.e., the cluster label and the identification of each point as a core, border, or noise point.

Selection of DBSCAN Parameters

There is, of course, the issue of how to determine the parameters Eps and $MinPts$. The basic approach is to look at the behavior of the distance from a point to its k^{th} nearest neighbor, which we will call the k -dist. For points that belong to some cluster, the value of k -dist will be small if k is not larger than the cluster size. Note that there will be some variation, depending on the density of the cluster and the random distribution of points, but on average, the range of variation will not be huge if the cluster densities are not radically different. However, for points that are not in a cluster, such as noise points, the k -dist will be relatively large. Therefore, if we compute the k -dist for all the data points for some k , sort them in increasing order, and then plot the sorted values, we expect to see a sharp change at the value of k -dist that corresponds to a suitable value of Eps . If we select this distance as the Eps parameter and take the value of k as the $MinPts$ parameter, then points for which k -dist is less than Eps will be labeled as core points, while other points will be labeled as noise or border points.

Figure 8.22 shows a sample data set, while the k -dist graph for the data is given in Figure 8.23. The value of Eps that is determined in this way depends on k , but does not change dramatically as k changes. If the value of k is too small, then even a small number of closely spaced points that are noise or outliers will be incorrectly labeled as clusters. If the value of k is too large, then small clusters (of size less than k) are likely to be labeled as noise. The original DBSCAN algorithm used a value of $k = 4$, which appears to be a reasonable value for most two-dimensional data sets.

Clusters of Varying Density

DBSCAN can have trouble with density if the density of clusters varies widely. Consider Figure 8.24, which shows four clusters embedded in noise. The density of the clusters and noise regions is indicated by their darkness. The noise around the pair of denser clusters, A and B , has the same density as clusters C and D . If the Eps threshold is low enough that DBSCAN finds C and D as clusters, then A and B and the points surrounding them will become a single

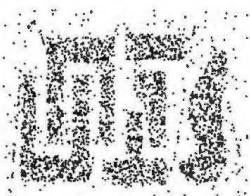


Figure 8.22. Sample data.

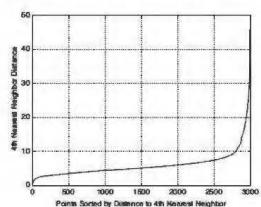


Figure 8.23. K-dist plot for sample data.

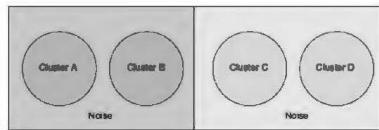


Figure 8.24. Four clusters embedded in noise.

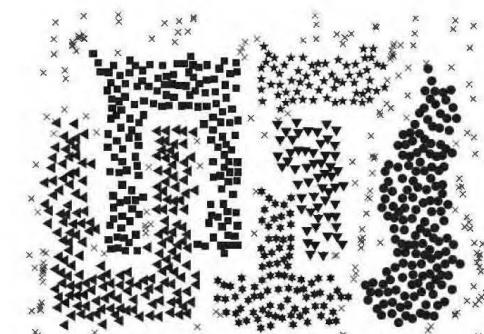
cluster. If the Eps threshold is high enough that DBSCAN finds A and B as separate clusters, and the points surrounding them are marked as noise, then C and D and the points surrounding them will also be marked as noise.

An Example

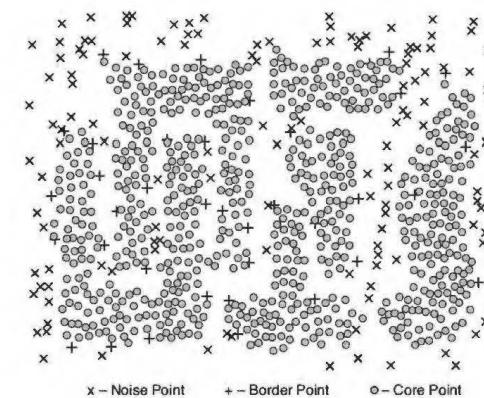
To illustrate the use of DBSCAN, we show the clusters that it finds in the relatively complicated two-dimensional data set shown in Figure 8.22. This data set consists of 3000 two-dimensional points. The Eps threshold for this data set was found by plotting the sorted distances of the fourth nearest neighbor of each point (Figure 8.23) and identifying the value at which there is a sharp increase. We selected $Eps = 10$, which corresponds to the knee of the curve. The clusters found by DBSCAN using these parameters, i.e., $MinPts = 4$ and $Eps = 10$, are shown in Figure 8.25(a). The core points, border points, and noise points are displayed in Figure 8.25(b).

8.4.3 Strengths and Weaknesses

Because DBSCAN uses a density-based definition of a cluster, it is relatively resistant to noise and can handle clusters of arbitrary shapes and sizes. Thus,



(a) Clusters found by DBSCAN.



(b) Core, border, and noise points.

Figure 8.25. DBSCAN clustering of 3000 two-dimensional points.

DBSCAN can find many clusters that could not be found using K-means, such as those in Figure 8.22. As indicated previously, however, DBSCAN has trouble when the clusters have widely varying densities. It also has trouble with high-dimensional data because density is more difficult to define for such data. One possible approach to dealing with such issues is given in Section 9.4.8. Finally, DBSCAN can be expensive when the computation of nearest neighbors requires computing all pairwise proximities, as is usually the case for high-dimensional data.

8.5 Cluster Evaluation

In supervised classification, the evaluation of the resulting classification model is an integral part of the process of developing a classification model, and there are well-accepted evaluation measures and procedures, e.g., accuracy and cross-validation, respectively. However, because of its very nature, cluster evaluation is not a well-developed or commonly used part of cluster analysis. Nonetheless, cluster evaluation, or cluster validation as it is more traditionally called, is important, and this section will review some of the most common and easily applied approaches.

There might be some confusion as to why cluster evaluation is necessary. Many times, cluster analysis is conducted as a part of an exploratory data analysis. Hence, evaluation seems like an unnecessarily complicated addition to what is supposed to be an informal process. Furthermore, since there are a number of different types of clusters—in some sense, each clustering algorithm defines its own type of cluster—it may seem that each situation might require a different evaluation measure. For instance, K-means clusters might be evaluated in terms of the SSE, but for density-based clusters, which need not be globular, SSE would not work well at all.

Nonetheless, cluster evaluation should be a part of any cluster analysis. A key motivation is that almost every clustering algorithm will find clusters in a data set, even if that data set has no natural cluster structure. For instance, consider Figure 8.26, which shows the result of clustering 100 points that are randomly (uniformly) distributed on the unit square. The original points are shown in Figure 8.26(a), while the clusters found by DBSCAN, K-means, and complete link are shown in Figures 8.26(b), 8.26(c), and 8.26(d), respectively. Since DBSCAN found three clusters (after we set Eps by looking at the distances of the fourth nearest neighbors), we set K-means and complete link to find three clusters as well. (In Figure 8.26(b) the noise is shown by the small markers.) However, the clusters do not look compelling for any of

the three methods. In higher dimensions, such problems cannot be so easily detected.

8.5.1 Overview

Being able to distinguish whether there is non-random structure in the data is just one important aspect of cluster validation. The following is a list of several important issues for cluster validation.

1. Determining the clustering tendency of a set of data, i.e., distinguishing whether non-random structure actually exists in the data.
2. Determining the correct number of clusters.
3. Evaluating how well the results of a cluster analysis fit the data *without* reference to external information.
4. Comparing the results of a cluster analysis to externally known results, such as externally provided class labels.
5. Comparing two sets of clusters to determine which is better.

Notice that items 1, 2, and 3 do not make use of any external information—they are unsupervised techniques—while item 4 requires external information. Item 5 can be performed in either a supervised or an unsupervised manner. A further distinction can be made with respect to items 3, 4, and 5: Do we want to evaluate the entire clustering or just individual clusters?

While it is possible to develop various numerical measures to assess the different aspects of cluster validity mentioned above, there are a number of challenges. First, a measure of cluster validity may be quite limited in the scope of its applicability. For example, most work on measures of clustering tendency has been done for two- or three-dimensional spatial data. Second, we need a framework to interpret any measure. If we obtain a value of 10 for a measure that evaluates how well cluster labels match externally provided class labels, does this value represent a good, fair, or poor match? The goodness of a match often can be measured by looking at the statistical distribution of this value, i.e., how likely it is that such a value occurs by chance. Finally, if a measure is too complicated to apply or to understand, then few will use it.

The evaluation measures, or indices, that are applied to judge various aspects of cluster validity are traditionally classified into the following three types.

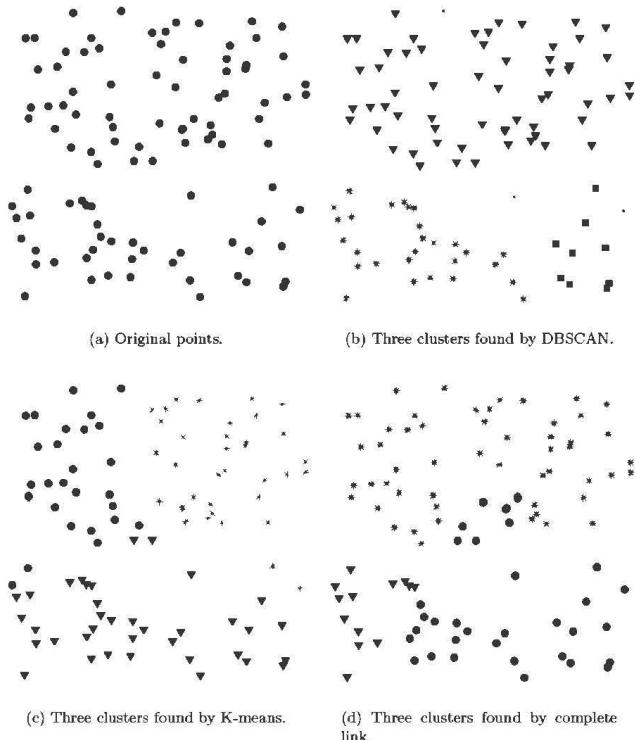


Figure 8.26. Clustering of 100 uniformly distributed points.

Unsupervised. Measures the goodness of a clustering structure without respect to external information. An example of this is the SSE. Unsupervised measures of cluster validity are often further divided into two classes: measures of cluster cohesion (compactness, tightness), which determine how closely related the objects in a cluster are, and measures of cluster separation (isolation), which determine how distinct or well-separated a cluster is from other clusters. Unsupervised measures are often called *internal indices* because they use only information present in the data set.

Supervised. Measures the extent to which the clustering structure discovered by a clustering algorithm matches some external structure. An example of supervised index is entropy, which measures how well cluster labels match externally supplied class labels. Supervised measures are often called *external indices* because they use information not present in the data set.

Relative. Compares different clusterings or clusters. A relative cluster evaluation measure is a supervised or unsupervised evaluation measure that is used for the purpose of comparison. Thus, relative measures are not actually a separate type of cluster evaluation measure, but are instead a specific use of such measures. As an example, two K-means clusterings can be compared using either the SSE or entropy.

