Data Mining Notes

Chapter 1: Introduction

- Data mining is a technology that blends traditional data analysis methods with sophisticated algorithms for processing large volumes of data. It has also opened up exciting opportunities for exploring and analyzing new types of data and for analyzing old types of data in new ways.
- **Business** Point-of-sale data collection (bar code scanners, radio frequency identification (RFID), and smart card technology) have allowed retailers to collect up-to-the-minute data about customer purchases at the checkout counters of their stores.
- Data mining techniques can be used to support a wide range of business intelligence applications such as **customer profiling, targeted marketing, workflow management, store layout, and fraud detection**. It can also help retailers answer important business questions such as "**Who are the most profitable customers?**" "**What products can be cross-sold or up-sold?**" and "**What is the revenue outlook of the company for next year?"**
- **Medicine, Science, and Engineering** Researchers in medicine, science, and engineering are rapidly accumulating data that is key to important new discoveries.

  Techniques developed in data mining can aid Earth scientists in answering questions such as "**What is the relationship between the frequency and intensity of ecosystem disturbances such as droughts and hurricanes to global warming?**" "**How is land surface precipitation and temperature affected by ocean surface temperature?**" and "**How well can we predict the beginning and end of the growing season for a region?**"

What is Data Mining ?

- **Data mining** is the process of automatically discovering useful information in large data repositories. Data mining techniques are deployed to scour large databases in order to find novel and useful patterns that might otherwise remain unknown.
- For example, looking up individual records using a database management system or finding particular Web pages via a query to an Internet search engine are tasks related to the area of **information retrieval**.

Data Mining and Knowledge Discovery

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

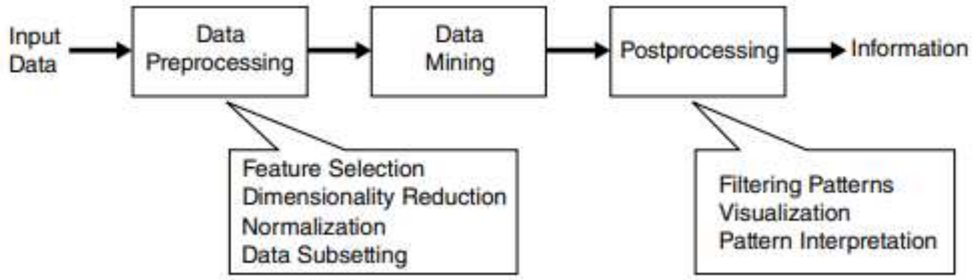


Figure 1.1. The process of knowledge discovery in databases (KDD).

- The **input data** can be stored in a variety of formats (flat files, spread-sheets, or relational tables) and may reside in a centralized data repository or be distributed across multiple sites.
- The purpose of **preprocessing** is to transform the raw input data into an appropriate format for subsequent analysis. The steps involved in data **preprocessing** include fusing data from multiple sources, cleaning data to remove noise and duplicate observations, and selecting records and features that are relevant to the data mining task at hand.
- "**Closing the loop**" is the phrase often used to refer to the process of integrating data mining results into decision support systems.
- In business applications, the insights offered by data mining results can be integrated with campaign management tools so that effective marketing promotions can be conducted and tested. Such integration requires a **postprocessing** step that ensures that only valid and useful results are incorporated into the decision support system.
- Statistical measures or hypothesis testing methods can also be applied during postprocessing to eliminate spurious data mining results.

Motivating Challenge

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

Scalability

- Because of advances in data generation and collection, data sets with sizes of gigabytes, terabytes, or even petabytes are becoming common. If data mining algorithms are to handle these massive data sets, then they must be scalable.
- Many data mining algorithms employ special search strategies to handle exponential search problems.
- Scalability may also require the implementation of novel data structures to access individual records in an efficient manner. Scalability can also be improved by using sampling or developing parallel and distributed algorithms.

High Dimensionality

- It is now common to encounter data sets with hundreds or thousands of attributes instead of the handful common a few decades ago.
- In bioinformatics, progress in microarray technology has produced gene expression data involving thousands of features.
- Data sets with temporal or spatial components also tend to have high dimensionality.
- For example, consider a data set that contains measurements of temperature at various locations. If the temperature measurements are taken repeatedly for an extended period, the number of dimensions (features) increases in proportion to Motivating Challenges the number of measurements taken.
- Traditional data analysis techniques that were developed for low-dimensional data often do not work well for such high-dimensional data. Also, for some data analysis algorithms, the computational complexity increases rapidly as the dimensionality (the number of features) increases.

Heterogeneous and Complex Data

- Traditional data analysis methods often deal with data sets containing attributes of the same type, either continuous or categorical.
- Examples of such non-traditional types of data include collections of Web pages containing semi-structured text and hyperlinks; DNA data with sequential and three-dimensional structure; and climate data that consists of time series measurements (temperature, pressure, etc.) at various locations on the Earth’s surface.

- Techniques developed for mining such complex objects should take into consideration relationships in the data, such as temporal and spatial autocorrelation, graph connectivity, and parent-child relationships between the elements in semi-structured text and XML documents.

Data Ownership and Distribution

- Sometimes, the data needed for an analysis is not stored in one location or owned by one organization. Instead, the data is geographically distributed among resources belonging to multiple entities. This requires the development of distributed data mining techniques. Among the key challenges faced by distributed data mining algorithms include

  (1) How to reduce the amount of communication needed to perform the distributed computation,

  (2) How to effectively consolidate the data mining results obtained from multiple sources, and

  (3) How to address data security issues.

Non-traditional Analysis

- The traditional statistical approach is based on a hypothesize-and-test paradigm. In other words, a hypothesis is proposed, an experiment is designed to gather the data, and then the data is analyzed with respect to the hypothesis. Unfortunately, this process is extremely labor-intensive.
- Current data analysis tasks often require the generation and evaluation of thousands of hypotheses, and consequently, the development of some data mining techniques has been motivated by the desire to automate the process of hypothesis generation and evaluation.

The Origins of Data Mining

- In particular, data mining draws upon ideas, such as

  (1) Sampling, estimation, and hypothesis testing from statistics and

  (2) search algorithms, modeling techniques, and learning theories from artificial intelligence, pattern recognition, and machine learning.

- Data mining has also been quick to adopt ideas from other areas, including optimization, evolutionary computing, information theory, signal processing, visualization, and information retrieval.

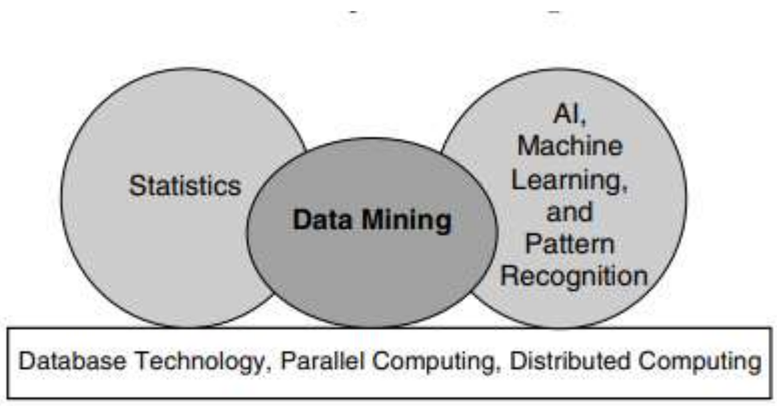

Statistics
Data Mining
AI, Machine Learning, and Pattern Recognition
Database Technology, Parallel Computing, Distributed Computing

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

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

Data Mining Tasks

- Data mining tasks are generally divided into two major categories:

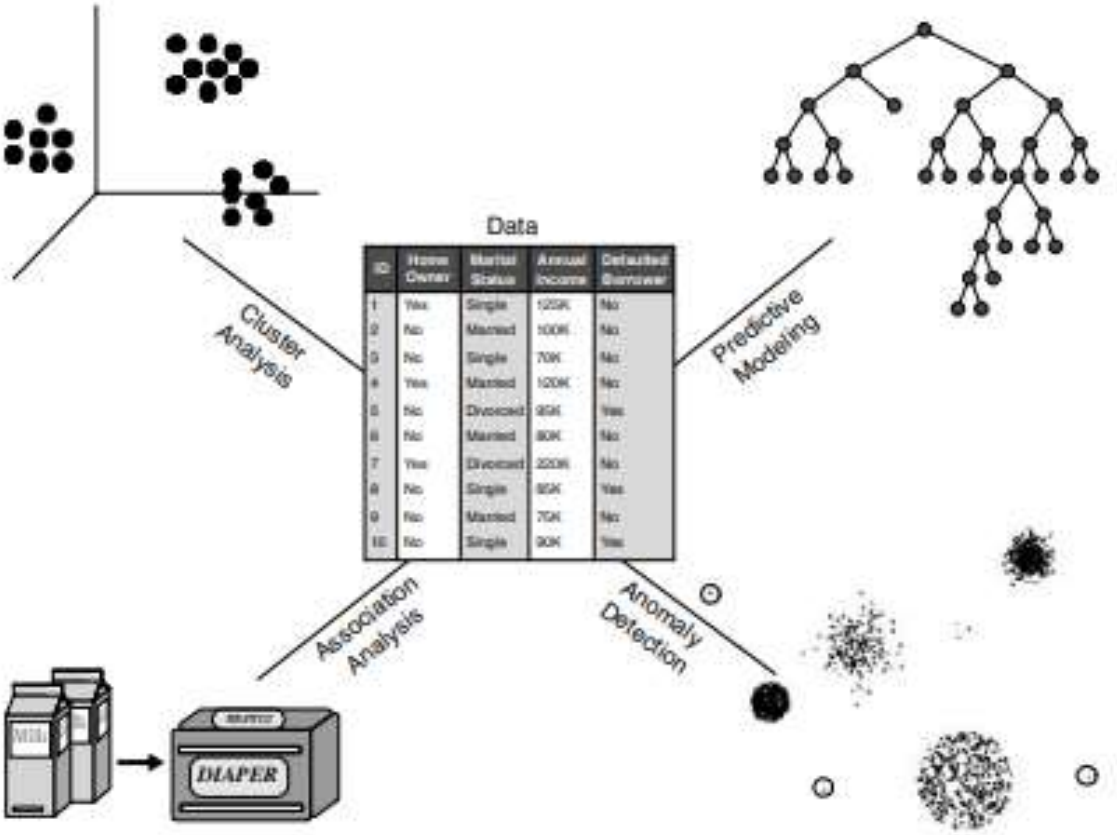
Predictive tasks

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

Descriptive tasks

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

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



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

Figure 1.3. Four of the core data mining tasks.

