

This document was created on July 3, 2003 at 12:31am.

Introduction to Data Mining

Pang-Ning Tan (ptan@cs.umn.edu),
Michael Steinbach (steinbac@cs.umn.edu), and
Vipin Kumar (kumar@cs.umn.edu)

© Pang-Ning Tan, Michael Steinbach, and Vipin Kumar 2003

Preface

This book is about data mining...

Contents

List of Figures	xii
List of Tables	xiv
1 Introduction	1
1.1 What is Data Mining?	2
1.1.1 Data Mining Tasks	3
1.1.2 Data Mining Techniques	3
1.2 What is not Data Mining?	5
1.3 Challenges of Data Mining	5
1.4 Other Issues in Data Mining	6
1.4.1 Privacy and Data Mining	6
1.4.2 Data Mining and Other Learning Techniques	7
1.5 Bibliographic Notes	7
1.6 Exercises	8
2 Data	9
2.1 Types of Data	11
2.1.1 Attributes and Measurement	11
2.1.2 Structured Data	17
2.2 Data Quality	24
2.2.1 Errors, Noise and Outliers	25
2.2.2 Missing Values	27
2.2.3 Inconsistent Values	28
2.2.4 Duplicate Data	29
2.2.5 Design of Experiments	30
2.2.6 Some final comments on data quality.	30
2.3 Data Preprocessing	30
2.3.1 Aggregation	31
2.3.2 Sampling	32
2.3.3 Dimensionality Reduction	36
2.3.4 Feature Subset Selection	38
2.3.5 Feature Creation	41

2.3.6	Discretization and Binarization	43
2.3.7	Attribute Transformation	47
2.4	Exercises	50
2.5	Bibliographic Notes	52
3	Classification	54
3.1	Problem Definition	55
3.2	General Approach to Solving a Classification Problem	56
3.3	Decision Tree Induction	58
3.3.1	How Decision Tree Works?	58
3.3.2	How to Build a Decision Tree?	60
3.3.3	Methods for Splitting	64
3.3.4	Measures for Selecting the Best Split	66
3.3.5	General Issues in Model Construction	71
3.3.6	Handling Overfitting	74
3.3.7	Handling Missing Attribute Values	79
3.3.8	Algorithm for Decision Tree Construction	81
3.3.9	Characteristics of Decision Tree Induction	82
3.4	Rule-based classifiers	85
3.4.1	How Rule-Based Classifier Works?	85
3.4.2	How to Build a Rule-Based Classifier?	88
3.4.3	Rule Ordering	89
3.4.4	Direct Methods for Rule Extraction	89
3.4.5	Indirect Methods for Rule Extraction	97
3.5	Nearest-neighbor classifiers	99
3.5.1	Algorithm	102
3.5.2	Characteristics of Nearest-neighbor classifiers	103
3.6	Bayesian classifiers	104
3.6.1	Bayes Theorem	104
3.6.2	Using Bayes Theorem for Classification	107
3.6.3	Naive Bayes Classifier	108
3.7	Artificial Neural Networks (ANN)	111
3.7.1	Back-Propagation Algorithm	115
3.7.2	Characteristics of Neural Networks	116
3.8	Support Vector Machine (SVM)	116
3.8.1	Preliminaries	118
3.8.2	How Support Vector Machine Works?	125
3.9	Ensemble Methods	129
3.9.1	Why Ensemble Methods Work?	130
3.9.2	Types of Ensemble Methods	130
3.9.3	Bagging	131
3.9.4	Boosting	132
3.10	Model Evaluation	134
3.10.1	Metric for Performance Evaluation	135

3.10.2	Methods for Performance Evaluation	139
3.10.3	Methods for Performance Comparison	143
3.11	Bibliographic Notes	151
3.12	Exercises	152
4	Association Analysis	167
4.1	Problem Definition	168
4.2	Frequent Itemset Generation	171
4.2.1	Apriori Principle	173
4.2.2	Apriori Algorithm	175
4.2.3	Candidate Itemset Generation	176
4.2.4	Support Counting	177
4.2.5	Alternative Frequent Pattern Mining Algorithms	182
4.2.6	FP-growth Algorithm	184
4.3	Rule Generation	188
4.4	Handling Continuous and Categorical Attributes	189
4.4.1	Categorical Attributes	191
4.4.2	Continuous Attributes	192
4.5	Handling Concept Hierarchy	198
4.6	Effect of Support Distribution	201
4.6.1	Multiple Minimum Support	202
4.7	Evaluation of Association Patterns	205
4.7.1	Objective Measures of Interestingness	205
4.7.2	Subjective Measures of Interestingness	219
4.8	Generalization of Association Analysis	221
4.8.1	Maximal and Closed Frequent Itemsets	221
4.8.2	Infrequent Patterns	225
4.8.3	Frequent Subgraphs	233
4.8.4	Constraint Association Rule Mining	239
4.8.5	Sequential Patterns	241
4.8.6	Spatial Associations	241
4.9	Bibliographic Notes	241
4.10	Exercises	245
4.11	Grab bag	254
4.11.1	Dependence Rules	254
4.11.2	Comparison between Classification and Association Rule Mining	256
4.11.3	General Procedure for Building Classification Models	257
4.11.4	Using association rules for rule-based classifiers	258
4.11.5	Using association patterns for Bayesian classifiers	261
4.11.6	Support and Confidence	261

5	Cluster Analysis	263
5.1	Introduction	264
5.1.1	What is cluster analysis?	264
5.1.2	What is not cluster analysis?	264
5.1.3	Different Types of Clusterings	265
5.1.4	Different Types of Clusters	268
5.2	Similarity and Distance	270
5.2.1	Similarity and Dissimilarity Between Simple Attributes	271
5.2.2	Distances Between Data Objects	272
5.2.3	Similarities Between Data Objects	274
5.2.4	Issues in Proximity Calculation	280
5.2.5	Selecting the 'right' proximity measure.	282
5.3	Density	283
5.4	Characteristics of Clustering Algorithms	283
5.5	Roadmap for Discussing Specific Clustering Techniques	285
5.6	Center-Based Clustering Techniques	285
5.6.1	K-means	286
5.6.2	K-medoid Clustering	305
5.7	Hierarchical Clustering	306
5.7.1	Agglomeration and Division	307
5.7.2	Divisive Algorithms	308
5.7.3	Basic Agglomerative Hierarchical Clustering Algorithms	308
5.7.4	Defining Proximity Between Clusters	309
5.7.5	MIN or Single Link	310
5.7.6	MAX or Complete Link or CLIQUE	311
5.7.7	Group Average	311
5.7.8	Ward's Method and Centroid methods	315
5.7.9	Key Issues in Hierarchical Clustering	315
5.7.10	The Lance-William Formula for Cluster Proximity	317
5.8	Density-Based Clustering	318
5.8.1	DBSCAN	318
5.8.2	DENCLUE	322
5.8.3	Subspace Clustering	324
5.9	Other Clustering Techniques	329
5.9.1	Fuzzy Clustering	329
5.9.2	Clustering via Mixture Models and the EM Algorithm	332
5.9.3	Self-Organizing Maps (SOM)	344
5.10	Scalable Clustering Algorithms	348
5.10.1	Birch	348
5.10.2	CURE	349
5.11	Cluster Evaluation	351
5.11.1	Overview	352
5.11.2	Measuring Cluster Validity via Correlation	355
5.11.3	Judging a Clustering Via Its Similarity Matrix	355

5.11.4	An internal measure of cluster validity: SSE	356
5.11.5	A statistical framework for cluster validity	359
5.11.6	Internal measures of cluster validity: cohesion and separation	359
5.11.7	Using External Measures of Cluster Validity	362
5.12	Bibliographic Notes	364
5.13	Exercises	365
6	Visualization	374
6.1	Introduction	374
6.1.1	What is Visualization?	374
6.1.2	Motivations for Visualization	375
6.1.3	Visualization and Different Types of Data	376
6.1.4	General Categories of Visualization	377
6.2	General Concepts	378
6.2.1	Representation: Mapping Data to Graphical Elements	378
6.2.2	Arrangement	379
6.2.3	Selection	379
6.2.4	Do's and Don'ts	381
6.3	Visualization Techniques	382
6.3.1	Visualizing One-dimensional Data	383
6.3.2	Visualizing Two-dimensional Data	390
6.3.3	Visualizing Three-dimensional Data	390
6.3.4	Visualizing Four-dimensional Data	390
6.3.5	Visualizing Higher-dimensional Data	390
6.4	Exercises	391
6.5	Bibliographic Notes	392
7	Anomaly Detection	393
8	Special Topics in Data Mining	394
8.1	Spatio-temporal Data Mining	394
8.2	Network Intrusion Detection	394
	Bibliography	394
A	Statistics	412
B	Linear Algebra	413

List of Figures

1.1	Number of Web pages indexed by the Google [©] search engine (Source: Internet Archive, http://www.archive.org).	1
1.2	Data mining techniques.	4
2.1	The measurement of the length of line segments on two different scales of measurement.	13
2.2	Different variations of record data.	19
2.3	Different variations of graph data.	21
2.4	Different variations of ordered data.	22
2.5	Noise in a time series context.	26
2.6	Noise in a spatial context.	26
2.7	Example of inconsistent time series data.	29
2.8	Standard Deviation and Coefficient of Variation for Monthly and Yearly Precipitation in Australia.	33
2.9	Example of the Loss of Structure with Sampling.	35
2.10	Finding representative points from 10 groups.	36
2.11	Decrease in relative distance between min and max distance with increasing dimensionality.	37
2.12	Flowchart of a feature subset selection process.	39
2.13	Different variations of record data.	42
2.14	Different discretization techniques.	45
2.15	Discretizing x and y attributes for four groups (classes) of points. . . .	47
3.1	Using a classification model for prediction.	56
3.2	Building a classification model for predicting tax evasion.	57
3.3	An illustrative example of the decision tree for mammal classification problem.	59
3.4	Classifying an unlabeled vertebrate.	60
3.5	Hunt's Algorithm for inducing decision trees.	61
3.6	Example of decision tree induction for a 2-dimensional data set.	62
3.7	Decision tree for the taxpayer classification problem.	63
3.8	Splitting instances based on categorical attributes.	64
3.9	Splitting the data set according to different attribute values.	65

3.10	Splitting continuous attributes.	66
3.11	Comparison between the impurity functions for two-class problems. . .	67
3.12	Splitting binary attributes.	68
3.13	Splitting categorical attributes.	69
3.14	Splitting continuous attributes.	70
3.15	Overfitting problem during model building.	73
3.16	Overfitting and Underfitting.	75
3.17	Minimum Description Length (MDL) principle.	77
3.18	Replication problem of decision trees.	83
3.19	Example of data set that cannot be partitioned optimally using test conditions involving single attributes.	84
3.20	Example of a Rule-Based Classifier for the taxpayer classification problem.	87
3.21	Difference between rule-based and class-based ordering schemes.	90
3.22	An illustrative example of the sequential covering algorithm.	91
3.23	Elimination of training instances by the sequential covering algorithm. R_1 , R_2 , and R_3 represent regions covered by three different rules. . . .	94
3.24	Converting a decision tree into classification rules.	98
3.25	Results of C4.5 and C4.5rules algorithms.	100
3.26	The 1-, 2- and 3-nearest neighbors of an instance.	101
3.27	Voronoi diagram for 1-nearest neighbor.	102
3.28	k -nearest neighbor classification with large k	102
3.29	Euclidean distance between pairs of unnormalized vectors.	104
3.30	Diagram to illustrate probability of events X , Y , $X \cap Y$, $\overline{X} \cap Y$, and $X \cap \overline{Y}$	105
3.31	The naive Bayes classifier for the taxpayer classification problem. . . .	111
3.32	Classifying boolean function using neural network.	112
3.33	Example of an artificial neural network (ANN).	113
3.34	Structure of a neuron (unit).	114
3.35	Types of activation functions for neurons.	114
3.36	An example of a two-class problem with two separating hyperplanes, B_1 and B_2	117
3.37	Finding the equation for the straight-line L	119
3.38	Stationary points of a function.	120
3.39	Plot for the function $f(x, y) = 3x^2 + 2y^3 - 2xy$	122
3.40	Decision boundary of a linear SVM.	125
3.41	Plot of α as a function of training error ϵ	133
3.42	Modifying the decision boundary (from B_1 to B_2) to reduce total mis- classification cost of a classifier.	138
3.43	A learning curve.	140
3.44	ROC curve for two 1-dimensional normal distributions.	144
3.45	ROC curves for two different classifiers.	145
3.46	Constructing an ROC curve.	146
3.47	ROC curve for the data shown in Figure 3.46.	147
3.48	Data set for Question 19.	165

3.49	Data set for Question 20.	166
4.1	The Itemset Lattice.	172
4.2	Counting the support of candidate itemsets.	172
4.3	An illustration of the Apriori principle. If $\{A, B, C\}$ is frequent, then all subsets of this itemset are frequent.	173
4.4	An illustration of support-based pruning. If $\{A, B\}$ is infrequent, then all supersets of $\{A, B\}$ are eliminated.	174
4.5	Illustration of Apriori algorithm	175
4.6	Counting the support of itemsets using hash structure.	178
4.7	Subset operation on the root of a candidate hash tree.	179
4.8	Subset operation on the left most subtree of the root of a candidate hash tree.	180
4.9	Hash tree configuration after adding the candidate itemset $\{3\ 5\ 9\}$. . .	182
4.10	Equivalent classes based on the prefix and suffix labels of itemsets. . .	184
4.11	Horizontal and vertical data format.	185
4.12	Construction of an FP-tree.	186
4.13	An illustrative example of the FP-growth algorithm for finding itemsets ending in E.	187
4.14	Pruning of association rules using confidence measure.	189
4.15	Frequency of words that appear in a collection of Los Angeles Times news snippets.	197
4.16	Example of an item taxonomy.	199
4.17	Support distribution of items for a synthetic data set created using the IBM Almaden synthetic data generator.	202
4.18	Effect of applying different minimum support threshold on number of frequent itemsets.	203
4.19	Frequent itemset generation with multiple minimum supports.	204
4.20	An example illustrating the effect of increasing item support.	212
4.21	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 . . .	214
4.22	Similarity between measures in terms of the five properties listed in Table 4.15.	215
4.23	Effect of Support Pruning on Contingency tables.	217
4.24	Similarity between measures at various ranges of support values. Note that the column labels are the same as the row labels.	218
4.25	Unexpected subjective measure	220
4.26	Maximal frequent itemset.	223
4.27	An illustrative example of frequent closed itemsets (with minimum support count equals to 2).	224
4.28	Relationships among frequent itemsets, maximal frequent itemsets, and closed frequent itemsets.	224
4.29	Relationships among infrequent patterns, negative patterns, and negatively correlated patterns.	227
4.30	Augmenting data set with negative items.	228

4.31	Mining interesting negative patterns using a concept hierarchy.	231
4.32	Indirect association between a pair of items.	231
4.33	Graph, subgraph, and induced subgraph definitions.	234
4.34	Mapping a collection of graph structures into market-basket transactions.	235
4.35	Vertex growing approach.	236
4.36	Edge growing approach.	237
4.37	Graph Isomorphism	238
4.38	Adjacency matrix and string encoding of a graph.	239
4.39	Multiplicity of Candidate Joining.	240
4.40	A summary of the research issues in mining association patterns.	242
4.41	An illustrative example for the CMAR algorithm.	259
5.1	Different ways of clustering the same set of points.	264
5.2	A hierarchical clustering of four points shown as nested clusters and as a dendrogram.	266
5.3	A non-traditional hierarchical clustering of four points shown as nested clusters and as a dendrogram.	267
5.4	Two well-separated clusters of 2 dimensional points.	269
5.5	Four center-based clusters of 2 dimensional points.	269
5.6	Eight contiguous clusters of 2 dimensional points.	269
5.7	Six dense clusters of 2 dimensional points.	270
5.8	Examples of shared property or 'conceptual' clusters.	270
5.9	Four two-dimensional points.	273
5.10	Mathematical illustration of the cosine measure.	277
5.11	Scatter plots illustrating correlations from -1 to 1.	279
5.12	Mahalanobis distance.	281
5.13	Finding three clusters in a set of 2D points.	288
5.14	Impact of Initial Centroids: Example 1.	289
5.15	Impact of Initial Centroids: Example 2.	290
5.16	Five pairs of clusters.	291
5.17	Five Pairs of Clusters with a Pair of Initial Centroids Within Each Pair of Clusters.	292
5.18	Five Pairs of Clusters with More or Fewer than Two Initial Centroids Within a Pair of Clusters.	293
5.19	Bisecting K-means on the 10 Clusters Example.	299
5.20	K-means with different size clusters.	300
5.21	K-means with different density clusters.	301
5.22	K-means with non-globular clusters.	302
5.23	Using K-means to Find Many Clusters.	303
5.24	Set of Six Two-dimensional Points.	308
5.25	Minimum Spanning Tree for Set of Six Two-dimensional Points.	309
5.26	Definition of Cluster Proximity	310
5.27	Single Link Clustering of Six Points.	312
5.28	Complete Link Clustering of Six Points.	313

5.29	Group Average Clustering of Six Points.	314
5.30	Wards Clustering of Six Points.	316
5.31	Core, Border and Noise Points for DBSCAN.	319
5.32	Four clusters embedded in noise.	322
5.33	DBSCAN Clustering of 3000 Two-Dimensional Points.	323
5.34	Example of the Gaussian influence (kernel) function and an overall density function. ($\sigma = 0.75$)	324
5.35	View of four sets of points in different subspaces.	326
5.36	Histogram showing distribution of points for the X attribute.	327
5.37	Distribution of points in the XY plane.	329
5.38	Fuzzy c-means clustering of a two-dimensional point set.	333
5.39	Histogram of 10,000 points from a normal distribution with a mean of 3 and a variance of 4.	334
5.40	Plot of the likelihood function for 7 heads out of 10 coin tosses.	335
5.41	Plot of the likelihood function for the toss of 10 coins where even and odd tosses have a different probability of being heads.	337
5.42	Figures of the expected likelihood for different iterations of the EM algorithm.	340
5.43	EM clustering of a two-dimensional point set generated from two multivariate normal distributions.	343
5.44	Two-dimensional 3 by 3 rectangular SOM neural network.	345
5.45	Distribution of reference vectors for a two-dimensional point set.	347
5.46	Clustering of 100 Randomly Distributed Points.	353
5.47	Similarity Matrix for Well-separated Clusters.	357
5.48	Similarity Matrices for Clusters from Random Data.	358
5.49	Plot of SSE versus K for 10 Clusters.	358
5.50	Plot of SSE versus K for a more complicated data set.	359
5.51	Histogram of SSE for 500 random data sets.	360
5.52	Illustration of proximity links involved in cohesion and separation measures.	361
5.53	Figures for exercise 1.	365
5.54	Figures for exercise 6.	368
5.55	Figures for exercise 10.	370
5.56	Figures for exercise 11.	371
5.57	Figures for exercise 12.	371
5.58	Figures for exercise 12.	372
5.59	Cluster tree for exercise 16.	373
6.1	The Gross Domestic Product of New Zealand (1990-1999), both adjusted for inflation is the baseline) and unadjusted.	375
6.2	Sea Surface Temperature (July, 1982).	376
6.3	A generic graph: nodes are objects, links represent relationships.	380
6.4	A generic graph: nodes are objects, links represent relationships.	381
6.5	Sepal length data from the Iris data set.	383
6.6	Stem and leaf plot for the sepal length from the Iris data set.	384

6.7	Stem and leaf plot for the sepal length from the Iris data set.	384
6.8	Dot plot for the sepal length from the Iris data set	385
6.9	Histograms of Four Iris Attributes - 10 bins.	386
6.10	Histograms of Four Iris Attributes - 20 bins.	387
6.11	Description of Box Plot.	387
6.12	Box plot for Iris attributes.	388
6.13	Empirical CDF's of Four Iris Attributes.	389
6.14	A parallel coordinates plot of the four iris attributes.	391
6.15	A parallel coordinates plot of the four iris attributes.	392

List of Tables

2.1	Definition of different attribute types.	15
2.2	Transformations that define attribute levels.	16
2.3	Conversion of a categorical attribute to a three binary attributes. . . .	43
3.1	The vertebrate data set.	55
3.2	Confusion matrix for a 2-class problem.	58
3.3	Decision Tree Algorithm.	81
3.4	The Sequential Covering Algorithm.	90
3.5	Comparison between various rule-based classifiers.	97
3.6	The vertebrate data set.	99
3.7	k -nearest neighbor classification algorithm.	103
3.8	Data set for Problem 2.	158
3.9	Data set for Problem 7.	161
4.1	An example of market-basket transactions.	167
4.2	A binary 0/1 representation of market-basket data.	169
4.3	The Apriori algorithm.	176
4.4	Algorithm for generating rules from frequent itemsets.	190
4.5	Example of Web data for mining association rules.	191
4.6	Example of Web data after binarizing the categorical attributes.	192
4.7	Example of Web data after discretizing continuous attributes.	193
4.8	Example of mining association rules in text data.	196
4.9	Normalized document-term matrix.	197
4.10	A 2-way contingency table for variables A and B	207
4.11	Definitions of objective interestingness measures.	209
4.12	Example of contingency tables.	210
4.13	Rankings of contingency tables using objective interestingness measures.	210
4.14	Example of a contingency table.	211
4.15	Properties of objective interestingness measures.	213
4.16	The Grade-Gender example.	215
4.17	Effect of high-support items on interest factor.	219
4.18	A transaction database for mining closed itemsets.	222
4.19	Example of market-basket transactions.	245

4.20	Market basket transactions.	247
4.21	Example of market-basket transactions.	248
4.22	Example of market-basket transactions.	251
4.23	Example of numeric data set.	252
4.24	Example of numeric data set.	253
5.1	Similarity and dissimilarity for simple attributes	272
5.2	X-Y coordinates of four points.	273
5.3	Euclidean distance matrix for Table 5.2.	273
5.4	L_1 distance matrix for Table 5.2.	273
5.5	L_∞ distance matrix for Table 5.2.	273
5.6	X-Y coordinates of six points.	307
5.7	Distance Matrix for Six Points	307
5.8	Table of Lance-William Coefficients for Common Hierarchical Clustering Approaches	318
5.9	K-means Clustering Results for LA Document Data Set	363
5.10	Confusion matrix for exercise 13	371
5.11	Table of cluster labels for Exercise 14	372
5.12	Similarity matrix for Exercise 14	372
6.1	A table of nine objects (rows) with six binary attributes (columns). . .	380
6.2	A table of nine objects (rows) with six binary attributes (columns) per- muted so that the relationships of rows and columns is clear.	380

Introduction

Data, data everywhere

Rapid progress in data collection and data storage technology has enabled many organizations to accumulate huge amount of observational, experimental, and operational data from their daily activities. For example, retailers are collecting large volume of point-of-sale data at their departmental and online stores, while government agencies such as NASA are constantly producing high-speed streaming data from their Earth-observing satellites.

To illustrate how much the size of a database has grown over the years, Figure 1.1 shows an example of the number of Web pages indexed by a popular Internet search engine since 1998. The graph is indicative of the remarkable growth the Internet has shown in the past decade.

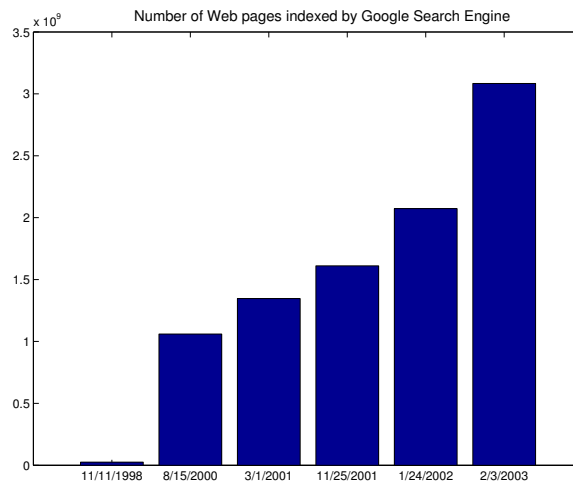


Figure 1.1. Number of Web pages indexed by the Google[®] search engine (Source: Internet Archive, <http://www.archive.org>).

The data collected by various organizations are useful for different purposes. Business enterprises are interested in utilizing such data to provide a better, customized service to their customers just to gain a significant edge over their competitors. The satellite images and remote sensing data collected by NASA can help scientists to advance their current understanding of how the universe works and how the Earth system is responding to natural and human-induced changes.

However, getting information out of such massive data sets is like *drinking from a fire hose*. As the amount of data collected by these organizations continues to grow, it is becoming increasingly difficult for analysts to manually sift through the data and find useful information. In particular, the sheer size of the data overwhelms the capacity of many traditional statistical data analysis techniques. Fueled by the tremendous need to rapidly analyze and summarize the data, researchers have turned to *data mining* techniques.

This book aims to provide a theoretical and practical foundation to the field of data mining. It is intended for both beginners and practitioners of data mining, and written in a self-contained way to help readers who may not be familiar with the mathematical rigor that underpins many of the techniques used in data mining. Most notably, the fundamental concepts behind a variety of data mining techniques and algorithms are explained. The practical challenges encountered when applying these techniques to real applications will also be highlighted.

1.1 What is Data Mining?

Imagine a sailor drifting aimlessly in the ocean. Even though he is surrounded by water, he still feels thirsty because the water surrounding him is not fit to drink.

In a nutshell, data mining is the task of *discovering interesting knowledge automatically from large data repositories*. Interesting knowledge has different meanings to different people. From a business perspective, the knowledge is interesting if it can be used by analysts or managers to make the right business decisions. For Earth Scientists, the knowledge is interesting if it reveals previously unknown information about changes in the Earth system. For system administrators, the knowledge is interesting if it indicates unauthorized or illegitimate use of system resources.

Data mining is often considered to be an integral part of another process, called *Knowledge Discovery in Databases* (or KDD). KDD refers to the overall process of turning raw data into interesting knowledge and comprises of a series of transformation steps, including data preprocessing, data mining, and postprocessing.

The objective of data preprocessing is to convert the data into the right format for subsequent analysis by selecting the appropriate data segments and extracting attributes that are relevant to the data mining task (feature selection and construction). For many practical applications, more than half of the knowledge discovery efforts are devoted to data preprocessing. A more detailed discussion about data and its preprocessing issues is presented in Chapter 2.

Postprocessing includes all additional operations one can perform to make the data mining results more accessible and easier to be interpreted by analysts. For example, the results can be sorted or filtered according to various *measures* to remove uninteresting patterns. In addition, *visualization* techniques can be applied to help analysts explore and interact with the data mining results.

1.1.1 Data Mining Tasks

In general, data mining tasks can be divided into two major categories:

Predictive Task. The task is to use some of the variables in the data to predict the values of other variables. For example, in Web mining, e-tailers are interested in predicting online users who will make a purchase at their Web site. Other examples include biologists who would like to predict the functionalities of protein structures in the human genome and stock market analysts who would like to forecast the future prices of various stocks.

Descriptive Task. The task is to find human-interpretable patterns that can describe the underlying relationships in the data. For example, Earth Scientists are interested to know what are the primary forcings influencing the climate patterns observed at various parts of the Earth. In network intrusion detection, analysts want to know the kinds of cyber-attacks launched against their network system. In document analysis, one could be interested in finding the set of documents that share similar topics.

1.1.2 Data Mining Techniques

The data mining tasks can be accomplished using a variety of data mining techniques, as shown in Figure 1.2.

Predictive Modeling. This technique is used primarily for predictive data mining tasks. The input data for predictive modeling techniques consists of two distinct types of variables: (1) explanatory variables, which define the essential properties of the data, and (2) target variable, whose value is to be predicted by the data mining task. For the Web Mining example given in the previous section, the input variables correspond to the demographic features of online users (such as age, gender, and salary) along with their browsing activities (*e.g.*, what are the pages accessed and how long are the pages being viewed.) The target variable is **Buy**, which has binary values, **Yes** or **No**, to denote whether the user will buy or not buy from the Web site. Predictive modeling techniques can be further divided into two categories: *classification* and *regression*. Classification techniques are used to predict the values of discrete target variables; such as the **Buy** variable for online users at a Web site. For example, they can be used to predict whether a customer will most likely be lost to a competitor (*i.e.*, customer churn or attrition) and to determine the

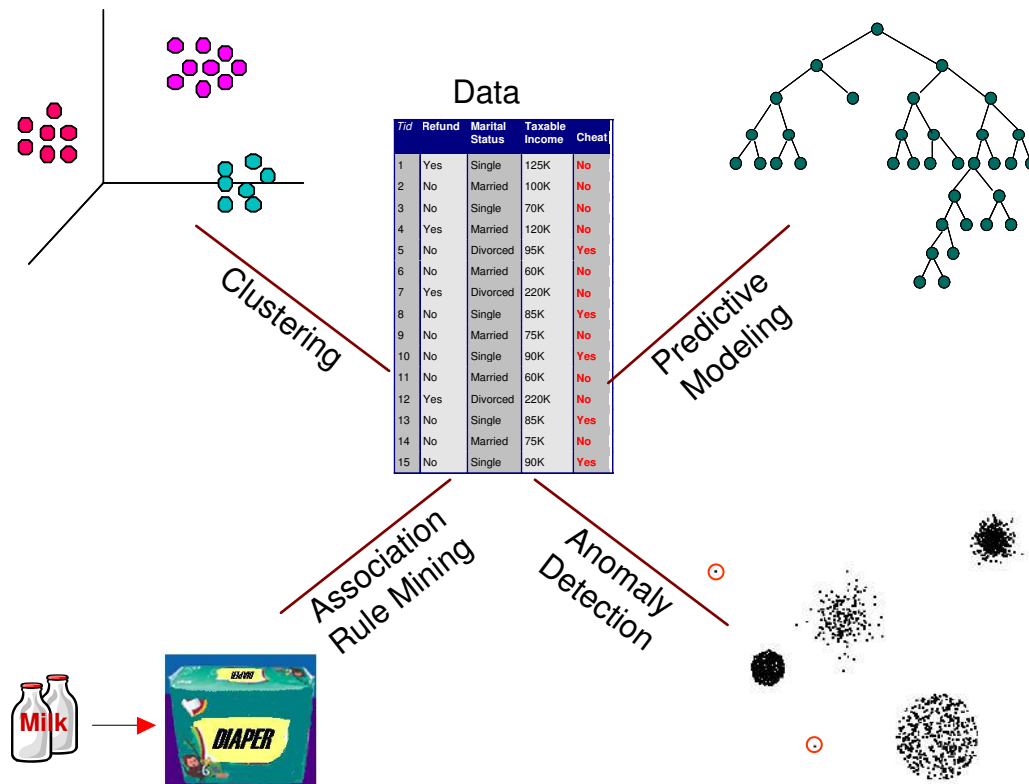


Figure 1.2. Data mining techniques.

category of a star or galaxy for sky survey cataloging. Regression techniques are used to predict the values of continuous target variables. For example, they can be applied to forecast the future price of a stock.

Association Rule Mining. This technique produces a set of dependence rules that predict the occurrence of a variable given the occurrences of other variables. For example, it can be used to identify products that are often bought together by sufficiently many customers, a task that is also known as *market basket analysis*.

Clustering. This technique tries to find homogeneous groupings of data points so that data points that belong in one cluster are more similar to each other compared to data points from a separate cluster. For example, clustering can be used to perform market segmentation of customers, document categorization, and land segmentation according to vegetation cover.

Anomaly Detection. This technique tries to find outliers or data points that are significantly different than the rest of the points in the data set. Anomaly detection techniques has been used to detect network intrusions and to predict fraudulent credit card transactions.

1.2 What is not Data Mining?

Besides data mining, there are other ways to extract information from large data repositories. Below, we illustrate some examples of activities that are not considered to be data mining

- Searching the Internet for Web pages about certain topics. This type of activity is considered to be more of an information retrieval task rather than a data mining task. Nevertheless, the technology employed by the search engines to determine which Web page is relevant to the query is based on data mining techniques. For example, clustering techniques can be used to group together related pages to facilitate the indexing of Web pages while classification can be used to predict which page is relevant to the query.
- Looking up the telephone number of a person. Directory lookup operations or database querying are not considered to be part of a data mining task. (However, these operations are needed during the mining process by the data mining algorithms.) Finding the address where John Doe lives is not a data mining task. Finding certain names that are most prevalent in certain parts of the United States (e.g., O'Brien, O'Rourke, or O'Reilly in Boston areas) is a data mining task.

Some of the key features of data mining that distinguishes it from other information extraction tasks include:

Intelligence. Some intelligence is needed to automatically extract information from the data, which is why a simple directory lookup or database querying is not data mining.

Scale of data. The amount of data from which knowledge should be extracted is large enough so that it is impossible for domain experts to extract such pattern manually (before knowing that such patterns actually exist in the data). For example, computing the average GPA score of five students is not data mining.

Complexity of Analysis. A data mining task is usually too complicated to be solved using simple pen-and-paper techniques. Thus, predicting the behavior of online users is part of data mining, whereas counting the number of hits each Web page receives is not.

1.3 Challenges of Data Mining

There are various challenges to data mining:

Scalability. Scalable techniques are needed to handle the massive scale of data. This may require the use of efficient data structures for data storing and indexing, efficient disk-resident and perhaps, parallel algorithms for .

Dimensionality. In some application domains, the number of dimensions (or attributes of a record) can be very large, which makes it difficult to be analyzed efficiently. For example, in bioinformatics, the development of advanced microarray technologies allows us to analyze gene expression data with thousands of attributes. The dimensionality of a data mining problem may also increase substantially due to the temporal, spatial, and sequential nature of the data.

Heterogeneity. Traditional statistical methods often have to deal with simple data types such as continuous and categorical attributes of the same type. However, recent years have seen more complicated data types being introduced such as graph-based data, free-form text data, structured and semi-structured data types. Traditional data analysis techniques may have to be modified to handle the heterogeneity of the data.

Imperfection. Many data sets are not perfect, as it may contain some missing values due to difficulties in obtaining the data, or noise, due to difficulties in getting precise values of certain measurements. As a result, even if a perfect data mining algorithm can be developed, information discovered from the data may be wrong due to imperfection in the data.

Data Ownership and Distribution. As the volume of data increases, it is no longer possible or safe to keep all the data in the same place. Therefore, the need for distributed data mining approaches has increased. The challenges for developing a distributed data mining solution include the need for efficient algorithm to cope with the distributed and possibly, heterogeneous data sets, the need to minimize the cost of communication, along with data security and data ownership issues.

1.4 Other Issues in Data Mining

This section describes some of the important issues concerning the use of data mining.

1.4.1 Privacy and Data Mining

Privacy and data mining are two notions with seemingly contradicting goals. While privacy intends to conceal information from others, data mining attempts to reveal interesting information about the data. A good example of such conflicting viewpoints can be found in the Web domain, where the availability of technology for tracking and analyzing user activities up to individual mouse-click level has worried many privacy advocates. On the one hand, data miners want to get their hand on as much information as possible about the users, while on the other, users want to surf as anonymously as possible across the Internet. Concerns about information privacy has become a subject of intense debate. Clarke coined the term *data surveillance* to refer to the “systematic use of personal data systems in the investigation and

monitoring of the actions or communications of one or more persons”. This type of activity encompasses the *personal surveillance* of an identified individual and *mass surveillance* of groups of people. The significance of this issue is so important that in 1995, the European Union (EU) has enacted a directive that requires its member countries to implement legislation for protecting the privacy of personal data collected by any organizations. Such a directive prevents the transfer of personal data of EU residents to non-EU countries that do not have similar laws. From a data mining perspective, various methods have been proposed to address the privacy concerns. First is the idea of aggregating attribute values of individuals into coarser concepts so as to avoid the direct access and manipulation of the personal data. Second is the idea of perturbing the data by swapping some of its values, replacing the data set with samples from the same population distribution, and adding noise to the original data. There is another school of thought who believes that many people are willing to disclose their personal information as long as they receive ample compensation.

1.4.2 Data Mining and Other Learning Techniques

Data mining draws upon many of the ideas originating from multidisciplinary research areas including machine learning, pattern recognition, database systems, and statistics. Techniques from these areas may not be easily applied due to the enormity of the data. For example, the first assembly of human genome by Celera Genomics@corporation consumes more than 80 terabytes of data, which would be almost impossible to analyze without using data mining techniques.

1.5 Bibliographic Notes

There are several textbooks in data mining including the work by Han *et al.* [74], Hand *et al.* [77], Witten *et al.* [197], and Dunham [51]. Other books related to several important topics in data mining include the books by Hastie *et al.* [78], Duda *et al.* [49], Cherkassky *et al.* [32].

There is also an extensive number of conferences related to data mining. The main conferences dedicated to the field of data mining include the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, the IEEE International Conference on Data Mining, and the SIAM International Conference on Data Mining. 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 International Conference on Data Engineering (ICDE), the International Conference on Machine Learning (ICML), and the National Conference on Artificial Intelligence (AAAI).

The journal publications on data mining include the Data Mining and Knowledge Discovery (DMKD), IEEE Transactions on Knowledge and Data Engineering (TKDE), Intelligent Data Analysis, Knowledge and Information Systems (KAIS), and Journal of Intelligent Information Systems.

1.6 Exercises

1. For each activity described below, determine whether it is a data mining task:
 - (a) Dividing customers of a company according to their gender.
 - (b) Dividing customers of a company according to their profitability.
 - (c) Computing the total sales of a company.
 - (d) Sorting the student database based on the student ids.
 - (e) Predicting the outcomes of tossing a fair dice.
 - (f) Predicting the future stock price of a company using its historical records.
 - (g) Monitoring the heart rate of a patient for abnormalities.
 - (h) Monitoring the seismic wave for earthquake activities.
 - (i) Extracting the frequencies of a sound wave.
2. Suppose you are working as a data mining consultant for an Internet Search Engine company. Describe how data mining can help the company. Give examples for which techniques such as (1) clustering, (2) classification, (3) association rule mining, (4) anomaly detection can be applied.
3. For each of the data given below, explain whether data privacy is an important issue.
 - (a) Census data collected from 1900-1950.
 - (b) IP addresses and visit times of Web users who are visiting your Web site.
 - (c) Images from NASA observation satellites.
 - (d) Name and addresses of people from the telephone book.
 - (e) Name and email addresses collected from the Web.

Data

This chapter is about data. More specifically, it is about several data-related questions that are important for successful data mining:

What type of data do we have?

What is the quality of the data?

What preprocessing steps can or should we apply to the data to make it more suitable for data mining?

To illustrate the importance of these questions, consider the following (completely) hypothetical situation. You receive an email from a medical researcher concerning a project that he previously discussed with you.

Hi,

I've attached the data file that I mentioned in the 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 100 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 other people to arrive, you happen to strike up a conversation with a statistician who is working on the project. When she learns that you have also been working on 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 500 patients?

Data Miner: 500? I only received data for 100 patients.

Statistician: I wonder if those are the 100 cases that I was having a hard time with?

Data Miner: Possibly. I did manage to get some results though.

Statistician: Amazing. There were so many data issues with this set of 100 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 serving as a missing value, but due to 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 out of the 100 that you were given have that problem. For the other 400 cases, field 4 was quite important in achieving good prediction.

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 were really quite good. Field 1 was 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.

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

2.1 Types of Data

There are two fundamental types of data — simple attributes and structured data. (But, do not confuse ‘types of data’ with ‘data types’ in programming languages.) *Attributes* — the ‘simple’ is omitted from here on — capture basic characteristics of an object or an event, e.g., the mass of a physical object or the time at which an event occurred, and can be regarded as indivisible chunks of information, at least for the purposes of a particular data analysis. (Other names for *attribute* are *variable*, *characteristic*, *feature*, or *observation*.) A collection of attributes describes an *object*. (Other names for an *object* are *record*, *point*, *case*, *sample*, *entity*, or *item*.) Structured data consists of collections of objects, and optionally, information about relationships among objects and/or relationships among attributes. Examples are records in a relational database, a set of HTML pages on the World Wide Web, a financial index, such as the Dow Jones Industrial Average, and proteins, as represented in a protein data bank. A more detailed discussion of structured data will be provided shortly, after a discussion of the different types of attributes.

2.1.1 Attributes and Measurement

In this section we indicate how to partially answer the question, “What type of data do we have?”, by answering the question, “What type of attributes do we have”? In particular, we consider the following questions:

- What is an attribute?
- What do we mean by the type of an attribute and why is it important?
- What are the different types of attributes?

What is an attribute?

We start with a slightly more detailed definition of an *attribute*.

Definition 1 An *attribute* is a property or characteristic of an object, system, or event that may vary, either from object to object, e.g., the eye color of a person, or from time to time, e.g., the temperature of an object.

Therefore, at the most basic level, attributes are not about numbers at all. However, to deal with attributes more precisely and to bring to bear the tools of mathematics and data mining, it is necessary to associate numbers or symbols with attributes. This involves *measurement* and *measurement scales*, which we define next.

Definition 2 *Measurement* is the process of assigning a number or symbol to an attribute of an object.

Definition 3 A *measurement scale* is a rule (function) for specifying which number or symbol is associated with an attribute of an object.

A measurement scale is a function that associates a number or symbol with an attribute of an object, and the process of measurement is the application of that function to a specific object.

In everyday usage, the measurements, i.e., the numbers or symbols, assigned to an attribute are called ‘values.’ However, what then, is the proper term for the different ‘states’ of an attribute to which these measurements correspond? Unfortunately there seems to be no general convention, and this is one reason that many discussions involving measurement are hard to understand. Thus, to avoid confusion, the reader should always keep in mind the distinction between an attribute and the values (numbers or symbols) that are used to represent different ‘states’ of the attribute.

While above definitions might seem a bit abstract, we engage in the process of measurement all the time, e.g., when we step on a bathroom scale to check our weight or when 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. We are also quite familiar with the fact that an attribute can be measured on different measurement scales, e.g., length in meters or feet and mass in pounds or kilograms. Indeed, the fact that an attribute can be measured on different scales is another demonstration of the fact that an attribute is distinct from the numbers used to measure it.

What do we mean by the type of an attribute and why is it important?

A consequence of the previous section is that the properties of an attribute need not be the same as the properties of the numbers or symbols used to measure it. (To simplify our terminology, we will, for the rest of this discussion, consider only measurements that are numbers.) In other words, the numbers used to represent an attribute, i.e., the values of an attribute, may have properties that are not properties of the attribute itself, and conversely, the properties of an attribute may not be reflected by its values. We will illustrate these possibilities with a couple of examples

First, consider two attributes that might be associated with an employee: *ID* and *age*, in years. Both of these attributes can be represented as integers, but 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, and consequently, the only valid operation for employee IDs should be to test whether they are equal. However, there is no hint of this limitation when integers are used to represent the employee ID attribute. For the age attribute, however, the properties of the integers used to represent age are very much the properties of the attribute. Even so, the correspondence is not complete, e.g., ages have a maximum, while integers do not.

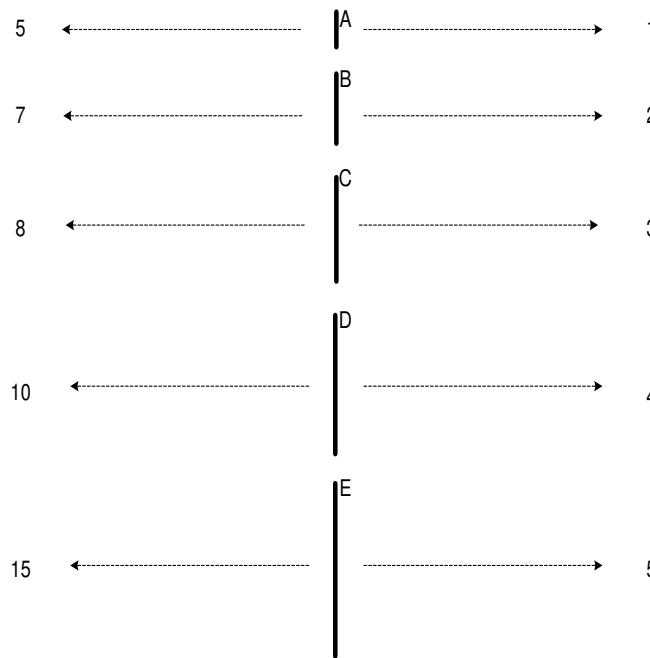


Figure 2.1. The measurement of the length of line segments on two different scales of measurement.

Second, consider Figure 2.1, which shows some objects, i.e., line segments, and how an attribute of these objects, i.e., length, can be associated with (mapped to) numbers in two different ways. If we append line segment A to itself, the resulting line segment will be the same length as line segment B. Thus, in a very real (physical) sense, line segment A is twice the length of line segment B. This is captured by the measurements on the right hand side of the figure, but not by those on the left hand side. This illustrates that an attribute can be measured in a way that does not capture all the properties of the attribute.

We can now answer the question, “What do we mean by the type of an attribute and why is it important?”, at least in general terms. The type of an attribute should tell us what properties of the attribute are reflected in the values used to measure it. Thus, knowing the type of an attribute is important because it tells us which properties of an attribute’s values map to the underlying properties of the attribute, and thus, allows us to avoid doing foolish things, such as computing the average employee ID.

As we have defined it, the type of an attribute depends both on the attribute and on how it is measured, i.e., its measurement scale. Since a particular measurement scale is always associated with one attribute, it is perhaps more precise to speak of the *type of a measurement scale* or more simply, the *type of a scale*, and indeed, this is common terminology. However, for consistency with the preceding discussion, we will stick with our original terminology, but when we use the term ‘attribute type,’ a particular measurement scale is always implied.

What are the different types of attributes?

A useful (and simple) way to specify the type of an attribute is to specify which properties of numbers correspond to underlying properties of the attribute. For example, an attribute such as length has many of the properties of numbers, e.g., it makes sense to compare and order objects by length, as well as to talk about the differences and ratios of length. More specifically, the following properties (operations) of numbers are typically used.

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 2.1 gives a definition of these types, along with information on the statistical operations that are valid for each type. (One of the initial motivations for defining these types was to be precise about which statistical operations were valid for what sorts of data. The appropriateness of various data mining techniques for different types of attributes is discussed when these techniques are described later in the book.) To understand this table, it is necessary to note that each attribute type possesses all the properties and operations of the attribute types below it. Thus, any property or operation that is valid for nominal, ordinal, and interval attributes is also valid for ratio attributes. Said another way, the definition of the attribute types is cumulative. (Hence, while there are four properties and four attribute types, there is *not* a one to one correspondence between properties and attribute types.) For this reason, these attribute types can be referred to as attribute levels, although the terms ‘measurement levels’ and ‘scale types’ are more common.

Nominal and ordinal attributes are collectively referred to as *categorical* or *qualitative* attributes. As the name suggests, qualitative attributes, such as employee ID, do not possess most of the properties of numbers. Even if represented by numbers, they should be treated more like symbols. The remaining two types of attributes, interval and ratio, are collectively referred to as *quantitative* (*numeric* or *continuous*) attributes. Quantitative attributes are represented by numbers and have most of the properties of numbers.

We can also describe the types of attributes in terms of transformations that do not change the meaning of an attribute. (Indeed, S. S. Stevens, the psychologist who originally defined the types of attributes shown in Table 2.1, defined them in terms of these *permissible* transformations.) For example, the meaning of a length attribute is unchanged if it is measured in meters instead of feet. This ‘invariance’ is important because the statistical operations that make sense for a particular type of attribute are those that will yield the same results even if the attribute is

Table 2.1. Definition of different attribute types.

Attribute Type	Description	Examples	Operations
Nominal	The values of a nominal attribute are just different names, i.e., nominal attributes provide only enough information to distinguish one object from another. (=, \neq)	zip codes, employee ID numbers, eye color, sex: $\{male, female\}$	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
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

Table 2.2. Transformations that define attribute levels.

Attribute Level	Transformation	Comments
Nominal	Any one-to-one mapping, e.g., a permutation of values	If all employee ID numbers were reassigned, would it 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 $\{0.5, 1, 10\}$.
Interval	$new_value = a * old_value + b$, a and b constants.	The Fahrenheit and Celsius temperature scales differ in terms of where their zero value is and the size of a degree (unit).
Ratio	$new_value = a * old_value$	Length can be measured in meters or feet.

transformed with a transformation that preserves the attributes meaning. More generally, data analysis should not yield different results depending on the scale of measurement that was used. (Note that this does not mean that the values of a result will be the same regardless of scale, but that the meaning will be the same. Thus, the average length of objects is different when measured in meters than in feet, but both represent the same length.) Table 2.2 shows the permissible, i.e., meaning preserving, transformations for the four attribute types of Table 2.1.

Temperature provides a good illustration of some of the concepts that have just been described. First, notice that 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° . However, this is not true when temperature is measured on either the Celsius or Fahrenheit scale, because, physically, a temperature of 1° Fahrenheit (Celsius) is not much different than a temperature of 2° Fahrenheit (Celsius). The problem is, of course, that the zero points of the Fahrenheit and Celsius scales are, in a physical sense, arbitrary and ratios of temperatures reflect no underlying property of the temperature attribute.

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 only a finite or countably infinite set of values, e.g., zip codes, counts, or the set of words in a collection of documents. Discrete attributes are often represented as integer variables. Note that *binary attributes* are a special case of discrete attributes and assume only two values, e.g., true/false, yes/no, male/female. Binary attributes are often represented as Boolean variables, or as integer variables that take on the values 0 or 1.

Continuous A continuous attribute is one whose values are real numbers, e.g., temperature, height, or weight. (Practically, real values can only be measured and represented to a finite number of digits.) Continuous attributes are typically represented as floating-point variables.

This is also a good place to mention *asymmetric binary attributes*. These are binary attributes for which only one value is important, typically a value that represents the presence of something, e.g., a word or an item. This type of attribute is particularly important for association analysis, but is also important in other areas of data mining.

2.1.2 Structured Data

The second part of the answer to the question, “What type of data do we have?”, is provided by determining the higher level structure of the data, i.e., the type of structured data that we have. There are many types of structured data, and as data mining develops and matures, more types of data are becoming the object of data mining efforts. In this section we provide a brief overview of some of the common types of structured data, as well as some of the types of data that will be used in examples throughout the book. To lend some organization to this discussion, we have grouped the types of structured data into three groups: record data, graph-based data and ordered data. This grouping is for convenience only.

General Characteristics of Structured Data

While the details of specific kinds of data are provided below, there are some characteristics that cut across a wide variety of types of data and that have a significant impact on the data mining approaches and techniques that are used. We briefly discuss three of these characteristics: dimensionality, sparsity, and resolution.

Dimensionality The *dimensionality* of data is the number of attributes that the objects in a data set possess. Not surprisingly, 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*. Both these issues are discussed in more depth later in this chapter.

Sparsity Some data, especially, but not exclusively, high dimensional data with binary or count attributes, is sparse, i.e., most attributes of an object have values of 0. In practical terms, sparsity is an advantage because often only the nonzero values need to be stored and manipulated, therefore resulting in significant savings with respect to computation time and storage. Indeed, some data mining algorithms are only effective for sparse data. Furthermore, sparsity is also important conceptually since it is often associated with asymmetric binary attributes.

Resolution It is often possible to obtain data at different levels of resolution, and often the properties of the data are different at different resolutions. For instance, surface of the Earth seems very uneven at a resolution of a few meters, but relatively smooth at a resolution of tens of kilometers. Furthermore, the patterns in the data will 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.

Record Data

Much of the original data mining work and much of today's current work is focused around record data, i.e., data that consists of a collection of records (data objects), each of which consists of a fixed set of data fields (attributes). To emphasize, the key characteristics are that there is no explicit relationship between records or data fields and every object has the same set of attributes. Typically record oriented data is stored either in *flat* files or in relational databases. (Relational databases are certainly more than just 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.

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 multi-dimensional space, where each dimension represents a distinct attribute describing the object. Thus, a set of data objects can be interpreted as an m by n matrix, where there are m rows, one for each object, 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, depending on the particular field. 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. the data matrix is the standard format for much statistical data. Figure 2.2b shows a sample data matrix.

<i>Tid</i>	Refund	Marital Status	Taxable Income	Cheat
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

Projection of x Load	Projection of y load	Distance	Load	Thickness
10.23	5.27	15.22	2.7	1.2
12.65	6.25	16.22	2.2	1.1

(b) Data Matrix

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

(c) Document/Term Matrix

<i>TID</i>	<i>Items</i>
1	Bread, Coke, Milk
2	Beer, Bread
3	Beer, Coke, Diaper, Milk
4	Beer, Bread, Diaper, Milk
5	Coke, Diaper, Milk

(d) Transaction Data

Figure 2.2. Different variations of record data.

Document Data For many practical tasks, e.g., search engine queries and document clustering, it is possible to ignore the structure of a document and consider it to be a collection of terms (words). Taking this view, each document becomes a ‘term’ vector, where each term is a component (attribute) of the vector, and where the value of each component of the vector is the number of times the corresponding term occurs in the document. More formally, documents are represented using the *vector-space model*, where the i^{th} document, d_i , is represented by the vector, $d_i = (tf_{i1}, \dots, tf_{in})$, where tf_{ij} = the frequency of the j^{th} term in the i^{th} document. The collection of all documents is often called a *document-term matrix*. Figure 2.2c shows a sample document-term matrix.

Transaction or Market Basket Data Transaction data is a special type of record data, where each record (transaction) involves a set of items. For example, consider a grocery store. The set of products purchased by a customer during one shopping trip constitute 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.’ This data is actually a collection of sets of items, but in practice such data is stored as a set of records whose fields are either binary or count attributes. When transaction data is analyzed, particularly for association analysis, these variables are treated as asymmetric binary attributes. This seems quite reasonable, since store owners are interested in what people bought, not what they didn’t buy. Figure 2.2d shows some sample transaction data. Each row in the table represents the purchases of a particular customer at a particular time.

Graph-Based Data

Often, the relationships among objects convey important information. In such cases, a graph is a convenient and powerful representation. Specifically, the data objects are mapped to nodes of the graph, while the relationships between objects are captured by the links between objects and the link properties, i.e., direction and weight. (See Figure 2.3a.) Alternatively, if the objects have structure, i.e., the objects have sub-objects that have relationships, then sometimes the objects themselves are graphs and the goal is to compare the structures of these complicated objects. Different types of graph data are described below and are shown in Figure 2.3.

Web Data Web pages on the World Wide Web 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. However, it is now well known that the links to and from each page provide a great deal of information about the relevance of a particular web page to a query, and must also be taken into consideration. Figure 2.3b shows a portion of an HTML file that defines a web page with links to other web pages.

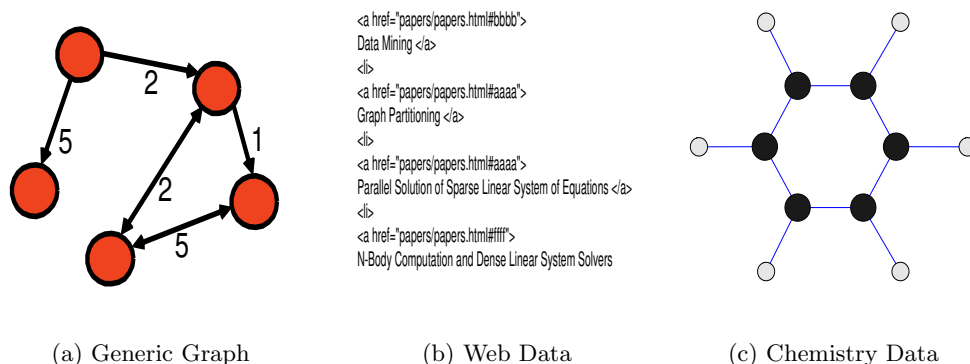


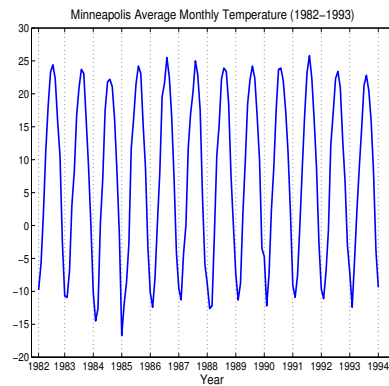
Figure 2.3. Different variations of graph data.

Chemistry Data Over the years, chemists have accumulated a great deal of data concerning the structure and properties of chemical compounds. While chemical compounds can be described to some extent by their properties, e.g., melting point and heat of formation, their structural information is best represented by a graph, where the nodes are atoms and the links between nodes are chemical bonds. Figure 2.3d shows a ball and stick diagram of the chemical compound, benzene, containing carbon (black) and hydrogen (gray).

While it may seem that chemical data is completely different than the other types of data described so far, there are similarities that are not apparent. For instance, an important way to compare two chemical compounds is to look at the substructures (subgraphs) that they share. Furthermore, it is interesting to see which substructures occur frequently in a set of compounds and whether the presence of any of these substructures is associated with the presence or absence of certain chemical properties. Thus, for a given set of compounds and their chemical substructures, the information about which compound contains which substructures can be represented as transaction data, where the transactions are compounds and the items are the substructures.

Ordered Data

For graph-based data, the important relationships are among the objects. However, for other types of data, it is the attributes that are strongly related. In many such cases, these relationships among attributes involve order in time or space. In the simplest situation, an object might consist of attributes, each of which represents a sequence of measurements on the same quantity in time or space. In more complicated cases, we might have a data set constructed by measuring the attributes of a set of objects at various times, i.e., the data set contains different versions of an object. Different types of ordered data are described below and are shown in Figure 2.4.



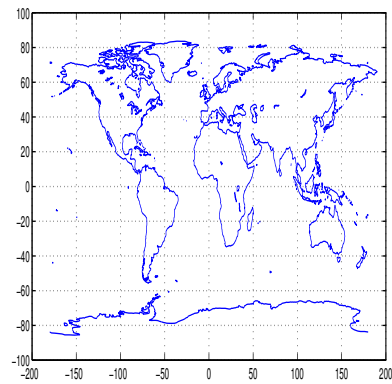
(a) Time Series

(A B) (D) (C E)
 (B D) (C) (E)
 (C D) (B) (A E)

(b) Sequential Data

GGTTCGCGCTTCAGCCCCGCGCC
 CGCAGGGCCCGCCCCGCGCCGTC
 GAGAAGGGCCCGCCTGGCGGGCG
 GGGGGAGGCGGGGCCGCCGAGC
 CCAACCGAGTCCGACCAGGTGCC
 CCCTCTGCTCGGCCTAGACCTGA
 GCTCATTAGGCGGCAGCGGACAG
 GCCAAGTAGAACACGCGAAGCGC
 TGGGCTGCCTGCTGCGACCAGGG

(c) Human Genome



(d) Spatial Data

Figure 2.4. Different variations of ordered data.

Temporal data Temporal data is data whose objects have attributes that represent measurements taken over time. For example, consider a financial data set whose objects are *time series* that give the daily prices of various stocks. (A time series is a sequence of measurements of some attribute, e.g., stock price or rainfall, taken at (usually regular) points in time.) As another example, in climate or weather data, points (or regions) on the surface of the Earth are associated with multiple time series that measure how temperature, pressure, etc. are changing at that point (region). 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. Figure 2.4a shows an example time series of the average monthly temperature for Minneapolis from the years 1982 to 1994.

Sequential Data Sequential data can be thought of as an extension of transaction data. The data still consists of a set of transactions and items, but *time* and *customer ID* attributes are associated with each transaction. In this way, a temporal dimension is added to the original data and it is possible to find patterns that involve time, e.g., people who buy DVD players tend to buy a lot of DVDs in the month immediately following the purchase. Many types of data besides retail data also fit this model, e.g., the transactions could be geographical regions and the items could be events such as droughts, excessive rainfall, or fires. Figure 2.4b shows an example of transaction data. Each of the three rows corresponds to a different object (shopper), and within each row, there are three transactions consisting of the same subset of five items, A, B, C, D, and E, and corresponding to different times.

Genetic Sequence Data Recently, biologists have accumulated large amounts of data about the genetic structure of a variety of plants and animals. This genetic information is in the form of sequences of amino acids and many of the problems associated with this data involve trying to find ways to predict similarities in structure and function from similarities between genetic sequences. Figure 2.4c shows a section of the human genetic code expressed using the four nucleotides from which all DNA is constructed, A, T, G and C.

Spatial data Spatial data is data whose objects have spatial attributes, e.g., positions or areas, either exclusively, or in addition to other sorts of attributes. Examples of spatial data are the locations of schools and weather data, e.g., measurements of precipitation, temperature, or pressure which is collected for a variety of geographical locations. To illustrate the distinctive properties of spatial data, we mention *spatial autocorrelation*, the fact that two objects that are physically close tend to be similar in other ways as well, e.g., two points on the Earth that are close to each other usually have similar temperatures and rainfall.

A very important example of spatial data are the many data sets in science or

engineering which are the result of measurements that are taken at regularly or irregularly distributed points on a two or three-dimensional grid or mesh. For example, Earth science data sets that 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 2.4d.) As another example, in the simulation of the flow of a gas, the speed and direction of flow can be recorded at each grid point in the simulation.

Data from Models

Not all data sets that are encountered arise directly from measurement. Instead, some or all of the data might be the result of a mathematical or simulation model. For example, if a physical structure is being simulated, then the data may be mechanical stress associated with each grid point in the simulation. As another example, a set of measured data is sometimes fitted to a model to remove some of the extraneous variation that results from the measurement process by enforcing well-known physical constraints between adjacent points. For example, the values of air pressure at various points on the Earth should satisfy certain physical models developed by Earth scientists. Finally, the quantity of interest might be the result of a model that has both real and modeled inputs, e.g., an economic model that predicts the Gross National Product (GNP) of a country.

2.2 Data Quality

The question, “What is the quality of the data?”, leads directly to the following three questions:

- What kinds of data problems are possible, i.e., what sorts of situations correspond to poor data quality?
- How can we detect problems with the data?
- What can we do about these problems?

With respect to the first question, the following is a (non-exhaustive) list of some well-known data problems:

- errors, noise and outliers
- missing values
- duplicate data
- inconsistent values

Of course, once we know what sort of problems might occur and these problems have been detected, the third question, “What can we do about these problems?”, needs to be answered. One approach is to ignore these problems. This may seem

naive, but this is the most common approach. Furthermore, depending on the number and types of problems and the data analysis technique being used, the results may not be significantly impacted by the presence of quality problems. Indeed, it is unrealistic to expect most data sets, especially large ones, to be error free. Often, while the owner or collector of the data may know that there are errors in the data, they may not have any way of detecting the errors, and the data may be ‘best available.’ The message for a data miner is ‘expect errors in data.’ Consequently, *error tolerance* is an important criterion for the evaluation of data mining algorithms.

The following four sections provide an overview of the problem areas of data quality that were listed above and the standard approaches used to deal with these problems. In every area, there is a tension between the desire to keep as much information as possible — more information typically translates into better data mining results — and the desire to eliminate data quality problems by eliminating the ‘problem’ data. Indeed, other than ignoring the issue of data quality altogether, eliminating problem data is often the simplest approach to dealing with data quality issues, provided the amount of problem data is relatively small with respect to the total amount of data.

2.2.1 Errors, Noise and Outliers

In a general sense, the term ‘error’ could encompass most data problems. However, here we will use it to cover the host of more specific situations not covered elsewhere by the other, more general types of data problems. An important aspect of these more specific types of errors is that, within particular domains, there are certain types of data errors that are commonplace, and sometimes there exist well-developed techniques for detecting and/or correcting these errors. For example, keyboard errors are common, and as a result, word processing programs have spell checkers that do a reasonable job of detecting and, with human intervention, correcting these errors.

Noise is very generic type of error and the term is often loosely used. In its most precise sense, it refers to a modification of the original attribute values, e.g., the distortion of a person’s voice when talking on a poor phone connection or ‘snow’ on a television screen, or to the presence of data points that are scattered among regular data points, but are not of interest. To illustrate, consider Figure 2.5, which shows a time series, before and after it has been disrupted by random noise, and Figure 2.6, which shows a set of data points before and after some noise points have been added. For this second example, notice how the noise points are intermixed with the non-noise points.

Noise may be random, i.e., generated by some process that is random in nature, or it may be the result of a more deterministic phenomena, e.g., a streak in the same place on a set of photographs. In the latter case, the data error is more commonly called an *artifact*. Furthermore, the term ‘noise’ is often used in connection with data that has a spatial or temporal component. In such cases, techniques from

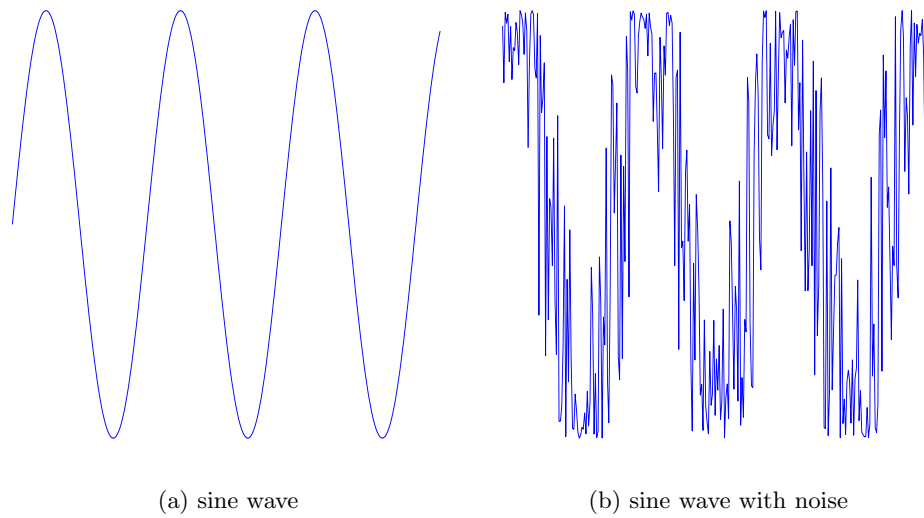


Figure 2.5. Noise in a time series context.

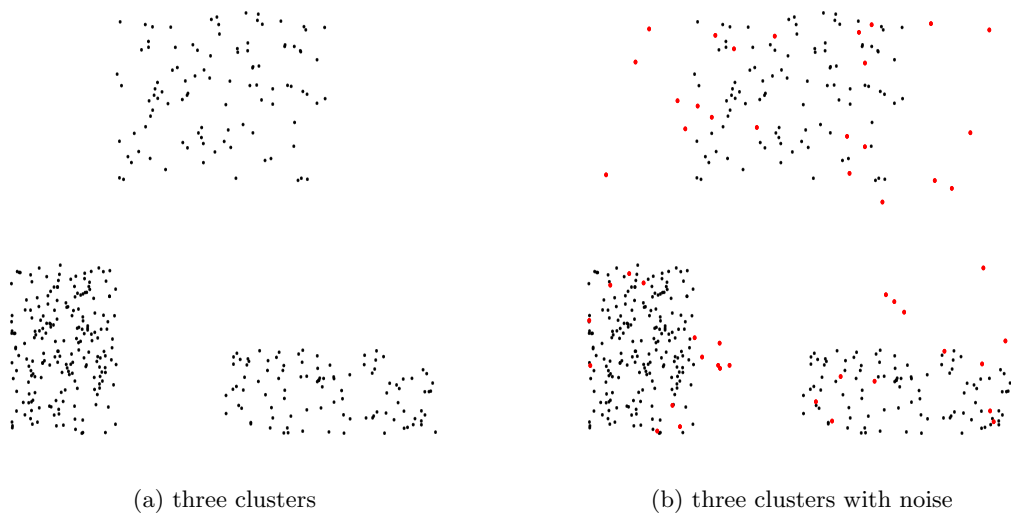


Figure 2.6. Noise in a spatial context.

signal or image processing can sometimes be used to reduce noise and thus, help to discover patterns (signals) that might be ‘lost in the noise.’ However, since the elimination of noise is frequently difficult, much work in data mining focuses on algorithms that work even when noise is present.

Outliers are data objects that, in some sense, have characteristics that are ‘different’ from ‘most’ of the other points in the dataset. Thus, 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. Most commonly, ‘outliers’ refers to a relatively small number of points that are distinct from the main mass of points. Also, unlike noise, outliers may sometimes be of interest, e.g., fraud detection and network intrusion detection consist mostly of finding unusual instances among a very large number of normal instances. Chapter 7 describes outlier detection in more detail.

2.2.2 Missing Values

It is not unusual for an object to be missing one or more attribute values. In some cases, the information was simply 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 only need to be filled out when a person answers a previous question in a certain way, but for simplicity, all fields are stored. In either case, the absence of a value is often indicated by the use of a special value that would not normally occur. Regardless, missing values need to 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 below, together with an indication of their advantages and disadvantages.

Eliminate Data Objects A simple and effective strategy is to eliminate those records with missing values. A related strategy is to eliminate attributes which have missing values. However, even a partially specified data object contains some information, and if many records have missing values, then a reliable analysis may become difficult or impossible. In addition, if the data objects have many ‘optional’ parts, then such an approach obviously will not work. Nonetheless, if a data set has only a few objects or attributes that have missing values, then it may be expedient to omit them.

Estimate Missing Values Sometimes the data set is such that 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, a nearest neighbor approach can be

used to estimate the missing value. More specifically, the attribute values of the points closest to the point with the missing value are used to estimate the missing value. If the attribute is continuous, then the average attribute value of the nearest neighbors is used, while if the attribute is categorical, then the most commonly occurring attribute value can be taken. For a concrete illustration of estimating missing values using nearest neighbors, consider precipitation measurements that are recorded by ground stations. In relatively unpopulated areas, where there are only a small number of ground stations, the precipitation for areas that do not contain a ground station is often obtained by estimation procedures that use the values observed at ground stations.

Ignore the Missing Value During Analysis Many data mining approaches can be modified to operate by ignoring 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 non-missing attributes. It is true that the similarity will only be approximate, but unless the number of attributes is small and/or the number of missing values is high, this degree of inaccuracy may not matter much. Likewise, many classification schemes can handle missing values relatively straightforwardly.

2.2.3 Inconsistent Values

Sometimes errors in data can be detected by checking to see if the data is consistent, either with external knowledge, e.g., a person's height should not be negative, or between different pieces of information in the same data set, e.g., the sum of a set of fields should agree with the field that is supposed to be their sum. Furthermore, if there is some redundancy in the data and/or an external source of information, then it may be possible to correct the data. For example, 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.

In many cases, inconsistency can be detected, but it is less clear how to correct the data. For instance, a 'real' object may be represented by two or more different data objects, perhaps because the data comes from multiple sources. Resolving inconsistencies between different versions of objects depends heavily on the particular domain. (This situation should not be confused with the situation where the state of an object at different times is captured by multiple data objects.)

More subtle types of data inconsistency are possible, and we present an example involving time series. Specifically, our example concerns the sea surface temperature (SST) at various points on the ocean. Before satellites, this data was collected by ocean-based measurements, e.g., from ships or buoys, but more recently, satellites are used to gather SST data. However, if a long-term data set is required, then both data sets must be used, and the data for each time series consists of satellite and non-satellite data. However, since the data comes from different sources, the two

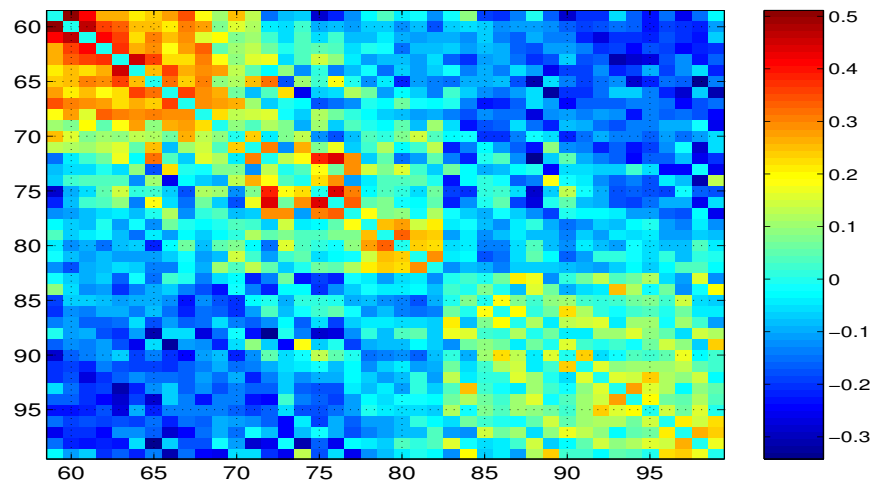


Figure 2.7. Example of inconsistent time series data.

data sets are subtly different. Figure 2.7, shows the pairwise correlation (similarity) between different years, i.e., each row and each column of the figure shows the correlation of the SST patterns for the corresponding pair of years displayed as colors, blue (low) and red (high). There is clearly a change in similarity where the data has been put together in 1983. However, 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 results.

2.2.4 Duplicate Data

Sometimes, a data set includes data objects that are duplicates, or almost duplicates of one another. For example, many people receive duplicate mailings because they are entered into a database with slightly different names. In this case, the duplication is an error and increases a company's cost of doing business. Consequently, considerable research has been expended towards finding and eliminating such duplicates. However, if there are two objects that actually represent a single object, then there is also the possibility that the values of corresponding attributes may differ, and thus, inconsistent values must also be resolved if duplicate data objects are to be combined into one object. Note, however, that care needs to be taken to avoid accidentally combining data objects that are similar, but not duplicates. The term *data cleaning* is often used to refer to the process of dealing with these issues.

In some situations, two or more objects are identical with respect to the attributes measured by the database, but represent different objects. In this case, duplicates are quite legitimate, but may still cause problems if the possibility of identical objects is not specifically accounted for in the design of the data mining algorithm.

2.2.5 Design of Experiments

From the previous sections, it is clear that dealing with data quality problems can be complex and time consuming, and therefore, if possible, it would be nice to avoid such problems in the first place. Indeed, in many fields, e.g., statistics and experimental sciences, experiments or observations are carefully planned in advance to ensure that the data collected contains the information of interest and is of high quality. (A common class offered in statistics is “Design of Experiments.”) As previously mentioned, data mining typically works with data that has already been gathered and sometimes the patterns sought have little to do with why the data was originally gathered. Nonetheless, two points should be made:

- It is useful to participate in the decision about what information should be gathered. Mistakes made in gathering data often mean that information is irretrievably lost or data quality is compromised.
- Certain basic questions that are asked by those who design experiments are also useful for evaluating what can be done with a given set of data. While it is beyond the scope of this book to discuss experimental design in any detail, these include questions such as “How much data needs to be gathered?” and “What combinations of variables need to be considered?” Thus, for the later question, data on the purchasing behavior of families with children probably will not be very useful in predicting the purchasing behavior of teenagers.

2.2.6 Some final comments on data quality.

While good quality data is helpful, data mining can and has yielded good results on data with quality problems, and conversely, sometimes it is impossible to extract any useful patterns even from good quality data. Furthermore, data quality has a cost, which is often the reason for poor quality data, and the cost of fixing these problems needs to be evaluated in terms of whether it is ‘worth it,’ i.e., if these efforts will make the project cost too much or if the improvements in results will justify the effort expended.

2.3 Data Preprocessing

In this section, we address the final question posed at the beginning of this chapter, “What preprocessing steps can or should we apply to the data to make it more suitable for data mining?” Data preprocessing is a large area and is consists of a number of different strategies and techniques that are interrelated in a fairly complicated way. This makes it difficult to present the important ideas and approaches in a nicely structured way. Rather than impose a rather artificial structure on this material, we will present some of the most important ideas and approaches and try to point out the interrelationships, of which there are many.

In particular, we shall focus on the following topics:

- aggregation
- sampling
- dimensionality reduction
- feature subset selection
- feature creation (feature extraction, new spaces, and feature construction)
- discretization and binarization
- attribute 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 make the data mining analysis better, less expensive or quicker. The details are provided in the following sections.

A quick note on terminology. Some preprocessing techniques use the ‘feature,’ instead of ‘attribute.’ Since this terminology is so common, we also use this terminology in the appropriate sections.

2.3.1 Aggregation

Sometimes ‘less is more’ and this is the case with *aggregation*, which refers to combining two or more attributes (or objects) into a single attribute (or object). For example, consider a data set consisting of transactions (the data objects) recording the daily sales of products in various departments for different days (the data attributes) over the course of a year. One way to aggregate transactions would be to replace all the transactions of a single department by a single ‘departmental’ transaction. This would reduce thousands of transactions to tens of transactions. Similarly, daily sales (attributes), could be aggregated by replacing all the sales for a single month with a ‘monthly’ sales total. This would reduce the number of attributes for the data set to a dozen.

There are several motivations for aggregation. Data reduction, as illustrated in the previous example is one. Smaller data sets require less memory and processing time, and thus, aggregation may allow the use of more expensive data mining algorithms. However, this data reduction can provide an additional benefit, i.e., if the aggregation is done properly — related attributes or objects are combined — then the aggregation can act as a change of scope or scale, providing a high level view of the data instead of a low level view.

Another benefit of aggregation is that 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 the aggregate quantities, such as averages or totals, have less variability or relative variability (on average) than the individual objects being aggregated. For totals, the actual amount of variation is larger

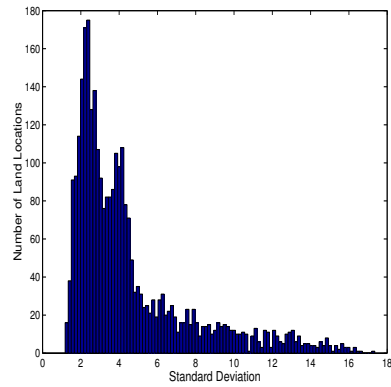
than that of individual objects (on average), but the percentage of the variation is smaller, while for means, the actual amount of variation is less (on average).

These ideas are illustrated in the following example, which concerns precipitation in Australia. In Figure 2.8a we show a histogram for the standard deviation of average monthly precipitation for 3030 $\frac{1}{2}^\circ$ by $\frac{1}{2}^\circ$ grid cells in Australia, while in Figure 2.8b we show a histogram for the standard deviation of the average yearly precipitation for the same locations. Clearly the average yearly precipitation is less variable than the average monthly precipitation. (Notice that the same scale is used on both graphs to allow for easy comparison, but this makes it look as though (b) has fewer points than (a), which is not the case.) Likewise, Figure 2.8c and 2.8d show, respectively, the histograms for the coefficient of variation for the average monthly precipitation and the yearly total precipitation for the same locations. (The coefficient of variation is the ratio of the standard deviation to the mean and measures the degree to which a quantity varies from its average level.) It is clear that the relative variation of the total yearly precipitation is much less than the relative variation of the average monthly precipitation. Although, not shown, aggregating precipitation spatially, i.e., considering the averages and sums of a collection of grid locations, also results in less variable measures of precipitation.

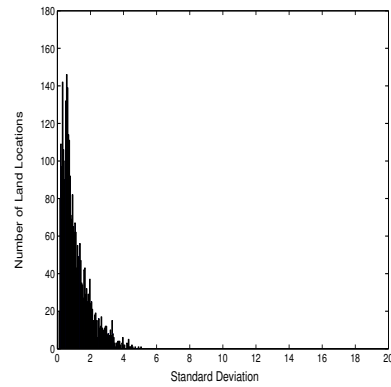
2.3.2 Sampling

While commonsense and domain knowledge can be useful approaches to data selection, typically sampling is the main technique employed; it is often used for both the preliminary investigation of the data and the final data analysis. Sampling has long been used to good effect in statistics and other areas, and can be very useful in data mining as well. (However, the motivation for sampling in statistics and data mining is quite different, i.e., statisticians sample because obtaining the entire set of data of interest is too expensive or time consuming, while sampling is used in data mining because it is too expensive or time consuming to process all the data.) Indeed, 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.

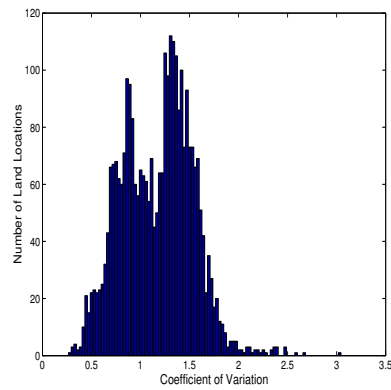
The key principle for effective sampling is the following: using a sample will work almost as well as using the entire data sets, if the sample is ‘representative.’ In turn, a sample is representative if it has approximately the same property (of interest) as the original set of data. Thus, if the standard deviation is the property of interest, then a sample is representative if the sample has a standard deviation that is close to that of the original data. Since sampling is a statistical process, the ‘representativeness’ of any particular sample will vary, and thus, the best that can be said of a particular sampling scheme is that, on average, it produces a representative sample with a high probability. Therefore, a sampling scheme needs to be chosen that will guarantee a high probability of getting a representative sample. This involves choosing the appropriate sample size and sampling techniques, as discussed below.



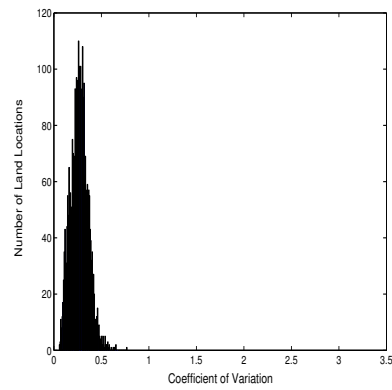
(a) Standard Deviation of Average Monthly Precipitation



(b) Standard Deviation of Average Yearly Precipitation



(c) Coefficient of Variation for Average Monthly Precipitation



(d) Coefficient of Variation for Total Yearly Precipitation

Figure 2.8. Standard Deviation and Coefficient of Variation for Monthly and Yearly Precipitation in Australia.

Sampling Approaches

There are a wide variety of sampling techniques, and only a few of the most basic and their variations, will be covered here. The most basic type of sampling is *simple random sampling*. In 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): sampling without replacement, i.e., as each item is selected it is removed from the set of all objects (the *population*), and sampling with replacement, i.e., objects are not removed from the population as they are selected for the sample. In sampling with replacement, the same object can be picked up more than once. Sampling without replacement is usually preferable, but the samples produced by the two methods are not much different when samples are relatively small compared to the data set size, and sampling with replacement is sometimes simpler.

While simple random samples are often fine, there is a problem in looking for patterns that involve different types of objects when the different types (classes) have widely different numbers of objects. For classification, rare classes may be missed, and as a result, a sampling scheme that can accommodate differing frequencies for the items of interest is needed. Stratified sampling is such an approach. Basically, stratified sampling starts with different groups of object, which must be pre-specified, e.g., by class labels. In the simplest version, equal numbers of objects are then drawn from each group even though the groups are of different sizes.

Many other sampling schemes are possible. Some of the sampling techniques for clustering and association analysis will be discussed later in the book, but for information beyond that, the reader is referred to the references in the bibliographic remarks.

Sample Size

Even if the proper sampling technique is known, it is still necessary to choose the sample size. Larger sample sizes increase the probability that a sample will be representative, but also eliminate much of the advantage of sampling. Conversely, with smaller sample sizes, patterns may be missed or erroneous patterns detected. To illustrate this consider Figure 2.9. Part (a) of this figure shows the full two-dimensional point set, which contains 8000 points, while parts (b) and (c) show, respectively samples of size 2000 and 500. While much of the structure of this point set is present in the sample of 2000 points, much of the structure is missing in the sample of 500 points.

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

For example, suppose that we want to use progressive sampling to learn a pre-

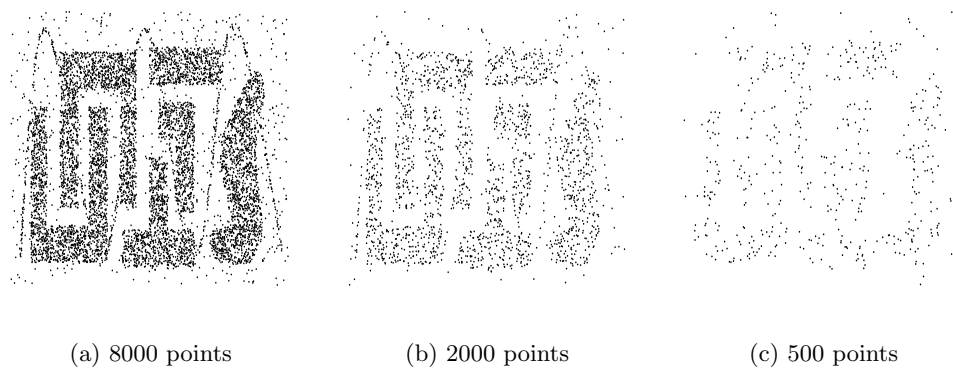


Figure 2.9. Example of the Loss of Structure with Sampling.

dictive model. The graph of prediction accuracy versus sample size is the *learning curve* and it is known that the while accuracy of predictive models increases as the sample size increases, at some point the increase in accuracy, i.e., the learning curve levels off. We want to stop increasing the sample size at this leveling off point. By keeping track of the change in accuracy as we take progressively larger samples, and by taking other samples in the neighborhood of the current one, we can get an estimate as to how close we are to flat part of the learning curve and thus, stop sampling.

Sample Size Example

We conclude this section with an extended example to illustrate that determining the proper sample size requires a methodical approach, not just guesswork.

Given a set of data which is thought to be divided into 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, say around 10. Figure 2.10a shows a possible set of groups of clusters from which these points might be drawn.

On the surface it would seem that we could solve this problem as follows: Take a small sample of data points, compute the pairwise similarities between points, and then form groups of points that are highly similar. But the question is, how big would the sample have to be?

Some people might answer 10, thinking that if a sample is random, it is also uniform, i.e., that one object will be selected from each group. Other, people might hesitate, realizing that there is some chance of getting more than one object from some groups, while other groups are missed. We can imagine someone guessing, “Fifteen or twenty, but certainly thirty. Am I right?”

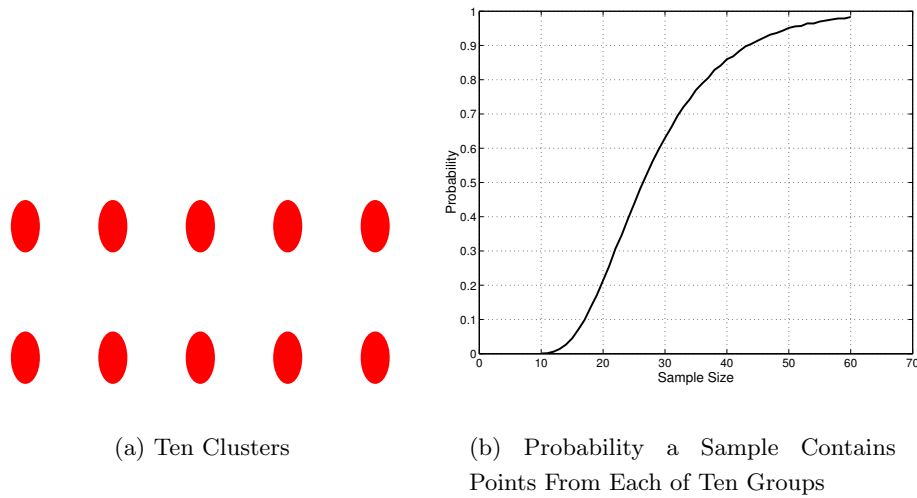


Figure 2.10. Finding representative points from 10 groups.

The answer is ‘only partly.’ Figure 2.10b 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 groups. Even with a sample of size 30, there is still a moderate chance (almost 40%) of getting a sample that doesn’t contain objects from all 10 groups.

2.3.3 Dimensionality Reduction

Many times the data sets of interest have a large numbers of features. For example, consider our previous example of a set of documents, where each document is represented as a vector whose components are the frequencies with which each word occurs. In such cases, there are typically thousands or tens of thousand of attributes (components), one for each word in the vocabulary. As another example, consider a set of time series that record 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.

The relevance of high dimensionality is that many data mining algorithms ‘work better’ if the dimensionality, i.e., the number of attributes in the data, is lower. As mentioned previously, this is due to the ‘curse of dimensionality,” which generally speaking, refers to the phenomenon that many types of data analysis become significantly harder as the dimensionality of the data increases. More specifically, as dimensionality increases, the data becomes increasingly sparse in the space that it occupies. For classification this means that there are not enough sample data points to allow for reliable assignment of a class to all possible values, while for clustering, the definitions of density and the distance between points, which is critical for clustering, become less meaningful. As a result, many clustering and classification algorithms (and statistical algorithms as well) have trouble with high dimensional

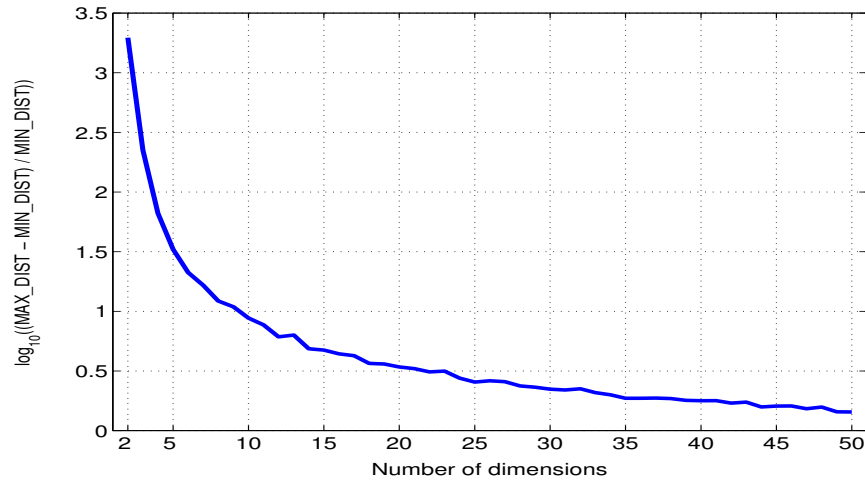


Figure 2.11. Decrease in relative distance between min and max distance with increasing dimensionality.

data, e.g., reduced classification accuracy and poorer quality clusters.

To illustrate the curse of dimensionality more directly, consider, Figure 2.11, which is an illustration of the fact that the relative distance between points decreases as dimensionality increases. To generate this figure, we randomly generated 500 points for each different number of dimensions — 2 to 50 — and for each set of points, computed the pairwise distance between all pairs of points. We then found the minimum and maximum distances, MIN_DIST and MAX_DIST , respectively, and formed the ratio, $\frac{MAX_DIST - MIN_DIST}{MIN_DIST}$, (All attributes (dimensions) were independent and identically distributed as a normal distribution with mean 0 and variance 1 and the plot is the average of 10 runs.) Figure 2.11 show the \log_{10} value of the ratio. When the number of dimensions is low, 2 - 5, the relative difference between the minimum and maximum distance of points is high, but as the dimensionality increases, the ratio drops rapidly. When the number of dimensions is 50, the ratio is 1.4 (the corresponding \log_{10} ratio is 0.16).

There are other benefits to dimensionality reduction besides avoiding the curse of dimensionality. The amount of time and memory required by the data mining algorithm is reduced with a reduction in dimensionality, and sometimes this allows the use of algorithms that would have otherwise been infeasible. Furthermore, a reduction of dimensionality may allow the data to be more easily visualized. (Note that even if a 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 combination is greatly reduced.) Also, if dimensionality reduction eliminates irrelevant features or reduces noise, then the quality of results may improve. Finally, dimensionality reduction can lead to a more understandable model.

Principal Components Analysis (PCA) and Singular Value Decomposition (SVD)

Some of the most common approaches for dimensionality reduction, particularly for continuous data, use a linear or non-linear projection of the data from a high dimensional space to a lower dimensional space. For instance, principal components analysis (PCA) is a linear algebra technique for continuous attributes that finds new attributes (principal components) that a) are a linear combinations of the original attributes, b) are orthogonal ('perpendicular') to each other and c) capture the maximum amount of the variation in the data, e.g., the first two principal components capture as much of the variation in the data as it is possible with orthogonal attributes that are linear combinations of the original attributes. Singular value decomposition (SVD) is another linear algebra technique that is related to PCA and is also commonly used for dimensionality reduction. For additional details, see Appendix 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 for two types of features:

Redundant features duplicate much or all of the information contained in one or more other attributes, e.g., the purchase price of a product and the amount of sales tax paid contain much the same information.

Irrelevant features contain no information that is useful for the data mining task at hand, e.g., students' ID numbers should be irrelevant to the task of predicting students' grade point averages.

Irrelevant and redundant features can reduce prediction accuracy or robustness and lead to poor quality clusters. While some of these attributes may be eliminated (or more precisely not used in the data mining process) immediately by using common sense or domain knowledge, selecting a subset of features frequently requires an automatic 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 to take the subset that produces the 'best' result as the best set of features. This does have the advantage of reflecting the objective and bias of the data mining algorithm that will eventually be used. However, since the number of subsets involving n attributes is 2^n , such an approach is impractical for most data mining situations, and alternative approaches are needed.

In particular, there are three standard approaches to feature selection:

Embedded approaches Feature selection occurs naturally as part of the data mining algorithm, e.g., algorithms for building decision trees select a subset of attributes to use during the process of building the tree.

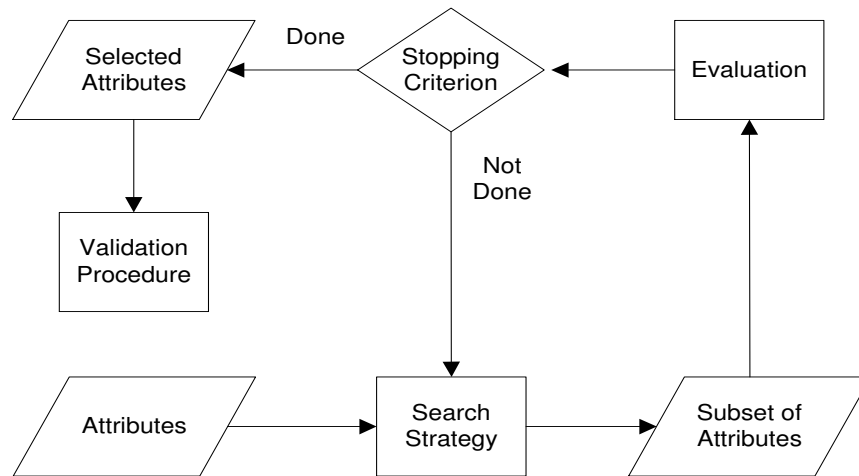


Figure 2.12. Flowchart of a feature subset selection process.

Filter approaches Features are selected before the data mining algorithm is run.

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.

An Architecture for Feature Subset Selection

It is possible to encompass both the filter and wrapper approaches within a common architecture. Specifically, 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 method for evaluating a subset, i.e., for a wrapper method, the subset evaluation method uses the target data mining algorithm, while for a filter approach, a different evaluation approach is used. The following paragraphs provide some details of this approach, which is summarized in Figure 2.12.

Search strategy Two important requirements on the search strategy are that it should be computationally feasible and that it should find optimal or near optimal sets of features. Not surprisingly, it is usually not possible to satisfy both requirements and thus, tradeoffs are necessary. For instance, an exhaustive (or exponential) search will find the optimal solution according to whatever criteria is used, but is only feasible for small feature sets. (However, various backtracking and branch and bound techniques are sometimes be used

to improve the efficiency of exhaustive search.) A sequential search, on the other hand, proceeds from one subset to another without backtracking. In particular, a sequential forward generation procedure starts with an empty set of features and continues to add features based on which feature increases the value of the subset evaluation measure. Likewise, a sequential backward generation procedure starts with all the features and eliminates the ones that lead to the smallest decrease in the subset evaluation measure. Of course, the backward and forward procedure can be combined and multiple features can be added and/or deleted in one step. Finally, a random search generates subsets randomly, and thus, may produce different sets of features from one run to another.

Evaluation The evaluation measure for a subset attempts to evaluate the ‘goodness’ of a subset with respect to a particular data mining task, e.g., classification or clustering. For example, the *consistency measure*, which is used when the task is classification, counts the number of cases where two objects are equal (using only the attributes in the subset under consideration), but are from different classes. Intuitively, subsets where this measure is high are not likely to perform well for classification. Many other measures for classification are possible.

The key point to notice about subset evaluation measures is that they are surrogate, i.e., stand in, measures that attempt to predict how well the actual data mining algorithm will perform on a given set of attributes. It is the performance of the data mining algorithm, e.g., the classification accuracy or cluster cohesion, that is the measure that counts. Thus, while subset evaluation measures reflect different approaches to evaluating the properties a set of attributes should have, the best subset of features as determined by the evaluation measure may very well be different than the *optimal feature subset* according to the target data mining algorithm. Indeed, a key advantage of the wrapper approach is that subset evaluation function matches the criteria used to measure the result of the data mining.

Stopping Strategy The stopping strategy typically involves 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.

Validation When a subset of features has been selected, the results of the target data mining algorithm on the selected subset should be evaluated. For example, a simple sanity check is to run the algorithm with the full set of features and compare the full results to the subset results. Alternatively, different feature selection algorithms can be used, and results can be compared.

This comparison is not always straightforward. For example, if the goal is to build a decision tree and the selected subset of features provides slightly lower accuracy than the full set of features, but produces a more compact and understandable tree, say according to a domain expert, then the smaller subset of features may still be more desirable.

2.3.5 Feature Creation

Sometimes a small number of new attributes can capture the important information in a data set much more efficiently than the original attributes. Also, the number of new attributes can often be much smaller than the number of original attributes, and thus, we reap all the previously described benefits of dimensionality reduction. Three general methodologies for created new attributes are described below.

Feature Extraction

One approach to dimensionality reduction is *feature extraction*, which is the creation of a new, smaller set of features from the original set of features. For example, consider a set of photographs, where each photograph is to be classified according to whether it contains a human face or not. 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, e.g., the presence or absence of certain types of edges and areas correlated with presence of human faces, then a much broader set of classification techniques can be applied to this problem.

Unfortunately, feature extraction, in the sense in which it is most commonly used, is not a very automated procedure. For a particular field, e.g., image processing, various features and the techniques to extract them are developed over a period of time, and often these techniques have limited applicability to other fields. Thus, whenever data mining is being applied to a relatively new area, a key task is the development of new features and feature extraction methods.

Mapping the Data to a New Space

Sometimes, a totally different view of the data can reveal important and interesting features. Consider, for example, time series data. Frequently time series data has a very periodic pattern. If there is only one such pattern and not a lot of noise, then such patterns are easily detected by visualization. However, if there are a number of periodic patterns and noise as well, then these patterns are hard to detect. Nonetheless, such patterns can be often be detected by applying a Fourier transformation to the data, which provides a representation in which frequency information is explicit. For what follows, it will not be necessary to know the details of the Fourier transform. Instead, it is enough to know that, for each time series, the Fourier transform produces a new data object, whose attributes are related to frequencies.

Consider the following example shown in Figure 2.13. The time series presented in (b) is the sum of three other time series, two of which are shown in (a) and which

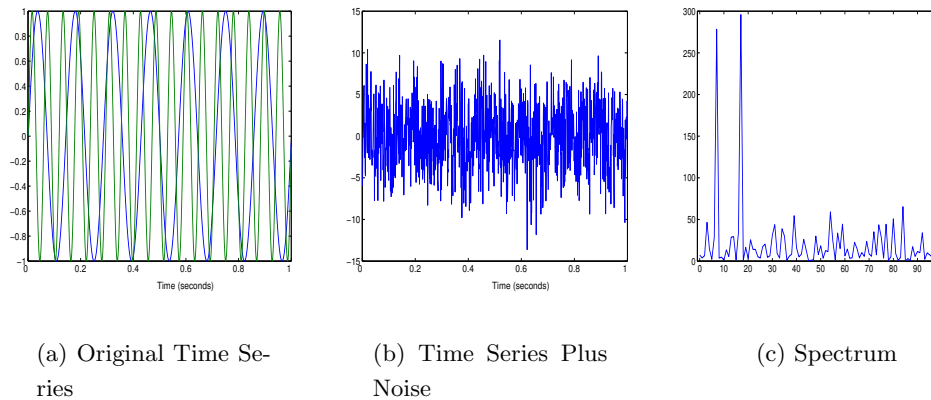


Figure 2.13. Different variations of record data.

have frequencies of 7 and 17 cycles per second. The third time series is random noise. In (c), we see the power spectrum that can be computed after applying a Fourier transform to the original time series. (Informally, this plot 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-noise time series. Thus, better features can more easily reveal important aspects of the data.

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

Statistical models are another way to obtain a different view of data. In particular, a statistical model is proposed for the objects in a set of data, and then the parameters of this model are estimated for each object. These statistical parameters can then be used as attributes of the objects. Once again, the new attributes can be more useful than the old, e.g., a particular type of statistical model, called an ARIMA model can be fit to time series, and the parameters of that model can then be used to evaluate the similarity of time series to each other.

Feature Construction

Sometimes features have the necessary information, but not in the form necessary for the data mining algorithm. In this case, one or more new features constructed out of the original features may be more useful. To illustrate this, consider two attributes that record the volume and mass of a set of objects. Suppose that there exists a classification of a set of training objects that is based on the material of which the objects are constructed (the objects are the same in terms of physical dimensions), then a density feature constructed from the original two features would clearly facilitate an accurate classification. Another example would be the logical combination of Boolean variables. While some research has been conducted with respect to automatic feature construction, the most common and effective approach is to construct features using domain expertise.

Table 2.3. Conversion of a categorical attribute to a 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

2.3.6 Discretization and Binarization

Some data mining algorithms, e.g., classification algorithms, require that the data be in the form of categorical attributes. Some other algorithms, especially algorithms that find association patterns, require the data in the form of binary attributes. Thus, it is often necessary to transform a continuous attribute into a categorical attribute, and both continuous and discrete attributes may need to be transformed into one or more binary attributes. Furthermore, some classification algorithms also see performance degradation even when working with categorical attributes, if the number of categories is too large. In such cases, it may be necessary to combine some of the categories.

Discretization and binarization are similar to feature selection in that the ‘best’ way to discretize or binarize an attribute is ‘whatever produces the best result for the data mining algorithm that will be used to analyze the data.’ And, as with feature selection, it is typically not practical to apply such a criterion. Consequently, discretization or binarization is done in 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 straightforward way to binarize a categorical attribute is as follows: If there are n categorical values, then uniquely assign each original value to an integer in the interval $[0, n - 1]$. 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 $[0, n - 1]$.) Next, convert each of these n integers to binary. Finally, noting that $m = \lceil \log_2(n) \rceil$ binary digits will be required to represent these integers, represent these binary numbers using m binary attributes. Thus, a categorical variable with 5 values $\{awful, poor, OK, good, great\}$, would require 3 binary variables x_1 , x_2 , and x_3 . The conversion is shown in Table 2.3.

However, such a transformation can cause complications. For example, it can create unintended relationships among the transformed attributes. Furthermore, in association analysis, the binary attributes are actually asymmetric binary attributes, i.e., only the ‘presence’ of the attribute (value = 1) is important. Thus, for association problems, it is necessary to introduce one binary attribute for each

categorical value. 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, it may be necessary to replace a single binary attribute with two asymmetric binary attributes for association problems. For instance, consider a binary attribute that records a person's gender. For traditional association rule algorithms to properly use this information, it would need 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 horribly inefficient, i.e., 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 problems. In general, the 'best' discretization depends on the classification algorithm being used, as well as the other attributes being considered. However, this context is typically ignored, and the discretization of an attribute is often considered in isolation.

Transformation of a continuous attribute to a categorical attribute involves two subtasks: deciding how many categories to have and deciding how to map the values of the continuous attribute to the categorical attribute. In the first step, after the values of the continuous attribute are sorted, they are then divided into m intervals by specifying $m - 1$ '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 these split points. The result can be represented either as a set of intervals $\{(x_0, x_1], (x_1, x_2], \dots, (x_{m-1}, x_m)\}$, where x_0 and x_m may, respectively be $+\infty$ or $-\infty$, or equivalently, as a series of inequalities $x_0 < x \leq x_1, \dots, x_{m-1} < x < x_m$.

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 example, the *equal interval width* approach, divides the range of the attribute into a user specified number of intervals. Such an approach can be badly affected by outliers, but an *equal frequency* approach, which tries to put the same number of objects into each interval, usually works better for such situations. As another example of unsupervised discretization, a clustering method, such as K-means (see chapter 5), can also be used. Finally, 'eyeballing' the data can sometimes be an effective technique for discretization.

We present an example to demonstrate how these approaches might work on a data set. In Figure 2.14a, we show data points belonging to four different groups, along with two outliers — the large dots on either end. We applied the techniques of the previous paragraph to discretize the x values of these data points into four categorical values. (The data set is two dimensional mainly to make it easy to see how many points are in each group. The y values were assigned randomly.)

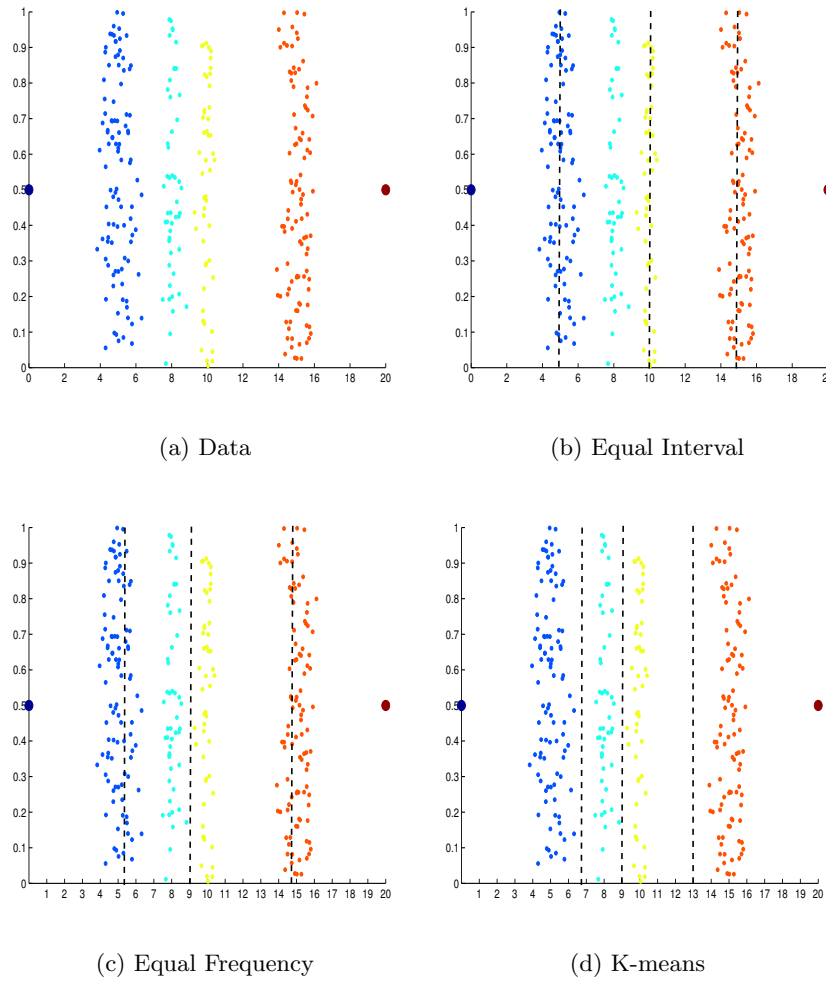


Figure 2.14. Different discretization techniques.

Eyeballing works quite well, but is not automatic, and we focus on the other three approaches. The split points produced by the techniques, equal interval width, equal frequency, and K-means, are shown in subfigures (b), (c), and (d), respectively. (The split points are represented as lines.) K-means performs best, followed by equal frequency, and finally, by equal interval width.

The discretization methods described above are usually better than no discretization, but as with feature selection, keeping the end purpose in mind and using the class label information often produces better results. This should not be surprising, since an interval constructed with no knowledge of class labels may contain a mixture of class labels. A conceptually simple approach is to try to place the splits so as to maximize 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 statisti-

cally 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. However, entropy based approaches are perhaps one of the most promising approaches to discretization, and a simple approach based on entropy will be presented as an example.

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

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

where $p_{ij} = n_{ij}/n_i$ 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.,

$$e = \sum_{i=1}^m f_i e_i$$

where $f_i = n_i/n$ is the fraction of points in the i^{th} interval. Intuitively, the entropy of an interval is a measure of the purity of an interval. If an interval contains only one class, i.e., is perfectly pure, then the entropy is 0 and it contributes nothing to the overall entropy. If the classes in an interval occur equally often, i.e., 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 requires considering each value as a possible split point, since 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 until a stopping criterion is satisfied.

To demonstrate this method, we used it to independently discretize both continuous attributes of the two-dimensional data shown in 2.15. In the first discretization, shown in Figure 2.15a, both the x and y attributes were split into three intervals. (The dashed lines indicate the split points.) In the second discretization, shown in Figure 2.15b, both the x and y attributes were split into five intervals.

This simple example illustrates a couple of issues. First, in two dimensions, the classes of points are well separated, but in one dimension, this is not so. Thus, in general, discretizing each attribute separately often guarantees suboptimal results. Secondly, 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 tries to automatically find the right number of partitions. Since these issues are very similar to those involved in finding good classification models, further discussion is deferred to Chapter 3.

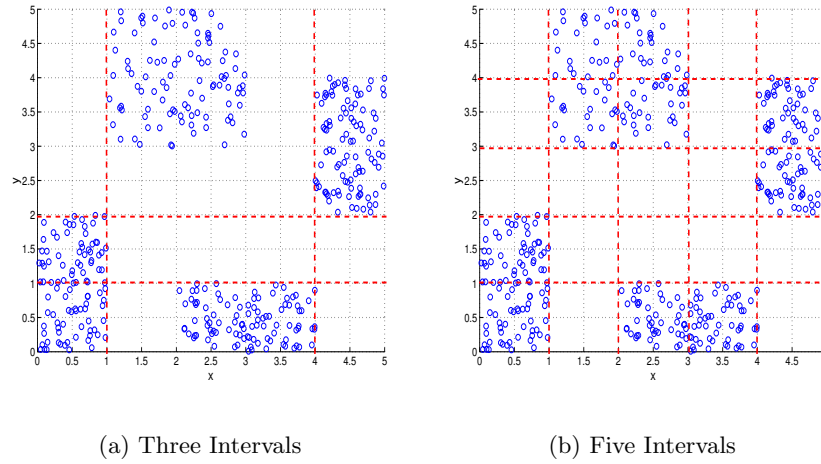


Figure 2.15. Discretizing x and y attributes for four groups (classes) of points.

Categorical Attributes With Too Many Values

As mentioned, 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. If the categorical attribute is nominal, however, then other approaches are needed. For example, a university often has a large number of departments, and consequently, a *department name* attribute might have dozens of different values. In this situation, by using domain knowledge, i.e., knowledge about the relationships among different departments, these departments could be combined into larger groups, e.g., all engineering departments could be grouped as ‘engineering.’ 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 together, only if such a grouping is beneficial for classification accuracy.

2.3.7 Attribute Transformation

An *attribute transformation* refers to a transformation that is applied to all the values of an attribute, i.e., for each object, the transformation is applied to the value of the attribute for that object. (This is more commonly known as a *variable transformation*.) For example, if only the magnitude of an attribute is important, then the values of the attribute may be transformed by taking the absolute value. More generally, an attribute transformation can be viewed as a function that maps the entire set of values of a given attribute to a new set of replacement values such that each old value can be identified with one of the new values. In the following, we discuss two important types of attribute transformations: simple functional transformations and standardization (normalization).

Simple Functions

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

For example, suppose that the attribute of interest is the number of data bytes in a session, and that the number of bytes ranges from 1 to 1 billion. This is a huge range, and it may well be advantageous to compress the range 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, e.g., network intrusion detection, this may be what is desired, since the first two sessions likely represent file transfers, while the latter two sessions may well be two quite different types of sessions.

However, three important cautions are in order for these types of transformations, especially the transformations *sqrt*, *log*, and $1/x$. First, a transformation typically operates differently on different attributes values, and the effects can sometimes be surprising. For instance, the transformation $1/x$ will reverse the order of the values, e.g., $\{1, 2, 3\}$ goes to $\{1, \frac{1}{2}, \frac{1}{3}\}$. If the goal was range compression, but not a reversal of order, then a multiplication by -1 will also need to be applied to achieve the desired effect. Furthermore, range compression will fail if some values are between 0 and 1, since these values will become large after they are transformed. Important questions to ask before applying a transformation are 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? Sometimes it is desirable to transform values to the range $[1, \infty]$ by adding a constant, at least if the values are mostly in the positive range.

The second caution is that transformations change the nature of the attribute scale. Thus, if a difference of values was meaningful before the transformation, then a difference of values may not be meaningful afterwards, and vice-versa. The following example illustrates the complexities of this issue. Consider the kinetic energy of an object measured at various velocities and recall that the formula for kinetic energy is $KE = \frac{1}{2}MV^2$, where M and V are the mass and velocity of the object, respectively. Differences in the original scale, the energy scale, make sense — they are the differences in the kinetic energy possessed by the object. If the transformation \sqrt{KE} is applied to the data, then the resulting values measure the velocity of the body, and the differences are again meaningful. However, if a $\sqrt[3]{KE}$ transformation is applied, then, while the order of the values is still meaningful, the differences and ratios are not. A ratio attribute has been transformed into an ordinal attribute.