In the remainder of this section, we provide specific details concerning cluster validity. We first describe topics related to unsupervised cluster evaluation, beginning with (1) measures based on cohesion and separation, and (2) two techniques based on the proximity matrix. Since these approaches are useful only for partitional sets of clusters, we also describe the popular cophenetic correlation coefficient, which can be used for the unsupervised evaluation of a hierarchical clustering. We end our discussion of unsupervised evaluation with brief discussions about finding the correct number of clusters and evaluating clustering tendency. We then consider supervised approaches to cluster validity, such as entropy, purity, and the Jaccard measure. We conclude this section with a short discussion of how to interpret the values of (unsupervised or supervised) validity measures.

8.5.2 Unsupervised Cluster Evaluation Using Cohesion and Separation

Many internal measures of cluster validity for partitional clustering schemes are based on the notions of cohesion or separation. In this section, we use cluster validity measures for prototype- and graph-based clustering techniques to explore these notions in some detail. In the process, we will also see some interesting relationships between prototype- and graph-based clustering.

In general, we can consider expressing overall cluster validity for a set of K clusters as a weighted sum of the validity of individual clusters,

$$\text{overall validity} = \sum_{i=1}^K w_i \text{validity}(C_i). \quad (8.8)$$

The *validity* function can be cohesion, separation, or some combination of these quantities. The weights will vary depending on the cluster validity measure. In some cases, the weights are simply 1 or the size of the cluster, while in other cases they reflect a more complicated property, such as the square root of the cohesion. See Table 8.6. If the validity function is cohesion, then higher values are better. If it is separation, then lower values are better.

Graph-Based View of Cohesion and Separation

For graph-based clusters, the cohesion of a cluster can be defined as the sum of the weights of the links in the proximity graph that connect points within the cluster. See Figure 8.27(a). (Recall that the proximity graph has data objects as nodes, a link between each pair of data objects, and a weight assigned to each link that is the proximity between the two data objects connected by the link.) Likewise, the separation between two clusters can be measured by the sum of the weights of the links from points in one cluster to points in the other cluster. This is illustrated in Figure 8.27(b).

Mathematically, cohesion and separation for a graph-based cluster can be expressed using Equations 8.9 and 8.10, respectively. The *proximity* function can be a similarity, a dissimilarity, or a simple function of these quantities.

$$\text{cohesion}(C_i) = \sum_{\substack{x \in C_i \\ y \in C_i}} \text{proximity}(x, y) \quad (8.9)$$

$$\text{separation}(C_i, C_j) = \sum_{\substack{x \in C_i \\ y \in C_j}} \text{proximity}(x, y) \quad (8.10)$$

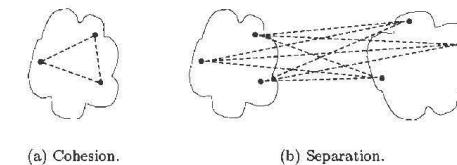


Figure 8.27. Graph-based view of cluster cohesion and separation.

Prototype-Based View of Cohesion and Separation

For prototype-based clusters, the cohesion of a cluster can be defined as the sum of the proximities with respect to the prototype (centroid or medoid) of the cluster. Similarly, the separation between two clusters can be measured by the proximity of the two cluster prototypes. This is illustrated in Figure 8.28, where the centroid of a cluster is indicated by a “+”.

Cohesion for a prototype-based cluster is given in Equation 8.11, while two measures for separation are given in Equations 8.12 and 8.13, respectively, where c_i is the prototype (centroid) of cluster C_i and c is the overall prototype (centroid). There are two measures for separation because, as we will see shortly, the separation of cluster prototypes from an overall prototype is sometimes directly related to the separation of cluster prototypes from one another. Note that Equation 8.11 is the cluster SSE if we let proximity be the squared Euclidean distance.

$$\text{cohesion}(C_i) = \sum_{x \in C_i} \text{proximity}(x, c_i) \quad (8.11)$$

$$\text{separation}(C_i, C_j) = \text{proximity}(c_i, c_j) \quad (8.12)$$

$$\text{separation}(C_i) = \text{proximity}(c_i, c) \quad (8.13)$$

Overall Measures of Cohesion and Separation

The previous definitions of cluster cohesion and separation gave us some simple and well-defined measures of cluster validity that can be combined into an overall measure of cluster validity by using a weighted sum, as indicated

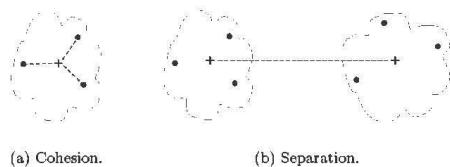


Figure 8.28. Prototype-based view of cluster cohesion and separation.

in Equation 8.8. However, we need to decide what weights to use. Not surprisingly, the weights used can vary widely, although typically they are some measure of cluster size.

Table 8.6 provides examples of validity measures based on cohesion and separation. I_1 is a measure of cohesion in terms of the pairwise proximity of objects in the cluster divided by the cluster size. I_2 is a measure of cohesion based on the sum of the proximities of objects in the cluster to the cluster centroid. E_1 is a measure of separation defined as the proximity of a cluster centroid to the overall centroid multiplied by the number of objects in the cluster. G_1 , which is a measure based on both cohesion and separation, is the sum of the pairwise proximity of all objects in the cluster with all objects outside the cluster—the total weight of the edges of the proximity graph that must be cut to separate the cluster from all other clusters—divided by the sum of the pairwise proximity of objects in the cluster.

Table 8.6. Table of graph-based cluster evaluation measures.

Name	Cluster Measure	Cluster Weight	Type
I_1	$\sum_{\substack{x \in c_i \\ y \in c_i}} proximity(x, y)$	$\frac{1}{m_i}$	graph-based cohesion
I_2	$\sum_{x \in c_i} proximity(x, c_i)$	1	prototype-based cohesion
E_1	$proximity(c_i, c)$	m_i	prototype-based separation
G_1	$\sum_{j=1}^k \sum_{\substack{x \in c_i \\ y \in c_j}} proximity(x, y)$	$\frac{1}{\sum_{\substack{x \in c_i \\ y \in c_j}} proximity(x, y)}$	graph-based separation and cohesion

Note that any unsupervised measure of cluster validity potentially can be used as an objective function for a clustering algorithm and vice versa. The CLUstering TOolkit (CLUTO) (see the bibliographic notes) uses the cluster evaluation measures described in Table 8.6, as well as some other evaluation measures not mentioned here, to drive the clustering process. It does this by using an algorithm that is similar to the incremental K-means algorithm discussed in Section 8.2.2. Specifically, each point is assigned to the cluster that produces the best value for the cluster evaluation function. The cluster evaluation measure I_2 corresponds to traditional K-means and produces clusters that have good SSE values. The other measures produce clusters that are not as good with respect to SSE, but that are more optimal with respect to the specified cluster validity measure.

Relationship between Prototype-Based Cohesion and Graph-Based Cohesion

While the graph-based and prototype-based approaches to measuring the cohesion and separation of a cluster seem distinct, for some proximity measures they are equivalent. For instance, for the SSE and points in Euclidean space, it can be shown (Equation 8.14) that the average pairwise distance between the points in a cluster is equivalent to the SSE of the cluster. See Exercise 27 on page 566.

$$\text{Cluster SSE} = \sum_{\mathbf{x} \in C_i} dist(\mathbf{c}_i, \mathbf{x})^2 = \frac{1}{2m_i} \sum_{\mathbf{x} \in C_i} \sum_{\mathbf{y} \in C_i} dist(\mathbf{x}, \mathbf{y})^2 \quad (8.14)$$

Two Approaches to Prototype-Based Separation

When proximity is measured by Euclidean distance, the traditional measure of separation between clusters is the between group sum of squares (SSB), which is the sum of the squared distance of a cluster centroid, c_i , to the overall mean, c , of all the data points. By summing the SSB over all clusters, we obtain the total SSB, which is given by Equation 8.15, where c_i is the mean of the i^{th} cluster and c is the overall mean. The higher the total SSB of a clustering, the more separated the clusters are from one another.

$$\text{Total SSB} = \sum_{i=1}^K m_i \ dist(\mathbf{c}_i, \mathbf{c})^2 \quad (8.15)$$

It is straightforward to show that the total SSB is directly related to the pairwise distances between the centroids. In particular, if the cluster sizes are

equal, i.e., $m_i = m/K$, then this relationship takes the simple form given by Equation 8.16. (See Exercise 28 on page 566.) It is this type of equivalence that motivates the definition of prototype separation in terms of both Equations 8.12 and 8.13.

$$\text{Total SSB} = \frac{1}{2K} \sum_{i=1}^K \sum_{j=1}^K \frac{m}{K} \text{dist}(c_i, c_j)^2 \quad (8.16)$$

Relationship between Cohesion and Separation

In some cases, there is also a strong relationship between cohesion and separation. Specifically, it is possible to show that the sum of the total SSE and the total SSB is a constant; i.e., that it is equal to the total sum of squares (TSS), which is the sum of squares of the distance of each point to the overall mean of the data. The importance of this result is that minimizing SSE (cohesion) is equivalent to maximizing SSB (separation).

We provide the proof of this fact below, since the approach illustrates techniques that are also applicable to proving the relationships stated in the last two sections. To simplify the notation, we assume that the data is one-dimensional, i.e., $\text{dist}(x, y) = (x - y)^2$. Also, we use the fact that the cross-term $\sum_{i=1}^K \sum_{x \in C_i} (x - c_i)(c - c_i)$ is 0. (See Exercise 29 on page 566.)

$$\begin{aligned} \text{TSS} &= \sum_{i=1}^K \sum_{x \in C_i} (x - c)^2 \\ &= \sum_{i=1}^K \sum_{x \in C_i} ((x - c_i) - (c - c_i))^2 \\ &= \sum_{i=1}^K \sum_{x \in C_i} (x - c_i)^2 - 2 \sum_{i=1}^K \sum_{x \in C_i} (x - c_i)(c - c_i) + \sum_{i=1}^K \sum_{x \in C_i} (c - c_i)^2 \\ &= \sum_{i=1}^K \sum_{x \in C_i} (x - c_i)^2 + \sum_{i=1}^K \sum_{x \in C_i} (c - c_i)^2 \\ &= \sum_{i=1}^K \sum_{x \in C_i} (x - c_i)^2 + \sum_{i=1}^K |C_i|(c - c_i)^2 \\ &= \text{SSE} + \text{SSB} \end{aligned}$$

Evaluating Individual Clusters and Objects

So far, we have focused on using cohesion and separation in the overall evaluation of a group of clusters. Many of these measures of cluster validity also can be used to evaluate individual clusters and objects. For example, we can rank individual clusters according to their specific value of cluster validity, i.e., cluster cohesion or separation. A cluster that has a high value of cohesion may be considered better than a cluster that has a lower value. This information often can be used to improve the quality of a clustering. If, for example, a cluster is not very cohesive, then we may want to split it into several subclusters. On the other hand, if two clusters are relatively cohesive, but not well separated, we may want to merge them into a single cluster.

We can also evaluate the objects within a cluster in terms of their contribution to the overall cohesion or separation of the cluster. Objects that contribute more to the cohesion and separation are near the “interior” of the cluster. Those objects for which the opposite is true are probably near the “edge” of the cluster. In the following section, we consider a cluster evaluation measure that uses an approach based on these ideas to evaluate points, clusters, and the entire set of clusters.

The Silhouette Coefficient

The popular method of silhouette coefficients combines both cohesion and separation. The following steps explain how to compute the silhouette coefficient for an individual point, a process that consists of the following three steps. We use distances, but an analogous approach can be used for similarities.