Predictive modeling

- **Predictive** refers to the task of building a model for the target variable as a function of the explanatory variables. There are two types of predictive modeling tasks:
    - **classification**, which is used for discrete target variables, and
    - **regression**, which is used for continuous target variables.

Example 1.1 (Predicting the Type of a Flower)

- Consider the task of predicting a species of flower based on the characteristics of the flower. In particular, consider classifying an Iris flower as to whether it belongs to one of the following three Iris species: Setosa, Versicolour, or Virginica. To perform this task, we need a data set containing the characteristics of various flowers of these three species. A data set with this type of information is the well-known Iris data set from the UCI Machine Learning Repository at http://www.ics.uci.edu/~mlearn.

- In addition to the species of a flower, this data set contains four other attributes: sepal width, sepal length, petal length, and petal width. (The Iris data set and its attributes are described further in Section 3.1.) Figure 1.4 shows a plot of petal width versus petal length for the 150 flowers in the Iris data set. Petal width is broken into the categories low, medium, and high, which correspond to the intervals [0, 0.75), [0.75, 1.75), [1.75, ∞), respectively. Also, petal length is broken into categories low, medium, and high, which correspond to the intervals [0, 2.5), [2.5, 5), [5, ∞), respectively. Based on these categories of petal width and length, the following rules can be derived:
  Petal width low and petal length low implies Setosa.
  Petal width medium and petal length medium implies Versicolour.
  Petal width high and petal length high implies Virginica.

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

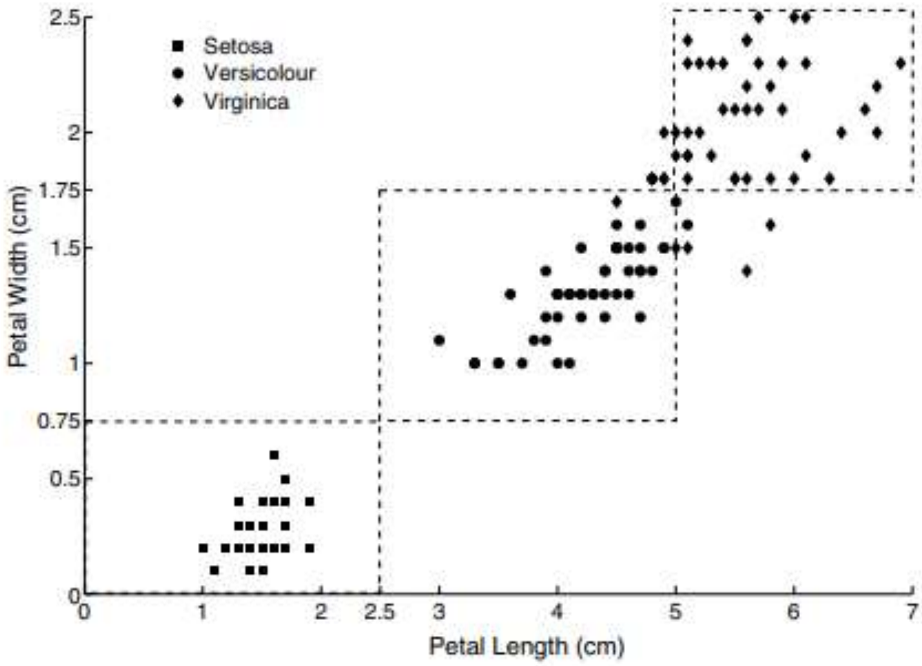


Figure 1.4. Petal width versus petal length for 150 Iris flowers.

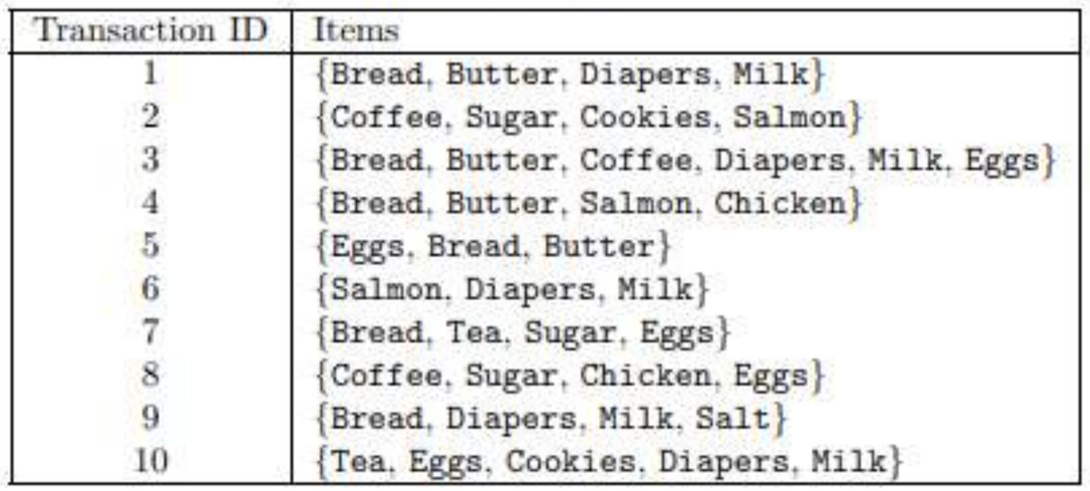
Association analysis

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

Example 1.2 (Market Basket Analysis)

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

Table 1.1. Market basket data.



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

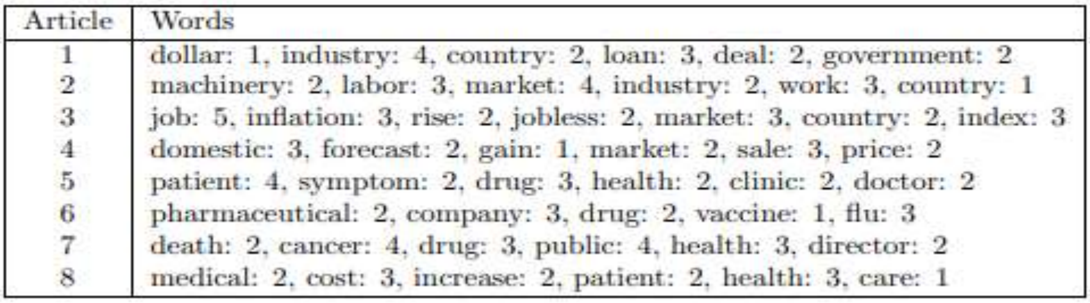
Cluster analysis

- **Cluster analysis** seeks to find groups of closely related observations so that observations that belong to the same cluster are more similar to each other than observations that belong to other clusters. Clustering has been used to group sets of related customers, find areas of the ocean that have a significant impact on the Earth's climate, and compress data.

Example 1.3 (Document Clustering)

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

Table 1.2. Collection of news articles.



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

Anomaly detection

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

Example 1.4 (Credit Card Fraud Detection)

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

Data Mining Notes

Chapter 2: Data

The Type of Data

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

The Quality of the Data

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

Pre-processing Steps to Make the Data More Suitable for Data Mining

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

Analyzing Data in Terms of Its Relationships

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

Types of Data

A **data** set can often be viewed as a combined collection of **data objects**. Other names for a data object are **record**, **point**, **vector**, **pattern**, **event**, **case**, **sample**, **observation**, or **entity**.

In turn, data objects are described by a number of attributes that capture the basic characteristics of an object, such as the mass of a physical object or the time at which an event occurred. Other names for an attribute are **variable**, **characteristic**, **field**, **feature**, or **dimension.**

Attributes and Measurement

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

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

Measurement :- The process of **measurement** is the application of a measurement scale to associate a value with a particular attribute of a specific object.

Measurement Scale :- A measurement scale is a rule (function) that associates a numerical or symbolic value with an attribute of an object.

The Different Types of Attributes

The following properties (operations) of numbers are typically used to describe attributes.