The final caution regards the interpretability of the results. It can be much harder to interpret the meaning of a variable after it has been transformed. In this

way, the situation is similar to the problems that arise when PCA or SVD are used. An exception to this is the following: If the results of the data mining involve values on the transformed scale, then the results can often be expressed on the original scale. For instance, if the results with respect to the transformed attribute involve an interval $[y_1, y_2]$, and the values of the original attribute that correspond to y_1 and y_2 are, respectively, x_1 and x_2 , then the corresponding interval for the original attribute values is $[x_1, x_2]$, provided that the attribute transformation is monotonic.

Standardization or Normalization

Another common type of attribute transformation is the *standardization* or *normalization* of an attribute. (Statisticians use the term ‘standardization’ because ‘normalization’ might be confused with the transformations used for making a variable ‘normal,’ i.e., Gaussian. In common usage, however, the terms are often used interchangeably.) The goal of standardization or normalization is to make an entire set of values have a particular property, and consequently, the transformation is determined, to some extent, by the actual values of the attribute.

For example, if the values of an attribute are measures of similarity between objects, but range from 1 (not at all similar) to 10 (completely similar), it might be desired to make the values fall within the range $[0, 1]$, which is more traditional for similarities. In this specific case, the transformation is given by $s' = (s - 1)/10$, where s is the original similarity value and s' is the new similarity value. In the more general case, the transformation of similarities to the interval $[0, 1]$ is given by the expression $s' = (s - \min)/(max - \min)$, where max and min are the minimum and maximum similarity values.

As another example, consider transforming a distance, e.g., the traditional Euclidean distance that is defined in the next paragraph, into a similarity value. Since two objects are ‘close’ if their distance is low or their similarity is high, the order of distances needs to be reversed. The easiest transformation is multiplication of distances by -1 , and this works well in many cases. To illustrate, the distances 0, 1, 10, and 100 would be transformed into 0, -1 , -10 , and -100 . However, the resulting similarities are not restricted to the range $[0, 1]$, and if that is desired, then the transformations $\frac{1}{x+1}$ or $e^{-distance^2}$ can be used. For the $\frac{1}{x+1}$ transformation, the distances 0, 1, 10, and 100 are transformed, respectively, into 1, 0.5, 0.0909, and 0.0099, while for the $e^{-distance^2}$ transformation, distances 0, 1, 10, and 100 are transformed, respectively, into 1, 0.3679, 0, and 0 (to four decimal places). In general, any monotonic decreasing function can be used to convert distances to similarities, although the exact function will have a strong effect on the behavior of data mining algorithms, e.g., clustering algorithms, that use similarities.

A more traditional example is that of ‘standardizing a variable’ in statistics. If μ is the mean (average) of the attribute values and σ is their standard deviation, then the transformation $v' = (v - \mu)/\sigma$ creates a new attribute that has a mean of 0 and a standard deviation of 1. If different attributes will be combined in some way, then such a transformation is often necessary to avoid having an attribute

with large values dominate the results of the calculation. For instance, the Euclidean distance between two objects, x and y , with numerical attributes, is given by $\sqrt{\sum_{i=1}^n (x_i - y_i)^2}$, where x_i and y_i are the i^{th} attributes of x and y , respectively. Consider trying to measure the similarity between people based on two attributes, age and income. Without standardization, the similarity between two people will be dominated by income.

Because the mean and standard deviation are strongly affected by *outliers*, the above transformation is often modified. First, the mean is replaced by the *median*, i.e., the middle value. Secondly, the standard deviation is replaced by the absolute standard deviation, i.e., $\sigma_A = \sum_{i=1}^n |v_i - \mu|$, where v_i is the i^{th} value of the attribute, n is the number of objects, and μ is either the mean or median.

As a final illustration of normalization transformations, we present an example involving document data. Here the problem is that similarities between documents can be dominated by common words, which appear in many documents and inflate the similarities between documents. While common words (called ‘stop’ words), such as ‘the’ and ‘that,’ are normally eliminated automatically, the problem can still remain. For example, if the documents are about sports, ‘team’ may appear in almost every document. To deal with this problem, a transformation, called ‘inverse document frequency’ can be applied.

This works as follows: For the each word, we count the number of documents in which the word appears. This is the word’s *document frequency*, and for the i^{th} word, we represent this by df_i . Recall that tf_{ij} is the frequency of the i^{th} word (term) in the j^{th} document. If n is the number of documents, then a new value for tf_{ij} is given by $tf'_{ij} = tf_{ij} * \log \frac{n}{df_i}$. If a word appears in all documents, i.e., has no usefulness for distinguishing one document from another, then its normalized value is 0.

2.4 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? Explain.
2. Classify the following attributes as binary, discrete, or continuous. Further classify the attributes as qualitative (nominal or ordinal) or quantitative (interval or ratio). Some of the 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. (Format of ISBN numbers is at <http://www.isbn.spk-berlin.de/html/userman/usm4.htm>)
 - (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. (Sometimes when you attend an event, you can give your coat to someone and they will give you a number that you can use to claim your coat when you leave.)
3. You are approached by the marketing director for a local company, who believes that he has devised a fool-proof 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, i.e., 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. Can you think of a situation in which identification numbers would be useful for prediction?
5. 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 ready for association analysis?
 - (b) In particular, what type of attributes would you have and how many of them are there?
6. Which of the following quantities is likely to show more temporal autocorrelation: daily rainfall or daily temperature? Why?

2.5 Bibliographic Notes

As stressed in the chapter, it is important to have an idea of the nature of the data that is being analyzed. At a very fundamental level, this is what measurement theory is all about. Typically students have little exposure to measurement theory, perhaps because the aspects that are important most situations can be summarized in terms of the attribute types described in this chapter. However, the situation is more complicated than we have described. In particular, log-interval scale and absolute scales are possible, in addition to the four scales we described in this chapter. For those who would like to investigate further, we recommend the following three sources: [106], [176], and [196].

Sampling is a subject that has been well studied in statistics and in related fields. In particular, many introductory statistics books [114] have some discussion on sampling, and there are entire books devoted to the subject [36]. A survey of sampling for data mining is provided in [64], while a survey of sampling for data bases is provided in [138]. Other data mining and data base related sampling references that may be of interest are [148, 141, 183, 206].

While data quality is a broad subject that spans every discipline that uses data, most of the easy accessible and generic information on data quality has been published in the database area [42, 195, 128]. However, often the knowledge to deal with specific data quality issues in a specific domain is best obtained by investigating what steps researchers in that field take to ensure data quality. For example, reading a general book on data quality may not give you much information on the steps that NASA takes to ensure that two satellite images are comparable even though they are taken at different times, under different cloud conditions, etc. I

In statistics, the traditional techniques that have been used for dimensionality reduction are Principal Components Analysis (PCA) [97], which is similar to the important linear algebra technique of Singular Value Decomposition (SVD) [43], and Multidimensional Scaling [107]. These techniques are based on the observation that the data can often be represented by a linear approximation in relatively small set of dimensions. More recently researchers have also investigated the usefulness of nonlinear transformations for dimensionality reduction [152, 181].

Discretization is a topic that has been well investigated in data mining. In particular, some classification and association rule algorithms will only work on categorical data, and thus, there is a significant motivation to investigate how to best categorize continuous variables. For association rules we refer the reader to [171], while some useful references for discretization in the area of classification include [56, 48, 89, 52].

Feature selection is another topic well investigated in data mining. A survey of this area can be found in [130], while some useful references are [104, 20, 120]. A couple of books on the subject are [119, 118].

The subject of variable transformations is also a difficult topic for which to provide general references, as 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, e.g., ensuring the normality of a variable or that variables have equal variance. Thus, we offer only the following two references [139, 188].

With some of the other topics in this chapter, it is much more difficult to give specific references. For instance, to find out about the different types of data in more detail, it is necessary to look at either domain specific literature of data mining literature that has addresses the data in question. Finally, aggregation is another topic that is very domain specific and must be handled on a case by case basis.

Classification

Classification is the task of assigning objects to their respective categories. Classification is useful because it helps us to understand the similarities and differences between objects that belong to different categories. For example, stock analysts are interested in classifying the stocks of publicly-owned companies as **buy**, **hold**, or **sell**, based on the financial outlook of these companies. Stocks classified as **buy** are expected to have stronger future revenue growth compared to those classified as **sell**.

Classification is also an integral part of the decision making process. Suppose a mobile phone company would like to promote a new cell-phone product to its existing customer base. Instead of mailing the promotional catalog to everyone, the company may want to reduce mailing costs by targeting only a small subset of the population. To do this, the company needs to know who are the people expected to buy the new product and distinguishes them from those who will not buy. After classifying their customers, the company can then take the appropriate action by mailing out the catalogs only to those persons classified as **buy**.

But how does the company decide whether a person should be classified as **buy** or **won't buy**? Each person is usually classified on the basis of their personal information such as household income, occupation, lifestyle, credit ratings, and where they live. Classifying the people manually is a challenging task due to several reasons. First, there are many driving factors that can affect the classification decision. It will be very difficult for analysts to take all these factors into consideration when classifying a person. Second, the number of people that needs to be classified is large. It will take human analysts a long time before they can complete this task.

This chapter focuses on automated techniques for solving the classification problem. As many techniques have been developed, choosing the right one for a given data set can be quite challenging. Therefore, it is important to know the strengths and limitations of each technique, particularly in the context of handling practical issues such as noisy data and missing values. Methods for comparing the performance of various classification techniques are also presented in this chapter.

3.1 Problem Definition

We assume that the data set consists of a collection of *records*. Each record, also known as an *instance* or *example*, is characterized by a tuple (\mathbf{x}, y) , where \mathbf{x} is the attribute set and y is the class label. A record is *labeled* if the value of y is known; otherwise, the record is said to be *unlabeled*. Each attribute $x_k \in \mathbf{x}$ can be discrete or continuous. On the other hand, the class label y must be a discrete variable whose value is chosen from a finite set $\{y_1, y_2, \dots, y_c\}$. If y is a continuous variable, then this problem is known as *regression*.

Example 1 Table 3.1 illustrates an example of the vertebrate data set. Each row represents a vertebrate and each column is an attribute of the vertebrate, with the last column representing the class label. For instance, the first record of this table contains the attributes and class label of a human being.

Table 3.1. The vertebrate data set.

Name	Blood type	Skin cover	Gives birth	Lays eggs	Can fly	Lives in water	Has legs	Hibernates	Class label
human	warm	hair	yes	no	no	no	yes	no	mammal
python	cold	scales	no	yes	no	no	no	yes	reptile
salmon	cold	scales	no	yes	no	yes	no	no	fish
whale	warm	hair	yes	no	no	yes	no	no	mammal
frog	cold	none	no	yes	no	sometimes	yes	yes	amphibian
komodo dragon	cold	scales	no	yes	no	no	yes	no	reptile
bat	warm	hair	yes	no	yes	no	yes	yes	mammal
pigeon	warm	feather	no	yes	yes	no	yes	no	bird
cat	warm	fur	yes	no	no	no	yes	no	mammal
leopard shark	cold	scales	yes	no	no	yes	no	no	fish
turtle	cold	scales	no	yes	no	sometimes	yes	no	reptile
penguin	warm	feather	no	yes	no	sometimes	yes	no	bird
porcupine	warm	quills	yes	no	no	no	yes	yes	mammal
eel	cold	scales	no	yes	no	yes	no	no	fish
salamander	cold	none	no	yes	no	sometimes	yes	yes	amphibian

The classification problem can be stated formally as follows:

Classification is the task of learning a function, $f : \mathbf{x} \rightarrow y$, that maps each attribute set \mathbf{x} into one of the class labels for y .

f is known as the *target function* or *classification model*. We will use both terms interchangeably throughout this chapter.

In general, f is chosen from a set of permissible functions \mathcal{H} , which is better known as the *hypothesis space*. Examples of hypothesis space include the set of boolean functions, polynomial functions, and mixtures of gaussian kernel functions. A hypothesis space is *expressive* if it can represent any types of relationships between

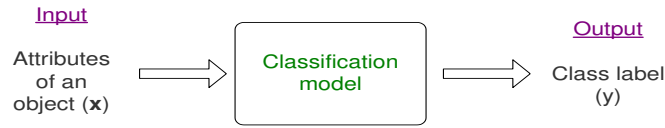


Figure 3.1. Using a classification model for prediction.

the attribute set and the class label. An expressive hypothesis space is desirable because it can model any relationships that exist in the data. The drawback is that such an expressive hypothesis space may require a huge amount of data to learn the correct model.

In general, a classification model can be used for the following purposes.

- It can serve as an explanatory tool for distinguishing objects of different classes. For example, it would be useful - for biologists and others - to have a model that summarizes the data shown in Table 3.1 and explains why each vertebrate should be categorized as a mammal, reptile, bird, fish, or amphibian. This is the *descriptive* element of the classification model.
- It can be used to predict the class labels of new records. This is the *predictive* element of the classification model. During prediction, a classification model can be treated as a black box that automatically generates an appropriate class label when attributes of the new record are presented to the model, as shown in Figure 3.1. For example, suppose we are given only partial information about the following vertebrate:

Name	Blood type	Skin cover	Gives birth	Lays eggs	Can fly	Lives in water	Has legs	Hibernates	Class label
gila monster	cold	scales	no	yes	no	no	yes	yes	?

How do we determine its class label? If a classification model is built from the data shown in Table 3.1, then we can apply this model to predict the type of species to which a gila monster belongs.

3.2 General Approach to Solving a Classification Problem

This section presents a general approach to solving the classification problem. We begin with an example illustrating how an organization such as the Internal Revenue Service (IRS) may want to build a classification model for predicting taxpayers who try to evade paying their taxes. The IRS keeps historical records containing personal information about the audited taxpayers. Each record is assigned a class label to indicate whether the audited taxpayer has evaded on his or her taxes. The goal of this classification problem is to build a model from the historical audit data and use it to predict the class label of new taxpayer records, as shown in Figure 3.2

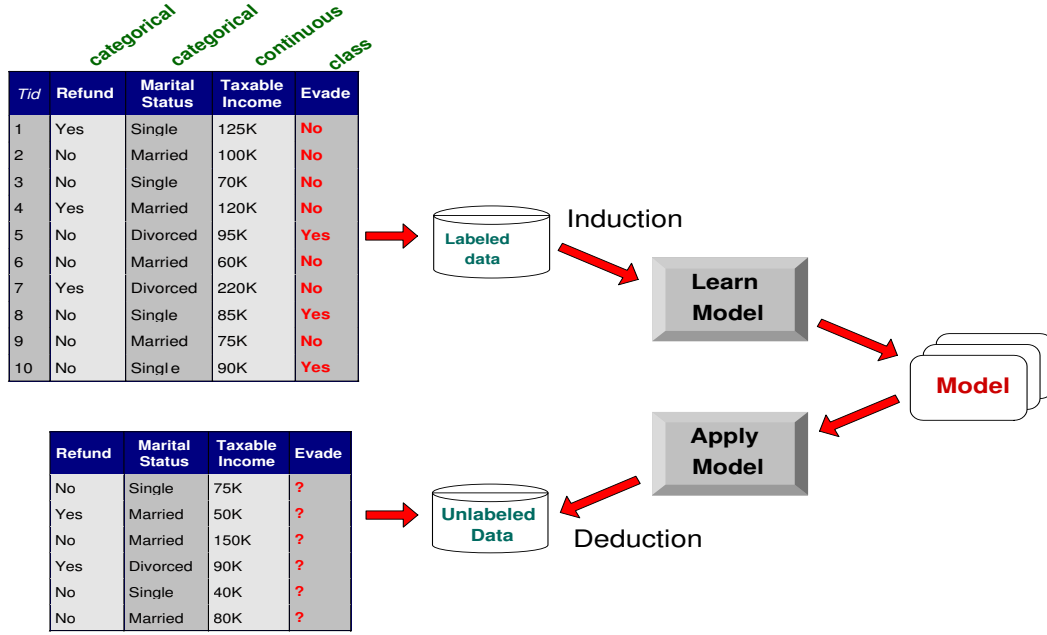


Figure 3.2. Building a classification model for predicting tax evasion.

To build such a classification model, the labeled data set is initially partitioned into two disjoint sets, known as the *training set* and *test set*, respectively. Next, a *classification technique* is applied to the training set to induce a *classification model*. Each classification technique employs a *learning algorithm* to search the hypothesis space \mathcal{H} and selects the model that produces outputs consistent with the class labels of the training examples. The output of a model f is consistent with a training example (\mathbf{x}, y) if and only if $f(\mathbf{x}) = y$.

However, a model that fits perfectly to the training set may not necessarily perform well on new examples it has never seen before. Thus, the goal of a learning algorithm is to build a model that has good *generalization* capability, i.e., it must not only fit the training set well, but can also predict correctly the class labels of many previously unseen examples. To evaluate how well the induced model performs on examples it has not seen before, we can apply it to the test set.

The performance of a model can be summarized by using a $c \times c$ confusion matrix, where c is the number of classes. The confusion matrix for a 2-class problem is shown in Table 3.2. In this table, the total number of correct predictions made by the classification model is $(a + d)$, while the total number of wrong predictions is $(b + c)$. Thus, we can define the accuracy of the model as

$$\text{Accuracy} = \frac{a + d}{a + b + c + d} = \frac{a + d}{N}, \quad (3.1)$$

where N is the total number of predictions. Similarly, the error rate of the model

is given by

$$\text{Error rate} = \frac{b + c}{a + b + c + d} = \frac{b + c}{N} \quad (3.2)$$

A good classification model must have high accuracy, or equivalently, low error rate when applied to the test set.

Table 3.2. Confusion matrix for a 2-class problem.

		Predicted Class	
		<i>Class</i> = y_1	<i>Class</i> = y_2
Actual Class	<i>Class</i> = y_1	<i>a</i>	<i>b</i>
	<i>Class</i> = y_2	<i>c</i>	<i>d</i>

In the next several sections, we will describe a variety of classification techniques used for building classification models.

3.3 Decision Tree Induction

The first classification technique we present in this chapter is called *decision tree induction*. This technique is introduced first because it is one of the most widely-used technique for classification and its concepts are relatively simple to understand. In this section, we will not only explain how decision tree induction works but will also discuss some of the general issues involved in building decision trees from data.

3.3.1 How Decision Tree Works?

To get an idea of how classification with decision tree works, let us consider the problem of classifying vertebrates as mammals and non-mammals. Suppose a new type of vertebrate has just been discovered deep in the jungles of the Amazon forest. How do we determine whether it is a mammal or non-mammal? A simple way to do this is to pose a series of questions to learn about the characteristics of the creature. First, one can ask whether it is a warm- or cold-blooded creature. If the vertebrate is cold-blooded, then it is definitely not a mammal. On the other hand, if the vertebrate is warm-blooded, then it could either be a bird or a mammal. We could ask a follow-up question to distinguish between a bird and a mammal - does the female species of the vertebrate give birth to the young? If the answer is **yes**, then it is definitely a mammal; otherwise, it is most likely a non-mammal (with the exception of egg-laying mammals such as platypus and spiny anteater).

The previous example illustrates how classification can be performed by simply asking a series of carefully crafted questions about the attributes of the unknown instances. Each time an answer is given, a follow-up question is asked until we reach a conclusion about the most likely class label of the instance. The series of questions and their possible answers can be represented in the form of a decision tree, which

is a hierarchical structure consisting of nodes and directed edges. There are three types of nodes in a decision tree:

- A *root node*, which has no incoming edges and zero or more outgoing edges.
- *Internal nodes*, each of which have exactly one incoming edge and two or more outgoing edges.
- *Leaf nodes*, each of which have exactly one incoming edge and no outgoing edges. Each leaf node also has a class label attached to it.

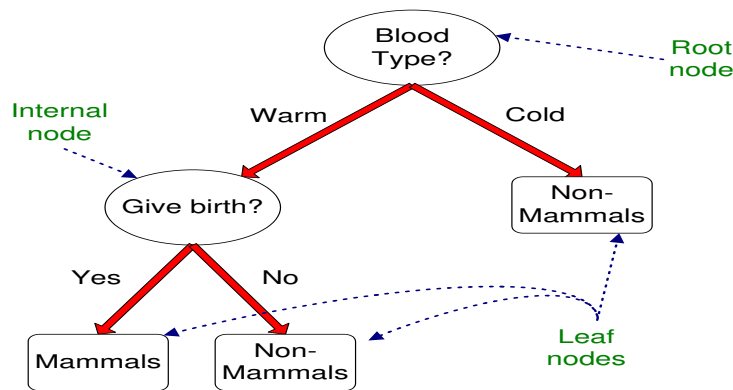


Figure 3.3. An illustrative example of the decision tree for mammal classification problem.

Figure 3.3 illustrates an example of a decision tree for the mammal classification problem. The tree contains a root node, an internal node, and three leaf nodes. The *non-terminal* nodes, which consist of the root node and internal nodes, are used to partition the examples into smaller subsets that are more homogeneous with respect to class. For example, the root node (*Blood Type?*) is used to eliminate cold-blooded non-mammals and the internal node (*Gives Birth?*) is used to distinguish between mammals and other warm-blooded non-mammals, which are mostly birds.

Once a decision tree is constructed, classifying new examples is a straightforward task. Starting from the test condition at the root node, we evaluate the attributes of the unlabeled example and follow the appropriate outgoing link 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 eventually, to a leaf node. When a leaf node is encountered, the new example is classified according to the class label of the leaf node, as shown in Figure 3.4.

In general, decision tree induction is a useful classification technique for the following reasons:

1. It is highly expressive in terms of capturing relationships among discrete variables.
2. It is relatively inexpensive to construct and extremely fast at classifying new instances.

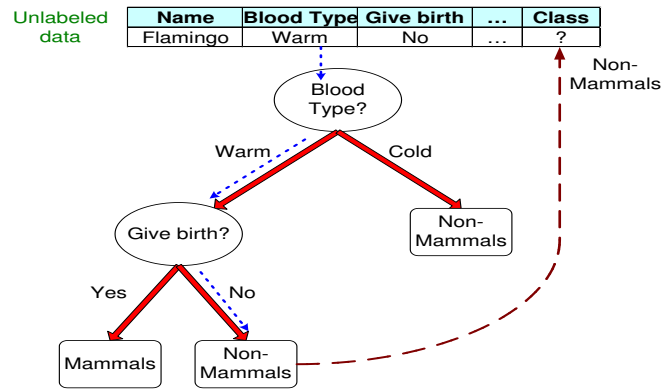


Figure 3.4. Classifying an unlabeled vertebrate.

3. For small-sized trees, it is relatively easy to interpret.
4. It can effectively handle both missing values and noisy data, as will be discussed later in this section.
5. It can achieve accuracy comparable to other classification techniques in many application domains.

3.3.2 How to Build a Decision Tree?

Hunt's Algorithm

Most of the decision-tree induction algorithms are based on the original ideas proposed in the Hunt's algorithm. The following is a recursive definition of Hunt's algorithm. Let D_t be the set of training instances and $y = \{y_1, y_2, \dots, y_c\}$ be the class labels.

1. If D_t contains instances that belong to the same class, y_k , then its decision tree consists of a leaf node labeled as y_k .
2. If D_t is an empty set, then its decision tree is a leaf node whose class label is determined from other information such as the majority class of the instances.
3. If D_t contains instances that belong to several classes, then a *test condition*, based on one of the attributes of D_t , is applied to split the data into more homogeneous subsets. The test condition is associated with the root node of the decision tree for D_t . D_t is then partitioned into smaller subsets, with one subset for each outcome of the test condition. The outcomes are indicated by the outgoing links originating from the root node. We then recursively apply this method to each subset created by the test condition.

Figure 3.5 illustrates an example of how to apply the Hunt's algorithm to the tax evasion problem defined in Section 3.1. Initially, the tree consists of a single root node with class label $Evade = No$ as most of the audited taxpayers shown

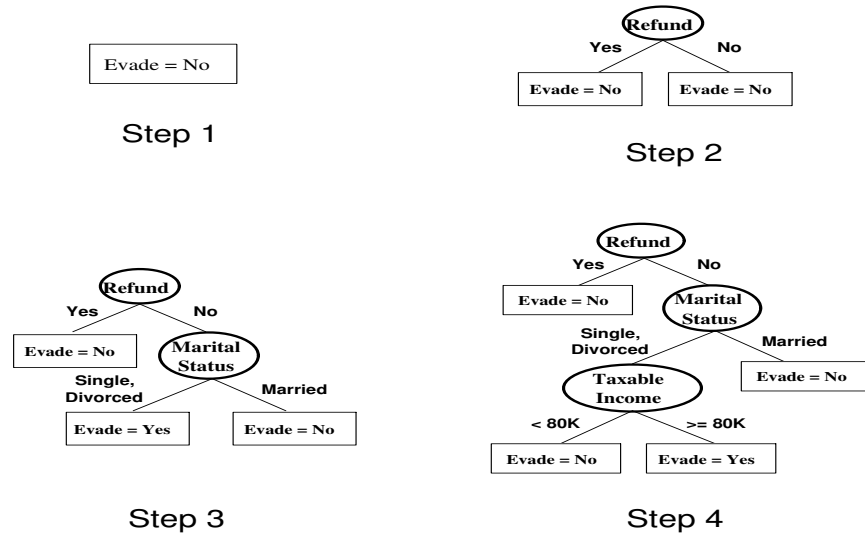
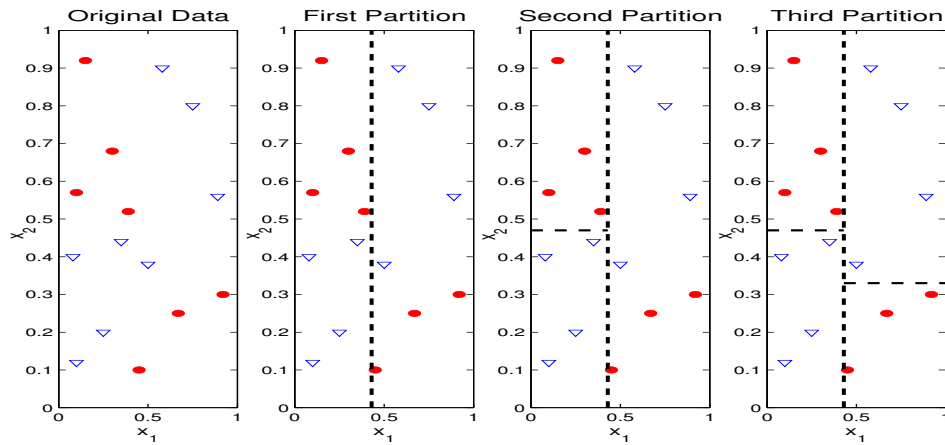


Figure 3.5. Hunt's Algorithm for inducing decision trees.

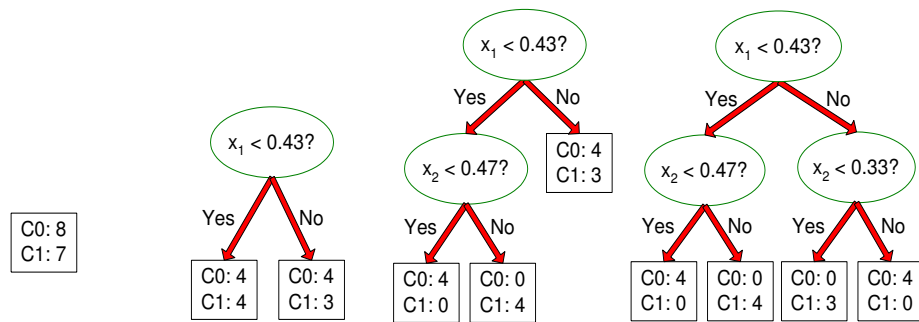
in Figure 3.2 do not evade on their taxes. Since the data contains instances from both classes, we may decide to split the data into smaller subsets according to the *Refund* attribute (Step 2). Notice that all the instances with (*Refund* = *No*) belong to the same class. As a result, a leaf node will be created and we do not have to split the instances any further. However, for *Refund* = *Yes*, the instances are still not quite pure. We then apply the recursive definition above repeatedly until all the instances associated with the leaf nodes belong to the same class (Steps 3 and 4).

As another example, consider the diagram shown in Figure 3.6. The data set consists of two-dimensional data points, marked as triangles and circles. Let us assume that the triangles belong to class 0 and the circles belong to class 1. Initially, the input space contains 8 triangles and 7 circles. This information is depicted by the rectangular node shown in Figure 3.6(b)(i), where C_0 is the number of points that belong to class 0 and C_1 is the number of points that belong to class 1. Suppose we decide to split the data along the horizontal axis at $x_1 = 0.43$, as shown in the second diagram of Figure 3.6(a). Splitting the data at this point is equivalent to converting the initial node into a root node that tests for the condition $x_1 < 0.43$ (see Figure 3.6(b)(ii)). Any data points that satisfy this condition will be assigned to the left child of the new node, while those that do not satisfy this condition will be assigned to the right child of the node.

For the left partition, observe that all the circles reside on the top half, while the triangles reside only on the bottom half. This observation suggests that a decision boundary can be created at $x_2 = 0.47$ to split the left partition into two regions of homogeneous classes, as shown in the third diagram of Figure 3.6(a). Splitting the left partition at this point is equivalent to transforming the left child of the root node into an internal node that tests for the condition $x_2 < 0.47$. Any data points

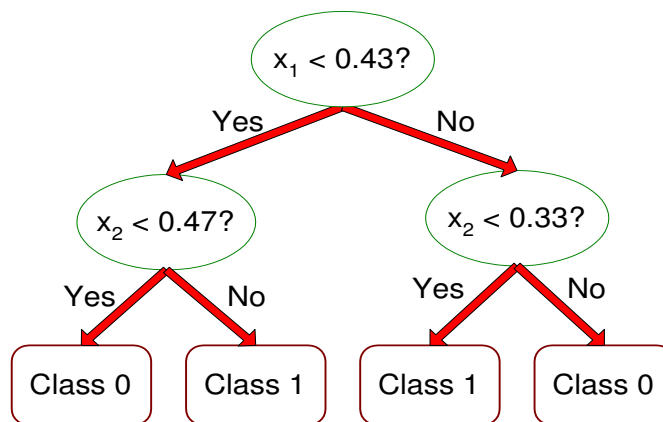


(a) Partitioning the input space



(i) Original Data (ii) First Partition (iii) Second Partition (iv) Third Partition

(b) Decision tree induction



(c) Final decision tree

Figure 3.6. Example of decision tree induction for a 2-dimensional data set.

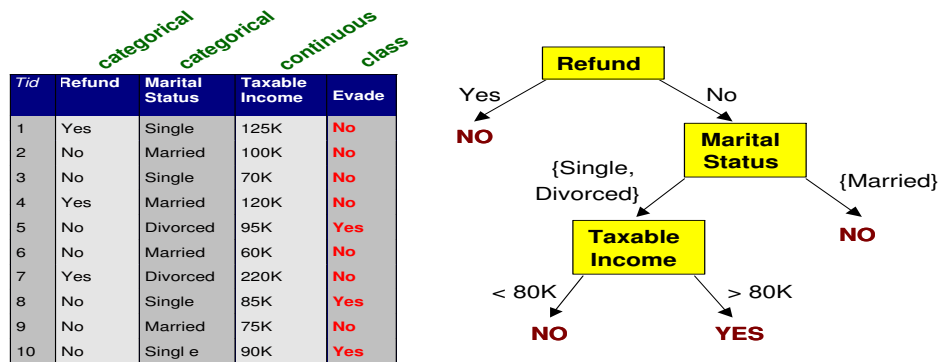


Figure 3.7. Decision tree for the taxpayer classification problem.

that satisfy this condition are assigned to the left child of the internal node, while those that do not satisfy the condition are associated with the right child. The current shape of the tree is shown in Figure 3.6(b)(iii).

We continue to partition the attribute space until each region contains mostly data points from the same class (see Figure 3.6(b)(iv)). The final decision tree is obtained by converting the class distributions of the leaf nodes into class labels that represent the majority class of points, as shown in Figure 3.6(c).

General Approach for Tree Induction

So far, we have described how a decision tree works without delving too deeply into the details of decision tree construction. In this section, we take a step back and examine how decision trees can be induced from data. For illustrative purposes, we consider the taxpayer classification problem, which was first introduced in Section 3.1. The decision tree for this problem is shown in Figure 3.7.

While various decision tree induction algorithms have been developed in recent years, the greedy, top-down recursive partitioning approach is still the most popular strategy and our discussion on decision tree induction will focus on this approach. In general, growing a decision tree involves the following tasks:

1. **Determine how to split the instances.** Decision tree algorithms often use greedy heuristics to make a series of locally optimum decisions about which attribute to use for partitioning the data. Such a greedy strategy may result in suboptimal solutions, especially when no backtracking is allowed by the algorithm. (One of the exercises in this chapter will illustrate a suboptimal solution of a decision tree.)

At each step of the greedy algorithm, a test condition is applied to split the data into subsets with a more homogeneous class distribution. Among the key issues to be addressed in this step include:

- (a) The different ways to do the splitting. Since decision tree induction works for various types of attributes, it is important to know how to specify the

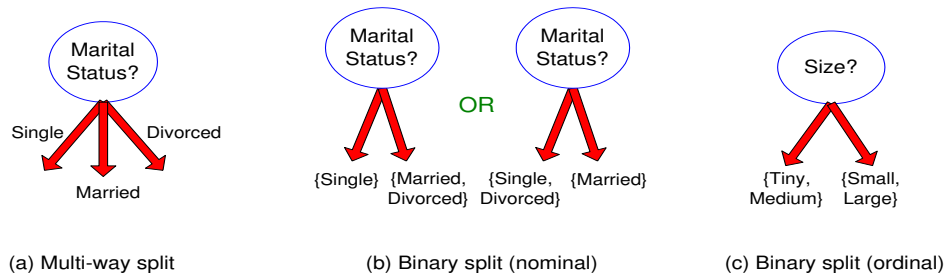


Figure 3.8. Splitting instances based on categorical attributes.

test condition for each attribute type. We will address this issue when we discuss methods of splitting in the next section

- (b) How to determine the best split. An objective measure is needed to determine how good a split is in terms of making the subsets purer. We will address the various measures later in this chapter.

2. **Determine when to stop splitting.** A stopping condition is needed to terminate the tree growing process. Two of the most widely-used stopping conditions are (1) stop expanding a node if all the instances belong to the same class, and (2) stop expanding a node if all the instances have similar attribute values. While these two conditions are often sufficient, other stopping conditions can be imposed to make the tree growing procedure stop earlier. The advantages of early termination are discussed later in this chapter.

3.3.3 Methods for Splitting

A key step towards building a decision tree is to find an appropriate test condition for splitting the data into purer subsets. Below, we describe the different methods available to specify the test condition for splitting instances. The methods are grouped by attribute type: categorical or continuous.

Categorical Attributes. For categorical attributes, the test condition can be expressed as an attribute-value pair ($A = v?$) whose outcomes are Yes/No, or as a question about the value of an attribute ($A?$). In the latter case, the number of outcomes for the test condition depends on the number of distinct attribute values. For example, since the attribute *Marital Status* has three possible values (single, married, or divorced), there are as many as three possible outcomes for the test condition (*Marital Status?*), as illustrated in Figure 3.8(a). While this approach is simple to implement, it may lead to undesirable effects, especially when the number of attribute values is too large.

For example, Figure 3.9 shows three different ways to split a data set that contains 10 instances of class 0 and 10 instances of class 1. If the attributes (*Own Car?*) and (*CarType?*) are compared, one could see that (*CarType?*) seems to provide a better way to split the data because the class distributions

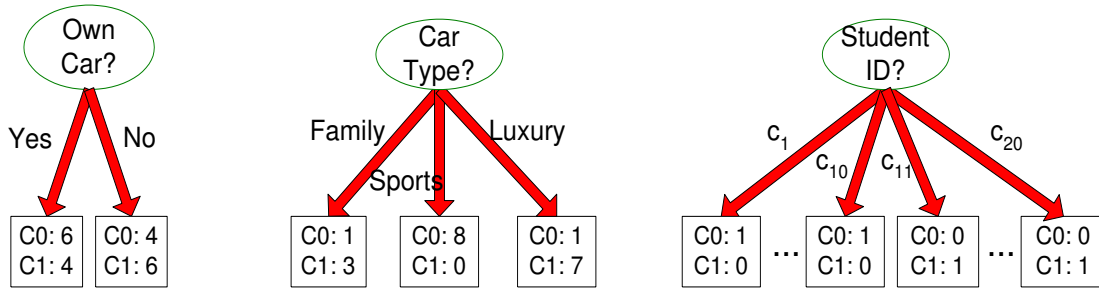


Figure 3.9. Splitting the data set according to different attribute values.

in the descendent nodes are more homogeneous. If we compare both attributes against (*Student Id?*), the latter seems to produce subsets with the most homogeneous class distributions. However, (*Student Id?*) is not useful as a test condition because the attribute value is unique for all instances. Such an attribute may not generalize well to instances it has not seen before. This is because whenever the number of instances covered by a node is too small, any decisions the node makes may not be statistically significant and is sensitive to noise. This is also known as the *small disjunct problem*.

One way to overcome this problem is to group together some of the attribute values into larger subsets, as shown in Figure 3.8(b). Grouping of attribute values is also required for a decision tree algorithm such as CART that uses 2-way splits. While grouping attribute values may overcome the small disjunct problem, it incurs additional overhead to find the optimal way for grouping the attribute values effectively. It is also important to distinguish between grouping ordinal and nominal categorical attributes. Unlike nominal attributes, some of the groupings for ordinal attributes could be counter-intuitive especially if they violate the order property of the attribute. For example, the binary split shown in Figure 3.8(c) violates the order property of the *Size* attribute.

Continuous Attributes. For continuous attributes, the test condition can be expressed in terms of a binary decision ($A < v?$) or ($A \geq v?$), whose outcomes are Yes/No, or as a range query whose outcomes are $v_i \leq A < v_{i+1}$, for $i = 1, \dots, k$ (see Figure 3.10). For the binary case, one may consider all possible split positions v and selects the one that gives the best split, i.e., partitions the instances into subsets of homogeneous class distributions. In the second approach, the ranges of the continuous values are obtained by using the discretization techniques described in Chapter 2. After discretization, a new ordinal attribute value can be assigned to each discretized interval. One may group some the new attribute values into larger subsets as long as the order property is not violated.

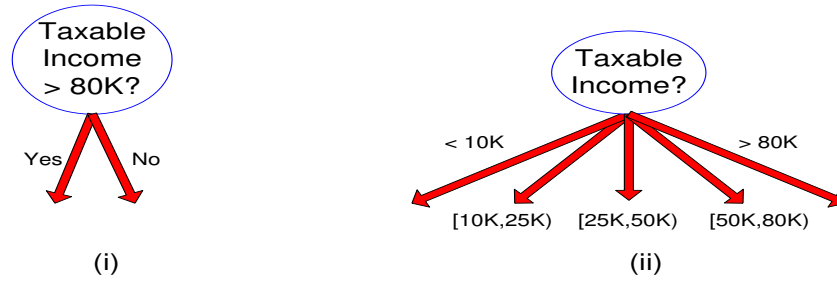


Figure 3.10. Splitting continuous attributes.

3.3.4 Measures for Selecting the Best Split

There are many test conditions one could apply to partition a collection of instances into smaller subsets. We now discuss the various measures available to determine which test condition provides the best split. The measures used to discriminate instances belonging to different classes are defined in terms of (1) the class distribution of instances in the parent node, and (2) the class distributions of instances in the child nodes.

Let $P(i|t)$ denotes the fraction of instances belonging to class i at a given node t . We express this fraction as $P(i|t) \equiv p_i$ when the context (node) for which the value is defined is clear. For a two-class problem, we denote the class distribution of instances at a given node as (p_1, p_2) , where $p_2 = 1 - p_1$. For example, in Figure 3.9, the class distribution for the parent node is $(0.5, 0.5)$ since instances from both classes are equally represented at the root node. For attribute A , the class distributions for its two child nodes are $(0.6, 0.4)$ and $(0.4, 0.6)$, respectively. For attribute B , the class distributions of its child nodes are $(0.25, 0.75)$, $(1, 0)$, and $(0.125, 0.875)$, respectively. For attribute C , the class distributions are $(1, 0)$ for child nodes connected via the links c_1, c_2, \dots, c_{10} and $(0, 1)$ for child nodes connected via the links $c_{11}, c_{12}, \dots, c_{20}$.

A splitting function decides which test condition to use by examining the degree of *impurity* of its child nodes. For example, a node with class distribution $(0, 1)$ would have zero degree of impurity since all of its instances belong to the same class. A test condition that produces such a node would be preferred over another that produces nodes with uniform class distribution $(0.5, 0.5)$.

Below, we define three impurity measures for a given node t .

$$\text{Entropy}(t) = - \sum_{i=1}^c p(i|t) \log_2 p(i|t) \quad (3.3)$$

$$\text{Gini}(t) = 1 - \sum_{i=1}^c [p(i|t)]^2 \quad (3.4)$$

$$\text{Classification error}(t) = 1 - \max_i [p(i|t)] \quad (3.5)$$

where c is the number of distinct classes. Figure 3.11 compares the values of the different impurity measures when applied to a node containing instances from two

different classes. The horizontal axis corresponds to the estimated probability for one of the classes while the vertical axis corresponds to the impurity measure for the node. Note that each measure attains its maximum value when the class distribution is uniform, and attains its minimum value when the estimated probability for one of the classes is equal to 0 or 1.

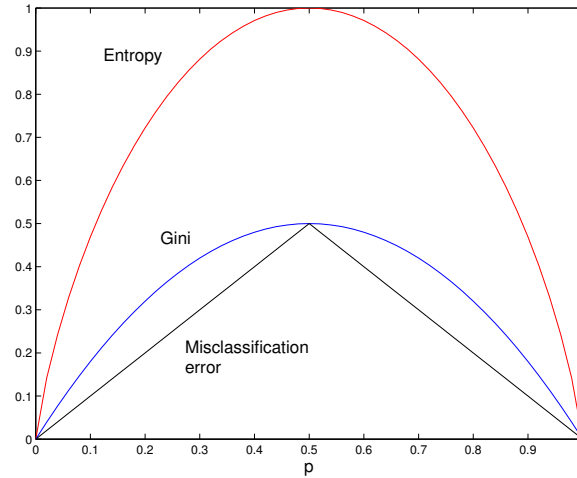


Figure 3.11. Comparison between the impurity functions for two-class problems.

Below, we illustrate a few examples for calculating the different impurity measures for a given node. Note that $0 \log_2 0 = 0$ in the entropy calculations.

Node N_1	Count
Class=0	0
Class=1	6

$$\text{Gini} = 1 - (0/6)^2 - (6/6)^2 = 0$$

$$\text{Entropy} = -(0/6) \log_2(0/6) - (6/6) \log_2(6/6) = 0$$

$$\text{Error} = 1 - \max[0/6, 1/6] = 0$$

Node N_2	Count
Class=0	1
Class=1	5

$$\text{Gini} = 1 - (1/6)^2 - (5/6)^2 = 0.278$$

$$\text{Entropy} = -(1/6) \log_2(1/6) - (5/6) \log_2(5/6) = 0.650$$

$$\text{Error} = 1 - \max[1/6, 5/6] = 0.167$$

Node N_3	Count
Class=0	3
Class=1	3

$$\text{Gini} = 1 - (3/6)^2 - (3/6)^2 = 0.5$$

$$\text{Entropy} = -(3/6) \log_2(3/6) - (3/6) \log_2(3/6) = 1$$

$$\text{Error} = 1 - \max[3/6, 3/6] = 0.5$$

To determine how good a test condition is, we need a splitting function that compares the impurity measure of the parent node (prior to splitting) against the weighted-averaged of the impurity measure of the child nodes (after splitting). The conditions for a splitting function are stated below.

1. Intuitively, dividing instances into smaller groups can only make the groups purer. As a result, when a node is partitioned into several child nodes, the

weighted average of the impurity measure for the child nodes should be lower than the impurity measure for the parent node.

2. The larger the difference between the impurity measure of the parent node and its child nodes, the more discriminating the test condition.

A splitting function that satisfies the above conditions is:

$$\text{Splitting function, } \Delta = I(\text{parent}) - \sum_{j=1}^k \frac{N(v_j)}{N} I(v_j) \quad (3.6)$$

where $I(\cdot)$ is the impurity measure of a given node, N is the total number of instances at the parent node, k is the number of attribute values, and $N(v_j)$ is the number of instances associated with a child node having attribute value v_j . When entropy is used as the impurity measure in Equation 3.6, the splitting function is known as *information gain*, Δ_{gain} .

Splitting function for Binary Attributes

Consider the diagram shown in Figure 3.12. Suppose there are two ways to split the data into smaller subsets. Before splitting, the gini index is 0.5 since there are equal number of instances in both classes. If we use the binary attribute A to split the data, the gini index for node N1 is 0.4898 and node N2 is 0.48. Using equation 3.6, the splitting function is $\Delta = 0.5 - (7/12) * 0.4898 - (5/12) * 0.48 = 0.5 - 0.486 = 0.014$. Similarly, we can show that the splitting function for attribute B is $0.5 - 0.375 = 0.125$. Since attribute B has a larger splitting function, it should be used as the test condition for the root node.

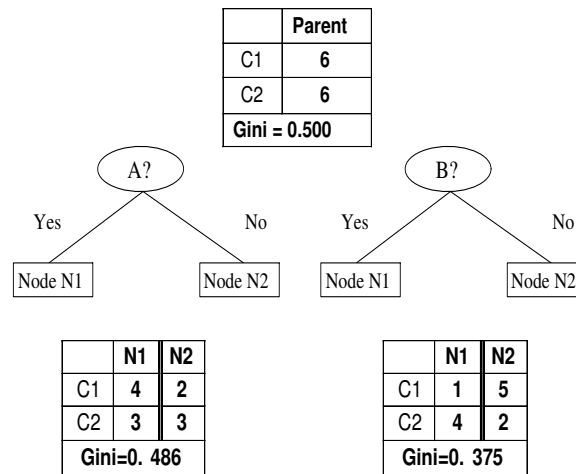


Figure 3.12. Splitting binary attributes.

Splitting function for Categorical Attributes

Consider the diagram shown in Figure 3.13. In this example, we compare the gini index for multi-way splitting of categorical attributes against the gini index for two-way splits. Since a two-way split merges some of the attribute values, its gini index must be higher than the gini index for any multi-way split.

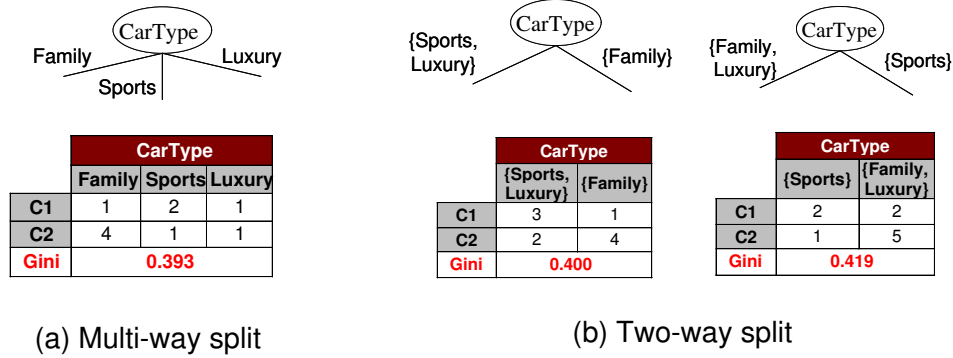


Figure 3.13. Splitting categorical attributes.

Splitting function for Continuous Attributes

Consider the example shown in Figure 3.14, where we are attempting to find the optimal split position (v) for the test condition $Taxable\ Income < v$ or $Taxable\ Income \geq v$. We assume that the split position is optimal if it minimizes the gini index. A trivial way to select v is to use the attribute values for all the N instances as the candidate split positions. We then compute the gini index for each candidate and choose the one that gives the lowest gini index. While this approach seems trivial, it can be expensive. This is because, for each candidate split position, computing the gini index requires another scan over the entire data set to count the number of instances less than or greater than v . Thus, each candidate split position would require $O(N)$ operations. Since there are N candidates, the overall complexity of this trivial approach is $O(N^2)$.

We can reduce the cost of this step from a quadratic complexity to $O(N \log N)$. First, the attribute values are sorted. The candidate split positions now correspond to values located between two adjacent sorted values. At the beginning, none of the classes have instances smaller than the first candidate split position, which is why the counts for both *Yes* and *No* classes are 0. We then compute the gini index at each adjacent split positions by scanning the sorted array from left to right while updating the counts for each class, as shown in Figure 3.14. The best split position corresponds to the one that produces the smallest gini index, i.e., $v = 97$. This step requires $O(N \log N)$ operations for sorting and $O(N)$ for computing the gini index at each candidate split position. The scanning step can be optimized further by considering only candidate split positions located between two adjacent attribute

values with different class labels. With this approach, we can ignore the candidate split positions at 55, 65, 72, 87, 92, 110, 122, 172, and 230 because they are located between instances that have identical class labels.

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

Figure 3.14. Splitting continuous attributes.

Example 2 Let us apply the information gain splitting function to the examples shown in Figure 3.9. The entropy measure of the parent node is $I(\text{parent}) = -0.5 \log_2 0.5 - 0.5 \log_2 0.5 = 1$. For attribute A , we need to compute the entropy measures for both of its child nodes:

$$I(a_1) = -0.6 \log_2 0.6 - 0.4 \log_2 0.4 = 0.971 \quad (3.7)$$

$$I(a_2) = -0.4 \log_2 0.4 - 0.6 \log_2 0.6 = 0.971 \quad (3.8)$$

Since there are 10 instances with attribute value a_1 and 10 instances with attribute value a_2 , $P(a_1) = P(a_2) = 0.5$. Applying equation 3.6, the information gain for attribute A is:

$$I(\text{parent}) - P(a_1)I(a_1) - P(a_2)I(a_2) = 1 - 2 \times 0.5 \times 0.971 = 0.029$$

Similarly, for attribute B ,

$$I(b_1) = -1/4 \log_2(1/4) - 3/4 \log_2(3/4) = 0.811 \quad (3.9)$$

$$I(b_2) = -0 \log_2 0 - 1 \log_2 1 = 0 \quad (3.10)$$

$$I(b_3) = -1/8 \log_2(1/8) - 7/8 \log_2(7/8) = 0.544 \quad (3.11)$$

where $0 \log 0 = 0$. Furthermore, $P(b_1) = 4/20 = 0.2$, $P(b_2) = 8/20 = 0.4$, and $P(b_3) = 8/20 = 0.4$. As a result, the information gain for attribute B is:

$$I(\text{parent}) - P(b_1)I(b_1) - P(b_2)I(b_2) - P(b_3)I(b_3) = 1 - 0.2 \times 0.811 - 0.4 \times 0.544 = 0.620$$

Since the information gain for attribute B is higher than attribute A , one should prefer attribute B over attribute A as the test condition.

If we repeat the above analysis for attribute C , the information gain for C would be higher than both A and B . This is because information gain tends to favor attributes that have a large number of distinct values. A decision tree algorithm called C4.5 attempts to address this problem by using a different splitting function called *gain ratio*, which is obtained by dividing information gain with the splitting information.

$$\text{Gain ratio} = \frac{\Delta_{\text{gain}}}{\text{Split Info}} \quad (3.12)$$

where $\text{Split Info} = -\sum_{i=1}^k P(v_i) \log_2 P(v_i)$. If each attribute value has the same number of instances, then $\forall i : P(v_i) = 1/k$ and the split information would be equal to $\log_2 k$. This suggests that if the number of attribute values is large (i.e., k is large), its splitting information would also be large. Gain ratio penalizes such attributes by placing the splitting information in the denominator.

3.3.5 General Issues in Model Construction

There are several practical challenges when applying this technique to real data sets. Below, we describe some of the well-known problems that need to be addressed by any classification techniques.

Overfitting: Suppose the accuracy obtained for training a model is 99.9%. Does this guarantee that the overall accuracy of the model for the entire data set will also be 99.9%?

Example 3 Consider the problem of classifying vertebrates into mammals and non-mammals. Suppose the training set contains the following instances:

Name	Gives birth	Can fly	Hibernates	Class
turtle	no	no	no	non-mammal
python	no	no	yes	non-mammal
pigeon	no	yes	no	non-mammal
owl	no	yes	yes	non-mammal
human	yes	no	no	mammal
bear	yes	no	yes	mammal
bat	yes	yes	yes	mammal

For simplicity, we use the entire table as our classification model - ignoring the first attribute, which is the name of the vertebrate. As a result, our model is 100% accurate on the training set. Suppose the test data contains the following instances:

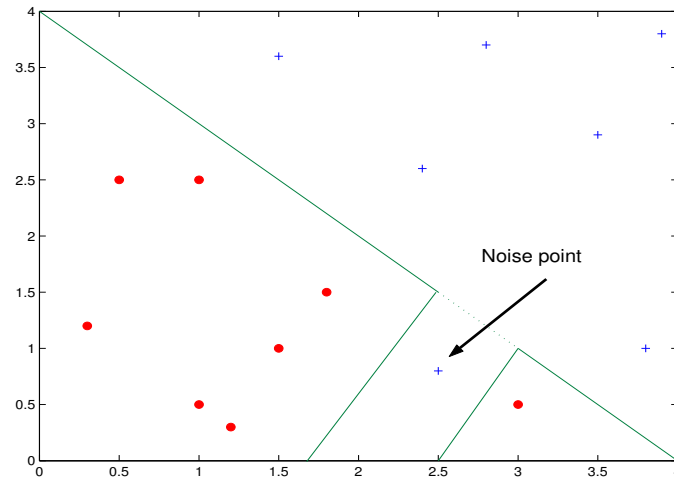
Name	Gives birth	Can fly	Hibernates	Class
platypus	no	no	no	mammal
leopard shark	yes	no	no	non-mammal

The attributes of a platypus are similar to those of a turtle, so it will be classified as a non-mammal, while the attributes of a leopard shark are similar to those of a human, so it will be classified as a mammal. In this extreme example, the accuracy obtained for the test set is 0%!

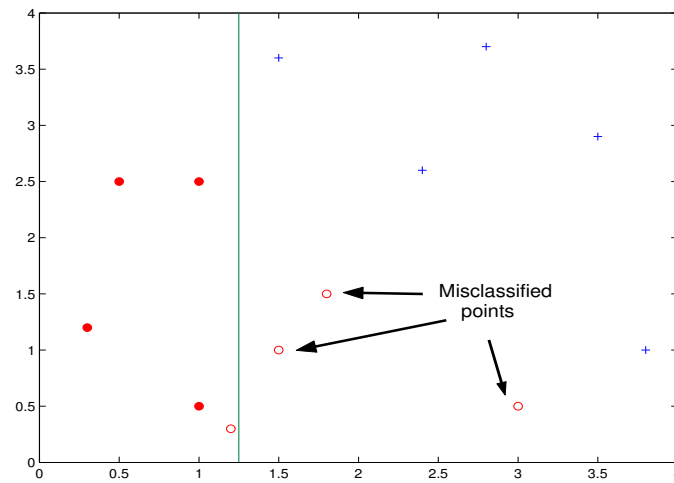
If the accuracy of a model degrades significantly when applied to the test set, then we say that the model *overfits* the training data. There are two situations in which overfitting may occur during model building. First, it may result from the presence of noise or errors in the training data. As a result, learning algorithms that attempt to fit perfectly to the training data would end up with the wrong model. For example, Figure 3.15(a) shows how a mislabeled data point can distort the decision boundary, producing a classification model that is more complex than necessary. (The decision boundary of a classification model corresponds to the border line that separates the different classes.) Such complex models often do not have a good generalization ability. If the noise point is labeled correctly, then the two classes can be distinguished easily by drawing a straight line that passes through the points (0,4) and (4,0). Second, even if the data is free of noise points, another possible reason for overfitting is the lack of training instances to learn the correct model. Figure 3.15(b) illustrates a training set that contains only four of the circular (opaque) points and five of the "+" points. The test set contains four circular (transparent) points. Since all the circular points of the training set are selected from the left side of the diagram, the decision boundary has changed dramatically. This causes many of the test circular points to be misclassified. We will examine the various ways to handle the overfitting problem later in the next section.

Missing Values: Many of the real data sets may contain missing values because no measurement is available on some of the attributes. These attributes could be missing because (1) it is costly to measure such attributes, (2) the measured value cannot be trusted, or (3) the attribute value is unavailable due to some other reasons. There are several generic approaches to handle this problem. First, one can ignore instances with missing values. However, this is not recommended, especially if we do not have enough examples for training. Alternatively, one could try to replace the missing attribute value with either the most common value throughout the data set, or with the most common value among instances from the same class. A third approach is to assign the missing value a probabilistic vector which gives a probability for each possible attribute value. This approach works only if the classification technique can handle attributes with probability vectors when computing the decision boundaries among classes. Finally, note that this problem must also be addressed during model application when unlabeled data contains missing attribute values.

Data Heterogeneity: Most of the examples from earlier sections have data sets with only one attribute type - either they are all continuous (for the



(a) Overfitting due to noise



(b) Overfitting due to insufficient number of training points

Figure 3.15. Overfitting problem during model building.

2-dimensional data set) or discrete (for the vertebrate data set). In reality, the data may contain a mixture of both continuous and discrete attributes. For such data sets, it is important to know how the different classification techniques will handle the different attribute types. Some techniques may require discretization of the continuous attributes, while others may require changing the continuous, distance-based metrics into similarity measures for discrete attributes.

Costs: How to deal with attributes that have different costs of measurement or classes that incur different costs of misclassification? For example, in the vertebrate data set, it is more expensive to measure the average length of each adult vertebrate than to determine whether the vertebrate can fly. In this case, the criteria for evaluating a classification model should take into account both model accuracy and the overall cost of the model. In other applications, such as detecting credit card frauds, the cost of misclassifying a fraudulent transaction as a valid transaction is more expensive than vice-versa. Therefore, one should prefer models that incur smaller misclassification costs. Some classification techniques may incorporate the cost of attributes or misclassification costs explicitly into their metrics during model building, while others may evaluate the overall cost only after the model has been constructed.

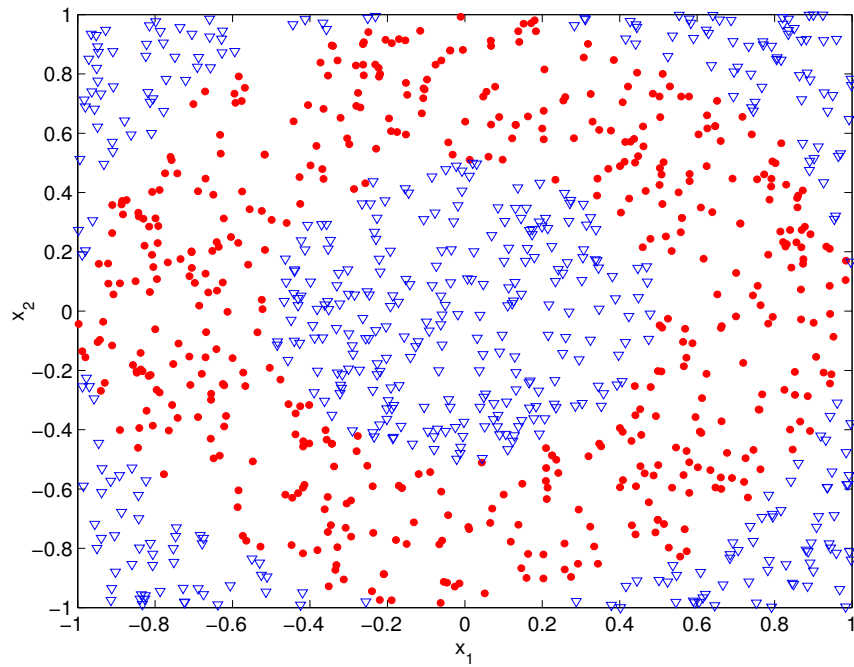
3.3.6 Handling Overfitting

So far, we have described how the training set can be used to induce a decision tree. We have also explained the role of the test set, which is to provide an unbiased estimate of the performance of the induced model. We now discuss the problem of overfitting and how decision tree induction algorithms attempt to fix this problem.

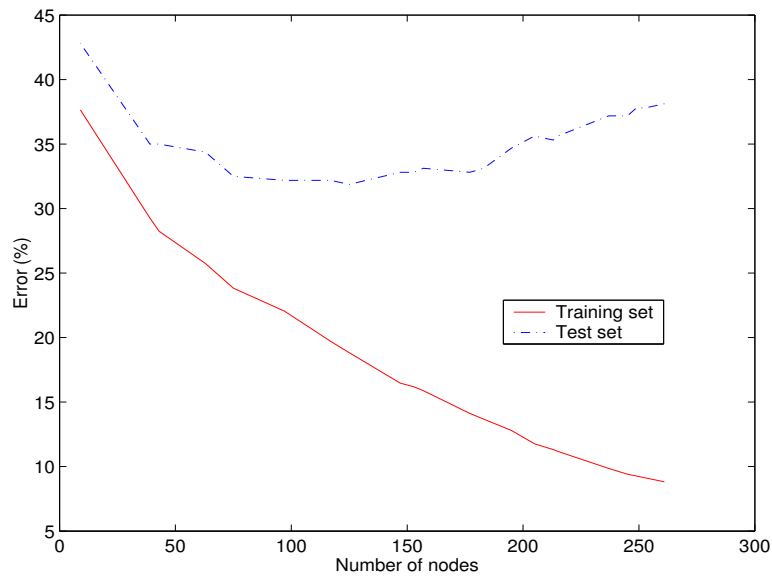
We begin with an example of a two-dimensional data set consisting of 500 triangular and 500 circular data points, as shown in Figure 3.16(a). The circular points are located between $0.5 \leq \sqrt{x_1^2 + x_2^2} \leq 1$ while the triangular points are located at $\sqrt{x_1^2 + x_2^2} < 0.5$ and $\sqrt{x_1^2 + x_2^2} > 1$. Two-third of the data points are used for training while the rest are used for testing. The C4.5 decision tree algorithm is applied to the data set with different threshold parameters to control the size of the decision tree. We then compute the misclassification error rate for both the training and test sets. Misclassification errors on the training set are often known as *resubstitution errors* while misclassification errors on test set are also known as *generalization errors*. The results of this experiment are shown in Figure 3.16(b).

Observe that when the size of the tree is too small, underfitting occurs and the model performs poorly on both training and test sets. If the size of the tree increases, then both training and test errors improve. However, if the decision tree is too large, the test errors increase even though the training errors are decreasing. In this case, overfitting has occurred and the model will not generalize well to unseen instances.

The above example illustrates how the complexity of a model affects its generalization ability. If the model is too simple, it may not fit the training and test



(a) Data set for experiment.



(b) Training and Test errors for decision trees at different model complexity.

Figure 3.16. Overfitting and Underfitting.

sets well. Likewise, if the model is too complex, overfitting may occur and reduce its ability to generalize beyond the training instances. During model building, the learning algorithm has access only to the training data. As a result, it cannot determine how well the model it has selected would perform on the overall data set. This requires some means to estimate the generalization error of a model. Here are several well-known techniques used to approximate the generalization error of a decision tree:

Optimistic approach. This approach assumes that the training data is a good representation of the overall data. It uses the training error as an optimistic estimate of the generalization error. In this approach, no test data is needed and the best model is one that minimizes the training data.

Pessimistic Approach. Analogous to the previous approach, this method assumes that the generalization error can be estimated directly from the training data. However, the difference is that the generalization error is estimated after making some statistical corrections to the training error. Quinlan had initially estimated the generalization error rate for a node t , $r'(t)$, by using the continuity correction to a Binomial distribution, i.e., $r'(t) = [e(t) + 1/2]/n(t)$, where $e(t)$ is the number of misclassification errors on the training data and $n(t)$ is the total number of instances classified by the node t . Note that for large $n(t)$, $r'(t) \mapsto e(t)/n(t)$. If T corresponds to the entire tree with leaf nodes (t_1, t_2, \dots, t_k) , then the estimated generalization error rate of the tree is:

$$r'(T) = \frac{\sum_{i=1}^k [e(t_i) + 1/2]}{\sum_{i=1}^k n(t_i)} = \frac{[e(T) + k/2]}{\sum_{i=1}^k n(t_i)}$$

For example, a decision tree that contains 30 leaf nodes and misclassifies 50 of the 1000 training instances would have a resubstitution error rate of 5% and a pessimistic error rate of $(50 + 0.5 \times 30)/1000 = 6.5\%$.

The above analysis shows that the above pessimistic pruning strategy would penalize complicated models by assigning a cost penalty to each leaf node of the decision tree. For his C4.5 decision tree algorithm, Quinlan re-defined the pessimistic error rate to be $e'(t)/n(t) = e(t)/n(t) + \epsilon$, where ϵ is the upper bound of the error term computed from a Binomial distribution.

Reduced Error Pruning Approach. In this approach, the generalization error is not computed from the training data. Instead, the training set is divided into two smaller subsets: one of which is used for training while the other, known as the *validation set*, is used for estimating the generalization error. A conventional way to do this is to reserve one-third of the training data for validation, while the remaining two-thirds are used for building the model. The generalization error can be computed as an optimistic or pessimistic estimate of the error made on the validation data set. The drawback of this approach is that less data is available for training. One way to address this problem

is by using a method known as k -fold cross-validation, where we divide the training data into k blocks. We build the decision tree on $k - 1$ blocks and then estimate the error by testing the model on the remaining block. This procedure is repeated until each block is used once as the validation data set. In this case, the generalization error is given by the average error made for the k runs.

The example at the beginning of this section illustrates that overfitting is a problem when the complexity of the model is higher than necessary. This problem arises primarily because the model has accidentally fitted some of the noise points in the training set. As a result, its generalization ability suffers. This example indicates that it pays to choose smaller decision trees than larger ones, which agrees with a well-known principle known as *Occam's razor* or the *principle of parsimony*:

Definition 4 Occam's Razor: *Given two models of similar generalization errors, one should prefer the simpler model over the more complex model.*

Occam's razor principle makes sense because, for a complex model, there is a greater chance that it was fitted accidentally by the data. In the words of Einstein, "Everything should be made as simple as possible, but not simpler."

The above discussion suggests that it would be advantageous to incorporate model complexity when selecting an appropriate model to represent the data. One way to explicitly incorporate model complexity is to use a cost function that takes into account both the misclassification error on training data as well as the complexity of the model. This approach is known as the Minimum Description Length (MDL) principle.

To illustrate the MDL principle, consider the example shown in Figure 3.17. In this example, both persons A and B are given a set of instances with attribute

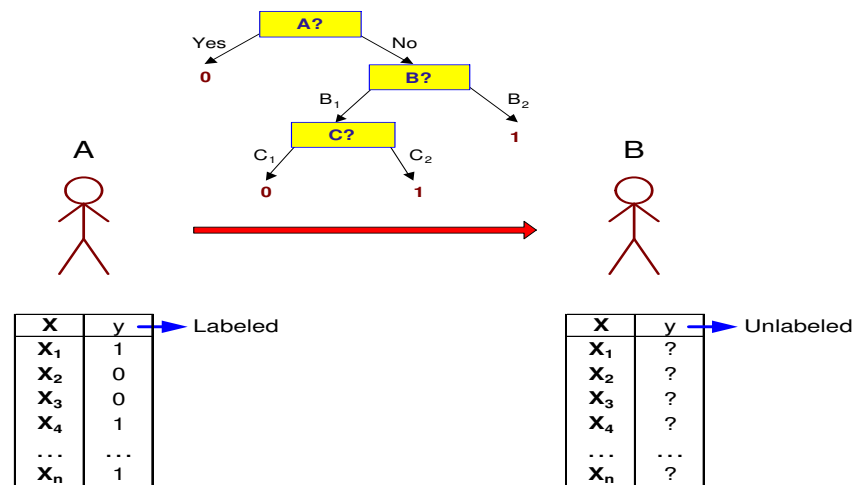


Figure 3.17. Minimum Description Length (MDL) principle.

values x . In addition, person A knows the exact class label for each instance but not

person B. The class label is assumed to be binary (0 or 1). One way for B to know the exact classification for each class would be to request A to transmit the class labels sequentially. Such a message would require $\Theta(n)$ bits of information. One way to reduce the message length is to build a decision tree that summarizes the relationship between \mathbf{x} and y , encodes the tree in a compact form, and transmits the code to B. According to the MDL principle, we should seek for a model that requires minimum description, i.e., produces the shortest code. The overall cost (or description length) for transmitting the information is:

$$Cost(model, data) = Cost(model) + Cost(data|model) \quad (3.13)$$

If the model is 100% accurate, then $Cost(data|model) = 0$. Otherwise, besides sending the model, we need to transmit information about which of the instances were misclassified by the model. The cost for doing this is given by $Cost(data|model)$. Thus, the MDL principle can be used to guide decision tree pruning by seeking for a decision tree that minimizes the overall cost given in Equation 3.13. An example for computing the description length of a decision tree is given as one of the exercises in this chapter.

The previous discussion illustrates the different ways to estimate the generalization ability of a decision tree. A decision tree with poor generalization ability tends to overfit the training data and should be less favorable than another with better generalization ability. Below, we describe several methods to overcome the overfitting problem.

Pre-pruning (Early Stopping Rule): In this approach, the tree growing algorithm is halted before it generates a fully-grown tree that fits the entire training data. This requires us to specify a stricter stopping condition than the ones described earlier in this section. Some of the conditions for stopping the tree expansion process prematurely are: (1) if the number of instances is too small to make any statistically significant decision, (2) if the distribution of instances in the current node are independent of the available features, e.g., they fail the standard χ^2 test of independence, and (3) if expanding the current node only increases the complexity of the model without improving the quality of prediction by a significant amount. We can use some of the techniques described previously (e.g. pessimistic pruning or reduced error pruning) to determine whether expanding a node into a subtree is better than leaving it as a leaf node.

Post-pruning: In this approach, the decision tree is initially grown to its entirety. This is immediately followed by a pruning step, which proceeds to trim some of the nodes of the current tree. Trimming is done by replacing a subtree with a new leaf node, whose class label is determined from the majority class of instances affiliated with the subtree. If the generalization error improves after trimming, the subtree will be replaced by the leaf node. The generalization error can be estimated using the optimistic or pessimistic approaches described

above. Pruning will stop when no further improvements can be achieved. Instead of estimating the generalization error explicitly from training data, an alternative strategy is to use Occam's razor principle and explicitly determine the model complexity. The assumption here is that the generalization error will be small when both training error and model complexity are low. We can use the minimum description length principle to determine whether pruning is needed.

3.3.7 Handling Missing Attribute Values

Decision tree induction can handle the missing attribute value problem using one of the generic approaches described in Section 3.3.5 or more sophisticated techniques that are specific to the decision tree induction algorithm. An example of the latter approach is given below for the C4.5 decision tree algorithm. Instances with missing attribute values can affect decision tree construction in three different ways: (1) they affect how the impurity measures are computed, (2) they affect how an instance with missing value is distributed to the child nodes, (3) they affect how a test instance with missing value is classified.

Computing Impurity Measure: Consider once again the taxpayer classification problem depicted in Figure 3.7. Suppose we replace the attribute value for *Refund* in the last record from “No” to “?” (missing). How would this affect the gain ratio calculations? Initially, our data set contains 7 instances of class *Yes* and 3 instances of class *No*. If we consider the attribute *Refund* as the test condition, the instances can be partitioned in the following way:

	Evade= <i>Yes</i>	Evade= <i>No</i>	Total
<i>Refund</i> = <i>Yes</i>	0	3	3
<i>Refund</i> = <i>No</i>	2	4	6
<i>Refund</i> = ?	1	0	1
<i>Total</i>	3	7	10

We implicitly assume that an additional value is “reserved” for the missing attribute value. Since the *Refund* value for the last record is missing, it will not contribute directly to the entropy calculations of the child nodes. Before splitting the attributes, the entropy of the parent node is $-(3/10)\log_2(3/10) - (7/10)\log_2(7/10) = 0.8813$. After splitting, the entropy for each child node is:

$$Entropy(Refund = Yes) = -0\log_2(0) - (3/3)\log_2(3/3) = 0 \quad (3.14)$$

$$\begin{aligned} Entropy(Refund = No) &= -(2/6)\log_2(2/6) - (4/6)\log_2(4/6) \\ &= 0.9183 \end{aligned} \quad (3.15)$$

The weighted average of entropy for the child nodes (excluding instances with missing value) is $0.3 * 0 + 0.6 * 0.9183 = 0.5510$. The formula for information gain has changed to:

$$\Delta_{gain} = g \times [Entropy(All) - Entropy(Children)]$$

where g is the fraction of instances whose attribute value for *Refund* is known, i.e., $g = 9/10 = 0.9$. Replacing all the terms into the above formula yields $\Delta_{gain} = 0.9 \times (0.8813 - 0.5510) = 0.3303$. Let us examine what happens if the *Refund* attribute for the last record is known to be “No”. The information gain for this case is $\Delta_{gain} = (0.8813 - 0.6897) = 0.1916$. This is an example where the information gain has improved because the missing attribute value somehow causes the child node for *Refund* = No to become purer. (Can you think of a counter-example where missing attribute values can hurt the information gain?) The split information is computed by treating “Missing” as another attribute value:

$$Split\ Info = -0.3 \log_2(0.3) - 0.6 \log_2(0.6) - 0.1 \log_2(0.1) = 1.2955$$

Therefore, the gain ratio for *Refund* is $0.3303/1.2955 = 0.2550$.

Distribution of Instances. Assuming that *Refund* still has the highest gain ratio among the three attributes (actual calculations would reveal that the attribute *TaxableIncome* has the highest gain ratio), it will be selected as the test condition for the root node. Next, we need to distribute the instances down the branches of the root node. For instances that have known values for *Refund*, we can distribute them to one of the branches in the same manner as before. On the other hand, if the value for *Refund* is missing, we would distribute the instance to the (*Refund* = Yes) branch with a fractional weight of 3/9 and the (*Refund* = No) branch with a fractional weight of 6/9. After distributing the last record, the *Refund* = Yes branch has 1/3 instances of class Yes and 3 instances of class No. On the other hand, the *Refund* = No branch has $2 + 2/3 = 2.67$ instances of class Yes and 4 instances of class No. Suppose we need to split the instances further according to the *Marital Status* attribute. This attribute has three possible values, *Single*, *Married*, and *Divorced*. Of the 6.67 instances that reaches the *Marital Status* node, their class distributions are shown below:

	<i>Married</i>	<i>Single</i>	<i>Divorced</i>	Total
<i>Class</i> = No	3	1	0	4
<i>Class</i> = Yes	0.67	1	1	2.67
Total	3.67	2	1	6.67

We can compute the gain ratio for this attribute in the same manner as before, except that each branch has a fractional count of instances.

Table 3.3. Decision Tree Algorithm.

```

Tree-Growth( $E$ : set of training instances,  $F$ : set of features)
1. if stopping_cond( $E, F$ ) = true then
    1a. leaf = createNode()
    1b. leaf.label = getMajorityClass( $E$ )
    1c. return leaf
2. else
    2a. root = createNode()
    2b. root.test_cond = find_best_split( $E, F$ )
    2c. let  $V = \{v | v \text{ is a possible outcome of } \textit{root.test\_cond} \}$ 
    2d. for each  $v \in V$  do
    2e.    $E_v = \{e | \textit{root.test\_cond}(e) = v \text{ and } e \in E\}$ 
    2f.   child = TreeGrowth( $E_v, F$ )
    2g.   add child as descendent of root and label the edge  $\textit{root} \rightarrow \textit{child}$  as  $v$ 
    2h. end
3. end
4. return root

```

Classifying Instances. Suppose an unlabeled instance has the following attributes: (*Refund* = *No*, *Marital Status* = ?, *Taxable Income* = 80), where the value for *Marital Status* is missing. Assuming that the final tree is similar to the one shown in Figure 3.7, the instance will first move to the right child of the *Refund* node. The next test condition applied to the instance is (*Marital Status*?). During the tree construction step described in the previous paragraph, the left branch {*Single*, *Divorced*} has 3 instances while the right branch {*Married*} has 3.67 instances (see the table above). Therefore, the instance will be assigned to the left child with a fractional weight of 3/6.67 and the right child with a fractional weight of 3.67/6.67. These fractions will propagate down the tree until they reach the leaf nodes, where the class label is determined by combining the predicted probabilities of the leaf nodes.

3.3.8 Algorithm for Decision Tree Construction

An algorithm for growing decision trees is summarized in Table 3.3. The algorithm works by recursively selecting the best feature to split the data (Step 2b) and creating internal nodes (Steps 2f and 2g) until a stopping criterion is met (Step 1).

The details of the above decision tree algorithm are explained below:

1. The *createNode*() function expands the decision tree by creating a new node. Each node in the tree can either have a test condition, denoted as *node.test_cond*, or a class label, denoted as *node.label*.
2. The *find_best_split*() function determines which feature should be selected as the test condition for splitting the training instances. As previously noted,

determining the test condition is equivalent to deciding which region of the input space should be partitioned next. In recent years, various forms of splitting functions have been proposed including entropy, the Gini index, and the χ^2 measure.

3. The `getMajorityClass()` function takes the current set of labeled instances and decides on the most likely class label to be assigned to the node. A typical implementation of this function would take the majority class label of the given instances. However, for classification schemes that are cost-sensitive, the cost metric can also be incorporated into this function.
4. Finally, the stopping condition function, *stopping_cond()*, is used to prevent the tree from growing any further. This condition is met when all the instances E that reach the current node have the same class label, or when the instances have satisfied some tree pruning criterion. Examples of the latter include the number of instances have fallen below some minimum threshold or the class distribution of the instances have satisfied some skewness criterion.

3.3.9 Characteristics of Decision Tree Induction

Some of the main characteristics of decision tree induction are summarized below:

1. Decision tree induction is a non-parametric approach for building classification models because it does not require making any *a priori* assumption about the probability distributions of classes and attributes.
2. Decision trees have an expressive representation for learning any discrete-valued function. However, they do not generalize well to certain types of Boolean functions such as parity functions, whose values are 1 when there is an even number of input (Boolean) variables with the value *True*, and 0 when there is an odd number of input (Boolean) variables with the value *True*. Such functions require the full-sized tree (having 2^d nodes, where d is the number of Boolean attributes) to be grown.
3. Finding an optimal decision tree is an NP-complete problem. Many decision tree algorithms employ a heuristic-based search strategy to guide their search in the vast hypothesis space. The bias introduced by the algorithm as a result of its search strategy is known as *preference bias* in the machine learning literature.
4. Most decision tree induction algorithms use a greedy, top-down, recursive partitioning approach for growing the tree. Nevertheless, algorithms that use bottom-up or bi-directional tree-growing search strategies do exist.
5. Decision tree algorithms are quite robust to noise, particularly when the pre-pruning or post-pruning methods described in the previous section are used.

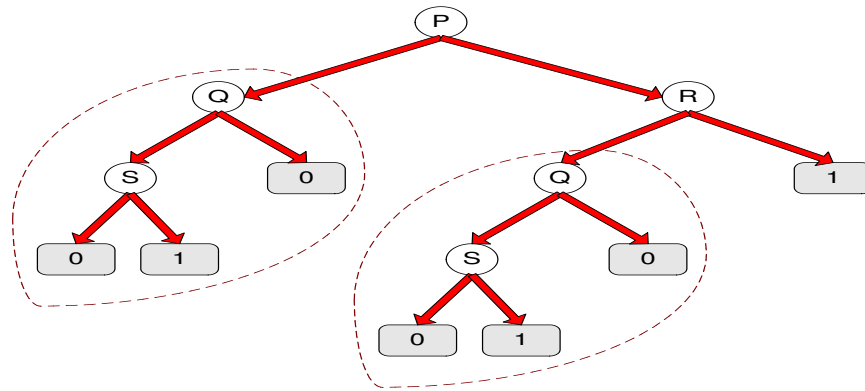


Figure 3.18. Replication problem of decision trees.

6. One major drawback of decision tree induction is the *data fragmentation* problem. In a top-down, recursive partitioning approach, the number of instances becomes smaller as you traverse down the tree. At the leaf nodes, the number of instances could be too small to make any statistically significant decision about the class representation of the instances. One possible solution is to disallow further splitting when the number of instances is too low. In addition, instead of having a decision tree that produces crisp decisions at the leaf nodes, the leaf nodes can be used to provide probability estimates about the class representation of a given instance. In addition, a *soft splitting criterion* can be used during decision tree construction, where an instance can be assigned to different partitions with different probabilities.
7. A subtree can be replicated multiple times at different branches of a decision tree, as illustrated in Figure 3.18. This makes the decision tree more complicated than necessary and more difficult to understand. This problem arises as a result of decision tree representations that rely on single attribute test at each node. Since many 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, which leads to the subtree replication problem.
8. Studies have shown that redundant attributes do not adversely affect the accuracy of decision trees [96]. However, data sets with redundant attributes do tend to produce trees that are larger than necessary.
9. The test conditions described in this section involve only one attribute at a time. Such a test condition will partition the attribute space into rectangular regions whose decision boundaries are parallel to the coordinate axes. Figure 3.19 illustrates an example of a data set that cannot be partitioned optimally by a decision tree algorithm that uses a greedy approach and test conditions involving only a single attribute at a time.

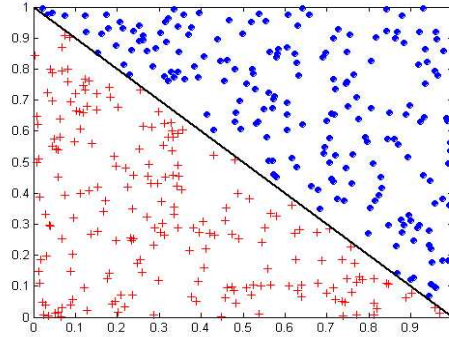


Figure 3.19. Example of data set that cannot be partitioned optimally using test conditions involving single attributes.

Oblique decision trees are decision trees that allow for test conditions involving more than one attribute. For example, the data set given in Figure 3.19 can be easily represented by an oblique decision tree containing a single node with test condition

$$x + y < 1.$$

Although such techniques can produce smaller trees, finding the optimal split for a given node is an expensive operation. Another way to induce decision trees that produce non-rectangular regions is to use a technique known as *constructive induction*. The central idea behind constructive induction is to create new attributes representing an arithmetic or logical combination of the existing attributes. The new attributes would provide a better discrimination of the classes and are treated in the same way as other attributes during tree construction. Unlike the oblique decision tree approach, constructive induction is less expensive because it identifies all the relevant combinations of attributes once, prior to decision tree construction. In contrast, oblique decision trees have to determine the right combination dynamically, each time when a tree node is to be expanded. Constructive induction may introduce attribute redundancy in the data since the new attribute is a combination of several existing attributes.

10. Recent studies have shown that the choice of impurity measure has little effect on the performance of decision tree induction algorithms. Instead, performance depends more on the techniques used for pruning the decision tree. This is because most impurity measures are quite consistent with each other, while the final size of the tree depends primarily on the pruning strategy employed by the learning algorithm.

3.4 Rule-based classifiers

A rule-based classifier is a technique for classifying instances by using a collection of “if ... then...” rules. The rules are represented in a disjunctive normal form, $R = (r_1 \vee r_2 \vee \dots \vee r_k)$, where R is known as the *rule set* and r_i 's are the classification rules or *disjuncts*. Each classification rule can be expressed in the following way:

$$r_i : (Condition_i) \longrightarrow y \quad (3.16)$$

The left-hand side of the rule is called the *rule antecedent* or *pre-condition*. It contains a conjunction of attribute tests:

$$Condition_i = (A_1 \text{ op } v_1) \wedge (A_2 \text{ op } v_2) \wedge \dots \wedge (A_k \text{ op } v_k), \quad (3.17)$$

where (A_j, v_j) is an attribute-value pair and *op* is a logical operator selected from the set $\{=, \neq, <, >, \leq, \geq\}$. Each attribute test $(A_j \text{ op } v_j)$ is known as a *conjunct*. The right hand side of the rule is called the *rule consequent* and contains the class label predicted by the rule.

A classification rule can be applied in the following way: if the attributes of an instance satisfy the pre-condition of a rule, then the instance is assigned to the class designated by the rule consequent. For example, the rule:

$$r : (Age < 35) \wedge (Marital \text{ Status} = Married) \longrightarrow Evade = No \quad (3.18)$$

suggests that every married taxpayer whose age is less than 35 years old will not likely evade paying taxes.

3.4.1 How Rule-Based Classifier Works?

This section describes how to apply a rule-based classifier to an unlabeled instance. We say that a rule r *covers* an instance \mathbf{x} if the attributes of the instance match the pre-condition of r . Put another way, a rule is said to be *fired* or *triggered* if the attributes of the instance satisfy the pre-condition of the rule.

Example 4 Consider the following two unlabeled instances:

$$\begin{aligned} \mathbf{x}_1 : & (Age = 29, Marital \text{ Status} = Married, Refund = No) \\ \mathbf{x}_2 : & (Age = 28, Marital \text{ Status} = Single, Refund = Yes) \end{aligned}$$

The rule r given in equation (3.18) covers \mathbf{x}_1 because the taxpayer is married and is less than 35 years old. On the other hand, r does not cover \mathbf{x}_2 because the taxpayer is not married; thus, violating one of the attribute tests for r .

To illustrate how a rule-based classifier works, consider the rule set shown below.

Rule set $R = (r_1 \vee r_2 \vee r_3)$

$r_1 : (\text{Blood Type} = \text{Cold}) \longrightarrow \text{Non-mammal}$

$r_2 : (\text{Blood Type} = \text{Warm}) \wedge (\text{Gives Birth} = \text{Yes}) \longrightarrow \text{Mammal}$

$r_3 : (\text{Blood Type} = \text{Warm}) \wedge (\text{Gives Birth} = \text{No}) \longrightarrow \text{Non-mammal}$

An unlabeled instance is classified by the rule it triggers. For example, a gila monster, which is a cold-blooded animal, triggers only the first rule and is therefore classified as a non-mammal. Such a rule-based classifier has the following properties:

1. It contains *mutually exclusive* rules, *i.e.*, rules that are independent of each other. As a result, every instance is covered by at most one rule.
2. It has an *exhaustive* coverage, *i.e.*, the rule set covers all possible combinations of attribute values. As a result, every instance is covered by at least one rule.

Together, these properties ensure that each test instance is classified by exactly one rule. There are many situations in which the rule set may not be mutually exclusive nor exhaustive. This may happen if the classifier attempts to simplify the rules by merging some of them. For example, if r_1 and r_3 are merged and replaced by the following rule:

$$r'_1 : (\text{Gives Birth} = \text{No}) \longrightarrow \text{Non-mammal},$$

then the classifier is no longer exhaustive. We will illustrate more examples of rule simplification in Section 3.4.5.

If the rule set is not exhaustive, then a *default* rule $r_d : () \longrightarrow y_d$ can be added to cover the remaining instances. The default rule has an empty antecedent and is applicable when all other rules have failed. y_d is known as the *default class* and is often assigned to the majority class among the remaining training instances.

If the rule set is not mutually exclusive, then an instance can be covered by multiple rules, some of which may predict conflicting class labels. Below, we describe several approaches to address this problem.

Strict enforcement of mutual exclusiveness. The simplest approach is to avoid generating rules that have overlapping coverage with previously selected rules. However, since most of the selected rules may not be 100% accurate, there are certain “regions” in the input space for which their predictions could go wrong. (Much like decision trees, rules are often generalized to prevent overfitting. Generalized rules tend to be shorter and may not have perfect accuracy on the training and test sets.) Even though there may exist better rules for such regions, these rules are not selected because their coverage overlap with the existing rules. Therefore, this approach can lead to suboptimal classifiers.

Ordered Rules. In this method, the rules are rank ordered according to their priority. The ordered rule set is known as a *decision list* and is denoted as $R^{(o)} = (r_1 \vee r_2 \vee \dots \vee r_k)$, where $\forall i : r_i \succ r_{i+1}$ (*i.e.*, r_i precedes r_{i+1} in the rule set) and r_k is the default rule. When a test instance is presented to the

		categorical	categorical	continuous	class
<i>Tid</i>	Refund	Marital Status	Taxable Income	Evade	
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	

$r_1: (\text{Refund}=\text{No}) \ \& \ (\text{Marital Status}=\text{Single})$
 $\ \& \ (\text{Taxable Income} > 80\text{K}) \implies \text{Yes}$

$r_2: (\text{Refund}=\text{No}) \ \& \ (\text{Marital Status}=\text{Divorced})$
 $\ \& \ (\text{Taxable Income} > 80\text{K}) \implies \text{Yes}$

Default:: () $\implies \text{No}$

Figure 3.20. Example of a Rule-Based Classifier for the taxpayer classification problem.

classifier, it is assigned to the class label of the highest ranked rule it has triggered. If none of the rules fires, the instance is assigned to the default class. For example, Figure 3.20 shows a rule-based classifier for the taxpayer classification problem described in the earlier sections. The classifier contains two rules for predicting the **Yes** class and a default rule that predicts the **No** class. Given a test instance, the rule r_1 is applied first. If the taxpayer satisfies the pre-condition of r_1 , he or she is classified as **Yes**. Otherwise, the rule r_2 is applied to the taxpayer. If this rule does not fire, then the taxpayer is classified by the default rule as **No**. The advantage of an ordered rule set is that one may not have to apply every rule to the test instance. The first rule that fires is used for doing the prediction. The drawback of this method is that rules are more difficult to interpret as each rule implicitly assumes the negation of its preceding rules. For example, given the following ordered rule set:

Rule set $R^{(o)} = (r_1 \vee r_2)$

$r_1: (\text{Marital Status} = \text{Single}) \wedge (\text{Refund} = \text{No}) \longrightarrow (\text{Evade} = \text{No})$

$r_2: (\text{Age} < 35) \longrightarrow (\text{Evade} = \text{No})$

The second rule should be interpreted as:

$(\text{Age} < 35) \wedge [(\text{Marital Status} \neq \text{Single}) \vee (\text{Refund} \neq \text{No})] \longrightarrow (\text{Evade} = \text{No}),$

where we have used De Morgan's theorem $(\overline{A \wedge B} = \overline{A} \vee \overline{B})$ to simplify the rule expression.

Unordered Rules. This method allows an instance to trigger multiple classification rules and considers the consequent of each triggered rule as a vote for that particular class. The votes are then combined to obtain the final predicted

class. In most cases, the final prediction is given by the class that receives the highest number of votes. Alternatively, we can apply a voting scheme that weights the vote for each predicted class by the accuracy of the rule. The advantage of the unordered method is that rules no longer have to be sorted or generated in certain order. The drawback of this method is that every rule in the rule set must be applied to obtain the class label of a test instance. This will increase the computation time during classification; thus affecting prediction tasks that require real-time responses.

3.4.2 How to Build a Rule-Based Classifier?

To build a rule-based classifier, we need to extract rules that identify key relationships between attributes of a data set and the class label. There are two broad classes of methods for extracting classification rules:

1. **Direct Methods** extract classification rules directly from data. These methods often apply a *sequential covering* algorithm for creating the rule set. Each rule is grown in a greedy fashion, optimizing certain objective function. Once a rule has been extracted, the algorithm eliminates training instances covered by the rule and repeats the procedure for growing the next rule. A detailed discussion of the sequential covering algorithm is presented in Section 3.4.4.
2. **Indirect Methods** extract classification rules from other classification models, such as decision trees and neural networks. Rules are extracted to produce more interpretable models. A detailed discussion of methods for extracting classification rules from decision trees is presented in Section 3.4.5.

There are many measures available to evaluate the quality of a classification rule. Two such measures are *coverage* and *accuracy*. Given a set of instances D and a classification rule $r : A \longrightarrow y$, the coverage of the rule is defined as the fraction of instances in D that are covered by r . On the other hand, accuracy or *confidence factor* is defined as the fraction of instances covered by r whose class labels are equal to y . The definitions of these measures are summarized below.

$$\begin{aligned}\text{Coverage}(r) &= \frac{|A|}{|D|} \\ \text{Accuracy}(r) &= \frac{|r|}{|A|}\end{aligned}\tag{3.19}$$

where $|A|$ is the total number of instances that satisfy the rule antecedent, $|r|$ is the number of instances that satisfy both the antecedent and consequent of r , and $|D|$ is the total number of instances. (Note that for sequential covering algorithms, $|D|$ is not a constant factor. In fact, it will decrease as the rule becomes longer.)

Example 5 Consider the data set shown in Figure 3.7. The rule

$$\text{Marital Status} = \text{Single} \longrightarrow \text{Evade} = \text{No}$$

has a coverage of 40% since four out of the ten audited taxpayers are single. The accuracy of this rule is 50% as two of the four single taxpayers have class labels equal to *No*.

3.4.3 Rule Ordering

Once the classification rules have been extracted, they can be rank ordered according to their priority. Rules generated using direct methods can be ranked in the order they are created while those generated using indirect methods can be sorted based on a variety of objective measures such as accuracy and entropy.

In general, the extracted rules can be ordered on a rule-by-rule basis or a class-by-class basis.

1. In the *rule-based ordering scheme*, individual rules are ranked based on their quality. The better is the quality, the higher is the rank. The advantage of this scheme is that each unlabeled instance is classified by the best rule it has triggered. The disadvantage of this scheme is that ordered rules are more difficult to interpret since each lower priority rule implicitly assumes the negation of its higher priority rules.
2. In the *class-based ordering scheme*, rules that belong to the same class appear together in the rule set R . The rules are collectively sorted on the basis of their class information. The advantage of this method is that rules within each class do not have to be ordered, and thus, are easier to interpret. The disadvantage of this method is that the classifier is biased towards higher ranked classes. The class-based ordering scheme can be implemented in two ways. For direct methods, the class-based ordering scheme is implemented by generating rules one-class-at-a-time. For indirect methods, the class-based ordering scheme can be implemented by computing the total description length for each class and then sorting the classes according to this measure. The class that has the lowest description length will be ranked highest, followed by the class with the next lowest description length, and so on.

The difference between rule-based and class-based ordering schemes is shown in Figure 3.21. For the class-based ordering scheme, notice that the rules for class *No* appear next to each other and are ranked higher than the rule for class *Yes*.

3.4.4 Direct Methods for Rule Extraction

This section presents techniques for extracting classification rules directly from data. We begin with a description of the sequential covering algorithm followed by discussion about two specific implementations of direct methods – Holte’s 1R and RIPPER algorithms.

Sequential Covering Algorithm

Many rule-based classifiers employ a *sequential covering* algorithm to induce classification rules from data. A high-level description of this algorithm is shown in Table

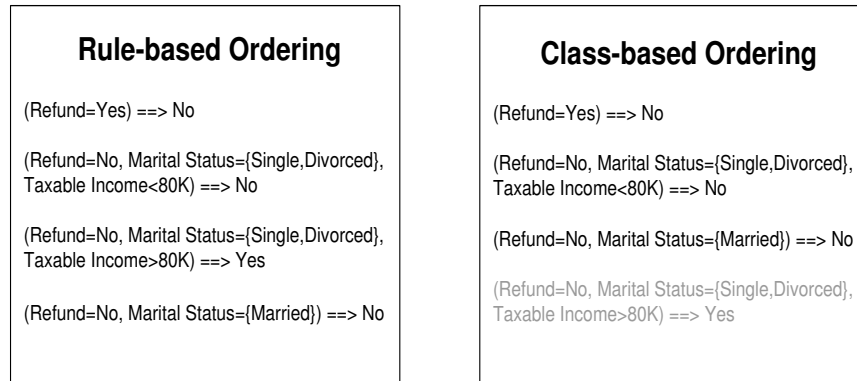


Figure 3.21. Difference between rule-based and class-based ordering schemes.

3.4.

Initially, the algorithm starts from an empty rule set R . Next, the Learn-One-Rule function is used to find the best rule that covers the current set of instances. Once such a rule is found, the algorithm eliminates instances covered by the rule and adds the rule to R . The rule growing and instance elimination steps are repeated until a stopping criterion is met.

Figure 3.22 demonstrates an example of how the sequential covering algorithm works. The data set contains instances that belong to two separate classes, one designated as the positive class while the other is the negative class. Initially, the rule that corresponds to Region $R1$ in Figure 3.22(b) is extracted first because it contains a large number of examples that belong to the same class. Instances covered by the rule are then removed and the algorithm continues to look for the next best rules to extract ($R2$ and subsequently, $R3$).

The algorithm presented in Table 3.4 can be modified to take into account class ordering. To do this, we can place lines 2-5 of the code inside another **for** loop, as shown below:

Table 3.4. The Sequential Covering Algorithm.

Sequential Covering Algorithm (E : training examples, A : set of attributes)

1. Let $R = \{\}$ be the initial rule set.
2. while stopping criterion is not met
3. $r \leftarrow \text{Learn-One-Rule}(E, A)$
4. Remove instances from E that are covered by r
5. Add r to rule set: $R = R \vee r$

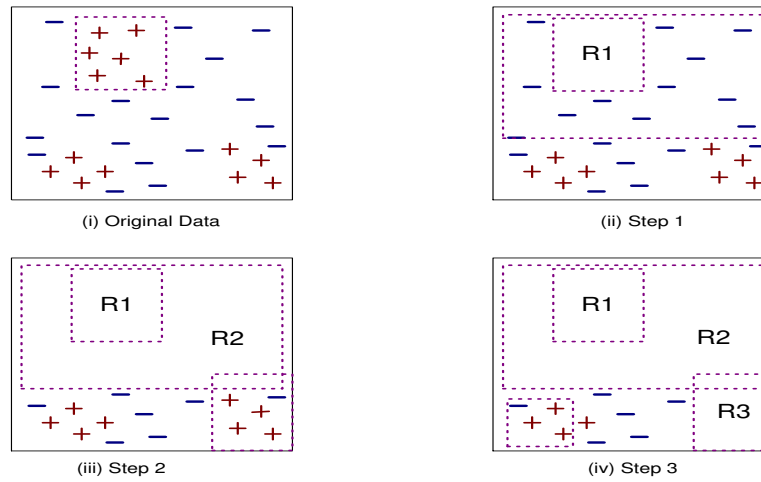


Figure 3.22. An illustrative example of the sequential covering algorithm.

- 1b. for each $class \in AllClasses$
2. while stopping criterion is not met
3. $r \leftarrow \text{Learn-One-Rule}(E, A, class)$
4. ...

In addition, the Learn-One-Rule function is modified to extract rules for a particular class. The details of the Learn-One-Rule function and instance elimination step are described next.

Learn-One-Rule

The objective of this function is to extract the best rule that covers the current set of training instances. There are several issues to consider when growing a rule:

1. What is the strategy used for rule growing?
2. What is the evaluation criteria for rule growing?
3. What is the stopping criteria for rule growing?
4. What is the pruning criteria, if there is one, for generalizing the rule?

We will discuss each of the key issues above in the remainder of this section.

Rule Growing Strategy. In the Learn-One-Rule function, rules are grown in an iterative manner. There are two common strategies for growing a rule: (1) general-to-specific approach, or (2) specific-to-general approach. In the general-to-specific approach, it is initially assumed that the best rule is the empty rule, $r : \{\} \rightarrow y$, where y is the majority class of the instances. It then iteratively adds new conjuncts to the left-hand side of the rule until the stopping criterion is met. In the specific-to-general approach, a positive instance is chosen as the initial seed for a rule. The

function keeps refining this rule by generalizing the conjuncts until the stopping criterion is met. A problem with this approach is that there are many initial points to start from, unlike the general-to-specific approach. One way to address this problem is to use several initial seeds as candidate rules, perform a specific-to-general search to generalize each rule, and then select the best candidate rule. In both general-to-specific and specific-to-general approaches, the rules are often grown in a greedy fashion. A well-known limitation of a greedy search algorithm is that it may get trapped in a local minimum, thus producing a suboptimal rule. Alternatively, the Learn-One-Rule function can perform a beam search, where, instead of keeping only the best rule, the function also maintains k of the best candidate rules. Each candidate will be grown separately by adding more conjuncts or generalizing the conjuncts. The quality of each candidate is then evaluated and k of the best new candidates are selected for the next iteration.

Rule Evaluation and Stopping Criteria. During rule growing, a rule evaluation metric is used to determine the quality of each candidate rule. Recall that there are two variants to the Learn-One-Rule function: (1) where the target class is specified, and (2) where the target class is not specified. If the target class is specified, then accuracy is a natural choice for the rule evaluation metric. However, one major drawback of accuracy is that it tends to prefer specific (longer) rules over general (shorter) ones. An ideal metric should take into account both the accuracy and coverage of the rule. Another popular metric is entropy $= -\sum_i p_i \log p_i$, where p_i is the fraction of instances covered by the rule that belong to class i . This metric is often used by Learn-One-Rule functions for which the target class is not specified. For example, the entropy is the same for both $p_0 = 1, p_1 = 0$ and $p_1 = 0, p_1 = 1$. As long as most of the instances belong to one particular class, its entropy will be maximized. Much like accuracy, entropy does not pay attention to the coverage of the rule. For example, given the following pair of rules:

r_1 : covers 500 positive examples and 1 negative example,
 r_2 : covers 2 positive examples and 0 negative example.

The entropy for r_1 is smaller than r_2 , even though r_1 appears to be the better rule. Thus, if entropy is chosen as the rule evaluation metric, it must be complemented by a statistical significance test to ensure that the extracted rule has sufficient coverage. Other rule evaluation metrics include:

$$\text{Laplace} = \frac{n_c + 1}{n + k} \quad (3.20)$$

$$\text{m-estimate} = \frac{n_c + kp}{n + k} \quad (3.21)$$

where n is the number of instances covered by the rule, n_c is the number of instances in the predicted class c covered by the rule, k is the total number of classes, and p is the prior probability (selected by the user). Note that Laplace measure is equivalent

to m-estimate if $p = 1/k$. The advantage of this measure is that it implicitly accounts for the coverage of the rule. For example, the Laplace measure for rule r_1 is higher than r_2 , even though the latter rule has higher accuracy and entropy.

A typical condition for terminating the rule growing process is to compare the evaluation metric of the previous candidate rule to the newly grown rule. If there is no significant performance improvement, then the new rule should be discarded and the Learn-One-Rule function will terminate. (A special case of this condition occurs when all remaining instances belong to the same class. Any further rule growing will not improve the performance of this rule, thus terminating the function.)

Rule Pruning. Each extracted rule can be pruned to improve their ability to generalize beyond the training instances. The pruning strategies described in Section 3.3.6 for decision trees are also applicable to classification rules. For example, pruning can be done by removing one of the conjuncts in the rule and then testing it against a validation set to determine whether the rule's generalization ability has improved.

Instance Elimination

After extracting a rule, why is it necessary for the sequential covering algorithm to eliminate instances covered by the rule? If we do not eliminate the instances, then the next rule extracted by the function will be exactly identical to the previous rule. Therefore, instance elimination can prevent the same rule from being generated again.

The example shown in Figure 3.22 uses an instance elimination scheme that removes both positive and negative instances from the data set. The positive instances correspond to those that have the same class label as the consequent of the latest extracted rule. But is this the only scheme available or can we remove only the positive (or negative) instances? To answer this question, we need to understand how instance elimination affects subsequent rule growing process.

Positive instances must be removed after each rule is extracted. This will ensure that the next rule generated by the Learn-One-Rule function is different than its previous rule. For negative instances, some rule-based classifiers (e.g., Ripper) would remove them prior to generating the next rule while others (e.g., CN2 and AR algorithms) would keep them. We illustrate the effect of removing negative instances in the example below.

Example 6 Consider the situation shown in Figure 3.23. For brevity, we assume that the classifier uses rule accuracy as its evaluation metric. The rule $R1$ is initially extracted because it has the highest accuracy.

$$Accuracy(R1) = \frac{12}{15} = 80\%.$$

In the next iteration, suppose the algorithm has two possible choices of candidate

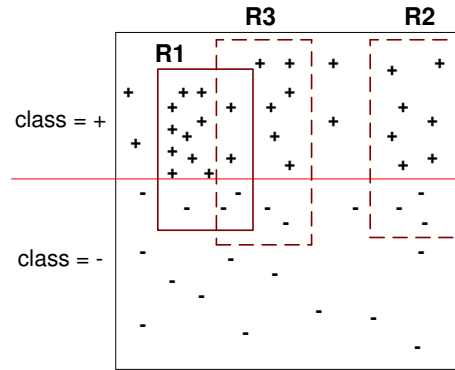


Figure 3.23. Elimination of training instances by the sequential covering algorithm. $R1$, $R2$, and $R3$ represent regions covered by three different rules.

rules: $R2$ or $R3$. The accuracy for $R2$ is

$$Accuracy(R2) = \frac{7}{10} = 70\%.$$

Since $R3$ overlaps with the previous rule $R1$, the accuracy for $R3$ depends on whether negative instances are removed after extracting $R1$. If the negative instances are removed, then

$$Accuracy(R3) = \frac{6}{8} = 75\%,$$

which is higher than the accuracy for $R2$. On the other hand, if the negative instances are kept, then

$$Accuracy(R3) = \frac{8}{12} = 66.7\%,$$

which is lower than the accuracy for $R2$. If the classifier eliminates negative instances, then it will prefer $R3$ over $R2$. On the other hand, if negative instances are kept, then it will prefer $R2$ over $R3$.

The above example illustrates the effect of removing negative instances on the accuracy of a rule. If the rules are ordered using the rule-based ordering scheme, then it makes more sense to prefer $R3$ over $R2$ since most of the errors committed by $R3$ were already accounted for by its preceding rule, $R1$. If the negative instances are not removed, then the accuracy of $R3$ would have been underestimated. Thus, classifiers with ordered rule-set should remove both positive and negative instances from the training data. Examples of rule-based classifiers that remove both positive and negative instances include Ripper and the (ordered) CN2 algorithms.

If the rules are ordered using a class-based ordering scheme, there are both advantages and disadvantages in removing negative instances. The advantage is to avoid missing high quality rules because their accuracies were underestimated. The disadvantage is that this will reduce the number of negative instances available to

learn rules for the negative class. Examples of rule-based classifiers that do not remove negative instances include AQR and the (unordered) CN2 algorithms.

Next, we present two specific implementations of rule-based classifiers that use the direct approach to extract classification rules.

Holte's 1R

Pareto Principle (80/20 Rule): A small number of causes is often responsible for a large percentage of the effect — Vilfredo Pareto (1848-1923)

Holte's 1R is a rule-based classifier that contains only one classification rule. In addition, the classification rule involves only a single attribute. 1R selects its attribute in the following way. First, each continuous attribute is discretized into several disjoint intervals. The interval must be wide enough to contain a sufficiently large number of instances. A new feature is then created for each discretized interval. The feature that produces the lowest misclassification error will be chosen as the pre-condition of the classification rule. The majority class of instances satisfying the pre-condition forms the rule consequent. If the data set contains missing values, 1R treats `MISSING` as another attribute value.

Despite its simplicity, Holte demonstrated that 1R performed almost as well as other rule-based classifiers for certain data sets that exhibit a strong one-to-one relationship between one of the attributes and the class label.

RIPPER Algorithm

RIPPER uses a greedy approach for creating its rule set R . For a 2-class problem, it chooses the majority class of training instances as the default class and then tries to learn specific rules for the minority class. For the multi-class problem, the classes are ordered according to their class prevalence, i.e., fraction of instances that belong to a particular class. Let the ordered classes be (y_1, y_2, \dots, y_c) , where y_1 is the least prevalent class and y_c is the most prevalent class. During the first iteration, y_1 is treated as the positive class while the rest of the classes form the negative class. RIPPER will attempt to build rules that can distinguish y_1 from other classes. After each rule is extracted, RIPPER eliminates both positive and negative instances covered by the rule. In the next iteration, RIPPER will extract rules for discriminating y_2 from the remaining classes, y_3, y_4, \dots, y_c . This process is repeated until the most prevalent class remains, which is assigned to the default class.

Growing a single rule. The Learn-One-Rule function for RIPPER starts from an empty rule, and keeps adding conjuncts until the stopping criteria is met. RIPPER uses FOIL's information gain to determine which conjunct should be added. This metric compares the performance of the rule before and after adding a new conjunct. Suppose the rule $r : A \longrightarrow y$ covers p_0 positive

instances and n_0 negative instances. After adding a new conjunct B , the rule $r' : A \wedge B \rightarrow y$ covers p_1 positive instances and n_1 negative instances. FOIL's information gain can be defined as follows:

$$\text{Foil's Information Gain} = t \times \left(\log_2 \frac{p_1}{p_1 + n_1} - \log_2 \frac{p_0}{p_0 + n_0} \right), \quad (3.22)$$

where t is the number of positive instances covered by both r and r' . For RIPPER, $t = p_1$ since r' is a specialization of r . RIPPER stops adding conjuncts when the rule no longer covers any negative instances. After a rule is grown, it is pruned immediately by using the incremental reduced error pruning technique. (Unlike traditional reduced error pruning techniques, which performs the pruning after the complete model has been generated, a rule is pruned before the entire rule set is generated. This is why the technique is called incremental reduced error pruning.) RIPPER uses the following measure for pruning: $(p - n)/(p + n)$, where p is the number of positive instances in the validation set covered by the rule and n is the number of negative instances in the validation set covered by the rule. Note that RIPPER prunes only a suffix of the pre-condition. For example, given a rule $ABCD \rightarrow y$, RIPPER will examine if D should be pruned first, followed by CD , BCD , etc. After pruning, the rule may cover some negative instances.

Building Rule Set. After a rule has been found, all the positive and negative instances covered by the rule are removed and the rule growing process continues until a stopping condition (based on the MDL principle) is met. Each time a new rule is added, the description length of the rule set is computed. The new description length is then compared against the smallest description length obtained so far. If the new description length is larger by a factor of d bits, then RIPPER stops adding new rules (d is typically chosen to be 64 bits). In addition, RIPPER will also stop adding rules if the new rule has an error rate greater than 50% on the validation set.

Optimization of Rule Set. Once the rule set has been established, RIPPER performs an additional rule optimization step to check whether a better rule set can be obtained by replacing or modifying some of the rules in the current rule set. For each rule r in the rule set R (starting from the highest priority rule), we consider two alternative rules for replacing it: (1) a replacement rule r_p , which is a completely new rule grown from an empty rule, and (2) a revised rule r_v , which is a modified rule grown starting from r . The minimum description length principle is used to determine whether a rule should be kept in the rule set or replaced by either a replacement or revised rule.

A summary of RIPPER and its comparison to three other rule-based classifiers are shown in Table 3.5.

Table 3.5. Comparison between various rule-based classifiers.

	RIPPER	CN2 (unordered)	CN2 (ordered)	AQR
Rule growing strategy	General-to-specific	General-to-specific	General-to-specific	General-to-specific (seeded by a positive example)
Evaluation Metric	FOIL's Info gain	Laplace	Entropy with likelihood ratio test	Number of positive true positives
Stopping condition for rule growing	All examples belong to the same class	No performance gain	No performance gain	Rules cover only positive class
Rule Pruning	incremental reduced error pruning	None	None	None
Instance Elimination	Positive and Negative	Positive only	Positive only	Positive and Negative
Stopping condition for adding rules	Error > 50% or based on MDL	No performance Gain	No performance Gain	All positive examples are covered
Pruning rule set	None but can replace/modify rules	Statistical tests	None	None
Search strategy	greedy	beam search	beam search	beam search

3.4.5 Indirect Methods for Rule Extraction

This section is still being modified

In this section, we describe a method for generating rules from other classification models. Specifically, we focus our attention on extracting rules automatically from a decision tree using the c4.5rules algorithm. For decision trees, each path from the root node to a leaf node can be expressed as a classification rule. The test conditions of each non-terminal node encountered along the path are combined using the logical AND operator to form the antecedent of the rule, while the class label of the leaf node becomes the rule consequent. For example, Figure 3.24 illustrates a decision tree with five leaf nodes, along with the five corresponding classification rules representing the tree.

Large decision trees can be overly complex, which makes it difficult to interpret. Rules can be generated to simplify the tree, as illustrated in the example below.

Example 7 Consider the five classification rules shown in Figure 3.24. The rule set is exhaustive and contains mutually exclusive rules. Three of the rules are used to predict the positive class.

$$\begin{aligned}
 r2 : (P = No) \wedge (Q = Yes) &\longrightarrow + \\
 r3 : (P = Yes) \wedge (R = No) &\longrightarrow + \\
 r5 : (P = Yes) \wedge (R = Yes) \wedge (Q = Yes) &\longrightarrow +
 \end{aligned}$$

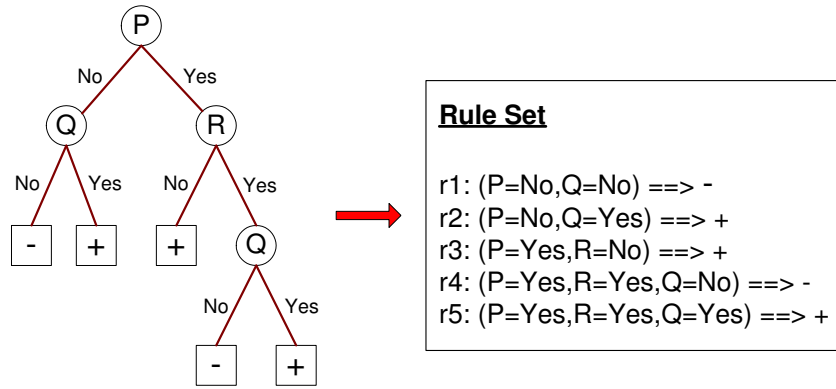


Figure 3.24. Converting a decision tree into classification rules.