1. For the i^{th} object, calculate its average distance to all other objects in its cluster. Call this value a_i .
2. For the i^{th} object and any cluster not containing the object, calculate the object’s average distance to all the objects in the given cluster. Find the minimum such value with respect to all clusters; call this value b_i .
3. For the i^{th} object, the silhouette coefficient is $s_i = (b_i - a_i) / \max(a_i, b_i)$.

The value of the silhouette coefficient can vary between -1 and 1 . A negative value is undesirable because this corresponds to a case in which a_i , the average distance to points in the cluster, is greater than b_i , the minimum average distance to points in another cluster. We want the silhouette coefficient to be positive ($a_i < b_i$), and for a_i to be as close to 0 as possible, since the coefficient assumes its maximum value of 1 when $a_i = 0$.

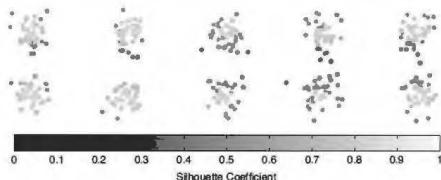


Figure 8.29. Silhouette coefficients for points in ten clusters.

We can compute the average silhouette coefficient of a cluster by simply taking the average of the silhouette coefficients of points belonging to the cluster. An overall measure of the goodness of a clustering can be obtained by computing the average silhouette coefficient of all points.

Example 8.8 (Silhouette Coefficient). Figure 8.29 shows a plot of the silhouette coefficients for points in 10 clusters. Darker shades indicate lower silhouette coefficients. ■

8.5.3 Unsupervised Cluster Evaluation Using the Proximity Matrix

In this section, we examine a couple of unsupervised approaches for assessing cluster validity that are based on the proximity matrix. The first compares an actual and idealized proximity matrix, while the second uses visualization.

Measuring Cluster Validity via Correlation

If we are given the similarity matrix for a data set and the cluster labels from a cluster analysis of the data set, then we can evaluate the “goodness” of the clustering by looking at the correlation between the similarity matrix and an ideal version of the similarity matrix based on the cluster labels. (With minor changes, the following applies to proximity matrices, but for simplicity, we discuss only similarity matrices.) More specifically, an ideal cluster is one whose points have a similarity of 1 to all points in the cluster, and a similarity of 0 to all points in other clusters. Thus, if we sort the rows and columns of the similarity matrix so that all objects belonging to the same class are together, then an ideal similarity matrix has a block diagonal structure. In other words, the similarity is non-zero, i.e., 1, inside the blocks of the similarity

matrix whose entries represent intra-cluster similarity, and 0 elsewhere. The ideal similarity matrix is constructed by creating a matrix that has one row and one column for each data point—just like an actual similarity matrix—and assigning a 1 to an entry if the associated pair of points belongs to the same cluster. All other entries are 0.

High correlation between the ideal and actual similarity matrices indicates that the points that belong to the same cluster are close to each other, while low correlation indicates the opposite. (Since the actual and ideal similarity matrices are symmetric, the correlation is calculated only among the $n(n-1)/2$ entries below or above the diagonal of the matrices.) Consequently, this is not a good measure for many density- or contiguity-based clusters, because they are not globular and may be closely intertwined with other clusters.

Example 8.9 (Correlation of Actual and Ideal Similarity Matrices). To illustrate this measure, we calculated the correlation between the ideal and actual similarity matrices for the K-means clusters shown in Figure 8.26(c) (random data) and Figure 8.30(a) (data with three well-separated clusters). The correlations were 0.5810 and 0.9235, respectively, which reflects the expected result that the clusters found by K-means in the random data are worse than the clusters found by K-means in data with well-separated clusters. ■

Judging a Clustering Visually by Its Similarity Matrix

The previous technique suggests a more general, qualitative approach to judging a set of clusters: Order the similarity matrix with respect to cluster labels and then plot it. In theory, if we have well-separated clusters, then the similarity matrix should be roughly block-diagonal. If not, then the patterns displayed in the similarity matrix can reveal the relationships between clusters. Again, all of this can be applied to dissimilarity matrices, but for simplicity, we will only discuss similarity matrices.

Example 8.10 (Visualizing a Similarity Matrix). Consider the points in Figure 8.30(a), which form three well-separated clusters. If we use K-means to group these points into three clusters, then we should have no trouble finding these clusters since they are well-separated. The separation of these clusters is illustrated by the reordered similarity matrix shown in Figure 8.30(b). (For uniformity, we have transformed the distances into similarities using the formula $s = 1 - (d - \min.d)/(\max.d - \min.d)$.) Figure 8.31 shows the reordered similarity matrices for clusters found in the random data set of Figure 8.26 by DBSCAN, K-means, and complete link.

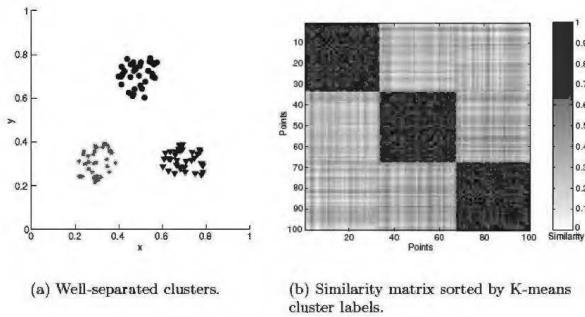


Figure 8.30. Similarity matrix for well-separated clusters.

The well-separated clusters in Figure 8.30 show a very strong, block-diagonal pattern in the reordered similarity matrix. However, there are also weak block diagonal patterns—see Figure 8.31—in the reordered similarity matrices of the clusterings found by K-means, DBSCAN, and complete link in the random data. Just as people can find patterns in clouds, data mining algorithms can find clusters in random data. While it is entertaining to find patterns in clouds, it is pointless and perhaps embarrassing to find clusters in noise. ■

This approach may seem hopelessly expensive for large data sets, since the computation of the proximity matrix takes $O(m^2)$ time, where m is the number of objects, but with sampling, this method can still be used. We can take a sample of data points from each cluster, compute the similarity between these points, and plot the result. It may be necessary to oversample small clusters and undersample large ones to obtain an adequate representation of all clusters.

8.5.4 Unsupervised Evaluation of Hierarchical Clustering

The previous approaches to cluster evaluation are intended for partitional clusterings. Here we discuss the cophenetic correlation, a popular evaluation measure for hierarchical clusterings. The cophenetic distance between two objects is the proximity at which an agglomerative hierarchical clustering tech-

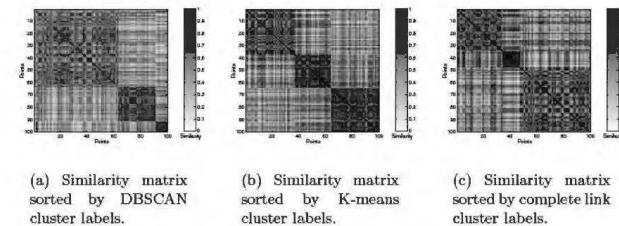


Figure 8.31. Similarity matrices for clusters from random data.

nique puts the objects in the same cluster for the first time. For example, if at some point in the agglomerative hierarchical clustering process, the smallest distance between the two clusters that are merged is 0.1, then all points in one cluster have a cophenetic distance of 0.1 with respect to the points in the other cluster. In a cophenetic distance matrix, the entries are the cophenetic distances between each pair of objects. The cophenetic distance is different for each hierarchical clustering of a set of points.

Example 8.11 (Cophenetic Distance Matrix). Table 8.7 shows the cophenetic distance matrix for the single link clustering shown in Figure 8.16. (The data for this figure consists of the 6 two-dimensional points given in Table 8.3.)

Table 8.7. Cophenetic distance matrix for single link and data in table 8.3

Point	P1	P2	P3	P4	P5	P6
P1	0	0.222	0.222	0.222	0.222	0.222
P2	0.222	0	0.148	0.151	0.139	0.148
P3	0.222	0.148	0	0.151	0.148	0.110
P4	0.222	0.151	0.151	0	0.151	0.151
P5	0.222	0.139	0.148	0.151	0	0.148
P6	0.222	0.148	0.110	0.151	0.148	0

The **CoPhenetic Correlation Coefficient (CPCC)** is the correlation between the entries of this matrix and the original dissimilarity matrix and is

a standard measure of how well a hierarchical clustering (of a particular type) fits the data. One of the most common uses of this measure is to evaluate which type of hierarchical clustering is best for a particular type of data.

Example 8.12 (Cophenetic Correlation Coefficient). We calculated the CPCC for the hierarchical clusterings shown in Figures 8.16–8.19. These values are shown in Table 8.8. The hierarchical clustering produced by the single link technique seems to fit the data less well than the clusterings produced by complete link, group average, and Ward's method.

Table 8.8. Cophenetic correlation coefficient for data of Table 8.3 and four agglomerative hierarchical clustering techniques.

Technique	CPCC
Single Link	0.44
Complete Link	0.63
Group Average	0.66
Ward's	0.64

8.5.5 Determining the Correct Number of Clusters

Various unsupervised cluster evaluation measures can be used to approximately determine the correct or natural number of clusters.

Example 8.13 (Number of Clusters). The data set of Figure 8.29 has 10 natural clusters. Figure 8.32 shows a plot of the SSE versus the number of clusters for a (hierarchical) K-means clustering of the data set, while Figure 8.33 shows the average silhouette coefficient versus the number of clusters for the same data. There is a distinct knee in the SSE and a distinct peak in the silhouette coefficient when the number of clusters is equal to 10. ■

Thus, we can try to find the natural number of clusters in a data set by looking for the number of clusters at which there is a knee, peak, or dip in the plot of the evaluation measure when it is plotted against the number of clusters. Of course, such an approach does not always work well. Clusters may be considerably more intertwined or overlapping than those shown in Figure 8.29. Also, the data may consist of nested clusters. Actually, the clusters in Figure 8.29 are somewhat nested; i.e., there are 5 pairs of clusters since the clusters are closer top to bottom than they are left to right. There is a knee that indicates this in the SSE curve, but the silhouette coefficient curve is not

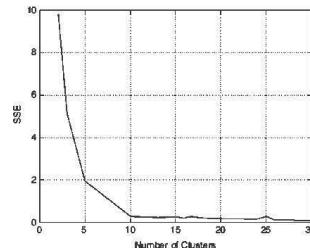


Figure 8.32. SSE versus number of clusters for the data of Figure 8.29.

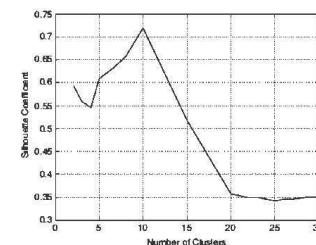


Figure 8.33. Average silhouette coefficient versus number of clusters for the data of Figure 8.29.

as clear. In summary, while caution is needed, the technique we have just described can provide insight into the number of clusters in the data.

8.5.6 Clustering Tendency

One obvious way to determine if a data set has clusters is to try to cluster it. However, almost all clustering algorithms will dutifully find clusters when given data. To address this issue, we could evaluate the resulting clusters and only claim that a data set has clusters if at least some of the clusters are of good quality. However, this approach does not address the fact the clusters in the data can be of a different type than those sought by our clustering algorithm. To handle this additional problem, we could use multiple algorithms and again evaluate the quality of the resulting clusters. If the clusters are uniformly poor, then this may indeed indicate that there are no clusters in the data.