1. Distinctness; = and !=

2. Order; <, ≤, >, and ≥

3. Addition; + and −

4. Multiplication; ∗ and /

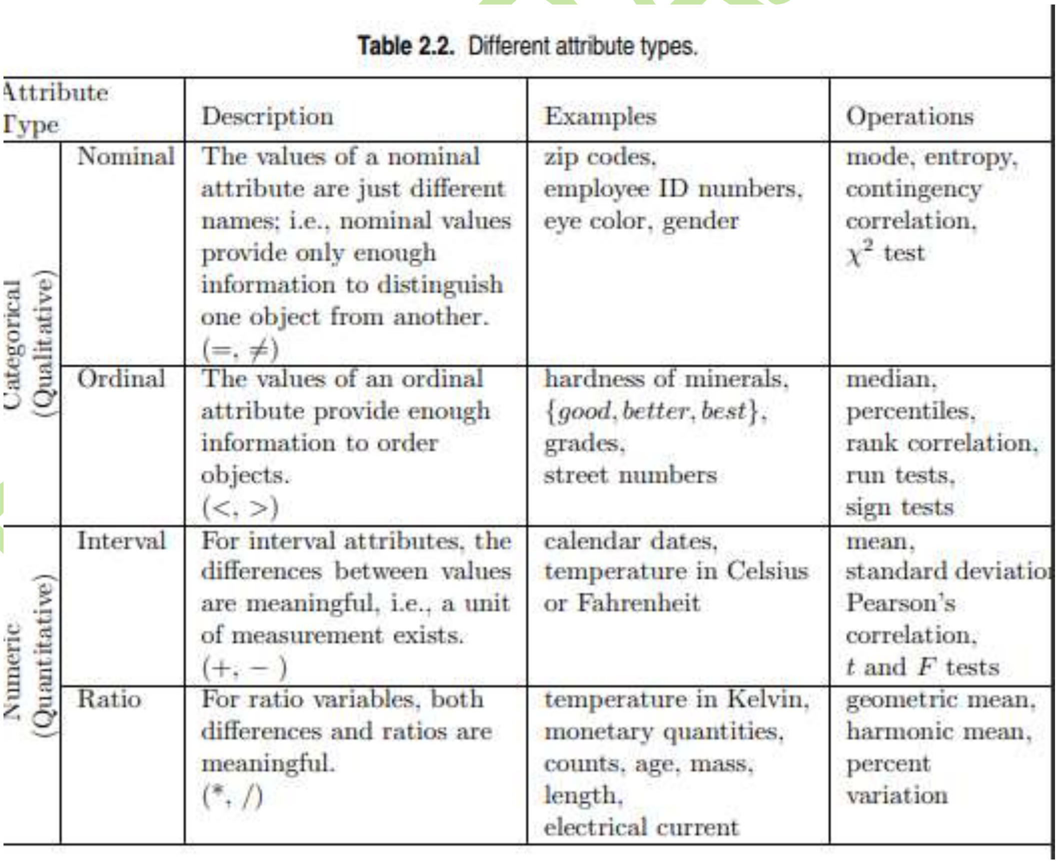
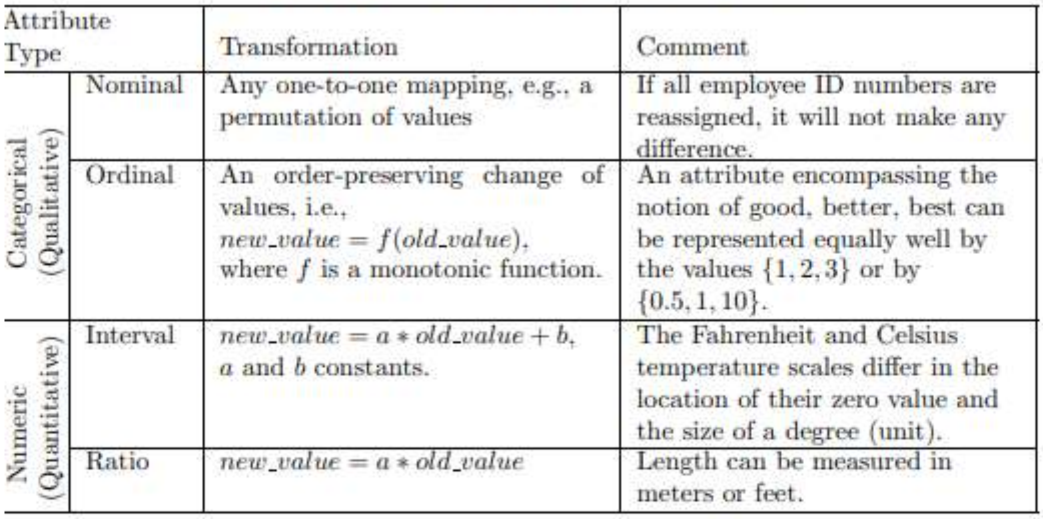


Table 2.2. Different attribute types.

Attribute Type		Description	Examples	Operations
Categorical (Qualitative)	Nominal	The values of a nominal attribute are just different names; i.e., nominal values provide only enough information to distinguish one object from another. ($=, \neq$)	zip codes, employee ID numbers, eye color, gender	mode, entropy, contingency correlation, χ^2 test
	Ordinal	The values of an ordinal attribute provide enough information to order objects. ($<, >$)	hardness of minerals, $\{good, better, best\}$, grades, street numbers	median, percentiles, rank correlation, run tests, sign tests
Numeric (Quantitative)	Interval	For interval attributes, the differences between values are meaningful, i.e., a unit of measurement exists. ($+, -$)	calendar dates, temperature in Celsius or Fahrenheit	mean, standard deviatio 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 2.3. Transformations that define attribute levels.



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

Describing Attributes by the Number of Values

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

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

Typically, **nominal** and **ordinal** attributes are **binary** or **discrete**, while **interval** and **ratio** attributes are **continuous**. However, **count** attributes, which are **discrete**, are also **ratio** attributes.

Asymmetric Attributes

For **asymmetric** attributes, only presence a non-zero attribute value is regarded as important.

Asymmetric binary attributes :- Binary attributes where only non-zero values are important are called **asymmetric binary attributes**.

Asymmetric discrete or **continuous attributes** :- If the number of credits associated with each course is recorded, then the resulting data set will consist of **asymmetric discrete** or **continuous attributes**.

Types of Data Sets

For convenience, we have grouped the types of data sets into three groups: **record data, graph-based data,** and **ordered data**. These categories do not cover all possibilities and other groupings are certainly possible.

General Characteristics of Data Sets

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

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

Resolution :- It is frequently possible to obtain data at different levels of resolution, and often the properties of the data are different at different resolutions. For instance, the surface of the Earth seems very uneven at a resolution of a few meters, but is relatively smooth at a resolution of tens of kilometers. The patterns in the data also depend on the level of resolution. If the resolution is too fine, a pattern may not be visible or may be buried in noise; if the resolution is too coarse, the pattern may disappear. For example, variations in atmospheric pressure on a scale of hours reflect the movement of storms and other weather systems. On a scale of months, such phenomena are not detectable.

Record Data

For the most basic form of record data, there is no explicit relationship among records or data fields, and every record (object) has the same set of attributes. Record data is usually stored either in flat files or in relational databases. Relational databases are certainly more than a collection of records, but data mining often does not use any of the additional information available in a relational database.

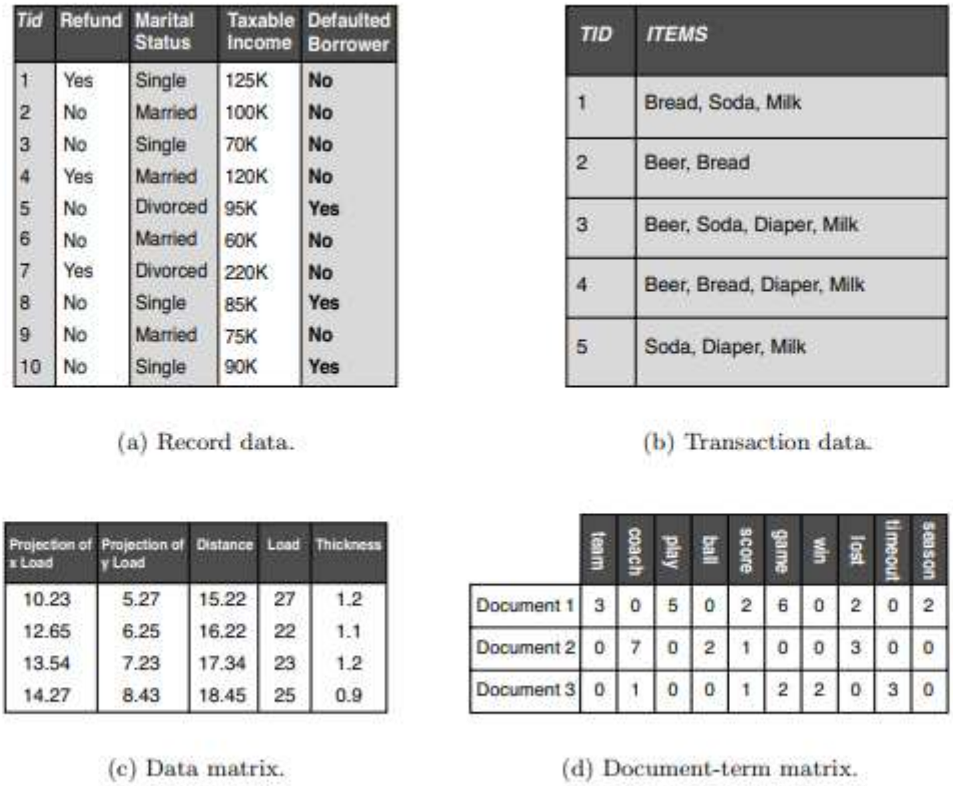
Transaction or Market Basket Data

Transaction data is a special type of record data, where each record (transaction) involves a set of items.