These rules can be further simplified as follows. Observe that the class label is always positive when the value of Q is known to be Yes. This observation suggests that the rules can be simplified into the following pair of rules, $r2' : (Q = \text{Yes}) \rightarrow +$ and $r3 : (P = \text{Yes}) \wedge (R = \text{No}) \rightarrow +$, where the second rule is retained to cover the remaining case for the positive class. Both rules are no longer mutually exclusive but are less complex, and thus, easier to interpret than the initial rules.

C4.5rules creates its initial set of classification rules from the unpruned decision tree produced by the C4.5 algorithm. We describe the rule extraction step below.

Creating an initial set of rules. Each classification rule corresponds to one of the leaf nodes in the decision tree. Hence, the conjuncts of the rules represent the test conditions and outcomes encountered while traversing from the root node to a leaf node of the decision tree. Next, for each rule $r : A \rightarrow y$, we consider an alternative rule, $r' : A' \rightarrow y$ where A' is obtained by removing one of the conjuncts in A . A pessimistic estimate of the generalization error for all r 's are computed and compared against the generalization error rate for r . If one of the r 's has a lower error rate, we choose that particular rule and delete the corresponding condition. This step will be repeated until we can no longer improve upon the generalization error. The pruning step is applied to each classification rule derived from the decision tree. After pruning, some of the rules may be identical and the duplicate rules are removed.

Ordering the Rules. Instead of ordering the rules, c4.5rules orders the rule subsets, where each subset contains rules belonging to the same class (class-based ordering). The description length for each subset is computed, and the subsets are then ordered in increasing order of the description length (i.e., the subset that has the smallest description length will precede the other subsets.) Note that the description length definition was modified to allow the user to weight model complexity against the amount of misclassification error of the classifier. Specifically, the description length is given as $L_{\text{exception}} + g \times L_{\text{model}}$, where $L(\cdot)$ is the description length and g is a parameter (whose default value

is 0.5). The extra parameter was introduced by Quinlan to reduce the model complexity due to the presence of redundant attributes.

For example, Figure 3.25 illustrates the decision tree and classification rules obtained for the data set given in Table 3.6.

Table 3.6. The vertebrate data set.

Name	Give birth	Lay eggs	Can fly	Live in water	Have legs	Class
human	yes	no	no	no	yes	mammals
python	no	yes	no	no	no	reptiles
salmon	no	yes	no	yes	no	fishes
whale	yes	no	no	yes	no	mammals
frog	no	yes	no	sometimes	yes	amphibians
komodo dragon	no	yes	no	no	yes	reptiles
bat	yes	no	yes	no	yes	mammals
pigeon	no	yes	yes	no	yes	birds
cat	yes	no	no	no	yes	mammals
leopard shark	yes	no	no	yes	no	fishes
turtle	no	yes	no	sometimes	yes	reptiles
penguin	no	yes	no	sometimes	yes	birds
porcupine	yes	no	no	no	yes	mammals
eel	no	yes	no	yes	no	fishes
salamander	no	yes	no	sometimes	yes	amphibians
gila monster	no	yes	no	no	yes	reptiles
platypus	no	yes	no	no	yes	mammals
owl	no	yes	yes	no	yes	birds
dolphin	yes	no	no	yes	no	mammals
eagle	no	yes	yes	no	yes	birds.

3.5 Nearest-neighbor classifiers

This section is still being modified

So far, the classification framework presented in this chapter involves a two-step process: (1) an inductive step for constructing classification models from data, and (2) a deductive step for applying the derived model to previously unseen instances (see Figure 3.2). For decision tree induction and rule-based learning systems, the models are constructed immediately after the training set is provided. Such techniques are known as *eager learners* because they intend to learn the model as soon as possible, once the training data is available.

An opposite strategy would be to delay the process of generalizing the training data until it is needed to classify the unseen instances. Techniques that employ such strategy are known as *lazy learners*. An example of a lazy learner is the *Rote classifier*, which memorizes the entire training data and performs classification only

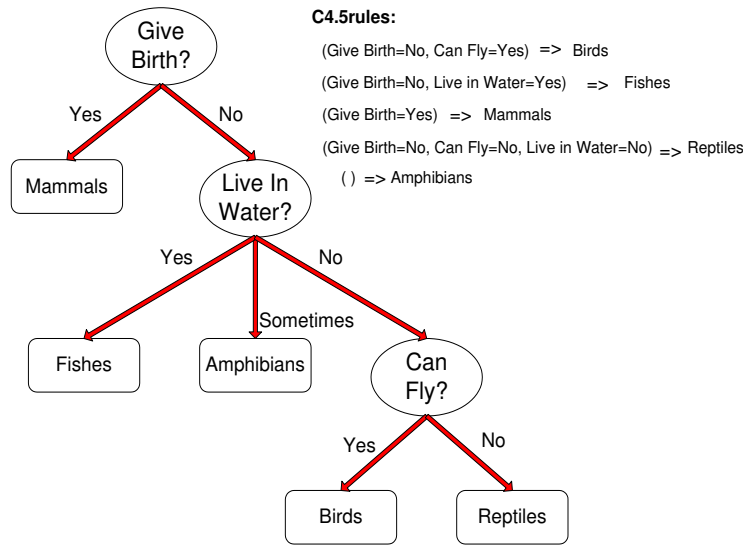


Figure 3.25. Results of C4.5 and C4.5rules algorithms.

if the attributes of a test instance matches one of the training examples exactly. The strict matching condition of a Rote classifier may cause many of the test instances to remain unclassified.

One way to make this approach more flexible is to find all training examples that are relatively similar to the attributes of the test instance. Such examples are known as the nearest neighbors of the test instance. The test instance can then be classified according to the class labels of its neighbors. This is the central idea behind the *nearest-neighbor classification* scheme, which is useful for classifying data sets with continuous attributes. The justification for nearest neighbor classification is best exemplified by the following saying

“If it walks like a duck, quacks like a duck, and looks like a duck, then it’s probably a duck.”

A nearest neighbor classifier represents each instance as a data point embedded in a d -dimensional space, where d is the number of continuous attributes. Given a test instance, we can compute its distance to the rest of the data points in the training set by using the standard Euclidean distance measure.

$$\text{Euclidean distance, } d(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=1}^d (p_i - q_i)^2} \quad (3.23)$$

As an example, let $\mathbf{p} = (1.5, 2.0)$ and $\mathbf{q} = (3.5, 2.5)$ be two data points, where (x_1, x_2) denotes the coordinate of the data point in a two-dimensional space. Using Equation (3.23), we can compute the distance between the two points as

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{(3.5 - 1.5)^2 + (2.5 - 2.0)^2} = 2.062$$

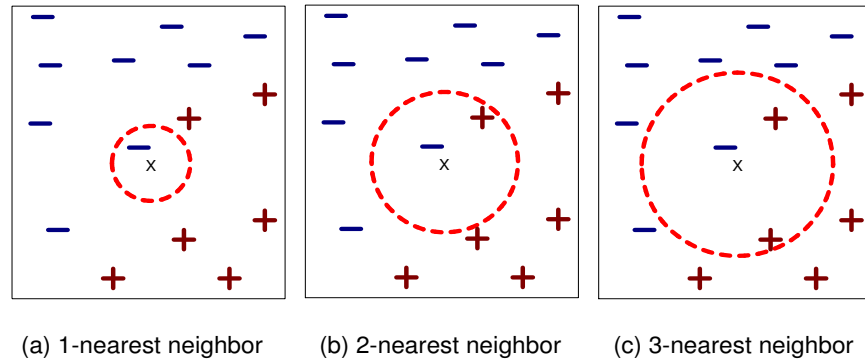


Figure 3.26. The 1-, 2- and 3-nearest neighbors of an instance.

The k -nearest neighbors of an instance z are defined as the data points having the k smallest distance to z . Figure 3.26 illustrates an example of the 1-, 2- and 3-nearest neighbors of an unknown instance \times located at the center of the circle. The instance can be assigned to the class label of its nearest neighbors. If the nearest neighbors contain more than one class label, one takes a majority vote among the class labels.

The nearest data point to the unknown instance shown in Figure 3.26(a) has a negative class label. Thus, in a 1-nearest neighbor classification scheme, the unknown instance would be assigned to a negative class. If we consider a larger number of nearest neighbors, such as three, the list of nearest neighbors would contain training examples from 2 positive classes and 1 negative class. Using the majority voting scheme, the instance would be classified as a positive class. If the number of instances from both classes are the same, as in the case of the 2-nearest neighbor classification scheme shown in Figure 3.26(b), we could choose either one of the classes (or the default class) as the class label.

The above discussion raises several issues concerning the k -nearest neighbor classification scheme:

1. How to define a good distance measure between two points. We have presented a simple approach based on Euclidean distance measure. However, this measure may not be meaningful for high-dimensional data due to the curse of dimensionality described in Chapter 2.
2. How to choose an appropriate value for k ? If $k = 1$, we can illustrate the decision boundary of each class by using a *Voronoi diagram*, as shown in Figure 3.27. Each polygon P is associated with a training instance \mathbf{x}_P , indicated by a circle in the diagram. If a data point z is located in a region enclosed by P , then z would be closer to \mathbf{x}_P than to all other training instances \mathbf{x}_q in the data set. The drawback of using such a small value of k is that the classifier is sensitive to noise points in the training set. However, if k is too large, the unseen instance can be misclassified because its list of nearest neighbors might

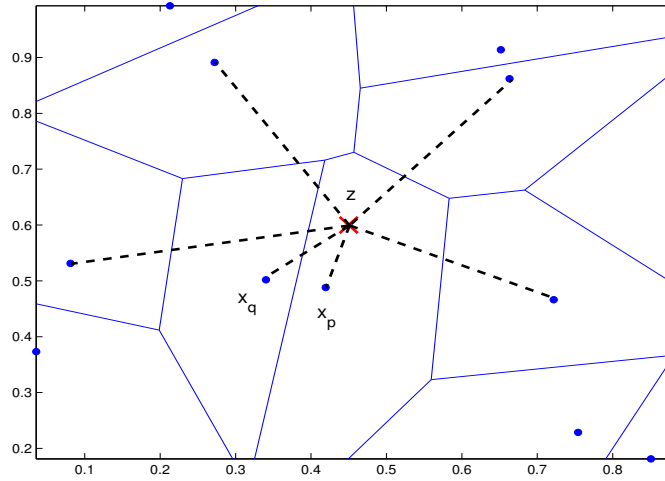


Figure 3.27. Voronoi diagram for 1-nearest neighbor.

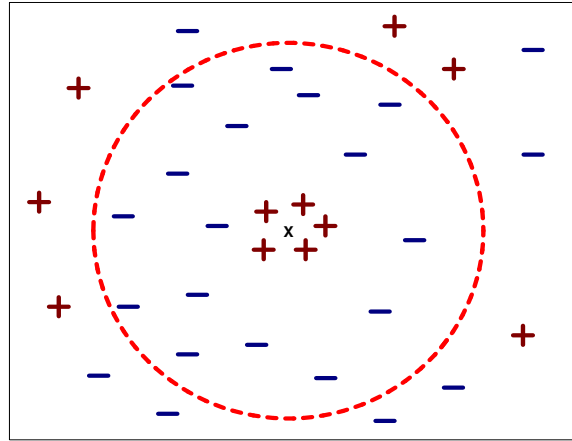


Figure 3.28. k -nearest neighbor classification with large k .

includes data points located beyond the local neighborhood of the unseen instance (see Figure 3.28).

3.5.1 Algorithm

A summary of the k -nearest neighbor classification algorithm is given in Table 3.7. Given an unlabeled instance, we need to determine its distance or similarity to all the training instances. This operation can be quite expensive and may require efficient indexing techniques to reduce the amount of computation.

Once the list of nearest neighbors E' are found, one could take a majority vote to select the most likely class label:

$$\text{Majority Voting: } class = \arg \max_v \sum_{i=1}^{|E'|} I(v, c'_i) \quad (3.24)$$

Table 3.7. k -nearest neighbor classification algorithm.

k -Nearest Neighbor (k :number of nearest neighbor, E :training instances, z : unlabeled instance)
1. Compute the distance or similarity of z to all the training instances.
2. Let $E' \subset E$ be the set of k closest training instances to z
3. Return the predicted class label for z : $class \leftarrow Voting(E')$.

where v is one of the class labels, c'_i is the class label of one of the nearest neighbors, and $I(p, q)$ is a function that returns the value 1 if $p = q$ and 0 otherwise. This approach may not be desirable because it assumes that the influence of the nearest neighbors are the same. It would be better to weight the influence of each nearest neighbor according to its distance. If the distance is too large, the influence should be weaker. The following weighting scheme can be used to penalize neighbors that are located far away from the unlabeled instance: $w_i = 1/d(z, e_i)^2$. In this distance-weighted voting scheme, the class label is determined from the following equation:

$$\text{Distance-Weighted Voting: } class = \arg \max_v \sum_{i=1}^{|E'|} w_i \times I(v, c'_i) \quad (3.25)$$

3.5.2 Characteristics of Nearest-neighbor classifiers

- The k -nearest neighbor algorithm is part of a more general technique known as instance-based learning, where the idea is to make predictions directly using the training instances. The discussion presented so far in this section is applicable to data sets containing continuous attributes. For nominal attributes, we have to re-define the concept of distance between instances in terms of a similarity measure.
- Lazy learners such as nearest neighbor classifiers do not perform any model building task. Without a model, classifying new instances is an expensive operation because we need to compute the distance (or similarity) between instances on the fly as test instances are presented to the classifier. This is in contrast to eager learners that spent the bulk of their computing resources during the model building phase. Once a model has been built, classifying new instances is extremely fast. Lazy learners often seek for models that are applicable only to a limited region, while eager learners are typically interested in finding global models that can explain the entire attribute space.
- The attributes may have to be scaled to prevent the distance measure from being dominated by one of the attributes. For example, suppose each instance corresponds to a person, with attributes height (measured in meters) and weight (measured in pounds). In the training set, the height of a person may vary from 1.5m to 1.85m, but the weight may vary from 90lb to 250lb.

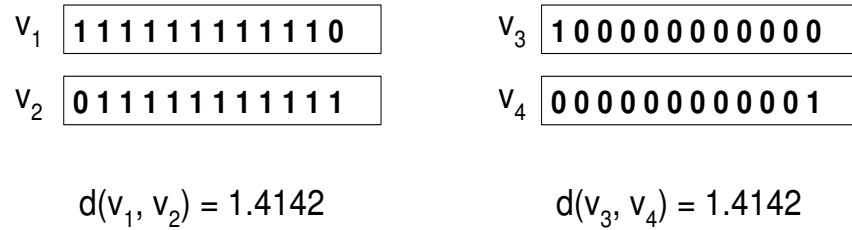


Figure 3.29. Euclidean distance between pairs of unnormalized vectors.

As a result, the distance between two instances is often dominated by their difference in weight.

- Another issue to be considered is the normalization of the attribute vector. Consider the two pairs of attribute vectors shown in Figure 3.29. If the vectors are not normalized, both pairs have the same distance according to the Euclidean measure. This could be a problem for domains such as text document classification where two documents are close to each other because they share many common words, not because they do not have many of the words in the vocabulary. In this regard, the first pair of vectors should have a smaller distance as compared to the second pair of vectors.

3.6 Bayesian classifiers

This section is still being modified

In this section, we describe a statistical approach for solving the classification problem. Specifically, we present the Bayesian classification approach, which allows us to combine the prior knowledge of a given domain with evidence gathered from the data. We begin this section with the Bayes theorem, which is the fundamental principle underlying this approach.

3.6.1 Bayes Theorem

Consider a football game between two rival teams, say team 0 and team 1. Suppose team 0 wins 65% of the time and team 1 wins the remaining matches. Among the games won by team 0, only 35% of them comes from playing at team 1's football field. On the other hand, 75% of the victories for team 1 are obtained while playing at home. If team 1 is to host the next match between the two teams, what is the probability that it will emerge as the winner?

This type of question can be answered by using the well-known Bayes theorem. To begin, we introduce some basic probability definitions. Let X be a random event, *i.e.*, an event that occurs by chance according to some probability $P(X)$.

Example 8 Consider the diagram shown in Figure 3.30 where each point corresponds to the outcome of a random experiment (e.g., tossing a die). Points that belong to the oval X denote events of type X (e.g., the outcome is divisible by 2), while those that belong to the oval Y denote events of type Y (e.g., the outcome is larger than 4). In this example, the probability for event X is $P(X) = 10/20 = 0.5$ between ten out of the twenty points are located inside the oval X . Similarly, we can show that the probability for event Y is $P(Y) = 8/20 = 0.4$.

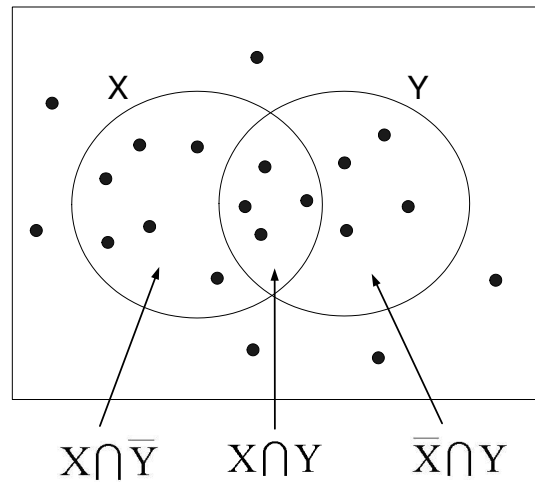


Figure 3.30. Diagram to illustrate probability of events X , Y , $X \cap Y$, $\bar{X} \cap Y$, and $X \cap \bar{Y}$.

The complement of an event corresponds to the opposite outcome of the event. For example, \bar{X} denotes the opposite of event X , *i.e.*, the outcome of the toss is not divisible by 2. In this diagram, \bar{X} is represented by all points that lie outside the oval X . An event of both types (e.g., outcome is divisible by 2 and is larger than 4) is depicted by the intersection between the two ovals and is denoted as $X \cap Y$.

A conditional probability is the probability of an event given that another event has occurred. For example, $P(Y|X)$ is the probability that the outcome is larger than 4 (Y) given that it is known to be divisible by 2 (X). The conditional probability can be computed in the following way:

$$\begin{aligned} P(Y|X) &= \frac{P(X, Y)}{P(X)} \\ &= \frac{4/20}{10/20} = 0.4 \end{aligned} \tag{3.26}$$

where $P(X, Y)$ is the joint probability for $X \cap Y$. Similarly, we can write the

conditional probability for X given Y as

$$P(X|Y) = \frac{P(X, Y)}{P(Y)} \quad (3.27)$$

The conditional probabilities $P(X|Y)$ and $P(Y|X)$ are related according to the following equation:

$$P(X, Y) = P(Y|X) \times P(X) = P(X|Y) \times P(Y) \quad (3.28)$$

We can re-arrange this equation to obtain:

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)} \quad (3.29)$$

Equation (3.29) is known as the *Bayes theorem*.

If $\{Z_1, Z_2, \dots, Z_k\}$ is the set of mutually exclusive and exhaustive outcomes of a random experiment, then the *law of total probability* states that

$$P(X) = \sum_{i=1}^k P(X, Z_i) = \sum_{i=1}^k P(X|Z_i)P(Z_i) \quad (3.30)$$

As an example, the set $\{1, 2, 3, 4, 5, 6\}$ is the set of possible outcomes from tossing a die.

Let us get back to the question posed at the beginning of this section. Let X_i ($i = 0$ or $i = 1$) corresponds to the team hosting the game and Y_i denotes the eventual winner of the game. We can summarize the above information as follows:

Probability that team 0 wins is $P(Y_0) = 0.65$.

Probability that team 1 wins is $P(Y_1) = 1 - P(Y_0) = 0.35$

Probability that team 1 hosted the match it had won is $P(X_1|Y_1) = 0.75$.

Probability that team 1 hosted the match won by team 0 is $P(X_1|Y_0) = 0.35$.

The above question can be solved by computing $P(Y_1|X_1)$, which is the conditional probability that team 1 wins the next match it hosts. Using the Bayes theorem, we obtain:

$$\begin{aligned} P(Y_1|X_1) &= \frac{P(X_1|Y_1) \times P(Y_1)}{P(X_1)} \\ &= \frac{P(X_1|Y_1) \times P(Y_1)}{P(X_1|Y_1)P(Y_1) + P(X_1|Y_0)P(Y_0)} \\ &= \frac{0.75 \times 0.35}{0.75 \times 0.35 + 0.35 \times 0.65} \\ &= 0.5357 \end{aligned}$$

where the law of total probability was applied in the second line. We can also use the Bayes theorem to obtain $P(Y_0|X_1) = 0.4643 = 1 - P(Y_1|X_1)$. From this analysis, we can conclude that team 1 has a higher probability of winning than team 0.

This is an example of a classification problem, where the goal is to predict who will win the upcoming match. Initially, we know the proportion of matches won by each team, $P(Y = 0) = 0.65$ and $P(Y = 1) = 0.35$. If no other information is available, it is safe to bet for team 0 to win simply because $P(Y = 0) > P(Y = 1)$. This is why $P(Y)$ is called the *prior probability* as it encodes our a priori knowledge about the most likely outcome of Y .

Now, suppose we are told that team 1 will be hosting the next match between both teams. How does this information affect our prediction for Y ? Using Bayes theorem, we have shown that team 1 has a higher chance of winning because the conditional probability $P(Y = 1|X = 1)$ is larger than $P(Y = 0|X = 1)$. $P(Y|X)$ is called the *posterior probability* for Y . Choosing an outcome of Y that leads to the highest posterior probability is called the *maximum a posteriori* (MAP) principle.

3.6.2 Using Bayes Theorem for Classification

Given an unlabeled instance, how do we apply the Bayes theorem to perform the classification task? The example given in the previous section describes one possible approach:

1. Given an unlabeled instance $z = (\mathbf{x}, y)$, compute the posterior probability $P(y|\mathbf{x})$ for all values of y .
2. Select the value of y that produces the maximum posterior probability.

Much of the work in Bayesian classification involves the first step, i.e., estimating the posterior probability of each class. The Bayes theorem is useful because it allows us to express the posterior probability in terms of the prior probabilities of each class $P(y)$ and the likelihood function $P(\mathbf{x}|y)$. If we are interested in the posterior probability $P(y|\mathbf{x})$, why do we have to estimate it indirectly using the Bayes theorem? The answer is because it is much easier to compute $P(\mathbf{x}|y)$ and $P(y)$ directly from data. Estimating the posterior probability $P(y|\mathbf{x})$ requires us to have an extremely large data set that covers every possible combination of attribute values \mathbf{x} . In contrast, estimating $P(\mathbf{x}|y)$ and $P(y)$ requires that the coverage for each class is sufficiently large.

There are two common approaches for estimating the posterior probability $P(y|\mathbf{x})$:

Direct estimation: In this approach, given a data set D and an unlabeled instance $z = (\mathbf{x}, y)$, we can estimate $P(\mathbf{x}|y)$ and $P(y)$ directly from data. Note that estimating $P(\mathbf{x}|y)$ can be quite tricky especially when the size of the data set is somewhat limited. Fortunately, by making additional assumptions about the dependencies between the attributes and the class label, one can come up with a practical way for estimating $P(y|\mathbf{x})$ using techniques such as naive Bayes and Bayesian Belief Networks (BBN). The naive Bayes approach is described in the next section.

Generative Models: We can estimate $P(y|\mathbf{x})$ by assuming that the data is generated from a collection of models h in the hypothesis space \mathcal{H} , where:

$$P(y|\mathbf{x}) = \sum_{h \in \mathcal{H}} P(y|h) \times P(h|\mathbf{x})$$

A classifier that uses this approach is known as a Bayes optimal classifier because on average, there is no other classifier that can outperform such a classifier. However, this method is expensive because one has to compute the posterior probability for all the hypotheses. In addition, we need to know the prior probabilities along with the parametric forms of the probability distributions.

3.6.3 Naive Bayes Classifier

In the Bayesian approach, the task of classification corresponds to finding the class label y that maximizes the posterior probability of the unknown instance. This is also known as the *maximum a posteriori principle* (MAP). In this section, we describe the naive Bayes classifier, which can be applied to estimate the posterior probabilities for data containing discrete and continuous attributes.

Let $\mathbf{x} = (x_1, x_2, \dots, x_d)$ be the set of attribute values for an unlabeled instance $z = (\mathbf{x}, y)$. The posterior probability for y given \mathbf{x} can be computed using the Bayes theorem:

$$P(y|\mathbf{x}) = P(y|x_1, x_2, \dots, x_d) = \frac{P(x_1, x_2, \dots, x_d|y) \times P(y)}{P(x_1, x_2, \dots, x_d)}. \quad (3.31)$$

Since we are only interested in comparing the posterior probabilities for different values of y , we can simply ignore the denominator term $P(x_1, x_2, \dots, x_d)$ during our analysis.

$P(y)$ can be estimated as the fraction of training instances that belong to class y . The difficult part is to determine the conditional probability $P(x_1, x_2, \dots, x_d|y)$ for every possible class. Although it is easier to compute than the posterior probability, it is difficult to obtain a reliable estimate for this term unless the size of the training set is sufficiently large.

A naive Bayes classifier attempts to resolve this problem by making additional assumptions regarding the nature of the relationships among attributes. Specifically, it assumes that the attributes are conditionally independent of each other when the class label y is known. In other words: $P(a_i a_j|y) = P(a_i|y) \times P(a_j|y)$ for all i 's and j 's. Therefore,

$$P(x_1, x_2, \dots, x_d|y) = \prod_{i=1}^d P(x_i|y) \quad (3.32)$$

This equation is more practical because instead of computing the conditional probability for every possible combinations of \mathbf{x} given y , we only have to estimate the conditional probability for each pair $P(x_i|y)$.

To classify an unknown instance $z = (\mathbf{x}, y)$, the naive Bayes classifier computes the posterior probability of y given \mathbf{x} using $\prod_{i=1}^d P(x_i|y)P(y)$ and selects the value of y that maximizes this product.

Probability Estimation

We now explain how the conditional probabilities $P(x_i|y)$ can be estimated from a data set D :

Discrete Attributes: For discrete attributes, the conditional probability is estimated by the fraction of instances in class y that take on the particular attribute value x_i . For example, in the taxpayer classification problem illustrated in Figure 3.7, three out of seven law-abiding taxpayers receives a refund. Hence, the conditional probability for $P(\text{Refund} = \text{Yes} | \text{Evade} = \text{No})$ is equal to $3/7$. Computing the conditional probability for grouped attribute values is performed by summing up the individual probabilities since the attribute values are mutually exclusive. For example:

$$\begin{aligned} P(\{\text{Single}, \text{Divorced}\} | \text{No}) &= P(\text{Single} | \text{No}) + P(\text{Divorced} | \text{No}) \\ &= 2/7 + 1/7 = 3/7 \end{aligned}$$

Continuous Attributes For continuous attributes, one could try to discretize the attributes first and creates a new ordinal attribute for each interval. However, this method is not advisable as it violates the independence assumption of a naive Bayes classifier. Alternatively, we can use a two-way split but keep only one of the splits as the new attribute (e.g., use $A < v$). Although this may preserve the independence assumption, it requires additional computation to determine the right splitting criterion. Another approach is to assume that each continuous attribute satisfies a certain form of probability distribution. Typically, a normal distribution is chosen to represent continuous attribute. A normal distribution is characterized by two parameters, the mean of the distribution μ , and the variance, σ^2 . Both μ and σ also depend on the class label. Thus, we may write:

$$P(X = x_i | Y = y_j) = \frac{1}{\sqrt{2\pi}\sigma_{ij}} \exp^{-\frac{(x_i - \mu_{ij})^2}{2\sigma_{ij}^2}} \quad (3.33)$$

(Note that a true interpretation for the right hand side of the above equation is that X should lie somewhere between x_i and $x_i + \epsilon$, where ϵ is a small parameter.)

We can estimate μ_{ij} in terms of the sample mean of all training instances that have the attribute value x_i and the class label y_j . Similarly, σ_{ij}^2 can be estimated by the sample variance of such instances.

For example, consider the taxable income attribute shown in Figure 3.7. For this attribute, its sample mean and variance for $Evade = No$ are:

$$\begin{aligned}\bar{x} &= \frac{125 + 100 + 70 + \cdots + 75}{7} = 110 \\ s^2 &= \frac{(125 - 110)^2 + (100 - 110)^2 + \cdots + (75 - 110)^2}{7(6)} = 2975 \\ s &= \sqrt{2975} = 54.54\end{aligned}$$

Thus, given an unlabeled instance z with taxable income equals to 120K, we can estimate the conditional probability:

$$P(\text{Income} = 120 | \text{Evade} = No) = \frac{1}{\sqrt{2\pi}(54.54)} \exp^{-\frac{(120-110)^2}{2 \times 2975}} = 0.0072$$

Note that the conditional probability $P(x_i|y_j)$ must be estimated for every pair of (attribute value, class label) combination.

Example 9 We shall use the following example to illustrate how naive Bayes classifier works. Given the data set shown in Figure 3.31, we can compute the conditional probabilities of each discrete attribute, along with the sample mean and variance for each continuous attribute. These probabilities and sample statistics are computed from the training set and are used to classify unseen instances.

Given the test instance $\mathbf{x} = (\text{Refund} = No, \text{Marital Status} = Married, \text{Income} = 120K)$, how would you predict its class label? Using the statistics shown in Figure 3.31,

$$\begin{aligned}P(\mathbf{x} | \text{Evade} = No) &= P(\text{Refund} = No | \text{Evade} = No) \\ &\quad \times P(\text{Marital Status} = Married | \text{Evade} = No) \\ &\quad \times P(\text{Income} = 120K | \text{Evade} = No) \\ &= 4/7 \times 4/7 \times 0.0072 = 0.0024 \\ P(\mathbf{x} | \text{Evade} = Yes) &= P(\text{Refund} = No | \text{Evade} = Yes) \\ &\quad \times P(\text{Marital Status} = Married | \text{Evade} = Yes) \\ &\quad \times P(\text{Income} = 120K | \text{Evade} = Yes) \\ &= 1 \times 0 \times 1.2 \times 10^{-9} = 0\end{aligned}$$

Since $P(\text{Evade} = No) = 7/10$, the posterior probability for $P(\text{Evade} = No | \mathbf{x})$ is proportional to $7/10 \times 0.0024 = 0.0016$, which is larger than the posterior probability for $P(\text{Evade} = Yes | \mathbf{x})$. Thus, the instance is classified as $Evade = No$.

The previous example illustrates one potential problem with estimating the conditional probabilities. If the estimated probability for one of the attribute is zero, then the entire posterior probability involving this attribute vanishes. One way to resolve this is to modify the probability estimate by using the Laplace correction

$$P(x_i|y_j) = \frac{n_c + mp}{n + m} \quad (3.34)$$

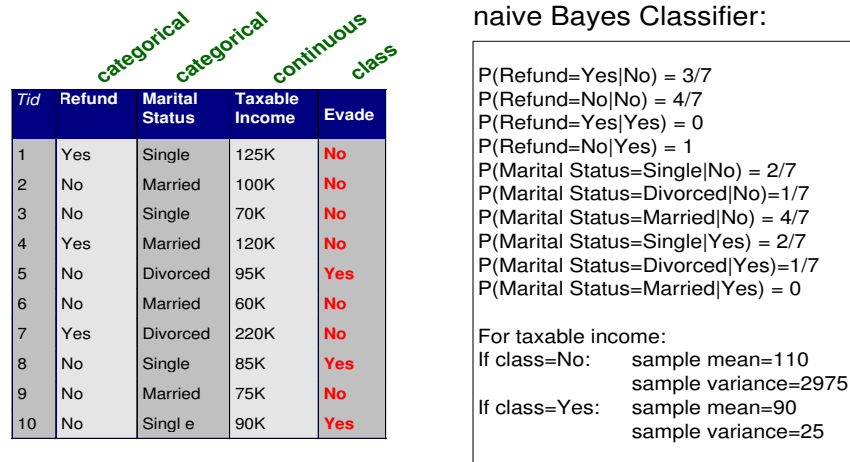


Figure 3.31. The naive Bayes classifier for the taxpayer classification problem.

where n is the total number of instances of class y_j , n_c is the number of instances of class y_j that take the attribute value x_i , m is a parameter known as the equivalent sample size, and p is the prior probability specified by the user. Even if n_c or n is zero due to insufficient samples, the overall conditional probability may not be zero. In the previous example, the conditional probability for (*Marital Status* = *Married*) given the class (*Evade* = *Yes*) is zero. However, if we set $m = 3$, and $p = 1/3$, the conditional probability is no longer zero:

$$P(\text{Marital Status} = \text{Married} | \text{Evade} = \text{Yes}) = (0 + 3 \times 1/3) / (3 + 3) = 1/6$$

Characteristics of Naive Bayes Classifiers

- Naive Bayes classifiers are robust to isolated noise points as they are averaged out when computing probability estimates from the data.
- Most naive Bayes classifiers would handle missing values by simply ignoring the instance during the probability estimate calculations.
- Naive Bayes classifiers are robust to irrelevant attributes.
- In general, the independence assumption may not hold for many practical data sets as most of the attributes are not entirely independent of each other. Alternative techniques such as Bayesian Belief Networks (BBN) are designed to provide a more flexible scheme, allowing the users to specify the prior probabilities as well as the conditional independence among the attributes.

3.7 Artificial Neural Networks (ANN)

This section is still being modified

Consider the classification problem depicted in Figure 3.32. The table on the left shows a data set containing three boolean variables X_1 , X_2 , and X_3 , along with an output variable, Y . As previously mentioned, a classification model can be regarded

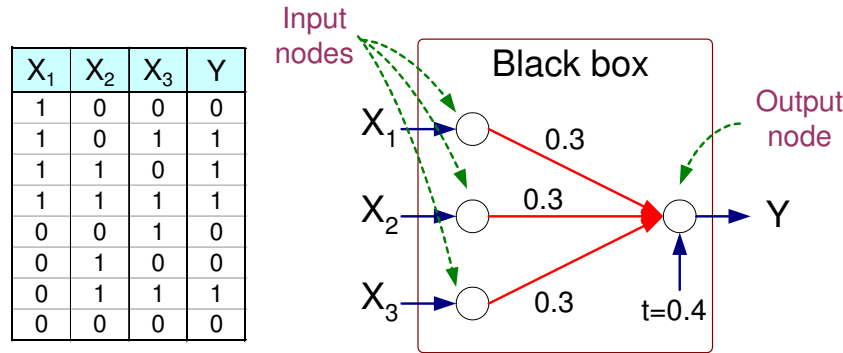


Figure 3.32. Classifying boolean function using neural network.

as a black box that reads the input values for X_1 , X_2 , and X_3 , and sends out an output value $f(X_1, X_2, X_3)$ that is consistent with the true output Y . One obvious question is how does the black box represents the target function?

The classification problem shown in Figure 3.32 can be solved in the following way: if less than two of the three input variables are equal to 1, then $Y = 0$. On the other hand, if at least two of the three input variables are equal to 1, then $Y = 1$. To do this, we can use a black box that models the target function $f(X_1, X_2, X_3) = 0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4$. It is straightforward to verify that $f(X_1, X_2, X_3) > 0$ only if at least two of the three variables are equal to 1. Based on this target function, the black box can produce an output value of 1 whenever $f(X_1, X_2, X_3) > 0$.

In the diagram shown in Figure 3.32, the black box attempts to model this target function using an assembly of inter-connected nodes and weighted links. Specifically, there are three input nodes in the black box, each of which is associated to one of the three input variables. The input nodes are all connected to a single output node, via directed links whose weights are equal to 0.3. A threshold value of $t = 0.4$ is also applied to the output node.

With this configuration, the output node can be regarded as a mathematical device that sums up each of its input value according to the weights of its links, and then subtract off the threshold value from the input sum. This is how the output node can be used by the black box to represent the target function $f(X_1, X_2, X_3)$. Such a black box, which represents its target function using a set of nodes and weighted links, is known as an artificial neural network.

The study of artificial neural networks was inspired by the attempts to model the way human brain works. The human brain consists primarily of nerve cells called *neurons*, linked together with other neurons via strands of fiber called *axons*. Axons are used to transmit nerve impulses from one neuron to another whenever

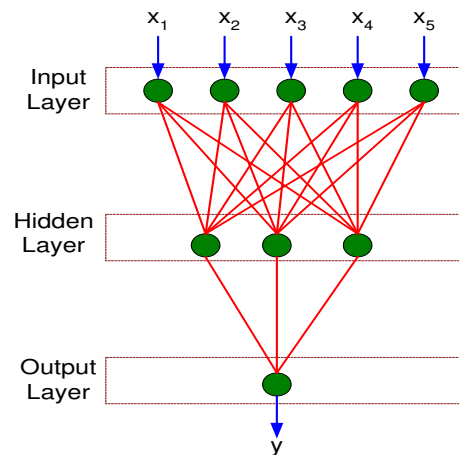


Figure 3.33. Example of an artificial neural network (ANN).

the neurons are stimulated. A neuron is connected to the axons of other neurons via dendrites, which are extensions from the cell body of the neuron. The contact point between a dendrite and an axon is called a synapse. Neurologists have discovered that the human brain learns by changing the strength of the synaptic connection between neurons upon repeated stimulation by the same impulse.

Analogous to the human brain structure, an artificial neural network (ANN) is composed of an inter-connected assembly of nodes and directed links. The nodes in an ANN are often called neurons or units. Each link is associated with a real-valued weight parameter to emulate the synaptic connection strength between neurons. Figure 3.33 illustrates an example of a *multi-layered feed-forward* neural network. It is called a *multi-layered* network because the nodes are arranged in a layered fashion. Likewise, it has a *feed-forward* network topology because each layer contains nodes that are connected only to the nodes in the next deeper layer. This differs from a *recurrent network* topology where backward links exist to connect a node back to its previous layer.

The input layer contains nodes that represent the input variables, while the output layer contains nodes that represent the target (output) variables. As an example, the neural network shown in Figure 3.33 contains 5 input nodes and 1 output node. Zero or more hidden layers may reside between the input and output layers. Neural network that do not have any hidden layers are known as *perceptrons*. By convention, the number of layers a neural network has depends only on the hidden and output layers, i.e., it does not include the input layer which all neural networks must have. For example, the neural network shown in Figure 3.33 has a 2-layered architecture.

During model building, a neural network is trained by adjusting the weights of the links until the outputs produced by the neural network are consistent with the class labels of the training data. The role of the neurons in the input layer is to simply transmit the value they receive from the incoming links to each of the

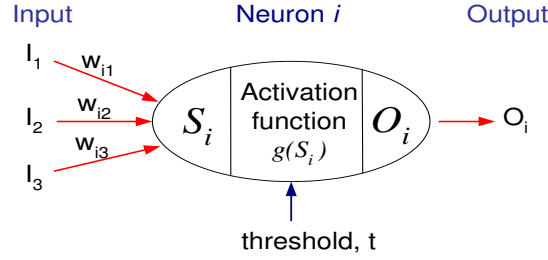


Figure 3.34. Structure of a neuron (unit).

outgoing links without performing any transformation on the input signal. The rest of the neurons are used to combine the input variables into composite features. The structure for such neurons are shown in Figure 3.34. The new composite feature is created by computing the weighted average of the input values (S), and then, applying an activation function g and threshold t to the weighted average. To illustrate how the output value is generated, consider the structure given in Figure 3.34. First, the neuron would sum up the input values, weighted by the weights of the respective input links, i.e., $S_i = \sum_{k=1}^d w_{ik} I_k$. Next, an activation function g is applied to the weighted average S_i in order to obtain its output value O_i . Some of the typical activation functions used for many neural network applications include the sign function and the sigmoid (logistic) function, as shown in the diagram below. In this figure, both activation functions have been displaced to the right by a factor

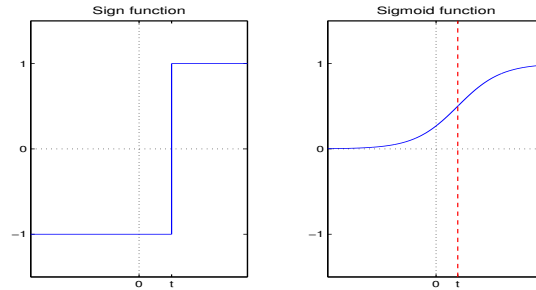


Figure 3.35. Types of activation functions for neurons.

of t to ensure that the output becomes 1 only if the weighted average of the inputs is greater than the threshold t . For example, a sigmoid function can be written as $g(x) = 1/(1 + \exp^{-x})$, where the transition from -1 to 1 occurs at $x = 0$. On the other hand, the sigmoid function shown in Figure 3.35 corresponds to $g(x - t)$, where the transition from -1 to 1 is centered at $x = t$. Thus, given a neuron i with threshold t_i and activation function g , the output can be expressed as $g(S_i - t_i)$, where S_i is the weighted average of the input signals. It is often convenient to absorb the term $-t_i$ into the expression for S_i and rewrite the equation for S_i as $\sum_{k=0}^d w_{ik} I_k$, where $w_{i0} = -t$ and $I_0 = 1$. This notation for S_i will be adopted in

the rest of this section.

Before we train a neural network to learn a classification task, the following steps need to be performed first:

1. Determine the number of nodes in the input layer. Assign an input node to each continuous or binary input variable. If the input variable is discrete, we could either create one node for each discrete value or encode the k -ary variable using $\lceil \log_2 k \rceil$ input nodes.
2. Determine the number of nodes in the output layer. For a two-class problem, it is sufficient to use a single output node. For a k -class problem, there are k output nodes.
3. Select the appropriate network topology (number of hidden layers and hidden nodes, feed-forward or recurrent network architecture, etc). Note that the target function representation depends on the weights of the links, the number of hidden nodes and hidden layers, biases in the nodes and type of activation function. Finding the right topology is not an easy task. One way to do this is to start from a fully-connected network with sufficiently large number of nodes and hidden layers, and then repeat the model building procedure with smaller number of nodes, which can be time-consuming. Alternatively, instead of repeating the model building procedure, we could remove some of the nodes and repeat the model evaluation procedure to select the right model complexity.
4. Initialize the weights and biases.
5. Remove training examples with missing values or replace them with their most likely values.

3.7.1 Back-Propagation Algorithm

Training a neural network amounts to updating the weights of all the link connections until the predicted output of the network is consistent with the actual class label of the training instances. In this section, we describe a well-known learning algorithm called *back-propagation* for finding the appropriate weights of a neural network.

How should a neural network modifies its weighted links? One obvious answer is to modify the weights in such a way that will reduce the misclassification error of the model. In the back-propagation algorithm, the error the network is trying to minimize is

$$E = \sum_{k=1}^N (Y_k - f(\mathbf{X}_k))^2 \quad (3.35)$$

where N is the total number of training instances and $f(\mathbf{X})$ is the overall target function represented by the neural network. In neural network learning, the weights

can be updated in two ways: (1) batch mode, where the weights are updated once after all the training instances have been presented, or (2) online mode (stochastic approximation), where the weights are updated once every time a new training instance is presented. Although both approaches seem quite similar, they are

In general, the weight update formula for each neuron can be written as:

$$w_{ij} = w_{ij} + \lambda \frac{\partial E}{\partial w_{ij}} \quad (3.36)$$

where λ is the learning rate, which is a parameter that controls how rapidly the weight should be updated. The partial derivative term determines how much the weight should be adjusted in order to reduce the observed error of the current configuration of the neural network. We omit the details for deriving the partial derivative term, and present the results below. Interested readers should refer to references such as [129].

$$\text{For output nodes: } \frac{\partial E}{\partial w_{ij}} = (Y - O_j)O_j(1 - O_j)X_{ji} \quad (3.37)$$

$$\text{For hidden nodes: } \frac{\partial E}{\partial w_{ij}} = O_j(1 - O_j) \sum_{k \in \text{Downstream}(j)} \delta_k w_{kj} \quad (3.38)$$

3.7.2 Characteristics of Neural Networks

The main characteristics of decision tree induction are summarized below:

1. Multi-layered neural networks are universal approximators, i.e., they can be used to approximate any target functions. Since it has a very expressive hypothesis space, it is important to choose the appropriate network topology of the problem. For example, it can be shown that a neural network without any hidden layers can approximate any linear functions. It is important to point out that linearity of the function here is with respect to the parameters of the function instead of the input variables.
2. Neural networks are robust to noise.
3. Training a neural network is compute-intensive. However, classifying an unlabeled instance is very fast.

3.8 Support Vector Machine (SVM)

Another classification technique that has received considerable attention in recent years is support vector machine (SVM). SVM has its roots in statistical learning theory and has been shown to outperform existing classifiers in many practical applications.

The basic idea behind support vector machine is illustrated with the example shown in Figure 3.36. In this example, the data is assumed to be *linearly separable*,

i.e., there exists a linear hyperplane (or decision boundary) that separates the points into two different classes. In the two-dimensional case, the hyperplane is simply a straight line. In principle, there are infinitely many hyperplanes that can separate the training data. Figure 3.36 shows two such hyperplanes, B_1 and B_2 . Both hyperplanes can divide the training examples into their respective classes without committing any misclassification errors. The question is, which hyperplane should we prefer?

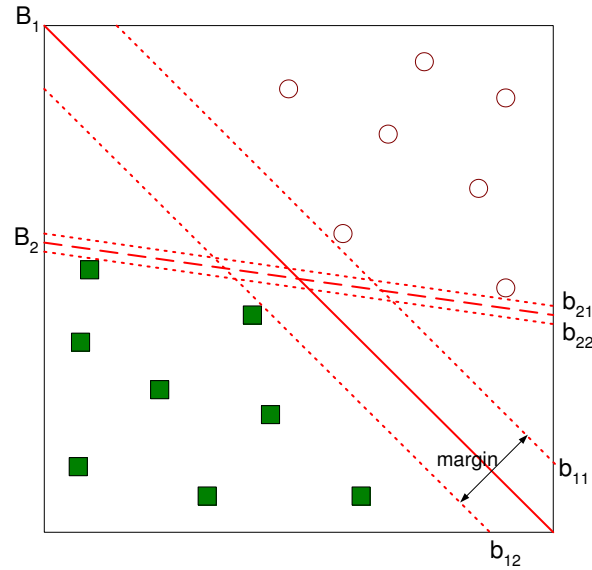


Figure 3.36. An example of a two-class problem with two separating hyperplanes, B_1 and B_2 .

Note that B_1 and B_2 differ not only in terms of their orientation, but also the distance between the nearest point of each class to the decision boundary. For example, b_{11} and b_{12} are two lines that are parallel to B_1 and intersect with the closest points of each class. The distance between these lines is known as the *margin* of the decision boundary. From this figure, B_1 appears to have a much larger margin compared to B_2 . We expect that the larger is the margin of separation, the better is the generalization ability of the classifier because it is less sensitive to minor perturbations of the decision boundary. Thus, B_1 should be the preferred separating hyperplane.

The training phase of SVM consists of finding a separating hyperplane that maximizes the margin between two classes. Once the separating hyperplane is found, test examples are classified depending on which side of the hyperplane they reside. Although the above example is illustrated for data sets that are linearly separable, SVM can also handle situations where the data is not linearly separable and decision boundaries that are non-linear. The trick here is to transform the data into a higher-dimensional space such that a linear separating hyperplane can be used to divide the examples into their respective classes. We will describe the details of how SVM

works in Section 3.8.2.

3.8.1 Preliminaries

We begin with a preliminary discussion of some fundamental concepts in geometry and optimization. These concepts are needed to understand how support vector machine works. Readers who are familiar with these concepts may skip this section and go directly to Section 3.8.2.

Fundamental geometry

In this section, we will briefly describe the general form of a hyperplane equation and how to compute its distance from the origin of the coordinate space. Although, the example shown here is for two-dimensional data, the results are equally applicable to higher-dimensional spaces.

Consider the diagram shown in Figure 3.37. We can compute the equation for the straight-line L in the following way. The general form of a straight-line equation can be expressed as $x_2 = m x_1 + c$, where m is the slope of the line and c is the x_2 -intercept of the line. Since the slope of the line is $-\alpha/\beta$, the equation for L is

$$x_2 = -(\alpha/\beta)x_1 + \alpha$$

or equivalently,

$$\alpha x_1 + \beta x_2 = \alpha\beta.$$

This equation can be expressed in vector notation as

$$\mathbf{w} \cdot \mathbf{x} + b = 0,$$

where $\mathbf{w} = \alpha\hat{\mathbf{i}} + \beta\hat{\mathbf{j}}$, $\mathbf{x} = x_1\hat{\mathbf{i}} + x_2\hat{\mathbf{j}}$ and $b = -\alpha\beta$. $\hat{\mathbf{i}}$ and $\hat{\mathbf{j}}$ denote the unit vectors along the x_1 and x_2 axes, respectively. (A unit vector is nothing more than a vector of unit length.) Such a compact notation is useful because it can be generalized to higher-dimensional spaces.

What is the meaning of the vector \mathbf{w} ? The following example will provide an interpretation of this vector.

Example 10 Given that $\mathbf{w} = \alpha\hat{\mathbf{i}} + \beta\hat{\mathbf{j}}$, the magnitude of this vector is $\|\mathbf{w}\| = \sqrt{\alpha^2 + \beta^2}$, which is the length of the line segment from $(0, \alpha)$ to $(\beta, 0)$. Furthermore, one can show that the direction of this vector is perpendicular to L , as shown in Figure 3.37. In order to prove this, let $\mathbf{L} = -\beta\hat{\mathbf{i}} + \alpha\hat{\mathbf{j}}$ denotes a vector that is parallel to L and directed from $(\beta, 0)$ to $(0, \alpha)$. The dot product of vectors \mathbf{L} and \mathbf{w} is:

$$\mathbf{w} \cdot \mathbf{L} = (\alpha\hat{\mathbf{i}} + \beta\hat{\mathbf{j}}) \cdot (-\beta\hat{\mathbf{i}} + \alpha\hat{\mathbf{j}}) = -\alpha\beta + \alpha\beta = 0, \quad (3.39)$$

Since the dot product vanishes, the direction of \mathbf{w} must be perpendicular to \mathbf{L} .

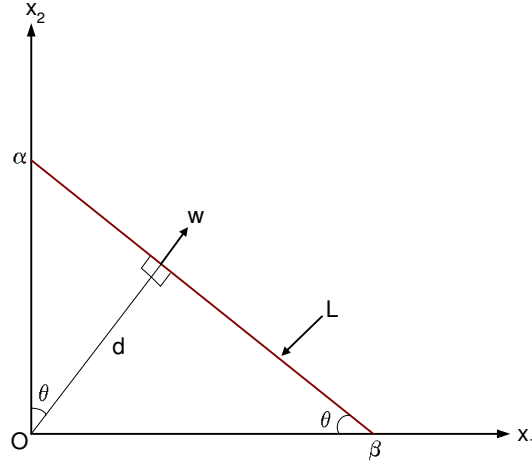


Figure 3.37. Finding the equation for the straight-line L .

Next, we will show how to compute the shortest distance from L to the origin of the coordinate space. The distance is denoted as d in Figure 3.37 and is perpendicular to the line segment L . Let θ be the angle between L and the horizontal axis x_1 , which is also the angle between the line segment for d and the x_2 axis. From trigonometry, we can write

$$\sin \theta = \frac{d}{\beta} \quad \text{and} \quad \cos \theta = \frac{d}{\alpha}.$$

Since $\sin^2 \theta + \cos^2 \theta = 1$, therefore

$$\begin{aligned} \frac{d^2}{\beta^2} + \frac{d^2}{\alpha^2} &= 1 \quad \text{or} \\ d &= \frac{\alpha\beta}{\sqrt{\alpha^2 + \beta^2}} \end{aligned} \tag{3.40}$$

The numerator of this expression is $|b|$ and its denominator is equivalent to $\|\mathbf{w}\|$. Thus, the perpendicular distance from the straight line $\mathbf{w} \cdot \mathbf{x} + b = 0$ to the origin is $d = \frac{|b|}{\|\mathbf{w}\|}$.

One can also show that points that are located above L would satisfy the inequality

$$\mathbf{w} \cdot \mathbf{x} + b > 0$$

while those located below L would satisfy the inequality

$$\mathbf{w} \cdot \mathbf{x} < 0.$$

In addition, any lines parallel to L would have the form

$$\mathbf{w} \cdot \mathbf{x} + b = k,$$

where k is its distance to L .

Optimization Problem

Optimization is an important part of classification because many classification algorithms seek to find an optimal value of an objective function that characterizes the desirability of a model. Often, the objective function is stated in terms of the misclassification error of the model. The goal of a classification algorithm is to find an optimal solution that minimizes this function. There are various techniques available to solve such problems. This section presents a brief overview of such techniques.

Unconstrained Optimization Problem Suppose $f(x)$ is a univariate function with continuous first-order and second-order derivatives. In an unconstrained optimization problem, the task is to find the solution x^* that maximizes or minimizes $f(x)$ without having any constraints on x^* .

The *stationary point* x^* can be found by taking the first derivative of f and setting it to zero:

$$\left. \frac{df}{dx} \right|_{x=x^*} = 0.$$

We can tell whether x^* corresponds to a maximum or minimum stationary point by examining the second-order derivative of the function at x^* :

- x^* is a maximum stationary point if $\frac{d^2f}{dx^2} < 0$ at $x = x^*$.
- x^* is a minimum stationary point if $\frac{d^2f}{dx^2} > 0$ at $x = x^*$.
- x^* is a point of inflection when $\frac{d^2f}{dx^2} = 0$ at $x = x^*$.

Figure 3.38 illustrates an example of a function that contains all three stationary points (maximum, minimum, and point of inflection).

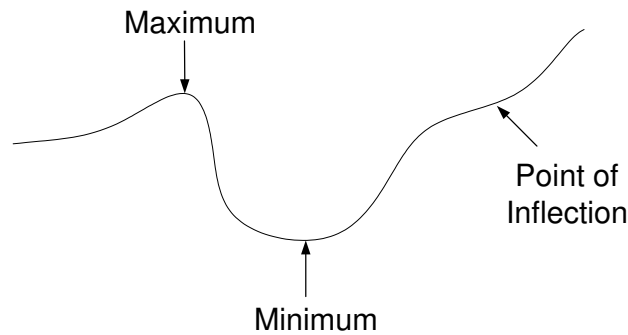


Figure 3.38. Stationary points of a function.

The above method can be extended to a multivariate function, $f(x_1, x_2, \dots, x_d)$.

Again, the condition for finding the stationary point $\mathbf{x}^* = [x_1^*, x_2^*, \dots, x_d^*]^T$ is

$$\left. \frac{\partial f}{\partial x_i} \right|_{x_i=x_i^*} = 0, \quad \forall i = 1, 2, \dots, d.$$

However, unlike univariate functions, it is more difficult to determine whether \mathbf{x}^* corresponds to a maximum or minimum stationary point. The difficulty arises because one has to consider the partial derivatives $\frac{\partial^2 f}{\partial x_i \partial x_j}$ for all possible pairs of i and j . The complete second-order partial derivatives is given by the Hessian matrix:

$$H(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \dots & \frac{\partial^2 f}{\partial x_1 \partial x_d} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2 \partial x_2} & \dots & \frac{\partial^2 f}{\partial x_2 \partial x_d} \\ \dots & \dots & \dots & \dots \\ \frac{\partial^2 f}{\partial x_d \partial x_1} & \frac{\partial^2 f}{\partial x_d \partial x_2} & \dots & \frac{\partial^2 f}{\partial x_d \partial x_d} \end{bmatrix} \quad (3.41)$$

- A Hessian matrix H is positive definite if and only if $\mathbf{x}^T H \mathbf{x} > 0$ for any non-zero vector \mathbf{x} . If $H(\mathbf{x}^*)$ is positive definite, then \mathbf{x}^* is a minimum stationary point.
- A Hessian is negative definite if and only if $\mathbf{x}^T H \mathbf{x} < 0$ for any non-zero vector \mathbf{x} . If $H(\mathbf{x}^*)$ is negative definite, then \mathbf{x}^* is a maximum stationary point.
- A Hessian is indefinite if $\mathbf{x}^T H \mathbf{x}$ is positive for some value of \mathbf{x} and negative for others. If $H(\mathbf{x}^*)$ is indefinite, then \mathbf{x}^* is a *saddle point*.

Example 11 Suppose $f(x, y) = 3x^2 + 2y^3 - 2xy$. The plot of this function is shown in Figure 3.39. The conditions for finding the optimum value of the function are

$$\begin{aligned} \frac{\partial f}{\partial x} &= 6x - 2y = 0 \\ \frac{\partial f}{\partial y} &= 6y^2 - 2x = 0 \end{aligned} \quad (3.42)$$

whose solutions are $x^* = y^* = 0$ or $x^* = 1/27, y^* = 1/9$.

The Hessian of f is

$$H(x, y) = \begin{bmatrix} 6 & -2 \\ -2 & 12y \end{bmatrix}.$$

At $x = y = 0$,

$$H(0, 0) = \begin{bmatrix} 6 & -2 \\ -2 & 0 \end{bmatrix}.$$

Since $[x \ y] H(0, 0) [x \ y]^T = 6x^2 - 4xy = 2x(3x - 2y)$, which can either be positive or negative, the Hessian is indefinite and $(0, 0)$ is a saddle point.

At $x = 1/27, y = 1/9$,

$$H(1/27, 1/9) = \begin{bmatrix} 6 & -2 \\ -2 & 12/9 \end{bmatrix}.$$

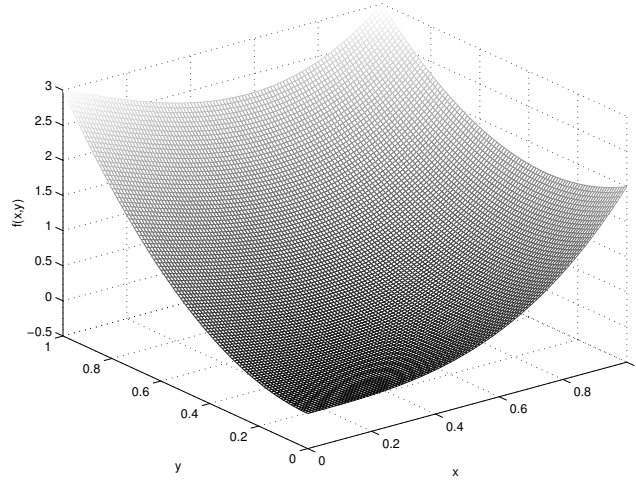


Figure 3.39. Plot for the function $f(x, y) = 3x^2 + 2y^3 - 2xy$.

Since $[x \ y] H(1/27, 1/9) [x \ y]^T = 4x^2 - 2xy + 4y^2/3 = 4(x - y/4)^2 + 13y^2/4 > 0$ for non-zero x and y , the Hessian is positive definite. Therefore, $(1/27, 1/9)$ is a minimum stationary point. The minimum value of f is -0.0014 .

Constrained Optimization Problem Next, we will examine how to solve an optimization problem when the variables are subjected to various types of constraints. While the techniques presented here are used to minimize an objective function $f(\mathbf{x})$, they are also applicable to maximization problems. This is because a maximization problem can be easily turned into a minimization problem by converting the function $f(\mathbf{x})$ to $-f(\mathbf{x})$.

Equality Constraints Consider the problem of finding the minimum value of $f(x_1, x_2, \dots, x_d)$ subjected to equality constraints of the form

$$g_i(\mathbf{x}) = 0, \quad i = 1, 2, \dots, p.$$

A method known as Lagrange multipliers can be used to solve the constrained optimization problem. This method involves the following steps:

1. Define the Lagrangian, $L(\mathbf{x}, \lambda) = f(\mathbf{x}) + \sum_{i=1}^p \lambda_i g_i(\mathbf{x})$, where λ_i is a dummy variable called the Lagrange multiplier.
2. Set the first-order derivatives of the Lagrangian with respect to \mathbf{x} and the Lagrange multipliers to zero,

$$\frac{\partial L}{\partial x_i} = 0, \quad \forall i = 1, 2, \dots, d$$

and

$$\frac{\partial L}{\partial \lambda_i} = 0, \quad \forall i = 1, 2, \dots, p.$$

3. Solve the $(d + p)$ equations in step 2 to obtain the stationary point \mathbf{x}^* and the corresponding values for λ_i 's.

We illustrate how the Lagrange multiplier method works with the following example.

Example 12 Let $f(x, y) = x + 2y$. Suppose we want to minimize the function $f(x, y)$ subject to the constraint $x^2 + y^2 - 4 = 0$. The Lagrange multiplier method can be used to solve this constrained optimization problem in the following way.

First, we introduce the Lagrangian:

$$L(x, y, \lambda) = x + 2y + \lambda(x^2 + y^2 - 4),$$

where λ is the Lagrange multiplier. To determine its minimum value, we need to differentiate the Lagrangian with respect to its parameters:

$$\frac{\partial L}{\partial x} = 1 + 2\lambda x = 0 \quad (3.43)$$

$$\frac{\partial L}{\partial y} = 2 + 2\lambda y = 0 \quad (3.44)$$

$$\frac{\partial L}{\partial \lambda} = x^2 + y^2 - 4 = 0$$

Solving these equations yield $\lambda = \pm\sqrt{5}/4$, $x = \mp 2/\sqrt{5}$, and $y = \mp 4/\sqrt{5}$. When $\lambda = \sqrt{5}/4$, $f(-2/\sqrt{5}, -4/\sqrt{5}) = -10/\sqrt{5}$. Similarly, when $\lambda = -\sqrt{5}/4$, $f(2/\sqrt{5}, 4/\sqrt{5}) = 10/\sqrt{5}$. Thus, the function $f(x, y)$ has its minimum value at $x = -2/\sqrt{5}$ and $y = -4/\sqrt{5}$.

Inequality Constraints Consider the problem of finding the minimum value of $f(x_1, x_2, \dots, x_d)$ subjected to inequality constraints of the form

$$h_i(\mathbf{x}) \leq 0, \quad i = 1, 2, \dots, q.$$

The method for solving this problem is quite similar to the Lagrange method described above. However, the inequality constraints impose additional conditions to the optimization problem. Specifically, the optimization problem stated above leads to the following Lagrangian:

$$L = f(\mathbf{x}) + \sum_{i=1}^q \lambda_i h_i(\mathbf{x}) \quad (3.45)$$

and constraints known as the Karush-Kuhn-Tucker (KKT) conditions:

$$\frac{\partial L}{\partial x_i} = 0, \quad \forall i = 1, 2, \dots, d \quad (3.46)$$

$$h_i(\mathbf{x}) \leq 0, \quad \forall i = 1, 2, \dots, q \quad (3.47)$$

$$\lambda_i \geq 0, \quad \forall i = 1, 2, \dots, q \quad (3.48)$$

$$\lambda_i h_i(\mathbf{x}) = 0, \quad \forall i = 1, 2, \dots, q \quad (3.49)$$

Notice that the Lagrange multipliers are no longer unbounded in the presence of inequality constraints.

Example 13 Suppose we need to minimize the function $f(x, y) = (x - 1)^2 + (y - 3)^2$ subjected to the following constraints:

$$x + y \leq 2, \quad \text{and} \quad y \geq x.$$

The Lagrangian for this problem is $L = (x - 1)^2 + (y - 3)^2 + \lambda_1(x + y - 2) + \lambda_2(x - y)$ subjected to the following KKT constraints:

$$\frac{\partial L}{\partial x} = 2(x - 1) + \lambda_1 + \lambda_2 = 0 \quad (3.50)$$

$$\frac{\partial L}{\partial y} = 2(y - 3) + \lambda_1 - \lambda_2 = 0 \quad (3.51)$$

$$\lambda_1(x + y - 2) = 0 \quad (3.52)$$

$$\lambda_2(x - y) = 0 \quad (3.53)$$

$$\lambda_1 \geq 0, \lambda_2 \geq 0, x + y \leq 2, y \geq x \quad (3.54)$$

To solve the above equations, we need to examine all the possible cases of equations (3.52) and (3.53).

Case 1: $\lambda_1 = 0, \lambda_2 = 0$. In this case, we obtain the following equations:

$$2(x - 1) = 0 \quad \text{and} \quad 2(y - 3) = 0,$$

whose solutions are given by $x = 1$ and $y = 3$. Since $x + y = 4$, this is not a feasible solution because it violates the constraint $x + y \leq 2$.

Case 2: $\lambda_1 = 0, \lambda_2 \neq 0$. In this case, we obtain the following equations:

$$x - y = 0, \quad 2(x - 1) + \lambda_2 = 0, \quad 2(y - 3) - \lambda_2 = 0,$$

whose solutions are given by $x = 2, y = 2$, and $\lambda_2 = -2$, which is not feasible solution because it violates the conditions $\lambda_2 \geq 0$ and $x + y \leq 2$.

Case 3: $\lambda_1 \neq 0, \lambda_2 = 0$. In this case, we obtain the following equations:

$$x + y - 2 = 0, \quad 2(x - 1) + \lambda_1 = 0, \quad -2(x + 1) + \lambda_1 = 0,$$

whose solutions are given by $x = 0, y = 2$, and $\lambda_1 = 2$, which is a feasible solution.

Case 4: $\lambda_1 \neq 0, \lambda_2 \neq 0$. In this case, we obtain the following equations:

$$x + y - 2 = 0, \quad x - y = 0, \quad 2(x - 1) + \lambda_1 + \lambda_2 = 0, \quad 2(y - 3) + \lambda_1 - \lambda_2 = 0,$$

whose solutions are $x = 1, y = 1, \lambda_1 = 2$, and $\lambda_2 = -2$, which is not a feasible solution.

Therefore, the solution for this problem is $x = 0$ and $y = 2$.

Solving the KKT conditions can be quite a laborious task especially if the number of constraining inequalities is large. In such cases, finding a closed-form solution is no longer possible and one has to use numerical techniques such as linear and quadratic programming.

3.8.2 How Support Vector Machine Works?

Now, let us turn to the question of how SVM works. We begin with the simplest case, where the data is assumed to be linearly separable. We then proceed to the more difficult cases, where the data is not linearly separable and for nonlinear decision boundaries.

Linear SVM: Separable Case

Consider a binary classification problem consisting of n training examples. Each example is denoted by the tuple (\mathbf{x}_i, y_i) ($i = 1, 2, \dots, n$), where $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{id})^T$ corresponds to the attributes of the example and $y_i \in \{-1, 1\}$ denotes its class label. Assuming that the data is linearly separable, the decision boundary can be expressed in the general form $\mathbf{w} \cdot \mathbf{x} + b = 0$, as shown in Figure 3.40.

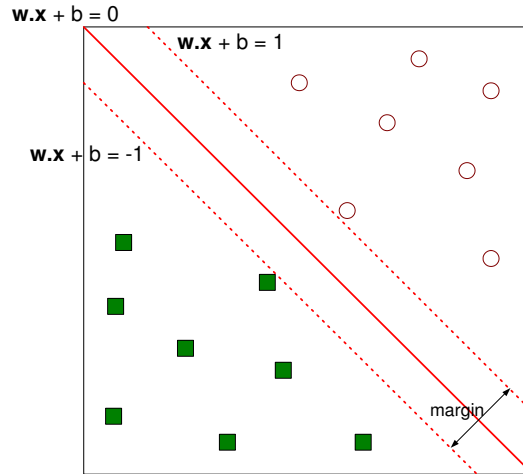


Figure 3.40. Decision boundary of a linear SVM.

Furthermore, each training example can be classified according to the following conditions:

$$\begin{aligned} \mathbf{w} \cdot \mathbf{x}_i + b &\geq 1 \text{ if } y_i = 1, \\ \mathbf{w} \cdot \mathbf{x}_i + b &\leq -1 \text{ if } y_i = -1 \end{aligned} \quad (3.55)$$

These conditions simply state that points located on or above the hyperplane $\mathbf{w} \cdot \mathbf{x} + b = 1$ will be classified as $y = 1$ and those located on or below the hyperplane

$\mathbf{w} \cdot \mathbf{x} + b = -1$ will be classified as $y = -1$. Both inequalities can be summarized in a more compact form as follows

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1, \quad i = 1, 2, \dots, n \quad (3.56)$$

The margin of the decision boundary is given by the distance between the two hyperplanes $\mathbf{w} \cdot \mathbf{x} + b = 1$ and $\mathbf{w} \cdot \mathbf{x} + b = -1$, which is equal to $2/\|\mathbf{w}\|$. Recall that SVM tries to find a decision boundary that maximizes this margin. However, maximizing the margin of separation is equivalent to minimizing the following objective function:

$$f(\mathbf{w}) = \frac{\|\mathbf{w}\|^2}{2}.$$

As a result, the training phase of SVM involves solving the above minimization problem to learn the parameters \mathbf{w} and b , subjected to the constraints given in (3.56). Such a constrained optimization problem can be solved using the Lagrange multiplier method described in the previous section. The Lagrangian of this problem is

$$L_P = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^n \lambda_i y_i (\mathbf{w} \cdot \mathbf{x}_i + b - 1) \quad (3.57)$$

Notice that the second term is subtracted from the first because the inequality (3.56) involves a \geq symbol, instead of a \leq symbol. Also, there are as many Lagrange multipliers as the number of training instances in the data set.

Taking the derivative of L_P with respect to \mathbf{w} and b and setting them to zero would give

$$\mathbf{w} = \sum_{i=1}^n \lambda_i y_i \mathbf{x}_i \quad (3.58)$$

$$\sum_{i=1}^n \lambda_i y_i = 0 \quad (3.59)$$

In addition, the KKT approach described in Section 3.8.1 introduces the following additional constraints

$$\lambda_i \geq 0 \quad \forall i = 1, 2, \dots, n \quad (3.60)$$

$$\lambda_i y_i (\mathbf{w} \cdot \mathbf{x}_i + b - 1) = 0 \quad \forall i = 1, 2, \dots, n \quad (3.61)$$

Equations (3.58) and (3.61) suggest that the decision boundary does not depend on all the training examples. More precisely, only training examples for which $\lambda_i > 0$ affects the decision boundary. These examples are known as *support vectors* and lie along the hyperplanes $\mathbf{w} \cdot \mathbf{x} + b = \pm 1$.

Solving the above optimization problem is a mathematically tedious task because they involve a large number of parameters: \mathbf{w} , b , and λ_i . It would be nice if we can reformulate the problem in such a way that involves only the λ_i 's. To do this,

we first substitute equations (3.58) and (3.59) into (3.57). This will produce a dual formulation of the Lagrangian,

$$L_D = \sum_{i=1}^n \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \quad (3.62)$$

which is an expression involving only the Lagrange multipliers and the training data. Notice that the quadratic term appears with a negative sign, which means that the problem has turned into a maximization problem involving the λ_i 's. Once the λ_i 's are found, we can use equations (3.58) and (3.61) to solve for \mathbf{w} and b . The decision boundary can be written as

$$\left[\sum_{i=1}^n \lambda_i y_i \mathbf{x}_i \cdot \mathbf{x} \right] + b = 0 \quad (3.63)$$

During the testing phase, a test instance \mathbf{z} can be classified by computing

$$f(\mathbf{z}) = \text{sign}(\mathbf{w} \cdot \mathbf{z} + b) = \text{sign}\left(\sum_{i=1}^n \lambda_i y_i \mathbf{x}_i \cdot \mathbf{z} + b\right).$$

Specifically, if $f(\mathbf{z}) = 1$, then the test instance is classified as a positive class, otherwise it is classified as a negative class.

Linear SVM: Nonseparable Case

For the nonseparable case, it is no longer possible to construct a decision boundary that does not commit any misclassification errors in the training set. We now show how the SVM formulation can be modified to handle the nonseparable case.

To do this, we need to introduce a *slack variable* into the constraints of the problem. Specifically, equation (3.55) can be modified into the following form,

$$\begin{aligned} \mathbf{w} \cdot \mathbf{x}_i + b &\geq 1 - \xi_i \quad \text{if } y_i = 1, \\ \mathbf{w} \cdot \mathbf{x}_i + b &\leq -1 + \xi_i \quad \text{if } y_i = -1 \end{aligned} \quad (3.64)$$

where $\xi_i \geq 0, \forall i$.

In addition, the objective function is modified into the following form,

$$f(\mathbf{w}) = \frac{\|\mathbf{w}\|^2}{2} + C \left(\sum_{i=1}^n \xi_i \right)^k,$$

where C is a parameter chosen to balance the size of the margin and cost of misclassification errors. For $k = 1$, one can show that the slack variables will not appear in the dual Lagrangian formulation.

It follows that the Lagrangian for this problem is

$$L_P = \frac{1}{2} \|\mathbf{w}\|^2 + C \left(\sum_{i=1}^n \xi_i \right) - \sum_{i=1}^n \lambda_i \{y_i (\mathbf{w} \cdot \mathbf{x}_i + b) - 1 + \xi_i\} - \sum_{i=1}^n \mu_i \xi_i \quad (3.65)$$

where the last term is due to the non-negativity constraints on the values of ξ_i 's. The Lagrangian is subjected to the following KKT conditions:

$$\begin{aligned}\xi_i &\geq 0, \alpha_i \geq 0, \mu_i \geq 0 \\ y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 + \xi_i &\geq 0 \\ \lambda_i \{y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 + \xi_i\} &= 0 \\ \mu_i \xi_i &= 0\end{aligned}$$

Setting the first-order derivative of L with respect to \mathbf{w} , b and ξ_i to zero would result in the following equations:

$$\frac{\partial L}{\partial w_j} = w_j - \sum_{i=1}^n \lambda_i y_i x_{ij} = 0 \quad (3.66)$$

$$\frac{\partial L}{\partial b} = - \sum_{i=1}^n \lambda_i y_i = 0 \quad (3.67)$$

$$\frac{\partial L}{\partial \xi_i} = C - \lambda_i - \mu_i = 0 \quad (3.68)$$

Substituting equations (3.66), (3.67), and (3.68) into the Lagrangian will produce a dual Lagrangian that is identical to Equation (3.59). The dual problem can then be solved numerically using quadratic programming techniques.

Nonlinear SVM

So far, the above analysis assumes that a linear decision boundary can be constructed to separate the instances into different classes. Nevertheless, there are many data sets for which the optimal decision boundary is nonlinear. Can we still use SVM for this type of data set? The answer is yes. The trick is to transform the data from its original coordinate space in \mathbf{x} to a higher-dimensional space $\Phi(\mathbf{x})$ in such a way that the decision boundary separating the instances becomes linear in the transformed space. That way, we can still apply the machinery presented in the previous sections in the transformed space.

Although this approach seems promising, it raises several implementation issues. First, it is not clear what is the right mapping function $\Phi(\mathbf{x})$ that ensures an accurate linear decision boundary can be constructed in the higher dimensional space. One possibility is to transform the data into an infinite-dimensional space, but such a space may not be that easy to work with. Second, even if we do find the appropriate transformation to a higher dimensional space, the amount of computation that is needed to solve the constrained optimization problem in the higher-dimensional space could become intractable.

An important observation from the results of the previous sections was that the decision boundary and dual Lagrangian formulation involves only dot products of the training instances $\mathbf{x}_i \cdot \mathbf{x}_j$ (see for example, equations (3.62) and (3.63)). Thus, if

we transform the coordinate space from \mathbf{x} to $\Phi(\mathbf{x})$, the decision boundary is equal to

$$f(\mathbf{x}) = \sum_{i=1}^n \lambda_i y_i \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}) + b \quad (3.69)$$

and the Lagrangian for the separable case is

$$L_D = \sum_{i=1}^n \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j) \quad (3.70)$$

Computing the dot product $\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$ in higher dimensional space can be quite expensive. However, if we can find a kernel function such that $K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$, then we do not have to explicitly compute the dot product between the Φ 's in the high dimensional space. Instead, the dot product can be computed using the kernel function itself, which is a much simpler operation that can be performed in the original space \mathbf{x} . This approach would address both of the issues described above.

However, what is the right kernel function to use? The answer to this question goes beyond the scope of this book. Interested readers should refer to the references given in the bibliography remarks.

Some of the typical kernel functions used in various SVM implementations are listed below:

$$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y} + 1)^p \quad (3.71)$$

$$K(\mathbf{x}, \mathbf{y}) = e^{-\|\mathbf{x} - \mathbf{y}\|^2 / (2\sigma^2)} \quad (3.72)$$

$$K(\mathbf{x}, \mathbf{y}) = \tanh(k\mathbf{x} \cdot \mathbf{y} - \delta) \quad (3.73)$$

3.9 Ensemble Methods

This section is still being modified

Given a set of training examples, most of the techniques described so far in this chapter assume that a single classification model is sufficient to represent the data. However, there is considerable interest in improving the accuracy of learning algorithms by aggregating the predictions made by multiple classifiers. These are known as ensemble methods for classification. Ensemble methods construct a set of classifiers from the training data and predict the classes of test examples by combining the predictions of these classifiers. In the following, we describe why ensemble methods can perform better than individual classifiers and methods developed to construct multiple classifiers from a given data set.

3.9.1 Why Ensemble Methods Work?

Would an ensemble of classifiers always perform better than a single classifier? It has been shown that a necessary and sufficient condition for an ensemble to work better is if it is composed of *base classifiers* that are reasonably accurate and independent of each other. Specifically, each classifier has to perform better than random guessing *i.e.*, the error rate should be less than 50%, and the errors made by the base classifiers should be uncorrelated.

Example 14 Suppose an ensemble of twenty-five base classifiers is constructed, each of which have an error rate of $\epsilon = 0.35$. The ensemble classifier takes a majority vote among its base classifiers as its overall prediction. If all the base classifiers are exactly identical, then the error rate of the ensemble remains as 0.35.

However, if the ensemble consists of independent base classifiers, the probability that it makes a wrong prediction is given by

$$\sum_{i=13}^{25} \binom{25}{i} \epsilon^i (1 - \epsilon)^{25-i} = 0.06 \quad (3.74)$$

which is much smaller than 0.35. Thus, ensemble methods depend not only on the accuracy of the individual classifiers, but also on the independence of the classifiers.

3.9.2 Types of Ensemble Methods

Constructing an ensemble of classifiers usually involves the following steps:

1. Create multiple versions of the training data, D_1, D_2, \dots, D_t .
2. For each version D_i , build a base classifier C_i .
3. Classify the test example, \mathbf{x} , by combining the predictions made by the base classifiers:

$$C^*(\mathbf{x}) = \text{Vote}(C_1(\mathbf{x}), C_2(\mathbf{x}), \dots, C_t(\mathbf{x}))$$

The voting scheme can simply take a majority vote of the individual predictions or weighted by the accuracy of the base classifiers.

Below, we describe several types of ensemble classifiers:

Bayesian ensemble. The Bayesian classifier approach is an example of an ensemble method that takes into account the predictions made by multiple classifiers. Test examples are classified by choosing the class that has the highest posterior probability.

Manipulate the data distribution. In this approach, the data is resampled several times, each time using a different subset of training examples. The data can be resampled using various approaches including bootstrap sampling, k -fold cross-validation, or random subsampling. A classifier is then constructed

for each subset. We will illustrate two of the more popular ensemble methods called bagging and boosting in the next sections. These approaches work well for unstable classifiers, *i.e.*, base classifiers that are highly sensitive to small perturbations in the training set.

Manipulate the input features. In this approach, a subset of the input features is used to create the training set, D_i . This approach works well especially when many of the features are correlated with each other. Otherwise, the performance of the ensemble is no better than using a single classifier.

Manipulate the class labels. This method can be used when the number of class labels is sufficiently large. Each data set D_i is constructed by randomly partitioning the classes into two subsets (*e.g.*, A_0 and A_1), and then, relabel the input data according to their new classes. More precisely, examples whose class labels belong to A_0 is relabeled as class 0 and those whose class labels belong to A_1 is relabeled as class 1. The data is then presented to the learning algorithm for training. An ensemble of classifiers is obtained by repeating the class relabeling step multiple times. When a new test example is presented, each base classifier C_i is used to classify the example. If the test example is predicted as class 0, then all class labels that belong to A_0 will receive a vote. On the other hand, if it is predicted as class 1, then all class labels that belong to A_1 will receive a vote. Eventually, the class label that receives the highest vote will be assigned to the test example. This method is also known as *error-correcting output coding* method.

Introduce randomness into the learning algorithm. Another type of ensemble method is to randomize the learning algorithm so that different runs of the algorithm on the same data can lead to different models. Injecting randomness to the algorithm could be as simple as randomizing the input parameters of the algorithm or may require modifications to the base classification algorithm. For example, in neural networks, the initial weights of the links can be chosen randomly to produce different classifiers. In decision tree construction, instead of choosing the best splitting condition at each node, we can randomly choose one of the top k conditions for splitting. Studies have shown that such randomized decision trees can perform better than having a single decision tree.

3.9.3 Bagging

Bagging, which is also known as bootstrap aggregating, is a technique that repeatedly sample (with replacement) from data according to a uniform probability distribution. Each bootstrap sample has the same size as the original data set. In addition, some training examples can be duplicated while others could be missing from the bootstrapped sample.

Given a data set D containing n examples, the basic algorithm for bagging involves the following steps:

1. Create k bootstrap samples of size n : D_1, D_2, \dots, D_k .
2. Train a base classifier C_i on each D_i .
3. Apply each base classifier to a new test instance. The final classification for the new instance is given by the majority vote of the base classifier C_i .

Bagging improves the generalization error by reducing the variance of its base classifiers. This will help to make unstable learners become more stable. Bagging is also useful for alleviating the overfitting problem in noisy environment. However, the performance of bagging depends on the stability of the base classifier. If the base classifier is stable, *i.e.*, robust to small changes in the training data, the error of the ensemble will be primarily caused by bias in the base classifier. In this situation, bagging will not improve the performance of the base learner and may even degrade its performance slightly because the effective size of the training set is smaller in each bootstrap sample.

3.9.4 Boosting

Boosting is an iterative procedure to adaptively change the distribution of training set by focusing more on previously misclassified examples. Initially, the examples are assigned equal weights, so that they are equally likely to be included in the training set. Once a classifier has been induced, it will be applied to all the examples. The weights for examples that are wrongly classified will be increased, while those that are classified correctly will be decreased. That way, the classifier will be forced to focus more on examples that are difficult to classify. The updated weights will be used to create a new training set for the next boosting round.

For new examples, instead of using a majority voting scheme as in the bagging method, boosting will weight the classifier differently according to the performance of the base classifiers.

There are several implementations of the boosting algorithm, the most popular of which is known as Adaboost. In Adaboost, the importance of each base classifier is computed based on its error rate. If ϵ_i is the error rate for classifier C_i , then the importance of C_i is given by the following factor,

$$\alpha_i = \frac{1}{2} \ln \left(\frac{1 - \epsilon_i}{\epsilon_i} \right).$$

Note that α_i is positive if the error rate is smaller than 0.5 and negative if the error rate is larger than 0.5, as shown in Figure 3.41.

Let $w_{i,j}$ be the weight assigned to example (\mathbf{x}_i, y_i) during the j^{th} boosting round. The weight update mechanism in Adaboost uses the following formula:

$$w_{i,j+1} = \frac{w_{i,j}}{Z_j} \times \begin{cases} \exp^{-\alpha_j} & \text{if } C_j(\mathbf{x}_i) = y_i \\ \exp^{\alpha_j} & \text{if } C_j(\mathbf{x}_i) \neq y_i \end{cases} \quad (3.75)$$

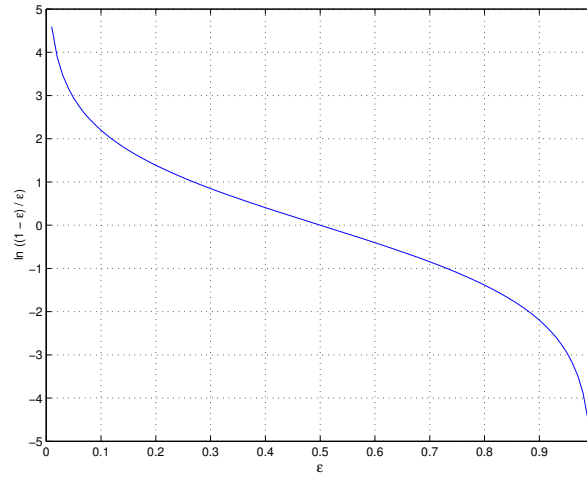


Figure 3.41. Plot of α as a function of training error ϵ .

where Z_j is a normalization factor. The above update formula will increase the weights of wrongly classified examples and decrease the weights of correctly classified examples.

Below, we summarize the Adaboost algorithm:

1. Let $w_{i1} = 1/n$ be the initial weights for all the examples (\mathbf{x}_i, y_i) . Draw a random sample of size n from the training set D according to this initial distribution. Denote D_1 as the initial training set.
2. For $i = 1 \cdots T$:
 - Construct a base classifier for D_i .
 - Estimate the error rate ϵ_i according to the misclassification errors on training set.
 - Calculate the α_i parameter.
 - Update the weight of each example according to equation (3.75).
 - Create a new subset of training examples D_{i+1} by randomly sampling the original data according to the adjusted weight factors.
3. The boosted classifier is $C^*(\mathbf{x}) = \text{sign}(\sum_{j=1}^T \alpha_j C_j(\mathbf{x}))$.

Theoretically, it can be shown that if ϵ_i is always less than 0.5, then the training error will decrease exponentially to zero. However, this does not guarantee that the generalization error will decrease as rapidly too. Freund and Shapire has provided a bound on generalization error of the boosted classifier in terms of several parameters including the training error, size of training set n , number of boosting rounds T , and the VC-dimension (which is a measure of the complexity of the set of target functions). Finally, it is important to point out that although boosting can dramatically improves the performance of base classifiers, it is susceptible to overfitting especially when the number of boosting rounds is large.

3.10 Model Evaluation

This section is still being modified

At this point, we have introduced several classification techniques and described some of their strengths and limitations. For any given data set, one can generate multiple models of the data

- by applying different classification techniques, or
- by choosing different parameters of the learning algorithm, e.g., the number of nearest neighbors in a nearest-neighbor classifier or the threshold for pruning decision trees and classification rules.

As a result, we are often faced with the problem of evaluating and comparing the performance of multiple, competing models. A typical criterion for evaluating the different models is how well the model predicts the class label of the entire data set, including instances it has not seen during model building.

The purpose of this section is to highlight the various issues concerning the model evaluation task and methods that have been developed to deal with these issues. We begin with three important aspects of the model evaluation task.

Metrics for Performance Evaluation: A key issue is how to evaluate the performance of a classification model. Typically, a performance metric is used to assess the quality of a model. We have briefly touched upon this issue in Section 3.2, where metrics such as accuracy and misclassification error rate were introduced. The limitations of these metrics have led to the development of numerous alternative measures for evaluating classification models.

Methods for Performance Evaluation: Since models are built from the training set, the performance metrics we compute are only estimates of the true model performance for the entire data set. Whether the estimates are reliable or not depends on how well the training and test sets represent the overall data. Other factors that may influence the reliability of these estimates include sample size and the dimensionality of the data. In general, estimates of the performance metric can be made more reliable by repeating the experiments several times using different partitions of training and test sets. The various *resampling* techniques for estimating the model performance are described later in this section.

Methods for Model Comparison: A third aspect of the model evaluation task is the comparison of the relative performance of different models. The easiest way to do this is to take the difference in their performance measure as the basis for comparison. Depending on the sampling scheme and size of the training set, the magnitude of the difference may not be a true indicator of

the significance of the observed differences. Unless the differences are shown to be statistically significant, one cannot substantiate any claims about the superiority of one model over another. While there are many statistical tests one could apply, we only discuss several of the more popular tests in this section.

The performance of a classification model can be strongly influenced by the following factors:

Class Distribution: Data sets with unbalanced class distributions are common in the real world. For example, the earth is populated by more non-mammal species than mammals, and there are relatively more law-abiding taxpayers than those who cheat on their taxes. A simple but effective strategy for classification would be to simply assign the majority class to all unknown instances. Although this approach would achieve high classification accuracy, it may not be desirable especially for applications that are more interested in detecting the minority class.

Cost of Misclassification: Related to the issue of unbalanced class distribution is the problem of having unequal misclassification costs. For example, diagnosing a sick person as healthy is often more costly than diagnosing a healthy person as sick. If the cost of misclassifying instances is non-trivial, *e.g.*, biased towards certain classes, then accuracy is no longer a suitable measure for evaluating the performance of different models. Instead, one should incorporate the cost directly into the objective function of the learning algorithm or uses performance metrics that are biased towards certain misclassification errors.

Sample Size: In many classification problems, the training and test sets are samples taken from the entire population. One of the main challenges of the model evaluation task is to infer the properties of the entire population based on information derived from training samples. If the sample size is too small, then it will be difficult to trust the statistical significance of the performance measure. In addition, a small training set also raises the issue of bias and variance in the derived model, two important questions that are discussed later in this chapter.

3.10.1 Metric for Performance Evaluation

As noted in Section 3.2, a classification model can be evaluated on the basis of its accuracy over the test set. Accuracy is a reasonable performance metric because it captures the success rate of a classifier in terms of predicting the class label of unseen instances. When the data set contains equal representatives from all classes, and the costs associated with misclassifying instances are the same across all classes, then accuracy is a sufficient measure to assess the performance of a model.

The statistics for computing the accuracy measure can be obtained from a confusion matrix, as illustrated in the table below. A confusion matrix summarizes the

number of instances classified correctly and incorrectly by the classifier in a tabular format.

		Predicted Class	
		<i>Class</i> = 1	<i>Class</i> = 0
Actual Class	<i>Class</i> = 1	<i>a</i>	<i>b</i>
	<i>Class</i> = 0	<i>c</i>	<i>d</i>

The counts *a*, *b*, *c*, and *d* are also known as:

- *a*: true positive (TP)
- *b*: false negative (FN)
- *c*: false positive (FP) or false alarm
- *d*: true negative (TN)

The true positive rate (*TPR*) or sensitivity is the fraction of positive instances predicted correctly by the model, i.e., $a/(a + b)$. Similarly, the true negative rate (*TNR*) or specificity is the fraction of negative instances predicted correctly by the classifier, i.e., $d/(c + d)$. The false positive rate (*FPR*) is the fraction of negative instances predicted as a positive class by the model, i.e., $c/(c + d)$ and the false negative rate (*FNR*) is the fraction of positive instances predicted as a negative class by the model, i.e., $b/(a + b)$.

Accuracy may not be a reliable measure if one of the following conditions holds:

1. The class distribution is skewed, thus containing a lot more instances from one class than the other. For example, in credit-card fraud detection, the number of fraudulent transactions is significantly less than the number of valid transactions.
2. There is a wide disparity in terms of misclassification costs for different classes. For example, in diagnosing cancer cells, it is more costly to misclassify malignant tissue as healthy tissue than the reverse.

Let $C(i|j)$ denotes the cost of misclassifying an instance of class *j* as belonging to class *i*. For example, $C(-|+)$ is the cost of committing a false negative error and $C(+|-)$ is the cost of making a false alarm. Negative costs can be used to reward models that make correct predictions.

Given a set of test instances, the total cost of a model is:

$$C_t(M) = TP \times C(+|+) + FP \times C(+|-) + FN \times C(-|+) + TN \times C(-|-) \quad (3.76)$$

If the benefits of making correct predictions are ignored, i.e., $C(+|+) = C(-|-) = 0$, the above equation can be simplified as:

$$\begin{aligned} C_t(M) &= FP \times C(+|-) + FN \times C(-|+) \\ &= N(-) \times FPR \times C(+|-) + N(+) \times (1 - TPR) \times C(-|+) \end{aligned} \quad (3.77)$$

where $N(+)$ ($N(-)$) is the total number of positive (negative) instances. In the above equation, we have expressed the total cost in terms of the true positive rate and false positive rate of the model. If $C(+|-) = C(-|+) = 1$, then the model is said to have a 0/1 cost function with a total cost given by

$$N(-) \times FPR + N(+) \times (1 - TPR)$$

Example 15 Consider the cost matrix shown in the table below:

		Predicted Class	
		Class = +	Class = -
Actual Class	Class = +	-1	100
	Class = -	1	0

The cost of committing a false negative error is a hundred times larger than the cost of committing a false positive error. Thus, one could tolerate models that commit up to ninety-nine false positive errors for each false negative error committed by another model. This type of cost function is typically encountered in domains where the positive class of interest is extremely rare. Thus, identifying correctly positive instances amounts to finding a needle in the haystack. In this example, no reward is given for classifying correctly the negative instances. Given two models:

M_1 : $TP = 150$, $FP = 60$, $TN = 250$, and $FN = 40$

M_2 : $TP = 250$, $FP = 5$, $TN = 200$, and $FN = 45$

The total cost of the models are:

$$Cost(M_1) = 150 \times (-1) + 60 \times 1 + 40 \times 100 = 3910$$

$$Cost(M_2) = 250 \times (-1) + 5 \times 1 + 45 \times 100 = 4255$$

One could see that despite the marked improvement in both true positive and false positive counts, M_2 is not desirable because the improvement comes at the expense of increasing the more costly false negative errors. The standard accuracy measure would prefer model M_2 over M_1 because it assumes a 0/1 cost function.

A cost-sensitive classifier takes the cost information into account during model building and produces an output model that commits less costly errors. For example, if false negative errors are more costly, a cost-sensitive classifier would reduce this type of error by moving its decision boundary towards the direction of the negative

class, This would allow the generated model to cover more positive instances even though this may come at the expense of committing more false positive errors, as illustrated in Figure 3.42. For Bayes classifiers, modifying the decision boundary is equivalent to changing the cutoff threshold for declaring a positive class. In the two-class problem, an instance is predicted to be a positive class if the posterior probability is maximum, i.e.,

$$\begin{aligned}
 P(+|\mathbf{x}) &> P(-|\mathbf{x}) \\
 \Rightarrow P(+|\mathbf{x}) &> (1 - P(+|\mathbf{x})) \\
 \Rightarrow 2 \times P(+|\mathbf{x}) &> 1 \\
 \Rightarrow P(+|\mathbf{x}) &> 0.5
 \end{aligned} \tag{3.78}$$

which is equivalent to choosing a cutoff threshold of 0.5 for $P(+|\mathbf{x})$.

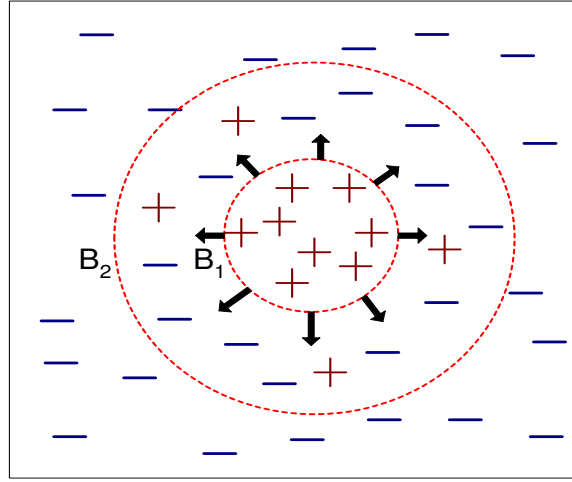


Figure 3.42. Modifying the decision boundary (from B_1 to B_2) to reduce total misclassification cost of a classifier.

A cost-sensitive learning algorithm makes its prediction by choosing the class y_i that minimizes the following quantity:

$$\sum_j P(y_j|\mathbf{x})C(y_i|y_j)$$

For a two-class problem where $\forall i : C(y_i, y_i) = 0$, an instance is declared to be a positive class if:

$$\begin{aligned}
 P(+|\mathbf{x})C(-|+) &< P(-|\mathbf{x})C(+|-) \\
 \Rightarrow P(+|\mathbf{x})C(-|+) &< (1 - P(+|\mathbf{x}))C(+|-) \\
 P(+|\mathbf{x}) &< \frac{C(+|-)}{C(+|-) + C(-|+)}
 \end{aligned} \tag{3.79}$$

which is equivalent to modifying the threshold to $t = C(+|-)/(C(+|-) + C(-|+))$. If $C(+|-) < C(-|+)$, then we should choose a cutoff threshold $t < 0.5$ for $P(+|\mathbf{x})$ as making a false negative error is more costly.

Even if the learning algorithm is not cost-sensitive, there are measures available to select appropriate models that are biased towards certain types of cost functions. For example, the precision and recall measures given below can be used to assess how good is a classifier in terms of reducing certain types of misclassification errors. For instance, recall is a measure that analyzes the trade-off between true positives and false negative errors. A model with high recall would commit fewer false negative errors when making predictions. Analogously, precision is a measure that analyzes the trade-off between false positive and true positive errors. A model with high precision would commit fewer false positive errors when making predictions.

$$\text{Precision, } p = \frac{a}{a + c} \quad (3.80)$$

$$\text{Recall, } r = \frac{a}{a + b} \quad (3.81)$$

$$\text{F-measure, } F_1 = \frac{2rp}{r + p} = \frac{2 \times TP}{2 \times TP + FP + FN} \quad (3.82)$$

F-Measure is the harmonic mean of recall and precision:

$$F_1 = \frac{2}{\frac{1}{r} + \frac{1}{p}}$$

To illustrate the meaning of a harmonic mean, consider two real numbers, a and b . The harmonic mean between both numbers, say c , is located in the interval $[a, b]$ where $(c - a)/a = (b - c)/b$. Solving this equality would yield $c = \frac{2ab}{a+b}$. Note that a harmonic mean is located closer to the smaller value between a and b . A comparison between harmonic, geometric, and arithmetic means is given below:

Example 16 Consider two positive numbers $a = 1$ and $b = 5$. Their arithmetic mean is $\mu_a = (a + b)/2 = 3$ and their geometric mean is $\mu_g = \sqrt{ab} = 2.236$. Their harmonic mean is $\mu_h = (2 \times 1 \times 5)/6 = 1.667$. Thus, a harmonic mean is closer to the smaller value between a and b than both arithmetic and geometric means.

3.10.2 Methods for Performance Evaluation

Once we have decided on the choice of a performance metric, the next step is to conduct experiments that can evaluate the performance of a classification model.

Learning curve

A classification technique is *stable* if it induces models that make similar predictions on test instances when the training set is slightly perturbed. If the size of the training set is too small, the models induced by the classification technique can

be quite unstable, thus producing erratic performance values. One way to check whether the training set is large enough to produce consistent results is to plot a *learning curve*, which depicts how the model accuracy on test set varies with increasing number of training instances.

Figure 3.43 illustrates an example of a learning curve obtained using a *geometric sampling schedule*. The size of the training set is incremented geometrically (2, 4, 8, 16, ...) rather than arithmetically (5, 10, 15, 20, ...) to reduce the number of experiments needed for obtaining such a learning curve. At each sample size, we repeated the experiments ten times by randomly selecting instances from the overall data set to be its training set. (Note that the size of the test set is fixed throughout the experiments.)

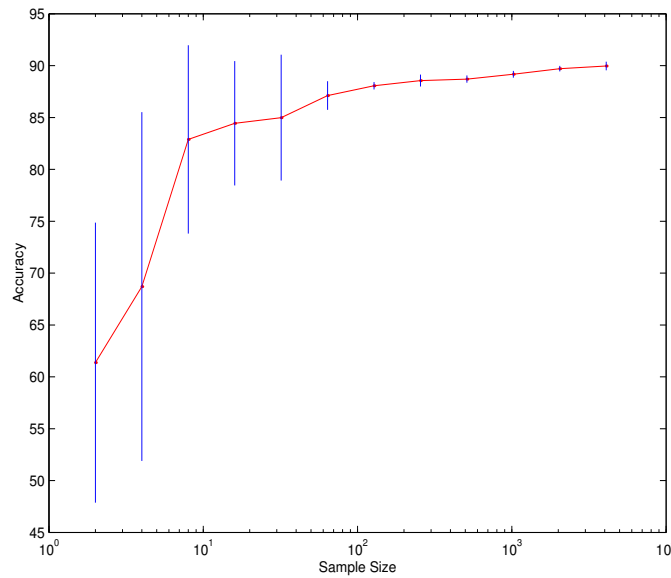


Figure 3.43. A learning curve.

When the size of the training set is small, the model accuracy is poor, with an average close to 60%. However, as the number of training instances increases, model accuracy improves until it converges to 90%. Let acc_t be the true accuracy of the classification technique and $acc_e(s)$ be the empirical accuracy obtained for sample size s . The bias of the model in terms of estimating accuracy is defined as $(acc_t - acc_e(s))$. For the learning curve shown in Figure 3.43, the true accuracy acc_t is located close to the convergence value of the learning curve. Thus, when the sample size is small, the bias of the model would be large. The variance in the accuracy measure is also extremely high. However, as we increase the sample size, both bias and variance become smaller, indicating a more stable model.

In principle, we like to have as many instances as possible for training and testing to ensure that the induced model is stable and that the performance measure is reliable. A large training set is needed to reduce the bias and variance of the

model. In addition, the model building and evaluation steps should be repeated as many times as possible to reduce the variance of the measure. We also need as many instances as possible for the test set to ensure that the estimate has a small confidence interval.

Estimation Methods

Using the training set for estimating the performance measure is unacceptable because it may not generalize well to previously unseen instances. In addition, the discussion in the previous section also suggests that the performance measure can be quite sensitive to the choice of training and test instances. Thus, more robust estimation methods such as resampling techniques are needed. Below, we describe some of the methods commonly used to obtain more a reliable estimate of the performance measure.

Holdout Method In this method, the data set is partitioned into two mutually disjoint sets, called the training set and the test set. A classification model is learned from the training set and its performance measure is evaluated using the test set. Typically, one would reserve two-thirds of the data for training and the rest for testing. The accuracy of the model is obtained using the formula given in Equation (3.1). The disadvantages of this method are (1) it reduces the amount of data available for training, and (2) the model can be biased by the composition of the training set. The more instances left out of the training set, the larger is the bias of the estimate. On the other hand, the smaller the size of the test set, the larger the confidence interval of the performance measure.

Random Subsampling The holdout method can be repeated several times in order to reduce the bias of the training instances. This is also known as the *random subsampling* method. The overall accuracy is estimated by taking the average accuracy of each holdout, i.e., $acc_{sub} = \frac{\sum_{i=1}^k acc_{h,i}}{k}$, where $acc_{h,i}$ is the accuracy value for each holdout i . As with the previous method, random subsampling typically does not use as much data as possible for training.

Cross-Validation An alternative to random subsampling is cross-validation. Random subsampling can be undesirable because one has no control over the number of times each instance is used for testing and training. As a result, some instances could be tested multiple times while others might be missed. We can avoid this problem by using the cross-validation approach, where each instance is used once for testing. To illustrate the idea of cross-validation, suppose we partition the data into two mutually exclusive subsets, where one is used for training while the other is used for testing. We then swap the role of the subsets so that the training set now becomes the test set, and vice-versa. This is called a 2-fold cross-validation approach. The final model accuracy can be estimated by averaging the accuracy over the two runs. In general, a

k -fold cross validation can be performed by segmenting the data into k disjoint partitions. At each run, one of the partition will be chosen for testing while the rest of them are used for training. Since the number of runs is equal to the number of partitions k , each partition is used only once for testing. The overall model accuracy is estimated by the average accuracy over the k runs. A special case of k -fold cross-validation is to set $k = N$, the total number of instances in the entire data set. In this *leave-one-out* approach, the test set contains only one instance. The advantage of this approach is that as much data as possible is used for training, and the test sets are mutually exclusive and cover the entire data set. The drawback of this approach is that it can be quite expensive to compute.

Stratification In this approach, the frequency of each class in a sample is changed according to their cost of misclassification. Stratification can be done by oversampling the minority class or undersampling the majority class. For example, given a data set that contains 100 positive instances and 1000 negative instances, stratification by oversampling would duplicate each positive instance 10 times so that the resampled data set contains 1000 positive and negative instances. In the case of stratification by undersampling, a random sample of 100 negative instances will be chosen so that the resampled data set contains 100 positive and negative instances. Undersampling reduces the amount of data available for training while oversampling increases the computation time. Stratification can be combined with other approaches such as cross-validation and random subsampling. However, stratification is not applicable to the leave-one-out approach because the test set contains only a single instance.

Bootstrap All the methods described so far assume that the instances are selected without replacement. As a result, there are no duplicates in the training and test sets unless the method involves stratification by oversampling. The bootstrap approach is an exception because it allows for sampling with replacement. Given a data set D containing N instances, a bootstrap sample is obtained by randomly selecting N instances (with replacement) from D . Sampling with replacement means the same instance can be re-drawn from the training set, thus allowing it to be used more than once. It can be shown theoretically that on average, a bootstrap sample contains only about 63.2% of the total training instances. This is because the probability that an instance is not chosen by a bootstrap sample is $(1 - 1/N)^N \simeq e^{-1} = 0.368$ while the probability that it is chosen is $1 - 0.368 = 0.632$. This percentage is much smaller than that of a 10-fold cross validation, which contains 90% of the training data. Any instances that are not included in the bootstrap sample become the test set for the current bootstrap iteration. The bootstrap sampling is repeated b times to generate b bootstrap samples.

There are many variations to the bootstrap sampling procedure in terms of how the accuracy or misclassification error is computed. One of the more

popular bootstrap approach is called *.632 bootstrap*, which determines the overall accuracy by taking a weighted average of the accuracies estimated from the test set of each bootstrap sample ($\epsilon_{0,i}$) and the accuracies estimated for the training set (acc_s):

$$\text{Accuracy, } acc_{boot} = \frac{1}{b} \sum_{i=1}^b (0.632 \times \epsilon_{0,i} + 0.368 \times acc_s) \quad (3.83)$$

Not the training accuracy of each bootstrap sample, while others use the training accuracy of a model built from the full data set.

Kohavi [103] has performed an extensive study comparing the average accuracies obtained for six different data sets using the various resampling techniques described above. Their results indicate that the best method for model selection is the ten-fold stratified cross validation.

3.10.3 Methods for Performance Comparison

In this section, we examine the different methods available for comparing the performance of different classification models. We first illustrate a method called the *ROC curve*, which originates from the field of signal detection theory. We then discuss the issue of evaluating the statistical significance of performance measures.

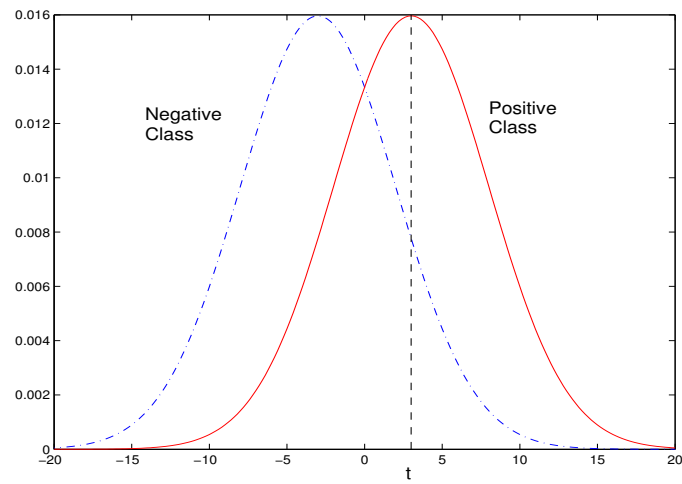
ROC curve

An ROC curve is a useful graphical tool to display the trade-off between the true positive rate (*TPR*) and false positive rate (*FPR*) of a classifier. In an ROC curve, the true positive rate is plotted on the *y*-axis and the false positive rate is shown on the *x*-axis. This type of analysis can be performed on models that produce continuous output values (e.g., estimates of the posterior probabilities) such as Naive Bayes classifiers. For example, Figure 3.44 illustrates an example of an ROC curve for a data set consisting of two univariate normal distributions.

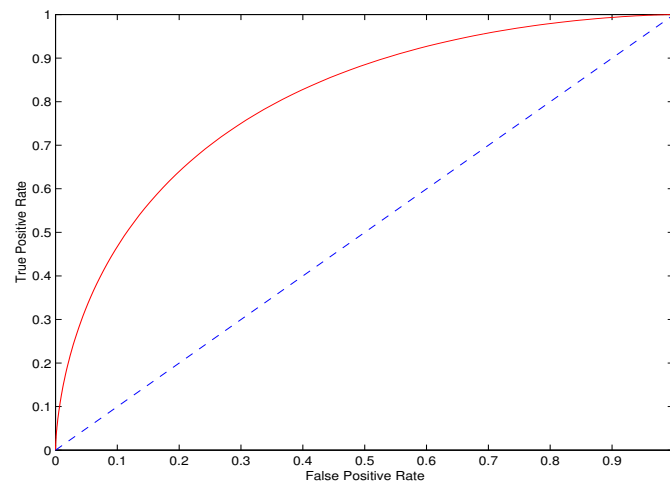
Each point along the ROC curve corresponds to a particular implementation of the classifier, i.e., a specific model that was generated by the classifier. There are several critical points along the ROC-curve that have well-known interpretation:

- (TPR=0, FPR=0): Model predicts every instance to be a negative class.
- (TPR=1, FPR=1): Model predicts every instance to be a positive class.
- (TPR=1, FPR=0): The ideal model.

A good model should have large true positive rate and small false positive rate. In other words, the point should be located as close as possible to the point ($TPR = 1, FPR = 0$). If a classifier makes only random guesses, then it would reside along the diagonal line that connects the points ($TPR = 0, FPR = 0$) and ($TPR =$



(a) Data set for two normal distributions.



(b) ROC curve

Figure 3.44. ROC curve for two 1-dimensional normal distributions.

1, $FPR = 1$). Another important property of the ROC curve is that it is non-decreasing.

An ROC curve is useful for comparing the relative performance of different classifiers. For example, Figure 3.45 illustrates the ROC curve for two different classifiers, M_1 and M_2 . In this example, M_1 is better than M_2 when FPR is less than 0.36. On the other hand, M_2 is a better classifier when the FPR is greater than 0.36. Thus, there is no classifier that clearly dominates the other.

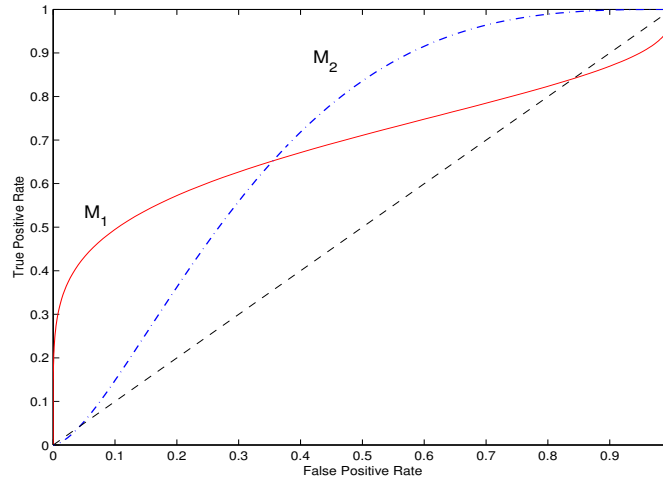


Figure 3.45. ROC curves for two different classifiers.

Nevertheless, if we are interested to know, on average, which model is better, then the area under the ROC curve would serve as a good metric. If the model is perfect, then its area under the ROC curve would equal to 1. If the model corresponds to random guessing, then its area under ROC curve would be equal to 0.5. Anything less than 0.5 would be worse than random guessing. A classifier that often outperforms another would have a larger area.

Suppose we have a classifier that generates the posterior probabilities $P(y|\mathbf{x})$ for each test instance. The following steps can be used to construct the ROC curve of the classifier:

1. Sort the instances according to $P(y|\mathbf{x})$ in decreasing order.
2. Set the cutoff threshold to be equal to the largest value of $P(y|\mathbf{x})$. That way, none of the instances will be classified as a positive class, and thus, $TPR = FPR = 0$.
3. Set the cutoff threshold to be the next largest value of $P(y|\mathbf{x})$. As a result, instances with posterior probabilities higher than the threshold (let's called them I_p) will be classified as a positive class. For each instance in I_p , if the class label associated with the instance is positive, increment TP and decrement

FN . Otherwise, increment FP and decrement TN . Compute TPR and FPR for the particular threshold.

4. Repeat step 2 and 3 until you reach the end of the list.
5. Plot TPR against FPR to obtain the ROC curve.

An example of how to compute the ROC curve is shown in Figure 3.46. Assume that the test set contains ten instances and two classes, positive and negative. The class labels of the instances are shown on the first row of the table. The second row corresponds to the sorted list of posterior probabilities for the instances. The next six rows contain entries for TP , FP , TN , FN , TPR , and FPR . The table can be filled from left to right, or right to left. If we start from right to left, initially, all instances are predicted to be the negative class. Thus, $TP = FP = TPR = FPR = 0$. When we move to the next cell to its left, we add at most one positive example, which increments the count for either TP or FP while reducing the count for either FN or TN . We keep repeating this until we reach the other end of the list, where $TPR = 1$ and $FPR = 1$. The ROC curve is plotted using information from the last two rows and is shown in Figure 3.46. Unlike the previous ROC curves, this curve increases monotonically in a step-wise fashion.