Alternatively, and this is the focus of measures of clustering tendency, we can try to evaluate whether a data set has clusters without clustering. The most common approach, especially for data in Euclidean space, has been to use statistical tests for spatial randomness. Unfortunately, choosing the correct model, estimating the parameters, and evaluating the statistical significance of the hypothesis that the data is non-random can be quite challenging. Nonetheless, many approaches have been developed, most of them for points in low-dimensional Euclidean space.

Example 8.14 (Hopkins Statistic). For this approach, we generate p points that are randomly distributed across the data space and also sample p actual

data points. For both sets of points we find the distance to the nearest neighbor in the original data set. Let the u_i be the nearest neighbor distances of the artificially generated points, while the w_i are the nearest neighbor distances of the sample of points from the original data set. The Hopkins statistic H is then defined by Equation 8.17.

$$H = \frac{\sum_{i=1}^p w_i}{\sum_{i=1}^p u_i + \sum_{i=1}^p w_i} \quad (8.17)$$

If the randomly generated points and the sample of data points have roughly the same nearest neighbor distances, then H will be near 0.5. Values of H near 0 and 1 indicate, respectively, data that is highly clustered and data that is regularly distributed in the data space. To give an example, the Hopkins statistic for the data of Figure 8.26 was computed for $p = 20$ and 100 different trials. The average value of H was 0.56 with a standard deviation of 0.03. The same experiment was performed for the well-separated points of Figure 8.30. The average value of H was 0.95 with a standard deviation of 0.006. ■

8.5.7 Supervised Measures of Cluster Validity

When we have external information about data, it is typically in the form of externally derived class labels for the data objects. In such cases, the usual procedure is to measure the degree of correspondence between the cluster labels and the class labels. But why is this of interest? After all, if we have the class labels, then what is the point in performing a cluster analysis? Motivations for such an analysis are the comparison of clustering techniques with the “ground truth” or the evaluation of the extent to which a manual classification process can be automatically produced by cluster analysis.

We consider two different kinds of approaches. The first set of techniques use measures from classification, such as entropy, purity, and the F-measure. These measures evaluate the extent to which a cluster contains objects of a single class. The second group of methods related to the similarity measures for binary data, such as the Jaccard measure that we saw in Chapter 2. These approaches measure the extent to which two objects that are in the same class are in the same cluster and vice versa. For convenience, we will refer to these two types of measures as classification-oriented and similarity-oriented, respectively.

Classification-Oriented Measures of Cluster Validity

There are a number of measures—entropy, purity, precision, recall, and the F-measure—that are commonly used to evaluate the performance of a classification model. In the case of classification, we measure the degree to which predicted class labels correspond to actual class labels, but for the measures just mentioned, nothing fundamental is changed by using cluster labels instead of predicted class labels. Next, we quickly review the definitions of these measures, which were discussed in Chapter 4.

Entropy: The degree to which each cluster consists of objects of a single class.

For each cluster, the class distribution of the data is calculated first, i.e., for cluster j we compute p_{ij} , the probability that a member of cluster i belongs to class j as $p_{ij} = m_{ij}/m_i$, where m_i is the number of objects in cluster i and m_{ij} is the number of objects of class j in cluster i . Using this class distribution, the entropy of each cluster i is calculated using the standard formula, $e_i = -\sum_{j=1}^L p_{ij} \log_2 p_{ij}$, where L is the number of classes. The total entropy for a set of clusters is calculated as the sum of the entropies of each cluster weighted by the size of each cluster, i.e., $e = \sum_{i=1}^K \frac{m_i}{m} e_i$, where K is the number of clusters and m is the total number of data points.

Purity: Another measure of the extent to which a cluster contains objects of a single class. Using the previous terminology, the purity of cluster i is $p_i = \max_j p_{ij}$, the overall purity of a clustering is $\text{purity} = \sum_{i=1}^K \frac{m_i}{m} p_i$.

Precision: The fraction of a cluster that consists of objects of a specified class.

The precision of cluster i with respect to class j is $\text{precision}(i, j) = p_{ij}$.

Recall: The extent to which a cluster contains all objects of a specified class.

The recall of cluster i with respect to class j is $\text{recall}(i, j) = m_{ij}/m_j$, where m_j is the number of objects in class j .

F-measure A combination of both precision and recall that measures the extent to which a cluster contains *only* objects of a particular class and *all* objects of that class. The F-measure of cluster i with respect to class j is $F(i, j) = (2 \times \text{precision}(i, j) \times \text{recall}(i, j)) / (\text{precision}(i, j) + \text{recall}(i, j))$.

Example 8.15 (Supervised Evaluation Measures). We present an example to illustrate these measures. Specifically, we use K-means with the cosine similarity measure to cluster 3204 newspaper articles from the *Los Angeles*

Table 8.9. K-means clustering results for the *LA Times* document data set.

Cluster	Entertainment	Financial	Foreign	Metro	National	Sports	Entropy	Purity
1	3	5	40	506	96	27	1.2270	0.7474
2	4	7	280	29	39	2	1.1472	0.7756
3	1	1	1	7	4	671	0.1813	0.9796
4	10	162	3	119	73	2	1.7487	0.4390
5	331	22	5	70	13	23	1.3976	0.7134
6	5	358	12	212	48	13	1.5523	0.5525
Total	354	555	341	943	273	738	1.1450	0.7203

Times. These articles come from six different classes: Entertainment, Financial, Foreign, Metro, National, and Sports. Table 8.9 shows the results of a K-means clustering to find six clusters. The first column indicates the cluster, while the next six columns together form the confusion matrix; i.e., these columns indicate how the documents of each category are distributed among the clusters. The last two columns are the entropy and purity of each cluster, respectively.

Ideally, each cluster will contain documents from only one class. In reality, each cluster contains documents from many classes. Nevertheless, many clusters contain documents primarily from just one class. In particular, cluster 3, which contains mostly documents from the Sports section, is exceptionally good, both in terms of purity and entropy. The purity and entropy of the other clusters is not as good, but can typically be greatly improved if the data is partitioned into a larger number of clusters.

Precision, recall, and the F-measure can be calculated for each cluster. To give a concrete example, we consider cluster 1 and the Metro class of Table 8.9. The precision is $506/677 = 0.75$, recall is $506/943 = 0.26$, and hence, the F value is 0.39. In contrast, the F value for cluster 3 and Sports is 0.94. ■

Similarity-Oriented Measures of Cluster Validity

The measures that we discuss in this section are all based on the premise that any two objects that are in the same cluster should be in the same class and vice versa. We can view this approach to cluster validity as involving the comparison of two matrices: (1) the ideal cluster similarity matrix discussed previously, which has a 1 in the ij^{th} entry if two objects, i and j , are in the same cluster and 0, otherwise, and (2) an ideal class similarity matrix defined with respect to class labels, which has a 1 in the ij^{th} entry if

two objects, i and j , belong to the same class, and a 0 otherwise. As before, we can take the correlation of these two matrices as the measure of cluster validity. This measure is known as the Γ statistic in clustering validation literature.

Example 8.16 (Correlation between Cluster and Class Matrices). To demonstrate this idea more concretely, we give an example involving five data points, p_1, p_2, p_3, p_4, p_5 , two clusters, $C_1 = \{p_1, p_2, p_3\}$ and $C_2 = \{p_4, p_5\}$, and two classes, $L_1 = \{p_1, p_2\}$ and $L_2 = \{p_3, p_4, p_5\}$. The ideal cluster and class similarity matrices are given in Tables 8.10 and 8.11. The correlation between the entries of these two matrices is 0.359.

Table 8.10. Ideal cluster similarity matrix.

Point	p1	p2	p3	p4	p5
p1	1	1	1	0	0
p2	1	1	1	0	0
p3	1	1	1	0	0
p4	0	0	0	1	1
p5	0	0	0	1	1

Table 8.11. Ideal class similarity matrix.

Point	p1	p2	p3	p4	p5
p1	1	1	0	0	0
p2	1	1	0	0	0
p3	0	0	1	1	1
p4	0	0	1	1	1
p5	0	0	1	1	1

More generally, we can use any of the measures for binary similarity that we saw in Section 2.4.5. (For example, we can convert these two matrices into binary vectors by appending the rows.) We repeat the definitions of the four quantities used to define those similarity measures, but modify our descriptive text to fit the current context. Specifically, we need to compute the following four quantities for all pairs of distinct objects. (There are $m(m - 1)/2$ such pairs, if m is the number of objects.) ■

f_{00} = number of pairs of objects having a different class and a different cluster

f_{01} = number of pairs of objects having a different class and the same cluster

f_{10} = number of pairs of objects having the same class and a different cluster

f_{11} = number of pairs of objects having the same class and the same cluster

In particular, the simple matching coefficient, which is known as the Rand statistic in this context, and the Jaccard coefficient are two of the most frequently used cluster validity measures.

$$\text{Rand statistic} = \frac{f_{00} + f_{11}}{f_{00} + f_{01} + f_{10} + f_{11}} \quad (8.18)$$

$$\text{Jaccard coefficient} = \frac{f_{11}}{f_{01} + f_{10} + f_{11}} \quad (8.19)$$

Example 8.17 (Rand and Jaccard Measures). Based on these formulas, we can readily compute the Rand statistic and Jaccard coefficient for the example based on Tables 8.10 and 8.11. Noting that $f_{00} = 4$, $f_{01} = 2$, $f_{10} = 2$, and $f_{11} = 2$, the Rand statistic $= (2+4)/10 = 0.6$ and the Jaccard coefficient $= 2/(2+2+2) = 0.33$. ■

We also note that the four quantities, f_{00} , f_{01} , f_{10} , and f_{11} , define a *contingency* table as shown in Table 8.12.

Table 8.12. Two-way contingency table for determining whether pairs of objects are in the same class and same cluster.

	Same Cluster	Different Cluster
Same Class	f_{11}	f_{10}
Different Class	f_{01}	f_{00}

Previously, in the context of association analysis—see Section 6.7.1—we presented an extensive discussion of measures of association that can be used for this type of contingency table. (Compare Table 8.12 with Table 6.7.) Those measures can also be applied to cluster validity.

Cluster Validity for Hierarchical Clusterings

So far in this section, we have discussed supervised measures of cluster validity only for partitional clusterings. Supervised evaluation of a hierarchical clustering is more difficult for a variety of reasons, including the fact that a preexisting hierarchical structure often does not exist. Here, we will give an example of an approach for evaluating a hierarchical clustering in terms of a (flat) set of class labels, which are more likely to be available than a preexisting hierarchical structure.

The key idea of this approach is to evaluate whether a hierarchical clustering contains, for each class, at least one cluster that is relatively pure and includes most of the objects of that class. To evaluate a hierarchical clustering with respect to this goal, we compute, for each class, the F-measure for each cluster in the cluster hierarchy. For each class, we take the maximum F-measure attained for any cluster. Finally, we calculate an overall F-measure for the hierarchical clustering by computing the weighted average of all per-class F-measures, where the weights are based on the class sizes. More formally,

this hierarchical F-measure is defined as follows:

$$F = \sum_j \frac{m_j}{m} \max_i F(i, j)$$

where the maximum is taken over all clusters i at all levels, m_j is the number of objects in class j , and m is the total number of objects.