This type of data is called market basket data because the items in each record are the products in a person's "market basket." Transaction data is a collection of sets of items, but it can be viewed as a set of records whose fields are asymmetric attributes.

The Data Matrix

If the data objects in a collection of data all have the same fixed set of numeric attributes, then the data objects can be thought of as points (vectors) in a multidimensional space, where each dimension represents a distinct attribute describing the object. A set of such data objects can be interpreted as an m by n matrix, where there are m rows, one for each object, and n columns, one for each attribute. (A representation that has data objects as columns and attributes as rows is also fine.) This matrix is called a **data matrix** or a **pattern matrix**. A data matrix is a variation of record data, but because it consists of numeric attributes, standard matrix operation can be applied to transform and manipulate the data.



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

(a) Record data.

TID	ITEMS
1	Bread, Soda, Milk
2	Beer, Bread
3	Beer, Soda, Diaper, Milk
4	Beer, Bread, Diaper, Milk
5	Soda, Diaper, Milk

(b) Transaction data.

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

(c) Data matrix.

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

(d) Document-term matrix.

Figure 2.2. Different variations of record data.

The Sparse Data Matrix

A sparse data matrix is a special case of a data matrix in which the attributes are of the same type and are asymmetric; i.e., only non-zero values are important.

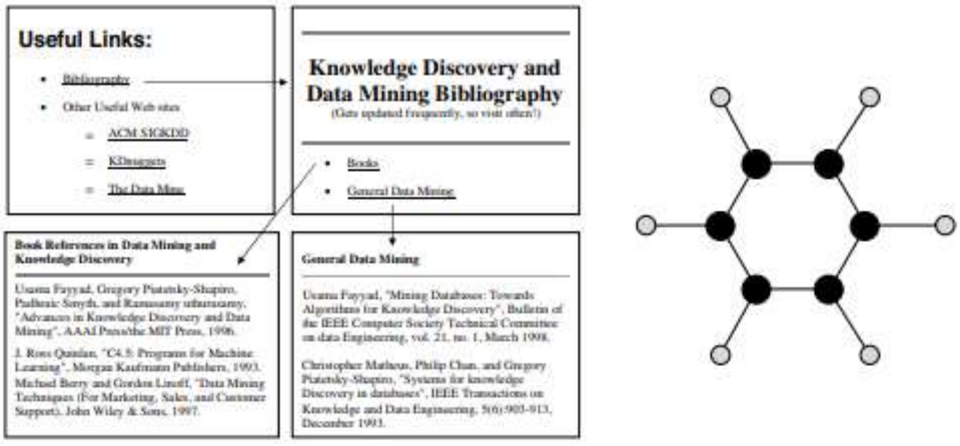
In particular, if the order of the terms (words) in a document is ignored, then a document can be represented as a term vector, where each term is a component (attribute) of the vector and the value of each component is the number of times the corresponding term occurs in the document. This representation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

Graph-Based Data

A graph can sometimes be a convenient and powerful representation for data. We consider two specific cases:

(1) the graph captures relationships among data objects and

(2) the data objects themselves are represented as graphs.



Useful Links:

- Bibliography
- Other Useful Web sites
  - ACM SIGKDD
  - KDnuggets
  - The Data Mine

Knowledge Discovery and Data Mining Bibliography

(Gets updated frequently, so visit often!)

- Books
- General Data Mining

Book References in Data Mining and Knowledge Discovery

Usama Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy uthurasamy, "Advances in Knowledge Discovery and Data Mining", AAAI Press/the MIT Press, 1996.

J. Ross Quinlan, "C4.5: Programs for Machine Learning", Morgan Kaufmann Publishers, 1993. Michael Berry and Gordon Linoff, "Data Mining Techniques (For Marketing, Sales, and Customer Support), John Wiley & Sons, 1997.

General Data Mining

Usama Fayyad, "Mining Databases: Towards Algorithms for Knowledge Discovery", Bulletin of the IEEE Computer Society Technical Committee on data Engineering, vol. 21, no. 1, March 1998.

Christopher Matheus, Philip Chan, and Gregory Piatetsky-Shapiro, "Systems for knowledge Discovery in databases", IEEE Transactions on Knowledge and Data Engineering, 5(6):903-913, December 1993.

(a) Linked Web pages.

(b) Benzene molecule.

Figure 2.3. Different variations of graph data.

Data with Relationships among Objects

The relationships among objects frequently convey important information. In such cases, the data is often represented as a graph. In particular, the data objects are mapped to nodes of the graph, while the relationships among objects are captured by the links between objects and link properties, such as direction and weight.

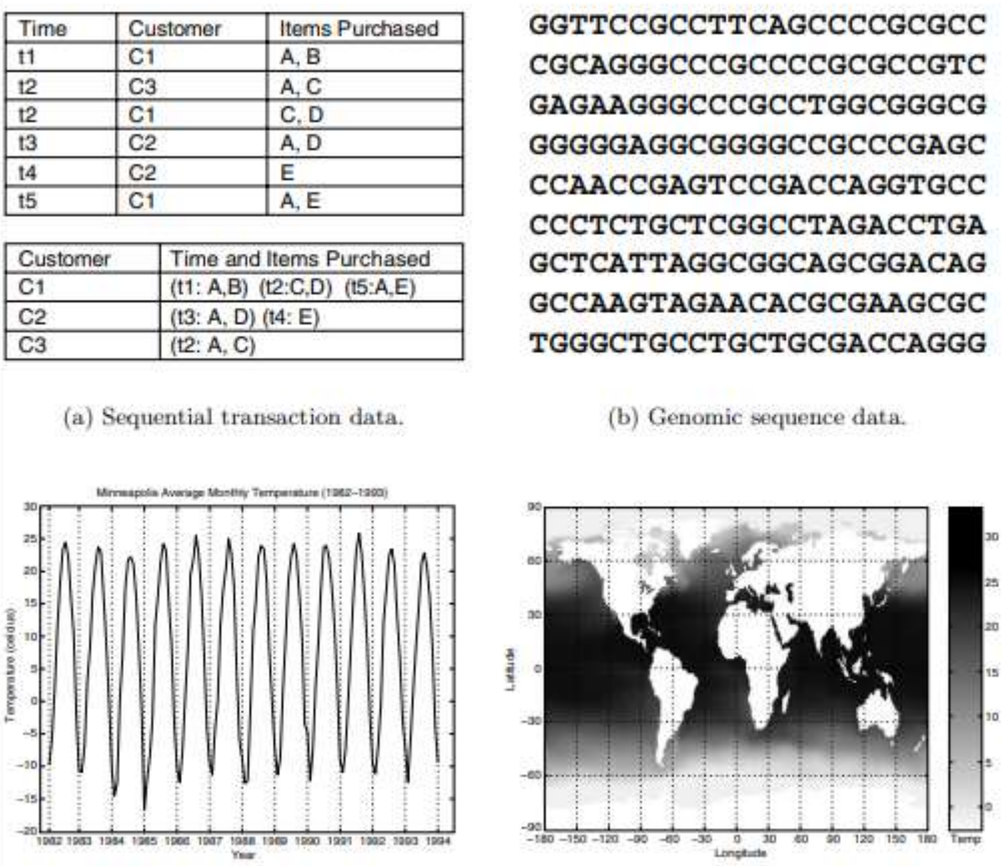
Data with Objects That Are Graphs

If objects have structure, that is, the objects contain sub-objects that have relationships, then such objects are frequently represented as graphs. For example, the structure of chemical compounds can be represented by a graph, where the nodes are atoms and the links between nodes are chemical bonds.

Ordered Data

Sequential Data

Sequential data, also referred to as **temporal data**, can be thought of as an extension of record data, where each record has a time associated with it. Consider a retail transaction data set that also stores the time at which the transaction took place. This time information makes it possible to find patterns such as "candy sales peak before Halloween." A time can also be associated with each attribute.



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

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

(a) Sequential transaction data.

GGTTCCGCCTTCAGCCCCGCGCC
CGCAGGGCCCGCCCCGCGCCGTC
GAGAAGGGCCCGCCTGGCGGGCG
GGGGGAGGCGGGGCCGCCCGAGC
CCAACCGAGTCCGACCAGGTGCC
CCCTCTGCTCGGCCTAGACCTGA
GCTCATTAGGCGGCAGCGGACAG
GCCAAGTAGAACACGCGAAGCGC
TGGGCTGCCTGCTGCGACCAGGG

(b) Genomic sequence data.

(c) Temperature time series.

(d) Spatial temperature data.

Figure 2.4. Different variations of ordered data.

Sequence Data

Sequence data consists of a data set that is a sequence of individual entities, such as a sequence of words or letters. It is quite similar to sequential data, except that there are no time stamps; instead, there are positions in an ordered sequence.

Time Series Data

Time series data is a special type of sequential data in which each record is a **time series**, i.e., a series of measurements taken over time.

When working with temporal data, it is important to consider **temporal autocorrelation**; i.e., if two measurements are close in time, then the values of those measurements are often very similar.

Spatial Data

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

Handling Non-Record Data

Record-oriented techniques can be applied to non-record data by extracting features from data objects and using these features to create a record corresponding to each object.