P	Class	+	-	+	-	-	-	+	-	+	+	
		0.25	0.43	0.53	0.76	0.85	0.85	0.85	0.87	0.93	0.95	1.00
	TP	5	4	4	3	3	3	3	2	2	1	0
	FP	5	5	4	4	3	2	1	1	0	0	0
	TN	0	0	1	1	2	3	4	4	5	5	5
	FN	0	1	1	2	2	2	2	3	3	4	5
	TPR	1	0.8	0.8	0.6	0.6	0.6	0.6	0.4	0.4	0.2	0
	FPR	1	1	0.8	0.8	0.6	0.4	0.2	0.2	0	0	0

Figure 3.46. Constructing an ROC curve.

How do we compute the ROC curve for a model evaluated using the k-fold cross-validation? Provost et al. suggested that one approach would be to take the ROC curve for each fold, fit a linear function between each pair of points, and then averaged the fitted functions to obtain an aggregated ROC curve.

Test of significance

Testing the statistical significance of a performance measure is an important aspect of model evaluation. The smaller the number of instances in the test set, the more crucial it is for us to determine how reliable is the estimated measure. For example, suppose you have a model M_A that attains 85% accuracy on a test set containing 30 instances and another model M_B , whose accuracy on the test set is merely 75% but has been tested on 5000 instances. Given this information, would you prefer model M_A over model M_B ?

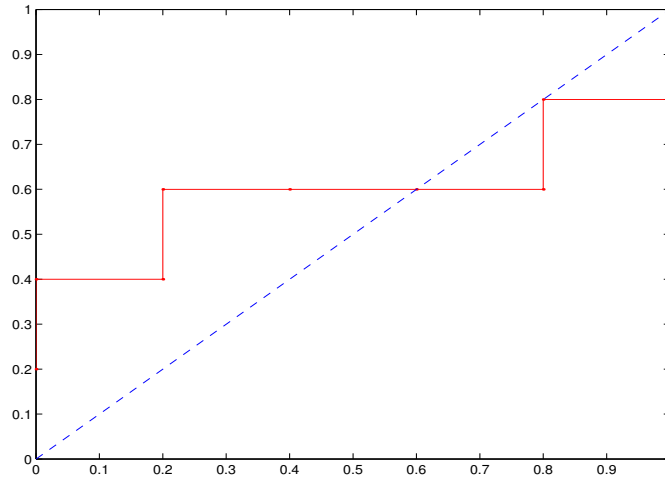


Figure 3.47. ROC curve for the data shown in Figure 3.46.

The above example raises two key issues regarding the statistical significance of the performance measures:

1. Although model M_A has an accuracy of 85%, it has only been tested on 30 test instances. How much confidence can we place on this estimated value?
2. Could the observed disparity in the performance measures be explained as a result of random fluctuations in the test sets?

These issues can be addressed by using well-known statistical methods described in this section. First, we will present a statistical method for computing the confidence interval of the accuracy measure.

Each prediction made by a classification model can be regarded as a Bernoulli experiment, as there are only two possible outcomes of the prediction: correct or wrong. (An example of a Bernoulli experiment is tossing a coin and predicting whether a head or tail turns up.) Let p be the probability that a test instance is predicted correctly. In other words, p corresponds to the true accuracy of the model. Suppose X is the number of test instances predicted correctly by the model, then X can be characterized by a binomial distribution with mean Np and variance $Np(1 - p)$, where N is the size of the test set. For example, if $N = 30$ and $p = 0.8$, we would expect the model to predict 24 out of the 30 test instances correctly.

In reality, we do not know the precise value for the true model accuracy, p . Given the values of N and X , our task is to provide a good estimate of p . Since every value of p has some probability of generating the outcome X , the best we can do is to use the estimated model accuracy, $acc = X/N$, which provides an unbiased estimate for p . Furthermore, if N is sufficiently large, acc has a normal distribution with mean p and variance $p(1 - p)/N$. We can standardize the estimated accuracy

acc so that its mean is 0 and its variance is 1. After standardization, we may write:

$$P(-Z_{\alpha/2} \leq \frac{acc - p}{\sqrt{p(1-p)/N}} \leq Z_{1-\alpha/2}) = 1 - \alpha \quad (3.84)$$

where $(1 - \alpha)$ is the confidence level associated with the standardized accuracy measure. $Z_{\alpha/2}$ and $Z_{1-\alpha/2}$ are the corresponding upper and lower bounds of the standardized accuracy. Since the standardized normal distribution is symmetric at $Z = 0$, it follows that $Z_{\alpha/2} = Z_{1-\alpha/2}$. Re-arranging the above inequality gives the following confidence interval for p ,

$$\frac{2 \times N \times acc + Z_{\alpha/2}^2 \pm Z_{\alpha/2} \sqrt{Z_{\alpha/2}^2 + 4Nacc - 4Nacc^2}}{2(N + Z_{\alpha/2}^2)} \quad (3.85)$$

The table below illustrates the different values of $Z_{\alpha/2}$ at various confidence levels $(1 - \alpha)$.

$1 - \alpha$	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 17 Consider a model that produces an accuracy of 80% when evaluated on 100 test instances. What is the confidence interval for its true model accuracy at 95% confidence level?

Since the confidence level is 95%, this would correspond to $Z_{\alpha/2} = 1.96$ according to the table given above. Replacing this term into Equation (3.85) would yield a confidence interval for p between 71.1% and 86.6%. If we had used a larger number of instances, say $N = 1000$, then a tighter confidence interval is produced, between 77.4% and 82.3%.

Now that we know how to estimate the confidence interval for the true accuracy measure, can we do the same when comparing the accuracies (or misclassification errors) between competing models and learning algorithms?

Comparing the performance of two models: Suppose we are given two models, M_1 and M_2 evaluated on two *independent* test sets, D_1 and D_2 . Let n_1 denotes the size of the test set D_1 and n_2 denotes the size of the test set D_2 . After applying the models, 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 difference between e_1 and e_2 are statistically significant.

If n_1 and n_2 are sufficiently large, then the error rates e_1 and e_2 can be approximated by using two independent normal distributions. If we represent the difference in error rate as $d = e_1 - e_2$, then d is also normally distributed.

(This comes from a well-known in fact in statistics that the sum or difference of two normal distributions is also normal.) The mean for d is given by d_t , which is the true difference between the error rates, while its variance, σ_d^2 , is the sum of variances for e_1 and e_2 .

$$\sigma_d^2 \simeq \hat{\sigma}_d^2 = \frac{e_1(1-e_1)}{n_1} + \frac{e_2(1-e_2)}{n_2} \quad (3.86)$$

where we have approximated the variance for e_1 and e_2 with $e_1(1-e_1)/n_1$ and $e_2(1-e_2)/n_2$. At the $(1-\alpha)\%$ confidence level, the confidence interval for the true difference is

$$d_t = d \pm z_{\alpha/2} \hat{\sigma}_d \quad (3.87)$$

Example 18 Consider the problem described at the beginning of this section. The first model M_A has an error rate of $e_1 = 0.15$ given $N_1 = 30$ test instances. The second model M_B has an error rate of $e_2 = 0.25$ given $N_2 = 5000$ test instances. The difference between their error rates is $d = |M_A - M_B| = |0.15 - 0.25| = 0.1$. (Here, we are considering only a 2-sided test, whether $d_t = 0$ or $d_t \neq 0$.) Assuming that the test sets are statistically independent, then the observed difference d is normally distributed with mean d_t and variance σ_d^2 , which can be estimated using Equation (3.86).

$$\hat{\sigma}_d^2 = \frac{0.15(1-0.15)}{30} + \frac{0.25(1-0.25)}{5000} = 0.0043$$

or $\hat{\sigma}_d = 0.0655$. The confidence interval for d_t at 95% confidence level is $0.1 \pm 1.96 \times 0.0655 = 0.1 \pm 0.128$. As the confidence interval spans the value zero, the observed difference is not statistically significant at 95% confidence level.

At what confidence level can we reject the hypothesis that $d_t = 0$? To do this, we need to find the value of $Z_{\alpha/2}$ for which $d > Z_{\alpha/2} \hat{\sigma}_d$. Thus, $Z_{\alpha/2} < d/\hat{\sigma}_d = 1.527$. This value occurs when $(1-\alpha) \lesssim 0.936$ (for a 2-sided test). Even though there is insufficient evidence to reject the null hypothesis $d_t = 0$ at 95% confidence level, the hypothesis can be rejected at 90% confidence level.

Although Equations (3.86) and (3.87) assume independence between the test sets, Mitchell noted that they are also applicable when both models are evaluated on the same test set.

Comparing the performance of two algorithms (Paired t-test): When comparing the performance of two learning algorithms, it is not sufficient to generate only one model for each algorithm. Instead, one should test the performance of the algorithms by generating multiple models from the data and use the models as the basis for making the comparison.

Suppose we want to compare the performance of two learning algorithms using the k -fold cross-validation method. First, we would divide the data set D into

k equal-sized partitions. Each learning algorithm is then applied to build a model from $k - 1$ of the partitions and test it on the remaining partition. This step is repeated k times, so that each disjoint partition is chosen once as the test set. The cross-validated models for algorithm L_1 are denoted as $M_{11}, M_{12}, \dots, M_{1k}$ and the corresponding models for algorithm L_2 are denoted as $M_{21}, M_{22}, \dots, M_{2k}$. Note that both M_{1i} and M_{2i} are obtained while testing on the same partition i of the data set. Let us denote the error rate for model M_{ij} as e_{ij} .

At each fold j of the cross-validation, we compute the difference in error rate $d_j = e_{1j} - e_{2j}$. If k is sufficiently large, then d_j is normally distributed with mean d_t^{cv} , which is the true error difference, and variance σ^{cv} . Unlike the previous approach, the variance is not known and must be estimated using the standard error:

$$\hat{\sigma}_{d^{cv}}^2 = \frac{\sum_{j=1}^k (d_j - \bar{d})^2}{k(k-1)} \quad (3.88)$$

As a result, we need to use the t -distribution instead of the standardized normal 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 $t_{(1-\alpha), k-1}$ can be obtained from a probability table with two input parameters, its confidence level $(1 - \alpha)$ and its degree of freedom, $k - 1$. The probability table for $t_{(1-\alpha), k-1}$ is shown below.

$k - 1$	$(1 - \alpha)$				
	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

Example 19 Suppose the estimated difference in accuracy of models generated by two learning algorithms have a mean equals to 0.05 and standard deviation equals to 0.002. If the accuracy is estimated using the 30-fold cross-validation approach, then at 95% confidence level, the true accuracy difference is

$$d_t^{cv} = 0.05 \pm 2.04 \times 0.002 = 0.0041 \quad (3.89)$$

Thus, the observed accuracy difference is statistically significant.

3.11 Bibliographic Notes

An excellent treatment of the various classification techniques from a machine learning perspective is given by Mitchell in [129]. Readers can also find an extensive literature on classification from other statistical pattern recognition and data mining books such as [49, 78, 32, 74, 77].

An extensive overview of decision tree learning can be found in the survey articles by Murthy [134], Safavian *et al.* [153], Buntine [26] and Moret [131]. An in-depth discussion of the C4.5 decision tree algorithm is given by Quinlan [149]. The book not only provides a detailed explanation of decision tree growing, pruning, and rule generation, but also the source code of his software. The CART algorithm developed by Breiman *et al.* [21] uses the gini index as its splitting function. For decision tree pruning, a good survey paper on this topic is available in [22] and [53]. However, these techniques require that the entire training data set can fit into the main memory. Recently, there has been considerable interest to develop parallel and more scalable version decision tree algorithms. These techniques include SLIQ [123], SPRINT [161], CMP [192], CLOUDS [11], RainForest [66], and ScalParC [98].

Direct methods for rule-based classification algorithms often employ the sequential covering method to induce the classification rules. Some of the well-known rule-based classification algorithms include RIPPER [37], CN2 [35, 34], 1R [87], AQ [125], RISE [45], and ITRULE [168]. Indirect methods for rule induction using decision trees is described by Quinlan in [149]. Another indirect approach for rule induction is by using neural networks [13]. For rule-based classifiers, the rule antecedent can be generalized to include any propositional or first order logical expression (e.g., Horn clauses). Readers who are interested in first-order logic rule-based classifiers may refer to references such as [129] or the vast literature on inductive logic programming (ILP).

Cover and Hart presented an overview of the nearest neighbor pattern classification method from a Bayesian perspective in [41]. PEBLS [40] is a nearest-neighbor classification algorithm for instances having discrete features. More information about instance-based classifiers is available in [9].

A discussion of the naive Bayes classification algorithm is available in [112, 110, 151, 154]. The naive Bayes approach assumes that the conditional probabilities of the attributes are independent of each other given the class. Even though this assumption is rather unrealistic in many practical applications, the method has worked surprisingly well for applications such as text classification. Another technique known as Bayesian belief network provides a more general approach by allowing attributes to be inter-dependent on each other. The dependencies are often represented in a network structure with conditional probabilities specified between the attributes. A good tutorial on Bayesian belief networks is given in [80].

[190], and [191] are two authoritative books on SVM. Other excellent introductory materials to SVM include the articles by Burges [27], Bennet *et al.* [17], and Hearst [79]. A survey of recent developments in SVM for data mining is given by Mangasarian in [121].

Overfitting and missing values are two practical issues one should address when applying classification techniques to real data. In general, overfitting is related to the size of the hypothesis space explored by the learning algorithm. The larger the hypothesis space, the more likely it is for us to find a model that fits the data purely by chance [95, 46].

3.12 Exercises

1. Give decision trees to represent the following boolean functions:

- (a) A and (not B)
- (b) A or (B and C)
- (c) A xor B
- (d) (A and B) or (C and D)
- (e) A and B and (not C)
- (f) ((not A) and B) or (C and D)
- (g) (A or B) xor (B and C)

2. Consider the following set of training examples:

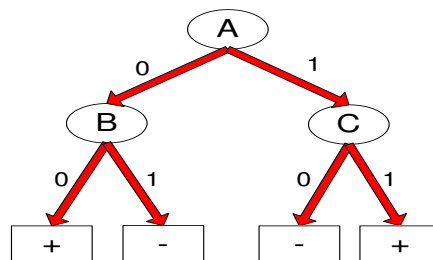
Instance	a_1	a_2	a_3	classification
1	T	T	1.0	+
2	T	T	6.0	+
3	T	F	5.0	-
4	F	F	4.0	+
5	F	T	7.0	-
6	F	T	3.0	-
7	F	F	8.0	-
8	T	F	7.0	+
9	F	T	5.0	-

- (a) What is the entropy of this collection of training examples with respect to the target function classification?
- (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 gains of 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 (among a_1 and a_2) according to classification error rate?

- (f) What is the best split (among a_1 and a_2) according to the gini index?
3. Solve the following questions for the training data tabulated below. (Note: There are total 200 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 taken by C4.5 algorithm. You can use the classification error rate as the criterion used for splitting. Compute the overall error rate of the tree on training data.
- (b) Use variable X as the first splitting attribute, then choose the best available splitting attribute at each of the two successor nodes. Compute the overall error rate of the resultant tree on training data.
- (c) Discuss the results obtained in parts (a) and (b) above. Comment on the suitability of the greedy heuristic used for splitting attribute selection.
4. Consider the decision tree shown in the diagram below.



Training:

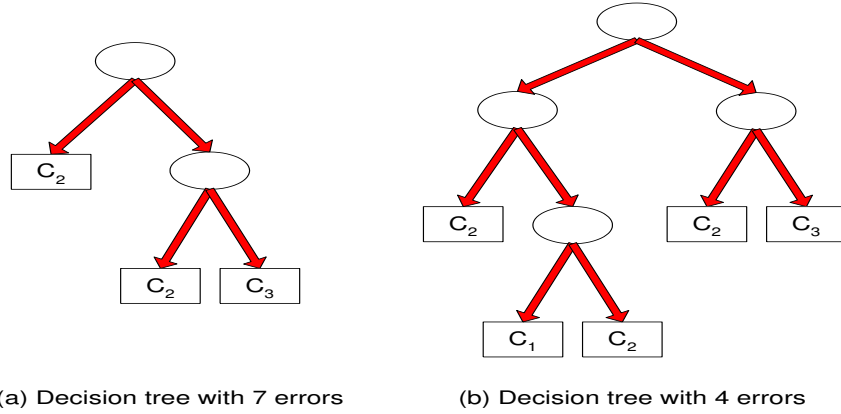
Instance	A	B	C	Class
1	0	0	0	+
2	0	0	1	+
3	0	1	0	+
4	0	1	1	-
5	1	0	0	+
6	1	0	0	+
7	1	1	0	-
8	1	0	1	+
9	1	1	0	-
10	1	1	0	-

Validation:

Instance	A	B	C	Class
11	0	0	0	+
12	0	1	1	+
13	1	1	0	+
14	1	0	1	-
15	1	0	0	+

- (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 given in the lecture notes.
- (c) Compute the generalization error rate of the tree using the reduced error pruning method (with the validation set shown above).
5. Consider the two decision trees shown in the diagram below.



The trees are induced from the training set given in Table 1. Each record in the data set contains 16 binary attributes and can be classified into four different classes, c_1 , c_2 , c_3 , c_4 .

Instance	A1	A2	A3	...	A12	A13	A14	A15	A16	Class
1	1	0	1	...	1	0	0	0	0	c_1
2	1	1	0	...	1	1	1	1	0	c_2
3	0	1	1	...	0	0	0	0	1	c_3
4	0	0	0	...	1	1	0	1	0	c_1
...
...
32	1	0	1	...	1	1	0	1	1	c_4

Compute the total description length of each decision tree according to the minimum description length principle.

- The total description length of a tree is given by:

$$Cost(tree, data) = Cost(tree) + Cost(data|tree)$$

- Each internal node of the tree is encoded by the id of the splitting attribute. If there are m attributes, the cost of encoding each attribute is $\log_2 m$ bits.
- Each leaf is encoded using the id of the class it is associated with. If there are k classes, the cost of encoding a class is $\log_2 k$ bits.

- $Cost(tree)$ is the cost of encoding all the nodes in the tree. To simplify the computation, you can assume that the total cost of the tree is obtained by adding up the cost of encoding each internal node and each leaf node.
- $Cost(data|tree)$ is encoded using the classification errors the tree commits on 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?

6. RIPPER is an extension of an earlier algorithm called IREP by Furnkranz and Widmer (1994). They both use the incremental reduced error pruning scheme to prune the rules generated by the Learn-One-Rule function. Consider the following pair of rules:

R1: $(Refund = No) \wedge (Age \geq 35) \longrightarrow (Evade = no)$

R2: $(Refund = No) \wedge (Marital\ Status = married) \longrightarrow (Evade = yes)$

To determine whether a rule should be pruned, IREP computes the following measure:

$$v_{IREP} = \frac{p + (N - n)}{P + N}$$

where P is the total number of positive examples in the validation set, N is the total number of negative examples in the validation set, p is the number of positive examples in the validation set covered by the rule, and n is the number of negative examples in the validation set covered by the rule.

which is similar to the classification accuracy over the validation data. IREP prefers rules that have higher v_{IREP} .

RIPPER computes the following measure for pruning:

$$v_{RIPPER} = \frac{p - n}{p + n}$$

RIPPER favors rules that have higher v_{RIPPER} .

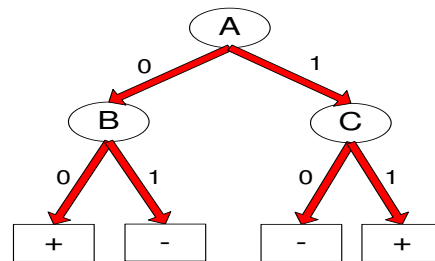
- Consider a validation set that contains 500 positive examples and 500 negative examples. For R1, suppose the number of positive examples covered by the rule is 200, and the number of negative examples covered is 50. For R2, suppose the number of positive examples covered by the rule is 100 and the number of negative examples is 5. Compute v_{IREP} for both rules. Which rule does IREP prefer?
- Compute v_{RIPPER} for the previous problem. Which rule does RIPPER prefer?
- In your opinion, which is the better measure? Explain.

7. C4.5rules is an implementation of an indirect method for generating rules from a decision tree. RIPPER is an implementation of a direct method for generating rules directly from data.
- Discuss the strengths and weaknesses of both methods.
 - Consider a data set that has a large difference in the class size (i.e., some classes are much bigger than others). We would like to find good rules (high accuracy rules) for the small classes. Which method (between C4.5rules and RIPPER) would be better? Please provide supporting arguments.
8. (a) Suppose the proportion of undergraduate students who smoke is 15% and the proportion of graduate students who smoke is 23%. If one-fifth of the college students are graduate students, and the rest of them are undergraduates, what is the probability that a college student who smoke is a graduate student?
- (b) Given the information in the previous section, if you had randomly selected one of the college students, is he/she most likely a graduate or undergraduate student?
- (c) Suppose you noticed that the student is a smoker, would you change your answer to the previous question?
- (d) Suppose 30% of the graduate students live in the dorm but only 10% of the undergraduate students live in the dorm. If you noticed that the student is a smoker and lives in the dorm, is he/she most likely a graduate or undergraduate student? You can assume independence between students who live in the dorm and those who smoke. (Hint: Use the naive Bayes approach to compute the conditional probability.)
9. Consider a table containing the following instances:

Instance	<i>A</i>	<i>B</i>	<i>C</i>	Class
1	0	0	0	+
2	0	0	1	−
3	0	1	1	−
4	0	1	1	−
5	0	0	1	+
6	1	0	0	+
7	1	0	1	−
8	1	0	1	−
9	1	1	0	+
10	1	0	1	+

- Estimate the conditional probabilities ($P(+|A)$, $P(+|B)$, $P(+|C)$, $P(-|A)$, $P(-|B)$, and $P(-|C)$) by using simple fraction for all binary values of A , B , and C .
- Use the estimate of conditional probabilities given in the previous question to predict the class label for the test sample ($A = 0, B = 1, C = 0$) using the naive Bayes approach.
- Find estimates for conditional probabilities using the m-estimate probability approach. Use $p=1/4$ and $m=4$.
- Repeat problem (b) above using the estimates of conditional probabilities given in problem (c).
- Compare the two methods for computing estimates for conditional probabilities in the above question. Which method is better and why?

10. Consider the decision tree shown in the diagram below.



Training:

Instance	A	B	C	Class
1	0	0	0	+
2	0	0	1	+
3	0	1	0	+
4	0	1	1	-
5	1	0	0	+
6	1	0	0	+
7	1	1	0	-
8	1	0	1	+
9	1	1	0	-
10	1	1	0	-

Validation:

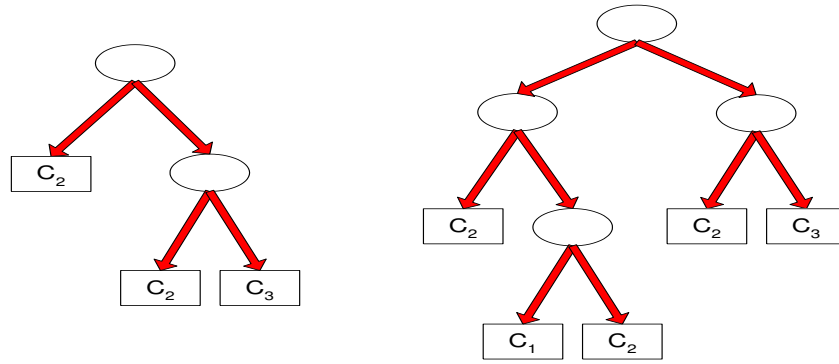
Instance	A	B	C	Class
11	0	0	0	+
12	0	1	1	+
13	1	1	0	+
14	1	0	1	-
15	1	0	0	+

- Compute the generalization error rate of the tree using the optimistic approach.
- Compute the generalization error rate of the tree using the pessimistic approach given in the lecture notes.
- Compute the generalization error rate of the tree using the reduced error pruning method (with the validation set shown above).

11. Consider the two decision trees shown in the diagram below.

The trees are induced from the training set given in Table 3.8. Each record in the data set contains 16 binary attributes and can be classified into four different classes, c_1 , c_2 , c_3 , c_4 .

Compute the total description length of each decision tree according to the minimum description length principle.



(a) Decision tree with 7 errors

(b) Decision tree with 4 errors

Table 3.8. Data set for Problem 2.

Instance	A1	A2	A3	...	A12	A13	A14	A15	A16	Class
1	1	0	1	...	1	0	0	0	0	c_1
2	1	1	0	...	1	1	1	1	0	c_2
3	0	1	1	...	0	0	0	0	1	c_3
4	0	0	0	...	1	1	0	1	0	c_1
...
...
32	1	0	1	...	1	1	0	1	1	c_4

- The total description length of a tree is given by:

$$Cost(tree, data) = Cost(tree) + Cost(data|tree)$$

- Each internal node of the tree is encoded by the id of the splitting attribute. If there are m attributes, the cost of encoding each attribute is $\log_2 m$ bits.
- Each leaf is encoded using the id of the class it is associated with. If there are k classes, the cost of encoding a class is $\log_2 k$ bits.
- $Cost(tree)$ is the cost of encoding all the nodes in the tree. To simplify the computation, you can assume that the total cost of the tree is obtained by adding up the cost of encoding each internal node and each leaf node.
- $Cost(data|tree)$ is encoded using the classification errors the tree commits on 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?

12. RIPPER is an extension of an earlier algorithm called IREP by Furnkranz and Widmer (1994). They both use the incremental reduced error pruning scheme

to prune the rules generated by the Learn-One-Rule function. Consider the following pair of rules:

- R1: $(Refund = No) \wedge (Age \geq 35) \longrightarrow (Evade = no)$
 R2: $(Refund = No) \wedge (Marital\ Status = married) \longrightarrow (Evade = yes)$

To determine whether a rule should be pruned, IREP computes the following measure:

$$v_{IREP} = \frac{p + (N - n)}{P + N}$$

where P is the total number of positive examples in the validation set, N is the total number of negative examples in the validation set, p is the number of positive examples in the validation set covered by the rule, and n is the number of negative examples in the validation set covered by the rule. This measure is similar to classification accuracy over the validation data. IREP prefers rules that have higher v_{IREP} .

RIPPER computes the following measure for pruning:

$$v_{RIPPER} = \frac{p - n}{p + n}$$

RIPPER favors rules that have higher v_{RIPPER} .

- (a) Consider a validation set that contains 500 positive examples and 500 negative examples. For R1, suppose the number of positive examples covered by the rule is 200, and the number of negative examples covered is 50. For R2, suppose the number of positive examples covered by the rule is 100 and the number of negative examples is 5. Compute v_{IREP} for both rules. Which rule does IREP prefer?
 - (b) Compute v_{RIPPER} for the previous problem. Which rule does RIPPER prefer?
 - (c) In your opinion, which is the better measure? Explain.
13. C4.5rules is an implementation of an indirect method for generating rules from a decision tree. RIPPER is an implementation of a direct method for generating rules directly from data.
 - (a) Discuss the strengths and weaknesses of both methods.
 - (b) Consider a data set that has a large difference in the class size (i.e., some classes are much bigger than others). We would like to find good rules (high accuracy rules) for the small classes. Which method (between C4.5rules and RIPPER) would be better? Please provide supporting arguments.

14. (a) Suppose the proportion of undergraduate students who smoke is 15% and the proportion of graduate students who smoke is 23%. If one-fifth of the college students are graduate students, and the rest of them are undergraduates, what is the probability that a college student who smoke is a graduate student?
- (b) Given the information in the previous section, if you had randomly selected one of the college students, is he/she most likely a graduate or undergraduate student?
- (c) Suppose you noticed that the student is a smoker, would you change your answer to the previous question?
- (d) Suppose 30% of the graduate students live in the dorm but only 10% of the undergraduate students live in the dorm. If you noticed that the student is a smoker and lives in the dorm, is he/she most likely a graduate or undergraduate student? You can assume independence between students who live in the dorm and those who smoke. (Hint: Use the naive Bayes approach to compute the conditional probability.)
15. Consider a table containing the following instances:

Instance	A	B	C	Class
1	0	0	0	+
2	0	0	1	−
3	0	1	1	−
4	0	1	1	−
5	0	0	1	+
6	1	0	0	+
7	1	0	1	−
8	1	0	1	−
9	1	1	0	+
10	1	0	1	+

- (a) Estimate the conditional probabilities ($P(A|+)$, $P(B|+)$, $P(C|+)$, $P(A|−)$, $P(B|−)$, and $P(C|−)$) by using simple fraction for all binary values of A , B , and C .
- (b) Use the estimate of conditional probabilities given in the previous question to predict the class label for the test sample ($A = 0, B = 1, C = 0$) using the naive Bayes approach.
- (c) Find estimates for conditional probabilities using the m-estimate (i.e., Laplace correction) probability approach. Use $p=1/4$ and $m=4$.
- (d) Repeat problem (b) above using the estimates of conditional probabilities given in problem (c).

- (e) Compare the two methods for computing estimates for conditional probabilities in the above question. Which method is better and why?

16. Consider the data set shown in Table 3.9:

Table 3.9. Data set for Problem 7.

Instance	A	B	C	Class
1	0	0	1	–
2	1	0	1	+
3	0	1	0	–
4	1	0	0	–
5	1	0	1	+
6	0	0	1	+
7	1	1	0	–
8	0	0	0	–
9	0	1	0	+
10	1	1	1	+

- (a) Estimate the conditional probabilities for $P(A = 1|+)$, $P(B = 1|+)$, $P(C = 1|+)$, $P(A = 1|–)$, $P(B = 1|–)$, and $P(C = 1|–)$ using the same approach as the previous problem.
- (b) Use the estimate of conditional probabilities given in the previous question to predict the class label for the test sample $(A = 1, B = 1, C = 1)$ using the naive Bayes approach.
- (c) Compare $P(A = 1)$, $P(B = 1)$ and $P(A = 1, B = 1)$. Are there any obvious relationships between these probabilities?
- (d) Repeat your analysis for part (c) using $P(A = 1)$, $P(B = 0)$, and $P(A = 1, B = 0)$. If you observe an obvious relationship in part (c), does it still hold?
- (e) Compare $P(A = 1, B = 1|Class = +)$ against $P(A = 1|Class = +)$ and $P(B = 1|Class = +)$. Are the two variables conditionally independent of each other? Compare the relationship between A and B in part (c) against the relationship you observe after conditioning on $Class = +$.
17. You are asked to evaluate the performance of two classification models, M_1 and M_2 . The test set you have chosen contains 26 binary attributes, labeled as A through Z . The first model, M_1 , was obtained using the naive Bayes approach while the second model, M_2 , was generated from RIPPER.

After applying the models to the test instances, you obtained the above posterior probability estimates for each instance. (We only show the posterior probabilities for the positive class. As this is a two-class problem, $P(–) = 1 - P(+)$)

Instance	A	B	\dots	Z	$Class$
1	0	0	\dots	1	+
2	1	0	\dots	1	+
3	0	1	\dots	0	−
4	1	0	\dots	0	−
5	1	0	\dots	1	+
6	0	0	\dots	1	+
7	1	1	\dots	0	−
8	0	0	\dots	0	−
9		1	\dots	1	+
10	1	1	\dots	1	−

and $P(-|A, \dots, Z) = 1 - P(+|A, \dots, Z)$.) Note that the class label we are interested in is the positive class.

Instance	True Class	$P(+ A, \dots, Z, M_1)$	$P(+ A, \dots, Z, M_2)$
1	+	0.73	0.61
2	+	0.69	0.03
3	−	0.44	0.68
4	−	0.55	0.31
5	+	0.67	0.45
6	+	0.47	0.09
7	−	0.08	0.38
8	−	0.15	0.05
9	+	0.45	0.01
10	−	0.35	0.04

- Explain how the posterior probability estimates can be obtained using RIPPER.
- Plot the ROC curve for both M_1 and M_2 . (You should plot them on the same graph.) Which model do you think is better? Explain your reasons.
- For model M_1 (naive Bayes), suppose you choose the cutoff threshold to be $t = 0.5$. In other words, any test instances whose posterior probability is greater than t will be classified as a positive example. Compute the precision, recall, and F-measure for the model at this threshold value.
- Repeat the analysis for part (c) using the same cutoff threshold on model M_2 . Compare the F-measure results for both models. Which model is better? Are the results consistent with what you expect from the ROC curve?
- Repeat part (c) for model M_1 using the threshold $t = 0.1$. Which threshold do you prefer, $t = 0.5$ or $t = 0.1$? Are the results consistent with

what you expect from the ROC curve. State your reasons clearly.

18. You have been chosen as one of the final judges of a data mining competition. After careful deliberations, three finalists were selected and you are asked to give your opinion on which of the three finalists should be declared as the overall winner.

In the competition, the participants were asked to build a classification model for three different data sets, D_1 , D_2 , and D_3 . The properties of each data set are summarized in the table below:

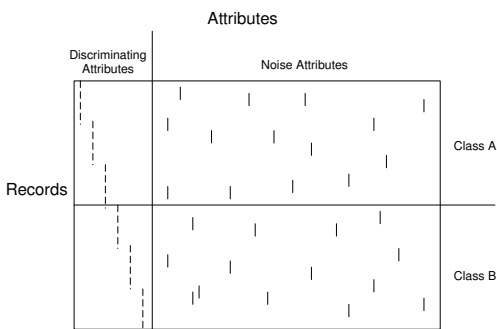
Data set	Number of records	Number of attributes	Number of Classes	% support for the majority class
D_1	50	5	2	90%
D_2	200	20	2	75%
D_3	50000	100	2	50%

The finalists were asked to submit the results of their model accuracies for each data set and describe the method they had used to evaluate their model. (Assume that all three finalists are honest about the methods they had used.) The first finalist (M_1) uses a rule-based system with stratified 10-fold cross validation strategy. The second finalist (M_2) uses a decision tree model, also with stratified 10-fold cross validation strategy. The third finalist (M_3) uses the 1-nearest neighbor approach with the holdout model evaluation method (with two-third of the data for training, and the remaining one-third for testing). The accuracies of their models are summarized in the table below:

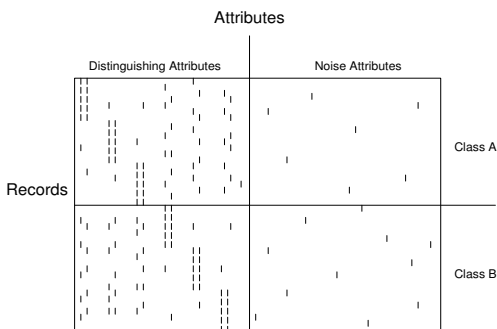
Data set	M_1	M_2	M_3
Accuracy on D_1	55.4%	59.1%	89.5%
Accuracy on D_2	57.3%	57.4%	57.7%
Accuracy on D_3	57.5%	54.5%	50.2%

- Which model do you think is better, M_1 or M_2 ? Please state your reasons clearly.
- Which model do you think is better, M_1 or M_3 ? Please state your reasons clearly.
- Which model do you think is better, M_2 or M_3 ? Please state your reasons clearly.
- Based on your reasons above, which method do you think deserve to win the competition?

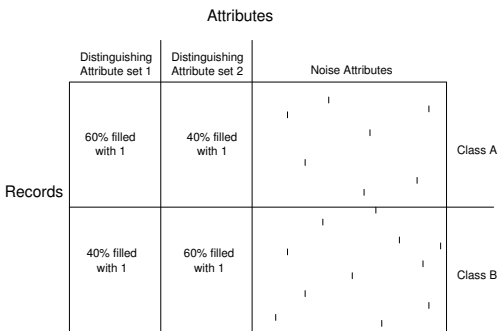
19. Given data sets shown in Figures 3.48, explain how different classification algorithms (decision trees, Naive Bayesian, and k-nearest neighbor) perform on these data sets.
20. Naive Bayes classifier does not work well in data sets having multiple modality.
 - (a) Explain how Naive Bayesian would perform on the data set shown in Figure 3.49.
 - (b) If each class is further divided such that there are four classes ($A1$, $A2$, $B1$, and $B2$), will Naive Bayes perform better? Please explain.
 - (c) How will decision tree perform on this data set (for the two-class problem)? What if there are four classes?



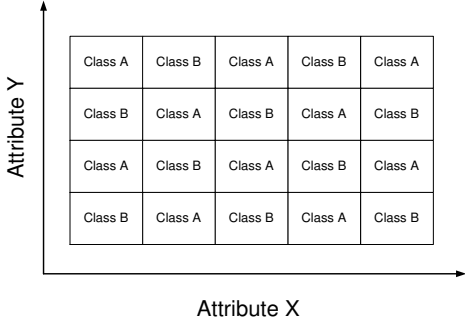
(a) Synthetic data set 1.



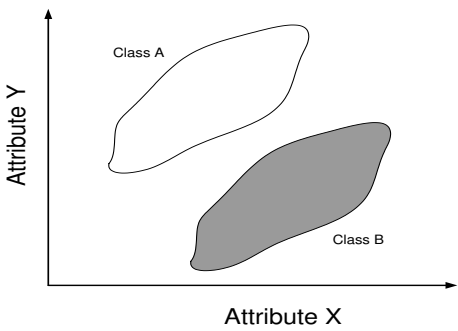
(b) Synthetic data set 2.



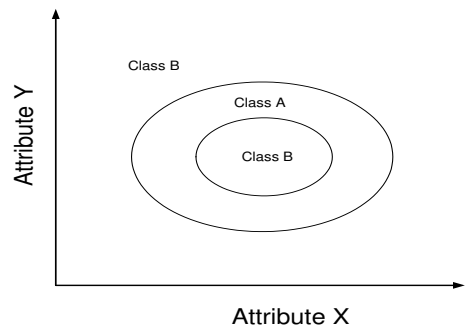
(c) Synthetic data set 3.



(d) Synthetic data set 4



(e) Synthetic data set 5.



(f) Synthetic data set 6.

Figure 3.48. Data set for Question 19.

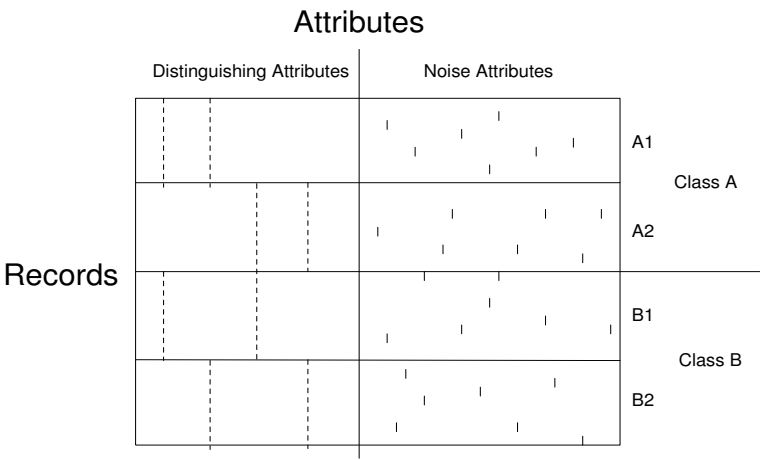


Figure 3.49. Data set for Question 20.

Association Analysis

A key objective of data mining is to discover relationships that are hidden in large data repositories. For example, grocery store retailers are interested to know which items are frequently sold together to their customers. Knowing such relationships can help retailers to promote these items together in order to improve the overall sales. Table 4.1 illustrates an example of the grocery store data, also known as *market-basket transactions*. Each row in this table corresponds to a transaction, which is uniquely identified by its transaction identifier, *Tid*, and contains the set of items bought by a particular customer. The data in this table suggests a strong relationship exists between the sale of bread and milk since most of the customers who buy bread also end up buying milk as well.

Table 4.1. An example of market-basket transactions.

<i>Tid</i>	Items
1	{ Bread , Milk}
2	{ Bread , Diaper, Beer, Eggs}
3	{ Milk , Diaper, Beer, Coke}
4	{ Bread , Milk , Diaper, Beer}
5	{ Bread , Milk , Diaper, Coke}

One way to find such relationships is to apply standard statistical techniques such as correlation analysis to determine the extent to which one item is associated with another. However, these techniques have several known limitations when applied to high-dimensional data sets, as will be explained in Section ????. In this chapter, we introduce a data mining technique called *association analysis* for identifying hidden relationships automatically from large transaction databases. These relationships are represented in the form of *association rules*, which are implication rules used to

predict the presence of certain items in a transaction based on the presence of other items. For example, the rule

$$\{Diaper\} \longrightarrow \{Beer\}$$

suggests that most of the customers who buy diaper at a grocery store also tend to buy beer. If indeed such unexpected relationships are found, then they can be used to identify new cross-selling opportunities or to design an effective layout for the grocery store.

Typically, there are exponentially many rules that can be extracted from a market basket data. Some of these rules might be spurious because they can happen purely by chance while other rules, such as the milk and bread example, are simply not interesting because they are well-known to the domain experts. As a result, the key challenges of mining association rules are two-fold: (1) to design an efficient algorithm for mining association rules in large data sets, and (2) to develop an effective approach for distinguishing interesting rules from spurious ones. Both issues are examined in detail in this chapter.

Association analysis is also applicable to numerous other application domains such as Web traffic analysis, document analysis, bioinformatics, computational chemistry, and fault diagnosis in telecommunication networks. The nature of the data in some of these domains is quite different than that of traditional market basket data. For example, Web traffic data tends to have a temporal nature, bioinformatics data tends to have a sequential nature, while computational chemistry data often contains graph structures. Handling such diverse data sets may require the development of new techniques for extending the capabilities of association analysis. We describe some of these extensions later in this chapter.

4.1 Problem Definition

We begin this section with a few basic definitions, followed by a description of the association rule mining problem. Let $I = \{i_1, i_2, \dots, i_d\}$ be the set of all items and $T = \{t_1, t_2, \dots, t_N\}$ be the set of all market-basket transactions.

Binary Representation The market-basket data is assumed to be represented in a binary format as shown in Table 4.2, where each row corresponds to a transaction and each column corresponds to an item. An item can be viewed as a binary variable — if it belongs to a particular transaction, then its corresponding column entry is denoted as 1; otherwise, it is denoted as 0. The presence of an item in a transaction is also considered to be more important than its absence, henceforth, it is an asymmetric binary variable. The *width* of a transaction is given by the number of non-vanishing entries in the corresponding row. 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 sale price of each item. We will describe ways for handling such non-binary data in Section 4.4.

Table 4.2. A binary 0/1 representation of market-basket data.

TID	Bread	Milk	Diaper	Beer	Eggs	Coke
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

Itemset and Support Count The fundamental unit of association rule is an *itemset*, which is a set consisting of one or more items. If an itemset contains k items, it is called a k -itemset. For example, $\{Milk, Diaper, Beer\}$ is a 3-itemset. An important property of an itemset is how frequently it appears in the data set — the more frequent it is, the more significant is the relationship. We define the frequency or *support count* of an itemset C as the number of transactions containing C . Mathematically speaking, the support count is given by the following expression:

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

where $|\{\cdot\}|$ denotes the number of elements that belong to a given set. As an example, the support count for the itemset $\{Diaper, Beer\}$ is equals to three because there are three transactions in Table 4.2 that contain both diaper and beer.

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 is determined by the *support* and *confidence* metrics. Support measures the fraction of transactions that contain all items belonging to the set $X \cup Y$. Confidence measures the fraction of times Y is present in transactions that contain X . Formally, these metrics are defined as follows:

$$\begin{aligned} \text{support, } s(X \longrightarrow Y) &= \frac{\sigma(X \cup Y)}{N} \quad \text{and} \\ \text{confidence, } c(X \longrightarrow Y) &= \frac{\sigma(X \cup Y)}{\sigma(X)}, \end{aligned} \tag{4.1}$$

where N is the total number of transactions.

Example 20 Consider the rule “customers who buy milk and diaper also tend to buy beer”. This rule can be expressed as $\{Milk, Diaper\} \longrightarrow \{Beer\}$. Since the support count for the itemset $\{Milk, Diaper, Beer\}$ is equal to 2 and the total number of transactions is 10, the support for this rule is $2/10 = 0.2$. The confidence for this rule is obtained by dividing the support count for the itemset $\{Milk, Diaper, Beer\}$, which is equal to 2, to the support count for the itemset $\{Milk, Diaper\}$, which is equal to 3. Therefore, the confidence of this rule is $2/3 = 0.67$.

Why Use Support and Confidence? Support is an important measure because it reflects the significance of a rule. Rules that have very low support are rarely observed and thus, are more likely to be the result of chance occurrence. This has led to the development of many association rule mining algorithms that rely on the support measure to eliminate spurious rules. (We will describe the advantage of using the support measure from a computational perspective in Section ??.)

Confidence is another useful measure because it reflects the reliability of the prediction made by the rule. Given an association rule $X \longrightarrow Y$, the higher is the confidence, the more likely it is for Y to be present in transactions that contain X . From a statistical perspective, confidence can be regarded as an estimate of the conditional probability for Y given X .

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

Given a set of transactions T , find all rules having support $\geq \text{minsup}$ and confidence $\geq \text{minconf}$, where minsup and minconf are the minimum support and minimum confidence thresholds, respectively.

A naive approach for generating association rules is to enumerate all possible rule combinations and then compute their respective support and confidence values. For a data set containing d items, the total number of possible association rules is

$$R = 3^d - 2^{d+1} + 1 \quad (4.2)$$

The proof of this is left as an exercise. From this equation, note that even if the data set contains only six items, the naive approach must enumerate all $3^6 - 2^7 + 1 = 602$ possible rules. Furthermore, for the data set shown in Table 4.1, if $\text{minsup} = 20\%$ and $\text{minconf} = 50\%$, there are only 102 association rules that satisfy both minimum support and minimum confidence requirements. In other words, nearly 83% of the association rules generated using the naive approach are eliminated once their support and confidence values are known. Therefore, a key computational challenge for mining association rules is to discover rules that satisfy both support and confidence thresholds without having to list all their possible combinations.

We can optimize the amount of computation needed for association rule mining by decoupling the support and confidence requirements. Notice that the support of an association rule $X \longrightarrow Y$ depends only on the support of its corresponding itemset, $X \cup Y$ (see equation 4.1). For example, the support for the rules

$$\{\text{Beer}, \text{Diaper}\} \longrightarrow \{\text{Milk}\},$$

$$\{\text{Beer}\} \longrightarrow \{\text{Diaper}, \text{Milk}\},$$

$$\{\text{Milk}\} \longrightarrow \{\text{Beer}, \text{Diaper}\}$$

are identical because they correspond to three different ways of partitioning the same itemset, $\{\text{Beer}, \text{Diaper}, \text{Milk}\}$. If the support of an itemset is less than minsup ,

then we do not have to enumerate its corresponding rules since they are guaranteed to be infrequent. This strategy of finding itemsets that have high support first before generating association rules has been adopted by many association rule generation algorithms. Such algorithms would decompose the association rule generation problem into two major subtasks:

Frequent Itemset Generation Find all itemsets that satisfy the *minsup* threshold. These itemsets are called *frequent itemsets*.

Rule Generation Extract all high confidence association rules from the frequent itemsets found in the previous step. These rules are called *strong* rules.

Among these two steps, frequent itemset generation tends to be much more compute-intensive. We will describe techniques for generating frequent itemsets efficiently in the next section. Once the frequent itemsets have been found, techniques for extracting association rules will be described in Section 4.3.

4.2 Frequent Itemset Generation

A lattice structure can be used to enumerate the list of all possible itemsets. For example, Figure 4.1 illustrates itemsets that are derivable from the set $\{A, B, C, D, E\}$. Some of the itemsets can be frequent, depending on the choice of minimum support threshold. In general, a data set that contains d items may generate up to $2^d - 1$ frequent itemsets, excluding the null set. Because d can be very large in many commercial databases, frequent itemset generation is a computationally expensive task.

A naive approach is to consider each itemset in the lattice as a *candidate* frequent itemset. Next, the support count for each candidate is obtained by matching the candidate against every transaction, an operation that is shown in Figure 4.2. If the candidate is contained within a transaction, its support count will be incremented. Such a counting method is extremely expensive because it requires $O(NM)$ matching operations, where N is the number of transactions and M is the number of candidate itemsets. In turn, each matching operation can take up to $O(w)$ computations, where w is the maximum width of a transaction.

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

1. *Eliminate some of the candidate itemsets (M).* The Apriori principle, to be described in the next section, is an intelligent way to eliminate some of these candidate itemsets before counting their actual support values.
2. *Reduce the number of transactions (N).* As the size of the candidate itemsets increases, fewer transactions will be supported by the itemsets. For instance, since the width of the first transaction in Table 4.1 is 2, it would be advantageous to remove this transaction before searching for frequent itemsets of size 3 and larger.

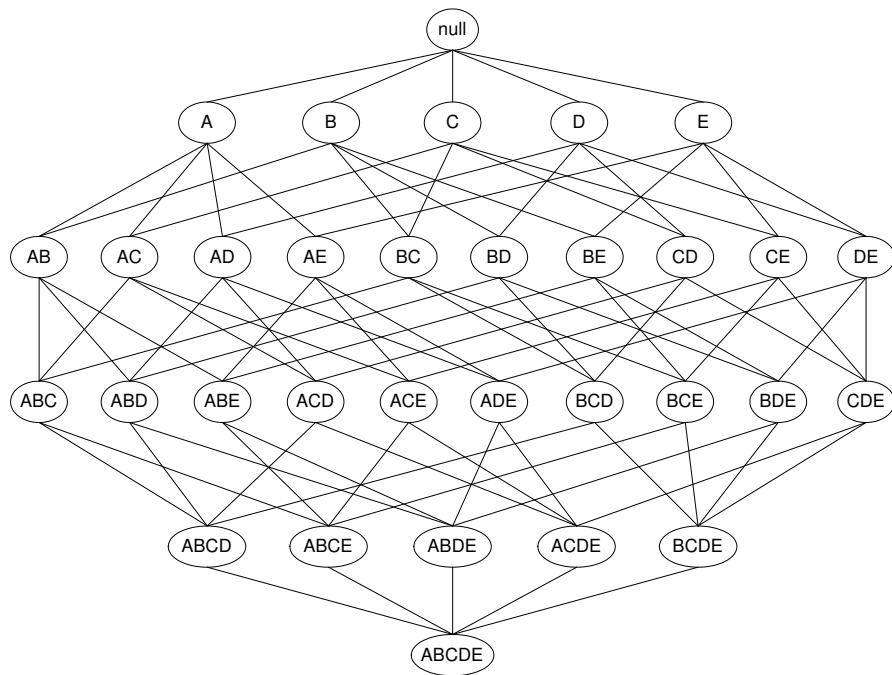


Figure 4.1. The Itemset Lattice.

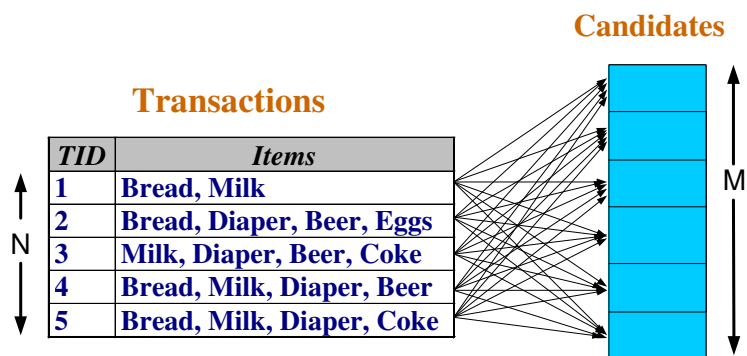


Figure 4.2. Counting the support of candidate itemsets.

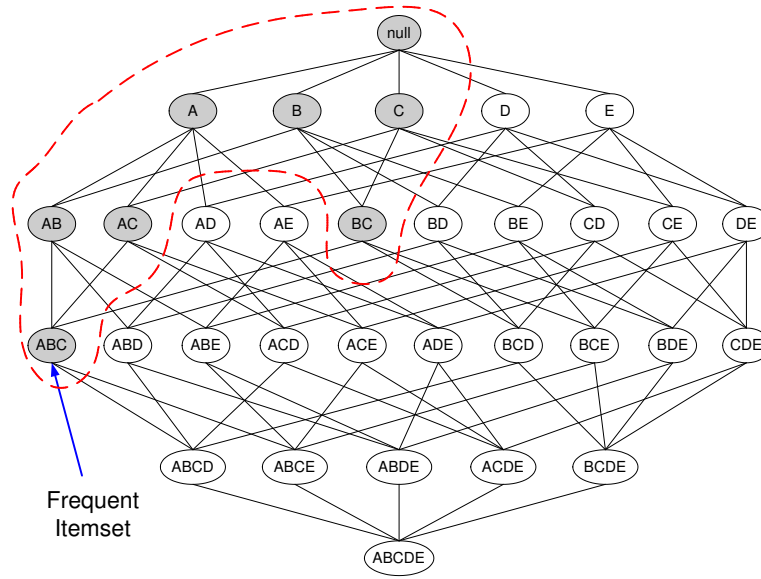


Figure 4.3. An illustration of the Apriori principle. If $\{A, B, C\}$ is frequent, then all subsets of this itemset are frequent.

3. *Reduce the number of candidate matching operations.* Instead of matching each candidate itemset against every transaction, various strategies can be used to reduce the number of comparisons. This strategy is often implemented by using a variety of advanced data structures to store the candidate itemsets or the transaction database. We will discuss two such approaches in Sections 4.2.4 and 4.2.6.

4.2.1 Apriori Principle

Many frequent itemset generation algorithms adopt the following principle to reduce the number of candidate itemsets.

Theorem 1 [Apriori Principle]: *If an itemset is frequent, then all of its subsets must also be frequent.*

As an illustration of this principle, consider the lattice shown in Figure 4.3. If an itemset such as $\{A, B, C\}$ is found to be frequent, then the Apriori principle suggests that all subsets of this itemset (the shaded itemsets in this figure) must also be frequent. Intuitively, any transaction that contains $\{A, B, C\}$ must also contain $\{A, B\}$, $\{A, C\}$, $\{B, C\}$, $\{A\}$, $\{B\}$, and $\{C\}$, i.e., all the non-empty subsets of the 3-itemset. As a result, if the support for $\{A, B, C\}$ is greater than the minimum support threshold, so must its subsets be.

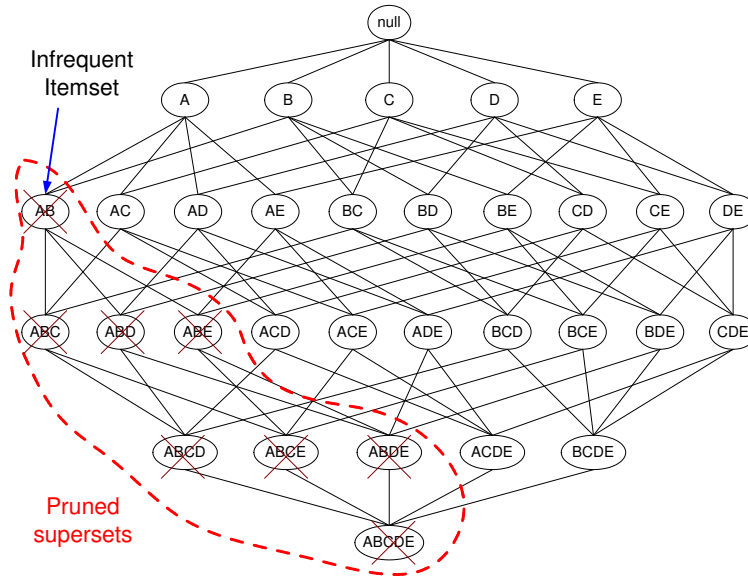


Figure 4.4. An illustration of support-based pruning. If $\{A, B\}$ is infrequent, then all supersets of $\{A, B\}$ are eliminated.

Conversely, if the support of an itemset such as $\{A, B\}$ is infrequent, then all supersets of this itemset must also be infrequent, as illustrated in Figure 4.4. Consequently, the entire subgraph containing supersets of $\{A, B\}$ can be pruned immediately once we know that $\{A, B\}$ is known to be infrequent. This strategy of trimming the search space based on the support measure can be used to reduce the number of candidate itemsets substantially and is known as *support-based pruning*.

The above discussion illustrates a key property of the support measure, namely, that the support for an itemset cannot exceed the support for any one of its subsets. This property is also known as the *anti-monotone* property of the support measure. In general, the monotonicity property of a measure f can be formally defined as follows:

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

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

which means that if X is a subset of Y , then $f(X)$ must not exceed $f(Y)$. Conversely, the measure f is *anti-monotone* (or downward closed) if:

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

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

The anti-monotone property of a measure is useful because it can be incorporated directly into the mining algorithm for pruning candidate patterns, as will be shown

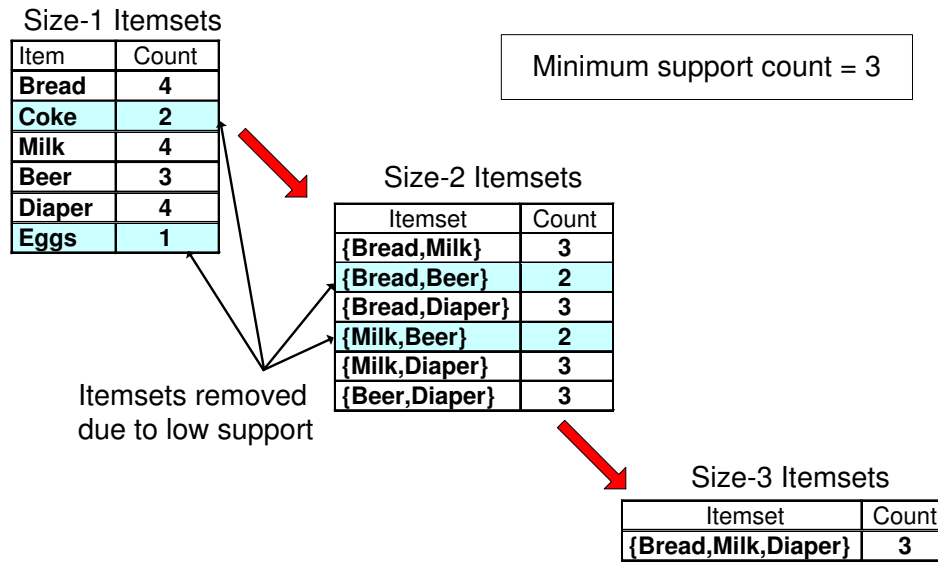


Figure 4.5. Illustration of Apriori algorithm

in the next section.

4.2.2 Apriori Algorithm

The *Apriori* algorithm is the first algorithm that pioneered the use of support-based pruning to systematically control the exponential growth in the number of candidate itemsets during frequent itemset generation. Figure 4.5 gives a high level illustration of the *Apriori* algorithm for the market-basket transactions shown in Table 4.1. The algorithm starts by finding all frequent 1-itemsets, i.e., individual items having support $\geq \text{minsup}$. Assuming that minsup is 60%, the 1-itemsets {Coke} and {Eggs} are removed because they do not have sufficient support. The remaining frequent 1-itemsets are then used to generate candidate itemsets of size 2. In this example, there are ${}^4C_2 = 6$ candidate 2-itemsets generated from the 4 frequent 1-itemsets. When the support for these candidate itemsets are counted, the itemsets {Bread, Beer} and {Milk, Beer} are found to be infrequent. The four remaining frequent 2-itemsets will be used to generate candidate itemsets of size 3 in the next iteration. A candidate itemset is kept only if all of its proper subsets are frequent. For example, {Bread, Milk, Diaper} is kept as a candidate 3-itemset because all of its subsets are frequent. In contrast, a 3-itemset such as {Milk, Diaper, Beer} is pruned because one of its proper subsets, {Milk, Beer}, is infrequent.

A glimpse of how effective is the *Apriori* pruning strategy can be seen by looking at the number of candidates that need to be generated if every subset up to size-3 is to be considered for counting. The naive strategy of generating all itemsets would create ${}^6C_1 + {}^6C_2 + {}^6C_3 = 41$ candidates. As shown in the figure, the use of the Apriori principle reduces this number down to ${}^6C_1 + {}^4C_2 + 1 = 13$.

Table 4.3. The Apriori algorithm.

```

1.  $F_1 = \{ \text{frequent 1-itemsets} \}$  ;
2. for (  $k = 2$ ;  $F_{k-1} \neq \emptyset$ ;  $k = k + 1$  ) {
3.    $C_k = \text{apriori\_gen}(F_{k-1})$ 
4.   for all transactions  $t \in T$  {
5.      $C_t = \text{subset}(C_k, t)$ 
6.     for all candidates  $c \in C_t$ 
7.        $c.\text{count}++$ 
8.   }
9.    $F_k = \{c \in C_k \mid c.\text{count}/N \geq \text{minsup}\}$ 
10. }
11. Answer =  $\bigcup F_k$ 

```

The pseudo code for the *Apriori* algorithm is shown in Table 4.2.2. Let C_k and F_k denote the set of candidate itemsets and frequent itemsets of size k , respectively. A high-level description of the algorithm is presented below.

- Initially, the database is scanned once to obtain F_1 , the set of all frequent 1-itemsets.
- The algorithm would iteratively look for larger-sized frequent itemsets by using a level-wise procedure that (a) generates candidate itemsets using the frequent itemsets found in the previous pass (step 3), (b) counts the support for each candidate by scanning the database (steps 4-8), and (c) identifies the frequent itemsets by comparing the support count of each candidate against the *minsup* threshold (step 9).
- The algorithm terminates when no new frequent itemset is found in the previous pass, i.e., $F_{k-1} = \emptyset$.

The candidate generation and support counting steps are described in further details in the next sections.

4.2.3 Candidate Itemset Generation

There are two major steps involved in candidate generation:

Join Step: During the join step, frequent itemsets found in the $(k - 1)$ th iteration are used to generate candidate itemsets of size k . For example, a pair of frequent itemsets, $\{Bread, Milk\}$ and $\{Bread, Diaper\}$, can be merged together to obtain a candidate 3-itemset $\{Bread, Milk, Diaper\}$. A brute-force approach for doing this is to join every pair of frequent $(k-1)$ -itemsets

but this would require $O(|F_{k-1}| \times |F_{k-1}|)$ join operations. This approach also has several other limitations. First, some of the candidates generated may have a size larger than k . For example, merging the frequent 2-itemsets $\{Bread, Milk\}$ with $\{Diaper, Beer\}$ will produce a candidate itemset of size-4, $\{Bread, Milk, Diaper, Beer\}$, which will have to be eliminated since we are only interested in candidate itemsets of size 3. To overcome this problem, we may restrict the join operation only to pairs of frequent $k - 1$ -itemsets having $k - 2$ items in common. Second, we might end up generating the same candidates more than once. For example, the candidate 3-itemset $\{Bread, Milk, Diaper\}$ can be produced by combining the frequent 2-itemsets $\{Bread, Milk\}$ with $\{Bread, Diaper\}$ or $\{Bread, Milk\}$ with $\{Diaper, Milk\}$. To avoid generating duplicate candidates, we can order the items within an itemset lexicographically, and then join the frequent $(k - 1)$ -itemsets only if their first $k - 2$ items are identical. Going back to the previous example, the frequent itemsets $\{Bread, Milk\}$ and $\{Bread, Diaper\}$ can be merged together because they share a common prefix item, which is Bread. On the other hand, the frequent itemsets $\{Bread, Milk\}$ and $\{Diaper, Milk\}$ should not be merged because their prefix items are different, i.e., Bread versus Diaper. This strategy ensures that the same candidate will not be reproduced when merging pairs of frequent itemsets.

Prune step: After the candidate itemsets are created through the join step described previously, a pruning step is applied to remove any candidate whose subsets are infrequent.

Example 21 Let $F_3 = \{\{A, B, C\}, \{A, B, D\}, \{A, B, E\}, \{A, C, E\}, \{A, D, E\}, \{B, D, E\}\}$ be the set of frequent 3-itemsets. The join step will produce the following candidate 4-itemsets, $C_4 = \{\{A, B, C, D\}, \{A, B, C, E\}, \{A, B, D, E\}\}$. However, after applying the pruning step, only $\{A, B, D, E\}$ survives as a candidate itemset in C_4 . The candidate $\{A, B, C, D\}$ is pruned because some of its subsets, such as $\{A, C, D\}$ and $\{B, C, D\}$, do not belong to F_3 . Similarly, the itemset $\{A, B, C, E\}$ is pruned because one of its subsets, $\{B, C, E\}$, is infrequent.

Only candidate itemsets that survive the pruning step will be counted for their support. We will discuss the support counting step in the next section.

4.2.4 Support Counting

As previously noted, a naive way for doing support counting is to simply match each transaction against every candidate itemset; a task that can be extremely time-consuming especially when both N and M are large (Figure 4.2). A more efficient approach is to use a data structure for indexing and storing the candidate itemsets. In the *Apriori* algorithm, the candidates are partitioned into different buckets and kept in a hash tree structure. During support counting, each transaction is also

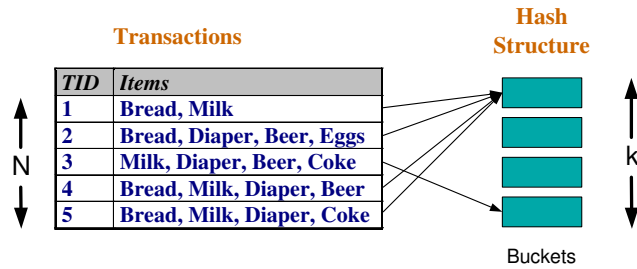


Figure 4.6. Counting the support of itemsets using hash structure.

hashed into its appropriate buckets. That way, instead of matching each transaction to every candidate itemset, one would match the transaction only to candidates that belong to the same bucket, as shown in Figure 4.6.

Hash Tree Structure

Figure 4.7 illustrates an example of a hash tree that stores candidate itemsets of size 3. The internal nodes of this tree contain hash tables while the leaf nodes contain all candidate 3-itemsets. Each leaf node corresponds to one of the buckets depicted in Figure 4.6. The root node is defined to be at depth 1, its children is defined to be at depth 2, and so on. The remainder of this section describes how a hash tree can be constructed and eventually used to count the support of candidate itemsets. The latter step, which explains the *subset* function (step 5) of Table 4.2.2, is discussed first.

Support Counting Using Hash Tree

The subset function is used to find all candidates that are contained in a given transaction. To do this, one must traverse the hash tree and find all leaf nodes (buckets) that contain such candidates. More specifically, the hash tree is traversed from the root down to the leaf nodes by hashing on items that belong to the transaction. At the root node, a hash function is applied to every item that is present in the transaction. The value of the hash function will determine which branch of the current node should be followed next. If an internal node is reached after hashing on the i^{th} item of a transaction t , we will apply the hash function on every item that appears after i in t and repeat the procedure recursively on the corresponding child nodes until we reach the leaf nodes. The list of items that appear after i in t is denoted as $\text{suffix}(t, i)$. Once a leaf node is reached, all candidate itemsets stored at the leaf are compared against the transaction. If a candidate itemset is contained in the transaction, its support count is incremented.

Figure 4.7 shows an example of a hash tree that contains 15 candidate itemsets, stored in 9 leaf nodes. Also shown in this figure is the hash function used by the internal nodes of the tree: items 1, 4 and 7 are always hashed to left child of the current node; items 2, 5, 8 to the middle child; and items 3, 6, 9 to the right

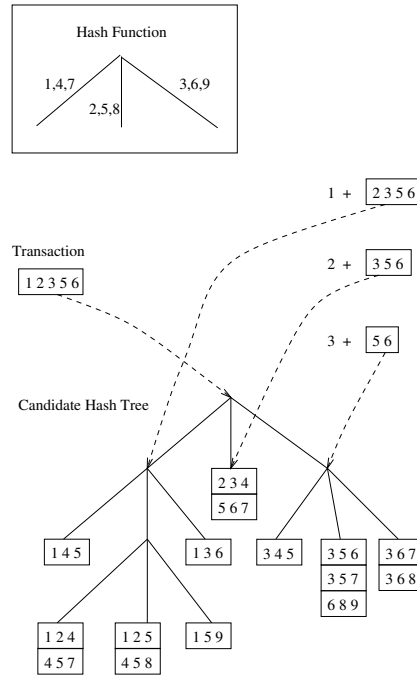


Figure 4.7. Subset operation on the root of a candidate hash tree.

child. Suppose we want to find candidates that are contained in the transaction $t = \{12356\}$. The iterative steps of the subset function are given below:

1. At the root node, the hash function is initially applied to the first item in t , which is equal to 1. Item 1 is then hashed to the left child node of the root and its suffix list of items is given as $\text{suffix}(t, 1) = \{2\ 3\ 5\ 6\}$.
2. At the left child of the root node, the hash function is applied to the first item in $\text{suffix}(t, 1)$, which is equal to 2. Item 2 is then hashed to the middle child node at depth 3 with its corresponding suffix list given as $\text{suffix}(t, 2) = \{3\ 5\ 6\}$, as shown in Figure 4.8.
3. At the child node of depth 3, the hash function is applied to the first item in $\text{suffix}(t, 2)$, which is equal to 3. Item 3 is then hashed to the right child at depth 4. The right child is a leaf node containing the candidate $\{159\}$. As t does not contain this candidate, its support count remains unchanged and the leaf node is marked to indicate that it has been visited. The subset function will then backtrack to its parent node at depth 3.
4. Next, the hash function is applied to the second item in $\text{suffix}(t, 2)$, which is equal to 5. Item 5 is hashed to the middle leaf node containing the candidates $\{125\}$ and $\{458\}$. The support for $\{125\}$ is updated because it is the only candidate contained in t . The leaf node is marked and the subset function backtracks once again to its parent node.

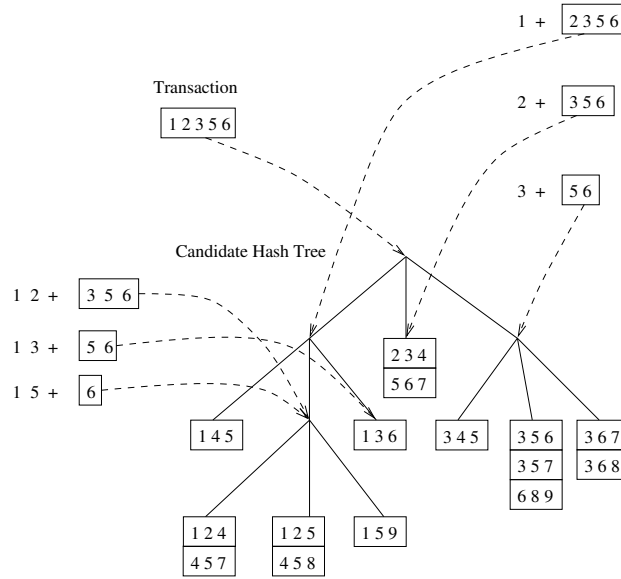


Figure 4.8. Subset operation on the left most subtree of the root of a candidate hash tree.

5. The hash function is then applied to the last item in suffix($t, 2$), which is equal to 6. Item 6 is hashed to the right child at depth 4. However, since the right child has already been visited, no further traversal will be necessary and the function backtracks to its parent node at depth 3.
6. At the parent node, since all items that belong to suffix($t, 2$) have been hashed, the internal node is marked and the function backtracks to its parent node at depth 2.
7. At the parent node of depth 2, the hash function is applied to the second item in suffix($t, 1$), which is equal to 3. Item 3 is hashed to the leaf node that contains the candidate {136}. Since the candidate is contained in t , its support count is incremented and the node is marked. The function then backtracks to its parent node at depth 2.
8. Next, the hash function is applied to the third item in suffix($t, 1$), which is equal to 5. Since item 5 is hashed to a marked internal node, the function backtracks again to its parent node at depth 2.
9. The hash function is applied to the fourth item in suffix($t, 1$), which is equal to 6. Again, since it is hashed to a marked leaf node, the function simply backtracks to its parent node at depth 2.
10. Because all items in suffix($t, 1$) have been hashed, the node is marked and the function backtracks to the root node. At the root node, the hash function is applied to the second item in t , which is equal to 2 (Figure 4.7). Item 2 is then hashed to the middle leaf node that contains two candidates, {234} and

- $\{567\}$. Since both candidates are not subsets of t , their support counts remain unchanged. The leaf node is marked and the function backtracks to the root node once again.
11. The hash function is then applied to the third item in t , which is equal to 3. Item 3 is hashed to the right child of the root node with a suffix list given as $\text{suffix}(t, 3) = \{56\}$.
 12. At the right child, the hash function is applied to the first item of $\text{suffix}(t, 3)$, which is equal to 5. Item 5 is hashed to the middle leaf node that contains three candidates, $\{356\}$, $\{357\}$, and $\{689\}$. The support count for candidate $\{356\}$ is incremented because it is the only candidate that is a subset of t . The leaf node is marked and the function backtracks to its parent node at depth 2.
 13. The hash function is applied to the last item in $\text{suffix}(t, 3)$, which is equal to 6. Item 6 is hashed to the right child, which is a leaf node containing two candidates. Since neither of the candidates are subsets of t , their support counts remain the same. The leaf node is marked and the function backtracks to its parent node.
 14. Since all items in $\text{suffix}(t, 3)$ have been hashed, the internal node is marked and the subset function backtracks to the root node.
 15. At the root node, hashing on items 5 and 6 will only bring it to marked internal nodes. As a result, no further traversal is necessary and the subset function terminates.

In the above example, 6 out of the 9 leaf nodes (buckets) are visited and 11 out of the 15 itemsets are matched against the transaction.

Hash Tree Construction

So far, we have described how a hash tree can be used to update the support counts of candidate itemsets. We now turn to the problem of constructing a hash tree to store the candidate k -itemsets. Initially, the hash tree contains only the root node. A new candidate itemset is inserted into the hash tree by hashing each successive item at the internal nodes and then following the appropriate branches according to the value of the hash function. Once a leaf node is encountered, the candidate itemset is inserted into the node as long as the number of candidates currently stored in the node does not exceed the maximum allowable size of a leaf node. Otherwise, the leaf node is converted into an internal node and new leaf nodes are created as children of the internal node. The candidate itemsets are then distributed to the children according to their hash values. For example, suppose we want to insert the candidate $\{3\ 5\ 9\}$ into the hash tree shown in Figure 4.7. At the root node, the hash function is applied to the first item of the candidate itemset, which is equal to 3. Item 3 is then hashed to the right child of the root node. Next, item 5, which

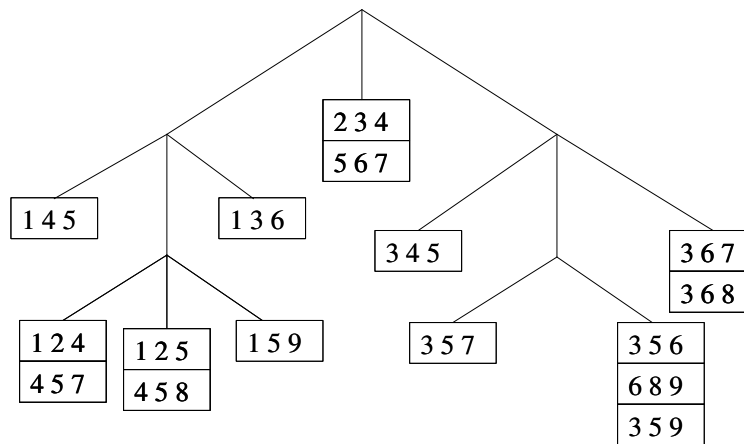


Figure 4.9. Hash tree configuration after adding the candidate itemset $\{3\ 5\ 9\}$.

is the second item of the candidate itemset, is hashed to the middle child node at depth 2. The child node is a leaf node that already contains 3 candidates, $\{3\ 5\ 6\}$, $\{3\ 5\ 7\}$, and $\{6\ 8\ 9\}$. If the maximum allowable number of candidates is equal to 3, then we cannot insert $\{3\ 5\ 9\}$ into the leaf node. Instead, the leaf node must be converted into an internal node and new child nodes are created to store the 4 candidates based on their hash values. The final hash tree after inserting the candidate $\{3\ 5\ 9\}$ is shown in Figure 4.9.

4.2.5 Alternative Frequent Pattern Mining Algorithms

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 gaining significant performance improvements, this algorithm works well only for sparse transaction data sets, for which the longest frequent itemset found does not contain too many items. The performance of Apriori would degrade considerably if the data set is dense because a large number of candidate itemsets will be generated and a huge amount of database scans is needed to determine the support counts of each candidate. Many alternative algorithms have been proposed to further improve the efficiency of frequent itemset generation. In order to put these algorithms in perspective, we present a high-level description of the various strategies employed by these algorithms:

Traversal of Itemset Lattice: The search for frequent itemsets can be conceptually represented as a traversal on the itemset lattice shown in Figure 4.1. The objective of frequent itemset generation algorithms is to discover all frequent itemsets in the least amount of time. While the Apriori principle does allow us to quickly determine which part of the lattice should be pruned, it does not specify the optimal way for traversing the lattice structure. Below, we present several methods for traversing the itemset lattice:

- Breadth-first versus Depth-first:** The *Apriori* algorithm traverses the lattice in a level-wise (breadth-first) manner. It first discovers all the size-1 frequent itemsets at level 1, followed by all the size-2 frequent itemsets at level 2, and so on, until no frequent itemsets are generated at a particular level. Alternatively, we can traverse the lattice in a depth-first manner. An example would be to start at, say itemset $\{A\}$, count its support, and then determine whether it is frequent. If it is frequent, we can keep going down the branch to $\{AB\}$, $\{ABC\}$, and so on, until we find an infrequent itemset, say $\{ABCD\}$. We then backtrack to another branch, and continue our search from there. This approach is typically used by algorithms that are designed to discover *maximal frequent itemsets*, i.e., frequent itemsets whose immediate supersets are infrequent. Figure 4.26 provides an example of a lattice containing three maximal frequent itemsets, $\{A, D\}$, $\{A, C, E\}$, and $\{B, C, D, E\}$. The *border* that separates frequent itemsets from infrequent itemsets is shown by the dashed line. Every itemset above this line is frequent, and every itemset below the line is infrequent. In addition, all frequent itemsets in the lattice must be a subset of at least one maximal frequent itemset. Thus, knowing the set of maximal frequent itemsets would allow us to enumerate the entire set of frequent itemsets.
- General-to-Specific versus Specific-to-General:** The *Apriori* algorithm uses a general-to-specific search strategy, where pairs of frequent itemsets of size k are merged together to obtain the more specific frequent itemsets of size $k + 1$. To make the search more effective, the Apriori principle is applied to prune the *supersets of infrequent itemsets*. Alternatively, a specific-to-general search strategy can be applied to find more specific frequent itemsets first before looking for more general itemsets. This strategy is useful to discover maximal frequent itemsets in a dense transaction data set. In this case, the Apriori principle is applied to prune the *subsets of maximal frequent itemsets*. Yet another search method is to combine both general-to-specific and specific-to-general strategies. This bidirectional approach is employed by algorithms such as Pincer-Search.
- Equivalent classes:** The set of all itemsets can be partitioned into disjoint groups (or equivalent classes). A frequent itemset generation algorithm may search for frequent itemsets within a particular equivalent class first before continuing its search to another equivalent class. As an example, the level-wise strategy used in *Apriori* can be considered as partitioning the itemsets on the basis of their sizes, i.e., the algorithm would find all frequent 1-itemsets first before proceeding to larger-sized itemsets. Alternatively, an equivalent class can 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 . For example, an algorithm could search for all itemsets starting with the prefix A before looking for itemsets starting with the prefix B .

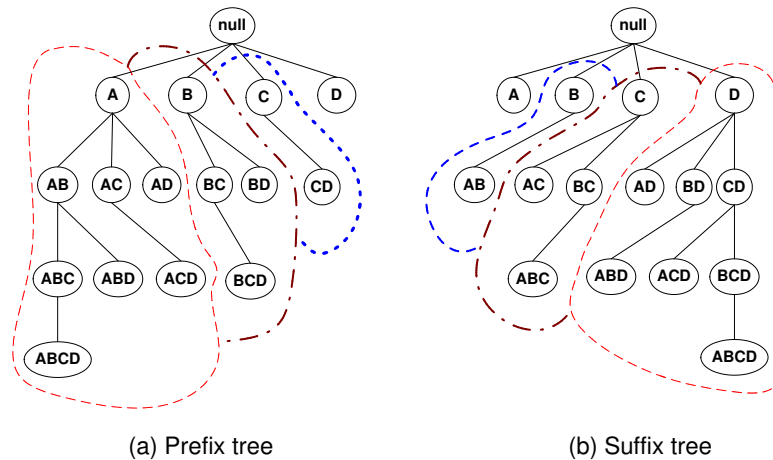


Figure 4.10. Equivalent classes based on the prefix and suffix labels of itemsets.

Each prefix or suffix equivalence relation can be represented using a set enumeration tree, as shown in Figure 4.10.

Representation of Transaction Database: There are many ways to represent a transaction database. The representation may affect the I/O costs incurred when computing the support of candidate itemsets. Figure 4.11 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) for each item. Such representation is known as the *vertical* data layout. The support of each candidate itemset can be counted by intersecting the tid-lists of their subsets. The length of the tid-lists would shrink as we progress to larger sized itemsets. One problem with this approach is that the initial size of the tid-lists could be too large to fit into main memory, thus requiring rather sophisticated data compression techniques. Another promising approach that has become increasingly popular is to compress the entire database so that it can fit into the main memory using an efficient data structure. This approach is desirable because it reduces the amount of database scans needed to determine the support counts of itemsets. We will discuss an example of using such a compact database representation in the next section.

4.2.6 FP-growth Algorithm

Recently, an interesting algorithm called FP-growth was proposed that takes a radically different approach to discover frequent itemsets. The algorithm does not subscribe to the generate-and-count paradigm of *Apriori*. It encodes the database using a compact data structure called an FP-tree and infers frequent itemsets directly from this structure.

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	9				
8	A,B,C					
9	A,C,D					
10	B					

Figure 4.11. Horizontal and vertical data format.

FP-tree Construction

First, the algorithm would scan the database once to find the frequent singleton items. An order is then imposed on the items based on decreasing support counts. Figure 4.12 illustrates an example of how to construct an FP-tree from a transaction database that contains five items, A, B, C, D, and E. Initially, the FP-tree contains only the root node, which is represented by a null symbol. Next, each transaction is used to create a path from the root node to some node in the FP-tree.

After reading the first transaction, $\{A, B\}$, a path is formed from the root node to its child node, labeled as A, and subsequently, to another node labeled as B. Each node in the tree contains the symbol of the item along with a count of transactions that reach the particular node. In this case, both nodes A and B would have a count equals to one. After reading the second transaction $\{B, C, D\}$ a new path extending from $\text{null} \rightarrow B \rightarrow C \rightarrow D$ is created. Again, the nodes along this path have support counts equal to one. When the third transaction is read, the algorithm will discover that this transaction shares a common prefix A with the first transaction. As a result, the path $\text{null} \rightarrow A \rightarrow C \rightarrow D$ is merged to the existing path $\text{null} \rightarrow A \rightarrow B$. The support count for node A is incremented to two, while the newly-created nodes, C and D, each have a support count equals to one. This process is repeated until all the transactions have been mapped into one of the paths in the FP-tree. For example, the state of the FP-tree after reading the first ten transactions is shown at the bottom of Figure 4.12.

By looking at the way the tree is constructed, it is easy to see why an FP-tree provides a compact representation of the database. If the database contains many transactions that share common items, then the size of an FP-tree would be considerably smaller than the size of the database. The best-case scenario would be that the database contains the same set of items for all transactions. The resulting FP-tree would contain only a single branch of nodes. The worse-case scenario would be that each transaction contains a unique set of items. In this case, there is no sharing of transactions among the nodes and the size of the FP-tree is the same as the size of the database.

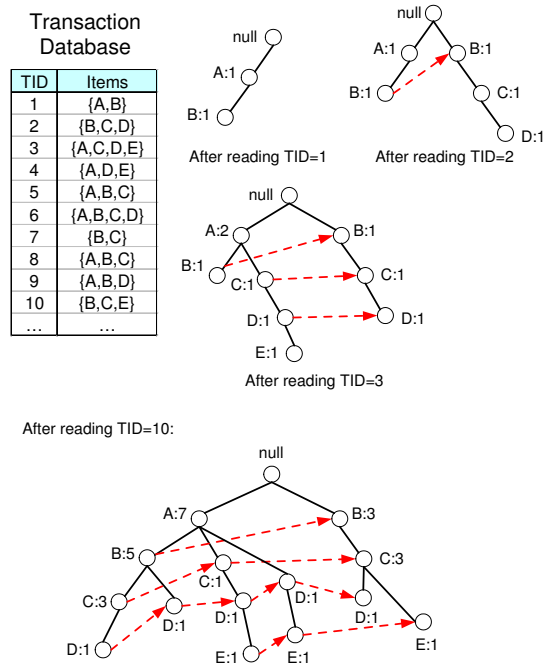


Figure 4.12. Construction of an FP-tree.

During tree construction, the FP-tree structure also stores an access mechanism for reaching every individual occurrence of each frequent item used to construct the tree. In the above example, there are five such *linked lists*, one for each A, B, C, D, and E.

Generating Frequent Itemsets from an FP-tree

The algorithm used for generating frequent itemsets from an FP-tree is known as *FP-growth*. Given the FP-tree shown in Figure 4.12, the algorithm divides the problem into several subproblems, where each subproblem involves finding frequent itemsets having a particular suffix. In this example, the algorithm will initially look for frequent itemsets that end in E by following the linked list connecting the nodes for E. After all frequent itemsets ending in E are found, the algorithm would look for frequent itemsets that end in D by following the linked list for D, and so on. Thus, the FP-growth algorithm can be considered as an implementation of the suffix tree traversal strategy described in Section 4.2.5.

How does FP-growth discover all frequent itemsets ending in E? Recall that an FP-tree stores the support counts of every item along each path, which reflect the number of transactions that are collapsed onto the particular path. In our example, there are only three occurrences of the node E. By collecting the *prefix paths* of E, we can solve the subproblem of finding frequent itemsets ending in E (see Figure 4.13). The prefix paths of E consist of all paths starting from the root node up to the parent nodes of E. These prefix paths can form a new FP-tree for which the

Generating Itemsets ending with E:

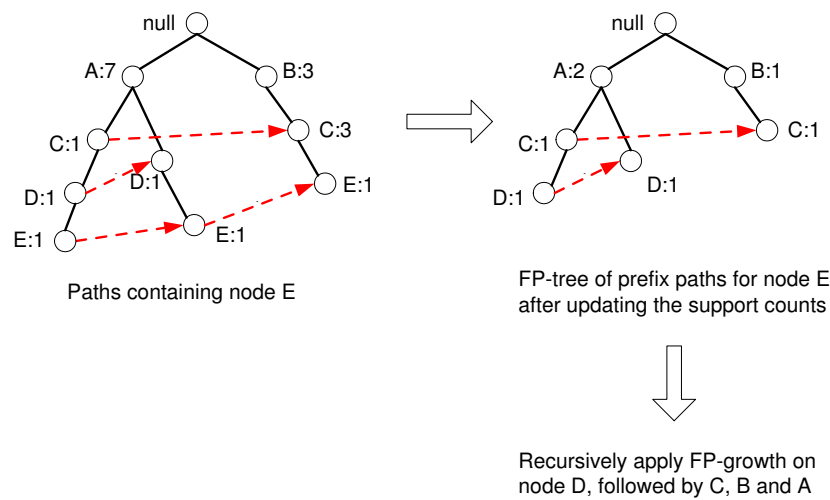


Figure 4.13. An illustrative example of the FP-growth algorithm for finding itemsets ending in E.

FP-growth algorithm can be recursively applied.

Before creating a new FP-tree from the prefix paths, the support counts of items along each prefix path must be updated. This is because the initial prefix path may include several transactions that do not contain the item E. For this reason, the support count of each item along the prefix path must be adjusted to have the same count as node E for that particular path, as illustrated in Figure 4.13. For example, the prefix path $\text{null} \rightarrow B:3 \rightarrow C:3$ will be updated to $\text{null} \rightarrow B:1 \rightarrow C:1$ since the support count for node E is 1.

After updating the counts along the prefix paths of E, some items may no longer be frequent and thus must be removed from further consideration (as far as our new subproblem is concerned). An FP-tree of the prefix paths is then constructed by removing the infrequent items. This recursive process of breaking up the problem into smaller subproblems will continue until the subproblem involves only a single item. If the support count of this item is greater than the minimum support threshold, then the label of this item will be returned by the FP-growth algorithm. The returned label is appended as a prefix to the frequent itemset ending in E.

The FP-growth algorithm is an interesting approach because it illustrates how a compact representation of the transaction database can be used to efficiently generate frequent itemsets. In addition, it has been shown that FP-growth can be several orders of magnitude faster than the standard *Apriori* algorithm for many transaction data sets. The performance of the FP-growth algorithm depends on the *compaction factor* of the database. If the resulting FP-tree is very bushy (in the worst case, a full prefix tree) in most of its subproblems, then the performance of this algorithm will degrade considerably because it has to generate a large number of subproblems in its recursive formulation.

4.3 Rule Generation

This section describes how to extract association rules efficiently from frequent itemsets. Given a frequent itemset L , an association rule is extracted by partitioning the itemset into two halves, l and $L - l$, such that $l \implies L - l$ satisfies the minimum confidence requirement. This binary partitioning approach ensures that every extracted rule has exactly the same support as its corresponding frequent itemset. In addition, a size- k frequent itemset can produce up to $2^k - 2$ association rules, ignoring rules that have empty antecedent or consequent ($\emptyset \implies L$ or $L \implies \emptyset$).

Example 22 Suppose $L = \{1, 2, 3\}$ is a frequent itemset. There are six possible rules that can be generated from this frequent itemset: $\{1, 2\} \implies \{3\}$, $\{1, 3\} \implies \{2\}$, $\{2, 3\} \implies \{1\}$, $\{1\} \implies \{2, 3\}$, $\{2\} \implies \{1, 3\}$ and $\{3\} \implies \{1, 2\}$. As the support for the rules are identical to the support for the itemset $\{1, 2, 3\}$, all the rules must satisfy the minimum support condition. The only remaining step during rule generation is to compute the confidence value for each rule.

Computing the confidence of an association rule does not require additional scans over the transaction database. For example, consider the rule $\{1, 2\} \implies \{3\}$, which is generated from the frequent itemset $L = \{1, 2, 3\}$. 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 for both itemsets were already found during frequent itemset generation, no additional database scans is needed to determine the confidence for this rule.

Anti-Monotone Property Unlike the support measure, confidence does not possess any monotonicity property. For example, the confidence for the rule $X \implies Y$ can be larger or smaller than the confidence for another rule $\tilde{X} \implies \tilde{Y}$, where \tilde{X} is a subset of X and \tilde{Y} is a subset of Y . Nevertheless, if we compare rules generated from the same frequent itemset L , the following theorem holds for the confidence measure.

Theorem 2 If a rule $l \implies L - l$ does not satisfy the minimum confidence threshold, then any rule $l' \implies L - l'$, where l' is a subset of l , must not satisfy the confidence threshold as well.

To prove this theorem, consider the following two rules: $a \implies L - a$ and $l \implies L - l$, where $a \subset l$. The confidence for both rules are $\sigma(L)/\sigma(a)$ and $\sigma(L)/\sigma(l)$, respectively. Since a is a subset of l , $\sigma(a) \geq \sigma(l)$, which is why the confidence of the former rule may never exceed the confidence of the latter rule.

Confidence Pruning The *Apriori* algorithm uses a level-by-level approach for generating association rules, where each level corresponds to the number of items that belong to the rule consequent. Initially, all high-confidence rules that have only a single item in the rule consequent are extracted. At the next level, the algorithm uses

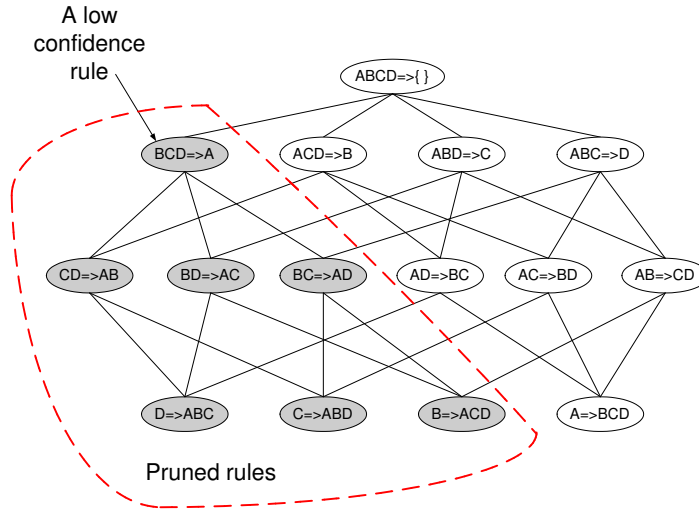


Figure 4.14. Pruning of association rules using confidence measure.

rules extracted from the previous level to generate new candidate rules. For example, if both $ACD \rightarrow B$ and $ABD \rightarrow C$ satisfy the minimum confidence threshold, then a candidate rule $AD \rightarrow BC$ is generated by merging their rule consequents. *Apriori* also uses Theorem 2 to substantially reduce the number of candidate rules. Figure 4.14 shows an example of the lattice structure for association rules that can be generated from the 4-itemset $\{A, B, C, D\}$. If any node in the lattice has low confidence, then according to Theorem 2, the entire subgraph spanned by the node can be immediately pruned. For example, if the rule $BCD \rightarrow A$ does not satisfy the minimum confidence threshold, we can prune away all rules containing item A in its consequent, such as $CD \rightarrow AB$, $BD \rightarrow AC$, $BC \rightarrow AD$, $D \rightarrow ABC$, etc. Finally, the confidence for all remaining candidate rules are computed. If the confidence of a rule is below the minimum confidence threshold, the rule is immediately pruned. A pseudocode for the rule generation step is given in Table 4.4.

4.4 Handling Continuous and Categorical Attributes

So far, we have described the problem of mining association rules in the context of asymmetric binary variables. What if the data contains both continuous and categorical attributes? Are the current techniques still applicable?

To illustrate this situation, consider the example Web data shown in Table 4.5. The data contains various information about the demographic and browsing activities of Web users such as their country of origin, the length of a Web session, the number of Web pages viewed, etc. Analysis of the Web data may reveal interesting patterns that describe the typical characteristics of Web users who made (or did not make) a purchase at their Web site. Examples of such patterns include:

$$(\text{Country} = \text{USA}) \wedge (\text{Session Length} > 1000) \rightarrow (\text{Buy} = \text{Yes}), \quad \text{or}$$

Table 4.4. Algorithm for generating rules from frequent itemsets.

```

1. forall large  $k$ -itemsets  $l_k$ ,  $k \geq 2$  do
2.    $H_1 = \{ \text{consequents of rules derived from } I_k \text{ with one item in the consequent} \};$ 
3.   call ap-genrules( $I_k, H_1$ );

procedure ap-genrules( $l_k$ : large  $k$ -itemset,  $H_m$ : set of  $m$ -item consequents)
1. if ( $k > m + 1$ ) then {
2.    $H_{m+1} = \text{apriori-gen}(H_m);$ 
3.   forall  $h_{m+1} \in H_{m+1}$  {
4.      $\text{conf} = \sigma(l_k) / \sigma(l_k - h_{m+1});$ 
5.     if ( $\text{conf} \geq \text{minconf}$ ) then
6.       output the rule  $(l_k - h_{m+1}) \implies h_{m+1};$ 
7.     else
8.       delete  $h_{m+1}$  from  $H_{m+1};$ 
9.   }
10.  call ap-genrules( $l_k, H_{m+1}$ );
11. }

```

Table 4.5. Example of Web data for mining association rules.

Session Id	Country of Origin	Session Length (sec)	Number of pages viewed	Gender	Browser Type	Buy
1	USA	982	8	Male	Internet Explorer	No
2	China	811	10	Female	Netscape	No
3	USA	2125	45	Female	Mozilla	Yes
4	Germany	596	4	Male	Internet Explorer	Yes
5	Australia	123	9	Male	Mozilla	No
...

$$(\text{Number of Pages} \in [5, 10)) \wedge (\text{Browser} = \text{Mozilla}) \longrightarrow (\text{Buy} = \text{No}).$$

Unlike previously seen association rules, the rules shown above involve a mixed combination of continuous and categorical attributes. In the remainder of this section, we describe how to modify the existing association rule formulation to extract such rules.

4.4.1 Categorical Attributes

A typical approach for handling categorical attributes is to transform them into asymmetric binary variables so that existing association rule mining algorithms can be applied to the data set. Categorical attributes are transformed into asymmetric binary variables by introducing as many new “items” as the number of distinct attribute-value pairs. For example, the categorical attribute **Browser Type** can be replaced by asymmetric binary variables such as **Browser=Internet Explorer**, **Browser=Netscape**, and **Browser=Mozilla**, while the attribute **Country of Origin** can be replaced by the names of all countries present in the data set. Even a symmetric binary attribute such as **Gender** can be handled in this way by replacing it with the asymmetric binary variables **Male** and **Female**. Table 4.6 shows the results of binarizing the categorical and symmetric binary attributes of the Web data. Despite the addition of new items, the width of each transaction remains unchanged as each Web user has only one country of origin, browser type, and gender.

However, one problem with this approach is that some categorical attributes such as **Country of Origin** can have a large number of distinct values. Creating a new item for each country name may increase the dimensionality of the Web data tremendously because there are more than 190 independent countries around the world. If the support threshold is low enough such that most of the newly created items are frequent, the problem becomes compute intensive since a large number of candidate itemsets must be generated from these frequent items. (Nevertheless, the maximum size of frequent itemsets found does not change even with the addition of new asymmetric binary variables because it depends only on the maximum width of the transactions.)

One way to address this problem is by reducing the number of newly created

Table 4.6. Example of Web data after binarizing the categorical attributes.

Session Id	USA	China	...	Male	Female	Internet Explorer	Netscape	...	Buy
1	1	0	...	1	0	1	0	...	No
2	0	1	...	0	1	0	1	...	No
3	1	0	...	0	1	0	0	...	Yes
4	0	0	...	1	0	1	0	...	Yes
5	0	0	...	1	0	0	0	...	No
...

items using a concept hierarchy. Attribute-value pairs that are related according to the concept hierarchy can be merged together into more meaningful concepts. For example, neighboring countries can be grouped together according to their geographic locations — Middle East, Scandinavia, North Africa, South-East Asia, etc. The effect of using concept hierarchy for association rule mining will be described in further details in Section 4.5.

Another way to improve the performance of existing algorithms is by eliminating candidate itemsets involving items of the same type. For example, all candidate itemsets of the form $\{\text{Browser=XXX}, \text{Browser=YYY}, \dots\}$ are discarded because their support counts are guaranteed to be zero. Eliminating such candidates early will help to reduce the amount of time needed for support counting.

4.4.2 Continuous Attributes

Various methods have been developed for handling data with continuous attributes. We will discuss three such methods in this section: (1) discretization-based methods, (2) statistics-based methods, and (3) non-discretization methods. The nature of the rules discovered by these methods are quite different, as will be shown in the next few sections.

Discretization-based Methods

Discretization is perhaps the most common strategy for dealing with continuous attributes. Each continuous attribute is discretized by partitioning its range of attributes values into several disjoint intervals. For example, the attribute **Session Length** can be partitioned into discrete intervals such as $\text{Session Length} < 100$, $\text{Session Length} \in [100, 200)$, etc., where $[a, b)$ indicates that the interval includes the value a but excludes the value b . Each discrete interval is subsequently mapped into an asymmetric binary variable, thus allowing existing association rule mining algorithms to be applied to the data set. Table 4.7 illustrates the results of discretizing the Web data shown in Table 4.5.

There are many ways to discretize the continuous attributes. In the simplest case, one may apply the equal interval width or equal frequency approaches described in

Table 4.7. Example of Web data after discretizing continuous attributes.

Session Id	Session Length < 100	Session Length $\in [100, 200)$...	Number of pages < 5	Number of pages $\in [5, 10)$...	Buy
1	0	0	...	0	1	...	No
2	0	0	...	0	0	...	No
3	0	0	...	0	0	...	Yes
4	0	0	...	1	0	...	Yes
5	0	1	...	0	1	...	No
...

Chapter 2. The drawback of these approaches is that they are somewhat blind to the end goal of association analysis, which is to produce high support and high confidence rules. In other cases, supervised discretization approaches have been known to produce better intervals that are tailored towards the objective function of certain analysis, e.g., minimizing the entropy of a target attribute. Unfortunately, unlike classification problems where the target attribute is fixed, it is difficult to apply supervised discretization approaches to association rule mining since the rule consequent may involve any one items present in the data set.

Discretization can also affect association rule mining in many ways. First, the support count of a discretized variable depends on its interval width. The wider is the interval, the higher its support count will be. As a result, if the discrete interval is too narrow, then its support may not be high enough to be picked up by existing association rule mining algorithms. Second, the interval width can also affect the confidence of an association rule. If the interval is too wide, rules involving such an interval may fail the minimum confidence threshold — a phenomenon that is known as the *information loss problem*. For example, suppose the confidence for the rule (Number of pages $\in [5, 10)$) \rightarrow (Buy = No) is 95%. Using a much wider interval for Number of pages may decrease the confidence of this rule significantly.

To overcome the low support problem, adjacent intervals can be merged together into wider intervals. For example, the adjacent intervals, Session Length $\in [100, 200)$, Session Length $\in [200, 300)$, and Session Length $\in [300, 400)$, can be merged into a wider interval, Session Length $\in [100, 400)$. Although this approach can increase the support of the interval, it may still suffer from the information loss problem.

To avoid the low support and information loss problems, it is advantageous to augment the data set with both coarser- and finer-grained intervals, even though this may lead to two additional problems:

1. *The computation becomes extremely expensive.* For example, suppose the variable Number of pages $\in [0, 10)$ is a frequent item. If we retain all possible mergings of this variable, e.g., Number of pages $\in [0, 10)$, Number of pages $\in [0, 20)$, ..., Number of pages $\in [0, 100)$, then a large number of frequent itemsets is generated based on different combinations of these intervals alone.

This problem can be addressed by modifying the frequent itemset generation algorithm to remove any candidate itemset containing more than one discrete interval of the same type. In other words, a candidate itemset that contains more than one item for **Number of pages** must be pruned prior to support counting.

2. *Some of the discovered rules are redundant.* For example, suppose the rules

$R_1 : \text{Number of pages} \in [40, 50), \text{Browser}=\text{Mozilla} \longrightarrow \text{Buy} = \text{Yes}$ and

$R_2 : \text{Number of pages} \in [30, 50), \text{Browser}=\text{Mozilla} \longrightarrow \text{Buy} = \text{Yes}$

both have very similar support and confidence. By definition, a rule r' is an ancestor rule for r (and r is the descendent rule for r') if the intervals for all of its continuous attributes are at least as wide as the intervals for r . In the example shown above, R_2 is an ancestor rule of R_1 since it has a wider interval for **Number of pages**. Furthermore, because R_1 is the more specific rule and has the same support and confidence as its ancestor rule, it should be pruned in favor of R_2 . We can generalize this approach for pruning rules whose support and confidence are not exactly identical to their ancestor rules. Given a pair of rules $r : AB \longrightarrow C$ and $r' : A'B' \longrightarrow C'$, where r' is the closest ancestor rule to r , we can prune r if its support (or confidence) is within ρ times its expected support (or confidence) computed from its ancestor rule r' , where ρ is a user-specified parameter and the expected support can be computed in the following way.

$$\begin{aligned} E(s(r)) &= s(r') \times \frac{s(A)}{s(A')} \times \frac{s(B)}{s(B')} \times \frac{s(C)}{s(C')} \\ E(c(r)) &= c(r') \times \frac{s(C)}{s(C')}. \end{aligned} \quad (4.3)$$

In this case, r is eliminated if $s(r)/E(s(r)) < \rho$ or $c(r)/E(c(r)) < \rho$.

Statistics-based Methods

A continuous attribute can also be used to understand the statistical properties of a population. For the Web data shown in Table 4.5, if the date of birth for each Web user is known, analysts may be interested in knowing the average age of certain groups of Web users. For example, the rule

$$\text{Browser}=\text{Mozilla} \wedge \text{Buy} = \text{Yes} \longrightarrow \text{Age}:\mu = 23$$

suggests that the average age of Web users who buy a product at their Web site and use Mozilla as their browser is 23 years old. Such rules assume that a continuous target variable, e.g., age, has been chosen to characterize interesting segments of the population that was identified by the rule antecedent.

To discover this type of rules, one must first withhold the continuous target variable from the rest of the data. Continuous attributes that are not part of the

target variable are discretized using the strategies described in the previous section. Existing frequent itemset generation algorithms can be applied once the data has been transformed into an asymmetric binary form. Each discovered frequent itemset identifies a certain segment of the population. The target variable is then used to characterize each discovered segment by computing descriptive statistics such as mean, median, or variance of the continuous variable. The example rule shown above is obtained by computing the average age of Web users covered by the frequent itemset $\{\text{Browser}=\text{Mozilla}, \text{Buy} = \text{Yes}\}$.

Since descriptive statistics can be computed for each frequent itemset, the number of rules extracted can be as many as the number of frequent itemsets. This could be an issue because not all frequent itemsets are interesting. One cannot evaluate how interesting is an itemset by simply looking at the average value of its population segment. Instead, one should evaluate an itemset based on how different it is from the average value computed for the population segment not covered by the itemset. For example, the rule shown above is interesting only if the average age of Web users who are not covered by the rule antecedent, is significantly different than 23 years old. Below, we formalize a statistical test for evaluating the significance of the difference in mean values of two populations.

Consider a pair of association rules, $A \longrightarrow B : \mu$ and $\bar{A} \longrightarrow B : \mu'$, where \bar{A} is the complement of the condition A . Each rule is assumed to characterize the properties of two different segments of the population, whose population means for the target attribute B are M and M' , respectively. We also assume that the rules have sufficiently large support counts (> 30) and the transactions covered by the rules can be regarded as independent samples randomly drawn from each population. Based on these assumptions, the expected values for the population means can be computed in terms of their sample means, i.e., $E(M) = \mu$ and $E(M') = \mu'$. We consider the rule $A \longrightarrow B : \mu$ to be interesting if the difference between μ and μ' is greater than some user-specified threshold, Δ .

We can apply a statistical test to determine whether the observed difference in the sample means of both populations is statistically significant at a given confidence level. (A general overview of hypothesis testing is given in the Appendix.)

Assuming that $\mu < \mu'$, then the null hypothesis to be tested is $H_0 : \mu' = \mu + \Delta$ versus the alternative hypothesis $H_1 : \mu' > \mu + \Delta$. We can define a Z-statistic for this test as

$$Z = \frac{\mu' - \mu - \Delta}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}. \quad (4.4)$$

which has a standard normal distribution with mean 0 and variance 1 under the null hypothesis.

Example 23 Suppose the rule $r : \text{Browser}=\text{Mozilla} \wedge \text{Buy} = \text{Yes} \longrightarrow \text{Age} : \mu = 23$ has a standard error of 3.5 while its complement has a mean of $\mu' = 30$ and standard error of 6.5. r is an interesting rule if the difference between μ' and μ is greater than 5 years. The support counts for the rule and its complement are 50 and 250, respectively.

Thus, using equation 4.4 we obtain

$$Z = \frac{30 - 23 - 5}{\sqrt{\frac{3.5^2}{50} + \frac{6.5^2}{250}}} = 3.11. \quad (4.5)$$

For a 1-sided test at 95% confidence level, the critical Z value for rejecting the null hypothesis is 1.64. Since $Z > 1.64$, we may conclude that the difference in the average age of the population segments is greater than 5 years. Thus, r is an interesting rule.

Non-discretization Methods

There are certain applications in which analysts are more interested in finding associations among the continuous attributes, rather than associations among discrete intervals of the continuous attributes. For example, consider the problem of finding

Table 4.8. Example of mining association rules in text data.

Document	$word_1$	$word_2$	$word_3$	$word_4$	$word_5$	$word_6$
d_1	3	6	0	0	0	2
d_2	1	2	0	0	0	2
d_3	4	2	7	0	0	2
d_4	2	0	3	0	0	1
d_5	0	0	0	4	1	3

word associations in text documents, as shown in Table 4.8. In this document-term matrix, each column contains the frequency counts of a word appearing in the corresponding documents. Often times, analysts are interested in finding associations between words in the vocabulary (e.g., **data** and **mining**) instead of associations between ranges of word frequencies (e.g., $\mathbf{data} \in [1, 4]$ and $\mathbf{mining} \in [2, 3]$).

A naive way for finding word associations is to transform the data into a 0/1 matrix, where the column entry is 1 if the word frequency is greater than 0, and 0 otherwise. This allows us to apply existing frequent itemset generation algorithms to the binarized data set. However, the drawback of this approach is that it does not take into account the significance of each word in a document. For example, a word that appears five times in a document could be more significant than a word that appears only once in the document. Therefore, techniques that can handle word frequencies without discretizing them are needed.

An immediate challenge is to determine the support for each word. Summing up the frequencies in each column is not a viable solution because the word frequency can be greater than the total number of documents. As a result, the support of a word can be greater than 100%. An alternative approach is to normalize the frequency of each word in a document so that its overall support is always less than or equal to 1. There are several ways to perform the normalization. First, we

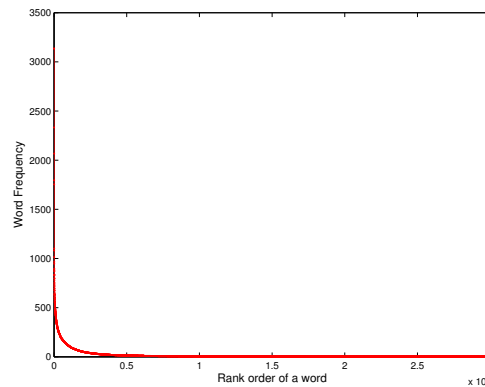


Figure 4.15. Frequency of words that appear in a collection of Los Angeles Times news snippets.

Table 4.9. Normalized document-term matrix.

Document	$word_1$	$word_2$	$word_3$	$word_4$	$word_5$	$word_6$
d_1	0.3	0.6	0	0	0	0.2
d_2	0.1	0.2	0	0	0	0.2
d_3	0.4	0.2	0.7	0	0	0.2
d_4	0.2	0	0.3	0	0	0.1
d_5	0	0	0	1.0	1.0	0.3

can look for the word having the highest frequency and then divide the frequencies of each column by this maximum frequency. This will ensure that only the most frequent word has a support equals to 100%, while the rest of the words have support less than 100%. For example, in Table 4.8, since the maximum word frequency is 10, the normalized word frequency is obtained by dividing each frequency by a factor of 10. Although this seems to be a natural approach, it is biased towards words that appear most frequently in the data set (e.g., stopwords like **the**, **of**, **to**, etc.) For example, Figure 4.15 shows the distribution of word frequencies in a text corpus consisting of news snippets collected from the Los Angeles Times. Despite having close to 30,000 words, only 11 of those words have frequencies higher than 1000, while close to 24,000 of the remaining words have frequencies less than 10. If the data set is normalized by the maximum word frequency, there is a huge discrepancy between the support of stopwords and the support of rarely used words. As a result, most of the frequent itemsets found are dominated by the presence of stopwords.

Another approach would be to normalize each column vector to have unit length, as shown in Table 4.9. In this approach, each word is weighted by its relative frequency across all documents. For example, although $word_5$ appears only once in document d_5 while $word_6$ appears three times, the relative weight for $word_5$ is higher than $word_6$ ($word_6$ might be an example of a stopwords.)

Another challenge is to determine the support of an “itemset.” For the first document, the normalized frequency of $word_1$ is 0.3 and the normalized frequency

of $word_2$ is 0.6. What should be the support for $\{word_1, word_2\}$? One way to do this is by summing up their average normalized frequencies across all documents:

$$s(word_1, word_2) = \frac{0.3 + 0.6}{2} + \frac{0.1 + 0.2}{2} + \frac{0.4 + 0.2}{2} + \frac{0.2 + 0}{2} = 1.$$

Unfortunately, this approach makes the support for all itemsets to be equal to 1 (the proof of this is left as an exercise.) Another approach is to sum up the minimum values of the normalized frequencies:

$$s(word_1, word_2) = \min(0.3, 0.6) + \min(0.1, 0.2) + \min(0.4, 0.2) + \min(0.2, 0) = 0.6.$$

This approach ensures that the support for all itemsets will vary between 0 and 1. Furthermore, this new definition of the support measure is anti-monotone, much like the traditional support definition. To illustrate the anti-monotone property, consider the pair of itemsets $\{A, B\}$ and $\{A, B, C\}$. Note that $s(\{A, B\}) \geq s(\{A, B, C\})$ since $\min(\{A, B\}) \geq \min(\{A, B, C\})$ — adding C can only decrease the minimum value of (A, B) . An algorithm known as **Min-apriori**, which is a modified version of the Apriori algorithm, was developed to explicitly use this minimum support measure to generate frequent itemsets from text documents such as the one shown in Table 4.8

4.5 Handling Concept Hierarchy

A concept hierarchy is a multi-level organization of the various entities or concepts defined in a particular domain. For example, in market-basket analysis, a concept hierarchy has the form of an item taxonomy describing the “is-a” relationships that exist among items sold at a grocery store — e.g., milk is a type of food (Figure 4.16). Concept hierarchies are often defined according to domain knowledge or based on a standard classification scheme defined by certain organizations (e.g., the Library of Congress classification scheme is used to organize library materials based on their subject categories.)