8.5.8 Assessing the Significance of Cluster Validity Measures

Cluster validity measures are intended to help us measure the goodness of the clusters that we have obtained. Indeed, they typically give us a single number as a measure of that goodness. However, we are then faced with the problem of interpreting the significance of this number, a task that may be even more difficult.

The minimum and maximum values of cluster evaluation measures may provide some guidance in many cases. For instance, by definition, a purity of 0 is bad, while a purity of 1 is good, at least if we trust our class labels and want our cluster structure to reflect the class structure. Likewise, an entropy of 0 is good, as is an SSE of 0.

Sometimes, however, there may not be a minimum or maximum value, or the scale of the data may affect the interpretation. Also, even if there are minimum and maximum values with obvious interpretations, intermediate values still need to be interpreted. In some cases, we can use an absolute standard. If, for example, we are clustering for utility, we may be willing to tolerate only a certain level of error in the approximation of our points by a cluster centroid.

But if this is not the case, then we must do something else. A common approach is to interpret the value of our validity measure in statistical terms. Specifically, we attempt to judge how likely it is that our observed value may be achieved by random chance. The value is good if it is unusual; i.e., if it is unlikely to be the result of random chance. The motivation for this approach is that we are only interested in clusters that reflect non-random structure in the data, and such structures should generate unusually high (low) values of our cluster validity measure, at least if the validity measures are designed to reflect the presence of strong cluster structure.

Example 8.18 (Significance of SSE). To show how this works, we present an example based on K-means and the SSE. Suppose that we want a measure of how good the well-separated clusters of Figure 8.30 are with respect to random data. We generate many random sets of 100 points having the same range as

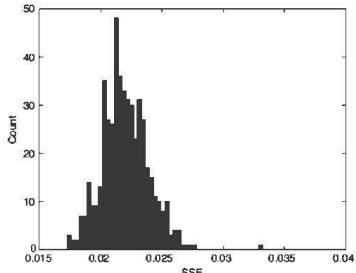


Figure 8.34. Histogram of SSE for 500 random data sets.

the points in the three clusters, find three clusters in each data set using K-means, and accumulate the distribution of SSE values for these clusterings. By using this distribution of the SSE values, we can then estimate the probability of the SSE value for the original clusters. Figure 8.34 shows the histogram of the SSE from 500 random runs. The lowest SSE shown in Figure 8.34 is 0.0173. For the three clusters of Figure 8.30, the SSE is 0.0050. We could therefore conservatively claim that there is less than a 1% chance that a clustering such as that of Figure 8.30 could occur by chance. ■

To conclude, we stress that there is more to cluster evaluation—supervised or unsupervised—than obtaining a numerical measure of cluster validity. Unless this value has a natural interpretation based on the definition of the measure, we need to interpret this value in some way. If our cluster evaluation measure is defined such that lower values indicate stronger clusters, then we can use statistics to evaluate whether the value we have obtained is unusually low, provided we have a distribution for the evaluation measure. We have presented an example of how to find such a distribution, but there is considerably more to this topic, and we refer the reader to the bibliographic notes for more pointers.

Finally, even when an evaluation measure is used as a relative measure, i.e., to compare two clusterings, we still need to assess the significance in the difference between the evaluation measures of the two clusterings. Although one value will almost always be better than another, it can be difficult to determine if the difference is significant. Note that there are two aspects to this significance: whether the difference is statistically significant (repeatable)

and whether the magnitude of the difference is meaningful with respect to the application. Many would not regard a difference of 0.1% as significant, even if it is consistently reproducible.

8.6 Bibliographic Notes

Discussion in this chapter has been most heavily influenced by the books on cluster analysis written by Jain and Dubes [396], Anderberg [374], and Kaufman and Rousseeuw [400]. Additional clustering books that may also be of interest include those by Aldenderfer and Blashfield [373], Everitt et al. [388], Hartigan [394], Mirkin [405], Murtagh [407], Romesburg [409], and Späth [413]. A more statistically oriented approach to clustering is given by the pattern recognition book of Duda et al. [385], the machine learning book of Mitchell [406], and the book on statistical learning by Hastie et al. [395]. A general survey of clustering is given by Jain et al. [397], while a survey of spatial data mining techniques is provided by Han et al. [393]. Behrkin [379] provides a survey of clustering techniques for data mining. A good source of references to clustering outside of the data mining field is the article by Arabie and Hubert [376]. A paper by Kleinberg [401] provides a discussion of some of the trade-offs that clustering algorithms make and proves that it is impossible to for a clustering algorithm to simultaneously possess three simple properties.

The K-means algorithm has a long history, but is still the subject of current research. The original K-means algorithm was proposed by MacQueen [403]. The ISODATA algorithm by Ball and Hall [377] was an early, but sophisticated version of K-means that employed various pre- and postprocessing techniques to improve on the basic algorithm. The K-means algorithm and many of its variations are described in detail in the books by Anderberg [374] and Jain and Dubes [396]. The bisecting K-means algorithm discussed in this chapter was described in a paper by Steinbach et al. [414], and an implementation of this and other clustering approaches is freely available for academic use in the CLUTO (CLUstering TOolkit) package created by Karypis [382]. Boley [380] has created a divisive partitioning clustering algorithm (PDDP) based on finding the first principal direction (component) of the data, and Savaresi and Boley [411] have explored its relationship to bisecting K-means. Recent variations of K-means are a new incremental version of K-means (Dhillon et al. [383]), X-means (Pelleg and Moore [408]), and K-harmonic means (Zhang et al [416]). Hanmerly and Elkan [392] discuss some clustering algorithms that produce better results than K-means. While some of the previously mentioned approaches address the initialization problem of K-means in some manner,

other approaches to improving K-means initialization can also be found in the work of Bradley and Fayyad [381]. Dbillon and Modha [384] present a generalization of K-means, called spherical K-means, that works with commonly used similarity functions. A general framework for K-means clustering that uses dissimilarity functions based on Bregman divergences was constructed by Banerjee et al. [378].

Hierarchical clustering techniques also have a long history. Much of the initial activity was in the area of taxonomy and is covered in books by Jardine and Sibson [398] and Sneath and Sokal [412]. General-purpose discussions of hierarchical clustering are also available in most of the clustering books mentioned above. Agglomerative hierarchical clustering is the focus of most work in the area of hierarchical clustering, but divisive approaches have also received some attention. For example, Zahn [415] describes a divisive hierarchical technique that uses the minimum spanning tree of a graph. While both divisive and agglomerative approaches typically take the view that merging (splitting) decisions are final, there has been some work by Fisher [389] and Karypis et al. [399] to overcome these limitations.

Ester et al. proposed DBSCAN [387], which was later generalized to the GDBSCAN algorithm by Sander et al. [410] in order to handle more general types of data and distance measures, such as polygons whose closeness is measured by the degree of intersection. An incremental version of DBSCAN was developed by Kriegel et al. [386]. One interesting outgrowth of DBSCAN is OPTICS (Ordering Points To Identify the Clustering Structure) (Ankerst et al. [375]), which allows the visualization of cluster structure and can also be used for hierarchical clustering.

An authoritative discussion of cluster validity, which strongly influenced the discussion in this chapter, is provided in Chapter 4 of Jain and Dubes' clustering book [396]. More recent reviews of cluster validity are those of Halkidi et al. [390, 391] and Milligan [404]. Silhouette coefficients are described in Kaufman and Rousseeuw's clustering book [400]. The source of the cohesion and separation measures in Table 8.6 is a paper by Zhao and Karypis [417], which also contains a discussion of entropy, purity, and the hierarchical F-measure. The original source of the hierarchical F-measure is an article by Larsen and Aone [402].

Bibliography

- [373] M. S. Aldenderfer and R. K. Blashfield. *Cluster Analysis*. Sage Publications, Los Angeles, 1985.
- [374] M. R. Anderberg. *Cluster Analysis for Applications*. Academic Press, New York, December 1973.
- [375] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander. OPTICS: Ordering Points To Identify the Clustering Structure. In *Proc. of 1999 ACM-SIGMOD Int'l Conf. on Management of Data*, pages 49–60, Philadelphia, Pennsylvania, June 1999. ACM Press.
- [376] P. Arabie, L. Hubert, and G. D. Soete. An overview of combinatorial data analysis. In P. Arabie, L. Hubert, and G. D. Soete, editors, *Clustering and Classification*, pages 188–217. World Scientific, Singapore, January 1996.
- [377] G. Ball and D. Hall. A Clustering Technique for Summarizing Multivariate Data. *Behavior Science*, 12:153–155, March 1967.
- [378] A. Banerjee, S. Merugu, I. S. Dhillon, and J. Ghosh. Clustering with Bregman Divergences. In *Proc. of the 2004 SIAM Int'l Conf. on Data Mining*, pages 234–245, Lake Buena Vista, FL, April 2004.
- [379] P. Berkhin. Survey of Clustering Data Mining Techniques. Technical report, Accrue Software, San Jose, CA, 2002.
- [380] D. Boley. Principal Direction Divisive Partitioning. *Data Mining and Knowledge Discovery*, 2(4):325–344, 1998.
- [381] P. S. Bradley and U. M. Fayyad. Refining Initial Points for K-Means Clustering. In *Proc. of the 15th Int'l Conf. on Machine Learning*, pages 91–99, Madison, WI, July 1998. Morgan Kaufmann Publishers Inc.
- [382] CLUTO 2.1.1: Software for Clustering High-Dimensional Datasets. www.cs.umn.edu/~karypis, November 2003.
- [383] I. S. Dhillon, Y. Guan, and J. Kogan. Iterative Clustering of High Dimensional Text Data Augmented by Local Search. In *Proc. of the 2002 IEEE Int'l Conf. on Data Mining*, pages 131–138. IEEE Computer Society, 2002.
- [384] I. S. Dhillon and D. S. Modha. Concept Decompositions for Large Sparse Text Data Using Clustering. *Machine Learning*, 42(1/2):143–175, 2001.
- [385] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. John Wiley & Sons, Inc., New York, second edition, 2001.
- [386] M. Ester, H.-P. Kriegel, J. Sander, M. Wimmer, and X. Xu. Incremental Clustering for Mining in a Data Warehousing Environment. In *Proc. of the 24th VLDB Conf.*, pages 323–333, New York City, August 1998. Morgan Kaufmann.
- [387] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proc. of the 2nd Int'l Conf. on Knowledge Discovery and Data Mining*, pages 226–231, Portland, Oregon, August 1996. AAAI Press.
- [388] B. S. Everitt, S. Landau, and M. Leese. *Cluster Analysis*. Arnold Publishers, London, fourth edition, May 2001.
- [389] D. Fisher. Iterative Optimization and Simplification of Hierarchical Clusterings. *Journal of Artificial Intelligence Research*, 4:147–179, 1996.
- [390] M. Halkidi, Y. Batistakis, and M. Vazirgiannis. Cluster validity methods: part I. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 31(2):40–45, June 2002.
- [391] M. Halkidi, Y. Batistakis, and M. Vazirgiannis. Clustering validity checking methods: part II. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 31(3):19–27, Sept. 2002.
- [392] G. Hamerly and C. Elkan. Alternatives to the k-means algorithm that find better clusterings. In *Proc. of the 11th Int'l Conf. on Information and Knowledge Management*, pages 600–607, McLean, Virginia, 2002. ACM Press.