Given a set of common substructures, each compound can be represented as a record with binary attributes that indicate whether a compound contains a specific substructure. Such a representation is actually a transaction data set, where the transactions are the compounds and the items are the substructures.

Consider spatio-temporal data consisting of a time series from each point on a spatial grid. This data is often stored in a data matrix, where each row represents a location and each column represents a particular point in time. However, such a representation does not explicitly capture the time relationships that are present among attributes and the spatial relationships that exist among objects.

Data Quality

Data mining applications are often applied to data that was collected for another purpose, or for future, but unspecified applications. For that reason, data mining cannot usually take advantage of the significant benefits of “addressing quality issues at the source.”

Because preventing data quality problems is typically not an option, data mining focuses on

(1) the detection and correction of data quality problems and

(2) the use of algorithms that can tolerate poor data quality.

The first step, detection and correction, is often called **data cleaning**.

Measurement and Data Collection Issues

It is unrealistic to expect that data will be perfect. There may be problems due to human error, limitations of measuring devices, or flaws in the data collection process. Values or even entire data objects may be missing. In other cases, there may be spurious or duplicate objects; i.e., multiple data objects that all correspond to a single "real" object.

For example, there might be two different records for a person who has recently lived at two different addresses. Even if all the data is present and "looks fine," there may be inconsistencies—a person has a height of 2 meters, but weighs only 2 kilograms.

Measurement and Data Collection Errors

The term **measurement error** refers to any problem resulting from the measurement process. A common problem is that the value recorded differs from the true value to some extent. For continuous attributes, the numerical difference of the measured and true value is called the **error**. The term **data collection error** refers to errors such as omitting data objects or attribute values, or inappropriately including a data object.

For example, a study of animals of a certain species might include animals of a related species that are similar in appearance to the species of interest. Both measurement errors and data collection errors can be either systematic or random.

Noise and Artifacts

Noise is the random component of a measurement error. It may involve the distortion of a value or the addition of spurious objects.

The term noise is often used in connection with data that has a spatial or temporal component.

In such cases, techniques from signal or image processing can frequently be used to reduce noise and thus, help to discover patterns (signals) that might be "lost in the noise." Nonetheless, the elimination of noise is frequently difficult, and much work in data mining focuses on devising robust algorithms that produce acceptable results even when noise is present.

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

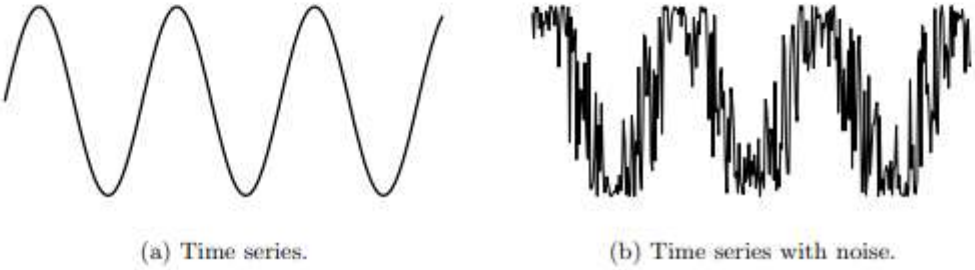


Figure 2.5. Noise in a time series context.

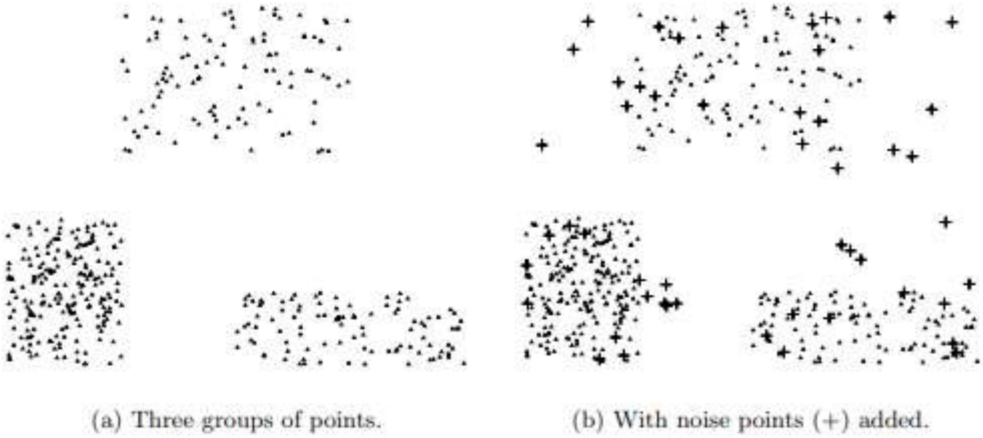


Figure 2.6. Noise in a spatial context.

Precision, Bias, and Accuracy

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

Bias: A systematic variation of measurements from the quantity being measured.

Accuracy: The closeness of measurements to the true value of the quantity being measured.

Outliers

Outliers are either (1) data objects that, in some sense, have characteristics that are different from most of the other data objects in the data set, or (2) values of an attribute that are unusual with respect to the typical values for that attribute.

Alternatively, we can speak of **anomalous** objects or values.

Missing Values

It is not unusual for an object to be missing one or more attribute values. In some cases, the information was not collected; e.g., some people decline to give their age or weight.

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

Eliminate Data Objects or Attributes

A simple and effective strategy is to eliminate objects with missing values. However, even a partially specified data object contains some information, and if many objects have missing values, then a reliable analysis can be difficult or impossible.

Estimate Missing Values

Sometimes missing data can be reliably estimated. For example, consider a time series that changes in a reasonably smooth fashion, but has a few, widely scattered missing values. In such cases, the missing values can be estimated (interpolated) by using the remaining values.

Ignore the Missing Value during Analysis

Many data mining approaches can be modified to ignore missing values. For example, suppose that objects are being clustered and the similarity between pairs of data objects needs to be calculated. If one or both objects of a pair have missing values for some attributes, then the similarity can be calculated by using only the attributes that do not have missing values.

Inconsistent Values

Data can contain **inconsistent** values. Consider an address field, where both a zip code and city are listed, but the specified zip code area is not contained in that city. It may be that the individual entering this information transposed two digits, or perhaps a digit was misread when the information was scanned from a handwritten form. Regardless of the cause of the inconsistent values, it is important to detect and, if possible, correct such problems.

Duplicate Data

A data set may include data objects that are **duplicates**, or almost **duplicates**, of one another. Many people receive duplicate mailings because they appear in a database multiple times under slightly different names. To detect and eliminate such duplicates, two main issues must be addressed. First, if there are two objects that actually represent a single object, then the values of corresponding attributes may differ, and these inconsistent values must be resolved. Second, care needs to be taken to avoid accidentally combining data objects that are similar, but not duplicates, such as two distinct people with identical names. The term deduplication is often used to refer to the process of dealing with these issues.

Issues Related to Applications

Data quality issues can also be considered from an application viewpoint as expressed by the statement "**data is of high quality if it is suitable for its intended use**."

Timeliness

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

Relevance

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

A common problem is **sampling bias**, which occurs when a sample does not contain different types of objects in proportion to their actual occurrence in the population.

Knowledge about the Data

Ideally, data sets are accompanied by documentation that describes different aspects of the data; the quality of this documentation can either aid or hinder the subsequent analysis.

Data Preprocessing

Data preprocessing is a broad area and consists of a number of different strategies and techniques that are interrelated in complex ways.

Specifically, we will discuss the following topics:

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

Aggregation

Sometimes "less is more" and this is the case with **aggregation**, the combining of two or more objects into a single object.

Quantitative attributes, such as price, are typically aggregated by taking a sum or an average. A qualitative attribute, such as item, can either be omitted or summarized as the set of all the items that were sold at that location.

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

There are several motivations for aggregation. **First**, the smaller data sets resulting from data reduction require less memory and processing time, and hence, aggregation may permit the use of more expensive data mining algorithms. **Second**, aggregation can act as a change of scope or scale by providing a high-level view of the data instead of a low-level view.

A **disadvantage** of aggregation is the potential loss of interesting details. In the store example aggregating over months loses information about which day of the week has the highest sales.

Sampling

Sampling is a commonly used approach for selecting a subset of the data objects to be analyzed.

The key principle for effective sampling is the following: Using a sample will work almost as well as using the entire data set if the sample is **representative**. In turn, a sample is representative if it has approximately the same property (of interest) as the original set of data.

Sampling Approaches

The simplest type of sampling is **simple random sampling**. For this type of sampling, there is an equal probability of selecting any particular item. There are two variations on random sampling (and other sampling techniques as well):

(1)**Sampling without replacement**—as each item is selected, it is removed from the set of all objects that together constitute the population, and

(2)**Sampling with replacement**—objects are not removed from the population as they are selected for the sample. In sampling with replacement, the same object can be picked more than once.

The samples produced by the two methods are not much different when samples are relatively small compared to the data set size, but sampling with replacement is simpler to analyze since the probability of selecting any object remains constant during the sampling process.

Stratified sampling, which starts with prespecified groups of objects, is such an approach. In the simplest version, equal numbers of objects are drawn from each group even though the groups are of different sizes. In another variation, the number of objects drawn from each group is proportional to the size of that group.

Progressive Sampling

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

Dimensionality Reduction

The term **dimensionality reduction** is often reserved for those techniques that reduce the dimensionality of a data set by creating new attributes that are a combination of the old attributes.

A key **benefit** is that many data mining algorithms work better if the dimensionality—the number of attributes in the data—is lower.