There are several reasons why it is useful to incorporate concept hierarchy into the association rule mining task:

1. Rules at lower levels of the hierarchy may not have enough support to appear in any frequent itemsets. For example, the sale of laptop AC adaptors and docking stations may be low but the collective sale of **laptop accessories** may be high. If the concept hierarchy is not used, then we may miss interesting patterns involving the laptop accessories.
2. There could be too many rules at lower levels of the hierarchy that are overly specific and not as interesting as rules at higher levels. For example, staple items such as bread and milk tend to produce many low-level rules such as **skim milk** \rightarrow **Wonder white bread**, **Foremost 2% milk** \rightarrow **Pepperidge Farm white bread**, and **Kemps 2% milk** \rightarrow **wheat bread**. Using the concept

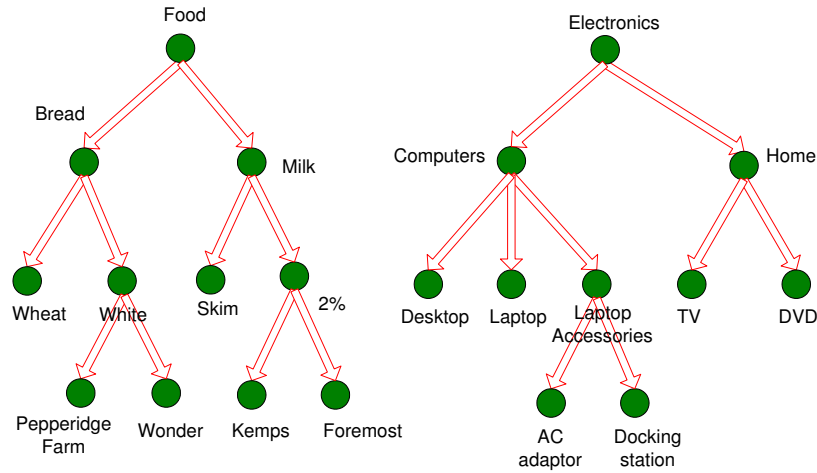


Figure 4.16. Example of an item taxonomy.

hierarchy, we can potentially summarize this set of related rules into a single rule, `milk` \rightarrow `bread`.

A concept hierarchy is often represented using a directed acyclic graph representation, as shown in Figure 4.16. If there is an edge in the graph from a node p to another node c , we call p as the *parent* of c and c as the *child* of p . For example, `milk` is the parent of `skim milk` because there is a directed edge from the node `milk` to `skim milk`. Furthermore, we say \hat{X} is an ancestor of X if there is a path from the node \hat{X} to node X in the directed acyclic graph (e.g., `food` is an ancestor of `skim milk`.)

Note that one cannot directly infer rules involving items at higher levels of the taxonomy based on the rules discovered for lower level items. Let us examine how the support and confidence of an itemset varies as we traverse the item taxonomy:

- If \hat{X} is the parent for both X_1 and X_2 , then $\sigma(X) \leq \sigma(X_1) + \sigma(X_2)$. This is because some transactions may contain both X_1 and X_2 . For example, the support for `food` can be less than the sum of the supports for `bread` and `milk` because there are some transactions that contain both `bread` and `milk`.
- If $\sigma(X \cup Y) \geq s$, then $\sigma(\hat{X} \cup Y) \geq s$, $\sigma(X \cup \hat{Y}) \geq s$, and $\sigma(\hat{X} \cup \hat{Y}) \geq s$. In other words, support always increases as you go up the item taxonomy. For example, the support for `AC adaptor` \rightarrow `DVD` cannot be greater than the support for `computers` \rightarrow `home electronics`.
- If $\text{conf}(X \Rightarrow Y) > \alpha$, then only $\text{conf}(X \Rightarrow \hat{Y})$ is guaranteed to be larger than α . For example, if the confidence for the rule `AC Adaptor` \rightarrow `DVD` is above the *minconf* threshold, then the rules `AC adaptor` \rightarrow `home electronics` and `AC adaptor` \rightarrow `electronics` are guaranteed to be higher than *minconf* too. However, the same cannot be said about the confidence of

the rules `laptop accessories` \longrightarrow `DVD`, `computers` \longrightarrow `home electronics`, or `laptop accessories` \longrightarrow `home electronics`.

One way to extend the current association rule formulation to handle concept hierarchies is as follows. Initially, each transaction t is replaced by an *extended transaction* t' , which contains all the items in the original transaction t as well as their corresponding ancestors. For example, the transaction $t = \{\text{DVD, wheat bread}\}$ can be extended to $t' = \{\text{DVD, wheat bread, home electronics, electronics, bread, food}\}$ where `home electronics` and `electronics` are the ancestors of `DVD` while `bread` and `food` are the ancestors of `wheat bread`. With this approach, we can apply existing algorithms such as Apriori to the extended database and obtain rules that can span different levels of the concept hierarchy. Nevertheless, such an approach may encounter several technical difficulties.

1. *Items that reside at higher levels of the concept hierarchy tend to have higher support counts compared to those that reside at the lower levels of the hierarchy.* Thus, if the minimum support threshold is set too high, only patterns from higher levels of the hierarchy are extracted. If the threshold is set too low, the algorithm becomes computationally inefficient. This observation suggests that it may no longer be sufficient to apply the same support threshold in dealing with items from different levels of the hierarchy.
2. *By introducing a concept hierarchy, the dimensionality of the problem will increase due to the increase in the number of items as well as the width of transactions.* The computational complexity will also increase since the number of candidate and frequent patterns generated by existing algorithms may grow exponentially as the width of transactions increases. In addition, many of the discovered rules are redundant because they describe the relationship between an item with its ancestor items. For example, the rule `skim milk` \longrightarrow `food` is redundant even though its support and confidence can be very high because `food` is an ancestor of `skim milk`.

One way to improve the performance of existing algorithms is by pruning the redundant itemsets, i.e., any itemset that contains the item X along with its ancestor items, \hat{X} . It is safe to eliminate such candidate patterns because support remains unchanged even if we remove all the ancestor items of X from the pattern. If a pattern Z that contains both X and \hat{X} is frequent, then the corresponding pattern $Z - \hat{X}$, that contains X but not \hat{X} , must be frequent too.

In some cases, one may be willing to ignore patterns involving items from different levels of the hierarchy — e.g., a cross-level pattern such as `2% Foremost milk` \longrightarrow `bread` can be ignored once a high-level pattern such as `milk` \longrightarrow `bread` were already discovered. This allows us to implement a more efficient pattern generation algorithm that employs a level-wise pattern enumeration strategy. The strategy works because the support for lower-level items is always smaller than the support for their ancestors. Thus, if an itemset at the higher level of the concept hierarchy is

infrequent, its corresponding itemset at the lower level of the hierarchy must be infrequent too (assuming that the same minimum support threshold is applied.) This top-down, level-wise algorithm can be described as follows. Initially, all frequent itemsets of size-1 at the top level of the concept hierarchy are generated. These itemsets will be denoted as $L(1, 1)$. For example, if both **food** and **electronics** are frequent, then $L(1, 1) = \{\text{food}, \text{electronics}\}$. Next, using $L(1, 1)$, the algorithm proceeds to generate all frequent itemsets of size 2, $L(1, 2)$. This procedure is repeated until all frequent itemsets involving items from the top level of the hierarchy, $L(1, k)$ ($k > 1$), are extracted. The algorithm can then proceed to the next level of the taxonomy, $L(2, 1)$. In this case, only the descendants of frequent itemsets in $L(1, 1)$ are considered to be candidates for $L(2, 1)$. For example, since both **food** and **electronics** belong to $L(1, 1)$, then the candidate itemsets for $L(2, 1)$ include **bread**, **milk**, **computers**, and **home electronics**. The algorithm then proceeds to generate $L(2, 2)$, $L(2, 3)$, \dots sequentially until all level-2 frequent itemsets are extracted. This procedure will move on to the next level and so on, until it terminates at the deepest level requested by the user.

There are two major limitations to this approach. First, the I/O requirements would increase dramatically due to the increasing number of database scans needed. Second, the approach is no longer complete, i.e., it ignores itemsets from different levels of the hierarchy. In addition, if a different minimum support threshold is applied at each level (higher thresholds at the upper levels of the hierarchy, and lower thresholds at the lower levels of the hierarchy), one could potentially miss many frequent itemsets using the top-down, level-wise approach because the anti-monotone property is violated. We will discuss this problem of using multiple minimum support in Section 4.6.1.

4.6 Effect of Support Distribution

When applying association analysis to real data sets, a natural question to ask is how to choose the right minimum support threshold? To answer this question, let us first examine the effect of support distribution on the choice of minimum support threshold. Figure 4.17 shows the support distribution of a data set created using the synthetic data generator available from IBM Almaden (<http://www.almaden.ibm.com/software/quest/Resources/index.shtml>). The synthetic data set contains 10,000 items and 100,000 transactions. In this figure, the items are sorted in increasing order of their support levels. Some of the items have support less than 0.01%, while others have support more than 1%. Given the wide range of support values, we can partition the items into three groups, G1, G2, and G3. G1 contains items with support less than 0.05%, G2 contains items with support between 0.05% and 1% transactions, and G3 contains items with support greater than 1%.

Choosing the appropriate minimum support threshold for this data set can be quite tricky due to its wide range of support distribution. If the threshold is set too high, we may miss interesting patterns involving items from low support levels. Conversely, if the minimum support threshold is set too low, the problem be-

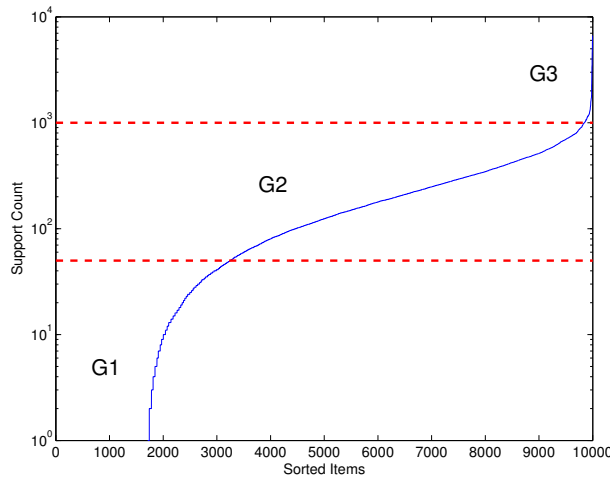


Figure 4.17. Support distribution of items for a synthetic data set created using the IBM Almaden synthetic data generator.

comes computationally intractable and can produce a huge number of uninteresting patterns. Figure 4.18 illustrates the effect of applying different minimum support thresholds on the number of extracted frequent itemsets. When the minimum support threshold is set to 0.01%, the number of frequent itemsets is extremely large and the patterns are also longer compared to those extracted at higher support thresholds.

Some of the patterns generated at low minimum support thresholds are not interesting because they contain items with significantly different support levels. As an example, many analysts may not consider the association between the sale of milk (a very frequent item) and caviar (a rare item) to be interesting even though the confidence for the rule `caviar` \rightarrow `milk` is high. The confidence of this rule is high simply because many of the transactions involve the purchase of milk. As a result, it is not surprising to find milk in transactions that contain caviar.

A standard algorithm such as Apriori can produce a large number of frequent itemsets when applied to the data set shown in Figure 4.17. For example, if the minimum support threshold is set to 0.01%, more than 11 million frequent itemsets up to size-10 are extracted. Out of this, there are 640,916 mixed frequent itemsets involving items from both G1 and G3. Such itemsets are often uninteresting because they contain items from significantly different support levels, i.e., the aforementioned caviar and milk problem.

4.6.1 Multiple Minimum Support

The discussion presented in the previous section suggests that choosing the appropriate minimum support threshold is a challenging task especially for data sets having a skewed support distribution. One must ensure that the support threshold is low enough to capture interesting patterns involving the low-support items

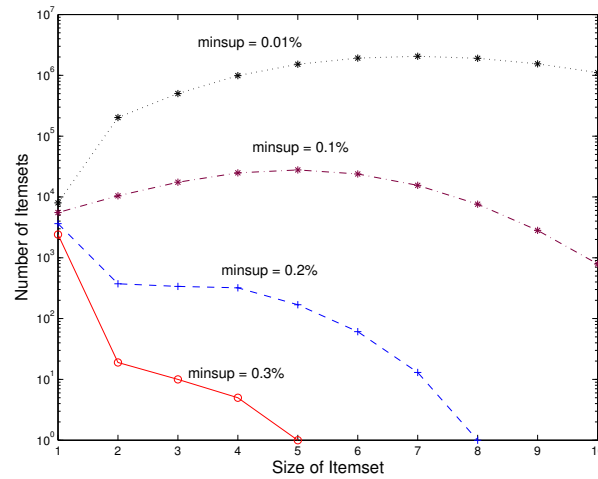


Figure 4.18. Effect of applying different minimum support threshold on number of frequent itemsets.

but high enough to be computationally tractable. Some may argue that using a single minimum support threshold alone is not sufficient because it will make the problem becomes computationally expensive. Instead, they propose to extend the association rule formulation to incorporate the use of multiple minimum support thresholds. The idea is to assign each item i a different threshold μ_i , depending on how frequent it is expected to appear in the data set. The higher is the expected frequency, the higher is the support threshold. For example, items that belong to G3 will be assigned a higher minimum support threshold compared to items that belong to G1.

There are several issues one has to consider when using multiple minimum support thresholds:

1. How to determine the minimum support threshold for an itemset given the minimum support thresholds of the individual items? A straightforward approach is to assign it to the lowest *minsup* threshold. For example, consider the situation shown in Figure 4.19. Suppose each item is assigned the following thresholds: $\mu(\text{Milk}) = 5\%$, $\mu(\text{Coke}) = 3\%$, $\mu(\text{Broccoli}) = 0.1\%$, and $\mu(\text{Salmon}) = 0.5\%$. The minimum support for any itemset $X = \{x_1, x_2, \dots, x_k\}$ is given by $\min[\mu(x_1), \mu(x_2), \dots, \mu(x_k)]$. As an example, $\mu(\text{Milk}, \text{Broccoli}) = \min[\mu(\text{Milk}), \mu(\text{Broccoli})] = 0.1\%$.
2. Support is no longer an anti-monotone measure. For example, suppose the support for $\{\text{Milk}, \text{Coke}\} = 1.5\%$ while the support for $\{\text{Milk}, \text{Coke}, \text{broccoli}\} = 0.15\%$. In this case, the itemset $\{\text{Milk}, \text{Coke}\}$ is infrequent whereas its superset $\{\text{Milk}, \text{Coke}, \text{Broccoli}\}$ is frequent, thus violating the anti-monotone property of the support measure.

One way to address the lack of anti-monotone property is by ordering the items, in ascending order, according to their minimum support thresholds. With the

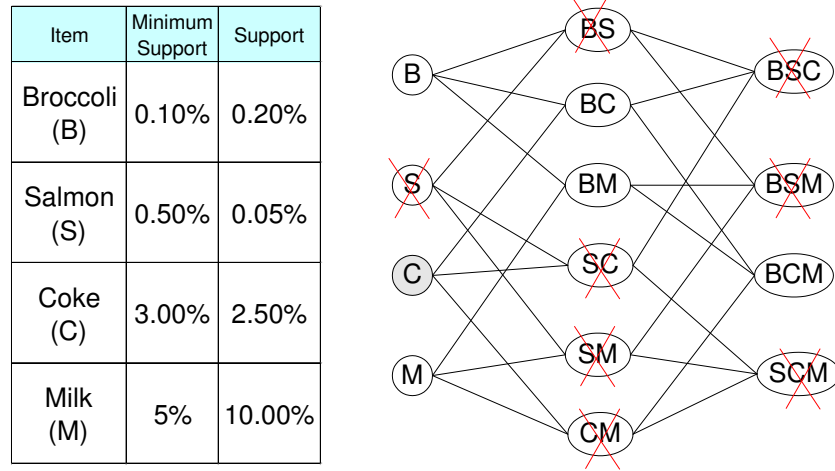


Figure 4.19. Frequent itemset generation with multiple minimum supports.

thresholds shown in Figure 4.19, the items can be ordered in the following way : Broccoli, Salmon, Coke, Milk. Based on their actual supports, only broccoli and milk will satisfy their respective minimum support thresholds. These frequent items can be used to generate larger sized frequent itemsets.

On the other hand, salmon and Coke are infrequent items. In the standard association rule mining approach, both items will be immediately pruned prior to looking for larger-size itemsets. However, when multiple minimum support thresholds are used, these two items need to be treated differently.

For salmon, since its support (0.05%) is lower than the minimum support thresholds for all items (including broccoli), it will never produce any frequent combinations with other items. Therefore, all candidate itemsets involving salmon can be immediately pruned. In the lattice structure shown in Figure 4.19, the candidate itemsets involving salmon include $\{Broccoli, Salmon\}$ and $\{Salmon, Milk\}$.

For Coke, one cannot immediately eliminate this item because it can still be part of a larger frequent itemset. For example, the candidate 2-itemset $\{Broccoli, Coke\}$ can still be frequent as long as its support is higher than 0.1%. However, one can still prune all candidate itemsets that begin with Coke. This is because such itemsets can only be extended with items that have higher minimum support thresholds. Therefore, a candidate such $\{Coke, Milk\}$ can be pruned because it is guaranteed to produce only infrequent itemsets.

The Apriori algorithm must be modified to incorporate items that have multiple minimum support thresholds. In the standard Apriori algorithm, a candidate $(k+1)$ -itemset is generated by merging two frequent itemsets of size k . The candidate is then pruned if any one of its subsets is found to be infrequent. To handle multiple minimum support thresholds, the pruning step of Apriori algorithm must be modified. Specifically, a candidate itemset is pruned only if at least one of its subsets that contain the first item (i.e., item with lowest minimum support) is infrequent. For example, $\{Broccoli, Coke, Milk\}$ can be obtained by merging the frequent 2-

itemsets $\{Broccoli, Coke\}$ and $\{Broccoli, Milk\}$. Even though $\{Coke, Milk\}$ can be infrequent, its superset, $\{Broccoli, Coke, Milk\}$, can be potentially frequent.

Finally, it is also possible to assign multiple minimum support thresholds based on the size of the itemset. This approach makes sense because the support of an itemset naturally decreases as the itemset becomes longer. Again, the problem lies in deciding the right threshold to use for a given itemset size. In this case the anti-monotone property of the support measure will also be violated.

Although the technique of using multiple minimum support thresholds can reduce the complexity of generating frequent itemsets, it cannot prevent the intermixing of items with significantly different support levels. Therefore, more research is needed in this area.

4.7 Evaluation of Association Patterns

Standard association analysis algorithms can potentially generate a large number of patterns. Just to give an example, the initial database shown in Table 4.1 contains only six items, but it can produce as many as 102 association rules at reasonable support and confidence thresholds. As the size and dimensionality of commercial databases can be very large, one might end up finding thousands or even millions of patterns, not all of which are interesting to the analysts. Thus, a key step in association analysis is to sift through the large number of extracted patterns and identifies the most interesting ones. This is not a trivial task because “one man’s trash is another man’s treasure.” It is important to establish a set of well-acceptable criteria for evaluating the interestingness of association patterns.

The first set of criteria can be derived from statistical arguments. It is intuitively clear that any pattern involving a set of independent items or covering too few transactions is statistically insignificant and thus, should be rejected. This set of criteria can be realized through the use of an *objective interestingness measure*, which evaluates a pattern based on statistics derived from the observed data. This include measures such as support, confidence, correlation, etc.

A second set of criteria can be derived from subjective arguments. Any pattern that reveals unexpected information about the data or provides actionable knowledge that can lead to more profitable uses should be considered as interesting. Such criteria are less well-defined because it is harder to measure how “surprising” or “actionable” is a pattern. Nevertheless, there has been some attempts to define such criteria through the use of *subjective interestingness measures*.

We will describe both the objective and subjective interestingness measures in the next two sections.

4.7.1 Objective Measures of Interestingness

An objective measure is a data-driven approach for evaluating the interestingness of a discovered pattern. It is domain-independent and requires minimal input from

users, except for specifying a cutoff value to eliminate uninteresting patterns. Support and confidence are examples of two objective measures originally proposed to analyze the quality of association patterns. Yet these two measures alone may not be sufficient to eliminate uninteresting patterns, as illustrated by the next example.

Example 24 *In a population of 1000 students, suppose there are 600 students who know how to swim and 700 students who know how to bike. There are only 420 students who know how to do both swimming and biking. In this example, the rule $\text{Swim} \rightarrow \text{Bike}$ has a 42% support and 70% confidence. A well-known fact from statistics states that a pair of variables, A and B , are statistically independent if the following condition holds.*

$$P(A, B) = P(A) \times P(B). \quad (4.6)$$

Since

$$\begin{aligned} P(\text{swim}) \times P(\text{bike}) &= 0.6 \times 0.7 = 0.42 \text{ and} \\ P(\text{swim}, \text{bike}) &= 0.42, \end{aligned}$$

we may conclude that students who know how to swim are independent of those who know how to bike. Thus, the rule $\text{Swim} \rightarrow \text{Bike}$ is spurious even though it has moderately high support and confidence values.

The above example illustrates the limitation of using only the support and confidence measures for evaluating association patterns. These measures may fail to detect patterns involving items that are statistically independent of each other. Keep in mind that there are other objective measures designed for eliminating statistically spurious patterns. This include the ϕ -coefficient, interest factor (I), odds ratio, and Piatetsky-Shapiro's PS measure, all of which can be employed to detect and filter out statistically independent items. We will provide a formal way to define these objective measures next.

Contingency Tables. Many objective measures can be defined in terms of the frequency counts tabulated in a contingency table, such as the one shown in Table 4.10. Each entry in this 2×2 table denotes the frequency count for one of the four possible combinations of items A and B . For example, f_{11} denotes the number of times A and B appear together in the same transaction while f_{01} denotes 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 . Likewise, f_{0+} and f_{+0} denote the number of transactions that do not contain A and B , respectively.

Given a 2×2 contingency table, we can define the ϕ -coefficient as follows,

$$\phi = \frac{f_{11}f_{00} - f_{01}f_{10}}{\sqrt{f_{1+}f_{+1}f_{0+}f_{+0}}}. \quad (4.7)$$

Table 4.10. A 2-way contingency table for variables A and B .

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

The ϕ -coefficient is an important measure because it is analogous to Pearson's correlation coefficient, a widely-used measure for computing the correlation between pairs of continuous variables. Other well-known interestingness measures include interest factor (I), odds ratio (α) and Piatetsky-Shapiro's PS measure.

$$I = \frac{P(A, B)}{P(A)P(B)} = \frac{Nf_{11}}{f_{1+}f_{+1}} \quad (4.8)$$

$$\alpha = \frac{P(A, B)P(\overline{A}, \overline{B})}{P(A, \overline{B})P(\overline{A}, B)} = \frac{f_{11}f_{00}}{f_{01}f_{10}} \quad (4.9)$$

$$PS = P(A, B) - P(A)P(B) = \frac{f_{11}}{N} - \frac{f_{1+}}{N} \frac{f_{+1}}{N} \quad (4.10)$$

Statistical Independence. All four measures defined above can be used to identify relationships between a pair of statistically independent items. To illustrate this, we can rewrite the independence condition given in Equation 4.6 in terms of the frequency counts of a 2×2 contingency table. After some manipulation, we arrive at the following condition for statistical independence.

$$Nf_{11} = f_{1+}f_{+1} \quad \text{or equivalently,} \quad f_{11}f_{00} = f_{10}f_{01}.$$

Replacing these conditions into Equations 4.7-4.10 would show that the ϕ -coefficient and PS measure are equal to zero, while the odds ratio and interest factor are equal to one, when applied to a pair of statistically independent items.

Symmetric versus Asymmetric Measures. Some objective measures are developed for analyzing rules instead itemsets. For example, given a rule $A \rightarrow B$, the confidence (c) and conviction (V) values for this rule are given below.

$$\text{Confidence, } c = \frac{f_{11}}{f_{1+}}$$

$$\text{Conviction, } V = \frac{f_{1+}f_{+0}}{Nf_{10}}$$

Measures developed for rules often have an asymmetric form. For example, confidence is an asymmetric measure since $c(A \rightarrow B) \neq c(B \rightarrow A)$. Other asymmetric measures include conviction, mutual information, J-measure, Gini index, Laplace, certainty factor, added value, and Klosgen factor. Each asymmetric measure can be transformed into a symmetric form by taking the maximum value between the rules $A \rightarrow B$ and $B \rightarrow A$. Symmetrizing these measures would allow us to apply both symmetric and asymmetric measures directly to contingency tables without concerning ourselves with the direction of the rule implication.

Range of an Objective Measure. Table 4.11 provides the definition for various objective measures along with their respective range of values. In most cases, the range is reported as two values, corresponding to the minimum and maximum values of the measure. For example, the range for support and confidence measures go from 0 to 1. In other cases, the range is reported as three values, $a \cdots s \cdots b$, where a corresponds to its minimum value, b corresponds to its maximum value, and s corresponds to the value at statistical independence. For example, the range for ϕ -coefficient goes from -1 (for perfect negative correlation) to $+1$ (for perfect positive correlation). Statistical independence is represented by the value $\phi = 0$. Note that if the range is reported as two values, either the minimum value corresponds to statistical independence (e.g., for mutual information) or the value for statistical independence can be arbitrary (e.g., for support). For other measures such as odds ratio and interest factor, their range of values can go from 0 to infinity, while the value at statistical independence is equal to 1. Such measures can be normalized to have a range between -1 and $+1$ by applying appropriate transformation functions such as

$$f(M) = \frac{M - 1}{M + 1}, \quad \text{or} \quad f(M) = \frac{\tan^{-1} \log(M)}{\pi/2}.$$

For example, Yule's Y and Q coefficients are normalized measures of odds ratio.

Measures for k -itemsets. Many of the measures defined in Table 4.11 can be generalized to evaluate patterns involving more than two items. For example, the interest factor for a k -itemset, $\{x_1, x_2, \dots, x_k\}$ is given by

$$\frac{N^{k-1} f_{11\dots 1}}{f_{1+\dots+} f_{+1+\dots+} \cdots f_{++\dots 1}},$$

where $f_{i_1 i_2 \dots i_k}$ corresponds to the frequency count in a k -way contingency table. More sophisticated techniques are available for analyzing k -way contingency tables, e.g., loglinear models, but such techniques are beyond the scope of this book. The rest of the discussion in the remainder of this section will focus primarily on 2×2 contingency tables.

Consistency among Objective Measures

Given the wide variety of available measures, it is only natural to wonder whether these measures are consistent with each other. By consistency we mean that patterns ranked highly according to one measure are also ranked highly by another measure, while those ranked lowly by the first measure will also be ranked lowly by the other.

For example, Table 4.12 shows an example of ten contingency tables, $E1$ - $E10$, defined in terms of their frequency counts f_{ij} . We can apply the various measures given in Table 4.11 to rank order these patterns, the results of which are shown in Table 4.13. The ϕ -coefficient considers $E1$ to be the most interesting pattern and $E10$ to be least interesting pattern. For these ten tables, the rankings produced by ϕ are consistent with the rankings produced by other measures such as Cohen's κ and

Table 4.11. Definitions of objective interestingness measures.

#	Measure	Range	Formula $M(A, B)$
1	ϕ -coefficient	$-1 \dots 0 \dots 1$	$\frac{N \times f_{11} - f_{1+}f_{+1}}{\sqrt{f_{1+}f_{+1}f_{0+}f_{+0}}}$
2	Goodman-Kruskal's (λ)	$0 \dots 1$	$\left[\frac{\sum_j \max_k f_{jk} + \sum_k \max_j f_{jk} - \max_j f_{j+} - \max_k f_{+k}}{2N - \max_j f_{j+} - \max_k f_{+k}} \right]$
3	Odds ratio (α)	$0 \dots 1 \dots \infty$	$\frac{f_{11}f_{00}}{f_{10}f_{01}}$
4	Yule's Q	$-1 \dots 0 \dots 1$	$\left[\frac{f_{11}f_{00} - f_{10}f_{01}}{f_{11}f_{00} + f_{10}f_{01}} \right] = \frac{\alpha - 1}{\alpha + 1}$
5	Yule's Y	$-1 \dots 0 \dots 1$	$\left[\frac{\sqrt{f_{11}f_{00}} - \sqrt{f_{10}f_{01}}}{\sqrt{f_{11}f_{00}} + \sqrt{f_{10}f_{01}}} \right] = \frac{\sqrt{\alpha} - 1}{\sqrt{\alpha} + 1}$
6	Kappa (κ)	$-1 \dots 0 \dots 1$	$\left[\frac{Nf_{11} + Nf_{00} - f_{1+}f_{+1} - f_{0+}f_{+0}}{N^2 - f_{1+}f_{+1} - f_{0+}f_{+0}} \right]$
7	Mutual Information (M)	$0 \dots 1$	$\max \left[\frac{\sum_i \sum_j \frac{f_{ij}}{N} \log \frac{Nf_{ij}}{f_{i+}f_{+j}}}{-\sum_i \frac{f_{i+}}{N} \log \frac{f_{i+}}{N}}, \frac{\sum_i \sum_j \frac{f_{ij}}{N} \log \frac{Nf_{ij}}{f_{i+}f_{+j}}}{-\sum_j \frac{f_{+j}}{N} \log \frac{f_{+j}}{N}} \right]$
8	J-Measure (J)	$0 \dots 1$	$\frac{f_{11}}{N} \log \frac{Nf_{11}}{f_{1+}f_{+1}} + \max \left[\frac{f_{10}}{N} \log \frac{Nf_{10}}{f_{1+}f_{+0}}, \frac{f_{01}}{N} \log \frac{Nf_{01}}{f_{0+}f_{+1}} \right]$
9	Gini index (G)	$0 \dots 1$	$\max \left[\begin{aligned} &\frac{f_{1+}}{N} \times \left[\left(\frac{Nf_{11}}{f_{1+}f_{+1}} \right)^2 + \left(\frac{Nf_{10}}{f_{1+}f_{+0}} \right)^2 \right] \\ &+ \frac{f_{0+}}{N} \times \left[\left(\frac{Nf_{01}}{f_{0+}f_{+1}} \right)^2 + \left(\frac{Nf_{00}}{f_{0+}f_{+0}} \right)^2 \right] \\ &- \left(\frac{f_{1+}}{N} \right)^2 - \left(\frac{f_{0+}}{N} \right)^2, \\ &\frac{f_{+1}}{N} \times \left[\left(\frac{Nf_{11}}{f_{1+}f_{+1}} \right)^2 + \left(\frac{Nf_{01}}{f_{0+}f_{+1}} \right)^2 \right] \\ &+ \frac{f_{+0}}{N} \times \left[\left(\frac{Nf_{10}}{f_{1+}f_{+0}} \right)^2 + \left(\frac{Nf_{00}}{f_{0+}f_{+0}} \right)^2 \right] \\ &- \left(\frac{f_{+1}}{N} \right)^2 - \left(\frac{f_{+0}}{N} \right)^2 \end{aligned} \right]$
10	Support (s)	$0 \dots 1$	$\frac{f_{11}}{N}$
11	Confidence (c)	$0 \dots 1$	$\max \left[\frac{f_{11}}{f_{1+}}, \frac{f_{11}}{f_{+1}} \right]$
12	Laplace (L)	$0 \dots 1$	$\max \left[\frac{f_{11}+1}{f_{1+}+2}, \frac{f_{11}+1}{f_{+1}+2} \right]$
13	Conviction (V)	$0.5 \dots 1 \dots \infty$	$\max \left[\frac{f_{1+}f_{+0}}{Nf_{10}}, \frac{f_{0+}f_{+1}}{Nf_{01}} \right]$
14	Interest (I)	$0 \dots 1 \dots \infty$	$\frac{Nf_{11}}{f_{1+}f_{+1}}$
15	Cosine (IS)	$0 \dots \sqrt{P(A, B)} \dots 1$	$\frac{f_{11}}{\sqrt{f_{1+}f_{+1}}}$
16	Piatetsky-Shapiro's (PS)	$-0.25 \dots 0 \dots 0.25$	$\frac{f_{11}}{N} - \frac{f_{1+}f_{+1}}{N^2}$
17	Certainty factor (F)	$-1 \dots 0 \dots 1$	$\max \left[\frac{\frac{Nf_{11}}{f_{1+}f_{+1}} - \frac{f_{+1}}{N}}{1 - \frac{f_{+1}}{N}}, \frac{\frac{Nf_{11}}{f_{1+}f_{+1}} - \frac{f_{1+}}{N}}{1 - \frac{f_{1+}}{N}} \right]$
18	Added Value (AV)	$-0.5 \dots 0 \dots 1$	$\max \left[\frac{Nf_{11}}{f_{1+}f_{+1}} - \frac{f_{+1}}{N}, \frac{Nf_{11}}{f_{1+}f_{+1}} - \frac{f_{1+}}{N} \right]$
19	Collective strength (S)	$0 \dots 1 \dots \infty$	$\left[\frac{f_{11}+f_{00}}{f_{1+}f_{+1}+f_{0+}f_{+0}} \times \frac{N-f_{1+}f_{+1}-f_{0+}f_{+0}}{N-f_{11}-f_{00}} \right]$
20	Jaccard (ζ)	$0 \dots 1$	$\frac{f_{11}}{f_{1+}+f_{+1}-f_{11}}$
21	Klosgen (K)	$\sqrt{\frac{2}{\sqrt{3}}-1}[2-\sqrt{3}] \dots 0 \dots \frac{2}{3\sqrt{3}}$	$\sqrt{\frac{f_{11}}{N}} \max \left[\frac{Nf_{11}}{f_{1+}f_{+1}} - \frac{f_{+1}}{N}, \frac{Nf_{11}}{f_{1+}f_{+1}} - \frac{f_{1+}}{N} \right]$

Table 4.12. Example of contingency tables.

Example	f_{11}	f_{10}	f_{01}	f_{00}
E1	8123	83	424	1370
E2	8330	2	622	1046
E3	9481	94	127	298
E4	3954	3080	5	2961
E5	2886	1363	1320	4431
E6	1500	2000	500	6000
E7	4000	2000	1000	3000
E8	4000	2000	2000	2000
E9	1720	7121	5	1154
E10	61	2483	4	7452

Table 4.13. Rankings of contingency tables using objective interestingness measures.

	ϕ	λ	α	Q	Y	κ	M	J	G	s	c	L	V	I	IS	PS	F	AV	S	ζ	K
E1	1	1	3	3	3	1	2	2	1	3	5	5	4	6	2	2	4	6	1	2	5
E2	2	2	1	1	1	2	1	3	2	2	1	1	1	8	3	5	1	8	2	3	6
E3	3	3	4	4	4	3	3	8	7	1	4	4	6	10	1	8	6	10	3	1	10
E4	4	7	2	2	2	5	4	1	3	6	2	2	2	4	4	1	2	3	4	5	1
E5	5	4	8	8	8	4	7	5	4	7	9	9	9	3	6	3	9	4	5	6	3
E6	6	6	7	7	7	7	6	4	6	9	8	8	7	2	8	6	7	2	7	8	2
E7	7	5	9	9	9	6	8	6	5	4	7	7	8	5	5	4	8	5	6	4	4
E8	8	9	10	10	10	8	10	10	8	4	10	10	10	9	7	7	10	9	8	7	9
E9	9	9	5	5	5	9	9	7	9	8	3	3	3	7	9	9	3	7	9	9	8
E10	10	8	6	6	6	10	5	9	10	10	6	6	5	1	10	10	5	1	10	10	7

the collective strength S . On the other hand, these rankings are quite inconsistent with the rankings produced by other measures such as interest factor (I) and added value (AV). For example, I considers $E10$ to be the most interesting pattern and $E3$ to be the least interesting pattern, which is quite contradictory to the evaluation made by the ϕ -coefficient. Looking at the rankings produced in Table 4.13, we can identify several groups of consistent measures such as (1) the group of ϕ -coefficient, κ , and collective strength (S) measures; (2) the group of odds ratio (α), Yule's Q and Y coefficients; (3) the group of confidence (c), Laplace (L), and conviction (V) measures; and (4) the group of cosine (IS) and Jaccard (ξ) measures.

The issue of consistency has a direct impact on the choice of measure used for association analysis. If all measures are consistent with each other, then any one of the measure is equally good (or bad) at evaluating the discovered patterns. On the other hand, if the measures are inconsistent, then it is important to know which measure has the desirable properties for ranking a set of association patterns.

Properties of Objective Measures

The discussion presented in the previous section suggests that a significant number of existing measures provide conflicting information about the interestingness of a pattern. It would be useful to know the inherent properties of these measures that make them prefer certain patterns to others. We review some of these properties in this section.

Effect of Increasing Support of a Pattern. One useful property to consider is the effect of increasing support on the magnitude of a measure. This property can be stated more formally as follows.

Property P1:

An objective measure M should increase monotonically with increasing value of f_{11} when both f_{1+} and f_{+1} remain the same.

We illustrate the intuition behind this property with the following example. Consider the two contingency tables shown in Table 4.14. Both tables have identical row and column sums, which mean that the support for A and B remain unchanged. However, the table on the right has a higher frequency count for $\{A, B\}$. Since the support for each item is fixed, the frequency counts for f_{10} and f_{01} must decrease to compensate for the higher value of f_{11} . In turn, the decreasing values of f_{01} and f_{10} result in a higher frequency count for f_{00} . Observe that the frequency counts for the table on the right are concentrated mostly along the (f_{11}, f_{00}) diagonal, which suggests that A and B are more likely to co-occur together or to be absent altogether in the same transaction (rather than to have one item without the other). Thus, the relationship between A and B can only grow stronger with increasing support of $\{A, B\}$.

	B	\overline{B}	
A	60	80	140
\overline{A}	120	160	280
	180	240	420

	B	\overline{B}	
A	100	40	140
\overline{A}	80	200	280
	180	240	420

Table 4.14. Example of a contingency table.

Effect of Increasing Item Support. Another useful property of a measure is to examine its behavior when the support of its constituent items increases without changing the support of the pattern.

Property P2:

An objective measure M should monotonically decrease with increasing value of f_{1+} (or f_{+1}) when all other parameters remain the same.

Consider the diagram shown in Figure 4.20. Let W be the set of transactions that contain A_1 but not B and Y be the set of transactions that contain A_2 but not B . Also, suppose X corresponds to the set of transactions that contain A_1 and B while Z corresponds to the set of transactions that contain A_2 and B . Both X and Z are assumed to have equal number of transactions. However, the support for A_1 is assumed to be much higher than the support for A_2 . One would expect the association between A_1 and B to be much stronger than the association between A_2 and B since most of the transactions involving item A_1 also contain item B ,

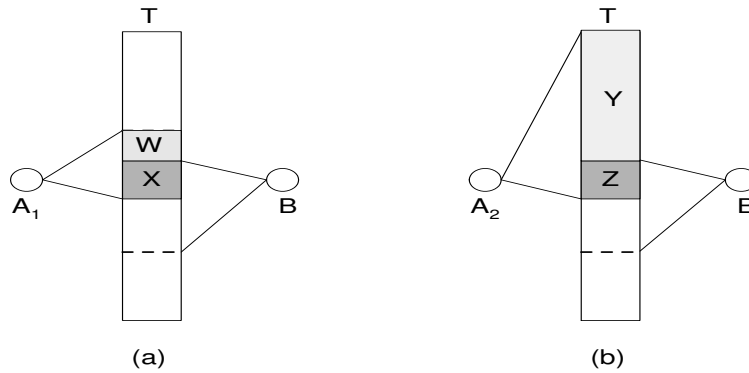


Figure 4.20. An example illustrating the effect of increasing item support.

whereas most of the transactions involving A_2 do not contain B . This observation eventually leads to the property P2 stated above.

Table 4.15 illustrates the extent to which some of the existing measures satisfy the two properties described above.

Effect of Inversion. The association analysis presented in this chapter assumes that the input data can be transformed into an asymmetric binary form before applying algorithms such as Apriori or FP-tree. However, because of the asymmetric binary nature of the data, it would seem most logical to evaluate the association between a pair of items on the basis of how frequent they are present together, rather than how frequent they are both missing from the same transaction. For example, it makes sense to regard the association between bread and milk as strong because they are bought together in many transactions. It does not make sense to regard the association between caviar and gold earring as strong just because they are both missing from most of the transactions.

To illustrate this, consider the example shown in Figure 4.21. Each column vector corresponds to an item, with a 0/1 bit indicating whether a transaction contains the particular item. For example, the column vector for A indicates that item A belongs to the first and last transaction. The first pair of vectors, A and B , correspond to items that are rarely present in any transactions, such as caviar and gold earring. The second pair of vectors, C and D , correspond to items that are frequently bought together, such as milk and bread. For asymmetric binary data such as market-basket transactions, the association between C and D should be considered as stronger than the association between A and B .

A closer examination of the vectors would reveal that C is actually related to vector A — its bit has been inverted from 0's (absence) to 1's (presence), and vice-versa. This process of flipping the bit vector is known as the inversion operation. Given an asymmetric binary data, it is important to consider how an objective measure behaves under the inversion operation. If the measure is invariant under this operation, it is called a symmetric binary measure; otherwise, it is called an asymmetric binary measure. Examples of symmetric binary measures include ϕ , odds ratio, κ

Table 4.15. Properties of objective interestingness measures.

Symbol	Measure	P1	P2	P3	P4	P5
ϕ	ϕ -coefficient	Yes	Yes	Yes	No	No
λ	Goodman-Kruskal's	No	No	Yes	No	No
α	odds ratio	Yes	Yes	Yes	No	Yes
Q	Yule's Q	Yes	Yes	Yes	No	Yes
Y	Yule's Y	Yes	Yes	Yes	No	Yes
κ	Cohen's	Yes	Yes	Yes	No	No
M	Mutual Information	Yes	Yes	Yes	No	No
J	J-Measure	No	No	No	No	No
G	Gini index	No	No	Yes	No	No
s	Support	Yes	No	No	No	No
c	Confidence	Yes	No	No	Yes	No
L	Laplace	Yes	No	No	No	No
V	Conviction	Yes	No	Yes	No	No
I	Interest	Yes	Yes	No	No	No
IS	Cosine	Yes	Yes	No	Yes	No
PS	Piatetsky-Shapiro's	Yes	Yes	Yes	No	No
F	Certainty factor	Yes	Yes	Yes	No	No
AV	Added value	Yes	Yes	No	No	No
S	Collective strength	Yes	Yes	Yes	No	No
ζ	Jaccard	Yes	Yes	No	Yes	No
K	Klosgen's	Yes	Yes	No	No	No

and collective strength, while the examples for asymmetric binary measures include I , IS , PS and Jaccard measure. Asymmetric binary measures are more suitable for handling asymmetric binary data compared to symmetric binary measures. For example, the ϕ -coefficient, which is a symmetric binary measure, considers A and B to have the same degree of association as C and D . In fact, the ϕ -coefficient for C and D is much weaker than the ϕ -coefficient for E and F , which is rather counter-intuitive from a market-basket analysis perspective! (Nevertheless, the ϕ -coefficient may still be useful to eliminate patterns due to statistically independent items, as described in the previous section.)

Property P3:

An objective measure M is symmetric binary if it is invariant when exchanging f_{11} with f_{00} and f_{10} with f_{01} .

Effect of Null Addition. Suppose we are interested in analyzing the relationship between a pair of words, say, **data** and **mining**, using a data set consisting of Computer Science journal articles. If we add a collection of Sports news articles to the data set, should the association between **data** and **mining** be affected?

The above example is an illustration of the null addition operation. If we represent each item as a column vector, then the null addition operation is equivalent to padding the pair of column vectors with more 0 bits.

Property P4:

An objective measure M is null-invariant if it is not affected by increasing f_{00} , while all other parameters are the same.

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

(a) (b) (c)

Figure 4.21. 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 .

For applications such as document analysis or market-basket analysis, we would expect the chosen measure to remain invariant under such operation. Otherwise, the relationship between the words **data** and **mining** could become independent by simply adding enough documents that do not contain both words! However, the only null invariant measures listed in Table 4.11 are confidence, cosine (IS) and Jaccard (ξ) measures. In general, the property P4 is useful for sparse transaction data sets, where there are considerably more 0's than 1's in the data set. As a result, co-presence of items together in the same transaction is more interesting than co-absence.

Effect of Row and Column Scaling. We now examine another property of an objective measure that is more suitable for symmetric binary data sets. Suppose we are interested in analyzing the performance of students on the basis of their gender. Table 4.16 illustrates the relationship between gender and grade of students for a particular Computer Science course in the years 1993 and 2003. In this toy example, notice that the data for 2003 is obtained by doubling the number of male students for 1993 and increasing the number of female students by a factor of 3. The male students in 2003 are not performing any better than those in 1993 since the proportion of male students who achieve a high grade to low grade is still the same, i.e., 3:4. Similarly, the female students in 2003 are performing no better than those in 1993. Thus, we expect the relationship between grade and gender to remain the same for both years given the frequency distribution shown in both tables. Any intuitive objective measure should consider the association in both tables as being equal, even though the sample sizes are different.

This property can be summarized by examining the behavior of an objective measure after re-scaling the rows or columns of a contingency table.

Table 4.16. The Grade-Gender example.

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

(a) Sample data from 1993.

(b) Sample data from 2003.

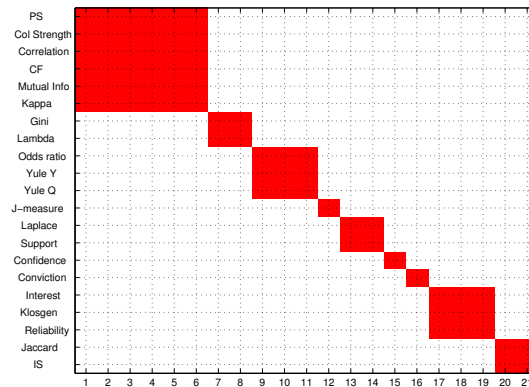
Property P5:

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 $[k_1k_3f_{11}; k_1k_3f_{10}; k_1k_4f_{01}; k_2k_4f_{00}]$, and k_1, k_2, k_3, k_4 are positive constants.

From Table 4.15, only odds ratio (α) as well as Yule's Q and Y coefficients are invariant under the row and column scaling operations. All other symmetric binary measures such as the ϕ -coefficient, κ , Gini index (G), mutual information (M) and collective strength (S) would change when the rows or columns of the contingency table are re-scaled by constant factors.

Summary

From the five properties listed in Table 4.15, we can identify several groups of measures that have identical properties, as shown in Figure 4.22. Some of these groupings are quite natural (e.g., odds ratio with Yule's measures or Jaccard with cosine measure), while others are quite unexpected, e.g., the group of support and Laplace measure. In the latter case, the measures are grouped together because the properties shown in Table 4.15 are not comprehensive enough to distinguish these measures. There may be some additional properties we can add to this list (e.g., monotonicity, statistical independence, etc.) to discriminate some of these measures.

**Figure 4.22.** Similarity between measures in terms of the five properties listed in Table 4.15.

For association rules, we expect the objective measure to be asymmetric binary (i.e., violates property P3), satisfies the properties for P_1 , P_2 , and P_4 , has a fixed (perhaps, minimum) value for statistical independence, as well as the ability to capture the notion of predictive accuracy. Even though the cosine and Jaccard measures possess most of these desirable properties, they do not capture the notions of statistical independence and predictive accuracy well. We do not expect any measure to capture all this criteria because some of the properties may be conflicting. For example, it is difficult to reconcile the requirements of having a fixed value for statistical independence while being an asymmetric binary measure. Instead, one may apply multiple measures to ensure that most of the requirements are satisfied. In practice, support and confidence are still reasonable measures as long as they are complemented by other measures such as the ϕ -coefficient, interest factor (I), collective strength, conviction, or odds ratio to eliminate statistically independent patterns.

Effect of Support-based Pruning

Support-based pruning is often used as a pre-filter prior to the application of other objective measures such as confidence, ϕ -coefficient, interest factor, etc. Because of its anti-monotone property, support allows us to effectively prune the exponential number of candidate patterns. Beyond this, little else is known about the advantages of applying this strategy. The purpose of this section is to discuss two additional effects it has on the rest of the objective measures.

Elimination of Poorly-Correlated Patterns First, we will analyze the quality of patterns eliminated by support-based pruning. Ideally, we prefer to eliminate only patterns that are poorly correlated. Otherwise, we may end up missing too many interesting patterns.

To study this effect, we have created a synthetic data set that contains 100,000 2×2 contingency tables. Each table contains randomly populated f_{ij} values subjected to the constraint $\sum_{i,j} f_{ij} = 1$. The support and ϕ -coefficient for each table can be computed using the formula shown in Table 4.11. By examining the distribution of ϕ -coefficient values, we can determine whether there are any highly correlated patterns inadvertently removed as a result of support-based pruning. The result of this experiment is shown in Figure 4.23.

For the entire data set of 100,000 tables, the ϕ -coefficients are normally distributed around $\phi = 0$, as depicted in the upper left-hand corner of both graphs. The remaining figures show the results of applying three different minimum support thresholds (1%, 3%, and 5%) to eliminate low-support tables. Observe that by applying a minimum support threshold, we will eliminate mostly contingency tables that are either uncorrelated ($\phi = 0$) or negatively correlated ($\phi < 0$). This observation is quite intuitive because, for a contingency table with low support, at least one of the values for f_{10} , f_{01} or f_{00} must be relatively high to compensate for the low frequency count in f_{11} . Such tables tend to be uncorrelated or negatively correlated unless their f_{00} values are extremely high.

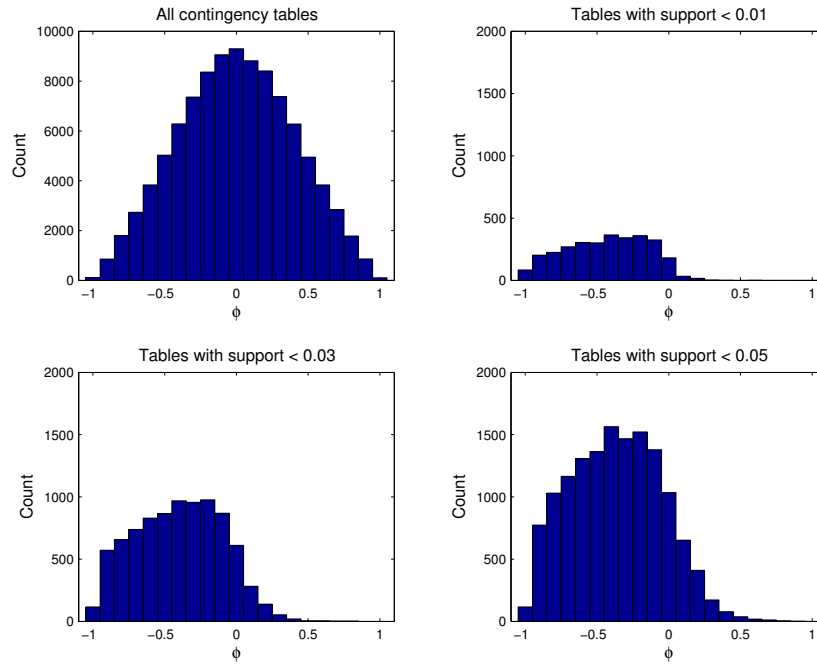


Figure 4.23. Effect of Support Pruning on Contingency tables.

Support-based pruning is a viable technique as long as only positively correlated tables are of interest to the data mining application. One such situation arises in market basket analysis where such a pruning strategy is used extensively.

Consistency of Measures under Different Support Constraints Support-based pruning also affects the issue of consistency among objective measures. To illustrate this, consider the diagram shown in Figure 4.24. This figure is obtained by generating a synthetic data set similar to the previous section except that the contingency tables are non-negatively correlated. Convex measures such as mutual information, Gini index, J-measure, and λ assign positive values to their negatively-correlated tables. Thus, they tend to prefer negatively correlated tables over uncorrelated ones, unlike measures such as ϕ -coefficient, Yule's Q and Y , PS , etc. To avoid such complication, our synthetic data set for this experiment is restricted only to uncorrelated and positively correlated tables.

We can determine the consistency between every pair of measures by computing the correlation between their ranking vectors. Figure 4.24 depicts the pair-wise correlation when various support bounds are imposed. The darker cells indicate that the correlation between the pair of measures is approximately greater than 0.8.

For this analysis, we apply two kinds of support-based pruning strategies. The first strategy is to impose a minimum support threshold on the value of f_{11} . This approach is identical to the support-based pruning strategy employed by most of the association analysis algorithms. The second strategy is to impose a maximum support threshold on both f_{1+} and f_{+1} . This strategy is equivalent to removing the

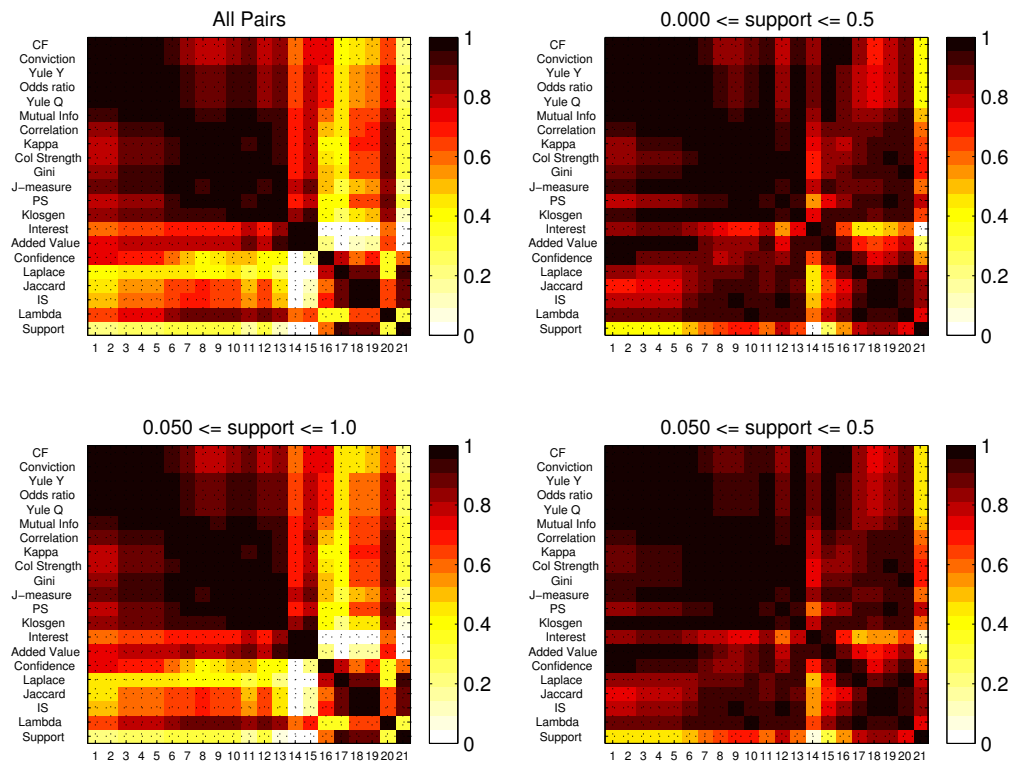


Figure 4.24. Similarity between measures at various ranges of support values. Note that the column labels are the same as the row labels.

most frequent items from a data set (e.g., staple products such as sugar, bread, and milk).

Initially, without support pruning (the upper left diagram), notice that most of the highly correlated measures agree with the groupings identified in Figure 4.22. If we start applying a maximum support threshold to the contingency tables, as shown by the upper right-hand diagram, the region of dark cells begins to grow, indicating that more measures are becoming highly correlated with each other. In the case of no support pruning, nearly 40% of the pairs have correlation above 0.85. With maximum support pruning, this percentage increases to more than 68%. For example, interest factor, which is quite inconsistent with almost all other measures except for added value, have become more consistent when high-support items are removed. This observation can be explained as an artifact of interest factor. Consider the contingency tables shown in Table 4.17, where A and B correspond to a pair of uncorrelated items, while C and D correspond to a pair of perfectly correlated items. However, because the support for item C is very high, $I(C, D) = 1.0112$, which is close to the value for statistical independence. By eliminating the high support items, we may resolve this type of inconsistency between interest factor and other objective measures.

Table 4.17. Effect of high-support items on interest factor.

A	B	\overline{B}		C	D	\overline{D}	
	100	200	300		890	0	890
	200	400	600		0	10	10
	300	600	900		890	10	900
\overline{A}				\overline{C}			

$$(a) \begin{aligned} I(A, B) &= 1, \\ \phi(A, B) &= 0. \end{aligned}$$

$$(b) \begin{aligned} I(C, D) &= 1.012, \\ \phi(C, D) &= 1. \end{aligned}$$

The bottom left diagram seems to suggest that there is very little improvement in the consistency among measures when the minimum support threshold is applied. Nevertheless, when used along with a maximum support threshold, the correlations among measures do show some slight improvements compared to applying the maximum support threshold alone — in fact, more than 71% of the pairs have correlation above 0.85. This analysis suggests that imposing a tighter bound on the support of association patterns may force many measures become highly correlated with each other.

4.7.2 Subjective Measures of Interestingness

Proponents of subjective measures have argued that objective measures alone may not be sufficient to capture all the intricacies of the knowledge discovery process. For example, a rule that agrees with the prior knowledge or expectation of the domain analysts is often considered to be uninteresting. A case in point is the association rule `bread` \rightarrow `milk`, which may have high support and high confidence, yet it is

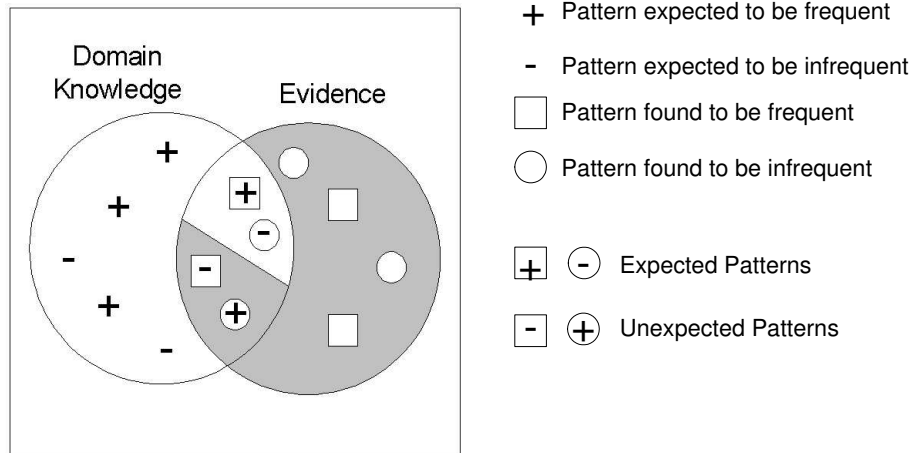


Figure 4.25. Unexpected subjective measure

quite obvious and not too useful for analysts.

In general, a pattern is considered to be subjectively interesting if it contradicts with the expectation of the analyst or is potentially actionable. Subjectively interesting patterns can appear in three different forms, (1) patterns that are both unexpected and actionable, (2) patterns that are unexpected but not actionable, and (3) patterns that are expected but actionable. Many data analysts are interested in finding rules which they can act on for their own benefits. Even though actionability is a key measure, in practice, it is often difficult to capture this concept formally. Nevertheless, in most cases, the unexpectedness of a pattern may provide a good indication of actionability because most of the unexpected patterns are actionable.

The unexpectedness of a rule can be modeled by defining the belief set of the problem domain. A belief set can be classified into two categories :

1. **Hard beliefs**, which are beliefs that can never be changed despite contradictory evidence.
2. **Soft beliefs**, which are beliefs that a user is willing to compromise if compelling new evidence are found.

Interesting rules for hard beliefs are defined to be rules that contradict the hard beliefs. For soft beliefs, one can define the interestingness measure in terms of how much the existing belief changed as a result of the observed pattern. There are several ways to model the soft beliefs using techniques from Bayesian statistics, Dempster-Shafer theory, frequentist approach, Cyc's approach and hypothesis testing methods. However, this approach assumes that the initial belief set of the domain expert encompasses the entire universe of discourse. This assumption can be relaxed to include rules that are previously unknown to the users. These are also known as unexpected condition rules.

The unexpected aspect of a subjective measure is illustrated by the Venn diagram in Figure 4.25. Each element in the Venn diagram represents a logical proposition.

H is the set of initial beliefs about the domain while E is the set of evidence obtained from the discovered patterns. Each proposition can have a truth value of TRUE (represented in the diagram by a positive or rectangle symbol) or FALSE (represented by a negative or circle symbol). The notion of unexpectedness can be defined in terms of patterns that contradict prior beliefs or patterns that are previously unknown to the users. The contradictory and novel patterns are illustrated by the shaded region in Figure 4.25.

4.8 Generalization of Association Analysis

In this section, we describe several extensions to the original association rule mining formulation. While some of these extensions are motivated by the need to reduce the number of extracted patterns, others are motivated by the need to describe new types of relationships, such as negative associations, sequential associations, as well as the more complex relationships in graph structures. We will describe these extensions in the remainder of this section.

4.8.1 Maximal and Closed Frequent Itemsets

Frequent itemset generation algorithms such as Apriori and FP-tree tend to produce a large number of patterns. Some of these patterns are simply subsets of larger frequent itemsets, while others can be redundant because they have identical support as their supersets. It would be advantageous to eliminate some of these subset patterns, especially those that are redundant, before performing further analysis.

Maximal and closed frequent itemsets are some of the concepts introduced to identify a minimal representative set of patterns from which all other frequent itemsets can be derived. To motivate the idea behind these concepts, consider the database shown in Table 4.18, which contains 10 transactions and 30 items. The items can be divided into three groups, (1) Group *A*, which contains the ten items A_1 through A_{10} , (2) Group *B*, which contains the ten items B_1 through B_{10} , and (3) Group *C*, which contains the ten items C_1 through C_{10} . The data is also constructed in such a way that items within each group are perfectly associated with each other, but they do not appear with items from another group.

By applying a minimum support threshold of less than 30%, all subsets of items within each group will become frequent. Since the longest frequent itemset one can construct from each group is of size 10, there are altogether $3 \times \sum_{k=1}^{10} \binom{10}{k} = 3069$ frequent itemsets. In addition, given the way the data is constructed, all frequent itemsets from the same group have identical support counts. For example, the support for the frequent itemsets $\{A_1, A_2\}$, $\{A_2, A_3\}$, and $\{A_4, A_5, A_8, A_9, A_{10}\}$ are identical to the support for their longest superset, $\{A_1, A_2, \dots, A_{10}\}$. The support for all frequent itemsets from group *B* are also identical to each other, and likewise for itemsets from group *C*.

In this example, the three longest frequent itemset, $\{A_1, A_2, \dots, A_{10}\}$, $\{B_1, B_2, \dots, B_{10}\}$, and $\{C_1, C_2, \dots, C_{10}\}$, may serve as the representative set of patterns from which

Table 4.18. A transaction database for mining closed itemsets.

TID	A_1	A_2	...	A_{10}	B_1	B_2	...	B_{10}	C_1	C_2	...	C_{10}
1	1	1	...	1	0	0	...	0	0	0	...	0
2	1	1	...	1	0	0	...	0	0	0	...	0
3	1	1	...	1	0	0	...	0	0	0	...	0
4	0	0	...	0	1	1	...	1	0	0	...	0
5	0	0	...	0	1	1	...	1	0	0	...	0
6	0	0	...	0	1	1	...	1	0	0	...	0
7	0	0	...	0	0	0	...	0	1	1	...	1
8	0	0	...	0	0	0	...	0	1	1	...	1
9	0	0	...	0	0	0	...	0	1	1	...	1
10	0	0	...	0	0	0	...	0	1	1	...	1

all other frequent itemsets can be inferred. It may be sufficient to retain only these three patterns, rather than the entire 3069 frequent itemsets, when presenting them to the analysts.

Maximal Frequent Itemsets

Definition 5 *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 4.26. The itemsets can be separated into two groups, those that are frequent against those that are infrequent. Every node located above the frequent itemset border, illustrated by the dash line, represents a frequent itemset while nodes located below the border (the shaded nodes) represents an infrequent itemset. Among the frequent itemsets located near the border, only $\{A,D\}$, $\{A,C,E\}$, and $\{B,C,D,E\}$ are considered as maximal frequent itemsets because all of their immediate supersets are infrequent, i.e., they reside on the other side of the border. In contrast, a node such as $\{A,C\}$, which also resides along the frequent itemset border, is non-maximal because one of its immediate supersets, $\{A,C,E\}$, is frequent.

Maximal frequent itemsets can be used to enumerate all frequent itemsets in the data. The reason being that each non-maximal frequent itemset is a subset of at least one maximal frequent itemset. Another advantage of maximal frequent itemsets is that they can be discovered much more rapidly compared to traditional frequent itemsets. For example, one may apply an algorithm that traverses the itemset lattice in a depth-first manner until a maximal frequent itemset is found. Instead of backtracking to the root, the algorithm can then jump to the next branch of the lattice that has not been explored before, and perform a depth-first search to identify the next maximal frequent itemset.

Closed Frequent Itemsets

Although maximal frequent itemsets provide a minimal representation for all frequent itemsets, they do not contain the support information for their subsets. For

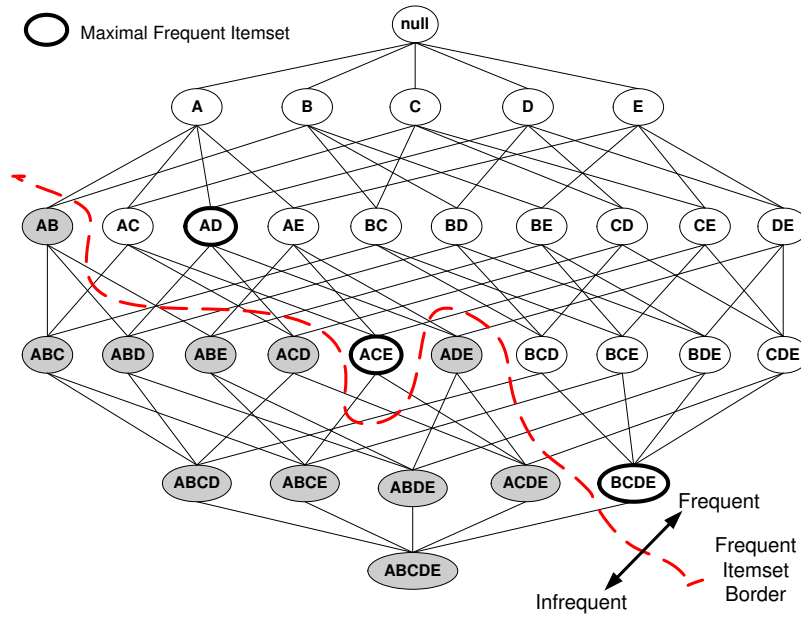


Figure 4.26. Maximal frequent itemset.

example, suppose $\{A, C, E\}$ is found to be a maximal frequent itemset. With this information, we can only deduce that $\{A, E\}$ is also a frequent itemset based on the anti-monotone property of the support measure. There is no way for us to know the exact support count for $\{A, E\}$ unless we scan the transaction database. Thus, although the maximal frequent itemsets allow us to identify the entire set of frequent itemsets, an additional database scan is needed to determine the actual support counts of the non-maximal frequent itemsets.

The notion of closed itemsets can be used to alleviate this problem. A formal definition of closed itemset is presented below.

Definition 6 An itemset X is a closed itemset if there exists no itemset X' such that (1) X' is a superset of X , and (2) all the transactions that contain X also contain X' .

In other words, a closed itemset is an itemset for which all of its supersets must have smaller support counts than itself. Furthermore, a closed itemset that has support greater than or equal to $minsup$ is known as a *frequent closed itemset*.

A detailed example of closed itemsets is presented in Figure 4.27. Given the transactions shown in this figure, we can associate each node (itemset) in the lattice with a list of transaction ids. For example, the node $\{B, C\}$ is associated with transaction ids 1, 2, and 3. To determine whether $\{B, C\}$ is a closed itemset, we need to examine the support of its immediate supersets, $\{A, B, C\}$, $\{B, C, D\}$, and $\{B, C, E\}$. Since none of these itemsets have the same support count as $\{B, C\}$, it is a closed itemset. Assuming that the minimum support threshold is 40%, $\{B, C\}$ is also considered to be a frequent closed itemset because it has a 60% support. Conversely,

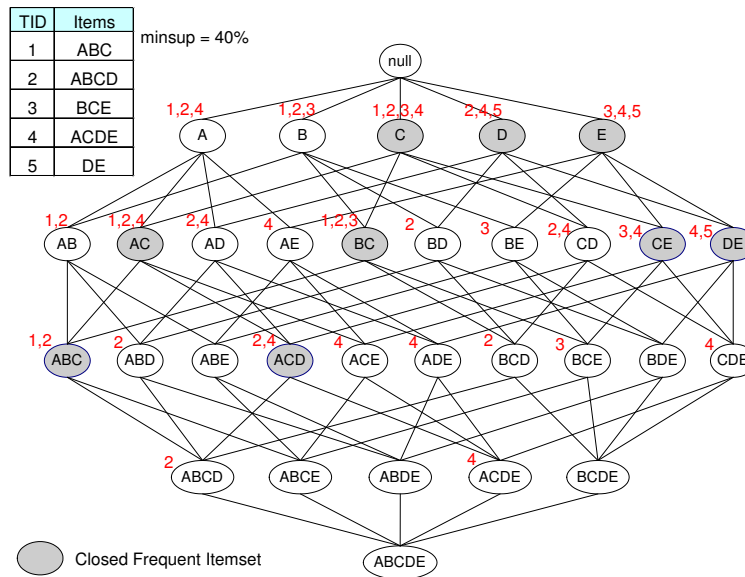


Figure 4.27. An illustrative example of frequent closed itemsets (with minimum support count equals to 2).

$\{A,B\}$ is not closed because it has the same support as one of its superset, $\{A,B,C\}$. All frequent closed itemsets in this diagram are represented by shaded nodes.

Finally, it is important to note that all maximal frequent itemsets are subsets of frequent closed itemsets. The relationships between frequent, maximal frequent, and closed frequent itemsets are shown in Figure 4.28.

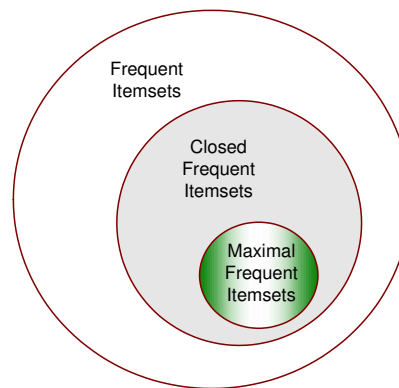


Figure 4.28. Relationships among frequent itemsets, maximal frequent itemsets, and closed frequent itemsets.

**** Not sure where to put this ****

Note that the notion of redundant patterns presented in this section is slightly different than the notion of redundant patterns described in Sections 4.4 and 4.5.

For those sections, a long pattern is considered to be redundant if there exists a shorter pattern that has similar support and confidence. For maximal and closed frequent itemsets, specific patterns are preferred over more general patterns because they can be used to derive all other subset patterns.

4.8.2 Infrequent Patterns

The entire premise of association analysis is based on the assumption that presence of items together in the same transaction is more interesting than their absence. As a result, patterns involving items that are rarely present together are often considered to be uninteresting and can be eliminated using the support measure. For example, {caviar, DVD} is an itemset involving a pair of unrelated items that are rarely purchased together in the same transaction. Such patterns are known as *infrequent patterns*.

Definition 7 *An infrequent pattern is an itemset or rule whose support is less than minsup.*

There are situations in which some of the infrequent patterns may be interesting. From a statistical perspective, infrequent patterns may be indicative of negative correlations, which are interesting relationships in their own right. For example, the lack of support for the sale of DVD and VCR together is primarily because these two items are negatively correlated with each other, i.e., any transaction that contains DVD will most likely not contain VCR, and vice-versa. Finding such patterns may allow us to discover *competitive items*, i.e., items that can be substituted for one another, such as DVD versus VCR, desktop versus laptop computers, and tea versus coffee.

From a subjective perspective, some infrequent patterns may signify the presence of interesting rare events or exceptions from the normal behavior of the population. For example, although the rule **Fire=Yes** \longrightarrow **Alarm=Off** may be infrequent, it is an interesting pattern because it may suggest a faulty alarm system. The trouble is, most transaction databases can produce an enormous number of infrequent patterns, many of which are not interesting. We shall describe the different ideas proposed for identifying interesting infrequent patterns in the remainder of this section.

Negative Patterns

Let $I = \{i_1, i_2, \dots, i_d\}$ be a set of items. For each item i_k , we can define a *negative item*, $\overline{i_k}$, to denote the absence of i_k from a given transaction. For example, $\overline{\text{coffee}}$ is a negative item whose value is 1 for each transaction that does not contain **coffee**. Based on this, we can define the concept of negative itemsets and negative association rules in the following way.

Definition 8 *A negative itemset X is defined as an itemset that satisfies the following properties: (1) $X = A \cup \overline{B}$, where A is a set of positive items, \overline{B} is a set of negative items, $|\overline{B}| \geq 1$, and (2) $s(X) \geq \text{minsup}$.*

Definition 9 A negative association rule is an association rule extracted from the itemset $A \cup \overline{B}$ and satisfies the following properties: (1) the support of the rule is greater than or equal to minsup and (2) the confidence of the rule is greater than or equal to minconf .

We will refer to the negative itemsets and negative association rules as *negative patterns* in the remainder of this section. An example of a negative association rule is $\text{tea} \rightarrow \overline{\text{coffee}}$, which suggests that people who drink tea tend to not drink coffee.

Negatively-Correlated Patterns

The definition of a negative pattern presented so far in this section does not guarantee that the relationship found is negatively correlated.

Definition 10 An itemset $X = \{x_1, x_2, \dots, x_k\}$ is a negatively-correlated pattern if the following condition holds: $P(\{x_1, x_2, \dots, x_k\}) < \prod_{j=1}^k P(x_j)$, where $P(\cdot)$ is the probability that a transaction contains the given set of items.

Recall that the right hand side expression corresponds to the probability that a group of items being statistically independent of each other. In practice, these probabilities can be estimated by the support of the corresponding items or itemsets:

$$P(X) \approx s(X), \quad \text{and} \quad P(x_i) \approx s(\{x_i\})$$

Thus, the condition for a negative-correlated pattern can be interpreted in the following way. If the support of an itemset is less than the support predicted according to the statistical independence assumption, then the itemset is negatively correlated. For example, given a pair of items, x and y , the condition for negative correlation is

$$\frac{\sigma(x, y)}{N} < \frac{\sigma(x)}{N} \times \frac{\sigma(y)}{N},$$

where N is the total number of transactions. We can also express this condition in terms of the support counts for positive and negative items of A and B .

$$\sigma(A, B) \times \sigma(\overline{A}, \overline{B}) < \sigma(A, \overline{B}) \times \sigma(\overline{A}, B).$$

The proof for this is left as an exercise.

Relationship between Infrequent Patterns, Negative Patterns and Negatively-Correlated Patterns.

So far, we have defined three closely related concepts, namely, infrequent patterns, negative patterns, and negatively-correlated patterns. The relationships among these concepts are illustrated in Figure 4.29.

First, note that most of the infrequent patterns are negative patterns. This is because if $s(A, B)$ is low, then one of $s(A, \overline{B})$, $s(\overline{A}, B)$, or $s(\overline{A}, \overline{B})$ must be high,

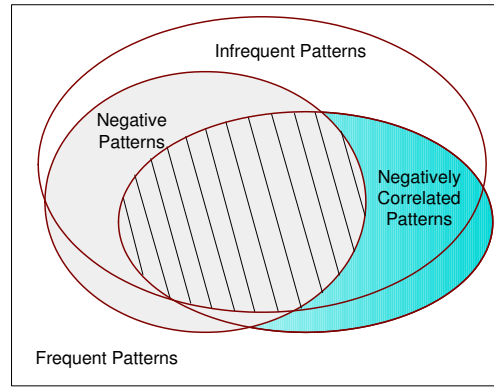


Figure 4.29. Relationships among infrequent patterns, negative patterns, and negatively correlated patterns.

indicating the presence of a negative pattern. Nevertheless, not all negative patterns are infrequent patterns, and vice-versa. This is because even if the support of a negative pattern such as $\{A, \bar{B}\}$ is higher than $minsup$, there is no guarantee that the support of the positive itemset $\{A, B\}$ will be less than $minsup$. Similarly, if $minsup$ is set too high, it is possible that none of $\{A, B\}$, $\{A, \bar{B}\}$, $\{\bar{A}, B\}$, and $\{\bar{A}, \bar{B}\}$ will be able to pass the minimum support threshold. In general, negative patterns whose positive counterparts are frequent tend to be uninteresting because the $minsup$ could have been chosen to be too small such that any combination of positive and negative items would have easily pass the $minsup$ threshold. Thus, most analysts are interested in a negative pattern $X = A \cup \bar{B}$ with the following additional restriction, $s(A \cup B) < minsup$.


Second, note that most of the negatively-correlated patterns are also negative patterns. This is because a negatively-correlated pattern tends to have larger product of $\sigma(A, \bar{B}) \times \sigma(\bar{A})$ compared to $\sigma(A, B) \times \sigma(\bar{A}, \bar{B})$. This type of situation may arise when (1) $\{A, \bar{B}\}$ or $\{\bar{A}, B\}$ is relatively large, (2) $\{A, \bar{B}\}$ or $\{\bar{A}, B\}$ is moderately large but $\{A, B\}$ is very small, or (3) $\{A, \bar{B}\}$ or $\{\bar{A}, B\}$ is moderately large but $\{\bar{A}, \bar{B}\}$ is very small. The first two cases suggest that a corresponding negative pattern should exist for the negatively-correlated pattern. The third case refers to the situation when both A and B appear very frequently in the data set, but their joint support is much lower than their expected support, a situation that is less likely to happen compared to the first two cases.

In general, infrequent patterns that are most interesting correspond to negative patterns that are negatively correlated and whose positive counterparts are frequent. We will use the term *negative association pattern* when referring to such patterns.

Definition 11 A negative association pattern corresponds to an itemset or rule involving a set of positive and negative items, $X = A \cup \bar{B}$, such that: (1) $s(A, B) \leq minsup$ but $s(A, \bar{B}) \geq minsup$, and (3) A and B are negatively correlated.

Negative association patterns are depicted by the region with crossed lines in Figure 4.29.

TID	Items								
1	{A,B}								
2	{A,B,C}								
3	{C}								
4	{B,C}								
5	{B,D}								



TID	A	\bar{A}	B	\bar{B}	C	\bar{C}	D	\bar{D}
1	1	0	1	0	0	1	0	1
2	1	0	1	0	1	0	0	1
3	0	1	0	1	1	0	0	1
4	0	1	1	0	1	0	0	1
5	0	1	1	0	0	1	1	0

Original Transactions
Transactions with Negative Items

Figure 4.30. Augmenting data set with negative items.

Techniques for Mining Negative Patterns

For negative patterns, since both positive and negative items are equally important, each item should be treated as a symmetric, rather than asymmetric, binary variable. Using the approach described earlier in Section 4.4, we can augment the original transactions with new variables that denote the negative items. Figure 4.30 illustrates an example of transforming an original set of transactions into transactions having negative items. By performing such transformation, we can apply existing algorithms such as Apriori to the new data set.

Such an approach would work well if the data set contains only a few symmetric binary variables. However, if all the items must be treated as symmetric binary, the problem becomes computationally intractable due to the following reasons.

1. *The number of items would doubled when negative items are augmented to the data set.* Instead of exploring an itemset lattice of size 2^d , where d is the number of items in the original data set, the lattice has grown to 2^{2d} . In addition, if most of the positive items have support less than 50%, then most of the negative items would have support higher than their positive counterparts. This will increase the number of negative patterns substantially.
2. *The width of each transaction increases to the total number of (positive) items in the data set.* Suppose there are d items available in the original data set. For sparse data sets such as market-basket transactions, the width of each transaction tends to be much smaller than d . As a result, the maximum size of a frequent itemset, which is bounded by the maximum transaction width, w_{\max} , tends to be relatively small. When negative items are included, the width of the transactions increases to d because an item is either present in the transaction or absent from the transaction, but not both. Since the maximum transaction width has grown from w_{\max} to d , this will increase the number of frequent itemsets exponentially. As a result, many existing algorithms tend to break down when applied to the extended data set.

The previous approach is computationally expensive because we need to determine the support of a large number of positive and negative patterns. An alternative approach is to compute only the support of the positive items. From these support counts, we can determine the support of any mixed combinations of positive and negative items. For example, the support for $\{x, \bar{y}, \bar{z}\}$ can be computed as follows.

$$s(\{x, \bar{y}, \bar{z}\}) = s(\{x\}) - s(\{x, y\}) - s(\{x, z\}) + s(\{x, y, z\})$$

The general formula for computing the support of any mixed itemset $A \cup \bar{B}$ is given by:

$$s(A \cup \bar{B}) = s(A) + \sum_{i=1}^n \sum_{C \subset B, |C|=i} \{(-1)^i \times s(A \cup C)\} \quad (4.11)$$

In order to use Equation 4.11, we need to know the support for A together with all the subsets of B . Thus, the minimum support threshold must be low enough to allow the support for $A \cup B$ to be computed. Since A and B may involve any sets of items, one could end up enumerating the support of the entire positive itemset lattice, which itself is an exponentially expensive task. Another way would be to simply ignore the support of B and subsets of B if they fall below the *minsup* threshold when using Equation 4.11.

There are several alternative suggestions to improve the performance of algorithms for mining negative patterns. The first approach is to restrict the number of negative items considered as interesting. For example, a negative item \bar{x} is considered to be uninteresting unless x itself is a frequent item. This strategy reduces the number of negative items to be augmented into the data set. A second approach is to restrict the type of negative patterns extracted. For example, all negative patterns that contain no positive items can be eliminated because they are not that interesting. However, one may not be able to prune them right away during pattern generation because one may lose the anti-monotone property of support. A third approach is to incorporate correlation-like measures such as χ^2 , interest factor (I) or the ϕ -coefficient directly into the mining process, but unfortunately, almost all such measures do not possess an anti-monotone property like support.

**** may not be needed ****

For example, although the χ^2 measure has been shown to be monotone (or upward closed), it is not useful for pruning purposes in a bottom-up level-wise algorithm. In addition, the statistical χ^2 test itself is non-monotone, i.e., if an itemset $\{A, B\}$ passes the statistical χ^2 test, there is no guarantee that its superset will pass the same test, even though the $\chi^2(\{A, B, C\}) \geq \chi^2(\{A, B\})$. This is because the χ^2 test depends on the χ^2 value as well as the total number of degrees of freedom. For an itemset of size k , the number of degrees of freedom is $2^k - k - 1$. A larger itemset will have a higher degrees of freedom, which in turn, increases the threshold needed for passing the χ^2 test. Thus, although the χ^2 value for $\{A, B, C\}$ is larger, this does not mean that it will pass the χ^2 test.

In short, mining negative patterns is a challenging problem especially for sparse transaction data. The existing techniques for mining negative patterns are either too expensive, produce too many uninteresting patterns, or uses approximate techniques to reduce its computational complexity.

Application of Concept Hierarchy

Using objective measures alone to eliminate uninteresting infrequent patterns may not be sufficient. For example, suppose **bread** and **laptop computer** are frequent items. Although **{bread, laptop computer}** is infrequent and perhaps negatively correlated, they are not interesting because their negative association is not surprising at all. This example suggests that a subjective way for identifying interesting negative patterns is needed.

In the previous example, **bread** and **laptop computer** belongs to two totally different product groups, which is why it is not surprising to find their support to be low. It would be advantageous to use such domain information to further eliminate uninteresting infrequent patterns. For market-basket data, the domain knowledge can be inferred from an item taxonomy, such as the one shown in Figure 11. The basic assumption is that items from the same product family are expected to have similar types of interaction with other items. For example, since Coke and Pepsi belong to the same product category, we expect the association between Coke and chips to be somewhat similar to the association between Pepsi and chips. If the actual support for any one of this two pairs is less than their expected support, then the infrequent pattern is interesting.

To illustrate how to compute the expected support, consider the diagram shown in Figure 4.31. Suppose the itemset $\{C, G\}$ is frequent. We can compute the expected support for any children or siblings of C and G using the formula shown below.

$$E(s(E, J)) = s(C, G) \times \frac{s(E)}{s(C)} \times \frac{s(J)}{s(G)} \quad (4.12)$$

$$E(s(C, J)) = s(C, G) \times \frac{s(J)}{s(G)} \quad (4.13)$$

$$E(s(C, H)) = s(C, G) \times \frac{s(H)}{s(G)} \quad (4.14)$$

For example, if **soft drink** and **chips** are frequent, then we can compute the expected support between **Pepsi** and **Ruffles** using equation 4.12 since they are children of **soft drink** and **chips**. If the actual support for **Pepsi** and **Ruffles** is considerably less than their expected value, then **Pepsi** and **Ruffles** form an interesting negative pattern.

Indirect Associations

*** This section is being modified ****

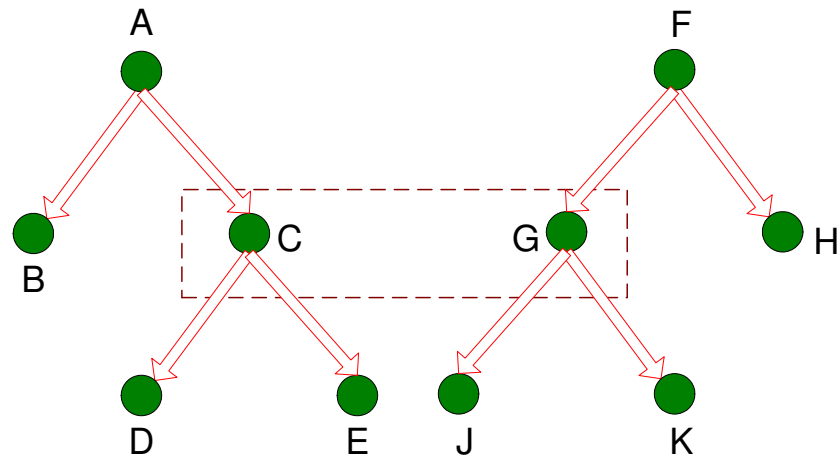


Figure 4.31. Mining interesting negative patterns using a concept hierarchy.

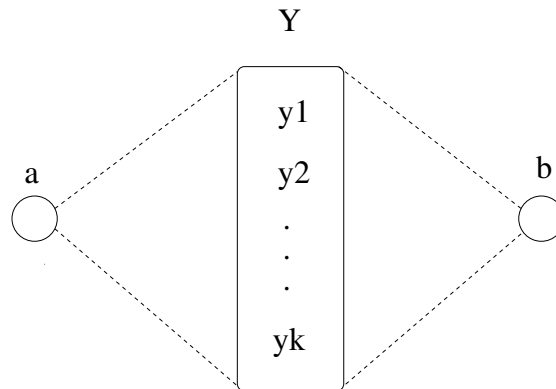


Figure 4.32. Indirect association between a pair of items.

Association rule mining would discover rules involving items that occur frequently together in the transaction database. Any itemsets that fail the support threshold criteria will be automatically removed. This action is justifiable since we are only interested in finding items that are directly associated with each other. However, under certain situations, it is possible that some of the lowly-supported itemsets can be interesting due to the presence of higher-order dependencies¹ among the items. Consider a pair of items, (a, b) , having a low joint support value (Figure 4.32). Let Y be a set that contains items that are highly dependent on both a and b . This figure illustrates an indirect association between a and b via a mediator set, Y . Unlike direct association, an indirect relationship is characterized not only by the two participants, a and b , but also by the items mediating this interaction.

One can think of many potential applications for indirect association. In the market basket scenario, a and b may represent competing products. Analysts may

¹direct association between items are considered to be first-order dependencies.

be interested in finding what are the common products (Y) bought together by customers who purchased the two competing products. With this information, they can then performed an in-depth analysis on why customers prefer one brand over the other based on the strength of dependencies between Y with a and b . Another potential application for indirect association is in textual data mining. For text documents, indirectly associated words may represent synonyms and antonyms or terms that appear in different contexts of another word. For example, in a collection of news articles, the words growth and decline may be indirectly associated via the word economy. Another example is the pair soviet and worker may refer to the different contexts in which the word union is being used. For stock market data, a and b may refer to disjoint events that will partition event Y . For example, the event $Y = Microsoft - Up$ can be partitioned into $a = Redhat - Down$ and $b = Intel - Up$ where a denotes the event where the competitor's stock value is down whereas b may represent the event in which most of the technology stocks for large corporations are up.

[180] proposed a general framework for defining indirect association in various application domains. Their formal definition of indirect association is :

Definition 12 *The pair a, b is indirectly associated via a mediator Y if the following conditions hold :*

1. $\sigma(a, b) < t_s$ (*Itempair Support condition*).
2. $\exists Y \neq \emptyset$ such that $\forall y_i \in Y$:
 - (a) $\sup(a, y_i) \geq t_f, \sup(b, y_i) \geq t_f$ (*Mediator Support condition*).
 - (b) $d(a, y_i) \geq t_d, d(b, y_i) \geq t_d$ where $d(p, q)$ is a measure of the dependence between p and q (*Dependence condition*).

In order to discover indirectly associated items, one must address the following questions :

1. What is a good measure of dependence between items?
2. What constitutes the elements of the mediator?
3. How to define the overall indirect association measure between a and b ?

The first question is related to the issue of finding good interestingness measures for itemsets (or association rules) which was described in Section 4.7. The second question deals with the problem of defining mediating elements either as single items, single itemsets or multiple itemsets. The last question allows one to rank the discovered itempairs according to the strength of their indirect association. [180] described several ways to tackle these issues and presented a simple algorithm for mining indirect itempairs.

4.8.3 Frequent Subgraphs

(This section is being modified)

As the field of association rule mining continues to mature, there is an increasing need to apply similar techniques to other disciplines, such as Web Mining, bioinformatics, and computational chemistry. The data for these domains is often available in a graph-based format, as described in Chapter 2. For example, in the Web Mining domain, the vertices would correspond to Web pages and the edges would correspond to hyperlinks traversed by the Web users. In the computational chemistry domain, the vertices would correspond to the atoms or ions while the edges would correspond to their chemical bonds. Mining associations from these kinds of data is more challenging due to the following reasons:

1. Support and confidence thresholds are not the only requirements the patterns must satisfy. The structure of the data may restrict the type of patterns that can be generated by the mining algorithm. Specifically, additional constraints can be imposed regarding the connectivity of the subgraph, i.e., the frequent subgraphs must be connected graphs.
2. In the standard market-basket analysis, each entity (item or discretized attribute) is treated as a binary variable. (Although the quantity of an item bought in a transaction can be larger than one, it is often recorded as 1 in the record-based format.) For graph-based data, an entity (vertex, event, or item) can appear more than once within the same data object. These entities cannot be combined into a single entity without losing the structure information. The multiplicity of these entities poses additional challenges to the mining problem.

As a result, the standard Apriori or FP-tree algorithms must be modified in order to mine frequent substructures efficiently from a collection of tree or graph objects. In this section, we would briefly describe what are the modifications needed and how they can be implemented effectively.

Problem Formulation

Let $V = \{v_1, v_2, \dots, v_k\}$ be the set of vertices and $E = \{e_{ij} = (v_i, v_j) | \forall v_i, v_j \in V\}$ be the set of edges connecting pairs of vertices. Each vertex v is associated with a vertex label denoted as $l(v)$. We will use the notation $L(V) = \{l(v) | \forall v \in V\}$ to represent the set of all vertex labels in the data set. Similarly, each edge e is associated with an edge label denoted as $l(e)$. We will use the notation $L(E) = \{l(e) | \forall e \in E\}$ to represent the set of all edge labels in the data set.

A *labeled graph* G is defined as a collection of vertices and edges with labels attached to each of them, i.e., $G = (V, E, L(V), L(E))$. A graph $G' = (V', E', L(V'), L(E'))$

is a *subgraph* of G if its vertices V' and edges E' are subsets of V and E , respectively. G' is called an *induced subgraph* of G if it satisfies the subgraph requirements along with the additional condition that all the edges connecting the vertices V' in G are also present in E' . Figure 4.33(a) illustrates an example of a labeled graph G that contains 6 vertices and 11 edges, with vertex labels obtained from the set $L(V) = \{a, b, c\}$ and edge labels obtained from the set $L(E) = \{p, q, r, s, t\}$. A subgraph of G that contains only 4 vertices and 4 edges is shown in Figure 4.33(b). It is not an induced subgraph because it does not include all the edges in G involving the subgraph vertices. Instead, the proper induced subgraph of G for these vertices is shown in Figure 4.33(c).

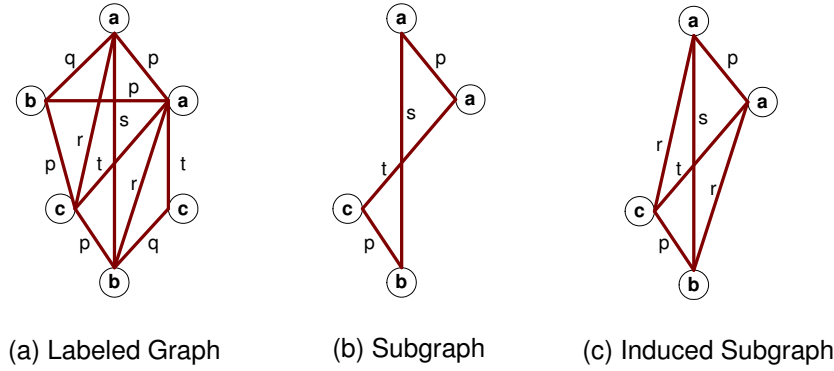


Figure 4.33. Graph, subgraph, and induced subgraph definitions.

There are many data sets that can be modelled by using the graph representation. For instance, a market-basket transaction can be represented as a graph, with vertices that correspond to each item and edges that correspond to the relationships between items. Furthermore, each transaction is essentially a clique because every item is associated with every other item in the same transaction. The problem of mining frequent itemsets is equivalent to finding frequent subgraphs among the collection of cliques. Although the graph representation is more expressive, it is much easier to work with the standard market-basket representation because we do not have to worry about the structural constraints.

If the graphs are not cliques, we can still map the problem of finding frequent subgraphs into the standard market-basket representation as long as all the edge and vertex labels within a graph are unique. In this case, each graph can be regarded as a transaction and each triplet of vertex and edge labels $t = (l(v_i), l(v_j), l(e_{ij}))$ corresponds to an item, as illustrated in Figure 4.34. Unfortunately, most of the real applications do not contain graphs with unique labels, thus requiring more advanced methods to obtain the frequent subgraphs.

Given a set of graphs \mathcal{G} , the support of a subgraph g is defined as:

$$s(g) = \frac{|\{G_i | g \subseteq_s G_i \wedge G_i \in \mathcal{G}\}|}{|\mathcal{G}|} \quad (4.15)$$

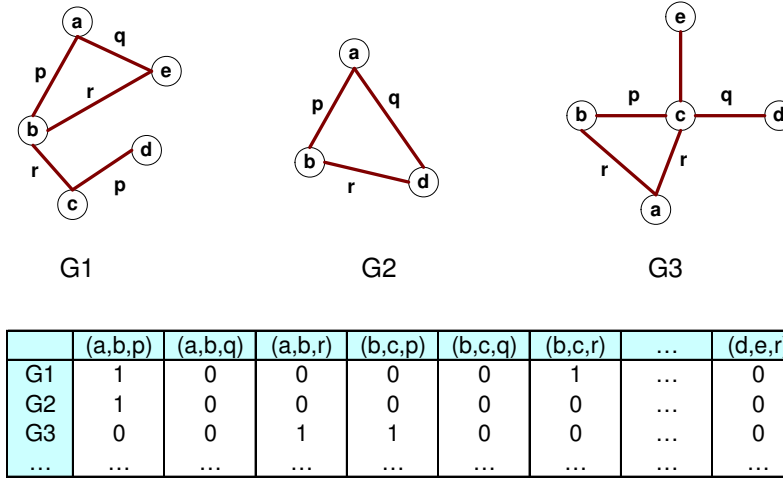


Figure 4.34. Mapping a collection of graph structures into market-basket transactions.

where \subseteq_s denotes the subgraph relation. This definition of the support measure also satisfies the anti-monotonicity property. The goal of *frequent subgraph mining* is to find all subgraphs whose support is greater than some minimum support threshold, *minsup*. In some cases, we could also be interested in finding association rules for the frequent subgraphs. The confidence of a graph association rule $g_1 \longrightarrow g_2$ can be defined as:

$$c(g_1 \longrightarrow g_2) = \frac{|\{G | g_1 \cup g_2 \subseteq_s G \wedge G \in \mathcal{G}\}|}{|\{G' | g_1 \subseteq_s G' \wedge G' \in \mathcal{G}\}|} \quad (4.16)$$

where $g_1 \cup g_2$ denotes any graph containing both g_1 and g_2 as its subgraphs.

Mining Frequent Subgraphs

The idea of using frequent pattern discovery algorithms such as Apriori for mining frequent subgraphs was initially proposed by Inokuchi et al. in [92]. There are several issues one has to consider when mining frequent subgraphs:

Subgraph growing: Most of the standard frequent itemset generation algorithms work in a bottom-up fashion, where the frequent itemsets of size k are used to enumerate the frequent itemsets of size $k + 1$. During frequent itemset generation step, the itemset lattice (search space) can be traversed in a breadth-first or depth-first manner. The same approach can also be applied to the frequent subgraph generation problem. However, unlike frequent itemset mining, k may refer to the number of vertices or edges in the subgraphs. If the algorithm enumerates frequent subgraphs one-vertex-at-a-time, the approach is called *vertex growing*. On the other hand, if the algorithm enumerates the frequent subgraphs one-edge-at-a-time, the approach is called *edge growing*:

1. Vertex growing has a meaningful interpretation because each iteration is equivalent to increasing the dimension of its corresponding $k \times k$ adjacency matrix by a factor of one, as illustrated in Figure 4.35. G_1 and

G_2 are two graphs whose adjacency matrices are given by $M(G_1)$ and $M(G_2)$, respectively. This approach was used by the AGM algorithm

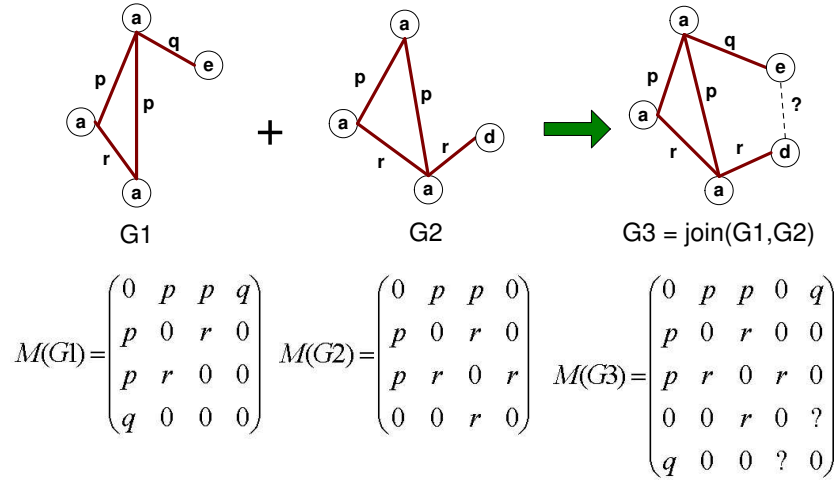


Figure 4.35. Vertex growing approach.

proposed by Inokuchi et al. in [92]. As the subgraphs are grown one vertex at a time, the algorithm would find only the frequent induced subgraphs. The drawback of this approach is that it can be computationally expensive due to the large number of candidate subgraphs enumerated by the algorithm. For example, in Figure 4.35, when the graphs G_1 and G_2 are joined together, the resulting graph G_3 will have a new edge connecting between the vertices d and e . However, the label of the new edge can be arbitrary, which increases the number of candidate subgraphs dramatically.

2. Edge growing was initially proposed by Kuramochi et al [109] to enable a more efficient candidate generation strategy. At each iteration, a new edge is added to the subgraph, which may or may not increase the number of vertices in the initial subgraph, as illustrated in Figure 4.36. However, the number of candidate subgraphs generated using this approach is much smaller than the number of candidate subgraphs generated using the vertex-growing approach. Kuramochi et al. [109] have also used the more efficient sparse graph representation instead of the adjacency matrix representation. In addition, edge-growing approach tend to produce more general patterns because it can discover both frequent subgraphs and frequent induced subgraphs.

Graph isomorphism: A key task in frequent itemset discovery is to determine whether an itemset is a subset of a transaction. Likewise, in frequent subgraph mining, an important task is to determine whether a graph is a subgraph of another graph. This requires a graph matching routine that can determine

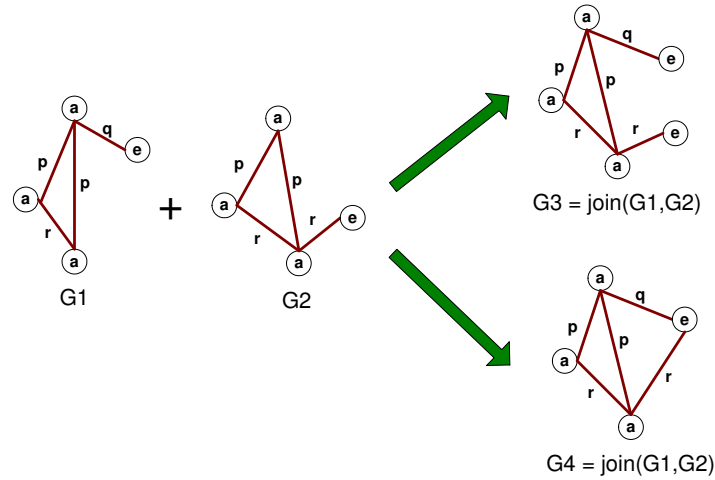


Figure 4.36. Edge growing approach.

whether a subgraph is topologically identical (*isomorphic*) to another graph. For example, Figure 4.37 illustrates two graphs that are isomorphic to each other because there is a one-to-one mapping between each vertex in the first graph to another vertex in the second graph that preserves the connections between the vertices.

Graph isomorphism is a complicated problem especially when many of the candidate subgraphs are topologically equivalent. If two graphs are isomorphic to each other, their adjacency matrices are simply row and/or column permutations of the other. If the labels on the vertices are unique, the graph isomorphism problem can be solved in a polynomial time. However, if the labels of the edges and vertices are not unique, the complexity of the graph isomorphism problem is yet to be determined (i.e., it is still an open problem whether the graph isomorphism problem is NP-hard). In Apriori-like algorithms such as AGM [92] and FSM [109], graph isomorphism presents the biggest challenge to frequent subgraph discovery because it is needed for both candidate generation and candidate counting steps:

1. It is needed during the join operation to check whether a candidate subgraph has already been generated.
2. It is needed during candidate pruning to check whether its $k - 1$ subgraph matches one of the previously discovered frequent subgraph.
3. It is needed during candidate counting to check whether a candidate subgraph is contained within another graph.

Most of the frequent subgraph discovery algorithms would handle the graph isomorphism problem by using a technique known as *canonical labeling*, where each graph or subgraph is mapped into an ordered string representation (otherwise known as its *code*), in such a way that two isomorphic graphs would

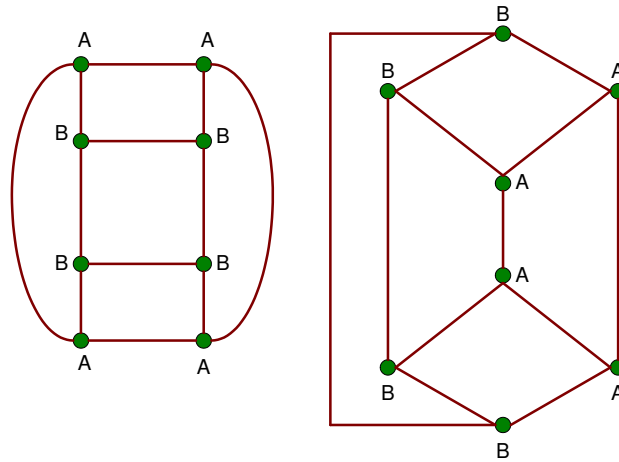


Figure 4.37. Graph Isomorphism

be mapped into the same canonical encoding. If a graph can have more than one representation, its canonical encoding is usually chosen to be the lowest precedence code. For example, Figure 4.38 illustrates two different ways to represent the same graph. (The graph has more than one representation because the vertices are not unique.) The code for each graph is obtained by doing a column-wise concatenation of the elements in the upper triangular portion of the adjacency matrix. The canonical labeling of the graph is obtained by selecting the code that has the lowest precedence, e.g., the code at the bottom of Figure 4.38 has a lower precedence than the code at the top of the figure.

Multiplicity of Candidate Joining: In Apriori-like algorithms, candidate patterns are generated by joining together frequent patterns found in the earlier passes of the algorithms. For example, in frequent itemset discovery, a candidate k -itemset is generated by joining two frequent itemsets of size $k - 1$ that contains $k - 2$ items in common. Similarly, in frequent subgraph mining, a candidate k -subgraph is generated by joining together two frequent subgraphs of size $k - 1$ (where k may correspond to the number of vertices or edges of the graph). New candidates are created as long as both $k - 1$ subgraphs contain a common $k - 2$ subgraph, known as its *core*. However, unlike frequent itemset discovery, each join operation in frequent subgraph mining may produce more than one candidate subgraph, as illustrated in Figures 4.35 and 4.36. For the edge-growing approach, Kuramochi et al [109] three different scenarios in which multiple candidates can be generated:

1. When joining two subgraphs with identical vertex labels, as illustrated in Figure 4.39(a). Multiple candidates are generated because there are more than one way to arrange the edges and vertices of the combined subgraph.

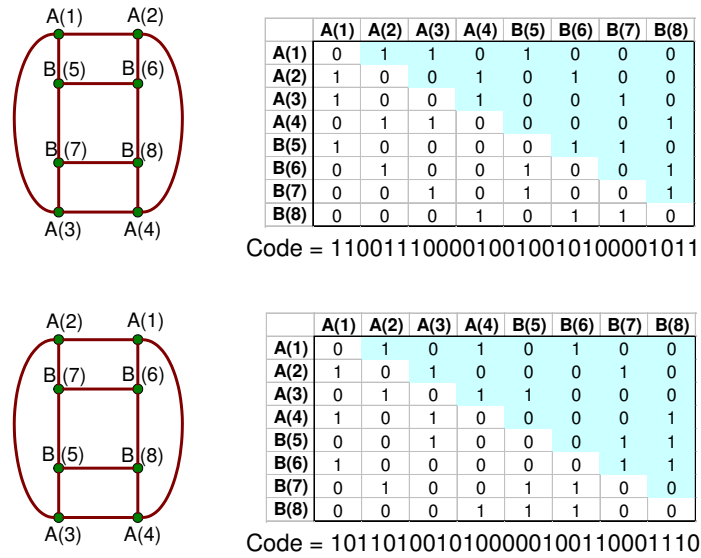


Figure 4.38. Adjacency matrix and string encoding of a graph.

2. When joining two subgraphs whose core contains identical vertex labels, as illustrated in Figure 4.39(b). In this example, the core contains vertices whose labels are equal to “a”.
3. When the joined subgraphs contain more than one core, as illustrated in Figure 4.39(c). The shaded vertices in this figure correspond to vertices that form the core of the join operation.

As a result, the number of candidate subgraphs generated can be prohibitively large. Numerous algorithms have been proposed to reduce the number of candidate subgraphs generated by the frequent subgraph mining algorithms. For example, the FSG algorithm [109] uses various *vertex invariants*, which are structural constraints imposed on the candidate subgraphs, to restrict the number of admissible candidates. A more recent algorithm, called g-span [199], was proposed by Yan et al to generate frequent subgraphs without enumerating any infrequent candidates. The algorithm uses a canonical labeling scheme called minimum DFS code to represent the subgraphs. By encoding the graphs according to their minimum DFS codes, the authors showed that the frequent subgraph mining problem can be turned into a sequential pattern mining problem, for which an efficient algorithm can be developed that does not require candidate generation [145].

4.8.4 Constraint Association Rule Mining

(This section is being modified)

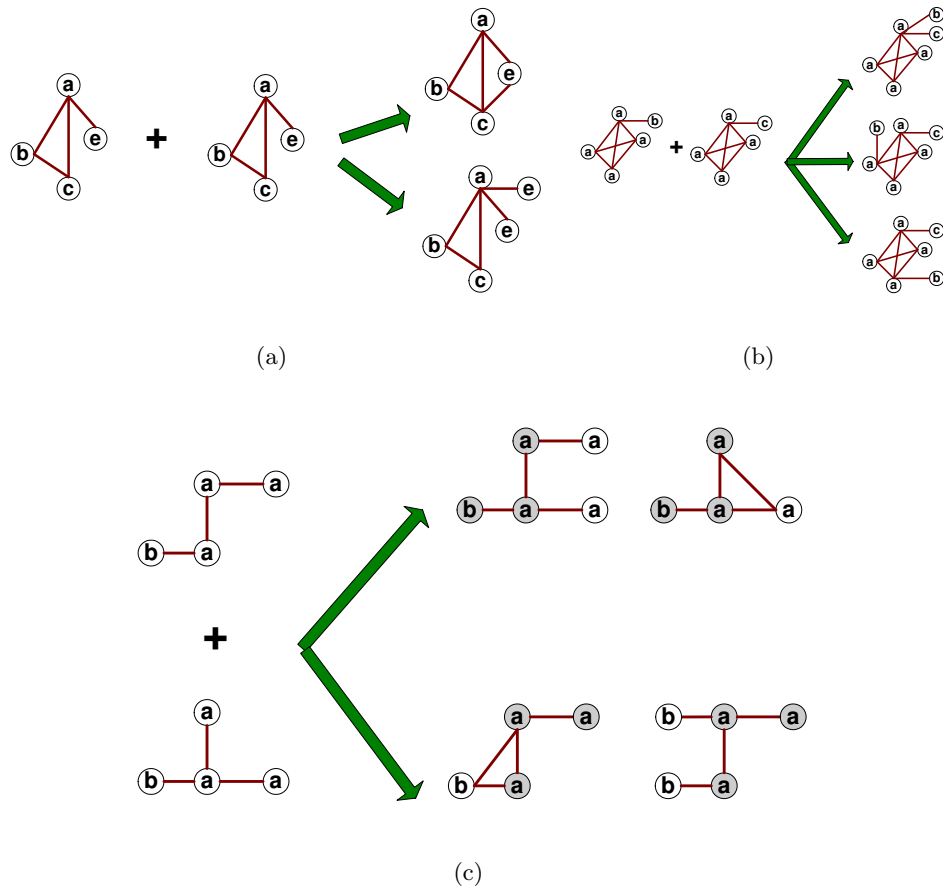


Figure 4.39. Multiplicity of Candidate Joining.

Srikant *et al.* [174] has considered the problem of discovering association rules in the presence of constraints, represented as boolean expressions. An example of a boolean constraint is $(\text{Cookies} \wedge \text{Milk}) \vee (\text{descendents}(\text{Cookies}) \wedge \neg \text{ancestors}(\text{Wheat Bread}))$, which looks for rules that contain both cookies and milk, or rules containing the descendent items of cookies but not ancestor items of wheat bread. The Apriori pruning strategy is no longer applicable because some frequent itemsets are not generated since they violate the item constraints. Constraints on the support of itemsets have been studied in [193]. The idea of support constraints is to specify the minimum support for different itemsets, so that only the necessary itemsets are generated. This is different from the approach taken in [116], which assigns a minimum support to each item, rather than to each itemset. A more recent work by Seno *et al.* [160] have attempted to specify the minimum support of an itemset dynamically, based on the length of the itemset. An algorithm for mining constrained association rules for text documents has also been proposed in [167]. Here, the constraints are specified by the concepts or structured values provided by the user. A similar approach was taken by Ng *et al.* in [137] in which the user is allowed to input the constraints via constrained association queries.

4.8.5 Sequential Patterns

(This section will be added)

4.8.6 Spatial Associations

(This section will be added)

4.9 Bibliographic Notes

The concept of association rule mining was first introduced by Agrawal *et al.* in [7,6] to discover interesting relationships among items in a transaction database. Since its inception, there has been extensive research: (1) to improve the theoretical understanding of the association rule mining task, (2) to handle other data types, (3) to capture new types of patterns, (4) to address the algorithmic, database and other implementation issues, and (5) to extend the analysis beyond market basket data to other application domains. A summary of these research activities is shown in Figure 4.40.