- [393] J. Han, M. Kamber, and A. Tung. Spatial Clustering Methods in Data Mining: A review. In H. J. Miller and J. Han, editors, *Geographic Data Mining and Knowledge Discovery*, pages 188–217. Taylor and Francis, London, December 2001.
- [394] J. Hartigan. *Clustering Algorithms*. Wiley, New York, 1975.
- [395] T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, Prediction*. Springer, New York, 2001.
- [396] 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.
- [397] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323, September 1999.
- [398] N. Jardine and R. Sibson. *Mathematical Taxonomy*. Wiley, New York, 1971.
- [399] G. Karypis, E.-H. Han, and V. Kumar. Multilevel Refinement for Hierarchical Clustering. Technical Report TR 99-020, University of Minnesota, Minneapolis, MN, 1999.
- [400] 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.
- [401] J. M. Kleinberg. An Impossibility Theorem for Clustering. In *Proc. of the 16th Annual Conf. on Neural Information Processing Systems*, December, 9–14 2002.
- [402] B. Larsen and C. Aone. Fast and Effective Text Mining Using Linear-Time Document Clustering. In *Proc. of the 5th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 16–22, San Diego, California, 1999. ACM Press.
- [403] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. of the 5th Berkeley Symp. on Mathematical Statistics and Probability*, pages 281–297. University of California Press, 1967.
- [404] G. W. Milligan. Clustering Validation: Results and Implications for Applied Analyses. In P. Arabie, L. Hubert, and G. D. Soete, editors, *Clustering and Classification*, pages 345–375. World Scientific, Singapore, January 1996.
- [405] B. Mirkin. *Mathematical Classification and Clustering*, volume 11 of *Nonconvex Optimization and Its Applications*. Kluwer Academic Publishers, August 1996.
- [406] T. Mitchell. *Machine Learning*. McGraw-Hill, Boston, MA, 1997.
- [407] F. Murtagh. *Multidimensional Clustering Algorithms*. Physica-Verlag, Heidelberg and Vienna, 1985.
- [408] D. Pelleg and A. W. Moore. X-means: Extending K-means with Efficient Estimation of the Number of Clusters. In *Proc. of the 17th Intl. Conf. on Machine Learning*, pages 727–734. Morgan Kaufmann, San Francisco, CA, 2000.
- [409] C. Romesburg. *Cluster Analysis for Researchers*. Life Time Learning, Belmont, CA, 1984.
- [410] J. Sander, M. Ester, H.-P. Kriegel, and X. Xu. Density-Based Clustering in Spatial Databases: The Algorithm GDBSCAN and its Applications. *Data Mining and Knowledge Discovery*, 2(2):169–194, 1998.
- [411] S. M. Savarese and D. Boley. A comparative analysis on the bisecting K-means and the PDDP clustering algorithms. *Intelligent Data Analysis*, 8(4):345–362, 2004.
- [412] P. H. A. Sneath and R. R. Sokal. *Numerical Taxonomy*. Freeman, San Francisco, 1971.
- [413] H. Späth. *Cluster Analysis Algorithms for Data Reduction and Classification of Objects*, volume 4 of *Computers and Their Application*. Ellis Horwood Publishers, Chichester, 1980. ISBN 0-85312-141-9.
- [414] M. Steinbach, G. Karypis, and V. Kumar. A Comparison of Document Clustering Techniques. In *Proc. of KDD Workshop on Text Mining, Proc. of the 6th Intl. Conf. on Knowledge Discovery and Data Mining*, Boston, MA, August 2000.
- [415] C. T. Zahn. Graph-Theoretical Methods for Detecting and Describing Gestalt Clusters. *IEEE Transactions on Computers*, C-20(1):68–86, Jan. 1971.
- [416] B. Zhang, M. Hsu, and U. Dayal. K-Harmonic Means: A Data Clustering Algorithm. Technical Report HPL-1999-124, Hewlett Packard Laboratories, Oct. 29 1999.
- [417] Y. Zhao and G. Karypis. Empirical and theoretical comparisons of selected criterion functions for document clustering. *Machine Learning*, 55(3):311–331, 2004.

8.7 Exercises

- Consider a data set consisting of 2^{20} data vectors, where each vector has 32 components and each component is a 4-byte value. Suppose that vector quantization is used for compression and that 2^{16} prototype vectors are used. How many bytes of storage does that data set take before and after compression and what is the compression ratio?
- Find all well-separated clusters in the set of points shown in Figure 8.35.

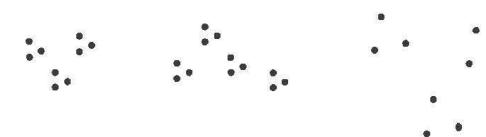


Figure 8.35. Points for Exercise 2.

- Many partitional clustering algorithms that automatically determine the number of clusters claim that this is an advantage. List two situations in which this is not the case.
- Given K equally sized clusters, the probability that a randomly chosen initial centroid will come from any given cluster is $1/K$, but the probability that each cluster will have exactly one initial centroid is much lower. (It should be clear that having one initial centroid in each cluster is a good starting situation for K-means.) In general, if there are K clusters and each cluster has n points, then the probability, p , of selecting in a sample of size K one initial centroid from each cluster is given by Equation 8.20. (This assumes sampling with replacement.) From this formula we can calculate, for example, that the chance of having one initial centroid from each of four clusters is $4!/4^4 = 0.0938$.

$$p = \frac{\text{number of ways to select one centroid from each cluster}}{\text{number of ways to select } K \text{ centroids}} = \frac{K!n^K}{(Kn)!} = \frac{K!}{K^K} \quad (8.20)$$

- (a) Plot the probability of obtaining one point from each cluster in a sample of size K for values of K between 2 and 100.
- (b) For K clusters, $K = 10, 100$, and 1000 , find the probability that a sample of size $2K$ contains at least one point from each cluster. You can use either mathematical methods or statistical simulation to determine the answer.
5. Identify the clusters in Figure 8.36 using the center-, contiguity-, and density-based definitions. Also indicate the number of clusters for each case and give a brief indication of your reasoning. Note that darkness or the number of dots indicates density. If it helps, assume center-based means K-means, contiguity-based means single link, and density-based means DBSCAN.

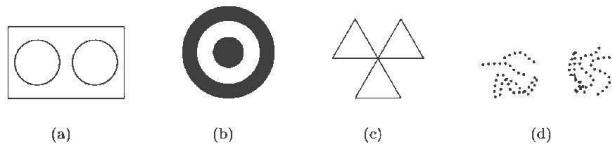


Figure 8.36. Clusters for Exercise 5.

6. For the following sets of two-dimensional points, (1) provide a sketch of how they would be split into clusters by K-means for the given number of clusters and (2) indicate approximately where the resulting centroids would be. Assume that we are using the squared error objective function. If you think that there is more than one possible solution, then please indicate whether each solution is a global or local minimum. Note that the label of each diagram in Figure 8.37 matches the corresponding part of this question, e.g., Figure 8.37(a) goes with part (a).

- (a) $K = 2$. Assuming that the points are uniformly distributed in the circle, how many possible ways are there (in theory) to partition the points into two clusters? What can you say about the positions of the two centroids? (Again, you don't need to provide exact centroid locations, just a qualitative description.)

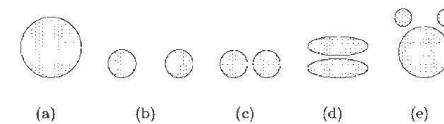


Figure 8.37. Diagrams for Exercise 6.

- (b) $K = 3$. The distance between the edges of the circles is slightly greater than the radii of the circles.
- (c) $K = 3$. The distance between the edges of the circles is much less than the radii of the circles.
- (d) $K = 2$.
- (e) $K = 3$. Hint: Use the symmetry of the situation and remember that we are looking for a rough sketch of what the result would be.

7. Suppose that for a data set
- there are m points and K clusters,
 - half the points and clusters are in “more dense” regions,
 - half the points and clusters are in “less dense” regions, and
 - the two regions are well-separated from each other.

For the given data set, which of the following should occur in order to minimize the squared error when finding K clusters:

- (a) Centroids should be equally distributed between more dense and less dense regions.
- (b) More centroids should be allocated to the less dense region.
- (c) More centroids should be allocated to the denser region.

Note: Do not get distracted by special cases or bring in factors other than density. However, if you feel the true answer is different from any given above, justify your response.

8. Consider the mean of a cluster of objects from a binary transaction data set. What are the minimum and maximum values of the components of the mean? What is the interpretation of components of the cluster mean? Which components most accurately characterize the objects in the cluster?
9. Give an example of a data set consisting of three natural clusters, for which (almost always) K-means would likely find the correct clusters, but bisecting K-means would not.

10. Would the cosine measure be the appropriate similarity measure to use with K-means clustering for time series data? Why or why not? If not, what similarity measure would be more appropriate?
11. Total SSE is the sum of the SSE for each separate attribute. What does it mean if the SSE for one variable is low for all clusters? Low for just one cluster? High for all clusters? High for just one cluster? How could you use the per variable SSE information to improve your clustering?
12. The leader algorithm (Hartigan [394]) represents each cluster using a point, known as a *leader*, and assigns each point to the cluster corresponding to the closest leader, unless this distance is above a user-specified threshold. In that case, the point becomes the leader of a new cluster.
 - (a) What are the advantages and disadvantages of the leader algorithm as compared to K-means?
 - (b) Suggest ways in which the leader algorithm might be improved.
13. The Voronoi diagram for a set of K points in the plane is a partition of all the points of the plane into K regions, such that every point (of the plane) is assigned to the closest point among the K specified points. (See Figure 8.38.) What is the relationship between Voronoi diagrams and K-means clusters? What do Voronoi diagrams tell us about the possible shapes of K-means clusters?

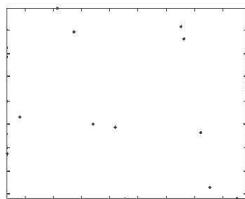


Figure 8.38. Voronoi diagram for Exercise 13.

14. You are given a data set with 100 records and are asked to cluster the data. You use K-means to cluster the data, but for all values of K , $1 \leq K \leq 100$, the K-means algorithm returns only one non-empty cluster. You then apply an incremental version of K-means, but obtain exactly the same result. How is this possible? How would single link or DBSCAN handle such data?
15. Traditional agglomerative hierarchical clustering routines merge two clusters at each step. Does it seem likely that such an approach accurately captures the

(nested) cluster structure of a set of data points? If not, explain how you might postprocess the data to obtain a more accurate view of the cluster structure.

16. Use the similarity matrix in Table 8.13 to perform single and complete link hierarchical clustering. Show your results by drawing a dendrogram. The dendrogram should clearly show the order in which the points are merged.

Table 8.13. Similarity matrix for Exercise 16.

	p1	p2	p3	p4	p5
p1	1.00	0.10	0.41	0.55	0.35
p2	0.10	1.00	0.64	0.47	0.98
p3	0.41	0.64	1.00	0.44	0.85
p4	0.55	0.47	0.44	1.00	0.76
p5	0.35	0.98	0.85	0.76	1.00

17. Hierarchical clustering is sometimes used to generate K clusters, $K > 1$ by taking the clusters at the K^{th} level of the dendrogram. (Root is at level 1.) By looking at the clusters produced in this way, we can evaluate the behavior of hierarchical clustering on different types of data and clusters, and also compare hierarchical approaches to K-means.