A reduction of dimensionality can lead to a more understandable model because the model may involve fewer attributes. Also, dimensionality reduction may allow the data to be more easily visualized.

The Curse of Dimensionality

The curse of dimensionality refers to the phenomenon that many types of data analysis become significantly harder as the dimensionality of the data increases. Specifically, as dimensionality increases, the data becomes increasingly sparse in the space that it occupies.

For classification, this can mean that there are not enough data objects to allow the creation of a model that reliably assigns a class to all possible objects.

For clustering, the definitions of density and the distance between points, which are critical for clustering, become less meaningful.

Linear Algebra Techniques for Dimensionality Reduction

Principal Components Analysis (PCA) is a linear algebra technique for continuous attributes that finds new attributes (principal components) that

(1) are linear combinations of the original attributes,
(2) are orthogonal (perpendicular) to each other, and
(3) capture the maximum amount of variation in the data.

For example, the first two principal components capture as much of the variation in the data as is possible with two orthogonal attributes that are linear combinations of the original attributes.

Singular Value Decomposition (SVD) is a linear algebra technique that is related to PCA and is also commonly used for dimensionality reduction.

Feature Subset Selection

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

Irrelevant features contain almost no useful information for the data mining task at hand. For instance, students’ ID numbers are irrelevant to the task of predicting students’ grade point averages.

There are three standard approaches to feature selection: embedded, filter, and wrapper.

Embedded approaches Feature selection occurs naturally as part of the data mining algorithm. Specifically, during the operation of the data mining algorithm, the algorithm itself decides which attributes to use and which to ignore.

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

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

An Architecture for Feature Subset Selection

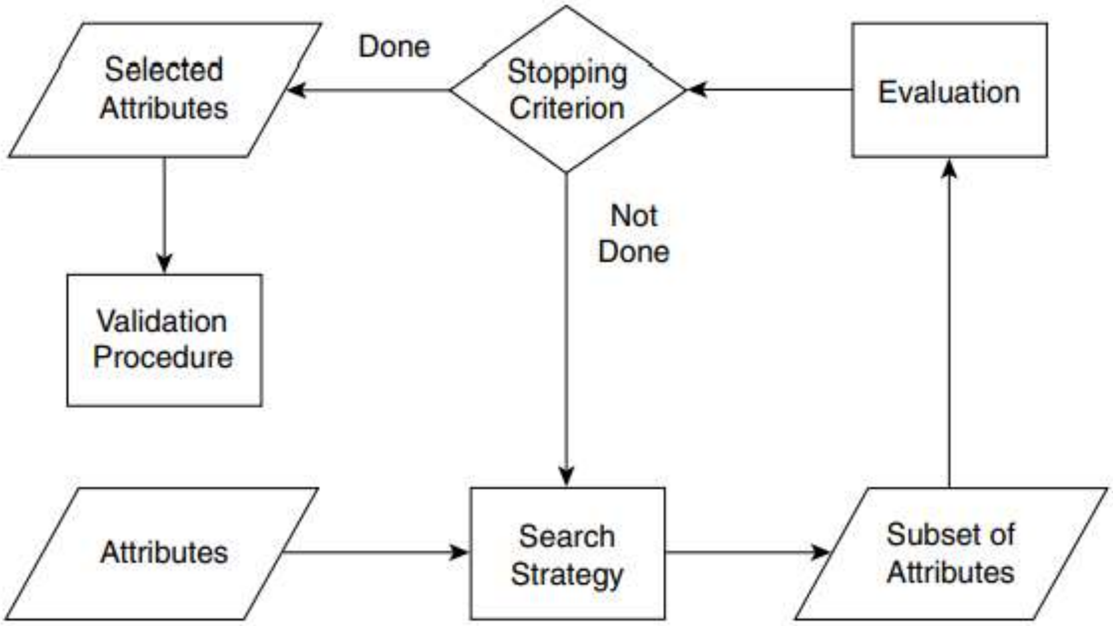


Figure 2.11. Flowchart of a feature subset selection process.

Feature Weighting

Feature weighting is an alternative to keeping or eliminating features. More important features are assigned a higher weight, while less important features are given a lower weight.

These weights are sometimes assigned based on domain knowledge about the relative importance of features. Alternatively, they may be determined automatically.

Feature Creation

Three related methodologies for creating new attributes are described next: feature extraction, mapping the data to a new space, and feature construction.

Feature Extraction

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

Mapping the Data to a New Space

A totally different view of the data can reveal important and interesting features. Consider, for example, time series data, which often contains periodic patterns. If there is only a single periodic pattern and not much noise, then the pattern is easily detected. If, on the other hand, there are a number of periodic patterns and a significant amount of noise is present, then these patterns are hard to detect. Such patterns can, nonetheless, often be detected by applying a **Fourier transform** to the time series in order to change to a representation in which frequency information is explicit.

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

Feature Construction

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

Discretization and Binarization

Algorithms that find association patterns require that the data be in the form of binary attributes. Thus, it is often necessary to transform a continuous attribute into a categorical attribute (**discretization**), and both continuous and discrete attributes may need to be transformed into one or more binary attributes (**binarization**).

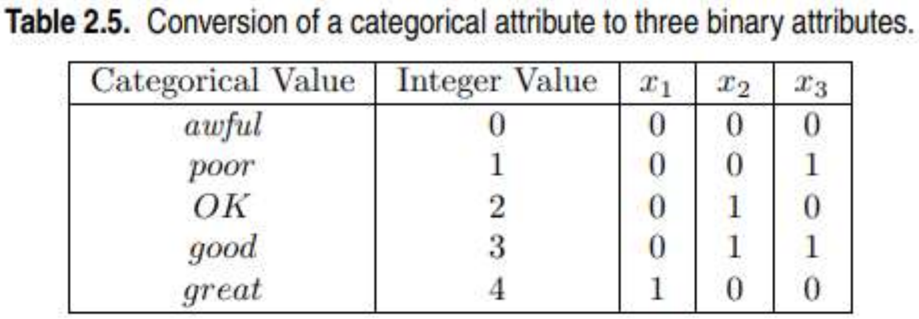


Table 2.5. Conversion of a categorical attribute to three binary attributes.

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

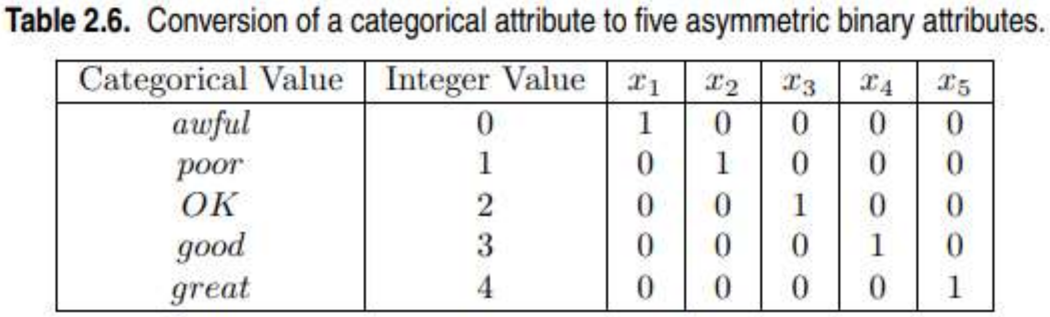


Table 2.6. Conversion of a categorical attribute to five asymmetric binary attributes.

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

Discretization of Continuous Attributes

Entropy

First, it is necessary to define entropy.

Let k be the number of different class labels,

mi be the number of values in the i th interval of a partition, and

mij be the number of values of class j in interval i.

Then the entropy ei of the i th interval is given by the equation

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

where pij = mij/mi 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.,

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

where m is the number of values, wi = mi/m is the fraction of values in the i th interval, and n is the number of intervals.

Variable Transformation

A **variable transformation** refers to a transformation that is applied to all the values of a variable.

Simple Functions

For this type of variable transformation, a simple mathematical function is applied to each value individually. If x is a variable, then examples of such transformations include xk, log x, ex, √x, 1/x, sin x, or |x|. In statistics, variable transformations, especially sqrt, log, and 1/x, are often used to transform data that does not have a Gaussian (normal) distribution into data that does.

Normalization or Standardization

Another common type of variable transformation is the **standardization** or **normalization** of a variable.

The goal of standardization or normalization is to make an entire set of values have a particular property. A traditional example is that of "standardizing a variable" in statistics. If $\bar{x}$ is the mean (average) of the attribute values and S_x is their standard deviation, then the transformation $x' = (x - \bar{x})/S_x$ creates a new variable that has a mean of 0 and a standard deviation of 1.

The mean and standard deviation are strongly affected by **outliers**, so the above transformation is often modified. First, the mean is replaced by the **median**, i.e., the middle value. Second, the standard deviation is replaced by the **absolute standard deviation**. Specifically, if x is a variable, then the absolute standard deviation of x is given by

$$\sigma_A = \sum_{i=1}^{m} |x_i - \mu|$$

where xi is the ith value of the variable, m is the number of objects, and µ is either the mean or median.

Measures of Similarity and Dissimilarity

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

Dissimilarity The dissimilarity between two objects is a numerical measure of the degree to which the two objects are different. Dissimilarities are lower for more similar pairs of objects. Frequently, the term distance is used as a synonym for dissimilarity, although, as we shall see, distance is often used to refer to a special class of dissimilarities.

Dissimilarities sometimes fall in the interval [0, 1], but it is also common for them to range from 0 to ∞.