Conceptual issues. Following the pioneering work by Agrawal *et al.*, there has been a vast amount of research on formulating a theoretical understanding of the association analysis problem. In [68], Gunopoulos *et al.* relates the

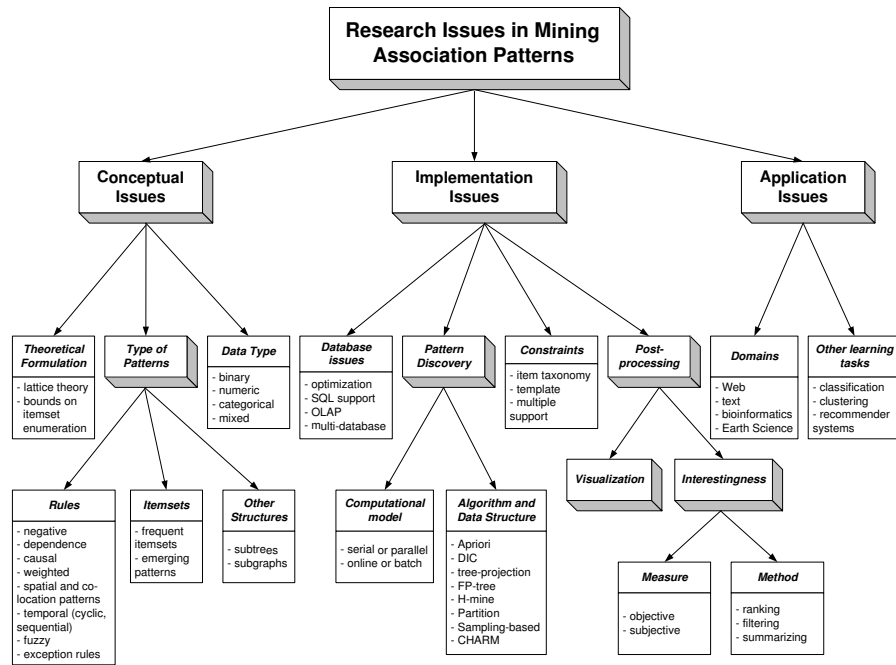


Figure 4.40. A summary of the research issues in mining association patterns.

problem of finding frequent itemsets to the hypergraph transversal problem. Zaki et al. [204,202] and Pasquier et al. [143] have studied the frequent itemset generation problem using formal concept analysis, work which has led them to introduce the concept of closed itemsets [204], which is related to bipartite cliques in graph theory. Friedman *et al.* [61] have formulated association analysis from the perspective of a statistical learning problem called bump hunting. In this formulation, frequent itemset generation is analogous to finding regions in the multi-dimensional space for which the probability density of the region is significantly higher than the expected probability. There has also been substantial research to extend the classical formulation to non-binary data types such as categorical [172], numeric [172,63,69,207,194], and interval [127] variables. New patterns such as profile association rules [3], cyclic association rules [140], fuzzy association rules [108], exception rules [178], negative associations [159,24], weighted association rules [150,29], dependence rules [166], sequential associations [8,173], peculiar rules [208], inter-transaction association rules [189,57], predictive association rules [122], spatial co-location patterns [163], partial classification rules [132,10], and emerging patterns [47] have also been developed.

The concept of closed itemsets was proposed independently by Zaki et al. [204] and Pasquier et al [143].

Implementation issues. Much of the research activities in this area revolve around (1) developing efficient, serial and parallel mining algorithms, (2) integrat-

ing the mining process into existing database technology, (3) handling user-specified or domain-specific constraints, and (4) post-processing the discovered patterns.

Over the years, many algorithms have been proposed to efficiently mine association patterns from large databases. Park *et al.* [142] proposed the DHP (dynamic hashing and pruning) technique that will reduce the size of the database after each pass of the itemset generation algorithm. The Partition algorithm [158] requires at most two database scans in order to generate frequent itemsets. A sampling-based approach for frequent itemset generation was proposed by Toivonen [184]. This approach requires only a single pass, but it tends to produce more candidate itemsets than necessary. The Dynamic Itemset Counting (DIC) algorithm [25] requires only 1.5 passes over the data, and generates less candidate itemsets than the sampling-based algorithm. Other notable frequent itemset generation algorithms include the tree-projection algorithm [2], FP-tree [76], CHARM [203], and H-Mine [144]. Some of the more recent surveys on frequent itemset generation algorithms are given by [4, 84].

All the algorithms described so far in this chapter are batch, serial algorithms. Parallel formulation of association rule mining has been investigated by many authors [1, 70, 133, 164, 205]. A survey of these algorithms is given by Zaki in [201]. Online and incremental versions of the association rule mining algorithm have been developed by Hidber [81] and Cheung *et al.* [33].

There are several advantages of integrating the association analysis task into existing database technologies. First, one can make use of the indexing and query processing capabilities of the database. Second, one can also exploit the DBMS support for scalability, check-pointing and parallelization [156]. The SETM algorithm developed by Houtsma *et al.* [88] was one of the earliest algorithm to support association rule discovery via standard SQL queries. Since then, various ideas have been proposed to extend the capability of standard SQL to support association rule mining. For example, query languages such as DMQL [73] and M-SQL [90] extend the basic SQL with new operators for mining association rules. The Mine Rule operator suggested by Meo *et al.* in [124] is an expressive SQL operator that can handle both clustered attributes and item hierarchies. A generate-and-test approach called query flocks has also been proposed in [186] while Chen *et al.* [31] have developed a distributed OLAP-based infrastructure for generating multi-level association rules.

Srikant *et al.* [174] has considered the problem of discovering association rules in the presence of constraints, represented as boolean expressions. An example of a boolean constraint is $(\text{Cookies} \wedge \text{Milk}) \vee (\text{descendents}(\text{Cookies}) \wedge \neg \text{ancestors}(\text{Wheat Bread}))$, which looks for rules that contain both cookies and milk, or rules containing the descendent items of cookies but not ancestor items of wheat bread. Constraints on the support of itemsets have been studied by Wang *et al.* in [193]. The idea of support constraints is to specify the

minimum support threshold for different itemsets, so that only the necessary frequent itemsets are generated. This is different from the approach taken by Liu et al. in [116] that assigns a minimum support threshold to each item. A more recent work by Seno *et al.* [160] have attempted to specify the minimum support threshold based on the length of the itemset. An algorithm for mining constrained association rules for text documents has also been proposed by Singh et al. in [167]. Here, the constraints are specified by the concepts or structured values provided by the user. A similar approach was taken by Ng et al. in [137] in which the user is allowed to input the constraints via constrained association queries.

One major problem with association rule generation is the large number of rules that are being generated. This problem can be handled by pruning or aggregating the related rules. Toivonen et al. [185] proposed the idea using structural rule covers to remove redundant rules and clustering the remaining rules to group together related rule covers. Liu et al. [117] used the standard χ^2 test for pruning insignificant rules and introduced the concept of direction setting rules to summarize the remaining patterns. Other researchers such as Srikant et al. [174] and Ng et al. [137] rely on the constraints provided by a user to limit the number of generated rules. Visualization also helps the user to quickly grasp the underlying structure of the discovered patterns. Many commercial data mining tools would display the complete set of rules (that satisfy both support and confidence threshold criteria) as a 2-dimensional plot, with each axis corresponds to the antecedent or consequent itemsets of the rule. Hofmann et al. [85] proposed using Mosaic plots and Double Decker plots for visualizing association rules. This approach can visualize not only a particular rule but the overall contingency table between itemsets in the antecedent and consequent parts of the rule. However, this technique assumes that the rule consequent consists of only of a single attribute.

The notion of subjective interestingness measures have been investigated by many authors. Silberschatz and Tuzhilin [165] presented two principles in which a rule can be considered as interesting from a subjective point of view. The concept of unexpected condition rules was introduced by Liu et al. in [115]. Cooley et al. [38] 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.

Application issues. Association rule mining has been applied to a variety of application domains such as Web mining [179, 146], text document analysis [86], telecommunication alarm diagnosis [102], network intrusion detection [111, 16, 44], and genomics [157]. In some of these applications, domain-specific knowledge can be used to improve upon the existing preprocessing, mining and post-processing tasks. Association patterns have also been applied to other learning problems such as classification [19, 113] and clustering [200, 71]. A comparison between classification and association rule mining was made by

Freitas in his position paper [60]. The use of association patterns for clustering has been studied by many authors including Han et al. [71], Kusters et al. [105] and Yang et al. [200].

4.10 Exercises

1. Support and Confidence.

For each of the questions listed below, provide an example of an association rule from the market basket domain that satisfies the given conditions. Also, describe whether such rules are interesting. State your reasons clearly.

- 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.

2. Market Basket Definition.

Consider the data set shown in Table 4.19 below.

Table 4.19. 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 $BD \rightarrow E$ and $E \rightarrow BD$. 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 $BD \rightarrow E$ and $E \rightarrow BD$.

- (e) 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 . (Hint: compare the rules $BD \rightarrow E$ and $A \rightarrow E$ in both cases.)

3. Properties of Confidence.

- (a) Let c_1 , c_2 , and c_3 be the confidence values of the rules $A \rightarrow B$, $A \rightarrow BC$, and $AC \rightarrow B$, 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?
- (b) Repeat the previous analysis assuming that all three rules given in the previous question have identical support? Which rule has the highest confidence?
- (c) 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 ? Justify your answer.

4. Monotonicity Property

For each question listed below, 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) A characteristic rule is a rule of the form $A \rightarrow B_1 B_2 \cdots B_n$, where the antecedent of the rule consists of 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 an itemset X :

$$\zeta(\{A_1, A_2, \dots, A_k\}) = \min(\quad c(A_1 \rightarrow A_2, A_3, \dots, A_k), \dots \\ c(A_k \rightarrow A_1, A_3 \cdots, A_{k-1}))$$

Is ζ monotone, anti-monotone or non-monotone? (Hint: compare the minimum confidence value of all characteristic rules generated from the itemset $\{A, B\}$ against the minimum confidence value of all characteristic rules generated from the itemset $\{A, B, C\}$.)

- (b) A discriminant rule is a rule of the form $B_1 B_2 \cdots B_n \rightarrow A$, where the consequent of the rule consists of 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 an itemset X :

$$\eta(\{A_1, A_2, \dots, A_k\}) = \min(\quad c(A_2, A_3, \dots, A_k \longrightarrow A_1), \dots \\ c(A_1, A_2, \dots, A_{k-1} \longrightarrow A_k))$$

Is η monotone, anti-monotone or non-monotone? (Hint: compare the minimum confidence value of all discriminant rules generated from the itemset $\{A, B\}$ against the minimum confidence value of all discriminant rules generated from the itemset $\{A, B, C\}$.)

- (c) Repeat the analysis in parts (a) and (b) by replacing the min function with a max function.

5. **Complexity of Association Rule Generation** Proof Equation ???. (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.)

6. Apriori Algorithm

Consider the market basket transactions shown in Table 4.20. Use this data set to answer the questions listed below.

Table 4.20. Market basket transactions.

Transaction ID	Items Bought
1	<i>{Milk, Beer, Diaper}</i>
2	<i>{Bread, Butter, Milk}</i>
3	<i>{Milk, Diaper, Cookies}</i>
4	<i>{Bread, Butter, Cookies}</i>
5	<i>{Beer, Cookies, Diaper}</i>
6	<i>{Milk, Diaper, Bread, Butter}</i>
7	<i>{Bread, Butter, Diaper}</i>
8	<i>{Beer, Diaper}</i>
9	<i>{Milk, Diaper, Bread, Butter}</i>
10	<i>{Beer, Cookies}</i>

- (a) How many possible association rules can be extracted from this data (including rules that have zero support)?
- (b) Given the transactions shown above, what is the largest size of an itemset we can extract?
- (c) Write an expression for the maximum number of size-3 itemsets that can be derived from this data set?
- (d) Which itemset (of size 2 or larger) has the largest support?

- (e) From this data set, find a pair of association rules, $A \longrightarrow B$ and $B \longrightarrow A$, that have the same confidence.

7. Apriori Algorithm

Consider the data set shown in Table 4.20.

- (a) Derive all frequent itemsets having support $\geq 30\%$
 (b) From the frequent itemsets discovered in the previous question, derive all association rules having confidence $\geq 80\%$.

8. Itemset Lattice

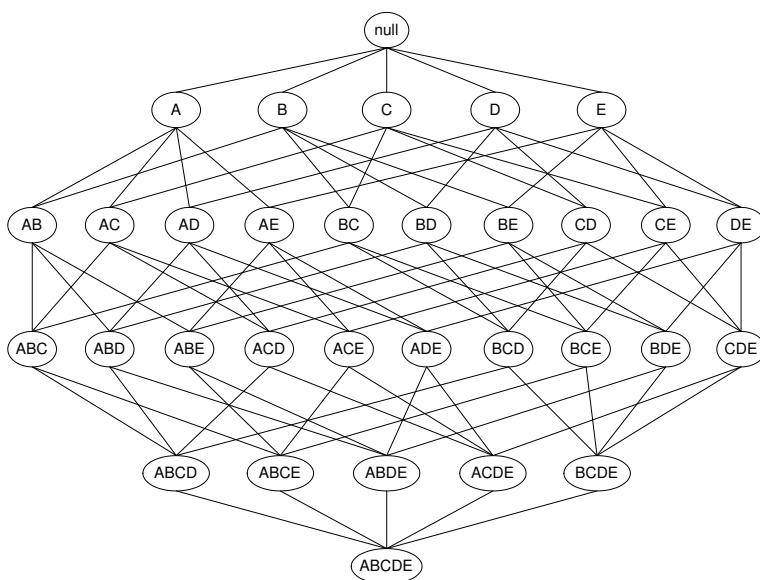
The Apriori algorithm uses the generate-and-count strategy for generating frequent itemsets. Candidate itemsets of size $k + 1$ are created by joining a pair of frequent itemsets of size k . For example, the candidate $\{P, Q, R\}$ is generated by merging the frequent itemsets $\{P, Q\}$ and $\{P, R\}$. A candidate is pruned if any one of its subsets is found to be infrequent. For example, the candidate $\{P, Q, R\}$ is pruned if $\{Q, R\}$ is infrequent.

Suppose you apply the Apriori algorithm to the data set given in Table 4.21. Let the minimum support threshold be equal to 30%, i.e., any itemset occurring less than 3 times is infrequent.

Table 4.21. 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	$\{B, D\}$

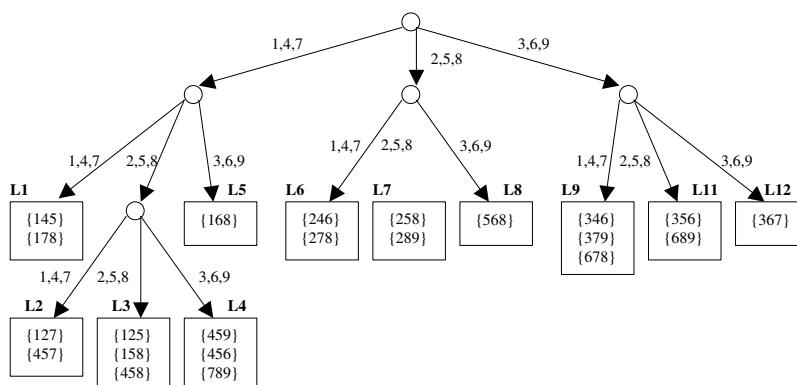
- (a) Draw a lattice structure representing all possible itemsets that can be generated from the data set given in Table 4.21. In your diagram, label each node as either:
- P : itemsets that are pruned because one of their subsets are infrequent,
 - F : candidate itemsets found to be frequent,
 - I : candidate itemsets found to be infrequent,
 - N : itemsets not generated as candidates by Apriori.



- What is the percentage of itemsets not generated by Apriori?
- What is the percentage of itemsets that are pruned because one of their subsets are infrequent?
- What is the pruning ratio of the Apriori algorithm on this data set? (Pruning ratio is the percentage of itemsets that are either (1) not generated as candidates or (2) pruned before their support is actually counted.)
- What is the percentage of itemsets that are frequent?
- What is the percentage of itemsets that are infrequent?

9. Hash Structure

The Apriori algorithm uses a hash-tree data structure to efficiently count the support of candidate itemsets. Consider the hash-tree for candidate 3-itemset shown in Figure 9.



- Suppose a new candidate itemset $\{5, 6, 7\}$ is added to the hash tree during

the hash tree construction. Which of the leaf node above (labeled as L1 through L12) will contain the new candidate?

- (b) 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?
- (c) 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. Hash Structure

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 even-numbered items are hashed to the right child.
- (b) How many leaf nodes are there in the candidate hash tree? How many internal nodes are there?
- (c) Suppose there is a transaction that contains the following items: $\{1, 2, 3, 5, 6\}$ and you would like to find all the candidate itemsets of size 3 contained in this transaction. Using the hash tree constructed in part (a), which leaf nodes will be checked against the transaction? What are the candidate itemsets of this transaction?

11. Item Taxonomy

Consider the transactions shown in Table 4.22. Suppose the items can be grouped together according to the item taxonomy shown in Figure 11.

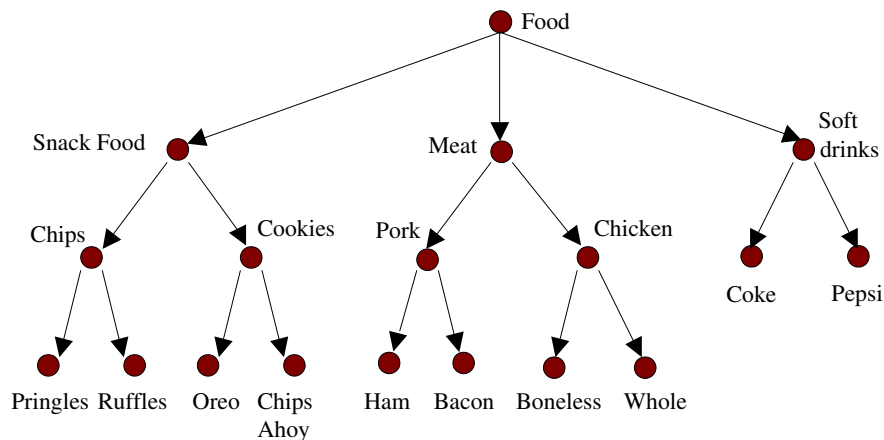


Table 4.22. Example of market-basket transactions.

Transaction ID	Items Bought
1	Pringles, Oreo, Coke, Ham
2	Ruffles, Pringles, Ham, Boneless Chicken, Pepsi
3	Ham, Bacon, Whole Chicken, Coke, Pepsi
4	Ruffles, Chips Ahoy, Ham, Boneless Chicken, Pepsi
5	Chips Ahoy, Bacon, Boneless Chicken
6	Ruffles, Ham, Bacon, Whole Chicken, Coke
7	Ruffles, Oreo, Boneless Chicken, Pepsi

- (a) Describe what are the main challenges of mining association rules with item taxonomy.
- (b) Consider the approach taken by Srikant and Agrawal [170], where each transaction t is replaced by an extended transaction t' that contains all the items in t as well as their respective ancestors. For example, the transaction $t = \{\textit{Pringles}, \textit{Oreo}\}$ will be replaced by $t' = \{\textit{Pringles}, \textit{Oreo}, \textit{Chips}, \textit{Cookies}, \textit{Snack Food}, \textit{Food}\}$. Use this approach to derive all frequent itemsets (up to size 4) with support $\geq 50\%$.
- (c) Repeat the analysis in part (b) using the approach taken by Han and Fu [72], where frequent itemsets at higher levels are used to find frequent itemsets at lower levels of the hierarchy.

12. Numeric Association Rules

- (a) Numeric attributes are often handled by discretizing the range of attribute values into disjoint intervals. Describe, in your own words, what are the main challenges of mining numeric association rules.
- (b) Explain what modifications would be needed in the candidate generation step of the Apriori algorithm in order to handle numeric attributes.
- (c) Consider the data set shown in Table 4.23. Suppose we apply the following discretization strategies to the numeric attributes:
 - D1: Partition the range of each numeric attribute into 3 equal-sized bins.
 - D2: Partition the range of each numeric attribute into 3 bins; where each bin contains equal number of data points
 For each strategy, answer the following questions:
 - i. Construct the transaction matrix having the new discretized attributes.
 - ii. Derive all the frequent itemsets having support ≥ 30
- (d) The numeric attribute can also be discretized using a clustering approach.
 - i. Draw a graph of Temperature versus Pressure for the data points given in Table 4.23.

Table 4.23. Example of numeric data set.

TID	Temperature	Pressure	Alarm 1	Alarm 2	Alarm 3
1	95	1105	0	0	1
2	85	1040	1	1	0
3	103	1090	1	1	1
4	97	1084	1	0	0
5	80	1038	0	1	1
6	100	1080	1	1	0
7	83	1025	1	0	1
8	86	1030	1	0	0
9	101	1100	1	1	1

- ii. From the graph, what do you think is a reasonable number of clusters for the data points? Label the clusters in the graph as C_1 , C_2 , C_3 , etc.
- iii. Which clustering technique is suitable to find the above clusters?
- iv. Replace the temperature and pressure columns in Table 4.23 with attributes C_1 , C_2 , etc. Construct a transaction matrix using the new attributes (along with attributes Alarm1, Alarm2 and Alarm3).
- v. Derive all the frequent itemsets having support $\geq 30\%$.

13. Interestingness Measures

In the original association rule formulation, support and confidence are the measures used to eliminate uninteresting rules. The purpose of the following exercise is to illustrate the applicability of other rule interest measures in assessing the quality of a rule. For each of the measure given below, compute and rank the following rules in decreasing order according to their respective measures. Use the transactions given in Table 4.21.

Rules: $B \longrightarrow C$, $A \longrightarrow D$, $B \longrightarrow D$, $E \longrightarrow C$, $C \longrightarrow A$.

- (a) Support.
- (b) Confidence.
- (c) $\text{Interest}(X \longrightarrow Y) = \frac{P(X,Y)}{P(X)}P(Y)$.
- (d) $\text{IS}(X \longrightarrow Y) = \frac{P(X,Y)}{\sqrt{P(X)P(Y)}}$.
- (e) $\text{Klogen}(X \longrightarrow Y) = \sqrt{P(X,Y)} \times (P(Y|X) - P(Y))$, where $P(Y|X) = \frac{P(X,Y)}{P(X)}$.
- (f) $\text{Odds ratio}(X \longrightarrow Y) = \frac{P(X,Y)P(\bar{X},\bar{Y})}{P(X,\bar{Y})P(\bar{X},Y)}$.

14. Consistency between Measures

Given the rankings you had obtained in question 1, 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?

15. Negative Associations

3. The original association rule mining framework defined by Agrawal et al. considers only itemsets that occur frequently together in the transaction database. There are situations in which itemsets that are infrequent can be potentially interesting. For instance, the itemset TV, DVD, \neg VCR refers to customers who buy TV and DVD but not VCR. In this problem, you are asked to extend the association rule framework to negative itemsets (i.e. itemsets that contain presence as well as absence of items). We will use the negation symbol (\neg) to refer to absence of items.

- (a) A naive way to derive negative itemsets is to extend each transaction to include absence of items as shown in the table below.

Table 4.24. Example of numeric data set.

TID	TV	\neg TV	DVD	\neg DVD	VCR	\neg VCR	...
1	1	0	0	1	0	1	...
2	1	0	0	1	0	1	...

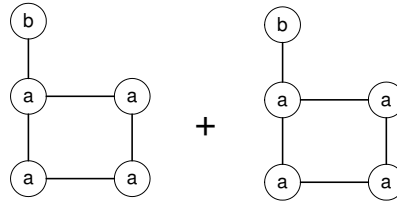
- i. Suppose the transaction database contains 1000 distinct items. What is the total number of positive itemsets that can be generated from these items? (Note: A positive itemset do not contain any negated items).
 - ii. What is total number itemsets (that may contain both positive or negative items) that can be generated from these items?
 - iii. Explain why such a naive method of extending each transaction with negative items would not be practical for deriving negative itemsets.
- (b) Consider the transaction table in Problem 10. What are the support and confidence for the negative association rule \neg Pepsi \rightarrow Coke?
- (c) Another method (suggested by Savasere et al [159]) is to compute the expected support of an itemset. If the expected support of an itemset, say Coke, Pepsi, is significantly smaller than its actual support, then the itemset can be potentially used for generating negative association rules. There are various methods to determine the expected support of a set of items. Suggest a method to determine the expected support of the itemset Coke, Pepsi.

16. Maximal and Closed Itemsets

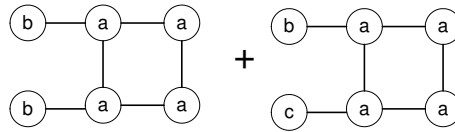
Given the lattice structure shown in Figure 8a and the transactions given in Table 4.21, label each node (itemset) as M if the itemset is a maximal frequent itemset, C if it is a closed frequent itemset, N if it is frequent but neither maximal nor closed, and I if it is infrequent. (Use minimum support equals to 30%.)

17. Frequent Subgraph Mining

Draw all candidate subgraphs obtained from joining the pair of graphs shown in the diagrams (a) and (b) below.



(a)



(b)

18. Interestingness Measures

- Prove that the ϕ coefficient is equals to 1 if and only if $f_{11} = f_{1+} = f_{+1}$.
- Show that if A and B are independent, then $P(A, B) \times P(A, \overline{B}) = P(A, \overline{B}) \times P(\overline{A}, B)$.
- Show that if $\frac{P(B|A) - P(B)}{1 - P(B)} < -1$, then $0 \leq \frac{P(A|B) - P(A)}{1 - P(A)} \leq -1$.
- Show that Yule's Y and Q coefficients are standardized versions of the odds ratio.
- Write a simplified expression for the value of each measure shown in Table 4.11 when the variables are statistically independent.

4.11 Grab bag

4.11.1 Dependence Rules

Association rules, as we learnt so far, use support and confidence measures to quantify and filter the relationships. A rule of the form $X \implies y$ is interesting if it

provides us two pieces of information: a) X and y occur together in sufficiently large number of transactions (support), so that their co-occurrence can be considered statistically significant, and b) whenever X is present in a transaction, y is also present in it with high probability (confidence). Confidence of this rule can be simply looked at as a *conditional probability of presence of y given presence of X* . Now, consider an example to critically evaluate the quality of this information.

In a computer store transaction data, out of 200 buyers of operating systems, 160 bought Windows operating system and 100 customers bought Linux operating system. Of the Linux buyers, 60 also bought the Windows operating system. Now a rule $Linux \rightarrow Windows$ has sufficiently large support (30%) and also sufficiently high confidence (60%). So, looking at this rule, one might conclude that whenever Linux is sold, Windows is also sold with 60% chance. Now, look at it from a different angle. If we didn't know anything about a customer, we know that she will buy Windows with 80% probability. However, the moment we know that a customer has bought Linux, her probability of purchasing Windows has gone *down* from 80% to 60%. This implies that although the rule $Linux \rightarrow Windows$ has sufficient confidence, it is misleading. Here is the reason. Let us assume that the store-owner decides to push the Windows sales up by 200 by sending ads to customers. If he were to follow just the prior probability of Windows, then he would need to mail ads to $200/0.8 = 250$ customers. Whereas, if he were to base his decision on the $Linux \rightarrow Windows$ rule, then he would need to increase the Linux sale by $200/0.6 = 333$, by identifying and attracting 333 more Linux customers. Moreover, following the first decision option, the store-owner needs to identify profile of his store's generic customer, whereas for second option, he has to identify profiles of Linux customers, which may be more difficult. On the other hand, if the confidence of the rule $Linux \rightarrow Windows$ was more than the prior probability of Windows, then following this rule would have made more sense, rather than merely relying on the prior probability.

This example stresses the need to get more information than mere conditional probability (confidence) of a rule. The impact of conditioning a presence of some item on presence of others can be judged by looking at whether the conditional probability is higher than or lower than or same as the prior probability of that item. This notion is precisely captured by *statistical dependence*. In statistics, a set of events $\{x, y\}$ is said to be independent, if their joint probability is the product of their individual probabilities; i.e., $P(x, y) = P(x)P(y)$. In the example above, the events were $Product_Sold = Linux$ or $Product_sold = Windows$. If the events are not independent, we can look at the ratio $P(x, y)/P(x)P(y)$. If the ratio is more than 1, then events are said to have *positive* dependence, which means that conditioning the presence of any of the events (x or y), on the presence of other, causes increase in the prior probability of occurrence of that event. This can be seen by rewriting the equation $P(x, y)/P(x)P(y) > 1$ as $P(x, y)/P(x) > P(y)$. The left hand side of the last equation is the confidence ($P(y||x)$) of the rule $x \rightarrow y$ rule, which conditions the presence of y on presence of x . The right hand side is the prior probability of y . Similarly, *negative* dependence means that $P(x, y)/P(x)P(y) < 1$, and can be

interpreted as decrease in prior probability of either of the events after conditioning its presence on the other's presence.

Thus, statistical dependence yields a possibly more useful information between two events, rather than merely looking at the confidence of a rule. Essentially, statistical dependence combines the prior probabilities and conditional probabilities into a single measure.

4.11.2 Comparison between Classification and Association Rule Mining

Although an association rule looks quite similar to a classification rule, their syntax are actually quite different. The antecedent of a classification rule may contain any number of input variables, while the consequent must have only one variable that represents the target class. For association rules, there is no distinction between input and target variables. As a result, the antecedent and consequent of an association rule may involve any number of binary variables.

Classification and association rule mining also differ from an application perspective. Classification seeks to develop a global representation of the data and is often used to predict the class label of unknown instances. Its objective function is to maximize the accuracy or F-measure of the classification models. To ensure that the models can produce accurate predictions, the data is partitioned into two disjoint subsets, *i.e.*, training and test sets. The training set is used to build the model while the test set is used to evaluate the performance of the model. On the other hand, association rule mining seeks to derive interesting local structures in the data. Its objective is to find all rules having support and confidence above the user-specified thresholds. The entire data set is used to extract association rules. This will ensure that the support and confidence values computed from the data are reliable probability estimates of the significance of the rules.

Techniques for building classification models, such as decision trees and rule-based classifiers, often use a greedy heuristic to efficiently search the exponential hypothesis space. Thus, the generated models may not be globally optimal. Association rule mining algorithms can perform a complete search over the entire space to look for rules that satisfy the support and confidence constraints. A complete search is possible because the algorithms employ an effective strategy for pruning the exponential search space. Since association rule mining algorithms have more effective search strategy, why should we use classification algorithms at all? The answer is because the hypothesis space of association rules is less expressive and is limited only to binary variables, whereas classification rules can be created for variables of any data types. Thus, there is a trade-off between the expressiveness of the hypothesis space and the completeness of the search algorithm. The more expressive is the hypothesis space, the more difficult it is to effectively search the entire space.

Despite their differences, association patterns can complement the task of building classification models in many ways. In this section, we describe two techniques

for utilizing association patterns to solve a classification problem:

1. Using association rules to build a rule-based classifier.
2. Using frequent itemsets to build a Bayesian classifier.

A detailed explanation of these methods is presented in the next sections.

4.11.3 General Procedure for Building Classification Models

Building classification models from association patterns involves the following steps:

Pattern Generation. The first step is to generate association patterns from the input data. This may require a preprocessing phase to convert the raw data into binary-valued format. Specifically, continuous-valued attributes must be discretized prior to creating a new binary variable for each discrete interval. For categorical attributes, a new binary variable is created for every attribute-value pair. Existing association mining algorithms can be applied once the attributes are transformed into binary variables. One important issue is how to handle data sets containing rare classes. We might have to apply a very low minimum support threshold to extract association patterns involving the rare class. This approach may not work well because lowering the support threshold tends to increase the number of patterns for the majority class, thereby leading to a substantial increase in the computation time of the algorithms. Another way to overcome this problem is to use different minimum support thresholds for each class, a technique that was described in Section 4.6. For example, if 5% of the instances belong to the rare class and the remaining 95% belong to the majority class, then instead of using a 1% minimum support threshold for the entire data set, we may use $1\% \times 5\% = 0.05\%$ support threshold for the rare class, and $1\% \times 95\% = 0.95\%$ support threshold for the majority class.

Model Construction. The association patterns discovered in the previous step can be used to build a classification model. However, there are several issues that need to be considered carefully. First, not all patterns are useful for classification. Only patterns that contain the class variable are of interest to the classification algorithm. Second, some of the redundant or overlapping patterns should be eliminated as they do not convey any additional information to the classification algorithm. Eliminating them early can help make the computation more efficient. The question is, how to determine which patterns should be eliminated? A rationale (*pattern selection criteria*) for choosing some patterns over the others must be given.

One potential drawback of using association patterns for building classification models is the information loss problem, which arises when the input data is discretized. Information is lost because the classifier has no knowledge of the different continuous values that are mapped onto the same discrete interval. If the original continuous data is not retained during model construction, then the errors incurred by the models may increase as a result of approximation errors due to discretization.

4.11.4 Using association rules for rule-based classifiers

Associative classification is a technique that uses association rules for creating the rule-set of a rule-based classifier. The main advantage of this technique is that it uses rules generated from a complete search rather than a greedy search.

In associative classification, the rules have the form of $C \longrightarrow y$, where C is an itemset and y is a class label. These rules are also known as *class association rules*. The itemset from which a class association rule is derived is called a *rule-item*, while the set C appearing in the antecedent of the rule is called a *conditional set*. For example, the class association rule $\{p, q\} \longrightarrow y$ is derived from the rule-item $\{p, q, y\}$ and contains the conditional set $\{p, q\}$.

It is important to note that all rule-items must contain a class variable. The size of a rule-item depends on the number of items that belong to its corresponding conditional set. For example, $\{p, y\}$ is a size-1 rule-item and $\{p, q, y\}$ is a size-2 rule-item.

In this section, we present two variants of the associative classification algorithms:

1. CBA (Classification Based on Associations), which creates an ordered rule-set for a rule-based classifier.
2. CMAR (Classification based on Multiple Association Rules), which creates an unordered rule-set for a rule-based classifier.

Pattern Generation

In CBA, the data set is initially partitioned into training and test sets. CBA then applies a variant of the Apriori algorithm to generate rule-items from the training set. Since it does not generate the conditional sets explicitly, each rule-item must keep track of two counts (1) the support of the conditional set and (2) the support of the rule-item. The rule-items are generated in a level-wise manner. The size-1 rule-items are created first, followed by the size-2 rule-items, and so on. The size- k frequent rule-items are used to create the size- $(k + 1)$ rule-items. Unlike Apriori, CBA generates the size- k class association rules immediately after the size- k frequent rule-items are found.

Example 25 Suppose the data set contains four items $\{a, b, c, d\}$ and two classes, $\{y_1, y_2\}$. Initially, the CBA algorithm will generate eight size-1 candidate rule-items, $\{a, y_1\}$, $\{a, y_2\}$, $\{b, y_1\}$, $\{b, y_2\}$, \dots , $\{d, y_2\}$. It then counts the conditional set support and rule-item support of each rule-item. All rule-items whose rule-item support is less than the minimum support threshold are then pruned away. Although there might be more than one rule-item with the same conditional set, the algorithm will generate only one class association rule for each conditional set. For example, $\{a, y_1\}$ and $\{a, y_2\}$ are two rule-items that have the same conditional set. During rule generation, CBA extracts only the class association rule that has the highest confidence. The algorithm proceeds to use the size-1 frequent rule-items to generate size-2 candidate rule-items.

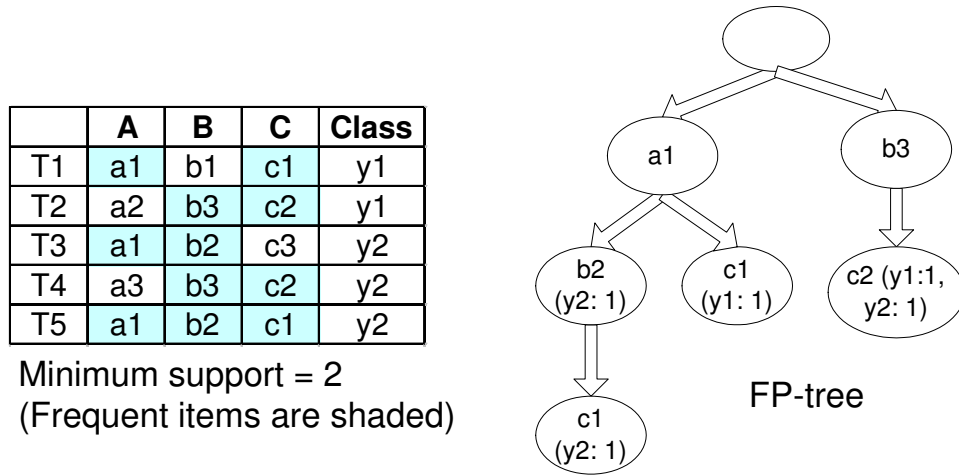


Figure 4.41. An illustrative example for the CMAR algorithm.

The original version of CBA uses a single minimum support threshold for all the classes. A later version of the algorithm adds more flexibility by allowing the user to choose different thresholds for each class. In order to ensure that the rules can generalize well to unseen instances, CBA compares the pessimistic error rate of a rule, r , against the pessimistic error rate of its parent rule r' (where r' is a subset rule obtained by removing one of the items in r). If the pessimistic error rate for r is higher than r' , then r will be pruned.

Unlike CBA, CMAR uses the FP-growth algorithm to generate the frequent patterns in a more efficient manner. Initially, an FP-tree is constructed from the training set. Each node of the tree contains an attribute value along with its distribution of class labels, as illustrated in the example below.

Example 26 Consider the training set shown in Figure 4.41. If the minimum support threshold is equal to 2, then only $a1$, $b2$, $b3$, $c1$, $c2$ are frequent. An FP-tree can be constructed to represent the data set in a compact manner using only the frequent items. For example, the first tuple ($A = a1, B = b1, C = c1$) contains an infrequent item $b1$ that can be ignored. The remaining items in the tuple ($A = a1, C = c1$) are inserted into the tree, thus forming a path from the root to the $c1$ node, as shown in Figure 4.41. The class label $y1$ is then attached to the last node, which keeps track of the count distribution for each class label. As another example, the second and fourth tuple of the training set can be aggregated to obtain the right-most branch of the tree.

The FP-tree structure is used to facilitate the automatic discovery of frequent rule-items from data. Once the frequent rule-items are found, class association rules are extracted and inserted into another tree structure called a CR-tree. The CR-tree is needed for efficient retrieval of class association rules. A CR-tree differs from an FP-tree in the following way:

- A CR-tree contains only rule-items that satisfy the minimum support and minimum confidence requirements. In contrast, an FP-tree contains the frequent items of each transaction in the data set.
- Unlike an FP-tree, each node in a CR-tree can have at most one class label, which corresponds to the most dominant class for that node.

Classifier Construction

CBA uses the sequential covering method to select the class association rules for constructing its classifier. First, the rules are sorted according to their confidence and support values. Specifically, given a pair of class association rules, r_1 and r_2 , the rules are ordered based upon the following criteria:

1. if confidence of r_1 is greater than or equal to r_2 , then r_1 has higher precedence.
2. If their confidence are the same, then the rule having the higher support will have a higher precedence.
3. If none of the conditions above are applicable, then the longest rule will have the highest precedence.

The CBA algorithm constructs its rule-set in an iterative manner. Initially, the rule set is empty. The algorithm then chooses a rule that covers the training instances, with the highest precedent rule selected first. Each time a rule is selected, the training instances covered by the rule will be removed and a default class (the majority class) is assigned to cover the remaining instances. Thus, each time a rule is added, a new classifier C_k is constructed. We can compute the number of misclassification errors of the current classifier. The rule selection step is repeated until no rule or training instances remain. Finally, CBA chooses the classifier that produces the lowest errors.

One of the problems with this approach is that the support and confidence of the rules are computed only once, prior to removing any training instances. As a result, the accuracies of the ordered rules may change after adding the first rule to the rule set. This problem is perhaps unavoidable because if the support and confidence values of the rules are to be re-computed, this will negate the advantages of generating all the class association rules unless each rule keeps track of the transaction ids for which the rule is applicable (this is known as the vertical mining approach.)

Another major problem with this approach is the use of confidence as the main criteria for selecting the rules. Confidence tends to prefer rules that are generated for the majority class. This problem is illustrated in the example below.

Example 27 Suppose the data set contains 10 instances of the rare class y_r and 990 instances of the majority class y_m . Assume that the condition C can be applied 8 times to the rare class and 20 times to the majority class. Thus, the confidence of the rule $C \rightarrow y_r$ is $8/(8 + 20) = 0.286$ while the confidence for the rule $C \rightarrow y_m$

is $20/(8 + 20) = 0.714$. Thus, the second rule is ranked much higher than the first rule even though the first rule seems to work very well for the rare class (it predicts correctly eight out of ten times).

This problem can be addressed if we use other measures to order the rules. For example, the ratio measure can be used to compare the fraction of instances belonging to each class that are covered by the rule. The ratio measure is defined as:

$$ratio = \frac{\frac{c_1}{n_1}}{\frac{c_2}{n_2}},$$

where c_1 and c_2 are the fractions of instances covered by the two classes and the size for each class is n_1 and n_2 , respectively. The ratio measure remains invariant if we stratify the data set so that $n_1 = n_2$, whereas confidence will change if the data is stratified.

4.11.5 Using association patterns for Bayesian classifiers

In this approach, frequent itemsets are used to provide joint probability estimates of the data. In Bayesian classifier, the task is to estimate the posterior probability of the class given the attributes. The posterior probability can be expressed as the joint probabilities using the Bayes theorem.

There are many ways to estimate the posterior probability. For example, suppose we are given the following attributes: (x_1, x_2, \dots, x_6) . How do we estimate the posterior probability, $P(y|x_1, x_2, \dots, x_6)$? First, we can use the Bayes theorem to express the probability in terms of the joint probability, $P(x_1, x_2, \dots, x_6, y)$. Below, we provide three different ways to compute the joint probability:

$$\begin{aligned} P(x_1, x_2, \dots, x_6, y) &= P(x_1, x_2, y) \times P(x_3, x_4|x_1, x_2, y) \times P(x_5, x_6|x_1, x_2, x_3, x_4, y) \\ P(x_1, x_2, \dots, x_6, y) &= P(x_1, x_2, x_3, y) \times P(x_5|x_1, y) \times P(x_4, x_6|x_2, x_3, y) \\ P(x_1, x_2, \dots, x_6, y) &= P(x_5, y) \times P(x_1, x_3, x_4|x_5, y) \times P(x_2, x_6|x_1, x_3, x_5, y) \end{aligned}$$

where each equation makes certain assumption regarding the independence of the attributes. Each conditional probability on the right hand side of the above equations can be re-written as a joint probability that can be estimated using the support of itemsets.

4.11.6 Support and Confidence

In the original association rule formulation, the support and confidence measures are used to assess the importance of the discovered rules. However, data mining practitioners often face with the dilemma of having to choose the appropriate minimum support and minimum confidence thresholds for their particular data sets. This is because choosing the appropriate minimum support threshold involves a trade-off between missing interesting patterns and computational efficiency. A minimum support threshold chosen too high has the effect of discovering very few patterns and

can potentially lose many interesting patterns. On the other hand, if the minimum support threshold is too low, then a large number of patterns can be generated, which increases the computation time of the algorithm as support-based pruning may no longer be effective.

The choice of minimum support threshold also depends on the nature of the data. For example, if the data set is too dense, then many items will be associated with many other items. In this case, it is computationally expensive to use a low minimum support threshold for finding the frequent itemsets and association rules. On the other hand, if the data set is sparse, one may apply a much lower support threshold without degrading the performance of the algorithm considerably.

While there is no absolute procedure that dictates how to select the best thresholds, there are several guidelines one may follow. A hint as to what support threshold one should use is given by the average width of the transactions. Suppose the data set contains d items and N transactions, with an average transaction width equals to w . Assuming that each item is equally likely to appear in the data set, we would expect each item to appear Nw/d times. This suggests that using a minimum support threshold of w/d , we can find items that appear more frequently than expected according to the uniform distribution assumption. In addition, since w is related to the density of the data, the above analysis suggests that it is safe to use a higher support threshold for dense data sets.

Another approach is to start from a high minimum support threshold, and then gradually lower the threshold until a manageable number of interesting patterns are found. This approach is more time-consuming as it may require multiple runs of the association rule mining algorithm.

Selection of the minimum confidence threshold is equally challenging. In most situations, one would consider using a confidence threshold greater than 50% because if any rule $A \longrightarrow B$ has a confidence less than 50%, then the opposite rule $A \longrightarrow \neg B$ is more predictive (as its confidence is higher than 50%). Nevertheless, there are data sets for which most of the rules have confidence less than 50%. For example, if most of the itemsets of size 2 have support less than half of the support for its corresponding items, i.e., $\sigma(A, B) < 0.5 \min(\sigma(A), \sigma(B))$, then it would be extremely difficult to find high confidence rules unless the minimum support threshold is very low.

Cluster Analysis

Cluster analysis divides data into meaningful or useful groups (clusters). If meaningful clusters are the goal, then the resulting clusters should capture the ‘natural’ structure of the data. For example, cluster analysis has been used to group related documents for browsing, to find genes and proteins that have similar functionality, and to provide a grouping of spatial locations prone to earthquakes. However, in other cases, cluster analysis is only a useful starting point for other purposes, e.g., data compression or efficiently finding the nearest neighbors of points. Whether for understanding or utility, cluster analysis has long been used in a wide variety of fields: psychology and other social sciences, biology, statistics, pattern recognition, information retrieval, machine learning, and data mining.

This chapter provides an introduction to cluster analysis in the field of data mining. We begin with a brief description of what cluster analysis is and is not, followed by a discussion of different ways a set of objects can be divided into a set of clusters and an explanation of the different types of clusters. As clustering algorithms are based either on the notion of similarity (distance) or density, these two concepts are then discussed in some detail. This is followed by an examination of other important characteristics of clustering algorithms. Most of the rest of the chapter is used to describe various types of clustering techniques and to illustrate the central concepts embodied by those techniques. However, the final section of this chapter is devoted to cluster validity, i.e., methods for evaluating the ‘goodness’ of the clusters produced by a clustering algorithm. While this chapter strives to be relatively self-contained from a conceptual point of view, the breadth of cluster analysis means that many details have necessarily been omitted, and, consequently, many references to relevant books and papers are provided in the bibliographic remarks.

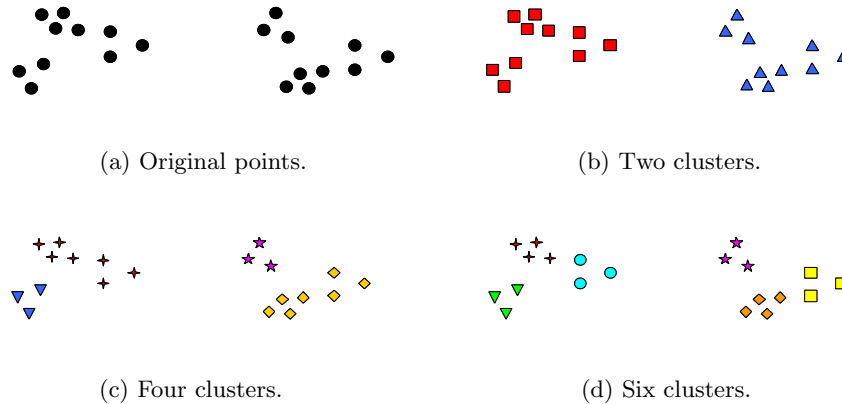


Figure 5.1. Different ways of clustering the same set of points.

5.1 Introduction

5.1.1 What is cluster analysis?

Cluster analysis groups data objects based on information found in the data that describes the objects and their relationships. The goal is that the objects in 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, what constitutes a cluster is not well defined, and clusters are often not well separated from one another. Nonetheless, most cluster analysis seeks, as a result, a partition of the data into non-overlapping groups. Fuzzy clustering, described later in Section 5.9.1, is one exception to this, and allows an object to partially belong to several groups. In hierarchical clustering, the clusters are nested, i.e., can have subclusters, but are otherwise non-overlapping.

To better understand the difficulty of deciding what constitutes a cluster, consider Figure 5.1, which shows twenty points and three different ways that they can be divided into clusters. If we allow clusters to be nested, then the most reasonable interpretation of the structure of these points is that there are two clusters, each of which has three subclusters. See figures 5.1b and 5.1d, 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. See Figure 5.1c. Thus, we stress once again that the definition of what constitutes a cluster is imprecise and that the best definition depends on the type of data and the desired results.

5.1.2 What is not cluster analysis?

In this section, we briefly illustrate the difference between cluster analysis and other techniques used to divide data objects into groups. For instance, cluster analysis

is not classification in the sense of Chapter 3. Although cluster analysis can be regarded as a form of classification in that it creates a labelling of objects with class (cluster) labels, it derives these labels only from the data. In contrast, classification in the sense of Chapter 3 is ‘supervised’ classification, i.e., new, unlabelled 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. However, when the term ‘classification’ is used without any qualification, it typically refers to supervised classification.

Also, while the terms ‘segmentation’ and ‘partitioning’ are often used as synonyms for ‘clustering,’ frequently these terms are used for approaches outside the traditional bounds of cluster analysis. For instance, ‘segmentation’ is often used to refer to a division of data into groups using simple techniques e.g., splitting people into groups based on income or their last name. Nonetheless, some work in market segmentation is cluster analysis, and there is also a mathematical framework for data mining that regards many data mining activities as ‘segmentation problems.’ Likewise, while many techniques for partitioning a data set do not fall within the area of cluster analysis, there is a strong connection between graph partitioning and clustering. See Section 5.4.

As a final illustration of the distinction between clustering and other techniques for dividing data objects into groups, we present one last example: a database or search engine query. Although a query divides a set of records or web pages into two groups—those retrieved by the query and those that are not—we do not regard the resulting two sets of data objects as clusters. A query represents a selection of objects while clustering strives to organize a data set by grouping similar objects.

5.1.3 Different Types of Clusterings

An entire collection of clusters is commonly referred to as a *clustering*, and in this section, we describe various types of clusterings: hierarchical versus partitional, exclusive versus non-exclusive, fuzzy versus non-fuzzy, partial versus complete, and object clustering versus attribute clustering. The most commonly discussed difference between collections of clusters is whether the clusters are nested or unnested or, in more traditional terminology, whether a set of clusters is *hierarchical* or *partitional*. A partitional or unnested set of clusters 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, i.e., a partition of the data objects. We previously saw examples of partitional clustering in figures 5.1b-d. A hierarchical or nested clustering is a set of nested clusters organized as a hierarchical tree, where the leaves of the tree are singleton clusters of individual data objects, and where the cluster associated with each interior node of the tree is the union of the clusters associated with its child nodes. The tree that represents a hierarchical clustering is called a *dendrogram*, a term that comes from biological taxonomy. Figure 5.2 shows an example of a nested clustering and the corresponding dendrogram.

The distinction between a hierarchical and partitional clustering is not as great as it might seem. Specifically, by looking at the clusters on a particular level of

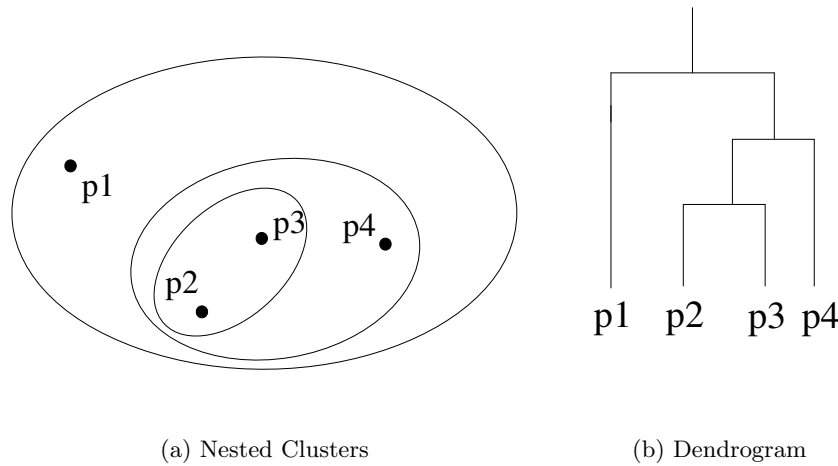


Figure 5.2. A hierarchical clustering of four points shown as nested clusters and as a dendrogram.

a hierarchical tree, a partitional clustering can be obtained. For example, in Figure 5.2, the partitional clusterings, from bottom to top, are $\{\{1\}, \{2\}, \{3\}, \{4\}\}$, $\{\{1\}, \{2, 3\}, \{4\}\}$, $\{\{1\}, \{2, 3, 4\}\}$, and $\{\{1, 2, 3, 4\}\}$. As another example, figures 5.1b and 5.1d can be regarded as two partitional clusterings from a hierarchical clustering of the 20 points of Figure 5.1a. Thus, a hierarchical partitioning can be viewed as a sequence of partitional clusterings, and a partitional clustering can be viewed as a particular ‘slice’ of a hierarchical clustering.

In the hierarchical clustering of Figure 5.2, the set of clusters at a given level and the set of clusters of the level immediately preceding it are the same except that one of the clusters in the given level is the union of two of the clusters from the immediately preceding level. While this approach is traditional and most common, it is not essential, and a hierarchical clustering can merge more than two clusters from one level to the next higher one. Figure 5.3 shows an example of a non-traditional hierarchical clustering.

In a *non-exclusive clustering*, a point can belong to more than one cluster. In the most general sense, a non-exclusive clustering is used to reflect the fact that an object may belong to more than one group (class) at a time, e.g., a person at a university may be both an enrolled student and an employee of the university. Note that the exclusive versus non-exclusive distinction is independent of the partitional (unnested) versus hierarchical (nested) distinction.

In a more limited sense, a non-exclusive clustering is sometimes used when an object could reasonably be placed in any of several clusters. Rather than make a somewhat arbitrary choice and place the object in a single cluster, such objects are placed in all of the “equally good” clusters. This type of non-exclusive clustering does not attempt to deal with multi-class situations, but is more similar to the fuzzy clustering approach that we describe next.

In a *fuzzy clustering*, every point belongs to every cluster, but with a weight

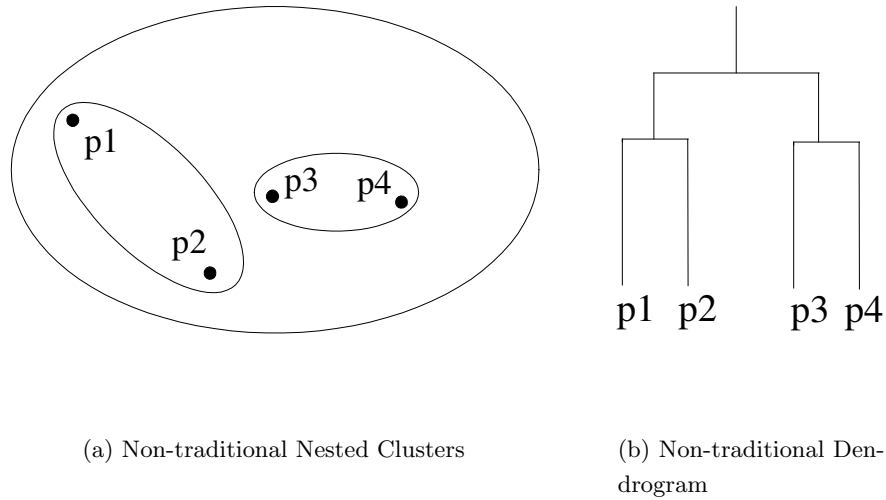


Figure 5.3. A non-traditional hierarchical clustering of four points shown as nested clusters and as a dendrogram.

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 where an object belongs to any set with a weight that is between 0 and 1. For any object, the sum of the weights must equal 1.) The motivation for this sort of clustering is to avoid the arbitrariness of assigning an object to only one cluster. Since the cluster membership weights for any object sum to 1, a fuzzy clustering does not capture multi-class situations where an object belongs to multiple classes, e.g., when a person is both a student and an employee. In a very similar way, some probabilistic clustering techniques assign each point a probability of belonging to each cluster, and these probabilities must also sum to one. Often a fuzzy or probabilistic clustering is converted to a 'crisp' clustering by assigning each object to the cluster for which its membership weight (probability) is highest.

A *complete clustering* assigns every object to a cluster, whereas a *partial clustering* does not. The motivation for a partial clustering is that not all objects in a data set may belong to well-defined groups. Indeed, many objects in the data set may represent noise, outliers, or "uninteresting background." For example, some newspaper stories may share a common theme, e.g., global warming, while other stories are more generic or one-of-a-kind. Hence, an application that was interested in finding the important topics in the last month of stories might want to find only clusters of documents that are tightly related by a common theme. Nonetheless, in many cases, a full clustering of the objects is desired, e.g., an application may want to use clustering to organize documents for browsing and may need to guarantee that all documents can be browsed, i.e., that every document belongs to some cluster.

While most clustering is *object clustering*, i.e., the clusters are groups of objects,

attribute clustering, i.e., the clusters are groups of attributes, can also be useful. For instance, given a set of documents, we may wish to cluster the words (terms) of the documents, as well as or instead of the documents themselves. Attribute clustering is less common than object clustering, as are clustering techniques that attempt to cluster the both objects and attributes simultaneously.

5.1.4 Different Types of Clusters

As mentioned above, the term, ‘cluster,’ does not have a precise definition. However, several working definitions of a cluster are commonly used and are described in this section. In these descriptions, we will use two dimensional points as our data objects to illustrate the differences between the different types of clusters, but the cluster types described are applicable to a wide variety of data sets.

Well-Separated A cluster is a set of points such that any point in a cluster is closer (or more similar) to every other point in the cluster than to any point not in the cluster. Sometimes a threshold is used to specify that all the points in a cluster must be sufficiently close (or similar) to one another. Figure 5.4 shows two well separated clusters.

Center-based Cluster A cluster is a set of objects such that an object in a cluster is closer (more similar) to the ‘center’ of a cluster, than to the center of any other cluster. The center of a cluster is often a centroid, i.e., the average of all the points in the cluster, or a medoid, the most ‘representative’ point of a cluster. Figure 5.5 shows four center-based clusters.

Contiguous Cluster (Nearest neighbor or Transitive Clustering) A cluster is a set of points such that a point in a cluster is closer (or more similar) to one or more other points in the cluster than to any point not in the cluster. Figure 5.6 shows eight contiguous clusters.

Density-based A cluster is a dense region of points, which is separated by low-density regions, from other regions of high density. This definition is more often used when the clusters are irregular or intertwined, and when noise and outliers are present. Note that the contiguous definition would find only one cluster in Figure 6. Also note that the three curves don’t form clusters since they fade into the noise, as does the bridge between the two small circular clusters. Figure 5.7 shows six density-based clusters.

Shared Property (Conceptual Clusters) More generally, we can define a cluster as a set of points that share some property. This definition encompasses all the previous definitions of a cluster, e.g., points in a center based cluster share the property that they are all closest to the same centroid. However, the shared property approach also includes new types of clusters. To illustrate, consider Figure 5.8a which shows a rectangular area (cluster) that is adjacent to a rectangular one and Figure 5.8b which shows two intertwined circles



Figure 5.4. Two well-separated clusters of 2 dimensional points.

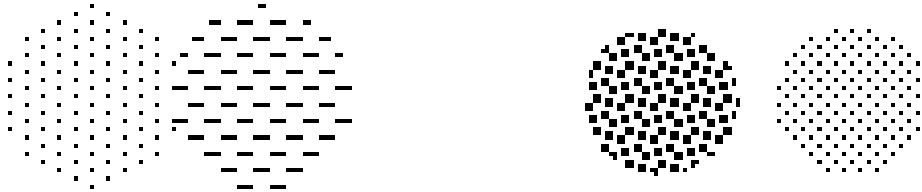


Figure 5.5. Four center-based clusters of 2 dimensional points.

(clusters). In both cases, a cluster finding algorithm would need to have a very specific ‘concept’ of a cluster in mind to successfully find these clusters. Finding such clusters is sometimes called ‘conceptual clustering.’ However, too sophisticated a notion of a cluster would bring us into the area of pattern recognition, and thus, we will only consider the simpler types of clusters in this chapter.

Clusters Defined Via Objective Functions Another general approach to defining a set of clusters by using an objective function that, given a set of clusters, returns a positive real number that measures how well this set of clusters fulfills the clustering objective. (Without loss of generality, for the remainder of this discussion assume that we want to minimize the objective function instead of maximizing it.) The idea is that we define our clustering as the set of clusters that minimizes the objective function. To illustrate, for two dimensional points, a common objective function is squared error, which is computed by calculating the sum of the squared distance of each point to the center of its cluster. For a specified number of clusters, K we could then define our clusters to be the partitioning of points into K groups that minimizes this objective. The K-means algorithm—see Section 5.6.1—is based on this objective function. The use of objective functions for clustering is discussed

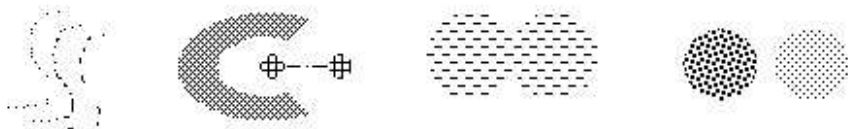


Figure 5.6. Eight contiguous clusters of 2 dimensional points.

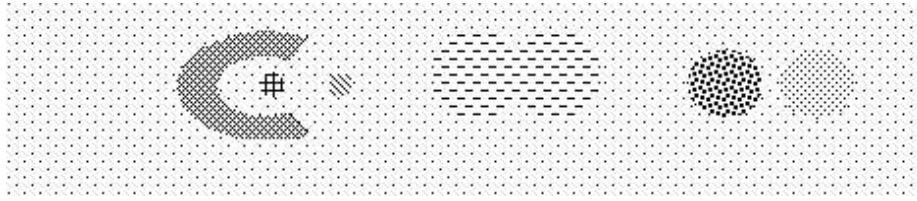


Figure 5.7. Six dense clusters of 2 dimensional points.

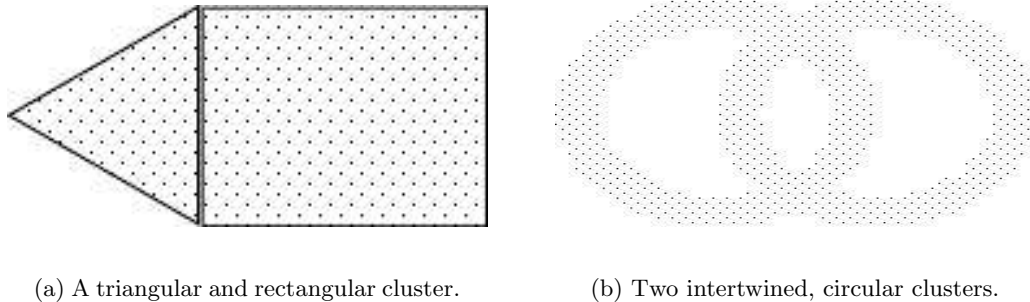


Figure 5.8. Examples of shared property or 'conceptual' clusters.

further in Section 5.4.

5.2 Similarity and Distance

If the goal of clustering is to put similar objects in the same cluster, then the measure of similarity is crucial. Informally, the *similarity* between two objects is a numerical measure of the degree to which the two objects are alike. The only absolute requirement on similarities is that they are *higher* when pairs of objects are more alike. However, similarities are usually non-negative and are often between 0 (no similarity) and 1 (complete similarity). Likewise, 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 below, distance is more properly used to refer to a special class of dissimilarities. For convenience, the term, *proximity*, is used to refer to either a similarity or a dissimilarity.

Since the proximity between two objects is, in general, a function of the proximity between the corresponding attributes of the the two objects, we begin this section with a description of similarity and dissimilarity between objects having only one simple attribute. With this background, we then describe some important distance and similarity measures for objects with multiple attributes. This includes measures such as correlation and Euclidean distance (and its generalizations), which are useful for 'dense' data such as time series or two-dimensional points, e.g., as well as

measures such as the Jaccard and cosine measures which are useful for sparse data such as documents. Next, several important issues concerning proximity measures are considered, and the section concludes with a brief discussion on selecting the ‘right’ proximity measure.

5.2.1 Similarity and Dissimilarity Between Simple Attributes

We begin with some simple examples, i.e., with objects that consist only of a single attribute. Consider first, 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 have the same nominal value or they do not. Hence, in this case the similarity between two objects is traditionally defined as 1, if attribute values match, and 0, if they do not. A dissimilarity would be defined in the opposite way: 0, if the objects’ attribute values match, and 1, if they do not.

For objects that consist of a single ordinal attribute, the situation is more complicated since we also have information about order that should be taken into account. Thus, if the attribute measures the quality of a product on the scale $\{poor, fair, OK, good, wonderful\}$, then it seems reasonable that a product, $P1$, which is rated *wonderful*, would be closer to a product $P2$, which is rated *good*, than 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., $\{poor = 0, fair = 1, OK = 2, good = 3, wonderful = 4\}$. Then, $dissimilarity(P1, P2) = 3 - 2 = 1$ or, if we want the dissimilarity to fall between 0 and 1, $dissimilarity(P1, P2) = \frac{3-2}{4} = 0.25$. A similarity for nominal attributes can be defined, as similarity often is, as $1 - dissimilarity$.

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

If objects have a single interval or ratio attribute, then 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.” Notice that in cases such as these, the dissimilarities typically range from 0 to ∞ , rather than from 0 to 1.

The proximity of interval and ratio scale attributes is often defined in terms of dissimilarity (differences) since it seems so natural, but it is possible to use similarity as well. The easiest approach is to define similarity as the negative of the dissimilarity. However, if we want a similarity with a range between 0 and 1, then we can set $similarity = \frac{1}{1+distance}$. This distorts the scale, i.e., there is less discrimination between highly dissimilar objects, which will all have similarities close to 0, but

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

Table 5.1. Similarity and dissimilarity for simple attributes

often this is what is wanted. If the range of the interval or ratio attribute is limited, then another option is to set $\text{similarity} = \frac{\text{distance} - \text{minimum_distance}}{\text{maximum_distance} - \text{minimum_distance}}$.

Table 5.1 summarizes the previous discussion. In this figure, p and q are two objects that have one attribute of the indicated type. Also, d and s are, respectively, the dissimilarity and similarity between p and q . Other approaches are possible; these are just the most common.

5.2.2 Distances Between Data Objects

The distance between two points (data objects), p and q , in two, three, or higher dimensional space is given by the following familiar formula for *Euclidean distance*:

$$\text{distance}(p, q) = \sqrt{\sum_{k=1}^n (p_k - q_k)^2}, \quad (5.1)$$

where n is the number of dimensions and p_k and q_k are, respectively, the k^{th} attributes (components) of p and q . We illustrate this formula with the points in Figure 5.9, Table 5.2, which gives the x and y coordinates of the points, and Table 5.3, which shows the *distance matrix*, i.e., the matrix that gives the pairwise distance between points.

Distances, such as the Euclidean distance, have some well known properties, which we now list. Let $d(x, y)$ be the distance (dissimilarity) between points (data objects), p and q .

- $d(p, q) \geq 0$ for all p and q and $d(p, q) = 0$ only if $p = q$. (Positivity)
- $d(p, q) = d(q, p)$ for all p and q . (Symmetry)
- $d(p, r) \leq d(p, q) + d(q, r)$ for all points p , q , and r . (Triangle Inequality)

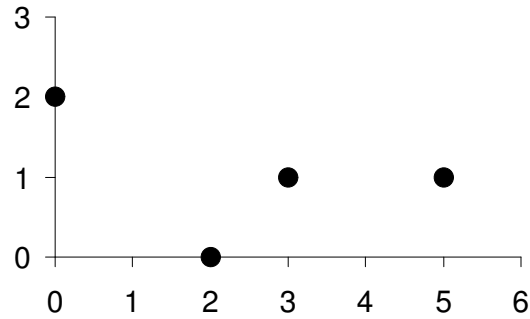


Figure 5.9. Four two-dimensional points.

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

Table 5.2. X-Y coordinates of four points.

	p1	p2	p3	p4
p1	0.000	2.828	3.162	5.099
p2	2.828	0.000	1.414	3.162
p3	3.162	1.414	0.000	2.000
p4	5.099	3.162	2.000	0.000

Table 5.3. Euclidean distance matrix for Table 5.2.

L_1	p1	p2	p3	p4
p1	0.000	4.000	4.000	6.000
p2	4.000	0.000	2.000	4.000
p3	4.000	2.000	0.000	2.000
p4	6.000	4.000	2.000	0.000

Table 5.4. L_1 distance matrix for Table 5.2.

L_∞	p1	p2	p3	p4
p1	0.000	2.000	3.000	5.000
p2	2.000	0.000	1.000	3.000
p3	3.000	1.000	0.000	2.000
p4	5.000	3.000	2.000	0.000

Table 5.5. L_∞ distance matrix for Table 5.2.

We make several comments. First, some people use the term ‘distance’ only for ‘dissimilarity’ measures that satisfy these properties, but that practice is often violated. (Mathematicians would say that a distance function that satisfies these properties is a ‘metric.’) Most often, a ‘non-distance’ (or non-metric) dissimilarity measure lacks only property 3. Second, property 2 implies that the distance matrix is symmetric. i.e., the ij^{th} entry is the same as the ji^{th} entry. For example, in Table 5.3, the fourth row of the first column and the fourth column of the first row both contain the entry, 5.099.

The Euclidean distance measure shown in 5.1 is generalized by the *Minkowski* distance metric shown in equation 5.2.

$$distance(p, q) = \left(\sum_{k=1}^n |p_k - q_k|^r \right)^{1/r}, \quad (5.2)$$

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 of this 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} norm, L_∞ norm) distance. This is the maximum difference between any attribute of the objects. More formally, the L_∞ distances is defined by equation 5.3.

$$distance(p, q) = \lim_{r \rightarrow \infty} \left(\sum_{k=1}^n |p_k - q_k|^r \right)^{1/r} \quad (5.3)$$

The r parameter should not be confused with the number of dimensions (attributes), n . For example, 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 5.4 and 5.5, respectively, give the proximity matrices for the L_1 and L_∞ distances using data from Table 5.2.

5.2.3 Similarities Between Data Objects

For similarities, the triangle inequality (or the analogous property) typically does not hold. Usually, however, symmetry and positivity do hold. To be explicit, if $s(p, q)$ is the similarity between points (data objects), p and q , then the typical properties of similarities are the following.

- $s(p, q) = 1$ only if $p = q$. ($0 \leq s \leq 1$)

- $s(p, q) = s(q, p)$ for all p and q . (Symmetry)

A similarity matrix is typically symmetric, although this is not always the case. As example consider an experiment in which people are asked to classify a small set of characters as they flash on a screen. Then the confusion matrix for this experiment records how often each character is classified as itself, or as another character. For instance, suppose that ‘0’ appeared 200 times and was classified as a ‘0’ 160 times, but as a ‘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 that is not symmetric. In such situations, the similarity measure is often made symmetric by setting $s'(p, q) = s'(q, p) = (s(p, q) + s(q, p))/2$, where s' indicates the new similarity measure.

Similarity Between Objects with Binary Attributes

There are many measures of similarity between objects that contain only binary attributes. These measures are referred to as *similarity coefficients*, and 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, but we will consider only a few of these coefficients. Let p and q 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:

M_{01} = the number of attributes where p was 0 and q was 1

M_{10} = the number of attributes where p was 1 and q was 0

M_{00} = the number of attributes where p was 0 and q was 0

M_{11} = the number of attributes where p was 1 and q was 1

One commonly used similarity coefficient is the simple matching coefficient, SMC , which is given by the following equation:

$$SMC = \frac{\text{number of matching attribute values}}{\text{number of attributes}} = \frac{M_{11} + M_{00}}{M_{01} + M_{10} + M_{11} + M_{00}} \quad (5.4)$$

This measure counts both presences and absences equally. For example, this measure might be appropriate to find students that had answered the questions similarly on a test that consisted only of true/false questions.

However, in some cases we might have asymmetric binary attributes. Suppose that p and q are data objects that represent two rows (two transactions) of a transaction matrix (see Chapter 2). Then, each binary attribute corresponds to a product: a 1 indicates that a product 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. Consequently, the *Jaccard* coefficient is frequently used to handle objects consisting of asymmetric binary attributes. The Jaccard coefficient, often symbolized by J , is defined in the following way.

$$J = \frac{\text{number of matching presences}}{\text{number of attributes not involved in 00 matches}} = \frac{M_{11}}{M_{01} + M_{10} + M_{11}} \quad (5.5)$$

To illustrate the difference between these two similarity measures, we calculate *SMC* and J for the following two binary vectors.

$$\begin{aligned} p &= (1, 0, 0, 0, 0, 0, 0, 0, 0, 0) \\ q &= (0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1) \end{aligned}$$

$$\begin{aligned} M_{01} &= 2 && \text{the number of attributes where } p \text{ was 0 and } q \text{ was 1} \\ M_{10} &= 1 && \text{the number of attributes where } p \text{ was 1 and } q \text{ was 0} \\ M_{00} &= 7 && \text{the number of attributes where } p \text{ was 0 and } q \text{ was 0} \\ M_{11} &= 0 && \text{the number of attributes where } p \text{ was 1 and } q \text{ was 1} \end{aligned}$$

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

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

Cosine Similarity

As mentioned in Chapter 2, 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 ignored and normalization is performed to account for different forms of the same word, differing document lengths, etc. (Again, see Chapter 2).

But even though the documents have thousands or tens of thousands of attributes (terms), each document is sparse, i.e., has relatively few non-zero attributes. (The normalizations used for documents do not create a non-zero where there was a zero, 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 00 matches are counted, most documents will be highly similar to most other documents. Consequently, we need a measure like the Jaccard measure, which ignores 00 matches, but which can handle non-binary vectors. The *cosine similarity*, defined below, is the most common measure of document similarity. If p and q are two document vectors, then

$$\text{cosine}(p, q) = \frac{p \bullet q}{\|p\| \|q\|}, \quad (5.6)$$

where \bullet indicates vector dot product, i.e., $p \bullet q = \sum_{k=1}^n p_k q_k$ and $\|p\|$ is the length of vector p , i.e., $\|p\| = \sqrt{\sum_{k=1}^n p_k^2} = \sqrt{p \bullet p}$.

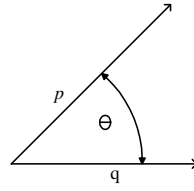


Figure 5.10. Mathematical illustration of the cosine measure.

To illustrate, we show how to calculate the cosine similarity for the following two data objects (binary vectors).

$$p = (3, 2, 0, 5, 0, 0, 0, 2, 0, 0)$$

$$q = (1, 0, 0, 0, 0, 0, 0, 1, 0, 2)$$

$$p \bullet q = 3 * 1 + 2 * 0 + 0 * 0 + 5 * 0 + 0 * 0 + 0 * 0 + 0 * 0 + 2 * 1 + 0 * 0 + 0 * 2 = 5$$

$$\|d1\| = \sqrt{3 * 3 + 2 * 2 + 0 * 0 + 5 * 5 + 0 * 0 + 0 * 0 + 0 * 0 + 2 * 2 + 0 * 0 + 0 * 0} = 6.480$$

$$\|d2\| = \sqrt{1 * 1 + 0 * 0 + 0 * 0 + 0 * 0 + 0 * 0 + 0 * 0 + 0 * 0 + 1 * 1 + 0 * 0 + 2 * 2} = 2.236$$

$$\cos(d1, d2) = .31$$

The definition of the cosine measure given in equation 5.6 may seem a bit mysterious at first. However, notice that since we divide by the norms of the vectors, we are normalizing both data objects (vectors) to have a length of 1. This means that cosine similarity does not take into account the magnitude of the two data objects when computing similarity. (If you want a proximity measure where magnitude is important, then a measure such as the Euclidean distance might be better.)

As indicated by Figure 5.10, cosine similarity really is a measure of the (cosine of the) angle between p and q . Thus if the cosine similarity is 1, the angle between p and q is 0° , and they are the same except for magnitude. If the cosine similarity is 0, then the angle between p and q is 90° , and they do not share any terms (words).

If we ‘normalize’ p and q to have unit length, i.e., $p' = p/\|p\|$ and $q' = q/\|q\|$, then $\cosine(p, q) = p' \bullet q'$, i.e., 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.

Finally, note that for cosine similarity to make sense all the attributes must be binary or continuous (interval or ratio) variables.

Tanimoto Coefficient (Extended Jaccard Coefficient)

We briefly mention that there is another similarity measure which can be used for document data and which reduces to the Jaccard coefficient in the case of binary

attributes. The Tanimoto coefficient, which we shall represent as T , is defined by the following equation.

$$T(p, q) = \frac{p \bullet q}{\|p\|^2 + \|q\|^2 - p \bullet q}, \quad (5.7)$$

Correlation

The correlation between two data objects that have binary or continuous (interval or ratio variables) variables is a measure of the linear relationship between the attributes of the objects. More precisely, *Pearson's correlation* coefficient between two data objects, p and q is defined as

$$\text{corr}(p, q) = \frac{\text{covariance}(p, q)}{\text{standard_deviation}(p) \text{ standard_deviation}(q)} \quad (5.8)$$

where

$$\begin{aligned} \text{covariance}(p, q) &= \frac{1}{n-1} \sum_{k=1}^n (p_k - m_p)(q_k - m_q) \\ \text{standard_deviation}(p) &= \sqrt{\frac{1}{n-1} \sum_{k=1}^n (p_k - m_p)^2}, \\ \text{standard_deviation}(q) &= \sqrt{\frac{1}{n-1} \sum_{k=1}^n (q_k - m_q)^2}, \end{aligned}$$

$$m_p = \frac{1}{n} \sum_{k=1}^n p_k \text{ is the mean of } p$$

$$m_q = \frac{1}{n} \sum_{k=1}^n q_k \text{ is the mean of } q$$

Correlation is always in the range -1 to 1. A correlation of 1 (-1) means that p and q have a perfect positive (negative) linear relationship, i.e., that $p = aq + b$, where a and b are constants. To illustrate this, the following two sets of values for p and q indicate cases where the correlation is, respectively, -1 and +1. In both cases the mean of p and q was chosen to be 0, for simplicity.

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

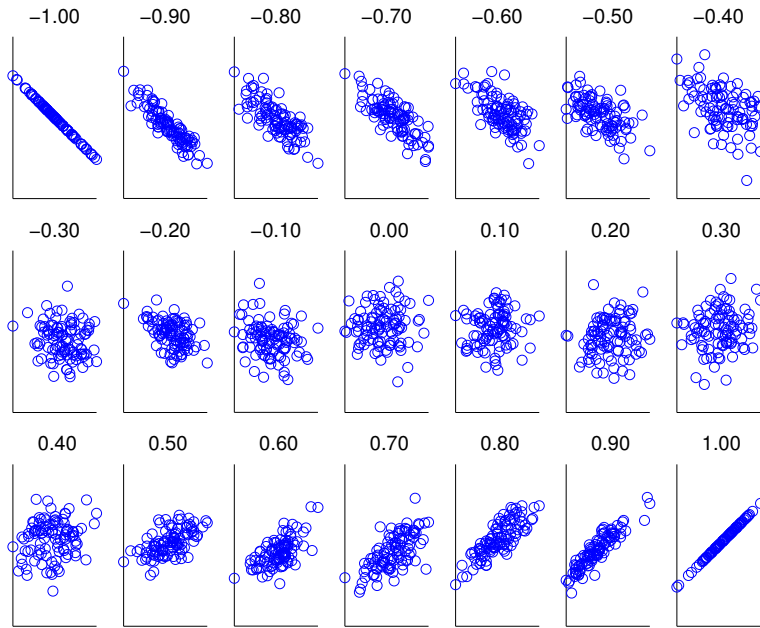


Figure 5.11. Scatter plots illustrating correlations from -1 to 1.

$$p = (3, 6, 0, 3, 6)$$

$$q = (1, 2, 0, 1, 2)$$

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, $q = p^2$, but their correlation is 0.

$$p = (-3, -2, -1, 0, 1, 2, 3)$$

$$q = (9, 4, 1, 0, 1, 4, 9)$$

While the correlation coefficient is a standard part of most statistics or mathematics packages, it is also easy to judge the correlation between two data objects p and q by plotting their attribute values. Figure 5.11 show what these plots look like in one specific case. p and q have 30 attributes and the values of these attributes are randomly generated (normal distribution) with a correlation that ranges from -1 to 1. Each circle in a plot represents one of the 30 attributes; its x-coordinate is the value of the attribute for p , while its y-coordinate is the value of the same attribute for q .

We remark that if we standardize, p and q , i.e.,
 $p' = (p - m_p)/\text{standard_deviation}(p)$ and $q' = (q - m_q)/\text{standard_deviation}(q)$,
then $\text{corr}(p, q) = p' \bullet q'$, i.e., the correlation coefficient can be calculated by taking the dot product.

5.2.4 Issues in Proximity Calculation

There are a number of issues that should be well-understood by users of proximity measures. Here we briefly consider (1) how to handle the case where attributes have different scales and/or are correlated, (2) how to calculate proximity between objects that are composed of different sorts of attributes, e.g., numerical and qualitative, (3) how to handle proximity calculation when attributes have different weights, i.e., when not all attributes should contribute equally to the proximity of objects, and (4) the impact of high dimensionality on proximity measures.

Standardization and Correlation for Distance Measures

An important issue with distances is whether all the attributes have the same range of values. (This situation is often described by saying that the variables have different ‘scales.’) For instance, in Chapter 2 we gave an example of trying to use the Euclidean distance to measure the dissimilarity between people based on two attributes: age and income. Without standardization, the distance between two people will be dominated by income.

Another issue, although one that requires more knowledge of statistics (see Appendix A), is whether there is correlation between some of the attributes. If so, then the straightforward application of distance formulas can also result in misleading distances. A generalization of Euclidean distance, the *Mahalanobis distance*, is useful when this is the case and when it is thought that the data may have a Gaussian (normal) distribution. The Mahalanobis distance between two objects (vectors) p and q is defined as follows:

$$\text{mahalanobis}(p, q) = (p - q)\Sigma^{-1}(p - q)^T \quad (5.9)$$

where Σ^{-1} is the inverse of the covariance matrix of the data.

In Figure 5.12, we see 1000 points, whose x and y attributes are highly correlated, e.g., 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, if the data set is small, 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. However, can anything be done if the attributes are of a wide variety of different types? The answer is, ‘yes,’ and we now describe one rather straightforward approach. The idea is to compute the similarity between each attribute separately using Table 5.1. However, we must choose a method that results in a similarity between 0 and 1, if we want all attributes to count equally,

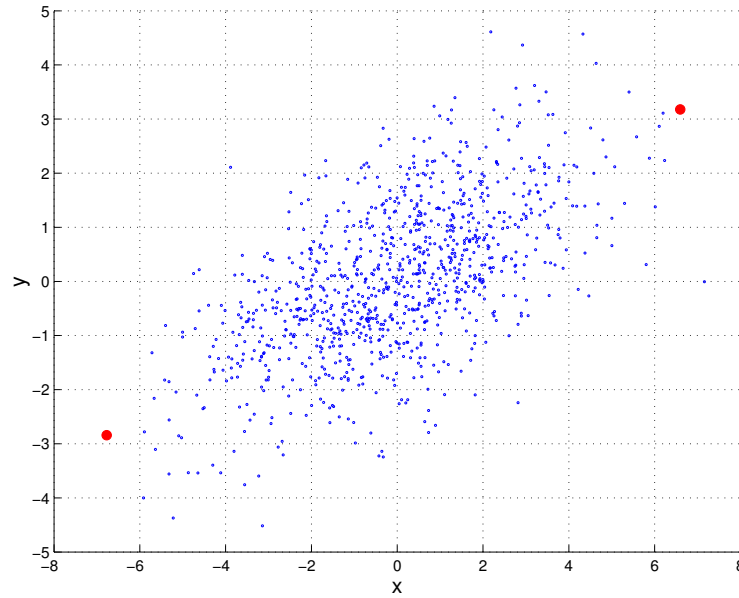


Figure 5.12. Mahalanobis distance.

and for now, we will assume that this is the case. At this point we might think that we could define the overall similarity as the average of all the similarities.

Unfortunately, while this is fine for most cases, this approach does not work well if some of the attributes are asymmetric binary attributes. To see this, consider that if all the attributes are asymmetric binary attributes, then this suggested similarity measure reduces to the simple matching coefficient, and we know that this measure is not appropriate for this situation. The easiest way to fix this problem is to omit asymmetric binary variables from the similarity calculation when their values are 0 for both of the objects whose similarity is being computed. This approach also works well for handling missing values.

In summary, a procedure for computing an overall similarity between two objects with different types of attributes is the following. (This procedure can be easily modified to work with dissimilarities.)

1. For the k^{th} attribute, compute a similarity, s_k , in the range $[0, 1]$.
2. Define an indicator variable, δ_k , for the k^{th} attribute as follows:

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

3. Compute the overall similarity between the two objects using the following

formula:

$$\text{similarity}(p, q) = \frac{\sum_{k=1}^n \delta_k s_k}{\sum_{k=1}^n \delta_k} \quad (5.10)$$

Using Weights

In the previous discussion of proximity measures, we have attempted to treat all attributes equally. However, sometimes, this is not desirable, e.g., the application objective dictates that some attributes are more important to the definition of similarity than others. In these situations, we can modify some of the formulas for distance or similarity by weighting the contribution of each attribute

If the weights, w_k sum to 1, then (5.10) becomes

$$\text{similarity}(p, q) = \frac{\sum_{k=1}^n w_k \delta_k s_k}{\sum_{k=1}^n \delta_k} \quad (5.11)$$

The definition of the Minkowski distance can also be modified.

$$\text{distance}(p, q) = \left(\sum_{k=1}^n w_k |p_k - q_k|^r \right)^{1/r}, \quad (5.12)$$

Impact of High Dimensionality on Proximity Measures

Discussion of Curse of Dimensionality and its impact on proximity, Shared Nearest Neighbor Distance

5.2.5 Selecting the ‘right’ proximity measure.

This section has described a number of proximity measures and has provided some information about the types of data sets for which they are appropriate. Nonetheless, the reader might still find it difficult to answer to the question, “What is the right proximity measure to use?”, for many of the data mining scenarios not covered here. Sometimes, a particular type of proximity measure is well accepted in a particular field, e.g., the cosine measure for computing the similarity of two documents, and thus, other data miners will have answered this question. Other times, the software package or clustering algorithm being used may drastically limit the choices.

However, if common practice or practical restrictions do not dictate a choice, then, in general, the proper choice of a proximity measure can be a time consuming task that requires careful consideration of domain knowledge and the purpose to which the measure is being put—often, but not always clustering. Frequently, a number of different similarity measures may need to be evaluated to see which ones produce results that make the most sense. For example, suppose that the goal is to cluster documents, and class labels are available for the documents. Then, similarity measures can be compared to see which one produces the ‘purest’ clusters. (Details of what constitutes a ‘pure’ cluster will be explained when we discuss cluster evaluation, later in this chapter—see Section 5.11.)

This discussion needs to be expanded.

5.3 Density

Euclidean density, grid-based approach, points-within-given-radius approach, multivariate density estimation and kernel density function approach, problems with density, especially in high dimensions, Shared Nearest Neighbor density

5.4 Characteristics of Clustering Algorithms

Clustering algorithms can be characterized in many ways. As preparation for the discussion of specific clustering algorithms that is to follow, we list some of the more important.

Type of clustering produced. The distinctions to be made here are the following: Partitional versus hierarchical, overlapping versus non-overlapping, fuzzy versus non-fuzzy, complete versus partial, and object versus attribute.

Type of clusters sought. For instance, are the clusters being sought well-separated, center-based, density-based, contiguity-based, concept/shared property-based, or defined via an objective function?

Handling of variations among clusters. A related issue is whether the clusters are relatively similar to one another, or whether they are of differing sizes, shapes and densities.

Handling of high dimensional data. An important characteristic of a clustering algorithm, is how well it handles high dimensional data. However, this in turn, depends on whether the proximity or density used by the clustering algorithm works well for high-dimensional data.

Dimensionality of the clusters found. Additionally, are the clusters found using the full set of attributes, or do they involve only subsets of the attributes, i.e., we are looking for clusters in a subspace of the entire space? Subspace clustering algorithms are discussed in Section 5.8.3.

Type of data the algorithm can handle. Some clustering algorithms make assumptions about the type of data they use. For example, the K-means algorithm (Section 5.6.1 algorithm) assumes that it is meaningful to take the mean (average) of a set of data objects. This makes sense for data that has continuous attributes and for document data, but not for record data that has categorical attributes. On the other hand, some clustering algorithms start from a proximity matrix and are less affected by the nature of the data.

How the Algorithm Handles Noise and Outliers regardless of whether a clustering algorithm works with the data matrix or the proximity matrix, its ability to handle noise and outliers often determines how useful the algorithm will be.

Type of similarity or density measure the algorithm uses. Since the proximity or density measure plays a key role in defining clusters, the type of proximity or density measure also needs careful consideration. For example, if the data has many asymmetric binary variables, a similarity measure such as the Jaccard coefficient or the cosine measure will often be used for clustering. As another example, if the data set consists of two- or three-dimensional point sets, then a grid-based approach can be used to define density.

Conceptual Approach of the Algorithm Many clustering algorithms have one or more concepts on which the algorithm is based. We mention some of the common concepts on which clustering algorithms are based.

Maximize an Objective Function In this case, the clustering problem then becomes an optimization problem, which, in theory, can be solved by enumerating all possible ways of dividing the points into clusters and evaluating the ‘goodness’ of each potential set of clusters by using the given objective function. Of course, this ‘exhaustive’ approach is computationally infeasible (NP hard) and as a result, a number of more practical techniques for optimizing a global objective function have been developed. One approach to optimizing a global objective function is to rely on algorithms that find solutions that are often good, but not optimal. An example of this approach is the K-means clustering (Section 5.6.1 algorithm), which tries to minimize the sum of the squared distances (error) between objects and their cluster centers. Still another approach is to forget about global objective functions. In particular, many hierarchical clustering techniques (Section 5.7) proceed by making local decisions at each step of the clustering process. These ‘local’ or ‘per-step’ decisions are also based on an objective function, but the overall or global result is not easily interpreted in terms of a global objective function.

Fit the Data to a Statistical Model Another approach is to fit the data to a model. Such models are specified by parameters which are determined from the data. An example of such techniques is mixture models (Section 5.9.2), which assume that the data is a ‘mixture’ of a number of statistical distributions. These clustering algorithms seek to find a solution to a clustering problem by finding the maximum likelihood estimates for the statistical parameters that describe the different distributions (clusters).

Transform the Clustering Problem to Another Domain Some clustering algorithms operate by mapping the clustering problem to a different domain, and solving a related problem in that domain. For instance, a proximity matrix defines a weighted graph, where the nodes are the points being clustered, and the weighted edges represent the proximities between points, i.e., the entries of the proximity matrix. Thus, clustering is equivalent to breaking the graph into connected components, one

for each cluster. Also, some clustering approaches work directly on the data matrix using linear algebra techniques such as Principal Components Analysis or Singular Value Decomposition (SVD). These approaches can identify the strongest Approaches based on neural nets map the clustering problem into a problem of finding a function or surface that best fits the given data. Yet other examples of other approaches involve viewing the data as a physical system, e.g., a collection of oscillators, or, in the case of two dimensional data, treating the data as an image.

5.5 Roadmap for Discussing Specific Clustering Techniques

While some approaches have been proposed for organizing clustering algorithms into groups, all of these approaches are somewhat arbitrary since, as the previous discussion indicates, there are many different ways in which clustering algorithms can be similar and different. However, in the upcoming discussion of specific clustering algorithms, we shall use the following groupings and order.

1. Center-based Clustering.
2. Hierarchical Clustering
3. Density-Based Clustering (including Subspace Clustering)
4. Graph-based Clustering
5. Selected Clustering Techniques

In the section on *Selected Clustering Techniques*, we will cover a variety of clustering techniques that were not covered in the preceding sections and which illustrate some important issues, e.g., clustering based on mixture models, fuzzy clustering, and scalable clustering approaches for handling large data sets.

5.6 Center-Based Clustering Techniques

As described earlier, partitional clustering techniques create a one-level partitioning of the data objects. There are a number of such techniques, but we shall describe only two center-based clustering approaches in this section: K-means and K-medoid. Both of these techniques are based on the idea that a center point can represent a cluster. (In the following discussion, we will use the terms ‘point’ or ‘data point’ to refer to a data object.) For K-means, we use the notion of a centroid, which is the mean or median point of a group of points. For K-medoid we use the notion of a medoid, which is the most representative (central) point of a group of points. While a centroid almost never corresponds to an actual data point, a medoid, by its definition, must be an actual data point.

5.6.1 K-means

Basic Algorithm

The K-means clustering technique is very simple and we immediately begin with a description of the basic algorithm. Note that K is a user specified parameter, i.e., the number of clusters desired.

Algorithm 1 Basic K-means Algorithm.

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

The initial centroids, or *seeds*, as they are often called, are typically chosen randomly from the set of all data points. Because of this, the set of clusters produced by the K-means algorithm can vary from one run to another. This raises the issue of how to evaluate which clustering is best, a topic that will be discussed shortly.

The proximity measure used to evaluate closeness varies, depending on the type of data. For instance, if we have points in two or three dimensional space, Euclidean distance is appropriate, while for documents, the cosine measure is typically used. While any proximity measure could potentially be employed, termination of the algorithm may not be guaranteed, as it is with Euclidean distance and the cosine measure.

The centroid associated with a cluster is typically the mean (average) of the points in the cluster. (This is the reason the algorithm is called K-means.) In particular, if C_i is the i^{th} cluster, $|C_i|$ is the number of points in the i^{th} cluster, x is a point in C_i , and x_j is the j^{th} attribute of that point, then m^i , the mean of the i^{th} cluster, is defined by $m_j^i = \frac{1}{|C_i|} \sum_{x \in C_i} x_j$. In other words, the mean of a set of points is obtained by taking the mean of each attribute for the points in the cluster.

In the absence of numerical problems, K-means always *converges* to a solution, i.e., reaches a state where no points are shifting from one cluster to another, and hence, the centroids don't change. However, since most of the convergence occurs in the early steps, the condition on line 5 is often replaced by a weaker condition, e.g., repeat until only 1% of the points change clusters.

Time and Space Complexity

Since only the data points and centroids are stored, the space requirements are basically $O((m + K)n)$, where m is the number of points and n is the number of attributes. The time requirements are $O(I * K * m * n)$, 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. Thus,

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 .

Evaluating K-means Clusters

Since the clusters produced by K-means may vary from run to run, some method is needed to select one set of clusters over another. If the centroid is taken to be the mean, and our proximity measure is Euclidean distance, then the quality of a clustering is measured by the total sum of the squared error (SSE).

$$\text{SSE} = \sum_{i=1}^K \sum_{x \in C_i} \sum_{j=1}^n (m_j^i - x_j)^2 \quad (5.13)$$

In other words, for each data point, we calculate its error, i.e., its Euclidean distance to the closest centroid, and then, as a measure of the overall goodness of the clustering, we calculate the total sum of the squared errors (SSE). Given two different sets of clusters, which are produced by two different runs of K-means, we prefer the one with the smallest squared error.

While the total SSE provides an overall measure of the quality of the clustering, the SSE contributed by each cluster is also of interest. (If there is any potential confusion about which SSE we are talking about, total or cluster, we will use the terminology, total SSE and cluster SSE, respectively.) For instance, if two clusters have roughly the same number of points, but one has a larger SSE, then that cluster is more diffuse, i.e., the points in the cluster are farther away from the cluster centroid on average. K-means can be regarded as a method which attempts to approximate or represent a set of points by a much smaller set of points, i.e., the centroids, and the cluster SSE measures how good this approximation is, on average, for the points in a cluster, while the total SSE measures the overall goodness of this approximation.

In Section 5.6.1, we will see how the K-means algorithm can be derived from the objective of minimizing SSE. Indeed, steps 3 and 4 of the K-means algorithm directly attempt to minimize SSE. In step 3, we form clusters by assigning points to the nearest centroid. Obviously, this results in the minimum SSE for a given set of centroids. In step 4, we recompute the centroids, replacing each cluster centroid with the centroid that is the mean of that cluster. As is shown in Section 5.6.1, this approach minimizes the squared error, i.e., the mean of a set of points is the best choice for a centroid to minimize a cluster's SSE. However, although the K-means algorithm is attempting to minimize squared error in steps 3 and 4, only a local minimum is guaranteed.

We illustrate the fact that different runs of K-means produce different total SSEs with the example data set presented in Figure 5.13. In particular, Figure 5.13a shows a set of two-dimensional points which has three 'natural' groupings of points. Figure 5.13b shows a clustering solution that represents a global minimum with respect to SSE for three clusters, while 5.13c shows a suboptimal clustering which represents a local minimum.

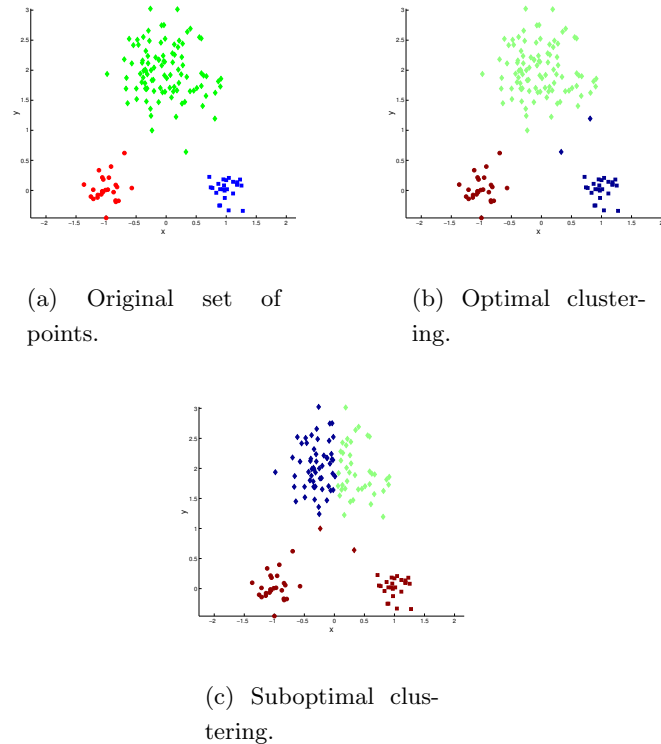


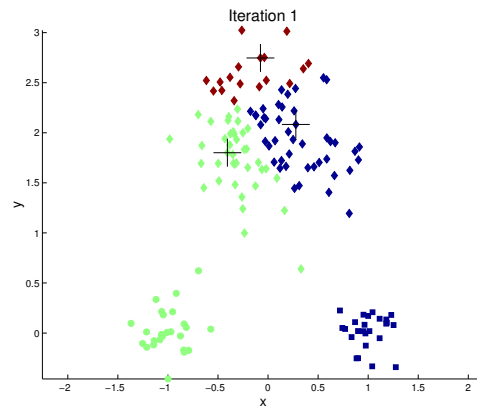
Figure 5.13. Finding three clusters in a set of 2D points.

An obvious way to reduce SSE is to find more clusters, i.e., use a larger K . While a ‘poor’ K-means clustering with more clusters can result in a higher SSE than a ‘good’ K-means clustering with fewer clusters, the best possible SSE always declines as K increases, provided K is less than the number of points.

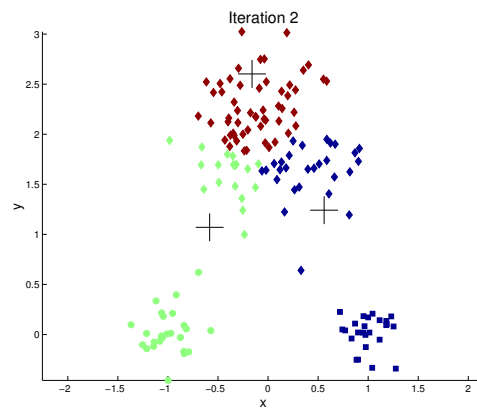
Choosing initial centroids

Choosing the proper initial centroids is the key step of the basic K-means procedure. It is easy and efficient to choose initial centroids randomly, but the results are often poor. We illustrate this with a simple example that uses the points previously shown in Figure 5.13. Figures 5.14 and 5.15 indicate the clusters that result for two particular choices of initial points. In the first case, even though all the initial centroids come from one ‘natural’ cluster, the minimum SSE clustering was still found. In the second case, however, even though the initial centroids seem ‘better distributed,’ a suboptimal clustering, i.e., higher squared error, results. For both figures, the positions of the cluster centroids in the various iterations are indicated by large crosses.

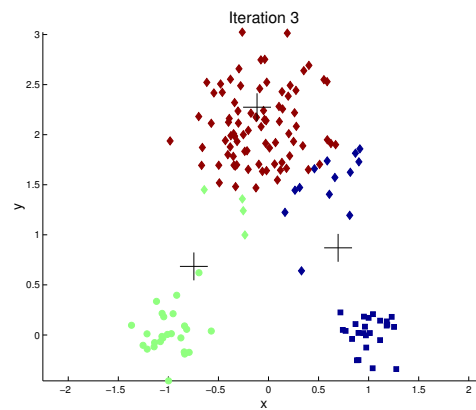
One technique that is commonly used to address the problem of choosing initial centroids is to perform multiple runs, each with a different set of randomly chosen initial centroids, and then select the set of clusters with the minimum SSE. However,



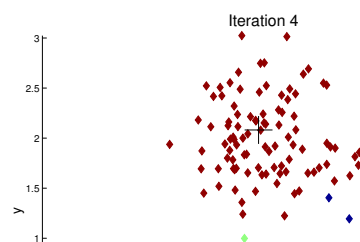
(a)

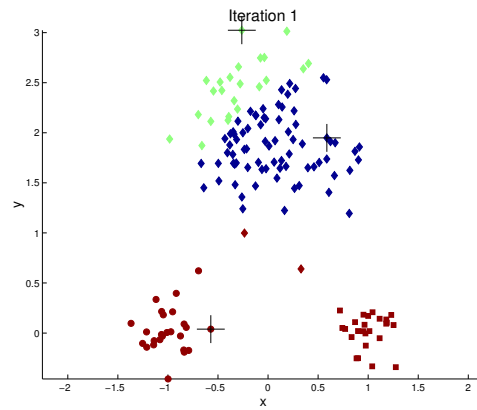


(b)

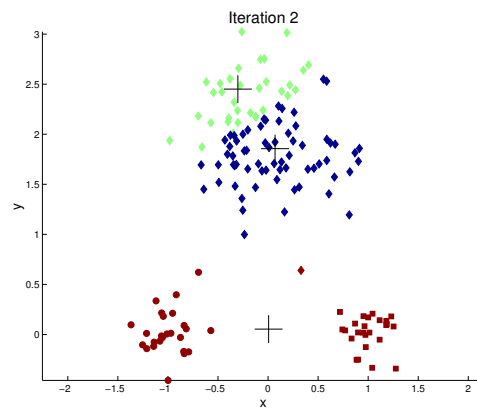


(c)

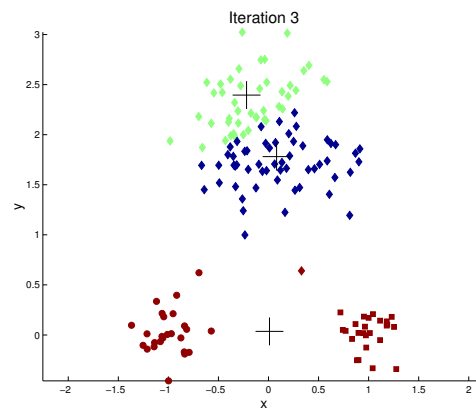




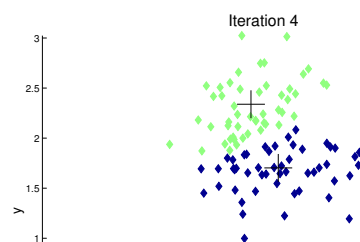
(a)



(b)



(c)



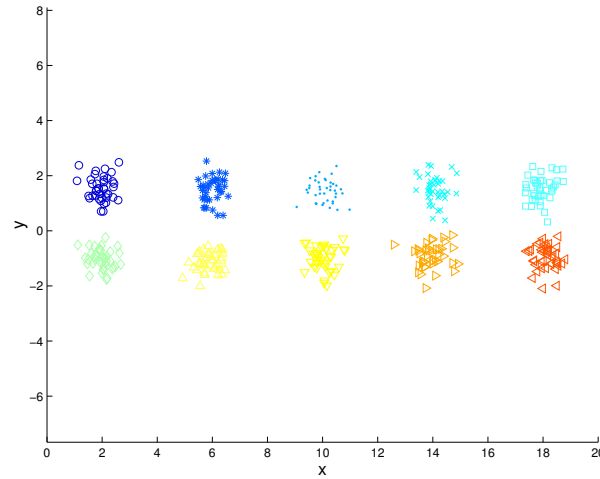


Figure 5.16. Five pairs of clusters.

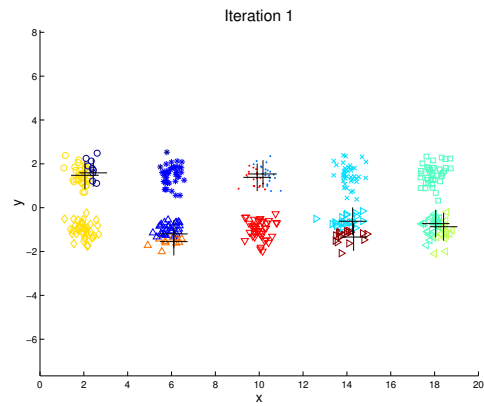
this strategy may not work very well, depending on the data set and the number of clusters sought. We constructed an example, shown in Figure 5.16, to show what can go wrong. The data consists of 5 pairs of clusters, where the clusters in a pair of clusters are closer to each other than to any other cluster. The probability that an initial centroid will come from any given cluster is 0.10, 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 usually a very good starting situation.) In general, if there are K clusters and each cluster has n points, then the probability, P , of selecting one initial centroid from each cluster is given by the following equation. (This assumes sampling with replacement.)

$$P = \frac{\text{number of ways to select one centroid from each cluster}}{\text{number of ways to select } K \text{ centroids}} = \frac{K!n^K}{(Kn)^K} = \frac{K!}{K^K} \quad (5.14)$$

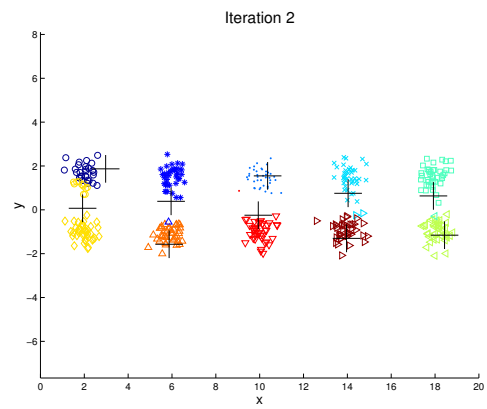
From this formula we can calculate that the chance of having one initial centroid from each cluster is $10!/10^{10} = 0.00036$.

Despite this, 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, it is relatively likely that at least one pair of clusters will have only one initial centroid. In this latter case, because the pairs of clusters are far apart, the K-means algorithm will not redistribute the centroids between pairs of clusters, and thus, only a local minimum will be achieved.

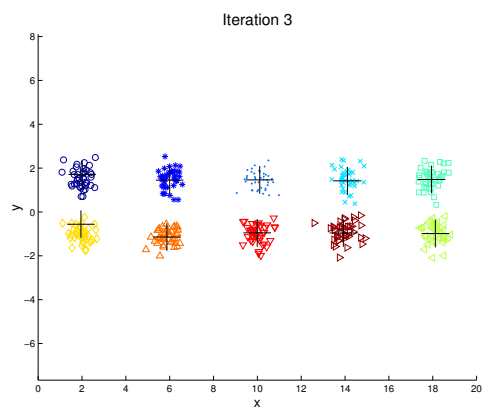
Figure 5.17 shows that if we start two initial centroids per pair of clusters, then, even when the both initial centroids are in a single cluster, the centroids will redistribute themselves so that the ‘true’ clusters are found. However, Figure 5.18 shows that if a pair of clusters has only one initial centroid, and another pair has three, then two of the ‘true’ clusters will be combined and one cluster will be split.



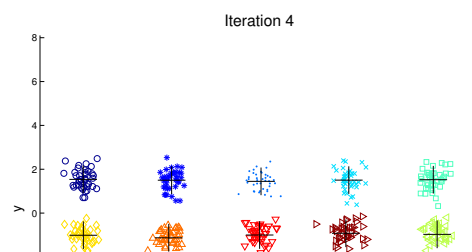
(a)

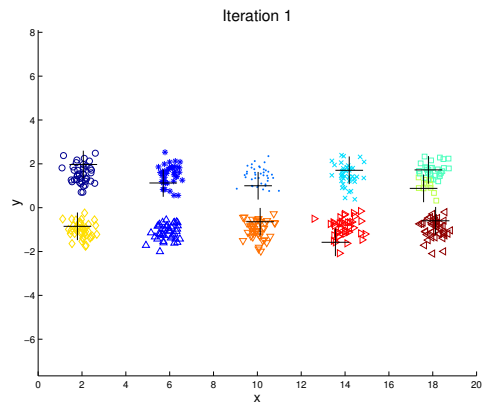


(b)

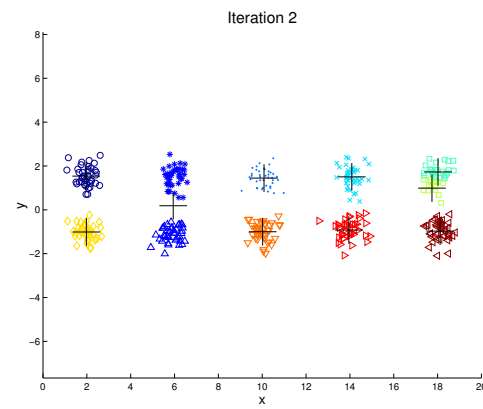


(c)

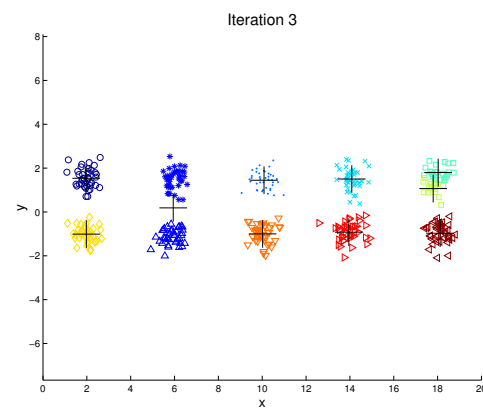




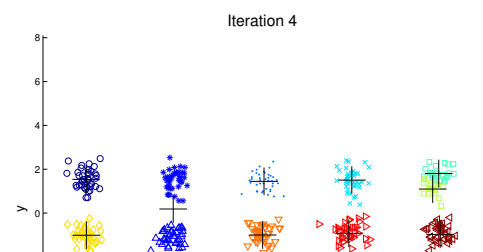
(a)



(b)



(c)



Because of the problems with using random seeds for initialization, problems that even repeated runs may not overcome, other techniques are often used for finding the initial centroids. One effective technique is to take a sample of points and cluster it 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 the sample is relatively small, e.g., a few hundred to a few thousand (hierarchical clustering is expensive), and if K is relatively small compared to the sample size.

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 is less susceptible to initialization problems, e.g., bisecting K-means, and using post-processing to ‘fix-up’ the set of clusters produced.

Handling Empty Clusters

One of the problems with the basic K-means algorithm given above is that empty clusters can be obtained if no points are assigned to a cluster during the assignment step. If so, then a strategy is needed to choose a replacement centroid. Otherwise, the squared error will certainly be larger than necessary.

One approach is to choose the point that is farthest away from any current center as the new centroid. If nothing else, this eliminates the point that currently contributes the most to the squared error. Another approach is to choose a replacement centroid from the cluster that has the highest SSE. (This will typically split the cluster.) If there are several empty clusters, then this process can be repeated several times. Like the previous approach, this approach is also motivated by the goal of reducing the SSE.

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. Incremental update guarantees that empty clusters are not produced since all clusters start with a single point, and if a cluster ever gets down to 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, i.e., the centroid becomes a weighted average of points, and the current point may have a weight greater than or less than 1. While this can result in better accuracy and faster convergence, it may 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 nets.

On the negative side, updating centroids incrementally introduces an order dependency problem, although this can be addressed by randomizing the order in which the points are processed. However, this randomization is not feasible unless the points are in main memory. Note that the basic K-means approach of updating the centroids after all points are assigned to clusters does not create an order

dependency problem.