The following is a set of one-dimensional points: {6, 12, 18, 24, 30, 42, 48}.

- (a) For each of the following sets of initial centroids, create two clusters by assigning each point to the nearest centroid, and then calculate the total squared error for each set of two clusters. Show both the clusters and the total squared error for each set of centroids.
 - i. {18, 45}
 - ii. {15, 40}
- (b) Do both sets of centroids represent stable solutions; i.e., if the K-means algorithm was run on this set of points using the given centroids as the starting centroids, would there be any change in the clusters generated?
- (c) What are the two clusters produced by single link?
- (d) Which technique, K-means or single link, seems to produce the “most natural” clustering in this situation? (For K-means, take the clustering with the lowest squared error.)
- (e) What definition(s) of clustering does this natural clustering correspond to? (Well-separated, center-based, contiguous, or density.)
- (f) What well-known characteristic of the K-means algorithm explains the previous behavior?

18. Suppose we find K clusters using Ward's method, bisecting K-means, and ordinary K-means. Which of these solutions represents a local or global minimum? Explain.
19. Hierarchical clustering algorithms require $O(m^2 \log(m))$ time, and consequently, are impractical to use directly on larger data sets. One possible technique for reducing the time required is to sample the data set. For example, if K clusters are desired and \sqrt{m} points are sampled from the m points, then a hierarchical clustering algorithm will produce a hierarchical clustering in roughly $O(m)$ time. K clusters can be extracted from this hierarchical clustering by taking the clusters on the K^{th} level of the dendrogram. The remaining points can then be assigned to a cluster in linear time, by using various strategies. To give a specific example, the centroids of the K clusters can be computed, and then each of the $m - \sqrt{m}$ remaining points can be assigned to the cluster associated with the closest centroid.

For each of the following types of data or clusters, discuss briefly if (1) sampling will cause problems for this approach and (2) what those problems are. Assume that the sampling technique randomly chooses points from the total set of m points and that any unmentioned characteristics of the data or clusters are as optimal as possible. In other words, focus only on problems caused by the particular characteristic mentioned. Finally, assume that K is very much less than m .

- (a) Data with very different sized clusters.
 - (b) High-dimensional data.
 - (c) Data with outliers, i.e., atypical points.
 - (d) Data with highly irregular regions.
 - (e) Data with globular clusters.
 - (f) Data with widely different densities.
 - (g) Data with a small percentage of noise points.
 - (h) Non-Euclidean data.
 - (i) Euclidean data.
 - (j) Data with many and mixed attribute types.
20. Consider the following four faces shown in Figure 8.39. Again, darkness or number of dots represents density. Lines are used only to distinguish regions and do not represent points.
- (a) For each figure, could you use single link to find the patterns represented by the nose, eyes, and mouth? Explain.
 - (b) For each figure, could you use K-means to find the patterns represented by the nose, eyes, and mouth? Explain.

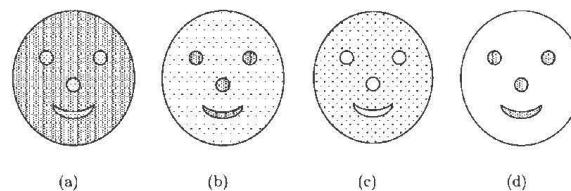


Figure 8.39. Figure for Exercise 20.

- (c) What limitation does clustering have in detecting all the patterns formed by the points in Figure 8.39(c)?

21. Compute the entropy and purity for the confusion matrix in Table 8.14.

Table 8.14. Confusion matrix for Exercise 21.

Cluster	Entertainment	Financial	Foreign	Metro	National	Sports	Total
#1	1	1	0	11	4	676	693
#2	27	89	333	827	253	33	1562
#3	326	465	8	105	16	29	949
Total	354	555	341	943	273	738	3204

22. You are given two sets of 100 points that fall within the unit square. One set of points is arranged so that the points are uniformly spaced. The other set of points is generated from a uniform distribution over the unit square.

- (a) Is there a difference between the two sets of points?
- (b) If so, which set of points will typically have a smaller SSE for $K=10$ clusters?
- (c) What will be the behavior of DBSCAN on the uniform data set? The random data set?

23. Using the data in Exercise 24, compute the silhouette coefficient for each point, each of the two clusters, and the overall clustering.

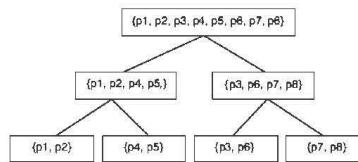
24. Given the set of cluster labels and similarity matrix shown in Tables 8.15 and 8.16, respectively, compute the correlation between the similarity matrix and the ideal similarity matrix, i.e., the matrix whose ij^{th} entry is 1 if two objects belong to the same cluster, and 0 otherwise.

Table 8.15. Table of cluster labels for Exercise 24. **Table 8.16.** Similarity matrix for Exercise 24.

Point	Cluster Label
P1	1
P2	1
P3	2
P4	2

Point	P1	P2	P3	P4
P1	1	0.8	0.65	0.55
P2	0.8	1	0.7	0.6
P3	0.65	0.7	1	0.9
P4	0.55	0.6	0.9	1

25. Compute the hierarchical F-measure for the eight objects $\{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8\}$ and hierarchical clustering shown in Figure 8.40. Class A contains points p_1, p_2 , and p_3 , while p_4, p_5, p_6, p_7 , and p_8 belong to class B.

**Figure 8.40.** Hierarchical clustering for Exercise 25.

26. Compute the cophenetic correlation coefficient for the hierarchical clusterings in Exercise 16. (You will need to convert the similarities into dissimilarities.)
27. Prove Equation 8.14.
28. Prove Equation 8.16.
29. Prove that $\sum_{i=1}^K \sum_{z \in C_i} (x - m_i)(m - m_i) = 0$. This fact was used in the proof that TSS = SSE + SSB in Section 8.5.2.
30. Clusters of documents can be summarized by finding the top terms (words) for the documents in the cluster, e.g., by taking the most frequent k terms, where k is a constant, say 10, or by taking all terms that occur more frequently than a specified threshold. Suppose that K-means is used to find clusters of both documents and words for a document data set.
- (a) How might a set of term clusters defined by the top terms in a document cluster differ from the word clusters found by clustering the terms with K-means?
 - (b) How could term clustering be used to define clusters of documents?
31. We can represent a data set as a collection of object nodes and a collection of attribute nodes, where there is a link between each object and each attribute,

and where the weight of that link is the value of the object for that attribute. For sparse data, if the value is 0, the link is omitted. Bipartite clustering attempts to partition this graph into disjoint clusters, where each cluster consists of a set of object nodes and a set of attribute nodes. The objective is to maximize the weight of links between the object and attribute nodes of a cluster, while minimizing the weight of links between object and attribute links in different clusters. This type of clustering is also known as co-clustering since the objects and attributes are clustered at the same time.

- (a) How is bipartite clustering (co-clustering) different from clustering the sets of objects and attributes separately?
 - (b) Are there any cases in which these approaches yield the same clusters?
 - (c) What are the strengths and weaknesses of co-clustering as compared to ordinary clustering?
32. In Figure 8.41, match the similarity matrices, which are sorted according to cluster labels, with the sets of points. Differences in shading and marker shape distinguish between clusters, and each set of points contains 100 points and three clusters. In the set of points labeled 2, there are three very tight, equal-sized clusters.

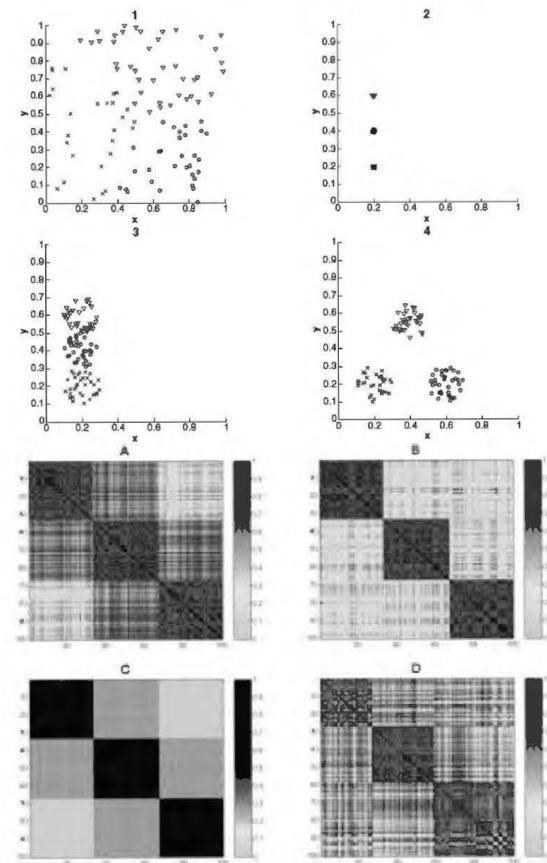


Figure 8.41. Points and similarity matrices for Exercise 32.

Errata for *Introduction to Data Mining*
by Tan, Steinbach, and Kumar.

Please send all error reports to dmbook@cs.umn.edu

Preface

Page x, last sentence of first paragraph: The email address for reporting errata has an error. Please use the one given above.

Chapter 2

1. Page 23: The title “What Is an attribute?” should be “What is an Attribute?”.
2. Page 60, equation in the last paragraph: “ $e_i = \sum_{j=1}^k p_{ij} \log_2 p_{ij}$ ” should be “ $e_i = - \sum_{j=1}^k p_{ij} \log_2 p_{ij}$ ”.
3. Page 69, fourth line from bottom: “of x and y ” should be “of \mathbf{x} and \mathbf{y} ”.
4. Page 70, second line from bottom: “ $d(\mathbf{x}, \mathbf{x}) \geq 0$ for all \mathbf{x} and \mathbf{y} ” should be “ $d(\mathbf{x}, \mathbf{y}) \geq 0$ for all \mathbf{x} and \mathbf{y} ”.
5. Page 75, second equation before the last paragraph: $\|\mathbf{y}\|$ should be 2.45, not 2.24.
6. Page 78, last sentence of the first paragraph: “ $x_k = y_k^{2n}$ ” should be “ $y_k = x_k^{2n}$ ”.
7. Page 91, Exercise 14: “what sort of similarity measure” should be “what sort of proximity measure”.

Chapter 3

1. Page 100 Table 3.1: The number of freshman should be 200 and the number of seniors should be 110, as shown in Table 1.

Table 1. Class size for students in a hypothetical college.

Class	Size	Frequency
freshman	200	0.33
sophomore	160	0.27
junior	130	0.22
senior	110	0.18

2. Page 126: Example 3.21: "Figure 3.25 is another parallel coordinates plot of the same data," should be "Figure 3.26 is another parallel coordinates plot of the same data,"

Chapter 4

1. Page 160, second line from the bottom of the second paragraph from the bottom: "the Gini index for attribute B is 0.375" should be "the Gini index for attribute B is 0.371".
2. Page 161, Figure 4.14, bottom right table. "Gini = 0.375" should be "Gini = 0.371".
3. Page 173, second from bottom line: "Figure 4.23(b) shows the training and test error rates" should be "Figure 4.23 shows the training and test error rates".
4. Page 189, sixth from bottom line, the equation should be:

$$P(X = v) = \binom{N}{v} p^v (1-p)^{N-v}.$$

5. Page 192, Equation 4.17:

$$d_t^{\text{cv}} = 0.05 \pm 1.70 \times 0.002.$$

6. Page 193, Table 4.6. Column headings are given in Table 2.

Table 2. Probability table for t -distribution.