Dissimilarities between Data Objects

Distances

The **Euclidean distance**, d, between two points, x and y, in one-, two-, three-, or higher dimensional space, is given by the following familiar formula:

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

where n is the number of dimensions and xk and yk are, respectively, the kth attributes (components) of x and y.

The Euclidean distance measure is generalized by the **Minkowski distance** metric.

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

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

of Minkowski distances.

- r = 1. City block (Manhattan, taxicab, L1 norm) distance. A common example is the Hamming distance, which is the number of bits that are different between two objects that have only binary attributes, i.e., between two binary vectors.

- r = 2. Euclidean distance (L2 norm).

- r = ∞. Supremum (Lmax or L∞ norm) distance. This is the maximum difference between any attribute of the objects.

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

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

1. **Positivity**

(a) d(x, x) ≥ 0 for all x and y,

(b) d(x, y) = 0 only if x = y.

2. **Symmetry**

d(x, y) = d(y, x) for all x and y.

3. **Triangle Inequality**

d(x, z) ≤ d(x, y) + d(y, z) for all points x, y, and z

Measures that satisfy all three properties are known as **metrics**.

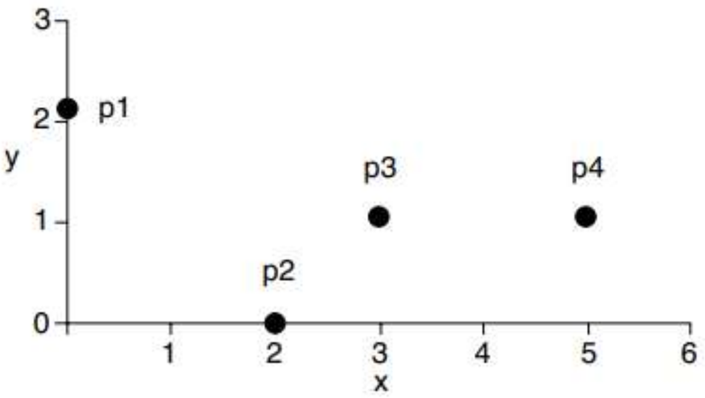
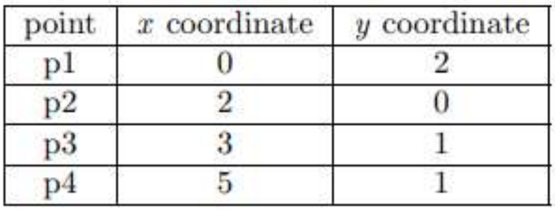


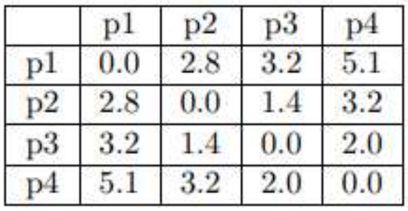
Figure 2.15. Four two-dimensional points.

Table 2.8. x and y coordinates of four points.



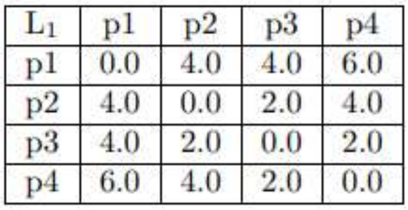
point	x coordinate	y coordinate
p1	0	2
p2	2	0
p3	3	1
p4	5	1

Table 2.9. Euclidean distance matrix for Table 2.8.



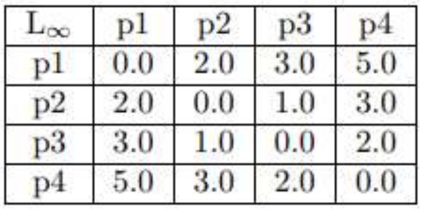
	p1	p2	p3	p4
p1	0.0	2.8	3.2	5.1
p2	2.8	0.0	1.4	3.2
p3	3.2	1.4	0.0	2.0
p4	5.1	3.2	2.0	0.0

Table 2.10. L_1 distance matrix for Table 2.8.



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

Table 2.11. L_∞ distance matrix for Table 2.8.



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

Similarities between Data Objects

For **similarities**, the triangle inequality (or the analogous property) typically does not hold, but symmetry and positivity typically do.

To be explicit, if s(x, y) is the similarity between points x and y, then the typical properties of similarities are the following:

1. s(x, y) = 1 only if x = y. ($0 \le s \le 1$)

2. s(x, y) = s(y, x) for all x and y. (Symmetry)

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

Chapter :- 4

Classification: Basic Concepts, Decision Trees, and Model Evaluation

Classification: Classification is the task of learning a target function f that maps each attribute set x to one of the predefined class labels y.

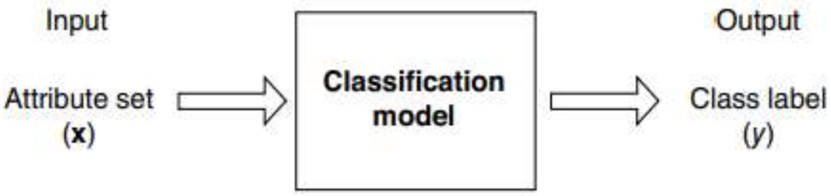


Figure 4.2. Classification as the task of mapping an input attribute set $\mathbf{x}$ into its class label y.

The target function is also known informally as a **classification model.**

Descriptive Modeling A classification model can serve as an explanatory tool to distinguish between objects of different classes.

Predictive Modeling A classification model can also be used to predict the class label of unknown records.

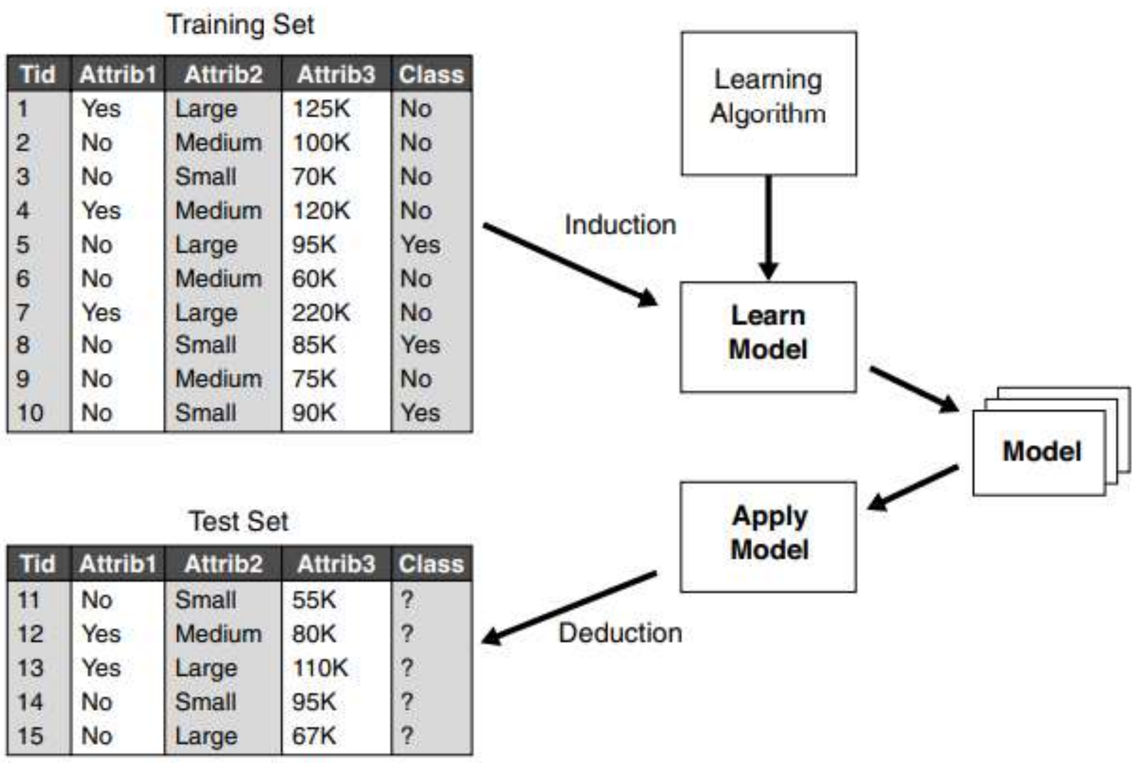
Classification techniques are most suited for predicting or describing data sets with binary or nominal categories. They are less effective for ordinal categories (e.g., to classify a person as a member of high-, medium-, or low-income group) because they do not consider the implicit order among the categories.

General Approach to Solving a Classification Problem

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

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

Evaluation of the performance of a classification model is based on the counts of test records correctly and incorrectly predicted by the model. These counts are tabulated in a table known as a **confusion matrix**.



Training Set

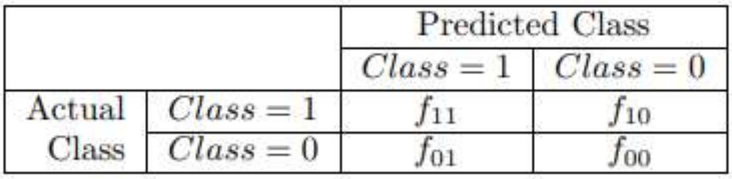
Tid	Attrib1	Attrib2	Attrib3	Class
1	Yes	Large	125K	No
2	No	Medium	100K	No
3	No	Small	70K	No
4	Yes	Medium	120K	No
5	No	Large	95K	Yes
6	No	Medium	60K	No
7	Yes	Large	220K	No
8	No	Small	85K	Yes
9	No	Medium	75K	No
10	No	Small	90K	Yes

Test Set

Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

Figure 4.3. General approach for building a classification model.