Another negative aspect of incremental updates is that they are more expensive. However, K-means tends to converge rather quickly and so the number of points switching clusters will tend to be small after a few passes over all the points.

Pre- and Post-Processing

Sometimes, the quality of the clusters that can be found is improved by pre-processing the data. For example, when we use the squared error criteria, outliers can unduly influence the clusters that are found. Thus, it is common to try to find these outliers and eliminate them with a preprocessing step. However, it is important to realize that there are certain clustering applications where outliers should not be eliminated. In data compression every point must be clustered, and in some cases, e.g., financial analysis, apparent outliers can be the most interesting points, e.g., unusually profitable customers.

Another common approach uses post-processing to try to fix up the clusters that have been found. For example, small clusters are often eliminated since they frequently represent groups of outliers. Also, two small clusters that have relatively low SSE and are close together can be merged, while a cluster with relatively high SSE can be split into smaller clusters.

ISODATA is a version of K-means that uses the previously discussed postprocessing techniques to produce high quality clusters. ISODATA first chooses initial centroids in a manner that guarantees that they will be well separated. More specifically, the first initial centroid is the mean of all data points, while each succeeding centroid is selected by considering each point in turn and selecting any point that is at least a distance, d (a user-specified parameter), from any initial centroid selected so far. ISODATA then repeatedly a) finds a set of clusters using the basic K-means algorithm and b) applies postprocessing techniques. Execution halts when a stopping criterion is satisfied. Note that the K-means algorithm is run after postprocessing because the clusters that result from postprocessing do not represent a local minimum with respect to SSE. The ISODATA algorithm is shown in more detail below. The reader should note that ISODATA has a large number of parameters.

The splitting phase of ISODATA is an example of strategies that decrease the total SSE by increasing the number of clusters. Two general strategies are given below:

Split a cluster. Often the cluster with the largest SSE is chosen, but in the case of ISODATA the cluster with the largest standard deviation for one particular attribute is chosen.

Introduce a new cluster centroid. Often the point that is farthest from any cluster center is chosen.

Likewise, the merging phase of ISODATA is just one example of strategies that

Algorithm 2 ISODATA.

```

1: Select  $K$  well-separated points as the initial centroids.
2: repeat
3:   repeat
4:     Form  $K$  clusters by assigning all points to the closest centroid.
5:     Recompute the centroid of each cluster.
6:   until The centroids don't change
7:   Discard clusters with 'too few' points.
8:   if There are 'too many' clusters or it is a 'splitting' phase then
9:     Split clusters that are 'too loose'
10:  end if
11:  if There are 'too few' clusters or it is a 'merging' phase then
12:    Merge pairs of clusters that are 'close'
13:  end if
14:  if The current phase is a 'merging' phase then
15:    Set the phase for the next iteration to be a 'splitting' phase
16:  else
17:    Set the phase for the next iteration to be a 'merging' phase
18:  end if
19: until Results are 'acceptable' or a pre-determined number of iterations has been
    exceeded
  
```

decrease the number of clusters. (This typically increases SSE.) Two general strategies are given below:

Disperse a cluster. In other words, remove the centroid that corresponds to the cluster and reassign the points to other clusters. Ideally the cluster which is dispersed should be the one that increases the total SSE the least.

Merge two clusters In ISODATA, the clusters with the closest centroids are 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, respectively, the same strategies that are used in the hierarchical clustering techniques known as the Centroid method and Ward's method. Both methods are discussed later in Section 5.7.

Finally, note that while pre- and post-processing techniques can be effective, they are based on heuristics that may not always work. Furthermore, they often require that the user choose values for a number of parameters.

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, and split it, and so on, until K clusters have been produced. In detail, bisecting K-means works as follows.

Algorithm 3 Bisecting K-means Algorithm.

- 1: Initialize the list of clusters to contain the cluster containing all points.
 - 2: **repeat**
 - 3: Select a cluster from the list of clusters
 - 4: **for** $i = 1$ to *number_of_iterations* **do**
 - 5: Bisect the selected cluster using basic K-means
 - 6: **end for**
 - 7: Add the two clusters from the bisection with the lowest SSE to the list of clusters.
 - 8: **until** Until the list of clusters contains K clusters
-

There are a number of different ways to choose which cluster is split. For example, we can choose the largest cluster at each step, the one with the least overall similarity, or use a criterion based on both size and overall similarity.

Bisecting K-means tends to be less susceptible to initialization problems. Partly this is because bisecting K-means performs several trial bisections and takes the best one in terms of SSE, and partly this is because there are only two centroids at each step.

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

To illustrate both the operation of bisecting K-means and to illustrate how it is less susceptible to initialization problems, we show, in Figure 5.19, how bisecting K-means finds 10 clusters in the data set originally shown in Figure 5.16.

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

Limitations and problems

K-means attempts to minimize the squared error of points with respect to their cluster centroids. While this is sometimes a reasonable criterion and leads to a simple algorithm, K-means and its variations have a number of limitations. In particular, K-means has difficulty detecting the ‘natural’ clusters, when clusters have widely different sizes, densities, or non-spherical shapes. This is illustrated by Figures 5.20, 5.21, and 5.22.

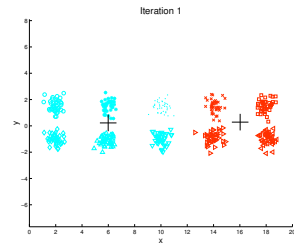
The difficulty in these three situations is that the K-means objective function is a mismatch for the kind of clusters we are trying to find since the K-means objective function is minimized by globular clusters of equal size and density or by clusters that are well separated. However, these limitations can be overcome, at least in some sense, if the user is willing to partition the data into a larger number of small clusters. Figure 5.23 shows what happens for the three previous data sets if we find 10 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.

Another limitation of the K-means algorithm is that it is restricted to data for which means (or medians) make sense. A related technique, K-medoid clustering, does not have this restriction and is discussed in Section 5.6.2.

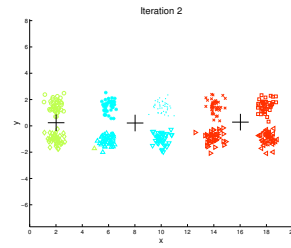
Finally, notice that bisecting K-means, while it typically works better than regular K-means, is no better at handling the previously illustrated problems with clusters of different sizes, shapes and densities. Also, if bisecting K-means splits a cluster during the clustering process, it may stay split, even after the final refinement step.

K-means as an Optimization Problem

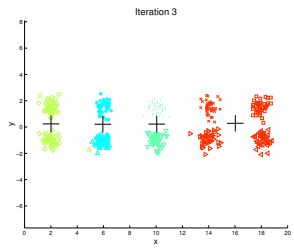
The K-means algorithm is derived by asking how we can partition a collection of data points into K clusters such that the total SSE is minimized. In mathematical terms we seek to minimize equation 5.13, which we repeat here.



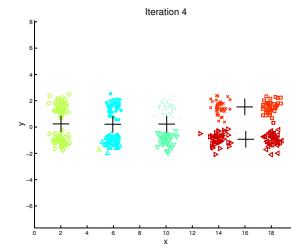
(a)



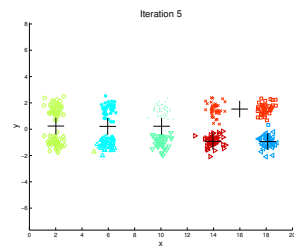
(b)



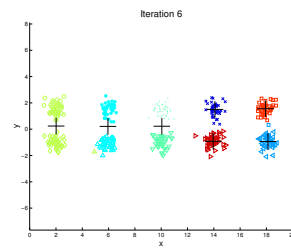
(c)



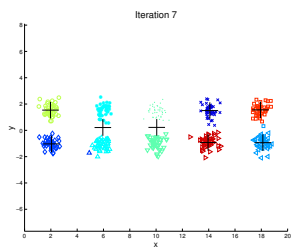
(d)



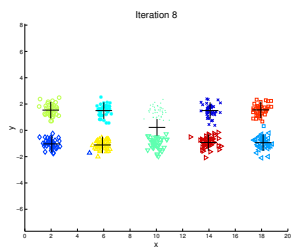
(e)



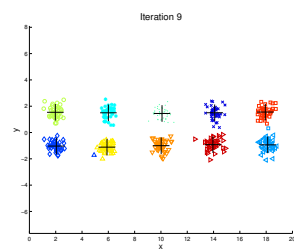
(f)



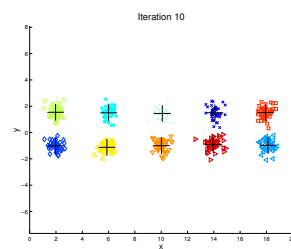
(g)



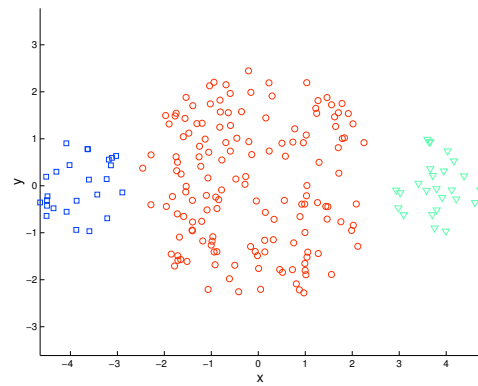
(h)



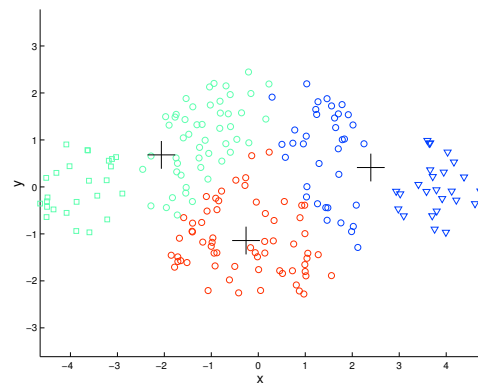
(i)



(j)

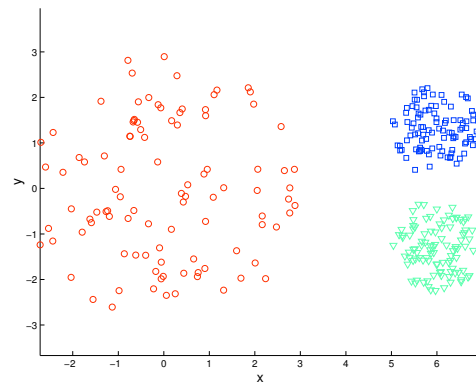


(a) Original Points

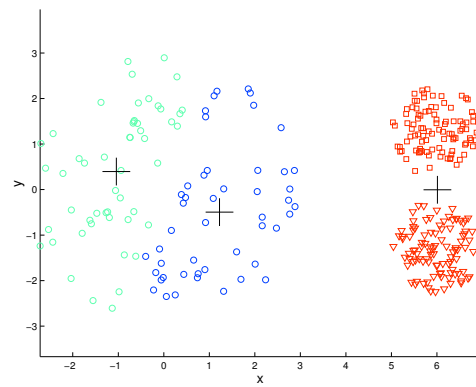


(b) Three K-means Clusters

Figure 5.20. K-means with different size clusters.

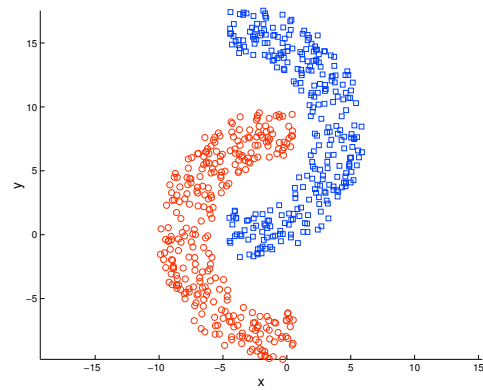


(a) Original Points

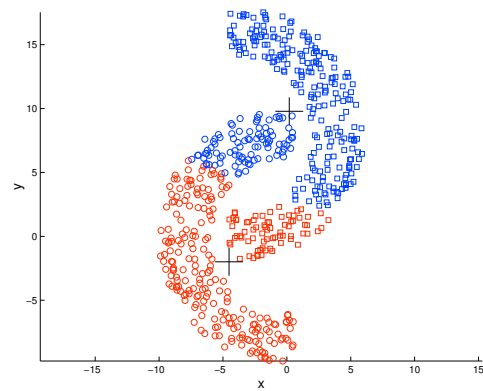


(b) Three K-means Clusters

Figure 5.21. K-means with different density clusters.

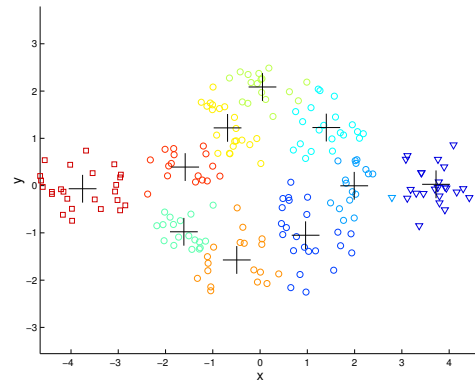


(a) Original Points

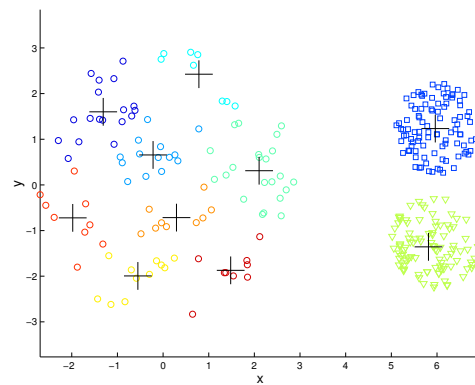


(b) Two K-means Clusters

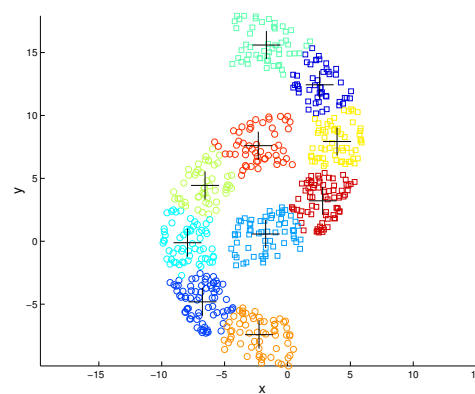
Figure 5.22. K-means with non-globular clusters.



(a) Unequal Sizes



(b) Unequal Densities



(c) Non-spherical Shapes

Figure 5.23. Using K-means to Find Many Clusters.

$$\text{SSE} = \sum_{i=1}^K \sum_{x \in C_i} \sum_{j=1}^n (m_j^i - x_j)^2 \quad (5.15)$$

Where C_i is the i^{th} cluster, $|C_i|$ is the number of points in the i^{th} cluster, x is a point in C_i , and x_j is the j^{th} attribute of that point, and m^i is the mean of the i^{th} cluster.

We can solve for the p^{th} component, m_k^p , $1 \leq k \leq n$, of the p^{th} centroid, m^p , by differentiating the SSE , setting it to 0, and solving, as indicated below.

$$\begin{aligned} \frac{\partial}{\partial m_k^p} \text{SSE} &= \frac{\partial}{\partial m_k^p} \sum_{i=1}^K \sum_{x \in C_i} \sum_{j=1}^n (m_j^i - x_j)^2 \\ &= \sum_{i=1}^K \sum_{x \in C_i} \sum_{j=1}^n \frac{\partial}{\partial m_k^p} (m_j^i - x_j)^2 \\ &= \sum_{x \in C_p} 2 * (m_k^p - x_k) = 0 \end{aligned}$$

$$\sum_{x \in C_p} 2 * (m_k^p - x_k) = 0 \Rightarrow |C_p| m_k^p = \sum_{x \in C_p} x_k \Rightarrow m_k^p = \frac{1}{|C_p|} \sum_{x \in C_p} x_k$$

Thus, we find that $m^p = \frac{1}{|C_p|} \sum_{x \in C_p} x$, the mean of the points in the cluster. This is pleasing since it agrees with our intuition of what the center of a cluster should be.

However, we can instead ask how we can obtain a partition of the data into K clusters such that the sum of the distances (not squared distance) of points from the centroid of their clusters is minimized. In mathematical terms, we now seek to minimize the sum of the absolute error, SAE , as shown in the following equation.

$$\text{SAE} = \sum_{i=1}^K \sum_{x \in C_i} \sqrt{\sum_{j=1}^n (m_j^i - x_j)^2} \quad (5.16)$$

This problem has been extensively studied in operations research under the name of the multi-source Weber problem and has applications to the location of warehouses and other facilities that need to be centrally located. We leave it as an exercise to the reader to verify that the centroid in this case is still the mean.

Finally, we can instead ask how to obtain a partition of the data into K clusters such that the sum of the L_1 distances of points from the ‘center’ of their clusters is minimized. We are seeking to minimize the the sum of the L_1 absolute errors, SAE_1 , as given by the following equation.

$$\text{SAE}_1 = \sum_{i=1}^K \sum_{x \in C_i} \sum_{j=1}^n |m_j^i - x_j| \quad (5.17)$$

We can solve for the p^{th} component, m_k^p , $1 \leq k \leq n$, of the p^{th} centroid, m^p , by differentiating the SAE_1 , setting it to 0, and solving, as indicated below.

$$\begin{aligned}
 \frac{\partial}{\partial m_k^p} SAE_1 &= \frac{\partial}{\partial m_k^p} \sum_{i=1}^K \sum_{x \in C_i} \sum_{j=1}^n |m_j^i - x_j| \\
 &= \sum_{i=1}^K \sum_{x \in C_i} \sum_{j=1}^n \frac{\partial}{\partial m_k^p} |m_j^i - x_j| \\
 &= \sum_{x \in C_p} \frac{\partial}{\partial m_k^p} |m_k^i - x_k| = 0
 \end{aligned}$$

$$\sum_{x \in C_p} \frac{\partial}{\partial m_k^p} |m_k^i - x_k| = 0 \Rightarrow \sum_{x \in C_p} \text{sign}(x_k - m_k^p) = 0$$

If we solve for m^p , we find that $m^p = \text{median}\{x \in C_p\}$, the point whose coordinates are the median values of the corresponding coordinate values of the points in the cluster. While the median of a group of points is straightforward to compute, this computation is not as efficient as the calculation of a mean.

In the first case we say we are attempting to minimize the within cluster squared error, while in the second case and third cases we just say that we are attempting to minimize the absolute within cluster error, where the error is measured either by the L_1 or L_2 distance.

5.6.2 K-medoid Clustering

As mentioned previously, K-means can only be used with data where means (or medians) are meaningful. If we have data for which that is not the case, e.g., strings, then other techniques must be used. In particular, the K-medoid clustering algorithm overcomes the limitations of K-means by trying to find a non-overlapping set of clusters such that each cluster has a most representative point, or medoid, i.e., a point that is a data point and is the most centrally located such point with respect to some proximity measure. In other words, if we compute the average similarity (distance) of each point in a cluster to every other point in that cluster, then the medoid is the point whose average similarity (average distance) is highest (lowest). (Equivalently, we can use the sum of proximities, instead of the average.) For a particular set of medoids, the overall ‘goodness’ of such a clustering is measured by summing the proximity of each point to its closest medoid.

The K-medoid algorithm is detailed below. (We use similarities as our proximities.) For this algorithm, the goal is to find the set of K medoids that maximize the sum of the similarities (SS) of each point to its closest medoid. As with K-means, K is a user supplied parameter that specifies the number of clusters desired.

Algorithm 4 Basic K-medoid Algorithm.

```

1: Initialization: Select K initial medoids.
2: Compute SS, the sum of the similarities of each point to its most similar medoid.
3: repeat
4:   for each pair  $(p, q)$ , where  $p$  is a medoid and  $q$  is a non-medoid do
5:     Swap  $p$  and  $q$ , i.e., make  $q$  a medoid and  $p$  a non-medoid.
6:     Calculate  $SS_{pq}$ , the new sum of similarities associated with the swap of  $p$ 
       and  $q$ .
7:   end for
8:   if the highest  $SS_{pq}$  is greater than the current SS then
9:     Swap  $p$  and  $q$  using the  $p$  and  $q$  that make  $SS_{pq}$  the highest.
10:  end if
11: until No swaps occur or an iteration limit is exceeded

```

Initial medoids are chosen in a simple way. (Again we illustrate using similarities.) The first initial medoid is chosen by finding the point that has the highest sum of similarities to all other points. Each of the remaining initial medoids is chosen by selecting the non-medoid point which has the highest sum of similarities to all non-medoid points, *excluding* those points that are more similar to one of the currently chosen initial medoids.

While the K-medoid algorithm is relatively simple, it is expensive compared to K-means. More recent improvements of the K-medoids algorithm have better efficiency than the basic algorithm, but are still relatively expensive and will not be discussed here. Relevant references may be found in the bibliographic notes.

5.7 Hierarchical Clustering

A hierarchical clustering algorithm is any algorithm that produces a hierarchical clustering as defined in Section 5.1.3. More specifically, the goal of such algorithms is to produce a sequence of nested clusters, ranging from singleton clusters of individual points to an all-inclusive cluster. As mentioned in Section 5.1.3, this hierarchy of clusters is often graphically represented by a dendrogram as illustrated by figures 5.2 and 5.3. A dendrogram captures the process by which a hierarchical clustering is generated by showing the order in which clusters are merged (bottom-up view) or clusters are split (top-down view).

One of the attractions of hierarchical techniques is that they correspond to taxonomies that are very common in the biological sciences, e.g., kingdom, phylum, genus, species, (Some cluster analysis work occurs under the name of ‘mathematical taxonomy.’) Another attractive feature is that hierarchical techniques do not assume any particular number of clusters. Instead, any desired number of clusters can be obtained by ‘cutting’ the dendrogram at the proper level. Also, hierarchical

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 5.6. 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 5.7. Distance Matrix for Six Points

techniques are sometimes thought to produce better quality clusters.

5.7.1 Agglomeration and Division

There are two basic approaches to 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 the notion of cluster proximity. Agglomerative techniques are most popular, and most of this section will be spent describing them.

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.

Sample Data

In the examples that follow we shall use the following data, which consists of six, two-dimensional points, to illustrate the behavior of the various hierarchical clustering algorithms. The x and y coordinates of the points and the distances between them are shown, respectively, in tables 5.6 and 5.7. The points themselves are shown in Figure 5.24.

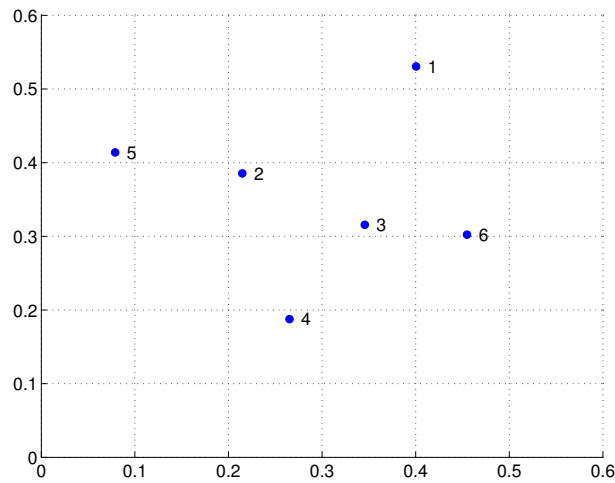


Figure 5.24. Set of Six Two-dimensional Points.

5.7.2 Divisive Algorithms

As mentioned, divisive techniques are less common. We have already seen an example of this type of technique, bisecting K-means, which was described in Section 5.6.1. Another simple hierarchical divisive technique, which we shall refer to as *MST*, starts with the minimum spanning of the proximity graph.

Conceptually, the minimum spanning tree of the proximity graph is built by starting with a tree that consists of any point. In successive steps, we look for the closest pair of points, p and q , such that one point, p , is in the current tree and one, q , is not. We add q to the tree and put an edge between p and q . Figure 5.25 shows the MST for the points in Figure 5.24.

The MST divisive hierarchical algorithm is shown below. This approach is the divisive version of the ‘single link’ agglomerative technique that we will see shortly. Indeed, the hierarchical clustering produced by MST is the same as that produced by single link. See Figure 5.27.

Algorithm 5 MST Divisive Hierarchical Clustering Algorithm

- 1: Compute a minimum spanning tree for the proximity graph.
 - 2: **repeat**
 - 3: Create a new cluster by breaking the link corresponding to the largest distance (smallest similarity).
 - 4: **until** Only singleton clusters remain
-

5.7.3 Basic Agglomerative Hierarchical Clustering Algorithms

Many agglomerative hierarchical clustering techniques are variations on a single approach: Starting with individual points as clusters, successively merge two clus-

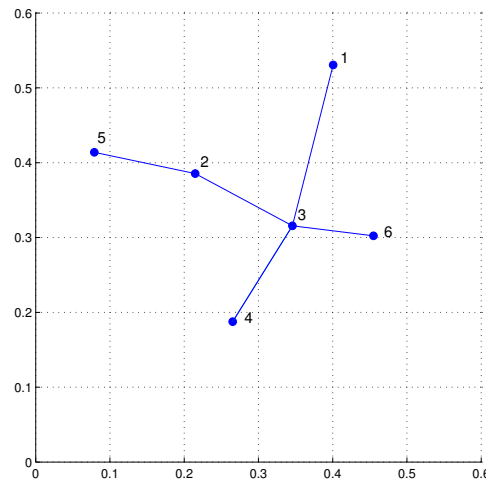


Figure 5.25. Minimum Spanning Tree for Set of Six Two-dimensional Points.

ters until only one cluster remains. This approach is expressed more formally in Algorithm 6.

Algorithm 6 Basic Agglomerative Hierarchical Clustering Algorithm

- 1: Compute the proximity graph, 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
-

5.7.4 Defining Proximity Between Clusters

The key step of the previous algorithm is the calculation of the proximity between two clusters, and this is where the various agglomerative hierarchical techniques differ. Cluster proximity is typically defined by a conceptual view of the clusters. For example, if we view a cluster as being represented by all the points, then we can take the proximity between the closest two points in different clusters as the proximity between the two clusters. This defines the *MIN* technique. Alternatively, we can take the proximity between the farthest two points in different clusters to be our definition of cluster proximity. This defines the *MAX* technique. (Notice that the names, ‘MIN’ and ‘MAX’ are only ‘appropriate’ if our proximities are distances, and thus, many prefer the alternative names, which, respectively, are *single link* and *complete link*. However, we shall prefer the terms ‘MIN’ and ‘MAX’ for their brevity.) Also, we can average the pairwise proximities of all pairs of two points from different clusters. This yields the *group average* technique. These three approaches

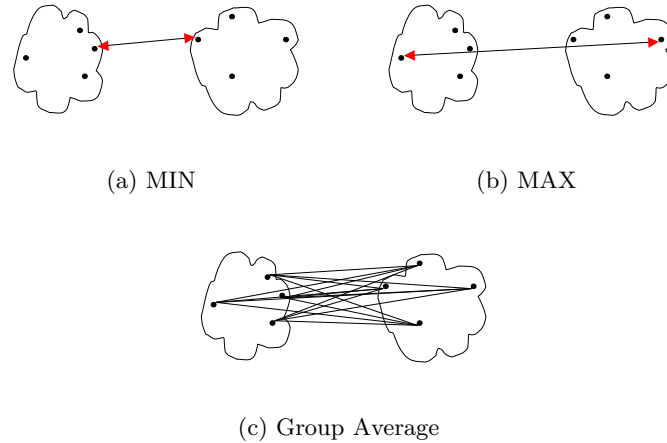


Figure 5.26. Definition of Cluster Proximity

are graphically illustrated by Figure 5.26.

If, instead, we represent each cluster by a centroid, then we find that different definitions of cluster proximity are more natural. For the *centroid* approach, the cluster proximity is defined as the proximity between cluster centroids. An alternative technique, *Ward's method*, also assumes that a cluster is represented by its centroid. However, it measures the proximity between two clusters in terms of the increase in the SSE that results from merging two clusters into one. Like K-means, Ward's method attempts to minimize the sum of the squared distance of points from their cluster centroids.

Time and Space Complexity

Hierarchical clustering techniques typically use a proximity matrix. This requires the computation and storage of m^2 proximities, where m is the number of data points. This is a factor that limits the size of data sets that can be processed. It is possible to compute the proximities on the fly and save space, but this increases computation time. Overall, the time required for hierarchical clustering is $O(m^2 \log m)$.

5.7.5 MIN or Single Link

For the single link or MIN version of hierarchical clustering, the proximity of two clusters is defined to be the minimum of the distance (maximum of the similarity) between any two points in the different clusters. (The technique is called 'single link' because, if you start with all points as singleton clusters, and add links between points, strongest links first, then these single links combine the points into clusters.) Single link is good at handling non-elliptical shapes, but is sensitive to noise and outliers. Figure 5.27 shows the result of applying MIN to our example data set of six points.

Figure 5.27a shows the nested clusters as a sequence of nested ellipses, where

the numbers associated with the ellipses indicate the order of the clustering. Figure 5.27b 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 5.7, we see that the distance between points 3 and 6 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 $dist(\{3, 6\}, \{2, 5\}) = \min(dist(3, 2), dist(6, 2), dist(3, 5), dist(6, 5)) = \min(0.1483, 0.2540, 0.2843, 0.3921) = 0.1483$.

5.7.6 MAX or Complete Link or CLIQUE

For the complete link or MAX version of hierarchical clustering, the proximity of two clusters is defined to be the maximum of the distance (minimum of the similarity) between any two points in the different clusters. (The technique is called “complete link” because, if you start with all points as singleton clusters, and add links between points, strongest 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 can break large clusters, and has favors globular shapes.

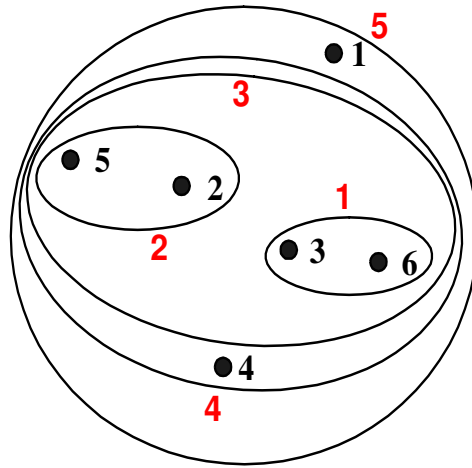
Figure 5.28 shows the results of applying MAX to the sample data set of six points. Again, points 3 and 6 are merged first. However, $\{3, 6\}$ is merged with $\{4\}$, instead of $\{2, 5\}$. This is because the $dist(\{3, 6\}, \{4\}) = \max(dist(3, 4), dist(6, 4)) = \max(0.1513, 0.2216) = 0.2216$, which is smaller than $dist(\{3, 6\}, \{2, 5\}) = \max(dist(3, 2), dist(6, 2), dist(3, 5), dist(6, 5)) = \max(0.1483, 0.2540, 0.2843, 0.3921) = 0.3921$ and $dist(\{3, 6\}, \{1\}) = \max(dist(3, 1), dist(6, 1)) = \max(0.2218, 0.2347) = 0.2347$.

5.7.7 Group Average

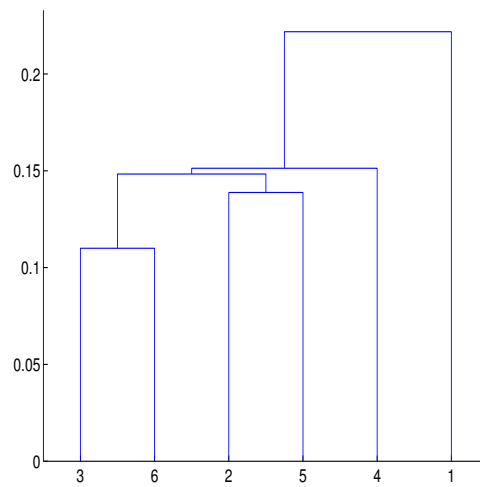
For the group average version of hierarchical clustering, the proximity of two clusters is defined to be the average of the pairwise proximities between all pairs of points in the different clusters. Notice that this is an intermediate approach between MIN and MAX. This is expressed by the following equation:

$$proximity(cluster_1, cluster_2) = \sum_{\substack{p_1 \in cluster_1 \\ p_2 \in cluster_2}} \frac{proximity(p_1, p_2)}{size(cluster_1) * size(cluster_2)} \quad (5.18)$$

Figure 5.29 shows the results of applying group average to the sample data set of six points. To illustrate how group average works, we calculate the distance between some clusters. $dist(\{3, 6, 4\}, \{1\}) = (0.2218 + 0.3688 + 0.2347)/(3 * 1) = 0.2751$. $dist(\{2, 5\}, \{1\}) = (0.2357 + 0.3421)/(2 * 1) = 0.2889$. $dist(\{3, 6, 4\}, \{2, 5\}) = (0.1483 + 0.2843 + 0.2540 + 0.3921 + 0.2042 + 0.2932)/(6 * 1) = 0.2637$. Because $dist(\{3, 6, 4\}, \{2, 5\})$ is smaller than $dist(\{3, 6, 4\}, \{1\})$ and $dist(\{2, 5\}, \{1\})$, these two clusters are merged at the fourth stage.

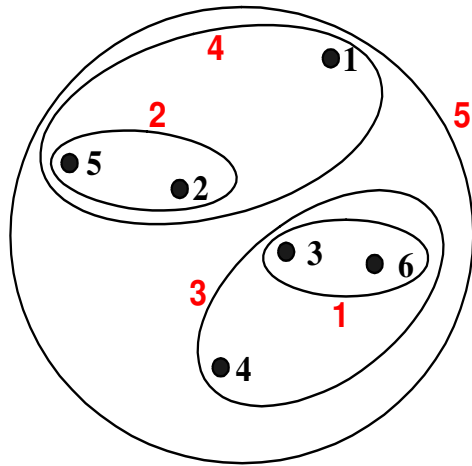


(a) Single Link Clustering

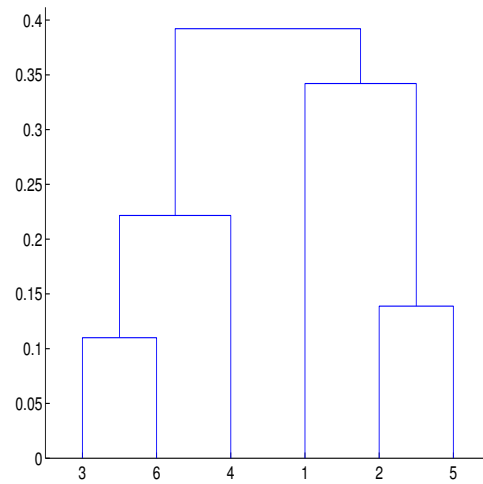


(b) Single Link Dendrogram

Figure 5.27. Single Link Clustering of Six Points.

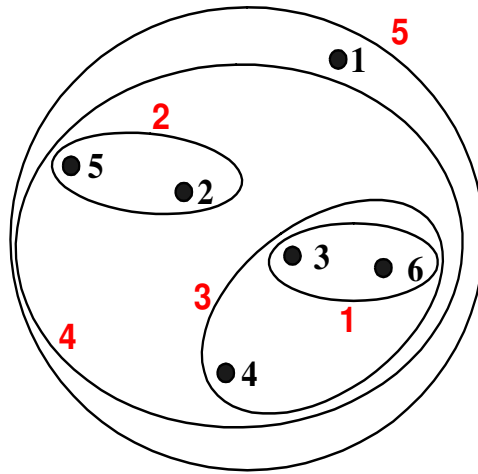


(a) Complete Link Clustering

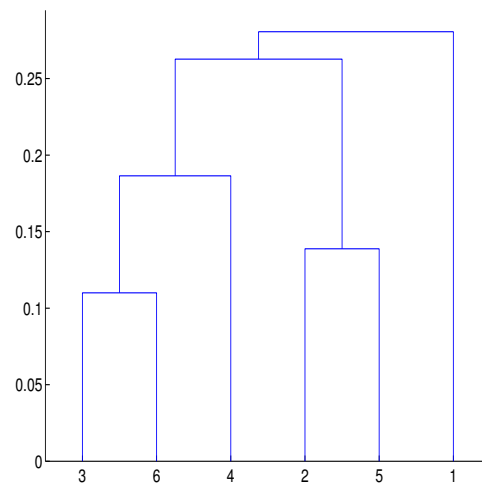


(b) Complete Link Dendrogram

Figure 5.28. Complete Link Clustering of Six Points.



(a) Group Average Clustering



(b) Group Average Dendrogram

Figure 5.29. Group Average Clustering of Six Points.

5.7.8 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 is used by K-means clustering. While it may seem that this makes this technique somewhat distinct from other hierarchical techniques, some algebra will show that this technique 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. Figure 5.30 shows the results of applying Ward's method to the sample data set of six points. The resulting clustering is somewhat different from those produced by MIN, MAX, 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 analogue.

Centroid methods also have a characteristic—often considered bad—that other hierarchical clustering techniques we have discussed don't possess: the possibility of *inversions*. To be specific, two clusters that are merged may be more similar (less distant) than the pair of clusters that were merged in a previous step. For other methods, the similarity of the clusters being merged monotonically decreases (the distance between merged clusters monotonically increases) as we proceed from singleton clusters to one all inclusive clusters.

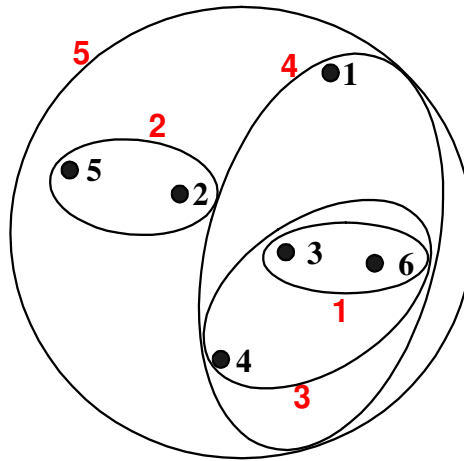
5.7.9 Key Issues in Hierarchical Clustering

Lack of a Global Objective Function

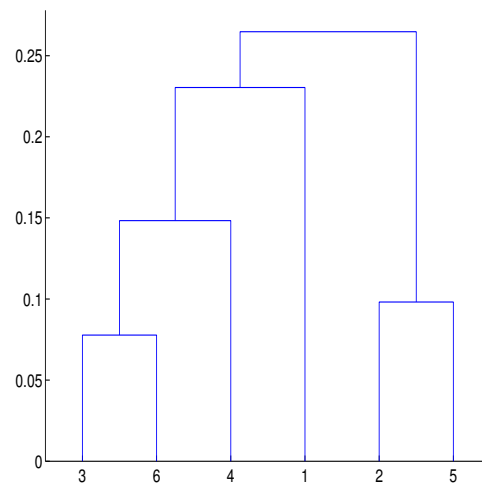
Previously, we mentioned that hierarchical clustering cannot be viewed as globally optimizing an objective function. Instead, hierarchical clustering techniques use various criteria to decide 'locally,' at each step, which clusters should be joined (or split for divisive approaches). This approach yields clustering algorithms that avoid the difficulty of trying to solve a hard combinatorial optimization problem. (As previously discussed, the general clustering problem for objective functions such as "minimize SSE" is NP hard.) 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.

The Impact of Cluster Size

Another aspect of agglomerative hierarchical clustering that should be considered is how to treat the relative sizes of the pairs of clusters that may be merged. (Note that this discussion only applies to cluster proximity schemes that involve sums, i.e., centroid and group average.) There are basically two schemes: weighted and unweighted. Weighted schemes treat all clusters equally, and thus, objects in smaller clusters effectively have larger weight. Unweighted schemes treat all objects equally.



(a) Ward's Clustering



(b) Ward's Dendrogram

Figure 5.30. Wards Clustering of Six Points.

Unweighted schemes are more popular, and in our previous discussions about the centroid and group average techniques, we discussed only the unweighted versions.

Merging Decisions are Final

Agglomerative hierarchical clustering algorithms tend to make good local decisions about combining two clusters since they have access to the proximity matrix. However, once a decision is made to merge two clusters, this decision cannot be undone at a later time. This prevents a local optimization criterion from becoming a global optimization criterion.

For example, in Ward's method, the 'minimize squared error' criteria from K-means is used in deciding which clusters to merge. However, this does not result in a clustering that could be used to solve the K-means problem. Even though the local, per-step decisions try to minimize the squared error, the clusters produced on any level, except perhaps the very lowest levels, do not represent an optimal clustering from a 'minimize global squared error' point of view. Furthermore, the clusters are not even 'stable,' in the sense that a point in a cluster may be closer to the centroid of some other cluster than to the centroid of its current cluster.

However, Ward's method can be used as a robust method of initializing a K-means clustering. Thus, a local minimize squared error objective function seems to have some connection with a global minimize squared error objective function.

Finally it is possible to attempt to fix up the hierarchical clustering produced by hierarchical clustering techniques. One idea is to move branches of the tree around so as to improve some global objective function. Another idea is to refine the clusters produced by a hierarchical technique by using an approach similar to that used for the multi-level refinement of graph partitions. Still another idea is to use a partitioning clustering technique, such as K-means to create many small clusters, and then perform hierarchical clustering using these small clusters as the starting point.

5.7.10 The Lance-William Formula for Cluster Proximity

Any of the cluster proximities that we discussed in this section can be viewed as a choice of different parameters (in the Lance-Williams formula shown below in equation 5.19) for the proximity between clusters Q and R , where R is formed by merging clusters A and B . (Note that in this formula $p(.,.)$ is a proximity function.) In words, this formula says that after you merge clusters A and B to form cluster R , then the proximity of the new cluster, R , to an existing cluster, Q , is a linear function of the proximities of Q from the original clusters A and B . Table 5.8 shows the values of these coefficients for the techniques that we discussed. n_A , n_B , and n_Q are the number of points in clusters A , B , and Q , respectively.

$$p(R, Q) = \alpha_{AP}p(A, Q) + \alpha_{BP}p(B, Q) + \beta p(A, B) + \gamma |p(A, Q) - p(B, Q)| \quad (5.19)$$

Table 5.8. Table of Lance-William Coefficients for Common Hierarchical Clustering Approaches

Clustering Method	α_A	α_B	β	γ
MIN	1/2	1/2	0	-1/2
MAX	1/2	1/2	0	1/2
Group Average	$\frac{n_A}{n_A+n_B}$	$\frac{n_B}{n_A+n_B}$	0	0
Centroid	$\frac{n_A}{n_A+n_B}$	$\frac{n_B}{n_A+n_B}$	$\frac{-n_A n_B}{(n_A+n_B)^2}$	0
Ward's	$\frac{n_A+n_Q}{n_A+n_B+n_Q}$	$\frac{n_B+n_Q}{n_A+n_B+n_Q}$	$\frac{-n_Q}{n_A+n_B+n_Q}$	0

Any hierarchical technique that can be phrased in this way does not need the original points, only the proximity matrix, which is updated as clustering occurs. However, while a general formula is nice, especially for implementation, it is often easier to understand the different hierarchical methods by looking directly at the definition of cluster proximity that each method uses, which was the approach taken in our previous discussion.

5.8 Density-Based Clustering

In this section, we describe some clustering algorithms that use the density-based definition of a cluster. Initially, we describe the algorithm, DBSCAN, which illustrates a number of important concepts in density-based clustering. We then examine DENCLUE, a clustering algorithm that can be parameterized to include the functionality of DBSCAN as a special case. To conclude the section, we discuss CLIQUE and MAFIA, which are two density-based clustering algorithms that are specifically designed for finding clusters in subspaces of high-dimensional data.

5.8.1 DBSCAN

DBSCAN is a density based clustering algorithm that works with a number of different distance metrics. After DBSCAN has processed a set of data points, a point will either be in a cluster or will be classified as a noise point. Furthermore, DBSCAN also makes a distinction between the points in clusters, classifying some as *core* points, i.e., points in the interior of a cluster, and some as *border* points, i.e., points on the edge of a cluster. Informally, any two core points that are ‘close enough’ 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. Noise points are discarded.

Classification of Points according to Density

Figure 5.31 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.

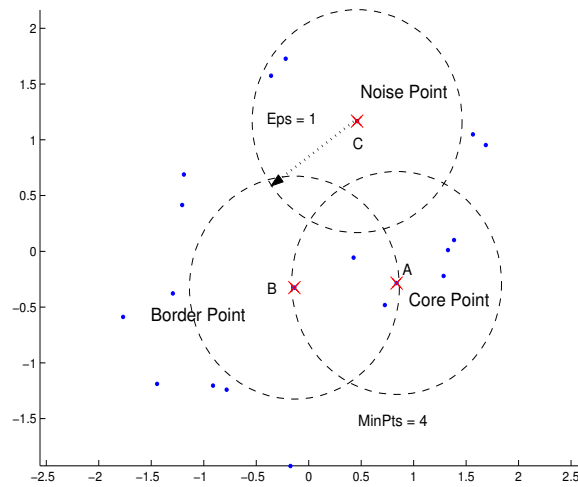


Figure 5.31. Core, Border and Noise Points for DBSCAN.

Core points. These are points that are at the interior of a cluster. A point is a core point if there are enough points in its neighborhood, i.e., if the number of points within a given neighborhood around the point, as determined by the distance function and a supplied distance parameter, Eps , exceeds a certain threshold, $MinPts$, which is also a supplied parameter. In Figure 5.31, point A is a core point, and the parameters are $Eps = 1$ and $MinPts = 5$.

Border points. A border point is a point that is not a core point, i.e., there are not enough points in its neighborhood for it to be a core point, but it falls within the neighborhood of a core point. In Figure 5.31, point B is a border point. A border point may fall within the neighborhoods of core points from different clusters.

Noise points. A noise point is any point that is not a core point or a border point. In Figure 5.31, point C is a noise point.

DBSCAN Algorithm

For DBSCAN, a cluster is the set of all core points whose neighborhoods transitively connect them together, along with some border points. An algorithm for the DBSCAN approach is shown below. In the original algorithm, the two loops: classify points as core, border, or noise points (lines 2-12) and assign points to clusters (lines 13-25), were merged, but here we separate them to simplify the presentation.

Time and Space Complexity

The basic complexity of the DBSCAN algorithm is $O(m * \text{time to find points in the } Eps\text{-neighborhood})$, where m is the number of points. In the worst case, high

Algorithm 7 DBSCAN Algorithm.

```

1: Initialization: Label all points as noise points and as belonging to no cluster.
   {First find core, border and noise points.}
2: for  $i = 1$  to  $n$  (the number of points) do
3:   Find all points within a distance  $EPS$  of the  $i^{th}$  point
   (These points, plus the  $i^{th}$  point itself, are the  $Eps$ -neighborhood of the point.)
4:   if the number of points in the  $Eps$ -neighborhood  $\geq MinPts$  then
5:     Label the  $i^{th}$  point as a core point
6:     for all points in the  $Eps$ -neighborhood, except  $i^{th}$  the point itself do
7:       if the point is still labelled as a noise point then
8:         Label the point as a border point
9:       end if
10:    end for
11:   end if
12: end for
   {Then we find the clusters.}
13: Eliminate noise points
14:  $current\_cluster\_label \leftarrow 1$ 
15: for all core points do
16:   if the core point has no cluster label then
17:      $current\_cluster\_label \leftarrow current\_cluster\_label + 1$ 
18:     Label the current core point with cluster label  $current\_cluster\_label$ 
19:   end if
20:   for all points in the  $Eps$ -neighborhood, except  $i^{th}$  the point itself do
21:     if the point does not have a cluster label then
22:       Label the point with cluster label  $current\_cluster\_label$ 
23:     end if
24:   end for
25: end for

```

dimensional data, this complexity is $O(m^2)$. However, in low dimensional spaces, there are data structures that allow the efficient retrieval of the 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 $O(m)$ since it is only necessary to keep a small amount of data for each point, e.g., the cluster label and the type of point, and (at least in the original version) the points in the Eps -neighborhood can be processed in a single pass.

Selection of DBSCAN parameters

There is, of course, the issue of how the parameters, Eps and $MinPts$ are determined. The basic approach is to look at the behavior of the distance from a point to its k^{th} nearest neighbor (the k -dist. For points in a cluster 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 not in a cluster, e.g., noise points, the k -dist will be relatively large. Thus, by computing the k -dist for all the data points for some k and plotting the distance, we can see which distances correspond to clusters and which ones correspond to noise. We select this distance to be the Eps parameter, and the value of k to be the $MinPts$ parameter. Since this procedure will depend on k it might seem as though there is still a problem. However, the value of Eps determined in this way does not change dramatically as k varies, and the creators of DBSCAN recommend $k = 4$ for two-dimensional data.

Since DBSCAN uses a density-based definition of a cluster, it is relatively resistant to noise and can handle clusters of arbitrary shapes and sizes. Consequently DBSCAN can handle many of the clusters that were difficult for K-means. However, DBSCAN does have trouble with higher dimensional data since density becomes more difficult to define in higher dimensions. Furthermore, DBSCAN may have trouble with density in lower dimensions if the density of clusters varies widely. For example, consider Figure 5.32. which shows four clusters embedded in noise. The density of the clusters and noise regions is indicated by their darkness and we see that the noise around the pair of denser clusters, A and B, has the same density as the pair of less dense clusters, C and D. DBSCAN is incapable of finding the less dense clusters.

However, to illustrate the strong points of DBSCAN, we show the clusters that it finds on a more complicated two-dimensional data, which consists of 3000 two-dimensional points as shown in Figure 5.33a. We find the Eps threshold for this data by plotting the sorted distances of the 4th nearest neighbor of each point (Figure 5.33b) and identifying the distance at which there is a sharp increase in these distances indicating noise. We select $Eps = 10$, although we admit that this is somewhat arbitrary. The clusters found by DBSCAN using these parameters, $MinPts = 4$, $Eps = 10$, are shown in Figure 5.33c. For completeness, we show the core points, border points, and noise points in Figure 5.33d.

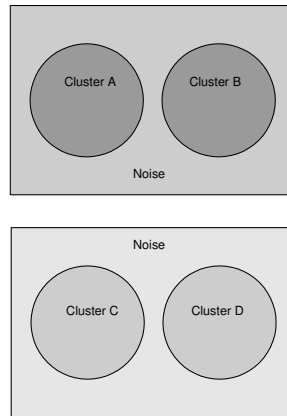


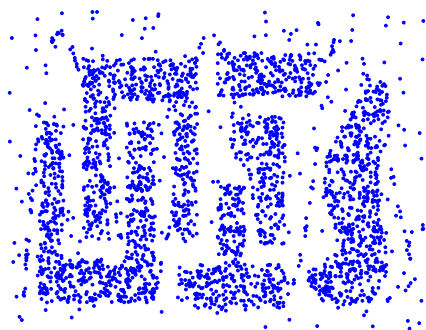
Figure 5.32. Four clusters embedded in noise.

5.8.2 DENCLUE

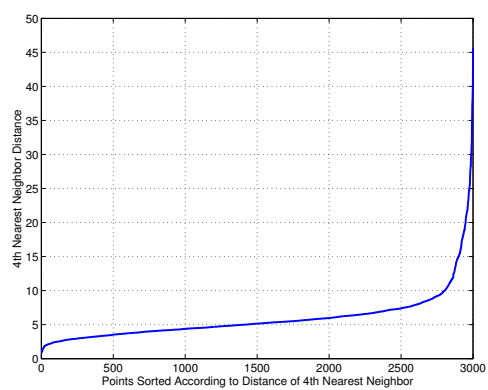
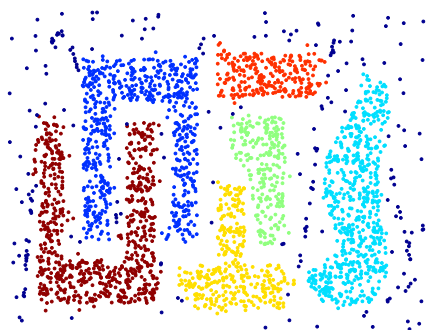
DENCLUE (DENSity CLUstEring) is a density clustering approach that models the overall density of a set of points as the sum of ‘influence’ functions associated with each point. The resulting overall density function will have local peaks, i.e., local density maxima, and these local peaks can be used to define clusters in a straightforward way. Specifically, for each data point, a hill climbing procedure finds the nearest peak associated with that point, and the set of all data points associated with a particular peak (called a local density attractor) becomes a (center-defined) cluster. However, if the density at a local peak is too low, then the points in the associated cluster are classified as noise and discarded. Also, if a local peak can be connected to a second local peak by a path of data points, and the density at each point on the path is above a minimum density threshold, ξ , then the clusters associated with these local peaks are merged. Thus, clusters of any shape can be discovered.

DENCLUE is based on a well-developed area of statistics and pattern recognition which is known as ‘kernel density estimation.’ The goal of kernel density estimation (and many other statistical techniques as well) is to describe the distribution of the data by a function. For kernel density estimation, the contribution of each point to the overall density function is expressed by an ‘influence’ (kernel) function. The overall density is then merely the sum of the influence functions associated with each point.

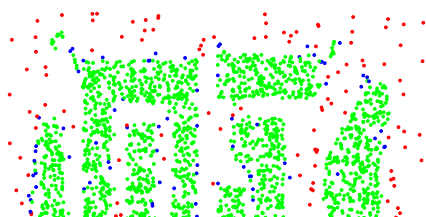
Typically the influence or kernel function is symmetric (the same in all directions) and its value (contribution) decreases as the distance from the point increases. For example, for a particular point, x , the Gaussian function, $K(x) = e^{-distance(x,y)^2/2\sigma^2}$, is often used as a kernel function. (σ is a parameter (the standard deviation) which governs how quickly the influence of point drops off.) Figure 5.34a shows what a Gaussian function would look for a single point in two dimensions, while Figure 5.34b shows the overall density function produced by applying the Gaussian influence function to the set of points shown in Figure 5.34b.



(a) Original Data Set

(b) Plot of Sorted 4^{th} Nearest Neighbor Distances

(c) Clusters Found By DBSCAN



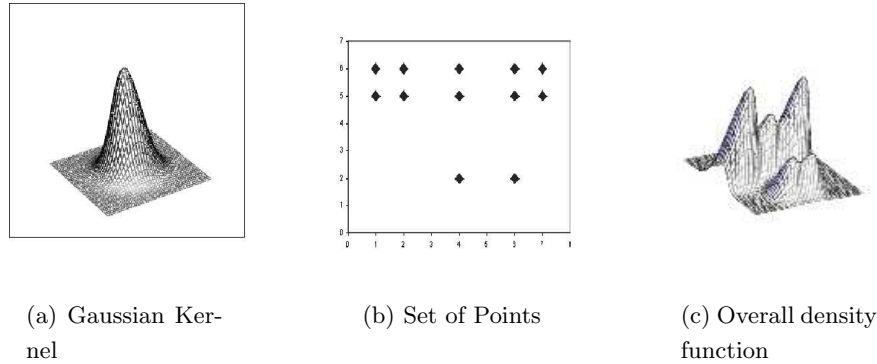


Figure 5.34. Example of the Gaussian influence (kernel) function and an overall density function. ($\sigma = 0.75$)

The DENCLUE algorithm has two steps, a preprocessing step and a clustering step. In the pre-clustering step, a grid for the data is created by dividing the minimal bounding hyper-rectangle into d -dimensional hyper-rectangles with an edge length of 2σ . The rectangles that contain points are then determined. (Actually, only the occupied hyper-rectangles are constructed.) The hyper-rectangles are numbered with respect to a particular origin (at one edge of the bounding hyper-rectangle) and these keys are stored in a search tree to provide efficient access during later processing. For each stored cell, the number of points, the sum of the points in the cell, and the connections to neighboring population cubes are also stored.

For the clustering step DENCLUE, considers only the highly populated cubes and the cubes that are connected to them. Starting with each of these cubes as a cluster, the algorithm proceeds as follows: For each point, x , the local density function is calculated only by considering those points that are from clusters which are a) in clusters that are connected to the one containing the point and b) have cluster centroids within a distance of $k\sigma$ of the point, where $k = 4$. As an efficiency measure, each point, x' , on the path from x to its density attractor is assigned to the same cluster as x if $\text{dist}(x, x') \leq \sigma/2$. As mentioned above, DENCLUE discards clusters associated with a density attractor whose density is less than ξ . Finally, DENCLUE merges density attractors that can be joined by a path of points, all of which have a density greater than ξ .

DENCLUE can be parameterized so that it behaves much like DBSCAN, but is more efficient than DBSCAN. DENCLUE can also be made to behave like K-means by choosing σ appropriately and by omitting the step that merges center-defined clusters into arbitrary shaped clusters. Finally, by doing repeated clusterings for different values of σ , a hierarchical clustering can be obtained.

5.8.3 Subspace Clustering

Until now, we found clusters by considering all of the attributes. However, if we consider only subsets of the features, i.e., subspaces of the data, then the clusters that we find can be quite different from one subspace, i.e., one set of features, to

another. Consider Figure 5.35. In Figure 5.35a we see the set of all points in three dimensional space. Clearly there are four clusters of points. Figure 5.35b shows the points plotted in the xy plane. (The z attribute is ignored.) Also, this figure contains histograms along the x and y axes that, respectively, show the distribution of the points with respect to their x and y coordinates. (A higher bar indicates that the corresponding interval contains relatively more points, and vice-versa.) Thus, there are four clusters when we consider only the x and y attributes, but only 3 clusters if we consider just the x or y attributes by themselves. From Figure 5.35c we see that there are 2 clusters if we consider only the z attribute. Finally, in Figure 5.35d, we see that there are three clusters when we consider just the x and z coordinates.

Techniques for Finding Subspace Clusters: CLIQUE

The previous example illustrates that the clusters we find depend on the subset of attributes that we consider. This raises a number of questions, but for now, we will focus on how to methodically find the clusters that might exist in different subspaces.

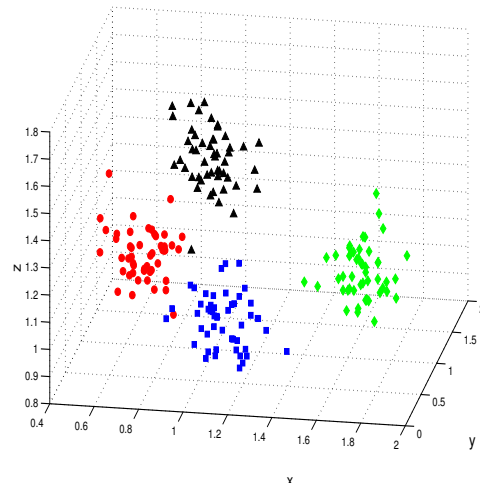
We start by describing the CLIQUE clustering algorithm. CLIQUE is a grid-based clustering algorithm. In particular, CLIQUE splits each dimension (attribute) into a fixed number (ξ) of equal length intervals. Conceptually, this partitions the data space into rectangular *units* of equal volume. Because these units (grid cells) have equal volume, we can measure the density of each unit by the fraction of points it contains. A unit is considered to be dense if the fraction of the overall points that it contains is above a user specified threshold, τ . A cluster is simply a group of collections of contiguous (touching) dense units.

The same approach can also be used to define clusters in subspaces, i.e., for subsets of the original attributes. Thus, for a single attribute (dimension) a cluster is a set of intervals that touch, i.e., are sequential and which each contain more than τ points. To illustrate this, Figure 5.36 shows a histogram of the data points of Figure 5.35 for the x attribute. (There are $\xi = 20$ intervals, each of size 0.1.) If our threshold for a dense unit was $\tau = 0.06$, or 6% of the points, then three “one-dimensional” clusters would be identified.

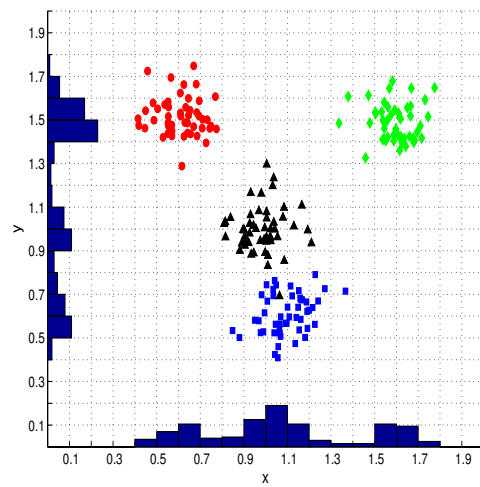
It is impractical to check each of the volume units to see if it is dense since the number of such units is equal to ξ^n . However, CLIQUE uses the following monotonicity property of density-based clusters.

Property 1 (Monotonicity property of density-based clusters) *If a set of points forms a density based cluster in k dimensions (attributes), then the same set of points is also part of a density based cluster in all possible subsets of those dimensions.*

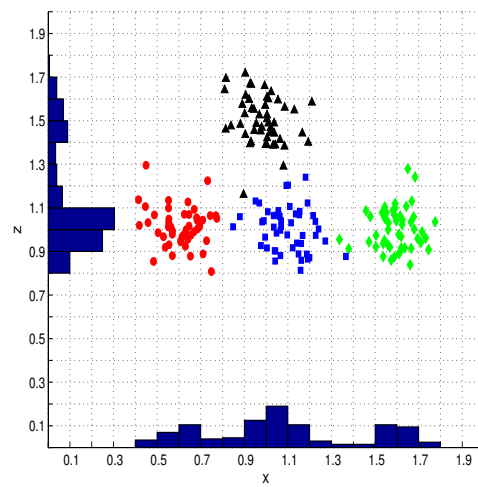
Given a set of units in a k dimensions, i.e., involving k specific attributes, we can find a corresponding set of units in $k - 1$ dimensions, i.e., by omitting one of the k dimensions, i.e., one of the k attributes. The lower dimensional units are



(a) Four clusters in three dimensions.



(b) View in the XY plane.



(c) View in the XZ plane.

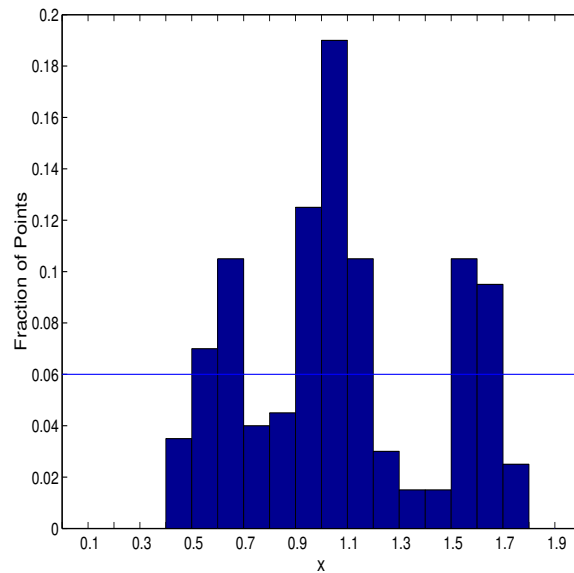


Figure 5.36. Histogram showing distribution of points for the X attribute.

all still touching, i.e., adjacent, and contain at least the same points (and perhaps additional points as well). Thus, the points are part of a lower dimensional cluster, i.e., the points form a cluster with the reduced set of attributes.

Thus, CLIQUE starts by finding all the dense areas in the one-dimensional spaces corresponding to each attribute. CLIQUE then generates the set of two-dimensional cells that might possibly be dense, by looking at pairs of dense one-dimensional cells, as each two-dimensional cell must be associated with a pair of dense one-dimensional cells. More generally, CLIQUE generates the possible set of k -dimensional cells that might possibly be dense by looking at dense $(k - 1)$ dimensional cells, since each k -dimensional cell must be associated with a set of k dense $(k - 1)$ -dimensional cells. Conceptually, the CLIQUE algorithm is similar to the APRIORI algorithm for finding frequent itemsets. (See Chapter 4).

Once CLIQUE finds the high-density units (cells) in all the subspaces, it then finds clusters by taking the union of all adjacent, high-density cells. For simplicity and ease of use, the list of cells that comprises a cluster is simplified into a smaller set of inequalities.

MAFIA (Merging of Adaptive Finite Intervals (And more than a CLIQUE))

It is possible to enhance CLIQUE. MAFIA is a modification of CLIQUE that runs faster (40+) and finds better quality clusters. There is also a parallel version of MAFIA, pMAFIA.

The main modification is to use an adaptive grid. Initially, each dimension (attribute) is partitioned into a large number of intervals, e.g., 1000. A histogram is generated that shows the number of data points in each interval. Groups of adjacent intervals, e.g., five, are grouped into windows, and the maximum number of points

in the window's intervals becomes the value associated with the window. Adjacent windows are grouped together if the values of the two windows are close — say within 20%. As a special case, if all windows are combined into one window, the dimension is partitioned into a fixed number of cells and the threshold for being considered a dense unit is increased for this dimension. A relatively uniform distribution of data in a particular dimension normally indicates that no clustering exists in this dimension.

Limitations of MAFIA and CLIQUE

Note that MAFIA and CLIQUE have limitations. In particular, both algorithms have a time complexity that is exponential in the number of dimensions. More generally, since both approaches are similar to the Apriori algorithm for finding frequent item sets, they both will have difficulty if ‘too many’ dense units are generated at lower stages. Also, these approaches may well fail if clusters are of widely differing densities, since the threshold is fixed. And, as with other clustering algorithms, determining the appropriate parameters, τ , and ξ , for a variety of data sets can be challenging.

Indeed, we remark that while CLIQUE and MAFIA can find clusters in many subspaces, it is not typically possible to find all clusters using the same threshold. The reason for this is that density drops as dimensionality increases, i.e., to find clusters in higher dimensions the threshold has to be set at a level that may well result in the merging of low dimensional clusters. To illustrate, Figure 5.37 shows the fraction of points in each of the two-dimensional units for the two dimensions x and y . Not surprisingly, the density in two dimensions has dropped dramatically from that of the x dimension shown in Figure 5.36; no two-dimensional unit (cell) has more than 6% of the points. In three dimensions (not shown), the maximum density is 3% of the points. However, in general, the main interest is in the higher dimensional clusters and, if so, this limitation may not be of great practical importance.

Additional Comments on Subspace Clustering

An obvious question is whether clusters that involve only some of the attributes are interesting. In some cases, the answer is yes. For example, if we are discretizing an attribute, then clustering with respect to a single attribute makes sense. Also, when K-means is used to find document clusters, the resulting clusters can typically be characterized by ten or so terms, i.e., the ‘true’ dimensionality of the document cluster is very small compared to number of dimensions (terms) overall which is usually in the thousands. However, if we have a categorical variable, which has only a few categories, then it would be possible to form clusters based on these categories. Although the clusters formed in this way are sometimes interesting and useful, in most data mining problems this is not the case. Also, we would not be too interested in a lower dimensional cluster if we knew that it was just a ‘projection’ of a higher dimensional cluster. For these reason, and because there are likely to be a lot of low dimensional clusters, i.e., higher dimensional clusters are rarer and hence,

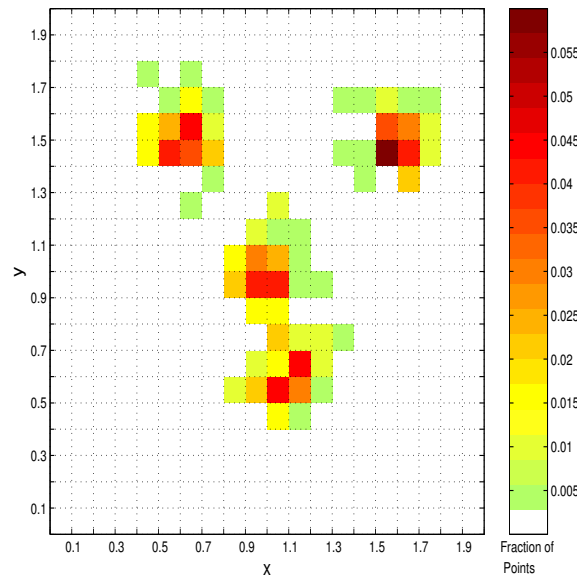


Figure 5.37. Distribution of points in the XY plane.

more interesting, the focus is on finding clusters that are of as high a dimension as possible. (In some ways this is analogous to looking at maximal itemsets instead of all frequent itemsets.)

Another important question, is why it is necessary to look for clusters in lower dimensional subsets at all. The reasons are much the same as those for doing dimensionality reduction for classification. If there are many attributes, many of the features may contribute little to creating structure within the data. For example, consider an attribute whose values are uniformly distributed. This type of feature has little value either for classification or clustering. Also, some features may be duplicates of other features. Finally, if no structure can be found with respect to all the features, it seems only sensible to look for whatever structure can be found.

5.9 Other Clustering Techniques

5.9.1 Fuzzy Clustering

If data points are distributed in well-separated groups, then a crisp classification of the points into disjoint clusters seems like an ideal approach. However, in most cases, the points in a data set cannot be partitioned into well-separated clusters, and therefore, there will be a certain arbitrariness in assigning a point to a particular cluster, e.g., consider a point that lies near the boundary of two clusters, but is just slightly closer to one of the two clusters. Thus, for each point, x_i , and each cluster, C_j , it would be beneficial to determine a weight, w_{ij} , that indicates the ‘degree’ to which x_i belongs to C_j .

For probabilistic approaches, e.g., mixture models, $w_{ij} = p_{ij}$, the probability

that x_i belongs to C_j . While this approach is useful in many situations, there are significant limitations to probabilistic models, i.e., times when it is difficult to determine an appropriate statistical model, and thus, it would be useful to have non-probabilistic clustering techniques that provide the same ‘fuzzy’ capabilities. Fuzzy clustering techniques are based on fuzzy set theory (see bibliographic notes) and provide a natural technique for producing a clustering where membership weights (the w_{ij}) have a natural (but not probabilistic) interpretation. In this section we will briefly describe the general approach of fuzzy clustering and provide a specific example in terms of fuzzy c-means (fuzzy K-means).

Lofti Zadeh introduced fuzzy set theory and fuzzy logic in 1965 as a way to deal with imprecision and uncertainty. Briefly, fuzzy set theory allows an object to partially belong to a set with a degree of membership between 0 and 1, while fuzzy logic allows a statement to be true with a degree of certainty between 0 and 1. (Traditional set theory and logic are special cases of their fuzzy counterparts which restrict the degree of set membership or the degree of certainty to be either 0 or 1.) Fuzzy concepts have been applied to many different areas, including control, pattern recognition, and data analysis (classification and clustering).

Consider the following example of fuzzy logic. The degree of truth of the statement “It is cloudy” can be defined to be the percentage of cloud cover in the sky, i.e., if the sky is 50% covered by clouds, then we would assign “It is cloudy” a degree of truth of 0.5. If we have two sets, “cloudy days” and “non-cloudy days,” then we can similarly assign each day a degree of membership in the two sets. Thus, if a day were 25% cloudy, it would have a 25% degree of membership in “cloudy days” and a 75% degree of membership in “non-cloudy days.”

We can now define fuzzy clusters. Consider a set of data points $X = \{x_1, \dots, x_m\}$, where each point, x_i , is an m dimensional point, i.e., $x_i = (x_{i1}, \dots, x_{im})$. We define a collection of fuzzy clusters, C_1, C_2, \dots, C_k to be a subset of all possible fuzzy subsets of X . (This simply means that the membership weights (degrees), w_{ij} , have been assigned values between 0 and 1 for each point, x_i , and each cluster, C_j .) However, we also want to impose the following ‘reasonable’ conditions on the clusters in order to ensure that the clusters form what is called a *fuzzy psuedo-partition*.

1. All the weights for a given point add up to 1.

$$\sum_{j=1}^k w_{ij} = 1$$

2. Each cluster (fuzzy subset) ‘contains’ at least one point, but not all of the points.

$$0 < \sum_{i=1}^m w_{ij} < m$$

While there are many types of fuzzy clustering – indeed, many data analysis algorithms can be ‘fuzzified’ – we only consider the fuzzy version of K-means, which is called fuzzy c-means. (In the clustering literature, the version of K-means which

does not use incremental updates of cluster centroids is sometimes referred to as c-means and this was the term adapted by the fuzzy community for the fuzzy version of K-means.) We first discuss some preliminaries and then present the fuzzy c-means algorithm.

First notice that we can also extend the definition of a centroid of a cluster to fuzzy sets in a straightforward way. For a cluster, C_j , the corresponding centroid, c_j , is defined by the following equation: $c_j = \sum_{i=1}^m w_{ij}^m x_i / \sum_{i=1}^m w_{ij}^m$, where m is a parameter, $1 < m < \infty$, that determines the influence of the weights (the membership degrees). Thus, the fuzzy centroid definition is much like the traditional definition except that all points are considered (any point can belong to any cluster, at least somewhat) and the contribution of each point to the centroid is weighted by its membership degree. In the case of traditional crisp sets, i.e., all w_{ij} are either 0 or 1, this definition reduces to the traditional definition of a centroid.

Furthermore, we can also modify the definition of overall cluster error as follows:

$$Error(C_1, C_2, \dots, C_k) = \sum_{j=1}^k \sum_{i=1}^m w_{ij}^m \|x_i - c_j\| \quad (5.20)$$

which is just the weighted version of the typical K-means error.

Lastly notice that during the c-means clustering process we will need to update the weights w_{ij} associated with each point and cluster. This can be accomplished by using the following equation.

$$w'_{ij} = \left(\sum_{p=1}^k \left(\frac{\|x_i - c_j\|^2}{\|x_i - c_p\|^2} \right)^{\frac{1}{m-1}} \right)^{-1} \quad (5.21)$$

However, if $\|x_i - c_j\|^2 = 0$ for one or more values of j , $1 \leq j \leq k$, then we need to use a different method of updating the weights associated with x_i . Since this corresponds to a situation where a point is the same as one or more centroids (hopefully centroids are unique, but they don't have to be), we want to set $w'_{ij} = 0$ for any cluster j where $x_i \neq c_j$ and split the weight of the point (which is 1) between the w'_{ij} where $x_i = c_j$. In the normal case, a vector will only be the same as one centroid and thus, we can just set all weights to 0, except for the w'_{ij} where $x_i = c_j$. In that case, we set $w'_{ij} = 1$.

Given the previous discussion the description of the fuzzy c-means algorithm is straightforward.

This algorithm is similar in structure to the K-means algorithm and also behaves in a similarly. In particular, fuzzy c-means works best for globular clusters, which are roughly the same size and density.

There are a few considerations when choosing the value of m . Choosing $m = 2$ simplifies the weight update formula. However, if m is chosen to be near 1, then fuzzy c-means behaves like traditional K-means. Going in the other direction, as

Algorithm 8 Basic Fuzzy c-means Algorithm.

-
- 1: Select an initial fuzzy pseudo-partition, i.e., assign values to all the w_{ij} .
(As with K-means this can be done randomly or in a variety of ways.)
 - 2: **repeat**
 - 3: Recompute the centroid of each cluster.
 - 4: Update the w_{ij} .
 - 5: **until** The centroids don't change
(Alternative stopping conditions are “if the change in the error is below a specified threshold” or “if the absolute change in any w_{ij} is below a given threshold.”)
-

m gets larger all the cluster centroids approach the centroid of the data. In other words, the partition becomes fuzzier as m increases.

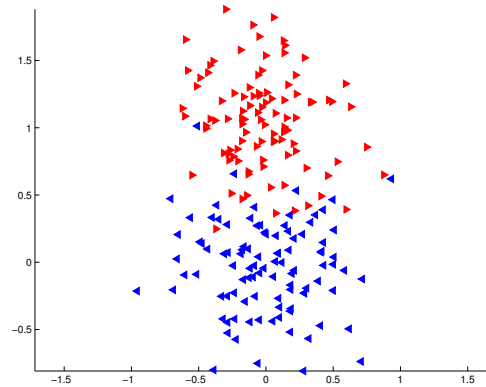
We mention a cautionary note. Fuzzy clustering, as presented here, deals well with situations where points are on the boundary of one or more clusters. It does not deal well with the situation where a point is an outlier, i.e., is not close to any cluster. The reason for this is that we assumed that all points have a weight of 1, a weight which is split between the different clusters. Thus, if we have an outlier, the point can have a relatively high weight with respect to one or more clusters, but not really be a ‘true’ member of any cluster. In this case the point should have low membership weights for all clusters, but then the weights would not add up to 1. However, with such an approach it would be possible to eliminate noise and outliers effectively, i.e., just eliminate points that do not have high membership weights in any cluster. Indeed, there is a variation of fuzzy c-means that takes this approach, but it has other problems, e.g., a tendency for all the centroids to become very close together.

To close this section, we illustrate, in Figure 5.38, the clusters found by fuzzy c-means for a set of two dimensional set of points. (Each cluster consists of 100 points that are generated from a two-dimensional Gaussian distribution. Since there are only two clusters, the color of the points reflects the membership weight with which points belongs to the lower cluster. Actual cluster membership is illustrated by the shape of the markers. Notice that membership in a cluster is strongest toward the center of the cluster and relatively weak for those points that are on the border of the two clusters.

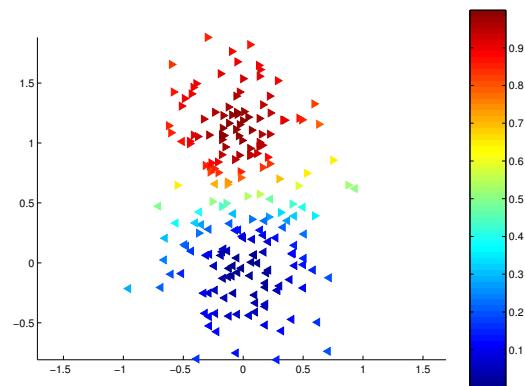
5.9.2 Clustering via Mixture Models and the EM Algorithm

Statistical Models for Data

Often it is convenient and effective to assume that the data has been generated as a result of a statistical process and to describe the data by finding the statistical *model* that best fits the data. More specifically, given a set of data, we assume a particular type of statistical distribution, e.g., Gaussian, and estimate the *parameters* of the data, e.g, the mean and variance. The estimated parameters then provide a succinct



(a) Original data. Marker type indicates the distribution from which it was generated.



(b) Clusters found by the fuzzy c-means data.

Figure 5.38. Fuzzy c-means clustering of a two-dimensional point set.

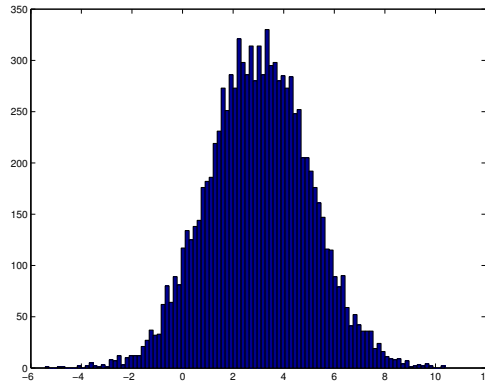


Figure 5.39. Histogram of 10,000 points from a normal distribution with a mean of 3 and a variance of 4.

description of each distribution (cluster).

To illustrate, consider Figure 5.39 which shows the histogram for a set of one-dimensional data. Since the histogram has the shape of a ‘normal’ curve, we might suspect that the data can be modelled as a normal distribution. From statistics, we know that a normal distribution is described by its mean and variance and that these can be estimated by calculating the mean and variance of the data. On doing so, we find a mean of 3.0133 and a variance of 4.0293. In fact this data was generated from a normal distribution and with mean 3 and a variance of 4.

More generally, a set of data might be viewed as a combination or mixture of several statistical distributions. We can think of a mixture model as a combination of a set of distributions, where each of the distributions is associated with a weight between 0 and 1 that represents the probability that a point is generated from that distribution. More formally, assume that we have k distribution, and that the probability density function for the i^{th} distribution is given by $prob_i$, i.e., $prob_i(x)$ is the probability density at point x with respect to the i^{th} distribution. Furthermore, let α_i , $1 \leq i \leq k$ be a set of weights such that $\sum_{i=1}^k \alpha_i = 1$. Then the probability density at any point x is given by the following equation.

$$prob(x) = \sum_{i=1}^k \alpha_i prob_i(x) \quad (5.22)$$

As an example, consider the set of points that are shown in Figure 5.38a. As previously mentioned, each cluster consists of 100 points that are generated from a two-dimensional Gaussian distribution. Each distribution has a variance of 0.1 in both the x and y attributes and no correlation between x and y . One cluster has a mean (center) of (0,0), while the other has a mean of (0,1).

Estimating the Model Parameters using Maximum Likelihood

If we assume a statistical model for the data, then it is necessary to estimate the parameters of that statistical model. A key approach for this parameter estimation

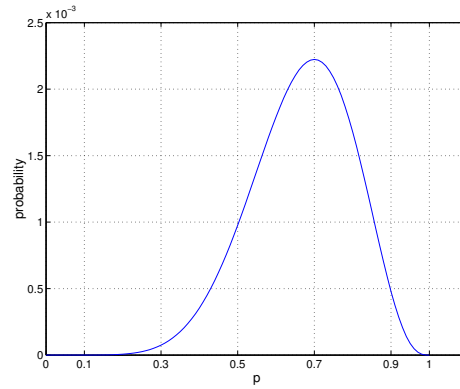


Figure 5.40. Plot of the likelihood function for 7 heads out of 10 coin tosses.

is that of *maximum likelihood* estimation, a concept which we now explain in greater depth.

To begin, consider the common statistical experiment of flipping a two-sided coin and recording the number of heads or tails that result on each toss. (Each toss of the coin is a data object with a single attribute, the result of the toss - heads or tails.) If the probability of a head is p for any particular toss, and the coin is flipped n times and there are k heads, then the probability $prob(k)$ of any particular sequence of tosses is given by equation 5.23.

$$prob(k) = p^k(1 - p)^{n-k} \quad (5.23)$$

Suppose that we toss the coin ten times and have 7 heads, but don't know what p is. What might the best estimate of p be? To get an idea, we plot the probability of 7 heads in ten tosses for different values of p between 0 and 1. This plot is shown in Figure 5.40.

Given a set of data, the probability of the data as a function of the parameters is called the likelihood function. We rewrite the previous equation (5.23 as equation 5.24 to emphasize that we view the statistical parameter p as our variable and that k , the number of heads, is a constant in this equation. Given such a likelihood function, a general principle for estimating the parameters of a statistical model is the *maximum likelihood principle*, i.e., choose those parameters that maximize the probability of the data. For the previous case, our estimate of p would be 0.7, which, in this case, is quite intuitive.

$$likelihood(p) = p^k(1 - p)^{n-k} \quad (5.24)$$

The general form of the likelihood equation for a single distribution is given by equation 5.25, where θ is a generic symbol for the parameters (θ may actually be a set of paramters) on which the statistical distribution depends), $prob(x_i|\theta)$ is the probability of the i^{th} data point given the parameters, and n is the number of data points. Note that the likelihood is just the product of the probabilities of each

individual data point, assuming that each data point was independently generated.

$$likelihood(\theta) = \prod_{i=1}^n prob(x_i|\theta) \quad (5.25)$$

In general, graphing the probability of the data for different values of the parameters won't work, at least if there are more than two parameters or if the range of data is large. Thus, in standard statistics books, the maximum likelihood estimates of statistical parameters are derived by using calculus, i.e., by taking the derivative of likelihood function, setting it equal to 0, and solving. It is left as an exercise for the reader to verify that this procedure used on equation 5.24 will yield a maximum likelihood estimate for p of k/n .

Estimating Mixture Model Parameters using Maximum Likelihood: Simple Case

We can also use the maximum likelihood approach to estimate the model parameters for a mixture model. In the simplest case, we know how many distributions we have, the type of the distributions, e.g., Gaussian or binomial, and which data objects come from which distributions. In this situation, we can estimate the parameters of each distribution separately by simply using the data generated by the distribution and our knowledge of the distribution type to create a likelihood function for the distribution. We can then choose our estimates for the parameters of the distribution to be those parameter values for which the likelihood is maximized. For most common distributions, the maximum likelihood estimates of the parameters are calculated from simple formulas involving the data, and thus, our task is easy.

For instance, suppose that we have a series of coins tosses, but that the even and odd tosses use two different coins. Furthermore, let the probability of a head on an odd toss be p_o , while the probability of a coin toss on an even toss is p_e . Then, if we have a sequence of 10 tosses, which is as follows: HTHHTTHTTT, we can estimate $p_o = 3/5$, while $p_e = 1/5$.

More formally, the likelihood for this type of data is given by the equation 5.26, where k_o = number of odd tosses that are heads, n_o = number of odd tosses, and k_e and n_e are defined analogously. A plot of this likelihood function for the string of tosses we have just considered is shown in Figure 5.41. Notice that this function is maximized when $p_o = 3/5$, while $p_e = 1/5$.

$$likelihood(p_o, p_e) = p_o^{k_o} (1 - p_o)^{n_o - k_o} p_e^{k_e} (1 - p_e)^{n_e - k_e} \quad (5.26)$$

Estimating Mixture Model Parameters using Maximum Likelihood: More Realistic Cases

We now consider more general (and more realistic) cases, where we know the number and type of distributions, but do not know which points were generated by which

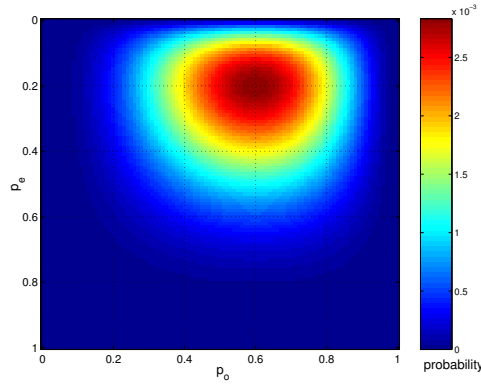


Figure 5.41. Plot of the likelihood function for the toss of 10 coins where even and odd tosses have a different probability of being heads.

distribution. In such cases it would seem that we are stuck, since to calculate the likelihood as given in equation 5.26, we need to calculate the probability of each data point, and to do that it would seem that we need to know which distribution each data point came from.

However, after a bit of thought, we might remember Bayes rule (see Appendix A) and decide to see if we can use this approach to compute the probability of a data point. Let $prob(x_i)$ be the probability of the i^{th} data point, $prob(x_i|\theta_j)$ be the probability of the i^{th} point if it comes from the j^{th} distribution which has parameters θ_j , Θ be the set of all parameters $\{\theta_1, \dots, \theta_k\}$, and α_j be the probability of the j^{th} distribution. Then by the application of Bayes rule we can rewrite equation 5.27 to get the following equation for the likelihood.

$$likelihood(\Theta) = \prod_{i=1}^n \sum_{j=1}^k \alpha_j prob(x_i|\theta_j) \quad (5.27)$$

However, this approach, which involves a product of sums does not work well in practice. Even if we take the log of the likelihood and try to maximize this quantity—a strategy typically followed because it converts a product into a sum, which is a nicer quantity to maximize by differentiation—we still have difficulties since we have the log of a sum, which is hard to work with. Thus, another approach is needed. Nonetheless, this approach will still depend heavily on Bayes rule.

The key insight is to ‘pretend’ that we know what distribution each data point came from by introducing a ‘distribution index variable,’ z_i , that takes on a value between 1 and k to indicate from which distribution an object was generated. Given such a variable we can rewrite our likelihood equation 5.27 as shown below.

$$likelihood(\Theta) = \prod_{i=1}^n \alpha_{z_i} prob(x_i|\theta_{z_i}) \quad (5.28)$$

Of course, while the form of this expression is simple, we don’t know—actually

have no way of knowing what the values of the z_i are. However, we can estimate the probability with which z_i takes certain values and thus, while we can't calculate $\text{likelihood}(\Theta)$, we can calculate the expected value, $E(\text{likelihood}(\Theta))$, of the likelihood function. By finding the parameters that maximize this expected likelihood function (or more commonly the expected log likelihood function), we can find estimates of the statistical parameters of the underlying distributions. (Expected values may be reviewed in A.) Thus, the equation that we will be using is shown in equation 5.29. Note that $z = (z_1, \dots, z_n)$ represents a particular set of distribution choices and $\text{prob}(z)$ is the probability of that set of distribution choices.

$$E(\text{likelihood}(\Theta)) = \sum_{\text{all possible } z} \left(\prod_{i=1}^n \alpha_{z_i} \text{prob}(x_i | \theta_{z_i}) \right) \text{prob}(z) \quad (5.29)$$

We explain this equation further. Suppose that there are k distributions. Then a set of n data objects can be generated in k^n different ways from the k distributions. For instance, consider an example of 10 coin tosses with two coins, coin 1 and coin 2. We might generate the 10 tosses by using coin 1 for the even tosses and coin 2 for the odd tosses, as in our example above. Or, we might generate the first five tosses using coin 1 and the last 5 using coin 2. Thus, we can represent each choice of distributions for generating the data as a tuple (or vector) of 10 numbers, where each component of the tuple is a number, 1 or 2, which indicates which coin (distribution) is to be used. For example, (2, 2, 1, 2, 1, 1, 2, 1, 1, 1).

The importance of all of this is the following: If we can determine the $\text{prob}(z)$ for all z , then equation 5.29 will depend only on the data and the choices for our parameters. Thus, we can use this equation to estimate the parameters by picking those parameters that result in the maximum expected likelihood.

In general, $\text{prob}(z)$ is given by equation 5.30, where $\text{prob}(z_i | x_i)$ is given by equation 5.31. We can think of this probability as the probability that point x_i was generated by distribution z_i . If the α_j 's are all equal, then they cancel out. For our examples, we will make that assumption.

$$\text{prob}(z) = \prod_{i=1}^n \text{prob}(z_i | x_i) \quad (5.30)$$

$$\text{prob}(z_i | x_i) = \frac{\alpha_{z_i} \text{prob}(x_i | \theta_{z_i})}{\sum_{j=1}^k \alpha_j \text{prob}(x_i | \theta_j)} \quad (5.31)$$

We illustrate how we can calculate the probability that a data point came from a particular distribution with an example. Suppose that we toss a coin, and the results is 'heads.' Additionally, assume that coin may actually be one of two coins, coin 1 and coin 2, and that the probability of a heads with coin 1 is p_1 , while the probability of a coin with coin 2 is p_2 . Then by Bayes theorem, the probability that coin 1 was used for the toss given a that the coin is a heads, $p(\text{coin 1} | H)$, is given by equation 5.32. We simplify the initial equation by assuming $\text{prob}(\text{coin 1}) = \text{prob}(\text{coin 2}) = 1/2$.

$$\begin{aligned}
\text{prob}(\text{coin 1}|H) &= \frac{\text{prob}(H|\text{coin 1}) \text{prob}(\text{coin 1})}{\text{prob}(H|\text{coin 1}) \text{prob}(\text{coin 1}) + \text{prob}(H|\text{coin 2}) \text{prob}(\text{coin 2})} \\
&= \frac{\text{prob}(H|\text{coin 1})}{\text{prob}(H|\text{coin 1}) + \text{prob}(H|\text{coin 2})}
\end{aligned}$$

To illustrate numerically, if $\text{prob}(H|\text{coin 1}) = 0.9$, $\text{prob}(H|\text{coin 2}) = 0.3$, then $\text{prob}(\text{coin 1}|H) = 0.9/(0.3+0.9) = 0.75$. Likewise, we can determine that $\text{prob}(\text{coin 2}|H) = 0.25$, $\text{prob}(\text{coin 1}|T) = 0.125$, and $\text{prob}(\text{coin 2}|T) = 0.875$. Thus, if we have 5 tosses, which yield the sequence HHTTH, with $z = (1, 2, 2, 1, 2)$, then, assuming all tosses are independent, we can compute $\text{prob}(z)$ as follows:

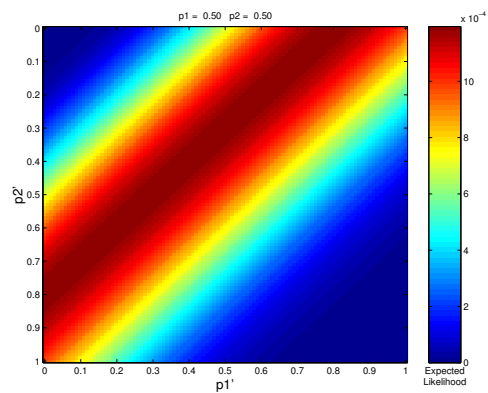
$$\begin{aligned}
\text{prob}(z) &= \text{prob}(1|H) * \text{prob}(2|H) * \text{prob}(2|T) * \text{prob}(1|T) * \text{prob}(2|H) \\
&= 0.75 * 0.25 * 0.875 * 0.125 * 0.25 \\
&= 0.0051
\end{aligned}$$

However, the reader might protest that we had to assume what the values of the parameters were in order to calculate the probability of z , and thus, we are once again stuck. Indeed, to find the expected likelihood, we need to know how to estimate the probability that a point was generated by a distribution, but to do this, we need to know what the distribution is. However, the solution is to make a guess for the parameters so that we can estimate the expected likelihood, and then to use the expected likelihood function to estimate a new guess for the parameters. If this process gives us a better guess for the parameters after each iteration, i.e., if the expected likelihood keeps increasing, then we can hope that our estimated parameters will eventually converge to a set of parameters which constitute ‘good’ estimates of the ‘true’ parameters. This is the EM algorithm that we will discuss further in the next section.

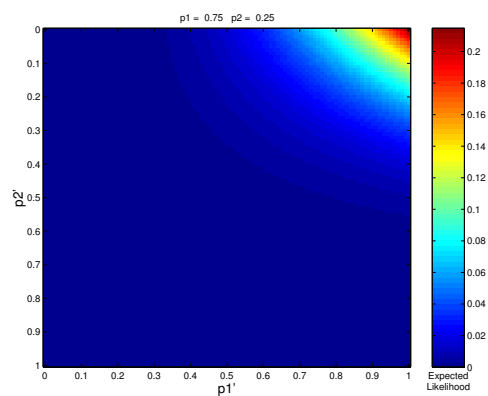
To close this section, we show the result of using this approach to estimate the probability of heads for two coins used to generate the sequence of tosses, HTH-HTTHTTT. Let $p_1 = 0.5$ and $p_2 = 0.5$ be our initial guesses for our parameters. Figure 5.43a shows the expected likelihood of the sequence of tosses under these assumptions as a function of possible values for the new values of our parameters, p'_1 and p'_2 . Many values of p'_1 and p'_2 yield the same maximum value, so we choose one, and perform the next iteration, which yields the graph of expected likelihood shown in Figure 5.43b. In this figure, $p'_1 = 1$ and $p'_2 = 0$ yields the maximum. The expected likelihood for this choice of parameters is chosen in Figure 5.43c. Since $p'_1 = 1$ and $p'_2 = 0$ again maximize the expected likelihood for this iteration, we have reached a stable point. There are a number of interesting details about this solution which are pursued in the exercises.

The EM Algorithm

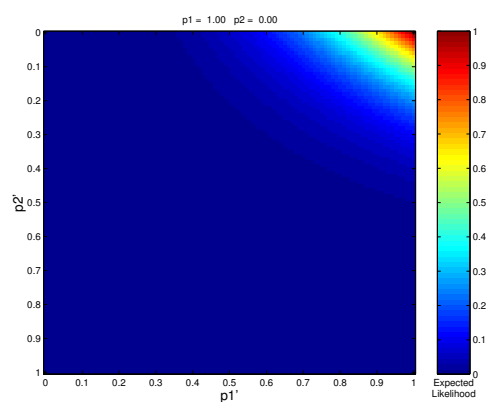
In the previous section, we presented an informal description of the EM algorithm in the context of a specific example: coin tosses. In this section we will be more



(a) First Iteration



(b) Second Iteration



(c) Last Iteration

Figure 5.42. Figures of the expected likelihood for different iterations of the EM algorithm.

general, although we will still be somewhat informal in our approach.

We begin by noting that with some algebraic manipulation, which we do not show, we can rewrite equation 5.29 as equation 5.32, where Θ' is the set of new parameters that we will choose so as to maximize the expected log likelihood, Θ is our current guess for the parameters, and $\text{prob}(j|x_i, \Theta)$ is the probability of distribution j given that we observe data point x_i and have parameters Θ . We have shifted to using the log likelihood, which allows us to split the expected log likelihood into two terms, each of which may be maximized separately.

$$E(\log(\text{likelihood}(\Theta', \Theta))) = \sum_{\text{all possible } z} \left(\sum_{i=1}^n \log(\alpha_{z_i} \text{prob}(x_i|\theta'_{z_i})) \right) \text{prob}(z) \quad (5.32)$$

$$= \sum_{j=1}^k \sum_{i=1}^n \log(\alpha'_j \text{prob}(x_i|\theta'_j)) \text{prob}(j|x_i, \Theta) \quad (5.33)$$

$$= \sum_{j=1}^k \sum_{i=1}^n \log(\alpha'_j) \text{prob}(j|x_i, \Theta) \quad (5.34)$$

$$+ \sum_{j=1}^k \sum_{i=1}^n \log(\text{prob}(x_i|\theta'_j)) \text{prob}(j|x_i, \Theta) \quad (5.35)$$

It is possible to find the new estimates of α'_j without any further assumptions. In particular, they are given by the equation 5.36. (Again, the details of the computation are left as an exercise.) For each data point, we calculate the probability of the j^{th} distribution given the occurrence of the given data point (and the current set of parameters) by using equation 5.31. We then take as our new estimate of the ‘mixing’ probability the average of these probabilities. Note that if we assume that all the α ’s are equal, then they are no longer parameters to be estimated and can be ignored in all further equations.

$$\alpha'_j = \frac{1}{k} \sum_{i=1}^n \text{prob}(j|x_i, \Theta) \quad (5.36)$$

In some cases the parameter estimates that maximizes the expected likelihood can also be expressed relatively simply in terms of the data and old parameters. For instance, if we have Gaussian distributions, then the estimates for the means and means and the covariance matrix are expressed by equations 5.37 and 5.38, respectively. In the case of the normal distribution, the estimates are ‘weighted’ versions estimates of the mean and covariance matrix for a single normal distribution, where each weight of a point is the probabilities that the point belongs to that distribution. It is interesting to compare these results to those of the fuzzy clustering of the last section.

$$\mu'_j = \frac{\sum_{i=1}^n x_i \text{prob}(j|x_i, \Theta)}{\sum_{i=1}^n \text{prob}(j|x_i, \Theta)} \quad (5.37)$$

$$\Sigma'_j = \frac{\sum_{i=1}^n (x_i - \mu'_i)(x_i - \mu'_i)^T p(j|x_i, \Theta)}{\sum_{i=1}^n p(j|x_i, \Theta)} \quad (5.38)$$

If we cannot directly calculate the parameters that maximize the expected likelihood, then we must try to find some scheme to efficiently search the parameter space. However, it is only necessary to search for a set of new parameters that yields a higher expected likelihood than the that of the previous iteration.

The EM algorithm is formally given by algorithm 9.

Algorithm 9 EM Algorithm.

- 1: Select an initial set of model parameters.
(As with K-means this can be done randomly or in a variety of ways.)
 - 2: **repeat**
 - 3: **Estimation Step:** For each object, calculate the probability of that each point belongs to each distribution, i.e, calculate $prob(j|x_i, \Theta)$.
 - 4: **Maximization Step:** Given the probabilities from the estimation step, find the new estimates of the parameters the maximize the expected likelihood.
 - 5: **until** The parameters do not change
(Alternative stopping conditions are if the change in the parameters is below a specified threshold.)
-

To complete this section, we present an example based on the data set used to illustrate the Fuzzy c-means algorithms - see Figure 5.38. For this data, we assumed that the data could be modelled as being generated by a mixture of two two-dimensional Gaussian distributions with different means and identical covariance matrices. Using an implementation of the EM clustering, we then performed the clustering and the results are shown in Figure 5.43. Since there are only two clusters, the color of the points reflects the probability that the points belong to the lower cluster. Actual cluster membership is illustrated by the shape of the markers. Notice that membership in a cluster is relatively weak for those points that are on the border of the two clusters, but strong elsewhere. It is interesting to compare the membership weights and probabilities of figures 5.43 and 5.38.

Summary

The basic idea of mixture models is to view the data as a set of observations from a mixture of different probability distributions. The probability distributions can be anything, but are often taken to be multivariate normal, since this type of distribution is well-understood, relatively easy to work with, and has been shown to work well in many instances. In this case, the types of distributions yield hyper-ellipsoidal clusters.

There are a number of different assumptions that can be made when taking a mixture model approach with normal distributions, i.e., whether all clusters have

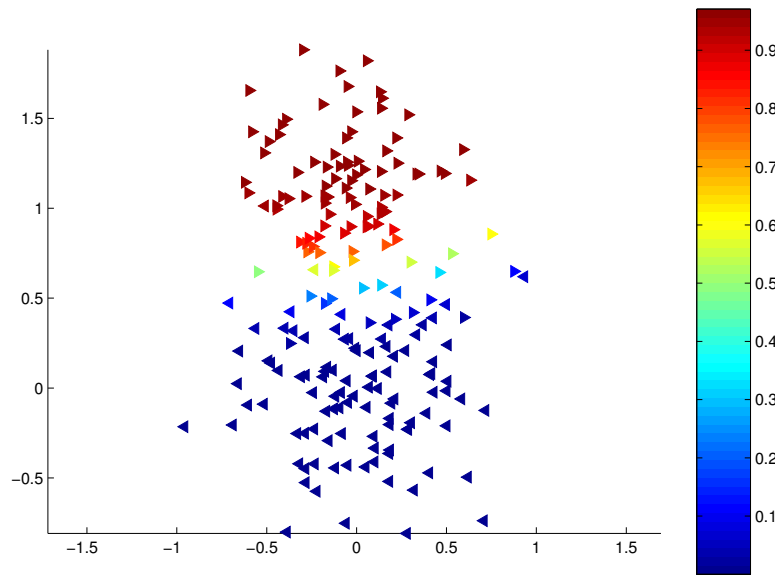


Figure 5.43. EM clustering of a two-dimensional point set generated from two multivariate normal distributions.

the same shape, volume, and orientation. These assumptions are reflected in the number of statistical parameters (mean vectors and covariance matrices), that need to be estimated. Of course, the more flexible the model, the more difficult and time consuming it is to find a solution.

Some form of the EM (expectation maximization) algorithm is typically used to perform the parameter estimation. This algorithm can be slow, is not practical for models with large numbers of components, and does not work well when clusters contain only a few data points or if the data points are nearly co-linear. There is also a problem in estimating the number of clusters or, more generally, in choosing the exact form of the model to use. This problem has typically been dealt with by applying a Bayesian approach, which roughly speaking, gives the odds of one model versus another, based on an estimate derived from the data. Mixture models may also have difficulty with noise and outliers, although work has also been done to deal with this problem.

In spite of all the apparent difficulties with mixture models, they do have the advantage of having a sound mathematical foundation. Also, a model-based approach provides a disciplined way of eliminating some of the complexity associated with data. (To see the patterns in data it often necessary to simplify the data, and fitting the data to a model is a good way to do that, at least if the model is a good match for the data.) Furthermore, it is easy to characterize the clusters produced, since they can be described by a small number of parameters. Finally, many sets of data are indeed the result of random processes and thus, should satisfy the statistical assumptions of these models.

5.9.3 Self-Organizing Maps (SOM)

The Kohonen Self-Organizing Feature Map (SOFM or SOM) [Koh97] is a clustering and data visualization technique based on neural networks. The ‘self-organizing’ part of the name derives from the ability of neural nets to learn their configuration from training data. The ‘map’ part of the name comes from the way that SOM neural nets work, i.e., when an input (a data point) is fed into a SOM neural network, the output is the label of a particular neuron in the network, and thus, a SOM neural network can be viewed as mapping a point to neuron. Finally, the ‘feature’ portion of the name comes from the observation that, in a trained SOM network, a small set of neurons are often associated with a particular data feature.

A detailed explanation of how SOM works is provided below, but conceptually, SOM can be viewed as a vector quantization technique, i.e., a compression technique in which each data point is represented by its closest *reference* vector, in which the reference vectors are learned by training a neural network. Thus, like vector quantization, the goal of SOM is to find a set of reference vectors and to assign each point in the data set to the reference vector which provides the best approximation of that vector. With the SOM approach, each reference vector is associated with a particular neuron, and the components of that reference vector become the ‘weights’ of that neuron. As with other neural network systems, a SOM neural net is trained by considering the data points one at a time and adjusting the weights (reference vectors) so as to best fit the data.

The final output of the SOM technique is a set of reference vectors which implicitly defines clusters. (Each cluster consists of the points closest to a particular reference vector.) However, while SOM might seem very similar to K-means or other vector quantization approaches, there is a fundamental conceptual difference. During the training process, SOM uses each data point to update the closest reference vector and *the reference vectors of nearby neurons*. In this way, SOM produces an ‘ordered’ set of reference vectors. In other words, the reference vectors of neurons which are close to each other in the SOM neural net will be more closely related to each other than to the reference vectors of neurons that are farther away in the neural net. Because of this constraint, the reference vectors in SOM can be viewed as lying on a smooth, elastic two-dimensional surface in m dimensional space, a surface which tries to fit the m dimensional data as well as possible. (Regression is a statistical techniques that tries to find the hyper-plane that best fits a set of data, and thus, you can think of the SOM reference vectors as the result of a nonlinear regression with respect to the data points.)

Details of a Two-dimensional SOM

There are many types of SOM neural networks, but we restrict our discussion to a simple variant where the neurons are organized as a two-dimensional lattice as shown in 5.44. Note that each neuron (node) is assigned a pair of coordinates (i, j) . Often such a network is drawn with links between adjacent neurons, but that is misleading since the influence of one neuron on another is via a neighborhood which

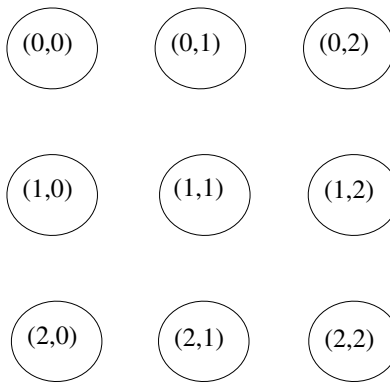


Figure 5.44. Two-dimensional 3 by 3 rectangular SOM neural network.

is defined in terms of coordinates, not links. Each neuron is associated with an m dimensional reference vector, where m is the dimensionality of the data points.

At a high level, clustering using the SOM techniques consists of the steps described in Algorithm 10. Initialization (line 1) can be performed in a number of

Algorithm 10 Basic SOM Algorithm.

- 1: Initialize the reference vectors.
 - 2: **repeat**
 - 3: Select the next training sample (point).
 - 4: Determine the neuron whose reference vector is closest to the current point.
 - 5: Update this reference vector and the reference vectors of all neurons which are close, i.e., in a specified neighborhood.
 - 6: **until** The training is complete
 - 7: When the training is complete, assign all points to the closest reference vectors and return the reference vectors and clusters.
-

ways. One approach is to choose each component of a reference vector randomly from the range of values observed in the data for that component. While this approach works, it is not necessarily the best approach, especially for producing rapid convergence to an equilibrium state. Another approach is to randomly choose the initial reference vectors from the available data points. (This is very much like one of the techniques for initializing K-means centroids.) There are also more sophisticated techniques - see the bibliographic notes.

The first step in the loop is the training or the convergence step (line 3). Selection of the next training sample is fairly straightforward, although there are some issues. Since training may require a 100,000 steps, each data point may be used multiple times, especially if the number of points is small. However, if the number of points is large, then not every point need be used. Also, it is possible to enhance the influence of certain groups of points, e.g., a less frequent class of points, by increasing

their frequency in the training set. (However, this becomes more like supervised classification situation.)

The determination of closest reference vector (line 4) is also straightforward, although it does require the specification of a distance metric. The Euclidean distance metric is often used, although a dot product metric is often used as well. When using the dot product distance, the data vectors are often normalized beforehand and the reference vectors are normalized at each step. In such cases, the dot product metric is equivalent to using the cosine measure.

The update step (line 5) is the most complicated. Let m_1, \dots, m_k be the reference vectors associated with each neuron. (For a rectangular grid, note that k = number of rows * number of columns.) For time step t , let $p(t)$ be the current training point and assume that the closest reference vector to $p(t)$ is m_j , the reference vector associated with neuron j . Then, for time $t + 1$, the j^{th} reference vector is updated by using the following equation. (We will see shortly that the update is really restricted to reference vectors whose neurons are in a small neighborhood of neuron j .)

$$m_j(t+1) = m_j(t) + h_j(t)(p(t) - m_j(t)) \quad (5.39)$$

Thus, at time t , a reference vector, $m_j(t)$, is updated by adding a term, $h_j(t)(p(t) - m_j(t))$, which is proportional to the difference, $p(t) - m_j(t)$, between the reference vector, $m_j(t)$, and the training point, $p(t)$. $h_j(t)$ determines the effect that the difference, $p(t) - m_j(t)$, will have and is chosen so that a) it diminishes with time and b) it enforces a neighborhood effect, i.e., the effect of a point is strongest on the neurons closest to the neuron j . Typically, $h_j(t)$ is chosen to be one of the following two functions:

$$h_j(t) = \alpha(t) \exp(-||r_j - r_k||^2 / (2\sigma^2(t))) \text{ (Gaussian function)} \quad (5.40)$$

$$h_j(t) = \alpha(t) \text{ if } ||r_j - r_k|| \leq \text{threshold}, 0 \text{ otherwise (step function)} \quad (5.41)$$

These functions require quite a bit of explanation. $\alpha(t)$ is a learning rate parameter, $0 < \alpha(t) < 1$, which decreases monotonically and which controls the rate of convergence. $r_k = (x_k, y_k)$ is the two-dimensional point which gives the grid coordinates of the k^{th} neuron. $||r_j - r_k||$ is the Euclidean distance of the two neurons, i.e., $\sqrt{(x_j - x_k)^2 + (y_j - y_k)^2}$. Thus, we see that for neurons that are ‘far’ from neuron j , the influence of training point $p(t)$ will be either greatly diminished or non-existent. Finally, note that σ is the typical Gaussian ‘variance’ parameter and controls the width of the neighborhood, i.e., a small σ will yield a small neighborhood, while a large σ will yield a wide neighborhood. The *threshold* used for the step function also controls the neighborhood size.

We emphasize that it is the neighborhood updating technique that enforces a relationship (ordering) between reference vectors associated with neighboring neurons.

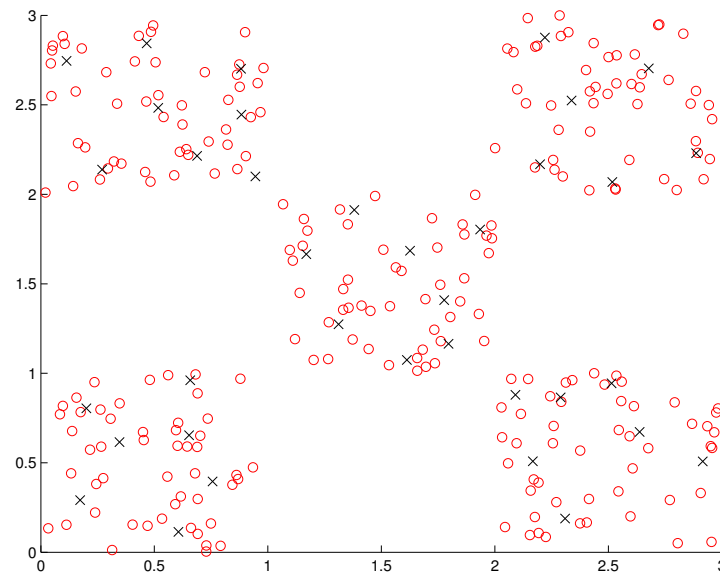


Figure 5.45. Distribution of reference vectors for a two-dimensional point set.

Deciding when training is complete is an important issue. Ideally, the training procedure should continue until convergence occurs, i.e., until the reference vectors are not changing much. The rate of convergence will depend on a number of factors, e.g., the data and $\alpha(t)$. We will not discuss these issues further except to mention that the most important issue, as with neural network in general, is the fact that convergence is often slow.

We provide only a simple illustration SOM, using the technique for two-dimensional data. In particular, Figure 5.45 shows a set of points - circles - and the positions of the 36 reference vectors - X's after running SOM. The data points are arranged in a checkerboard pattern, and a 6 by 6 two dimensional grid of reference vectors was used with random initialization.) The reference vectors tend to distribute themselves to the dense areas, even within the squares.

Applications

Once the SOM vectors are found, they can be used for many purposes besides clustering. For example, with a two-dimensional SOM it is possible to associate various quantities with the grid points associated with each neuron and to visualize the results via various types of plots. For example, plotting the number of points associated with each reference vector (neuron) yields a surface plot that reveals the distribution of points among neurons (clusters). (A two-dimensional SOM is a non-linear projection of the original probability distribution function into two dimensions. This projection attempts to preserve topological features and thus, SOM has been compared to the process of 'pressing a flower'.)

Besides data exploration, SOM and its supervised learning variant, LVQ (Learning Vector Quantization), have been used for many useful tasks, e.g., image segmen-

tation, organization of document files, and speech processing. There are thousands of papers that deal with SOM neural networks and the basic techniques have been extended in many directions. References are provided in the bibliographic notes.