$k - 1$	(1 - α)				
	0.90	0.95	0.975	0.99	0.995
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

7. Page 198, Exercise 3(a): "What is the entropy of this collection of training examples with respect to the positive class?" should be "What is the entropy of this collection of training examples with respect to the class attribute?".

8. Page 200, Exercise 5(c) both instances of "monotonously" should be "monotonically".

Chapter 5

1. Page 208, sixth from top line: "and op is a logical operator chosen" should be "and op is a comparison operator chosen".
2. Page 213, Algorithm 5.1 line 8: " $R \longrightarrow R \vee r$ " should be " $R \leftarrow R \vee r$ ".
3. Page 218, tenth from bottom line: "rules r_1 and r_2 given in the preceding example are 43.12 and 2" should be "rules r_1 and r_2 given in the preceding example are 63.87 and 2.83".
4. Page 233, Equation 5.16 should be:

$$P(X_i = x_i | Y = y_j) = \frac{1}{\sqrt{2\pi}\sigma_{ij}} \exp\left[-\frac{(x_i - \mu_{ij})^2}{2\sigma_{ij}^2}\right].$$

5. Page 264, sixth and seventh from bottom line, equations should be:

$$\begin{aligned} w_1 &= \sum_i \lambda_i y_i x_{i1} = 65.5261 \times 1 \times 0.3858 + 65.5261 \times -1 \times 0.4871 = -6.64. \\ w_2 &= \sum_i \lambda_i y_i x_{i2} = 65.5261 \times 1 \times 0.4687 + 65.5261 \times -1 \times 0.611 = -9.32. \end{aligned}$$

6. Page 271, Equation (5.55):

$$\Phi : (x_1, x_2) \longrightarrow (x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, 1).$$

In the transformed space, we can find the parameters $\mathbf{w} = (w_0, w_1, \dots, w_5)$ such that:

$$w_5 x_1^2 + w_4 x_2^2 + w_3 \sqrt{2}x_1 + w_2 \sqrt{2}x_2 + w_1 \sqrt{2}x_1x_2 + w_0 = 0.$$

7. Page 271, tenth from bottom line: "all the circles are located in the lower right-hand side of the diagram" should be "all the circles are located in the lower left-hand side of the diagram".
8. Page 273, second from top line: "instance z can be classified" should be "instance \mathbf{z} can be classified".

9. Page 273, Equation (5.60):

$$\begin{aligned}\Phi(\mathbf{u}) \cdot \Phi(\mathbf{v}) &= (u_1^2, u_2^2, \sqrt{2}u_1, \sqrt{2}u_2, \sqrt{2}u_1u_2, 1) \cdot (v_1^2, v_2^2, \sqrt{2}v_1, \sqrt{2}v_2, \sqrt{2}v_1v_2, 1) \\ &= u_1^2v_1^2 + u_2^2v_2^2 + 2u_1v_1 + 2u_2v_2 + 2u_1u_2v_1v_2 + 1 \\ &= (\mathbf{u} \cdot \mathbf{v} + 1)^2.\end{aligned}$$

10. Page 274, second line in the second paragraph: "A test instance \mathbf{x} is classified" should be "A test instance \mathbf{z} is classified".

11. Page 288, Equation 5.69 should be:

$$w_i^{(j+1)} = \frac{w_i^{(j)}}{Z_j} \times \begin{cases} e^{-\alpha_j} & \text{if } C_j(\mathbf{x}_i) = y_i, \\ e^{\alpha_j} & \text{if } C_j(\mathbf{x}_i) \neq y_i, \end{cases}$$

12. Page 315, Exercise 1(a): "exclusive" should be "exclusive".

13. Page 317, Exercise 5(d) and 5(e): "examples covered by R1 are discarded" should be "examples covered by R1 are discarded".

14. Page 323, Exercise 17(c) and 17(d): "part (c)" should be "part (b)".

Chapter 6

- Page 356, caption in Figure 6.17: "(with minimum support count equal to 40%" should be "(with minimum support equals to 40%)".
- Page 408, Exercise 9(b): "Use the visited leaf nodes in part (b)" should be "Use the visited leaf nodes in part (a)".
- Page 411, Exercise 15(b): " $P(A, B) \times P(A, \bar{B}) = P(A, \bar{B}) \times P(\bar{A}, B)$ " should be " $P(A, B) \times P(\bar{A}, \bar{B}) = P(A, \bar{B}) \times P(\bar{A}, B)$ ".
- Page 413, Exercise 17: "If the support" should be "Assume the support".
- Page 413, Exercise 17(d)(i): $c(\{\bar{a}\} \rightarrow \{b\}) > c(\{\bar{a}\} \rightarrow \{b\})$ should be $c(\{\bar{a}\} \rightarrow \{b\}) > c(\{a\} \rightarrow \{b\})$.

Chapter 7

- Page 421, the rule " $R_{12}^{(4)} : \text{Age} \in [20, 24] \rightarrow \text{Chat Online} = \text{No}$ " should be " $R_{12}^{(4)} : \text{Age} \in [20, 24] \rightarrow \text{Chat Online} = \text{Yes}$ ".

2. Page 437, fourth from bottom line: "events in one element must occur immediately after the events" should be "events in one element must occur after the events".

3. Page 449, Figure 7.13 should be as shown in Figure 1.

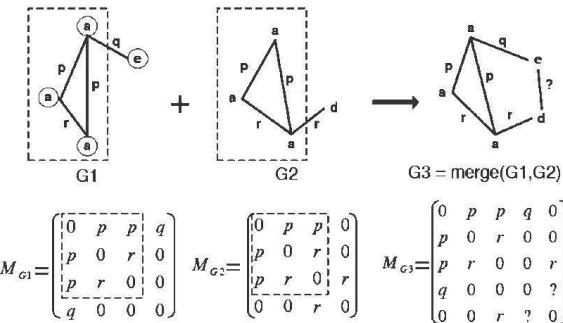


Figure 1. Vertex-growing strategy.

4. Page 450, Figure 7.14 should be as shown in Figure 2.

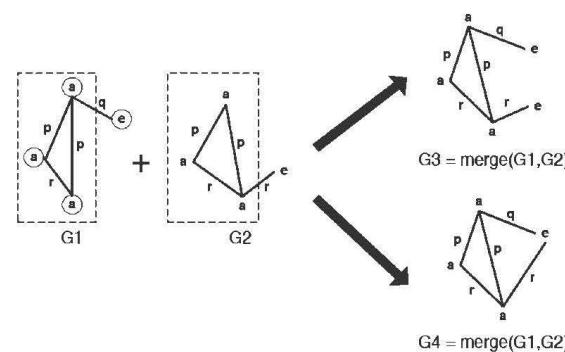


Figure 2. Edge-growing strategy.

5. Page 480, Exercise 12(c): “ $w = \langle\{A\}\{B, C, D\}\{A\}\rangle$ ” should be “ $w = \langle\{A\}\{A, B, C, D\}\{A\}\rangle$ ”.
6. Page 483, Exercise 19(a): “join the two undirected and unweighted subgraphs shown in Figure 19a” should be “join the two undirected and unweighted subgraphs shown below”.

Chapter 8

Page 519: The numbers in Tables 8.3 and 8.4 were rounded to two decimal places. Thus, if the x and y coordinates of the points given in Table 8.3 are used to compute the pairwise distances, the results don't quite match those shown in Table 8.4. The original, more precise values are given in Tables 3 and 4.

point	x coordinate	y coordinate
p1	0.4005	0.5306
p2	0.2148	0.3854
p3	0.3457	0.3156
p4	0.2652	0.1875
p5	0.0789	0.4139
p6	0.4548	0.3022

Table 3. X-Y coordinates of six points.

	p1	p2	p3	p4	p5	p6
p1	0.0000	0.2357	0.2218	0.3688	0.3421	0.2347
p2	0.2357	0.0000	0.1483	0.2042	0.1388	0.2540
p3	0.2218	0.1483	0.0000	0.1513	0.2843	0.1100
p4	0.3688	0.2042	0.1513	0.0000	0.2932	0.2216
p5	0.3421	0.1388	0.2843	0.2932	0.0000	0.3921
p6	0.2347	0.2540	0.1100	0.2216	0.3921	0.0000

Table 4. Distance Matrix for Six Points

Page 517, the fifth line of the first paragraph: “see Section 8.1.2” should be “see Section 8.1.3”.
 Page 522, the fourth line from the bottom:
 $\text{dist}(\{3, 6, 4\}, \{2, 5\}) = (0.15 + 0.28 + 0.25 + 0.39 + 0.20 + 0.29)/(6 * 2)$ should be
 $\text{dist}(\{3, 6, 4\}, \{2, 5\}) = (0.15 + 0.28 + 0.25 + 0.39 + 0.20 + 0.29)/(3 * 2)$
 Page 549, the third line of the paragraph with the heading, Entropy: “for cluster j we compute p_{ij} ” should be “for cluster i we compute p_{ij} ”.

Chapter 9

- Page 586, in Equations 9.9 and 9.10, as well as in the first line below Equation 9.10, u should be μ .
 Page 596, the first line after Equation 9.16: “the difference, $p(t) - m_j(t)$, between the centroid, $m_j(t)$, and the current object, $p(t)$ ” should be “the difference, $p(t) - m_j(t)$, between the current object, $p(t)$, and the centroid, $m_j(t)$ ”.
 Page 605, Figure 9.11: “(c) View in the xy plane” should be “(c) View in the xz plane”; “(d) View in the xy plane” should be “(d) View in the yz plane”.
 Page 618, Equation 9.17: “ $RC =$ ” should be “ $RC(C_i, C_j) =$ ”.
 Page 619, Equation 9.18: “ $RI =$ ” should be “ $RI(C_i, C_j) =$ ”.
 Page 637, the fourth line before Algorithm 9.14: “the total number of clusters is m/pq ” should be “the total number of clusters is m/q ”.
 Page 639, the first line: “Overall, m/pq clusters are produced” should be “Overall, m/q clusters are produced”.
 Page 639, the third line: “is not pq ” should be “is not q ”.
 Page 639, the fourth line: “ m/pq of the intermediate clusters” should be “ m/q of the intermediate clusters”.

Chapter 10

- Page 661, the first line below Equation 10.1: “ $\text{prob}(|x|) \geq c = \alpha$ ” should be “ $\text{prob}(|x| \geq c) = \alpha$ ”.
 Page 669, All occurrences of y should be bold (\mathbf{y}) in Equation 10.7.

Appendix A

1. Equation (A.4) should be as follows:

$$\cos(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\| \cdot \|\mathbf{v}\|}.$$

Page 700, first line of the bibliographic notes: “Strang” should be “Strang”.

Appendix C

1. Page 727, eighth from bottom line: “variance $s(X) \times s(X)/N$ ” should be “variance $s(X) \times (1 - s(X))/N$ ”.
 2. Page 727, fourth from bottom line: “variance $\text{minsup} \times \text{minsup}/N$ ” should be “variance $\text{minsup} \times (1 - \text{minsup})/N$ ”.