5.10 Scalable Clustering Algorithms

5.10.1 Birch

BIRCH (Balanced and Iterative Reducing and Clustering using Hierarchies) is a highly efficient clustering technique for data in Euclidean vector spaces, i.e., data for which averages make sense. BIRCH can efficiently cluster such data with a single pass and can improve that clustering in additional passes. BIRCH also can deal effectively with outliers.

BIRCH is based on the notion of a clustering feature (CF) and a CF tree. The idea is that a cluster of data points (vectors) can be represented by a triple of numbers (N, LS, SS), where N is the number of points in the cluster, LS is the linear sum of the points, and SS is the sum of squares of the points. These are common statistical quantities and a number of different inter-cluster distance measures can be derived from them.

A CF tree is a height-balanced tree. Each interior node has entries of the form $[CF_i, child_i]$, where $child_i$ is a pointer to the i^{th} child node. The space that each entry takes and the page size, P, determine the number of entries, B, in an interior node. The space of each entry is, in turn, determined by the size of the points, i.e., by the dimensionality of the points.

Leaf nodes consist of a sequence of clustering features, CF_i , where each clustering feature represents a number of points that have been previously scanned. Leaf nodes are subject to the restriction that each leaf node must have a diameter that is less than a parameterized threshold, T. The space that each entry takes and the page size, P, determine the number of entries, L, of a leaf.

By adjusting a threshold parameter, T, the height of the tree can be controlled. T controls the fineness of the clustering, i.e., the extent to which the data in the original set of data is reduced. The goal is to keep the CF tree in main memory by adjusting the T parameter as necessary.

A CF tree is built as the data is scanned. As each data point is encountered, the CF tree is traversed, starting from the root and choosing the closest node at each level. When the closest “leaf” cluster for the current data point is finally identified, a test is performed to see if adding the data item to the candidate cluster will result in a new cluster with a diameter greater than the given threshold, T. If not, then the data point is ‘added’ to the candidate cluster by updating the CF information. The cluster information for all nodes from the leaf to the root is also updated.

If the new cluster would have a diameter greater than T, then a new entry is created if the leaf node is not full. Otherwise the leaf node must be split. The two entries (clusters) that are farthest apart are selected as ‘seeds’ and the remaining entries are distributed to one of the two new leaf nodes, based on which leaf node

contains the closest “seed” cluster. Once the leaf node has been split, the parent node is updated, and split if necessary, i.e., if the parent node is full. This process may continue all the way to the root node.

BIRCH follows each split with a merge step. At the interior node where the split stops, the two closest entries are found. If these entries do not correspond to the two entries that just resulted from the split, then an attempt is made to merge these entries and their corresponding child nodes. This step is intended to increase space utilization and avoid problems with skewed data input order.

BIRCH also has a procedure for removing outliers. When the tree needs to be rebuilt, because it has run out of memory, then outliers can optionally be written to disk. (An outlier is defined to be node that has “far fewer” data points than average.) At certain points in the process, outliers are scanned to see if they can be absorbed back into the tree without causing the tree to grow in size. If so, they are re-absorbed. If not, they are deleted.

BIRCH consists of a number of phases beyond the initial creation of the CF tree. All the phases of BIRCH are described briefly in Algorithm 11.

5.10.2 CURE

CURE (Clustering Using Representatives) is a clustering algorithm that uses a variety of different techniques to create an approach which can handle large data sets, outliers, and clusters with non-spherical shapes and non-uniform sizes.

CURE represents a cluster by using multiple ‘representative’ points from the cluster. These points will, in theory, capture the geometry and shape of the cluster. The first representative point is chosen to be the point furthest from the center of the cluster, while the remaining points are chosen so that they are farthest from all the previously chosen points. In this way, the representative points are naturally relatively well distributed. The number of points chosen, is a parameter, c , but it was found that a value of 10 or more worked well.

Once the representative points are chosen, they are shrunk toward the center by a factor, α . This helps moderate the effect of outliers, which are usually farther away from the center and thus, are ‘shrunk’ more. For example, a representative point that was a distance of 10 units from the center would move by 3 units (for $\alpha = 0.7$), while a representative point at a distance of 1 unit would only move 0.3 units.

CURE uses an agglomerative hierarchical scheme to perform the actual clustering. The distance between two clusters is the minimum distance between any two representative points (after they are shrunk toward their respective centers). While this scheme is not exactly like any other hierarchical scheme that we have seen, it is equivalent to centroid based hierarchical clustering if $\alpha = 0$, and roughly the same as single link hierarchical clustering if $\alpha = 1$. Notice that while a hierarchical clustering scheme is used, the goal of CURE is to find a given number of clusters as specified by the user.

CURE takes advantage of certain characteristics of the hierarchical clustering process to eliminate outliers at two different points in the clustering process. First,

Algorithm 11

- 1: **Load the data into memory by creating a CF tree that ‘summarizes’ the data.**
 - 2: **Build a smaller CF tree if it is necessary for phase 3.** T is increased, and then the leaf node entries (clusters) are reinserted. Since T has increased, some clusters will be merged.
 - 3: **Determine the neuron whose reference vector is closest to the current point.**
 - 4: **Perform global clustering.** Different forms of global clustering (clustering which uses the pairwise distances between all the clusters) can be used. However, an agglomerative, hierarchical technique was selected. Because the clustering features store summary information that is important to certain kinds of clustering, the global clustering algorithm can be applied as if it were being applied to all the points in cluster represented by the CF.
 - 5: **Redistribute the data points using the centroids of clusters discovered in step 3 and thus, discover a new set of clusters.** This overcomes certain problems that can occur in the first phase of BIRCH. Because of page size constraints and the T parameter, points that should be in one cluster are sometimes split, and points that should be in different clusters are sometimes combined. Also, if the data set contains duplicate points, these points can sometimes be clustered differently, depending on the order in which they are encountered. By repeating this phase, multiple times, the process converges to a (possibly local) optimum solution.
-

if a cluster is growing slowly, then this may mean that it consists mostly of outliers, since by definition outliers are far from others and will not be merged with other points very often. In CURE this first phase of outlier elimination typically occurs when the number of clusters is $1/3$ the original number of points. The second phase of outlier elimination occurs when the number of clusters is on the order of K , the number of desired clusters. At this point small clusters are again eliminated.

Since the worst case complexity of CURE is $O(m^2 \log m)$, it cannot be applied directly to large data sets. For this reason, CURE uses two techniques for speeding up the clustering process. The first technique takes a random sample and performs hierarchical clustering on the sampled data points. This is followed by a final pass that assigns each remaining point in the data set to one of the clusters by choosing the cluster with closest representative point.

In some cases, the sample required for clustering is still too large and a second additional technique is required. In this situation, CURE partitions the sample data into p partitions of size m/p and then clusters the points in each partition until there are m/pq clusters in each partition. This pre-clustering step is then followed by a final clustering of the m/q intermediate clusters. Both clustering passes use CURE's hierarchical clustering algorithm and are followed by a final pass that assigns each point in the data set to one of the clusters.

We summarize our description of CURE by explicitly listing the different steps in Algorithm 12.

Algorithm 12 CURE.

- 1: **Draw a random sample from the data set.** The CURE paper is notable for explicitly deriving a formula for what the size of this sample should be in order to guarantee, with high probability, that all clusters are represented by a minimum number of points.
 - 2: **Partition the sample into p equal sized partitions.**
 - 3: **Cluster the points in each cluster using the hierarchical clustering algorithm to obtain m/pq clusters in each partition and a total of m/q clusters.** Note that some outlier elimination occurs during this process.
 - 4: **Eliminate outliers.** This is the second phase of outlier elimination.
 - 5: **Assign all data to the nearest cluster to obtain a complete clustering.**
-

5.11 Cluster Evaluation

In supervised classification, the evaluation of the resulting classification model is critically important and there are relatively well-accepted measures and procedures that are a normal part of the process of developing a classification model, e.g., accuracy and cross-validation, respectively. However, despite the efforts of many researchers, 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.

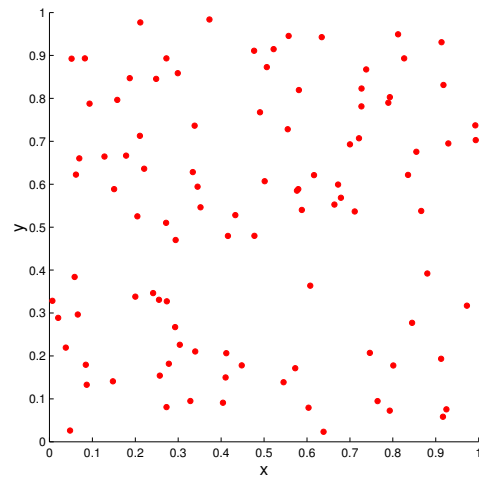
There might be some confusion as to why cluster evaluation is necessary. After all, many times cluster analysis is conducted as a part of exploratory data analysis and hence, evaluation seems like an unnecessarily complicated addition to what is supposed to be an informal process. Furthermore, since 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 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. In particular, a key motivation is that virtually any clustering algorithm will produce clusters given a data set, even if that data set has no natural structure. For instance, consider Figure 5.46, which shows the result of clustering 100 points that are randomly (uniformly) distributed on the unit square. The original points are shown in Figure 5.46a, while the clusters found by DBSCAN, K-means and complete link are shown in figures 5.46b, 5.46c, and 5.46d, respectively. Since DBSCAN found three clusters (after we set *EPS* by looking at the distances of the 4th nearest neighbor), we set K-means and complete link to find three clusters as well. (In Figure 5.46b the noise is shown by the small markers.) However, for none of the three methods do the clusters look extremely compelling. Furthermore, in higher dimensions, such problems cannot be so easily detected.

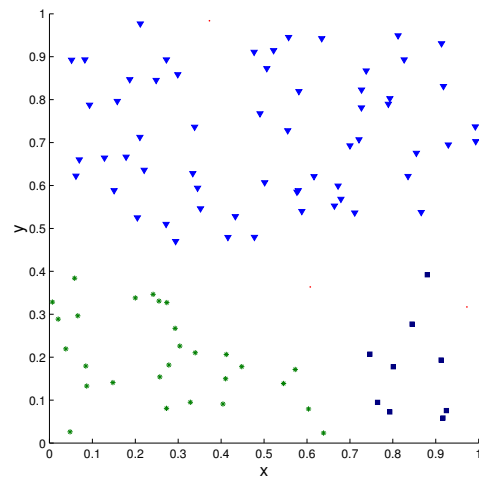
5.11.1 Overview

Being able to distinguish if there is any non-random structure in the data is just one important aspect of cluster validation. The following is an enumeration of several important aspects of cluster validation.

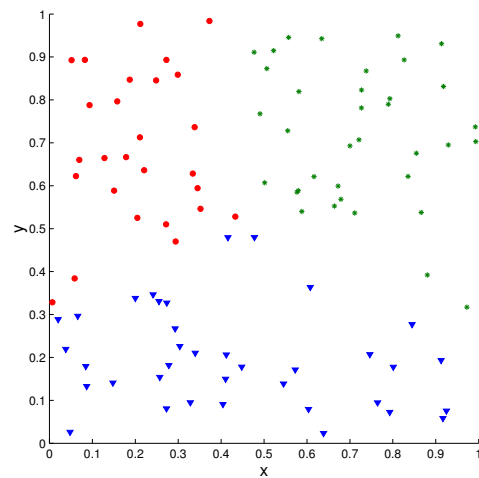
1. Determining the *clustering tendency* of a set of data, i.e., distinguishing whether non-random structure actually exists in the data.
2. Comparing the results of a cluster analysis to externally known results, e.g., to externally given class labels. (We may wish to determine if a clustering technique can automatically provide a classification for objects which can be classified manually, but with much effort.)
3. Evaluating how well the results of a cluster analysis fit the data *without* reference to external information.
4. Comparing two different sets of clusters to determine which is better.
5. Determining the ‘correct’ number of clusters.



(a) Original Data Set



(b) Three Clusters Found By DBSCAN



(c) Three Clusters Found by K-means

Notice that a further distinction can be made with respect to points 2, 3 and 4: whether 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 problems. 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 have been for two or three dimensional spatial data. Secondly, we need a framework to interpret any measure. For example, if our measure for evaluating how well cluster labels match externally provided labels has the value, 10, does that represent a good, fair, or poor match? The goodness of a match might be measured statistically, i.e., by how unlikely such a value is to occur by chance. Finally, if the measure is too complicated to apply or to understand, then people will be reluctant to use it.

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

External Index. Used to measure the extent to which the clustering structure discovered by a clustering algorithm matches some external structure. For example, as already mentioned, do cluster labels match externally supplied class labels? Entropy is an example of an external index.

Internal Index Used to measure the goodness of a clustering structure without respect to external information. An example of this is SSE.

Relative Index Used to compare two different clusterings or clusters. Often an external or internal index is used for this function. For instance, two K-means clusterings can be compared by comparing their total SSE or entropy.

These measures are sometimes referred to as *criteria* instead of indices. However, at other times, the term, ‘criterion,’ is used for general strategies, while the term ‘index’ is used for the numerical measure that implements the criterion.

In the remainder of this section, we shall describe some of the most commonly used techniques for evaluating cluster validity. In most cases these will be different external or internal indices. (All the relative indices that we will discuss are just external or internal indices used for comparing clustering results, and so, we do not make this a separate topic.) However, in some cases, the techniques will be graphical approaches that also can yield very useful information.

We must confess though, that we cover few of the cluster validity indices developed by earlier researchers. There are several reasons for this. For instance, many of the approaches require more statistical background than we wish to require of the reader. Furthermore, many of these approaches have appeared only in specialized literature and have not seen widespread use. Finally, even many of the indices which have been more frequently used were developed for smaller data sets and different types of data than are currently encountered in many data mining situations.

5.11.2 Measuring Cluster Validity via Correlation

If we are given the proximity matrix and the cluster labels for a data set, then we can evaluate the ‘goodness’ of the clustering by looking at the correlation between the proximity matrix and an idealized version of the proximity matrix, which we call the *incidence matrix*. (For simplicity in what follows, we assume that our proximity matrix is a similarity matrix.) More specifically, an ideal cluster is one which has a similarity of 1 to all points in the cluster, and has a similarity of 0 for all points in other clusters. Thus, a ‘perfect’ similarity matrix has a *block diagonal* structure, i.e., the similarity is 0 outside the blocks of the similarity matrix whose entries represent intra-cluster similarity. (Within those blocks, the similarity is 1, but this is not required of the definition of a block diagonal matrix.) The incidence matrix is easily constructed by creating a matrix that has one row and one column for each data point—just like a similarity matrix—and assigning a 1 to an entry if the associated pair of points belong to the same cluster. All other entries are 0.

High correlation between the incidence and similarity matrices indicates that points that belong to the same cluster are close to each other, while low correlation indicates the opposite. Thus, this is not a good measure for many density or contiguity based clusters, since they are not globular and may be closely intertwined with other clusters. Also, since the similarity and incidence matrices are symmetric, only the correlation between $n(n-1)/2$ entries needs to be calculated.

To illustrate this measure, we calculated the correlation between the similarity matrix and incidence matrix for the K-means clusters shown in Figure 5.46c (random data) and Figure 5.47a (data with three well-separated clusters). The correlations, were respectively, 0.5810 and 0.9235, which reflect that fact 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.

5.11.3 Judging a Clustering Via Its Similarity Matrix

The above approach suggest a more general, albeit less quantitative approach to judging a set of clusters, i.e., order the proximity matrix with respect to cluster labels and then plot it. In theory, if we have well-separated clusters, then the proximity matrix should be roughly block-diagonal. If not, then the patterns displayed in the similarity matrix can reveal the relationships between clusters.

This idea is best illustrated graphically. Consider the points in Figure 5.47a. If we run K-means on these points, looking for three clusters, then we will have no trouble finding these clusters since they are well-separated. This is illustrated by the reordered similarity matrix that is shown in Figure 5.47b. (For uniformity, we have transformed the distances into similarities using the formula, $s = 1 - \frac{d - \min_d}{\max_d - \min_d}$.) The reordered similarity matrices for random data set of Figure 5.46 and the DBSCAN, K-means and complete link clustering approaches are shown in Figure 5.48.

While this approach to evaluating the overall goodness of a clustering is qualitative, it is clear that well-separated clusters show a very strong, block-diagonal

pattern in the reordered proximity matrix. However, while the pattern for the clusters in random data is much weaker, there is some pattern. We emphasize that just as people can find patterns in clouds, data mining algorithms can find clusters in random data. But while we find it 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 $O(n^2)$ time. However, if the number of clusters is not too large, then sampling may still allow this method to be used. More specifically, the idea would be to 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 allow for an adequate representation of all clusters.

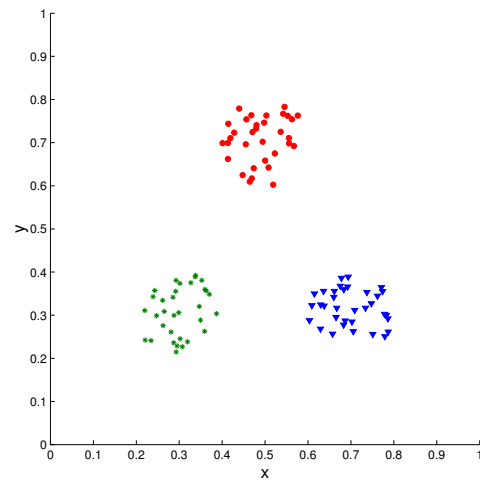
5.11.4 An internal measure of cluster validity: SSE

An internal measure of cluster validity, e.g., SSE, can be used to evaluate a clustering either in absolute terms or with respect to another clustering. Furthermore, an internal measure can often be used to evaluate either an entire clustering or an individual cluster. This is because it often makes sense to express the overall goodness of the clustering as a sum or weighted sum of the individual goodness (or validity) of each cluster.

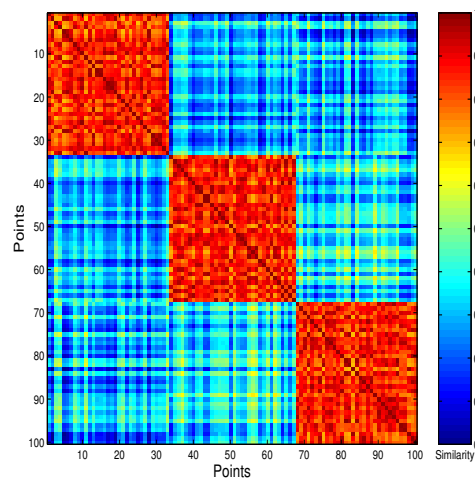
In this section, we shall use SSE as our example of an internal validity index. However, we stress that SSE is only appropriate when we are looking for globular clusters with a K-means type of algorithm, and for other types of clusters and clustering algorithms, different sorts of cluster validity indices are more appropriate.

For comparing two clusterings, on the same data with the same number, it is sufficient to compare the corresponding values of SSE. We take the clustering with the lower SSE. (In practice, there may be other factors involved, and what may happen is that a clustering with a low SSE is selected over the lowest SSE clustering.) However, for two clusters, a straight comparison does not take into account the different numbers of points in each cluster. Hence, the mean SSE of a cluster is a more appropriate measure for comparison. Nonetheless, the comparison of clusterings or clusters is straightforward.

We wrap up this section by showing that the proper number of clusters, at least for K-means, can sometimes be determined approximately by looking at the SSE curve. Figure 5.49 shows the plot of SSE for a (bisecting) K-means clustering of the ten clusters shown in Figure 5.16. While the knee is particularly sharp for this data, often this is not the case, i.e., the change in SSE can be much more gradual. This is illustrated by Figure 5.50, which shows the SSE curve for the points in Figure 5.33a. However, in many cases this approach does provide a rough estimate of the number of clusters.



(a) Well-Separated Clusters



(b) Similarity Matrix Sorted by K-means Cluster Labels

Figure 5.47. Similarity Matrix for Well-separated Clusters.

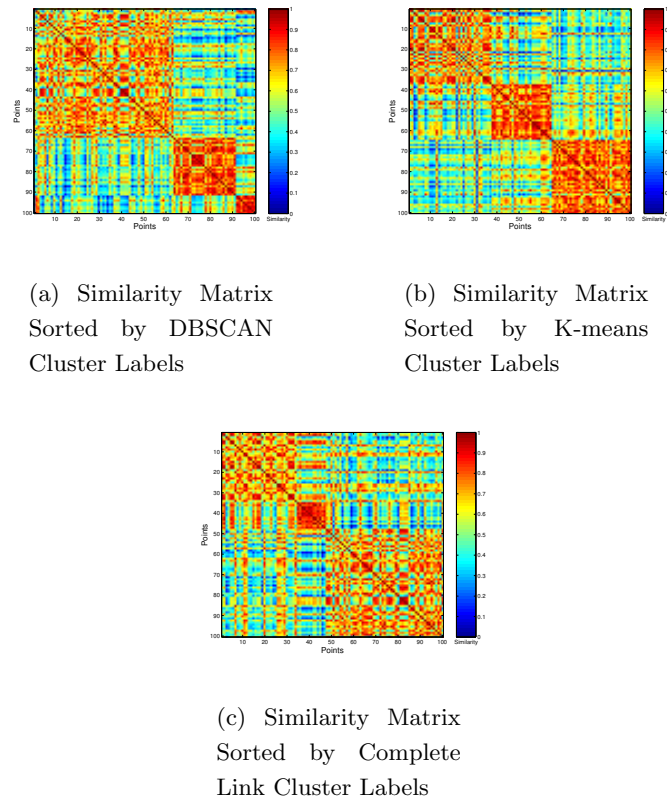


Figure 5.48. Similarity Matrices for Clusters from Random Data.

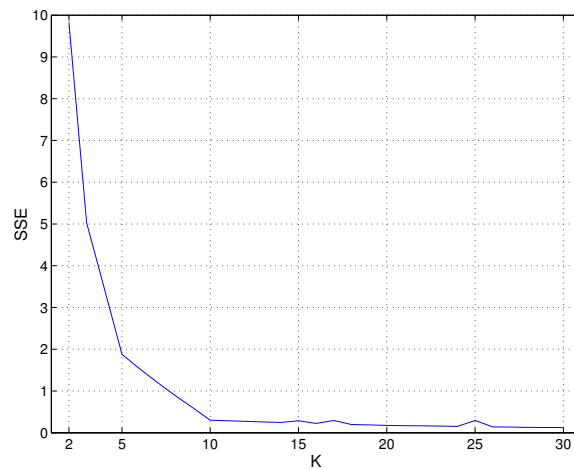


Figure 5.49. Plot of SSE versus K for 10 Clusters.

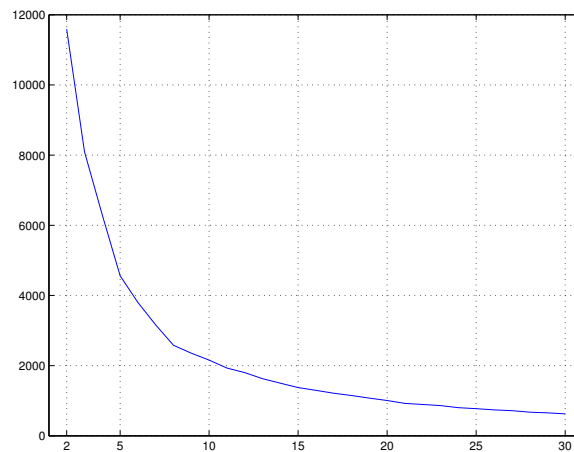


Figure 5.50. Plot of SSE versus K for a more complicated data set.

5.11.5 A statistical framework for cluster validity

However, suppose that we have clustered a set of data using K-means, perhaps with multiple runs, bisecting K-means, and/or postprocessing, and we feel that we have attained the lowest SSE clustering for K clusters that is possible given our tools and computational resources. Is there any way that we can more objectively assess how good the clusters are using SSE? The answer is ‘yes.’ In some cases, we can obtain a statistical evaluation of how good our clusters are with respect to the same number of clusters on random data.

We illustrate this with an example. Suppose that we wanted an objective measure of how good our clusters were with respect to random data for the well-separated clusters of Figure 5.47. The idea is simple: we generate many random sets of 100 points having the same range as the points in the three clusters, find three clusters in each data set using K-means and then accumulate the distribution of SSE for these clusterings. We can then compare the SSE for our original three clusters and see how ‘likely’ it is for a set of clusters with this SSE to occur. Figure 5.51 shows the histogram of the SSE that results from 500 random runs. The lowest SSE shown is 0.0173. For the three clusters of Figure 5.47, the SSE is 0.0050. Informally, we could then conservatively say that there is a less than a 1% chance that a clustering such as that of 5.47 could result by chance.

5.11.6 Internal measures of cluster validity: cohesion and separation

Internal measures of cluster validity can often be separated into two important classes: measures of cluster cohesion (compactness, tightness, which measure how closely related the objects in a cluster are, and measures of cluster separation (isolation), which measure how distinct or well-separated a cluster is from other clusters. Notice that these measures are often more appropriate for center based clusters, since contiguous or density-based clusters can consist of points which are not all that similar to one another and not all that isolated from other points in other

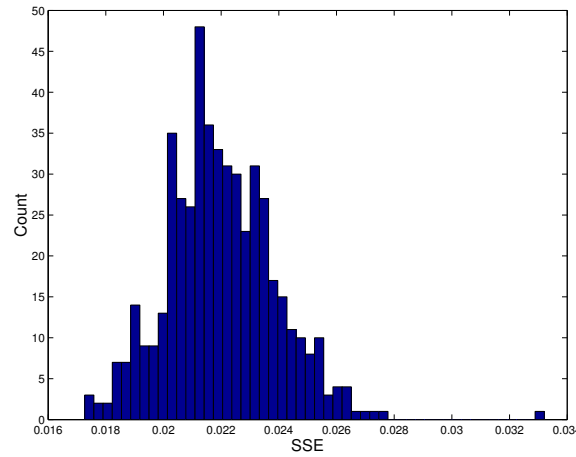


Figure 5.51. Histogram of SSE for 500 random data sets.

clusters.

SSE and SSB

The SSE measure that we have seen a number of times before provides a natural measure of cluster cohesion. As mentioned, the proper measure is actually the mean SSE of a cluster, i.e., the SSE of a cluster divided by the number of points in the cluster. To measure cluster separation, we can rely on a similar measure, the between cluster sum of squares, SSB, which is the sum of the squared distance of a cluster centroid, m_i , to the overall mean, m , of the data points. By summing the SSB over all clusters, we can obtain a total SSB, as given by equation 5.42. m_j^i is the j^{th} component of the i^{th} mean, while m_j is the j^{th} component of the overall mean.

$$SSB = \sum_{i=1}^K |C_i| \sum_{j=1}^n (m_j^i - m_j)^2 \quad (5.42)$$

The higher the total SSB of a clustering, the more separated the clusters are from one another. Once again, though, to evaluate the isolation of an individual cluster, we only consider the portion of the SSB due to one cluster, and should also divide this quantity by the number of points in the cluster to adjust for cluster size. In statistics the SSE is often called the “within cluster sum of squares,” while the SSB is referred to as the “between cluster sum of squares.” It is a straightforward exercise to shown that the sum of the total SSE and the total SSB is a constant, i.e., it is equal to the sum of squares of the distance of each point to the overall data mean (centroid).

A graph based approach

Another way to define cluster cohesion and separation is via a graph based approach. Note that this approach is more relevant for contiguity-based clusters. Specifically

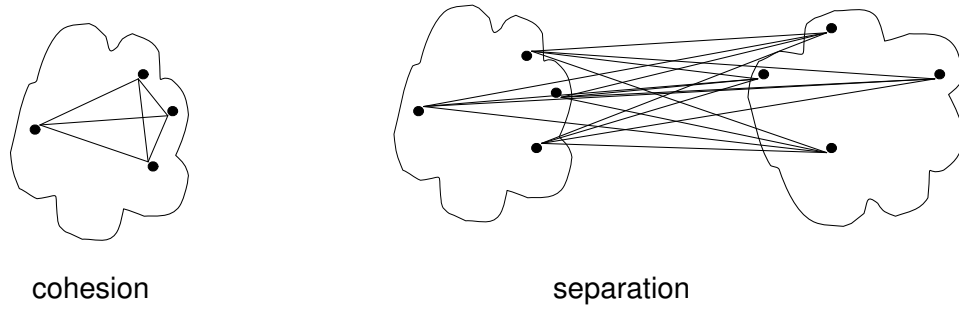


Figure 5.52. Illustration of proximity links involved in cohesion and separation measures.

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. (Recall that the proximity graph is the graph that has data objects as nodes, a link between each node, and a weight assigned to each link that is the proximity between the two nodes (data objects) involved in the link.) Likewise, the isolation can be measured by the sum of the weights of the links that are from points in the cluster to points outside the cluster. This is illustrated in Figure 5.52. To simplify this measure, often only the k nearest neighbors of a point are considered. Furthermore, especially when only the nearest neighbors are considered, each link is sometimes assigned a weight of 1.

Silhouette Coefficient

We briefly present the method of silhouette coefficients. This approach consists of the following three steps. To be specific, we use distances, but a similar 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 quantity a_i .
2. For the i^{th} object, calculate its average distance to each object of a given cluster, and then find the minimum such value. Call this quantity b_i .
3. For the i^{th} object, compute the silhouette coefficient. $s_i = \begin{cases} 1 - \frac{a_i}{b_i} & \text{if } a_i \leq b_i \\ \frac{a_i}{b_i} - 1 & \text{if } a_i > b_i \end{cases}$

We can then construct what is called a silhouette plot. To do this, we order the data points in a plot in terms of decreasing silhouette coefficient. The points at the top of the plot are the ones that are more in the center of the cluster, while those at the bottom are more towards the edge.

Notice that we can also compute the average silhouette coefficient of a cluster, which is a measure of the goodness of a cluster, both in terms of its cohesion and its separation. An overall measure of the goodness of a clustering can be obtained by calculating the weighted average of the silhouette coefficients of all the clusters, where the weight is based on cluster size.

5.11.7 Using External Measures of Cluster Validity

Typically, when we have external information, it is externally derived class labels for the data points, and in such cases, it is desired to measure the degree of fit between the cluster labels and the class labels. The reader might wonder why this is of interest. After all, if we have the class labels, then what is the point in performing a cluster analysis. There are various reasons, e.g., comparison of clustering techniques or parameterizations of a technique with the “ground truth,” and the evaluation of the extent to which a manual classification process can be automatically derived by cluster analysis.

There have been a large number of measures developed by members of the statistics and classification communities for comparing two partitions of data. As far as we can tell, very few of them are used in the data mining community. We consider only two measures that are widely used:

entropy 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 j belongs to class i as follows: $p_{ij} = m_{ij}/m_j$, where m_j is the number of values in cluster j and m_{ij} is the number of values of class i in cluster j . Then using this class distribution, the entropy of each cluster j is calculated using the standard formula, $e_j = \sum_{i=1}^L p_{ij} \log_2 p_{ij}$, where the 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., $e = \sum_{j=1}^K \frac{m_j}{m} e_j$, where m_j is the size of cluster j , K is the number of clusters, and m is the total number of data points.

purity Using the terminology derived for entropy, the purity of cluster j , is given by $\text{purity}_j = \max_i p_{ij}$ and the overall purity of a clustering by $\text{purity} = \sum_{j=1}^K \frac{m_j}{m} \text{purity}_j$.

To illustrate these two measures, we consider an example of using K-means to cluster documents. For documents we use cosine similarity as the measure of document similarity and transform the raw document-term matrix by first performing the TFIDF normalization, and then normalizing all document vectors to have an L_2 norm of 1. As our sample dataset, we chose a data set that contains 3204 documents and 31,472 terms. The documents which are articles from the LA times and come from 6 different classes: ‘Entertainment,’ ‘Financial,’ ‘Foreign,’ ‘Metro,’ ‘National,’ and ‘Sports.’

The following table indicates the results of a K-means run to find 6 clusters. (The following analysis can also be performed when the number of clusters does not match the number of classes.) The first column indicates the cluster, 1-6, while the next six columns are the confusion matrix, e.g., column 3 records how many documents of each cluster were from the Financial section of the newspaper. The last two columns are, respectively, the entropy and purity of each cluster.

Table 5.9. K-means Clustering Results for LA 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

Notice that one cluster is exceptionally pure, Cluster 3, which contains mostly documents from the Sports section. The purity of the other clusters is not exceptional, but purity can typically be greatly improved by finding more clusters.

5.12 Bibliographic Notes

Quite a number of books and articles on clustering have been published - only a few of which can be mentioned here. Three of the most well-known introductions to clustering are [12], [93], and [101]. A more statistically oriented approach to clustering is given in Chapter 10 of [49]. A relatively up-to-date, general survey of clustering can be found in [94], while more data mining specific surveys of clustering are contained in [75] and [18]. A good source of references to clustering outside of the data mining field can be found in [15].

Center-based clustering approaches, such as K-means and K-medoids, are a core set of clustering techniques. The K-medoid approach is discussed in [101] and [136], while K-means the algorithm is described in detail in [12] and [93]. The bisecting K-means algorithm is described in [175] and an implementation of this and other clustering approaches is freely available for academic use in the CLUTO (CLustering TOOLkit) package [67].

Thorough discussions of hierarchical clustering are found in [12], [93], and [101]. The main focus is traditional agglomerative hierarchical clustering approaches, but divisive approaches are also described in [93], and [101]. More recent developments of hierarchical clustering include the use of representative points (CURE [177]), the importance of the proper similarity measure for transaction data (ROCK [65]) and the value of dynamically modelling the data (Chameleon [99]). There has also been recent work ([58] and [100]) in trying to alleviate the brittleness of traditional agglomerative hierarchical clustering.

Density-based clustering as embodied by DBSCAN is described in [54] and [155]. Extension of this work has led to OPTICS (Ordering Points To Identify the Clustering Structure) [14], a new approach for visualizing cluster structure and to LOF (Local Outlier Factor) [23], an approach for classifying the degree to which a point is an outlier.

MAFIA and CLIQUE, which were discussed in sections 5.8.3 and 5.8.3 are examples of subspace clustering algorithms. The CLIQUE algorithm is described in more detail in [5], while more information on MAFIA can be found in [135]. While the presentation in sections 5.8.3 and 5.8.3 emphasized the density based nature of CLIQUE and MAFIA, they are also examples of algorithms of grid-based clustering algorithms. Other examples of clustering algorithms that are grid-based are WaveCluster [162], DENCLUE [82], and OptiGrid [83].

AutoClass [CS96] is one of the most prominent of the mixture-model based programs and has been used for data mining to some extent. A readable and current technical report on clustering using mixture models is given by [FR98]. This source also provides a pointer to state-of-the-art mixture modeling software. An approach to using mixture models for co-occurrence data (frequent itemsets of size 2) is given in [HP98].

[93] provides a thorough discussion of cluster validity, while [101] introduces the notion of silhouette coefficients that we saw in Section 5.11.6.

5.13 Exercises

1. Identify the clusters in the following diagram using the center-based, contiguous and density 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 = K-means, contiguous = MIN, and density = DBSCAN.

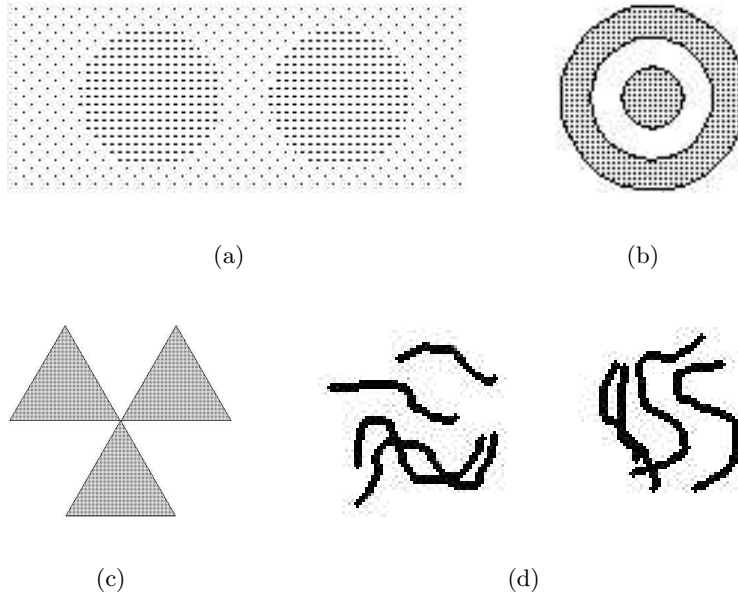
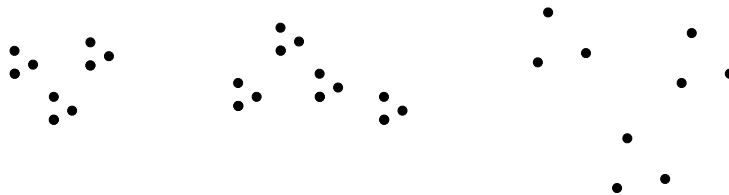


Figure 5.53. Figures for exercise 1.

2. Find all well separated clusters in the following set of points.



3. The following attributes are measured for members of a typical herd of Asian Elephants: *weight*, *height*, *tusk length*, *trunk length*, and *ear area*. Based on these measurements, What sort of similarity measure from Section 5.2 would you use to compare or cluster these elephants? Justify your answer and explain any special circumstances.
4. Similarity measures.
 - (a) For binary data, the L1 distance corresponds to the Hamming distance,

i.e., 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.

$a = 0101010001$

$b = 0100011000$

- (b) Which approach, Jaccard or Hamming distance, is more similar to the Simple Matching Coefficient and which approach is more similar to the cosine measure? Explain. (Note: the Hamming measure is a distance, while the other three measures are similarities, but don't let this confuse you)
 - (c) Suppose that you are comparing how similar two organisms of different species are in terms of the number of genes that 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 the each animal is represented as a binary vector, where each attribute is 1 if a particular gene is present in the organism and 0 otherwise.)
 - (d) If you wanted to compare the genetic makeup of two organisms of the same species, e.g., two human beings, would you use the Hamming distance, the Jaccard coefficient or a different measure of similarity or distance? Explain. (Note that two human beings share $> 99.9\%$ of the same genes.)
5. 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 both clusters of documents and words for a document data set.
- (a) How might a set of word clusters defined by the top terms in a document cluster might differ from the word clusters found using K-means clustering of terms?
 - (b) Under what ideal (and unrealistic) circumstances, would the set of word clusters defined by the top terms in a document cluster and the set of document clusters produced by K-means be identical?
 - (c) How could term clustering be used to define clusters of documents?
6. Use the following similarity matrix to perform MIN and MAX hierarchical clustering. Show your results by drawing a dendrogram. The dendrogram should clearly show the order in which the points are merged.

	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

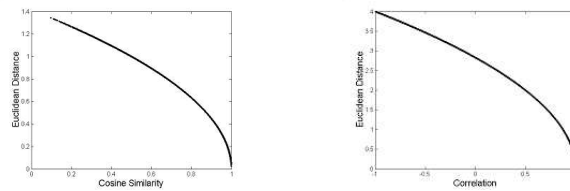
7. Similarity Measures.

- (a) For the following vectors, x and y , calculate the indicated similarity or distance measures.
 - i. $x = (1\ 1\ 1\ 1)$, $y = (2\ 2\ 2\ 2)$ cosine, correlation, Euclidean
 - ii. $x = (0\ 1\ 0\ 1)$, $y = (1\ 0\ 1\ 0)$ cosine, correlation, Euclidean, Jaccard
 - iii. $x = (0\ -1\ 0\ 1)$, $y = (1\ 0\ -1\ 0)$ cosine, correlation, Euclidean
 - iv. $x = (1\ 1\ 0\ 1\ 0\ 1)$, $y = (1\ 1\ 1\ 0\ 0\ 1)$ cosine, correlation, Jaccard
 - v. $x = (2\ -1\ 0\ 2\ 0\ -3)$, $y = (-1\ 1\ -1\ 0\ 0\ -1)$ cosine, correlation
- (b) What is the range of values that are possible for the cosine measure?
- (c) True or false, if two objects have a cosine measure of 1, are they identical? Explain.
- (d) 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.)
- (e) Figure 5.54a 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?
- (f) Figure 5.54b 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?

8. Understanding the behavior of K-means and MIN

Hierarchical clustering is sometimes used to generate K clusters, $K \geq 1$ 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) Relationship between Euclidean distance and the cosine measure

(b) Relationship between Euclidean distance and correlation

Figure 5.54. Figures for exercise 6.

- (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 MIN?
 - (d) Which technique, K-means or MIN, 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?
9. K-means and clusters of varying density Suppose that a data set has
- m points and K clusters and
 - half the points and clusters are in "more dense" regions and
 - half the points and clusters are in a "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 “more dense” region.

Note: please don’t 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, please justify your response.

10. Sampling to improve cluster efficiency.

Hierarchical clustering algorithms require at least $O(m^2)$ time and thus, are impractical to use directly on larger data sets. One possible technique for reducing the time required is to sample the dataset. For example, if K clusters are desired, then points are sampled from the m points, and a hierarchical clustering algorithm will produce a hierarchical clustering in $O(m)$ time. K clusters can be extracted from this hierarchical clustering by looking at the clusters on the K^{th} level of the dendrogram. The remaining points can then be assigned in linear time, by using a variety of strategies. For example, the centroids of the K clusters can be calculated and then each of the m points can be assigned to the closest cluster.

For each of the following types of data or clusters, discuss briefly if a) sampling will cause problems and b) what those problems are. Assume that the sampling technique randomly chooses points from the total set of m points and that the 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.

11. Understanding the behavior of K-means and MIN

Consider the following four “faces” shown in figure 5.55. Again, darkness or number of dots represents density.

- (a) Could you use MIN to find the patterns represented by the nose, eyes and mouth in all figures? Explain.

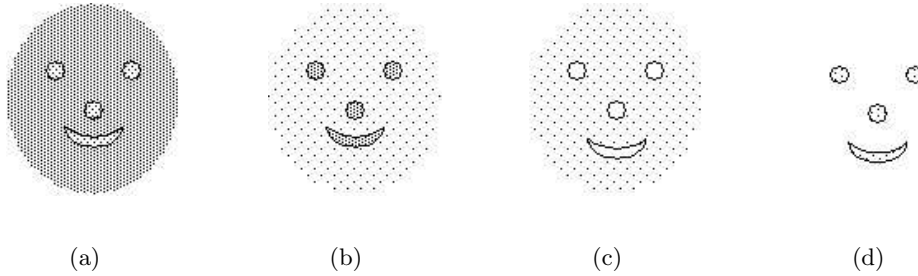


Figure 5.55. Figures for exercise 10.

- (b) Could you use K-means to find the patterns represented by the nose, eyes and mouth in the figures? Explain.
- (c) What limitation does clustering have in detecting all the patterns formed by patterns of points formed by Figure 5.55c?

12. K-means Clustering.

For the following sets of two-dimensional points, provide a) a sketch of how they would be split into clusters by k-means for the given number of clusters and b) indicate approximately where the resulting centroids would be. Assume that we are using the squared error objective function. If you think there is more than one possible solution, then please indicate for each solution whether it is a global or local minimum.

- (a) $K = 2$. Assuming that the points are uniformly distributed in the circle, how many possible ways of partitioning the points into two clusters are there (in theory)? What can you say about the positions of the resulting two centroids? (Again you don't need to provide exact centroid locations, just a qualitative description.)
- (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.

13. Match the following similarity matrices, which are sorted according to cluster labels, with the sets of points that are shown. Clusters are shown in the figures by differences in color and shape of the markers. There are 100 points in each figure.

14. Compute the entropy and purity for the following confusion matrix.

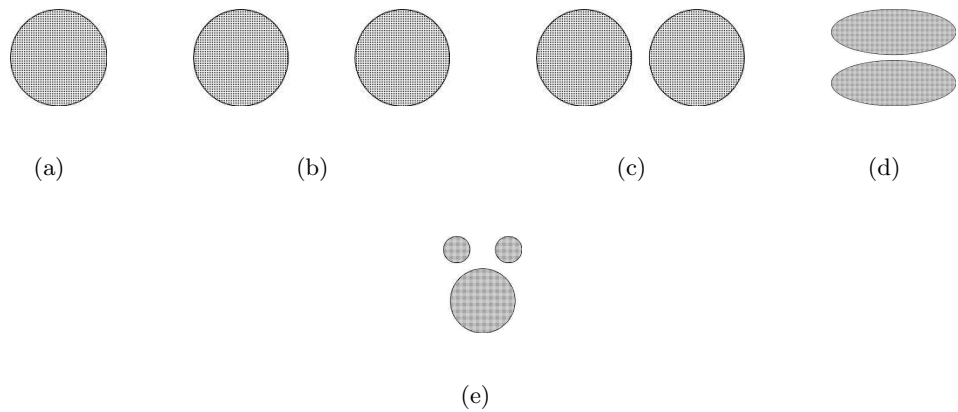


Figure 5.56. Figures for exercise 11.

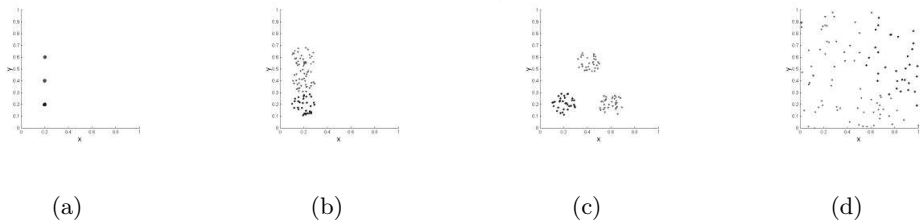


Figure 5.57. Figures for exercise 12.

15. Given the following similarity matrix and set of cluster labels, compute the correlation between the similarity matrix and the incidence matrix.
16. Using the data in problem 14, compute the silhouette coefficient for each point, each of the two clusters, and the overall clustering.
17. F measure for evaluating a hierarchical clustering

An external quality measure for hierarchical clustering is the F measure, a measure that combines the ideas of precision and recall. We treat each cluster as if it were the result of a query and each class as if it were the desired set of documents for a query. We then calculate the recall and precision of that

Table 5.10. Confusion matrix for exercise 13

Cluster	Entertainment	Financial	Foreign	Metro	National	Sports	Total	Entropy	Purity
#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		

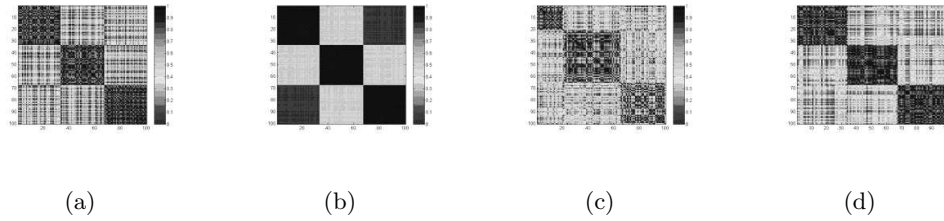


Figure 5.58. Figures for exercise 12.

Table 5.11. Table of cluster labels for Exercise 14

Point	Cluster Label
P1	1
P2	1
P3	2
P4	2

cluster for each given class. More specifically, for cluster j and class i

$$Recall(i, j) = n_{ij}/n_i$$

$$Precision(i, j) = n_{ij}/n_j$$

where n_{ij} is the number of members of class i in cluster j , n_j is the number of members of cluster j and n_i is the number of members of class i .

The F measure of cluster j and class i is then given by

$$F(i, j) = (2 * Recall(i, j) * Precision(i, j)) / (Precision(i, j) + Recall(i, j))$$

For an entire hierarchical clustering the F measure of any class is the maximum value it attains at any node in the tree and an overall value for the F measure is computed by taking the weighted average of all values for the F measure as given by the following.

$$F = \sum_i \frac{n_i}{n} \max F(i, j)$$

where the max is taken over all clusters at all levels, and n is the number of documents.

Compute the F measure for the eight objects {p1, p2, p3, p4, p5, p6, p7, p8} and hierarchical clustering shown below. p1, p2, and p3 belong to class A,

Table 5.12. Similarity matrix for Exercise 14

Point	P1	P2	P3	P4
P1	1	0.2	0.35	0.45
P2	0.2	1	0.3	0.4
P3	0.35	0.3	1	0.1
P4	0.45	0.4	0.1	1

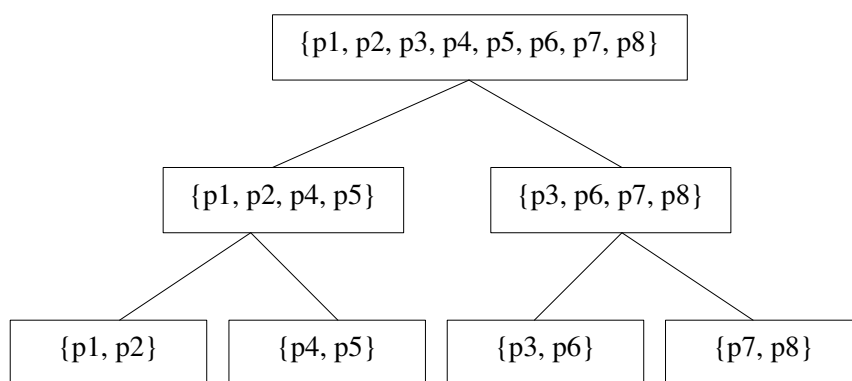


Figure 5.59. Cluster tree for exercise 16.

while $p4, p5, p6, p7$, and $p8$ belong to class B.)

Visualization

6.1 Introduction

Visualization is the process of converting data (information) into a visual format so that the characteristics of the data and the relationships among data items or attributes can be analyzed or reported. In everyday life, visual techniques such as graphs and tables are often the preferred approach used to explain, for example, 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 often play a key role in these fields. Regardless of the application, visualization takes advantage of the well-known abilities of human beings to rapidly interpret large amounts of visual information.

The goal of this chapter is to provide an introduction to the usefulness of visual techniques for exploring data, both for understanding the data beforehand and for detecting patterns - visual data mining if you will. To that end, we first discuss some basic concepts of visualization, and then present a number of visualization techniques that are widely applicable. Nonetheless, the reader should be aware that visualization is a broad topic, and we will only be able to explore a small (but hopefully quite useful) part.

6.1.1 What is Visualization?

Data visualization is the display of information in a graphic or tabular format, either on the printed page or on a display. One important component of this process is the mapping of information to a visual format. This involves mapping the objects, attributes, and relationships involved in a set of information onto to visual objects, attributes and relationships. For example, consider Figure 6.1, which shows the adjusted (for inflation—1995-1996 is the baseline) and unadjusted Gross Domestic Product (GDP) of New Zealand from 1990 to 1999. The objects correspond to the economic information for a year, while the attributes of each object are the year, the unadjusted GDP, and the adjusted GDP.



Figure 6.1. The Gross Domestic Product of New Zealand (1990-1999), both adjusted for inflation (the baseline) and unadjusted.

Another key component of visualization is the interpretation of the visualized information by the observing human being and the formation of a mental model of the information. For example, upon seeing Figure 6.1, most people would quickly interpret the data as consisting of two GDP time series. Furthermore, they also would rapidly notice relationships across objects such as the overall upward trend of GDP in each individual time series. It might take a bit longer to notice the relationships between attributes, such as the fact that the growth in the inflation adjusted GDP is always less than that of the unadjusted GDP.

6.1.2 Motivations for Visualization

The overriding motivation for using visualization is that people are very good at quickly absorbing large amounts of visual information and finding patterns in such information. For instance, consider Figure 6.2, which shows the Sea Surface Temperature (SST) in Celsius for July, 1982. This one picture summarizes the information in about 250,000 numbers and is readily interpreted in a few seconds. For a person to try to interpret the data in any other way, would require far more time, and patterns, such as the boundaries between areas of different temperature, would not be so easy to detect.

Another very general motivation for visualization is to make use the domain knowledge that is ‘locked up in people’s heads.’ While the use of domain knowledge is a very 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 best way of finding patterns of interest since, by using domain knowledge, a person can often quickly eliminate many uninteresting patterns and direct focus to the patterns that are of interest.

Even if visualization is not sufficient for the complete analysis, often it is a useful supplementary technique. Thus, visualization is often used as part of data exploration in order to get an idea of the nature of the data. For instance, by plotting a single attribute, we can determine if the values of that attribute have a

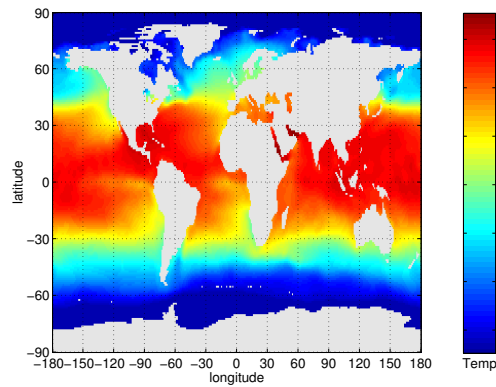


Figure 6.2. Sea Surface Temperature (July, 1982).

well-know distribution, e.g., normal, what the range of values are, etc. Additionally, plotting pairs of attributes may reveal facts about the relationships of variables, e.g., may show that two attributes strongly related. Furthermore, visualizing data as a preliminary step may reveal problems with the data, e.g., a person may realize that one attribute has a lot of values that are -1 and further investigation may show that this indicates a missing value that wasn't clearly identified in the data description.

6.1.3 Visualization and Different Types of Data

As we saw in Chapter 2, there are a number of different types of data, i.e., record (matrix), graph, and ordered, and these different types of data have different characteristics, e.g., sparsity or the number of dimensions. The type of data greatly affects the way in which information is plotted. For example, time series data is often plotted as a line graph (see Figure 6.1, when only a few time series are involved. However, if many time series are involved and these time series have a spatial relationship to one another, e.g., the sea surface temperature at each point on the ocean, then we may choose to display only a slice of the data, e.g., the temperature at each point on the ocean for a particular month—see Figure 6.2.

Dimensionality has a very strong effect on the type of visualization. Specifically, while data objects that have only one, two or three dimensions, and can often be easily mapped directly to a two- or three-dimensional graphical representation, it is often not immediately clear how multi-dimensional data should be visually represented. Thus, often visualization techniques are classified with respect to the dimensionality of the data to which they can be applied—1, 2, 3, or higher dimension.

The type of attributes are also important. For example, a continuous attribute can be associated with the x , y , or z co-coordinate of a plot, while a nominal attribute, since its values lack a natural order, does not lend itself to such a representation. On the other hand, the values of a categorical attribute can easily be associated with separate columns in a table, while a continuous variable would need to be discretized. As another example, for data that represents information about

physical objects, e.g., the velocity of air in the atmosphere, often our data objects are two- or three-dimensional locations that have attributes associated with them, e.g., speed and direction, and thus, visualization techniques for such data typically map each physical point to point on the image. As an example of a more abstract type of data, graph data, e.g., the links between a set of web pages, might be best displayed by using graph drawing techniques, which try to display the nodes and links of a graph—often an inherently high-dimensional object—in a way that does not obscure potentially useful information, such as groupings of highly linked web pages.

6.1.4 General Categories of Visualization

The following discussion hints at the fact that visualization techniques are often specialized to the type of data. 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 types of data visualization tasks. Despite this specialization and the *ad hoc* nature of visualization, researchers have nonetheless developed some generic classes of visualization techniques. Once such classification is strictly based on the type of data:

- one-dimensional
- two-dimensional
- three-dimensional
- multi-dimensional
- hierarchical
- graph
- other

Another sort of categorization, which is based on generic classes of applications, is the following:

scientific visualization The basic data concerns physical objects, e.g., the atmosphere or a kidney.

information visualization The data is not physical, but rather is more conceptual or symbolic, e.g., a set of documents or the data describing the structural and dynamic characteristics of a communications network.

statistical graphics Typical multivariate continuous or categorical data of the type traditionally associated with statistics, such as the results of controlled experiment on the effect of a drug or the results of a survey.

Notice that last set of three categories may overlap to some extent and that these three categories are roughly independent of the previous set. Other categorizations are also possible, e.g., a distinction is frequently made between techniques used for categorical and continuous data.

In what follows, we will present graphical techniques using the first set of categories, i.e., one-dimensional, etc., while at the same time attempting to clearly state the applicability of each technique, e.g., “this technique is only useful for categorical data.”

6.2 General Concepts

In this section, we will explore some of the general concepts related to visualization. Mostly, we will consider general approaches for mapping the data and its attributes to visual characteristics, including issues related to the arrangement of visual objects. We will, however, also consider some general *do's and don'ts* for ‘good’ visualization.

We briefly discuss a number of visualization techniques, but postpone illustrating them until we talk about specific approaches later in this chapter. We will however, assume that the reader is familiar with line graphs, bar charts and scatter plots. If not the reader, may wish to postpone reading the following sections until have read the section on specific visualization techniques.

6.2.1 Representation: Mapping Data to Graphical Elements

The first step in visualization is to map data objects, their attributes, and the relationships between data objects to graphical elements. Objects are usually represented in one of three ways. First, if only a single categorical variable 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 as an entry in a table or an area on a screen (consider a bar charts). Secondly, if an object has multiple attributes, then the object can be displayed as a row (or column) of the table or as a line on a graph. Finally, the object is often interpreted as a point in two- or three-dimensional space, where graphically, the point might be a small circle, cross, box, etc.

For single 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: location along the x , y , or z axes, intensity, color, or size (diameter, width, height, etc.). For categorical attributes, we can use distinct graphical features, e.g., position, color, shape, orientation, embellishments, etc., to indicate the different categories, but care should be taken not to read any additional meaning into graphical features, such as color and position, which have an inherent ordering associated with their values.

Representing relationships via graphical elements occurs either explicitly or implicitly. If we have graph data, then 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, as indicated in the previous paragraph. To illustrate, if the nodes are cities and the links are highways, then the links of the cities might represent population, while the width of the links might represent the volume of traffic on the highway.

In most cases, though, the relationships in the data are implicitly mapped to relationships between graphical elements when the objects and attributes are mapped to graphical elements. In the simplest example, if the data object represents a physical object that has a location, e.g., a city or a part of a part of physical object, 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, then if these attributes are taken as the coordinates of the data points, the resulting graph often gives considerable insight into the relationships of the attributes and the data points.

However, in general, it is hard to ensure that a mapping of objects and attributes will also result in the relationships being mapped to easily observed relationships among graphical elements. We will make some specific comments on this issue as we examine specific visualization techniques, but for now, we merely observe that this is one of the more challenging aspects of visualization. In other words, given a set of data, there are many implicit relationships in the data; thus, a key task of visualization is to choose a technique that makes the relationships of interest easily visible.

6.2.2 Arrangement

In the previous section it was mentioned representation of objects and attributes was important for good visualization. The arrangement of items within the visual display is also crucial. We will illustrate this with three examples: one involving tables, one involving graphs, and one involving three dimensional data.

We begin by illustrating the importance of rearranging a table of data. In Table 6.2.2, which shows nine objects with six binary attributes, there is not clear relationship among object and attributes, at least at first glance. However, if we permute the rows and columns of this table as shown in Figure 6.2.2, 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.

As another example of the value of arrangement, consider Figure 6.3, which shows a graph. If the connected components of the graph are separated, as in Figure 6.4, then the relationships between nodes and graphs become much simpler to understand.

6.2.3 Selection

Another key concept in visualization is selection, which is the elimination or the de-emphasis of certain objects and attributes. Consider, for example, if the data objects

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

	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

Table 6.2. A table of nine objects (rows) with six binary attributes (columns) permuted so that the relationships of rows and columns is clear.

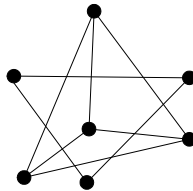


Figure 6.3. A generic graph: nodes are objects, links represent relationships.

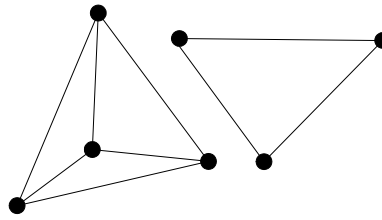


Figure 6.4. A generic graph: nodes are objects, links represent relationships.

being considered have many attributes, e.g., 10, then it becomes difficult to visualize all the attributes at the same time. (In section ??, we discuss ways to visualize many attributes, but such solutions are only partially satisfactory.) Likewise, if there are many objects, visualizing all the objects may result in a display that is too crowded.

The most common approach to handling many attributes is *projection*, i.e., only a subset of attributes—usually two—is chosen for display. Such an approach can be implemented manually; if the dimensionality is not too high, a matrix of such bivariate-plots can be constructed for simultaneous viewing. Alternatively, a visualization program can automatically show a series of two-dimensional projections, where the sequence is based on some strategy. In both cases, the hope is that visualizing a collection of data ‘slices’ will give a more complete view of the data.

When the number of data points is high—in some cases more than a few hundred—or if the range of the data is large, a visualization often becomes less useful. Some data points obscure other data points, or a data object may not occupy enough pixels to allow its features to be clearly displayed, e.g., the shape of an object can not be used to encode a characteristic of the object if only one pixel can be used to display the object. In these situations, it is useful to be able to eliminate some of the objects, either by ‘zooming’ in on a ‘region’ of the data, or by simply taking a sample of the data points.

6.2.4 Do's and Don'ts

The following is a short list of visualization do's and don'ts. While most of these guidelines should make a lot of sense, they should not be followed blindly, i.e., as always, guidelines are no substitute for thoughtful consideration of the problem at hand.

The following are the *ACCENT* principles for effective graphical display which were put forth by D. A. Burn.

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?

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.

6.3 Visualization Techniques

In this section, we describe specific visualization techniques. As mentioned above we will pursue the techniques in the following order:

- one-dimensional
- two-dimensional
- three-dimensional
- multi-dimensional
- hierarchical
- graph
- other

Most of these techniques are available in a wide variety of mathematical and statistical packages. Also, there are a number of publicly available data sets available on the World Wide Web. Thus, we encourage the reader to actually try these visualization techniques for themselves as they proceed the following sections.

A word about data. In the following illustrations of plots, we will use several different data sets. We describe these data sets briefly.

One of data sets that we use is the Iris data set. It consists of information on 150 iris flowers from three different types of irises—50 flowers from each type of iris. Each flower is characterized by five attributes:

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

6.3.1 Visualizing One-dimensional Data

Stem and Leaf Plots

Stem and leaf plots can be use 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 for 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 our values are two digit integer, e.g., 35, 36, 42, and 51, then the stems will be the high-order digits, e.g., 3, 4 and 5, while the leafs are the low order digits, e.g., 1, 2, 5, and 6. By plotting the stems vertically and leafs horizontally, we can provide a visual representation of the distribution of the data.

To illustrate consider the following set of integers, which 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.

Figure 6.5. Sepal length data from the Iris data set.

```
43 44 44 44 45 46 46 46 46 47 47 48 48 48 48 48 49 49 49 49 49 49 50 50 50 50 50 50 50 50
50 51 51 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 55
56 56 56 57 57 57 57 57 57 57 57 58 58 58 58 58 58 58 59 59 59 60 60 60 60 60 60 61 61 61 61
61 61 62 62 62 62 63 63 63 63 63 63 63 63 64 64 64 64 64 64 64 65 65 65 65 65 66 66 67 67
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
```

The steam and leaf plot for this data is shown in Figure 6.6. Each number in Figure 6.5 is first put into one of the vertical groups—4, 5, 6, or 7— according to its

Figure 6.6. Stem and leaf plot for the sepal length from the Iris data set.

```

4 : 34444566667788888999999
5 : 000000000011111111222234444445555556666667777777888888999
6 : 00000011111122223333333334444444555556677777778889999
7 : 0122234677779

```

Figure 6.7. Stem and leaf plot for the sepal length from the Iris data set.

```

4 : 3444
4 : 566667788888999999
5 : 00000000001111111122223444444
5 : 55555556666667777777888888999
6 : 00000011111122223333333334444444
6 : 555556677777778889999
7 : 0122234
7 : 677779

```

tens 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,' we repeat the stem '4' twice, putting all values 40-44 in the bucket corresponding to the first stem and all values 45-49 in the bucket corresponding to the second stem. This approach is shown in the stem and leaf plot of Figure 6.3.1. Other variations are also possible.

Dot Plots

A dot plot is a variation on a stem and leaf plot, where we do not show the digits in each bucket, but rather, only show a dot for each data object that falls in each bucket. A dot plot for the sepal length data is shown in Figure 6.8.

Histograms

Histograms are a way of representing the distribution of values for attributes with numerical values. The idea is to divide the range of values into bins—typically, but not necessarily, of equal width—and to count the number of values in each bin. A *bar plot* is then constructed such that each bin is represented by one bar and the area of each bar is proportional to the number of values (data objects) that fall into the corresponding range.

If all intervals are equal width, then all bars are the same width and the height is proportional to the number of values. If the attribute is categorical, then often one bin is allotted to each value, unless the number of values is large.

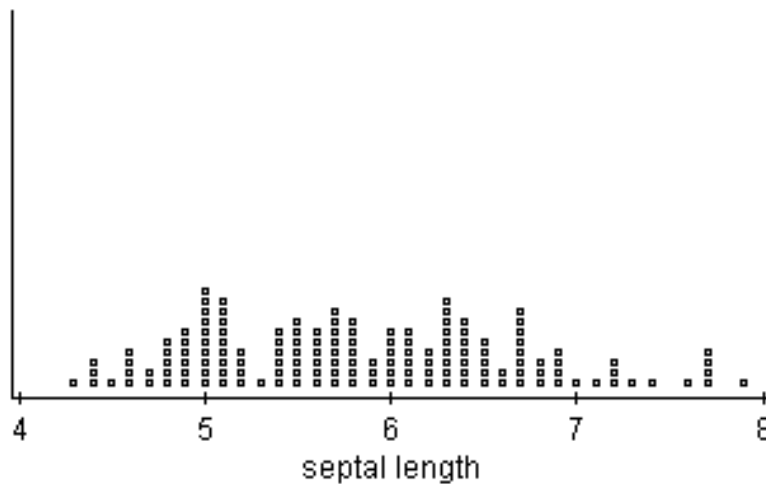


Figure 6.8. Dot plot for the sepal length from the Iris data set

To illustrate this type of plot, we have constructed a variety of histograms using sepal width, sepal length, petal width, and sepal length in Figure 6.9. Since the shape of a histogram depends on the number of bins, we show the same histograms first with 10 bins in Figure 6.9, and then with 20 bins in Figure 6.10.

There are also some variations of the histogram as well. A relative (frequency) histogram replaces the count by the relative frequency. However, this is just a change in scale of the y axis, and the shape of the histogram does not change. Another common variation, especially for categorical data, is the Pareto histogram, which is the same as a normal histogram except that the categories are sorted horizontally in decreasing order of count from left to right.

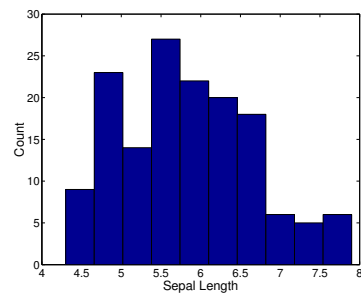
Tukey Box Plots

Box plots are another way of showing the distribution of the values of a single numerical attribute. Figure 6.11 shows a labelled plot. The lower and upper ends of the box indicate the 25th and 75th percentiles, respectively, while the middle line in the box indicates the values of the 50th percentile. The top and bottom lines of the ‘tails’ indicate the 10th and 90th percentiles. Outliers are shown by ‘+’ marks.

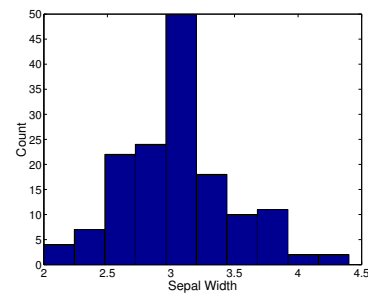
We give a specific examples of box plots using, once more, the first four attributes of the Iris data set. The box plots for these attributes are shown in Figure 6.12. Notice that since box plots are relatively compact, many of them can be shown on the same plot. Simplified versions of the box plot, which take less space, can also be used.

Percentile Plots and Empirical Cumulative Distribution Functions

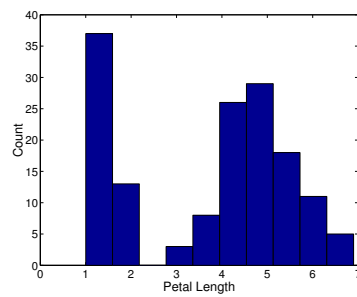
A type of diagram that is more quantitative in terms of showing the distribution of the data is the plot of an empirical cumulative distribution function (ECDF). While this may sound complicated, the concept is straightforward. In statistics, a



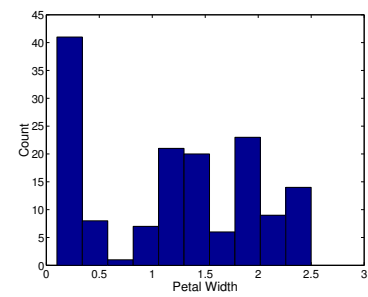
(a) Sepal Length.



(b) Sepal Width.

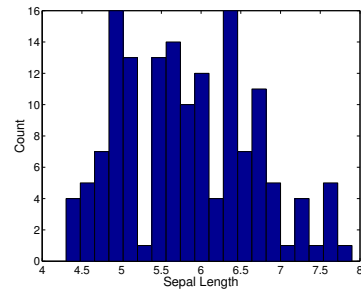


(c) Petal Length.

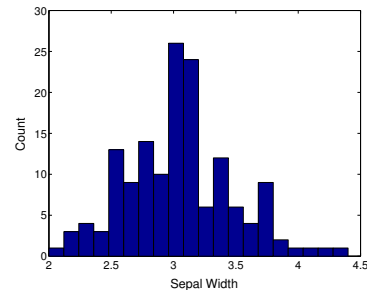


(d) Petal Width.

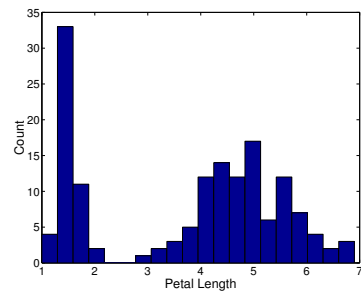
Figure 6.9. Histograms of Four Iris Attributes - 10 bins.



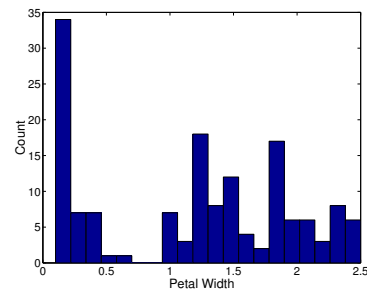
(a) Sepal Length.



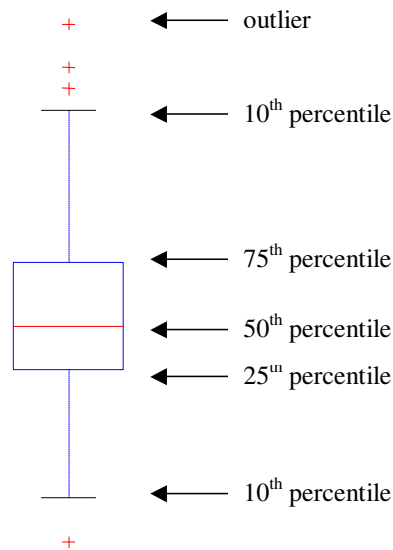
(b) Sepal Width.



(c) Petal Length.



(d) Petal Width.

Figure 6.10. Histograms of Four Iris Attributes - 20 bins.**Figure 6.11.** Description of Box Plot.

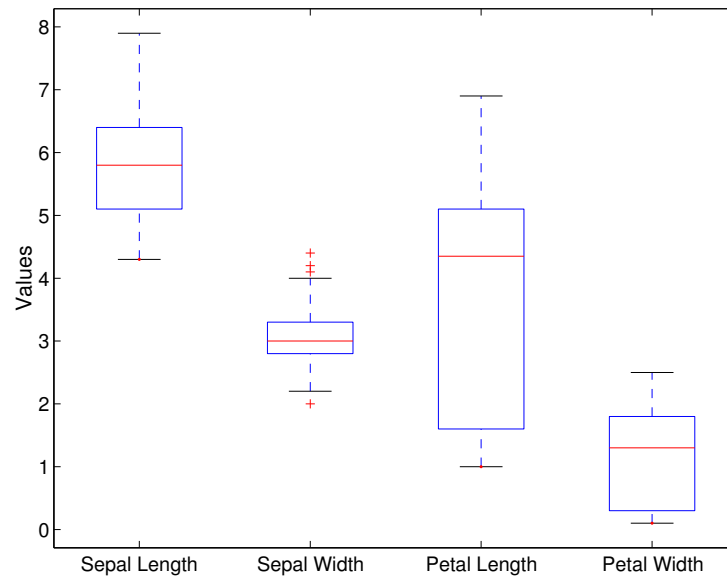
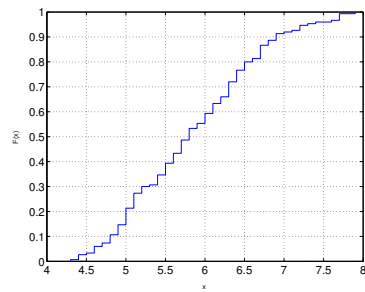


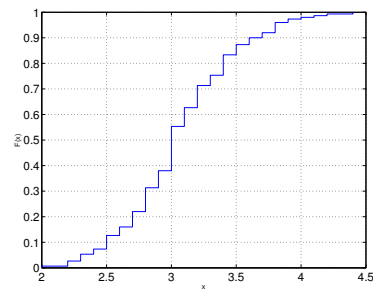
Figure 6.12. Box plot for Iris attributes.

cumulative distribution function shows, for each value of a statistical distribution, the fraction of points that are less than that value. An empirical cumulative distribution function shows, for the value of each point, 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.

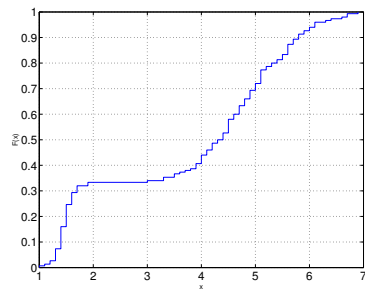
To illustrate, we show the ECDF's of the IRIS attributes in Figure 6.13.



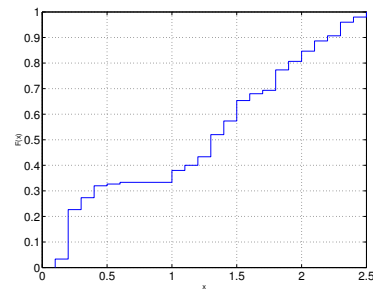
(a) Sepal Length.



(b) Sepal Width.



(c) Petal Length.



(d) Petal Width.

Figure 6.13. Empirical CDF's of Four Iris Attributes.

Quantile-Quantile Plot**Normal Probability Plot****Bar Plot****Pie Chart****6.3.2 Visualizing Two-dimensional Data****Scatter Plot****Tables****Matrices****6.3.3 Visualizing Three-dimensional Data****Scatter Plots****Matrices****Surface Plots****Contour Plots****Slices****Vector Field Plots****6.3.4 Visualizing Four-dimensional Data****Surface Plots****Scatter Plots****Slices****Visualizing Temporal Data****Frequency Plots****Animation****6.3.5 Visualizing Higher-dimensional Data****Parallel Coordinates**

Parallel coordinates are an approach for displaying data that have many attributes. The idea is the following: We have one coordinate axis for each attribute, but instead trying to place coordinate axes perpendicular to each other, as is normal, we place them parallel to each other. Also, instead of representing an object as a point, we represent it as a line, i.e., for each value of an attribute of an object, we put a point on the corresponding coordinate axis value and then connect all of these points.

It might be expected 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

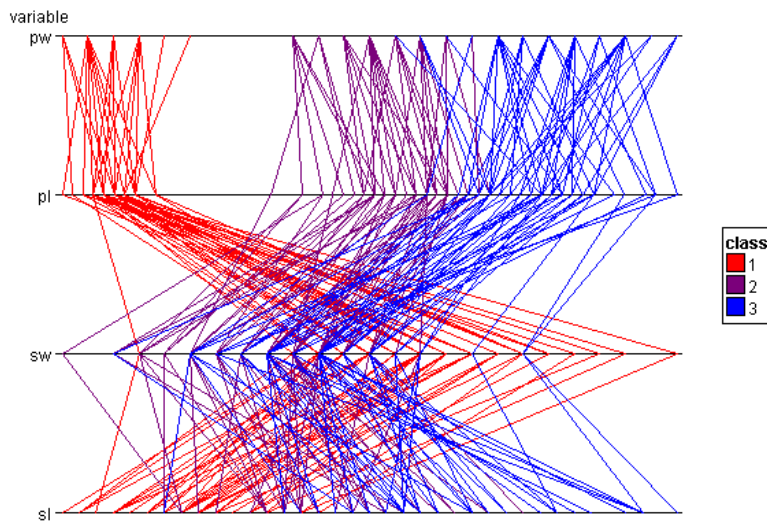


Figure 6.14. A parallel coordinates plot of the four iris attributes.

have similar values for their attributes. If so, and if the number of data object is not too large, then the resulting parallel coordinates plot, can be quite illuminating.

To illustrate, we show a parallel coordinates plot of the four numerical attributes of the Iris data set in Figure 6.14. We have ‘colored’ the lines with their class label to make the plot more informative. Notice that the parallel coordinates plot shows that the classes are reasonably well-separated in the petal width and petal length variables, but less well-separated for sepal length and sepal width.

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 are crossing over one another a lot, the picture can become confusing, and thus, it can be desirable to order the coordinate axes in such a way that, we sequences of axes with less crossover. The reader should compare Figure 6.15, where we have put the attribute that is most ‘mixed,’ i.e., sepal width, at the top of the figure to Figure 6.14, where this attribute is in the middle.

Star Coordinates

Faces and Multidimensional Icons

6.4 Exercises

This chapter has no formal exercises. Nonetheless, it is recommended that students obtain one or more of the data sets available at the UCI Machine Learning Repository (<http://www.ics.uci.edu/mlearn/MLRepository.html>) and apply as many of the different visualization techniques described in the chapter as possible. To do that it is necessary to have access to software that supports the visualization techniques. The bibliographic notes provide some pointers in that regard, including some resources that are freely available (R and WebStat).

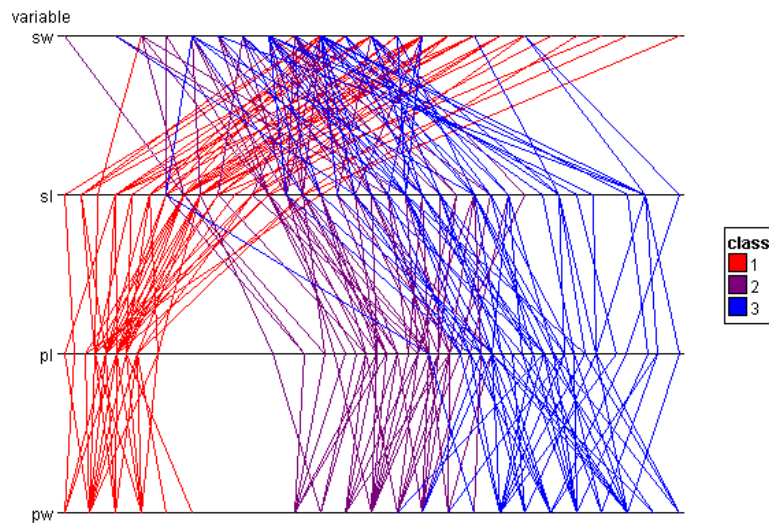


Figure 6.15. A parallel coordinates plot of the four iris attributes.

6.5 Bibliographic Notes

The basic visualization techniques are readily available, being an integral part of most spreadsheets (Microsoft EXCEL [126]), statistics programs (SAS [91], SPSS [169], R [59], and S-PLUS [39]), and mathematics software (MATLAB [182] and Mathematica [198]). Furthermore, documentation of visualization techniques—often with illustrative examples—is often provided by these packages. Most of the graphics in this chapter were generated by using MATLAB, although some were created using WebStat [147], a convenient Web accessible statistics package with graphics capabilities. The statistics package R is also freely available.

The literature on visualization is large and extensive, covering many fields and many decades. One of the classics of the field is *The Visual Display of Quantitative Information* by Edward R. Tufte [187]. A useful reference for information visualization—both principles and techniques—is *Visualization Information* by Robert Spence. This book also provides a thorough discussion of many “dynamic” visualization techniques that are not covered in this chapter, e.g., brushing and zoom and pan. Two other books that may be of interest to those interesting in pursuing visualization are *Readings in Information Visualization: Using Vision to Think* [30] and *Information Visualization in Data Mining and Knowledge Discovery* [55].

Finally, there is a great deal of information available 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 Michael Friendly [62]. The ACCENT Principles for effective graphical display can be found there, or in their original publication [28].

Anomaly Detection

Special Topics in Data Mining

8.1 Spatio-temporal Data Mining

8.2 Network Intrusion Detection

Bibliography

- [1] R. Agarwal and J. Shafer. Parallel mining of association rules. *IEEE Transactions on Knowledge and Data Engineering*, 8(6):962–969, March 1998.
- [2] R. C. Agarwal, 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.
- [3] C. Aggarwal, Z. Sun, and P. Yu. Online generation of profile association rules. In *Proc. of the Fourth Int'l Conference on Knowledge Discovery and Data Mining*, pages 129–133, New York, NY, 1996.
- [4] C. Aggarwal and P. Yu. Mining large itemsets for association rules. *Data Engineering Bulletin*, 21(1):23–31, March 1998.
- [5] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In L. M. Haas and A. Tiwary, editors, *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA*, pages 94–105, Seattle, Washington, June 1998. ACM Press.
- [6] R. Agrawal, T. Imielinski, and A. Swami. Database mining: a performance perspective. *IEEE Transactions on Knowledge and Data Engineering*, 5:914–925, 1993.
- [7] 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 D.C., USA, 1993.
- [8] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proc. of Int. Conf. on Data Engineering*, Taipei, Taiwan, 1995.
- [9] D. Aha. *A study of instance-based algorithms for supervised learning tasks: mathematical, empirical, and psychological evaluations*. PhD thesis, University of California, Irvine, 1990.

- [10] K. Ali, S. Manganaris, and R. Srikant. Partial classification using association rules. In *Proc. of the Third Int'l Conference on Knowledge Discovery and Data Mining*, pages 115–118, Newport Beach, CA, August 1997.
- [11] K. Alsabti, S. Ranka, and V. Singh. Clouds: A decision tree classifier for large datasets. In *Proc. of the Fourth Int'l Conference on Knowledge Discovery and Data Mining*, pages 2–8, 1998.
- [12] M. R. Anderberg. *Cluster Analysis for Applications*. Academic Press, New York, December 1973.
- [13] R. Andrews, J. Diederich, and A. Tickle. A survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge Based Systems*, 8(6):373–389, 1995.
- [14] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander. Optics: Ordering points to identify the clustering structure. In A. Delis, C. Faloutsos, and S. Ghandeharizadeh, editors, *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadelphia, Pennsylvania, USA*, pages 49–60. ACM Press, June 1999.
- [15] P. Arabie, L. Hubert, and G. De Soete. An overview of combinatorial data analysis. In P. Arabie, L. Hubert, and G. De Soete, editors, *Clustering and Classification*, pages 188–217. World Scientific, Singapore, January 1996.
- [16] D. Barbara, 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.
- [17] K. Bennett and C. Campbell. Support vector machines: Hype or hallelujah. *SIGKDD Explorations*, 2(2):1–13, 2000.
- [18] P. Berkhin. Survey of clustering data mining techniques. Technical report, Accrue Software, San Jose, CA, 2002.
- [19] L. Bing, W. Hsu, and Y. Ma. Integrating classification and association rule mining. In *Proc. of the Fourth Int'l Conference on Knowledge Discovery and Data Mining*, New York, NY, 1998.
- [20] A. Blum and P. Langley. Selection of relevant features and examples in machine learning. *Artificial Intelligence*, 97(1-2):245–271, 1997.
- [21] L. Breiman, J. H. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Chapman & Hall, New York, 1984.
- [22] L. Breslow and D. Aha. Simplifying decision trees: A survey. *Knowledge Engineering Review*, 12(1):1–40, 1997.

- [23] M. M. Breunig, H.-P. Kriegel, R. Ng, and J. Sander. Lof: identifying density-based local outliers. In W. Chen, J. F. Naughton, and P. A. Bernstein, editors, *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA*, pages 93–104, Dallas, TX, May 2000. ACM Press.
- [24] 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.
- [25] 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 Int. Conf. on Management of Data*, pages 255–264, Tucson, AZ, June 1997.
- [26] W. Buntine. Learning classification trees. In D. Hand, editor, *Artificial Intelligence frontiers in statistics*, pages 182–201. Chapman & Hall, London, 1993.
- [27] C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.
- [28] D. A. Burn. Designing effective statistical graphs. In C. R. Rao, editor, *Handbook of Statistics 9*. Elsevier/North-Holland, Amsterdam, The Netherlands, September 1993.
- [29] C. Cai, A. Fu, C. Cheng, and W. Kwong. Mining association rules with weighted items. In *Proc. of IEEE International Database Engineering and Applications Symposium*, pages 68–77, 1998.
- [30] 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.
- [31] Q. Chen, U. Dayal, and M. Hsu. A distributed olap infrastructure for e-commerce. In *Proc. of the Fourth IFCIS International Conference on Cooperative Information Systems*, pages 209–220, Edinburgh, Scotland, 1999.
- [32] V. Cherkassky and F. Mulier. *Learning From Data: Concepts, Theory, and Methods*. Wiley Interscience, 1998.
- [33] D. Cheung, S. Lee, and B. Kao. A general incremental technique for maintaining discovered association rules. In *Proc. of the Fifth Intl. Conf. on Database Systems for Advanced Applications*, Melbourne, Australia, 1997.
- [34] P. Clark and R. Boswell. Rule induction with cn2: Some recent improvements. In *Machine Learning - Proc. of the Fifth European Conference (EWSL-91)*, pages 151–163, 1991.

- [35] P. Clark and T. Niblett. The cn2 induction algorithm. *Machine Learning*, 3(4):261–283, 1989.
- [36] W. G. Cochran. *Sampling Techniques*. John Wiley & Sons, 3rd edition, July 1977.
- [37] W. Cohen. Fast effective rule induction. In *Machine Learning: Proc of the Twelfth International Conference*, 1995.
- [38] R. Cooley, P. 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.
- [39] I. Corporation. S-plus. <http://www.insightful.com>, 2003.
- [40] S. Cost and S. Salzberg. A weighted nearest neighbor algorithm for learning with symbolic features. *Machine Learning*, 10:57–78, 1993.
- [41] T. Cover and P. Hart. Nearest neighbor pattern classification. *Knowledge Based Systems*, 8(6):373–389, 1995.
- [42] T. Dasu and T. Johnson. *Exploratory Data Mining and Data Cleaning*. John Wiley & Sons, 2003.
- [43] J. W. Demmel. *Applied Numerical Linear Algebra*. Society for Industrial & Applied Mathematics, September 1997.
- [44] P. Dokas, L. Ertöz, V. Kumar, A. Lazarevic, J. Srivastava, and P. Tan. Data mining for network intrusion detection. In *Proc. NSF Workshop on Next Generation Data Mining*, Baltimore, MD., 2002.
- [45] P. Domingos. The rise system: Conquering without separating. In *Proc. of the Sixth IEEE Int’l Conf. on Tools with Artificial Intelligence*, New Orleans, LA, 1994.
- [46] P. Domingos. The role of occam’s razor in knowledge discovery. *Data Mining and Knowledge Discovery*, 3(4):409–425, 1999.
- [47] G. Dong and J. Li. Efficient mining of emerging patterns: Discovering trends and differences. In *Proc. of the Fifth Int’l Conference on Knowledge Discovery and Data Mining*, San Diego, CA, 1999.
- [48] J. Dougherty, R. Kohavi, and M. Sahami. Supervised and unsupervised discretization of continuous features. In *International Conference on Machine Learning*, pages 194–202, 1995.
- [49] R. Duda, P. Hart, and D. Stork. *Pattern Classification*. John Wiley & Sons, Inc., New York, second edition, 2001.

- [50] W. duMouchel and D. Pregibon. Empirical bayes screening for multi-item associations. In *Proc. of the Seventh Int'l Conference on Knowledge Discovery and Data Mining*, pages 67–76, San Francisco, CA, August 2001.
- [51] M. Dunham. *Data Mining: Introductory and Advanced Topics*. Prentice Hall, 2002.
- [52] T. Elomaa and J. Rousu. General and efficient multisplitting of numerical attributes. *Machine Learning*, 36(3):201–244, 1999.
- [53] 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.
- [54] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In E. Simoudi, J. Han, and U. Fayyad, editors, *KDD '96, Proceedings of Second International Conference on Knowledge Discovery and Data Mining, August 2-4, 1996, Portland, Oregon, USA*, pages 226–231. AAAI Press, August 1996.
- [55] U. Fayyad, G. G. Grinstein, and A. Wierse, editors. *Information Visualization in Data Mining and Knowledge Discovery*. Morgan Kaufmann Publishers, San Francisco, CA, September 2001.
- [56] U. M. Fayyad and K. B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proc. 13th Int. Joint Conf. on Artificial Intelligence*, pages 1022–1027. Morgan Kaufman, 1993.
- [57] L. Feng, H. Lu, J. Yu, and J. Han. Mining inter-transaction associations with templates. In *Proc. 1999 Int. Conf. on Information and Knowledge Management (CIKM'99)*, pages 225–233, Kansas City, Missouri, Nov 1999.
- [58] D. Fisher. Iterative optimization and simplification of hierarchical clusterings. *Journal of Artificial Intelligence Research*, 4:147–179, 1996.
- [59] T. R. P. for Statistical Computing. R: A language and environment for statistical computing and graphics. <http://www.r-project.org/>, 2003.
- [60] A. Freitas. Understanding the crucial differences between classification and discovery of association rules - a position paper. *SIGKDD Explorations*, 2(1):65–69, 2000.
- [61] J. Friedman and N. Fisher. Bump hunting in high-dimensional data. *Statistics and Computing*, 9(2):123–143, April 1999.
- [62] M. Friendly. Gallery of data visualization. <http://www.math.yorku.ca/SCS/Gallery/>, 2003.

- [63] T. Fukuda, Y. Morimoto, S. Morishita, and T. Tokuyama. Mining optimized association rules for numeric attributes. In *Proceedings of the 15th Symposium on Principles of Database Systems*, pages 182–191, Montreal, CA, June 1996.
- [64] F. H. Gaohua Gu and H. Liu. Tra6/00: Sampling and its application in data mining: A survey. Technical report, National University of Singapore, Singapore, 2000.
- [65] M. Garofalakis, R. Rastogi, and K. Shim. Spirit: Sequential pattern mining with regular expression constraints. In *Proc. of the 25th VLDB Conference*, pages 223–234, Edinburgh, Scotland, 1999.
- [66] 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.
- [67] George Karypis. Cluto 2.1: Software for clustering high-dimensional datasets. /www.cs.umn.edu/~karypis, August 2002.
- [68] D. Gunopulos, R. Khardon, H. Mannila, and H. Toivonen. Data mining, hypergraph transversals, and machine learning. In *Proc. 16th Symposium on Principles of Database Systems*, Tucson, AZ, May 1997.
- [69] E. 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.
- [70] E. Han, G. Karypis, and V. Kumar. Scalable parallel data mining for association rules. In *Proc. of 1997 ACM-SIGMOD Int. Conf. on Management of Data*, 1997.
- [71] E. Han, G. Karypis, V. Kumar, and B. Mobasher. Clustering based on association rule hypergraphs. In *SIGMOD'97 Workshop on Research Issues in Data Mining and Knowledge Discovery*, Tucson, AZ, 1997.
- [72] J. Han and Y. Fu. Discovery of multiple-level association rules from large databases. In *Proc. of the 21st VLDB Conference*, pages 420–431, Zurich, Switzerland, 1995.
- [73] J. Han, Y. Fu, K. Koperski, W. Wang, and O. Zaiane. Dmql: A data mining query language for relational databases. In *Proc. of 1996 ACM-SIGMOD Int. Conf. on Management of Data*, Montreal, Canada, 1996.
- [74] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, San Francisco, 2001.
- [75] J. Han, M. Kamber, and A. K. H. 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.
- [76] 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)*, Dallas, TX, May 2000.
 - [77] D. Hand, H. Mannila, and P. Smyth. *Principles of Data Mining*. MIT Press, 2001.
 - [78] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, Prediction*. Springer, 2001.
 - [79] M. Hearst. Trends & controversies: Support vector machines. *IEEE Intelligent Systems*, 13(4):18–28, 1998.
 - [80] D. Heckerman. Bayesian networks for data mining. *Data Mining and Knowledge Discovery*, 1(1):79–119, 1997.
 - [81] C. Hidber. Online association rule mining. In *Proc. of 1999 ACM-SIGMOD Int. Conf. on Management of Data*, Philadelphia, PA, 1999.
 - [82] A. Hinneburg and D. A. Keim. An efficient approach to clustering in large multimedia databases with noise. In R. Agrawal and P. Stolorz, editors, *KDD '98, Proceedings of Fourth International Conference on Knowledge Discovery and Data Mining, August 27-31, 1998, New York City, New York, USA*, pages 58–65. AAAI Press, August 1998.
 - [83] A. Hinneburg and D. A. Keim. Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering. In M. P. Atkinson, M. E. Orlowska, P. Valduriez, S. B. Zdonik, and M. L. Brodie, editors, *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, pages 506–517, New York, September 1999. Morgan Kaufmann.
 - [84] J. Hipp, U. Guntzer, and G. Nakhaeizadeh. Algorithms for association rule mining - a general survey. *SigKDD Explorations*, 2(1):58–64, June 2000.
 - [85] H. Hofmann, A. Siebes, and A. Wilhelm. Visualizing association rules with interactive mosaic plots. In *Proc. of the Sixth Int'l Conference on Knowledge Discovery and Data Mining*, pages 227–235, 2000.
 - [86] J. Holt and S. Chung. Efficient mining of association rules in text databases. In *Proc. of the 1999 ACM CIKM Int'l Conf on Information and Knowledge Management*, pages 234–242, Kansas City, Missouri, 1999.
 - [87] R. Holte. Very simple classification rules perform well on most commonly used data sets. *Machine Learning*, 11:63–91, 1993.

- [88] M. Houtsma and A. Swami. Set-oriented mining for association rules in relational databases. In *Proc. of the Eleventh Int'l Conference on Data Engineering*, pages 25–33, Taipei, Taiwan, 1995.
- [89] F. Hussain, H. Liu, C. L. Tan, and M. Dash. Trc6/99: Discretization: an enabling technique. Technical report, National University of Singapore, Singapore, 1999.
- [90] T. Imielinski, A. Virmani, and A. Abdulghani. Discovery board application programming interface and query language for database mining. In *Proc. of the Second Int'l Conference on Knowledge Discovery and Data Mining*, Portland, Oregon, 1996.
- [91] S. I. Inc. Sas:statistical analysis system. <http://www.sas.com/>, 2003.
- [92] A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *Proc. of The Fourth European Conference on Principles and Practice of Knowledge Discovery in Databases*, Lyon, France, 2000.
- [93] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice Hall Advanced Reference Series. Prentice Hall, Englewood Cliffs, New Jersey, March 1988.
- [94] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323, September 1999.
- [95] D. Jensen and P. Cohen. Multiple comparisons in induction algorithms. *Machine Learning*, 38(3):309–338, March 2000.
- [96] G. John, R. Kohavi, and K. Pfleger. Irrelevant features and the subset selection problem. In *Machine Learning: Proc. of the Eleventh International Conference*, pages 121–129, San Francisco, CA, 1994.
- [97] I. T. Jolliffe. *Principal Component Analysis*. Springer Verlag, 2nd edition, October 2002.
- [98] M. Joshi, G. Karypis, and V. Kumar. Scalparc: A new scalable and efficient parallel classification algorithm for mining large datasets. In *Proc. of 12th International Parallel Processing Symposium (IPPS/SPDP)*, Orlando, April 1998.
- [99] G. Karypis, E.-H. Han, and V. Kumar. Chameleon: A hierarchical clustering algorithm using dynamic modeling. *IEEE Computer*, 32(8):68–75, August 1999.
- [100] G. Karypis, E.-H. Han, and V. Kumar. Technical report 99-020: Multilevel refinement for hierarchical clustering. Technical report, University of Minnesota, Minneapolis, MN, 1999.

- [101] 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.
- [102] M. Klemettinen. *A Knowledge Discovery Methodology for Telecommunication Network Alarm Databases*. PhD thesis, University of Helsinki, 1999.
- [103] R. Kohavi. A study on cross-validation and bootstrap for accuracy estimation and model selection. In *Proc of the Int'l Joint Conference on Artificial Intelligence (IJCAI'95)*, 1995.
- [104] R. Kohavi and G. H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 1997.
- [105] W. Kusters, E. Marchiori, and A. Oerlemans. Mining clusters with association rules. In *The Third Symposium on Intelligent Data Analysis (IDA99)*, Amsterdam, August 1999.
- [106] D. H. Krantz, R. D. Luce, P. Suppes, and A. Tversky. *Foundations of Measurements (Vol I)*. Academic Press, New York, 1971.
- [107] J. B. Kruskal and E. M. Uslander. *Multidimensional Scaling*. Sage Publications, August 1978.
- [108] C. Kuok, A. Fu, and M. Wong. Mining fuzzy association rules in databases. *ACM SIGMOD Record*, 27(1):41–46, March 1998.
- [109] M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *Proc. of the 2001 IEEE International Conference on Data Mining*, pages 313–320, San Jose, CA, November 2001.
- [110] P. Langley, W. Iba, and K. Thompson. An analysis of bayesian classifiers. In *Proc. of the 10th National Conference on Artificial Intelligence*, 1992.
- [111] W. Lee, S. Stolfo, and K. Mok. Adaptive intrusion detection: A data mining approach. *Artificial Intelligence Review*, 14(6):533–567, 2000.
- [112] D. Lewis. Naive (bayes) at forty: The independence assumption in information retrieval. In *10th European Conference on Machine Learning (ECML 1998)*, 1998.
- [113] W. Li, J. Han, and J. Pei. Cmar: Accurate and efficient classification based on multiple class-association rules. In *Proc. of the 2001 IEEE International Conference on Data Mining (ICDM'01)*, pages 369–376, 2001.
- [114] B. Lindgren. *Statistical Theory*. CRC Press, January 1993.
- [115] B. Liu, W. Hsu, and S. Chen. Using general impressions to analyze discovered classification rules. In *Proc. of the Third Int'l Conference on Knowledge Discovery and Data Mining*, pages 31–36, 1997.

- [116] B. Liu, W. Hsu, and Y. Ma. Mining association rules with multiple minimum supports. In *Proc. of the Fifth Int'l Conference on Knowledge Discovery and Data Mining*, pages 125–134, 1999.
- [117] B. Liu, W. Hsu, and Y. Ma. Pruning and summarizing the discovered associations. In *Proc. of the Fifth Int'l Conference on Knowledge Discovery and Data Mining*, pages 125–134, San Diego, CA, 1999.
- [118] 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.
- [119] 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.
- [120] 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.
- [121] O. Mangasarian. Data mining via support vector machines. Technical Report Technical Report 01-05, Data Mining Institute, May 2001.
- [122] N. Megiddo and R. Srikant. Discovering predictive association rules. In *Proc. of the 4th Int'l Conference on Knowledge Discovery in Databases and Data Mining*, pages 274–278, New York, August 1998.
- [123] M. Mehta, R. Agrawal, and J. Rissanen. Sliq: A fast scalable classifier for data mining. In *Extending Database Technology*, pages 18–32, 1996.
- [124] R. Meo, G. Psaila, and S. Ceri. A new sql-like operator for mining association rules. In *Proc. of the 22nd VLDB Conference*, pages 122–133, Bombay, India, 1996.
- [125] R. S. Michalski, I. Mozetic, J. Hong, and N. Lavrac. The multi-purpose incremental learning system aq15 and its testing application to three medical domains. In *Proc. of 5th National Conference on Artificial Intelligence*, Orlando, August 1986.
- [126] I. Microsoft. Microsoft excel 2002. <http://www.microsoft.com/office/excel/default.asp>, 2003.
- [127] R. Miller and Y. Yang. Association rules over interval data. In *Proc. of 1997 ACM-SIGMOD Int. Conf. on Management of Data*, pages 452–461, Tucson, AZ, May 1997.
- [128] MIT. Mit total data quality management program. <http://web.mit.edu/tdqm/www/index.shtml>, 2003.

- [129] T. Mitchell. *Machine Learning*. McGraw-Hill, 1997.
- [130] L. C. Molina, L. Belanche, and ngela Nebot. Feature selection algorithms: A survey and experimental evaluation. In *IEEE International Conference on Data Mining*, 2002.
- [131] B. Moret. Decision trees and diagrams. *Computing Surveys*, 14(4):593–623, 1982.
- [132] 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 Conference*, pages 380–391, New York, August 1998.
- [133] A. Mueller. Fast sequential and parallel algorithms for association rule mining: A comparison. Technical Report CS-TR-3515, University of Maryland, August 1995.
- [134] S. Murthy. Automatic construction of decision trees from data: A multi-disciplinary survey. *Data Mining and Knowledge Discovery*, 2(4):345–389, 1998.
- [135] H. Nagesh, S. Goil, and A. Choudhary. Parallel algorithms for clustering high-dimensional large-scale datasets. In R. L. Grossman, C. Kamath, P. Kegelmeyer, V. Kumar, and R. R. Namburu, editors, *Data Mining for Scientific and Engineering Applications*, pages 335–356. Kluwer Academic Publishers, Dordrecht, Netherlands, October 2001.
- [136] R. Ng and J. Han. Efficient and effective clustering method for spatial data mining. In J. B. Bocca, M. Jarke, and C. Zaniolo, editors, *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*, pages 144–155, New York, September 1994. Morgan Kaufmann.
- [137] R. Ng, L. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained association rules. In *Proc. of 1998 ACM-SIGMOD Int. Conf. on Management of Data*, 1998.
- [138] F. Olken and D. Rotem. Random sampling from databases - a survey. *Statistics & Computing*, 5(1):25–42, March 1995.
- [139] J. Osborne. Notes on the use of data transformations. *Practical Assessment, Research & Evaluation*, 28(6), 2002.
- [140] B. Ozden, S. Ramaswamy, and A. Silberschatz. Cyclic association rules. In *Proc. of the 14th Int'l Conf. on Data Eng.*, pages 412–421, Orlando, FL, Feb 1998.

- [141] 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.
- [142] J. Park, M. Chen, and P. Yu. An effective hash-based algorithm for mining association rules. *SIGMOD Record*, 25(2):175–186, 1995.
- [143] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In *7th Int'l Conf. on Database Theory*, Jan 1999.
- [144] J. Pei, J. Han, H. Lu, S. Nishio, and S. Tang. H-mine: Hyper-structure mining of frequent patterns in large databases. In *Proc. of the 2001 IEEE International Conference on Data Mining*, pages 441–448, San Jose, CA, November 2001.
- [145] J. Pei, J. Han, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M. Hsu. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *Proc of Int'l Conf. on Data Engineering (ICDE'01)*, Heidelberg, Germany, April 2001.
- [146] J. Pei, J. Han, B. Mortazavi-Asl, and H. Zhu. Mining access patterns efficiently from web logs. In *PAKDD 2000*, April 2000.
- [147] G. G. A. S. Procedures. Webstat 3.0. <http://www.stat.sc.edu/rsrch/gasp/>, 2003.
- [148] F. J. Provost, D. Jensen, and T. Oates. Efficient progressive sampling. In *Knowledge Discovery and Data Mining*, pages 23–32, 1999.
- [149] J. Quinlan. *C4.5: Programs for Machine Learning*. Morgan-Kaufmann Publishers, 1993.
- [150] G. Ramkumar, S. Ranka, and S. Tsur. Weighted association rules: Model and algorithm. <http://www.cs.ucla.edu/~czdemo/tsur/>, 1997.
- [151] M. Ramoni and P. Sebastiani. Robust bayes classifiers. *Artificial Intelligence*, 125:209–226, 2001.
- [152] S. Roweis and L. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, December 2002.
- [153] S. Safavian and D. Landgrebe. A survey of decision tree classifier methodology. *IEEE Trans. Systems, Man and Cybernetics*, 22:660–674, May/June 1998.
- [154] M. Sahami. Learning limited dependence bayesian classifiers. In *Proc. of the Second Int'l Conference on Knowledge Discovery and Data Mining*, 1996.
- [155] 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.

- [156] S. Sarawagi, S. Thomas, and R. Agrawal. Integrating mining with relational database systems: Alternatives and implications. In *Proc. of 1998 ACM-SIGMOD Int. Conf. on Management of Data*, Seattle, WA, 1998.
- [157] K. Satou, G. Shibayama, T. Ono, Y. Yamamura, E. Furuichi, S. Kuhara, and T. Takagi. Finding association rules on heterogeneous genome data. In *Proc. of the Pacific Symposium on Biocomputing*, pages 397–408, Hawaii, January 1997.
- [158] 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)*, Zurich, Switzerland, Sept 1995.
- [159] A. Savasere, E. Omiecinski, and S. Navathe. Mining for strong negative associations in a large database of customer transactions. In *Proc. of the 14th International Conference on Data Engineering*, pages 494–502, Orlando, Florida, February 1998.
- [160] M. Seno and G. Karypis. Lpminer: An algorithm for finding frequent itemsets using length-decreasing support constraint. In *Proc. of the 2001 IEEE International Conference on Data Mining*, pages 505–512, San Jose, CA, November 2001.
- [161] J. Shafer, R. Agrawal, and M. Mehta. Sprint: A scalable parallel classifier for data mining. In *Proc. of the 22nd VLDB Conference*, pages 544–555, 1996.
- [162] G. Sheikholeslami, S. Chatterjee, and A. Zhang. Wavecluster: A multi-resolution clustering approach for very large spatial databases. In A. Gupta, O. Shmueli, and J. Widom, editors, *VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA*, pages 428–439, New York, August 1998. Morgan Kaufmann.
- [163] S. Shekhar and Y. Huang. Discovering spatial co-location patterns: A summary of results. In *Proc. of the 7th International Symposium on Spatial and Temporal Databases(SSTD01)*, Los Angeles, CA, 2001.
- [164] T. Shintani and M. Kitsuregawa. Hash based parallel algorithms for mining association rules. In *Proc of the 4th Intl. Conf. Parallel and Distributed Info. Systems*, December 1996.
- [165] 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.
- [166] 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.

- [167] L. Singh, B. Chen, R. Haight, and P. Scheuermann. An algorithm for constrained association rule mining in semi-structured data. In *PAKDD-99*, pages 148–158, 1999.
- [168] P. Smyth and R. Goodman. An information theoretic approach to rule induction from databases. *IEEE Trans. on Knowledge and Data Engineering*, 4(4):301–316, 1992.
- [169] I. SPSS. Spss: Statistical package for the social sciences. <http://www.spss.com/>, 2003.
- [170] R. Srikant and R. Agrawal. Mining generalized association rules. In *21st Very Large Database Conference*, pages 407–419, Zurich, Switzerland, 1995.
- [171] R. Srikant and R. Agrawal. Mining quantitative association rules in large relational tables. In H. V. Jagadish and I. S. Mumick, editors, *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 1–12, Montreal, Quebec, Canada, 4–6 1996.
- [172] R. Srikant and R. Agrawal. Mining quantitative association rules in large relational tables. In *Proc. of 1996 ACM-SIGMOD Int. Conf. on Management of Data*, pages 1–12, Montreal, Canada, 1996.
- [173] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *Proc. of the Fifth Intl Conf. on Extending Database Technology (EDBT)*, Avignon, France, 1996.
- [174] R. Srikant and R. Vu, Q. and Agrawal. Mining association rules with item constraints. In *Proc. of the Third Int'l Conference on Knowledge Discovery and Data Mining*, 1997.
- [175] M. Steinbach, G. Karypis, and V. Kumar. A comparison of document clustering techniques. In *KDD Workshop on Text Mining*, August 2000.
- [176] S. S. Stevens. Measurement. In G. M. Maranell, editor, *Scaling: A Sourcebook for Behavioral Scientists*, pages 22–41. Aldine Pub. Co, Chicago, 1974.
- [177] K. S. Sudipto Guha, Rajeev Rastogi. Cure: An efficient clustering algorithm for large databases. In L. M. Haas and A. Tiwary, editors, *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA*, pages 73–84. ACM Press, June 1998.
- [178] E. Suzuki. Autonomous discovery of reliable exception rules. In *Proc. of the Third Int'l Conference on Knowledge Discovery and Data Mining*, pages 259–262, 1997.

- [179] P. Tan and V. Kumar. Mining association patterns in web usage data. In *International Conference on Advances in Infrastructure for e-Business, e-Education, e-Science and e-Medicine on the Internet*, L'Aquila, Italy, January 2002.
- [180] P. Tan, V. Kumar, and J. Srivastava. Indirect association: Mining higher order dependencies in data. In *Proc. of the 4th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 632–637, Lyon, France, 2000.
- [181] J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, December 2002.
- [182] I. The MathWorks. Matlab 6.5. <http://www.mathworks.com>, 2003.
- [183] H. Toivonen. Sampling large databases for association rules. In T. M. Vijayaraman, A. P. Buchmann, C. Mohan, and N. L. Sarda, editors, *In Proc. 1996 Int. Conf. Very Large Data Bases*, pages 134–145. Morgan Kaufman, 09 1996.
- [184] H. Toivonen. Sampling large databases for association rules. In *Proc. of the 22nd VLDB Conference*, pages 134–145, Bombay, India, 1996.
- [185] 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*, 1995.
- [186] 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 Int. Conf. on Management of Data*, Seattle, Washington, 1998.
- [187] E. R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, Cheshire, CT, USA, March 1986.
- [188] J. W. Tukey. On the comparative anatomy of transformations. *Annals of Mathematical Statistics*, 28(3):602–632, September 1957.
- [189] A. Tung, H. Lu, J. Han, and L. Feng. Breaking the barrier of transactions: Mining inter-transaction association rules. In *Proc. of the Fifth Int'l Conference on Knowledge Discovery in Databases and Data Mining*, pages 297–301, San Diego, CA, August 1999.
- [190] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer Verlag, New York, 1995.
- [191] V. Vapnik. *Statistical Learning Theory*. John Wiley & Sons, New York, 1998.

- [192] H. Wang and C. Zaniolo. Cmp: A fast decision tree classifier using multivariate predictions. In *Proc. of the Sixteenth Int'l Conference on Data Engineering*, 449-460, 2000.
- [193] K. Wang, Y. He, and J. Han. Mining frequent itemsets using support constraints. In *Proc. of the 26th VLDB Conference*, Cairo, Egypt, 2000.
- [194] K. Wang, S. Tay, and B. Liu. Interestingness-based interval merger for numeric association rules. In *Proc. of the Fourth Int'l Conference on Knowledge Discovery and Data Mining*, New York, NY, 1998.
- [195] 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.
- [196] Warren S. Sarle. Measurement theory: Frequently asked questions. <ftp://ftp.sas.com/pub/neural/measurement.faq>, 1996.
- [197] I. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 1999.
- [198] I. Wolfram Research. Mathematica 4.2. <http://www.wolfram.com/>, 2003.
- [199] X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *Proc of the IEEE Int'l Conf. on Data Mining (ICDM'02)*, Maebashi City, Japan, December 2002.
- [200] C. Yang, U. Fayyad, and P. Bradley. Efficient discovery of error-tolerant frequent itemsets in high dimensions. In *Proc. of the Seventh Int'l Conference on Knowledge Discovery and Data Mining*, pages 194-203, San Francisco, CA, August 2001.
- [201] M. Zaki. Parallel and distributed association mining: A survey. *IEEE Concurrency, special issue on Parallel Mechanisms for Data Mining*, 7(4):14-25, December 1999.
- [202] M. Zaki. Generating non-redundant association rules. In *Proc. of the Sixth Int'l Conference on Knowledge Discovery and Data Mining*, 2000.
- [203] M. Zaki and C. Hsiao. Charm: An efficient algorithm for closed association rule mining. Technical Report TR 99-10, Rensselaer Polytechnic Institute, 1999.
- [204] M. Zaki and M. Orihara. Theoretical foundations of association rules. In *Proc. of 3rd SIGMOD'98 Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD'98)*, Seattle, WA, June 1998.

- [205] M. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New algorithms for fast discovery of association rules. In *Proc. of the Third Int'l Conference on Knowledge Discovery and Data Mining*, 1997.
- [206] 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.
- [207] Z. Zhang, Y. Lu, and B. Zhang. An effective partitioning-combining algorithm for discovering quantitative association rules. In *Proc. of the First Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 1997.
- [208] N. Zhong, Y. Yao, and S. Ohsuga. Peculiarity oriented multi-database mining. In *Principles of Data Mining and Knowledge Discovery: Third European Conference*, pages 136–146, Prague, Czech Republic, 1999.

Statistics

Linear Algebra