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



		Predicted Class	
		$Class = 1$	$Class = 0$
Actual Class	$Class = 1$	f_{11}	f_{10}
	$Class = 0$	f_{01}	f_{00}

Each entry fij in this table denotes the number of records from class i predicted to be of class j. For instance, f01 is the number of records from class 0 incorrectly predicted as class 1. Based on the entries in the confusion matrix, the total number of correct predictions made by the model is (f11 + f00) and the total number of incorrect predictions is (f10 + f01).

This can be done using a performance metric such as accuracy, which is defined as follows:

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

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

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

Decision Tree Induction

The tree has three types of nodes:

- A root node that has no incoming edges and zero or more outgoing edges.

- Internal nodes, each of which has exactly one incoming edge and two or more outgoing edges.

- Leaf or terminal nodes, each of which has exactly one incoming edge and no outgoing edges.

In a decision tree, each leaf node is assigned a class label. The nonterminal nodes, which include the root and other internal nodes, contain attribute test conditions to separate records that have different characteristics.

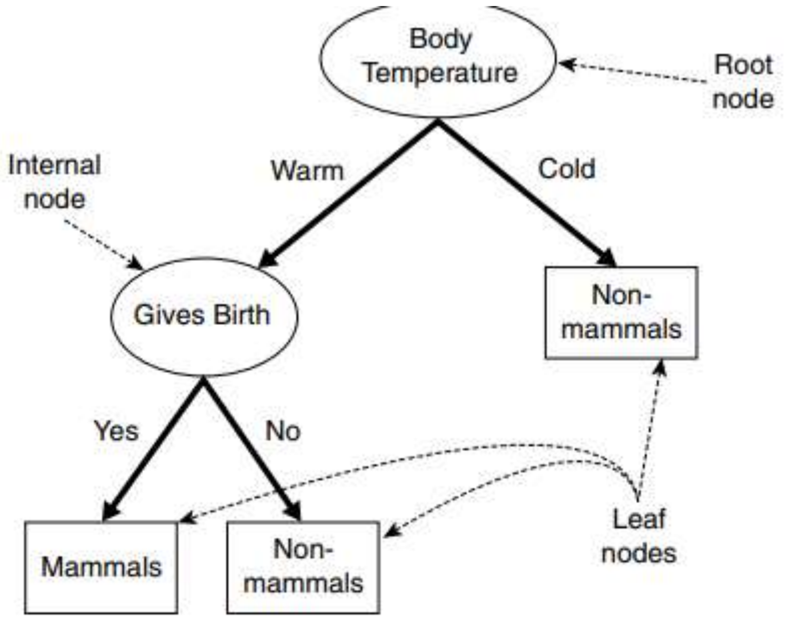


Figure 4.4. A decision tree for the mammal classification problem.

How to Build a Decision Tree

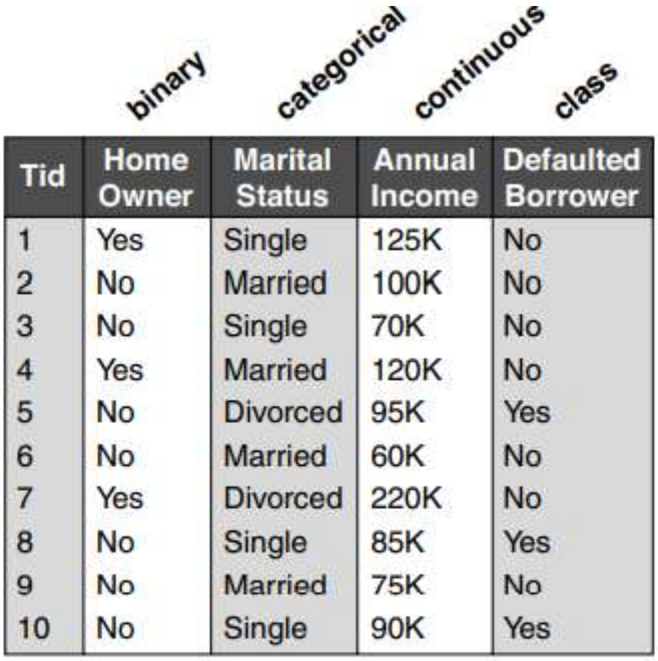
One such algorithm is Hunt's algorithm, which is the basis of many existing decision tree induction algorithms, including ID3, C4.5, and CART.

Hunt's Algorithm

In Hunt's algorithm, a decision tree is grown in a recursive fashion by partitioning the training records into successively purer subsets. Let Dt be the set of training records that are associated with node t and y = {y1, y2,...,yc} be the class labels. The following is a recursive definition of Hunt's algorithm.

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

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


binary
categorical
continuous
class

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

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

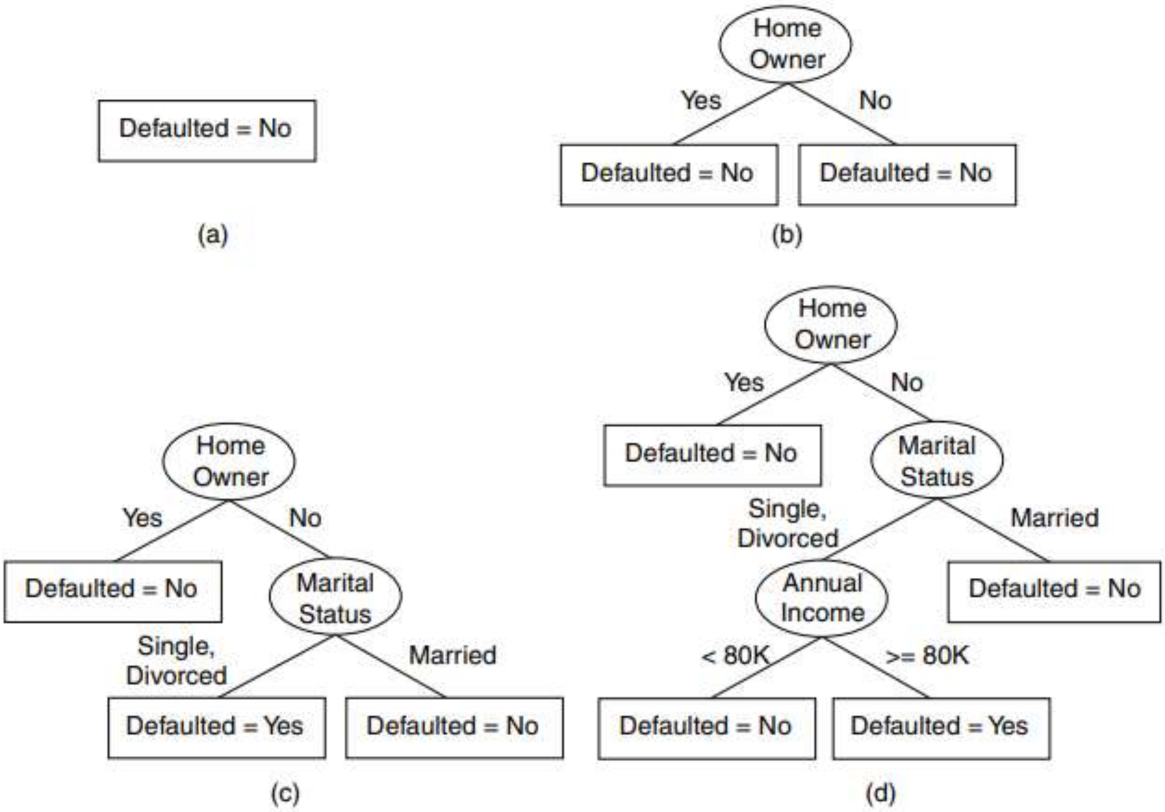


Figure 4.7. Hunt's algorithm for inducing decision trees.

Hunt's algorithm will work if every combination of attribute values is present in the training data and each combination has a unique class label.

These assumptions are too stringent for use in most practical situations. Additional conditions are needed to handle the following cases:

1. It is possible for some of the child nodes created in Step 2 to be empty; i.e., there are no records associated with these nodes. This can happen if none of the training records have the combination of attribute values associated with such nodes. In this case the node is declared a leaf node with the same class label as the majority class of training records associated with its parent node.

2. In Step 2, if all the records associated with Dt have identical attribute values (except for the class label), then it is not possible to split these records any further. In this case, the node is declared a leaf node with the same class label as the majority class of training records associated with this node.

Design Issues of Decision Tree Induction

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

1. **How should the training records be split?** Each recursive step of the tree-growing process must select an attribute test condition to divide the records into smaller subsets. To implement this step, the algorithm must provide a method for specifying the test condition for different attribute types as well as an objective measure for evaluating the goodness of each test condition.

2. **How should the splitting procedure stop?** A stopping condition is needed to terminate the tree-growing process. A possible strategy is to continue expanding a node until either all the records belong to the same class or all the records have identical attribute values. Although both conditions are sufficient to stop any decision tree induction algorithm, other criteria can be imposed to allow the tree-growing procedure to terminate earlier.

Methods for Expressing Attribute Test Conditions

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

Binary Attributes The test condition for a binary attribute generates two potential outcomes.

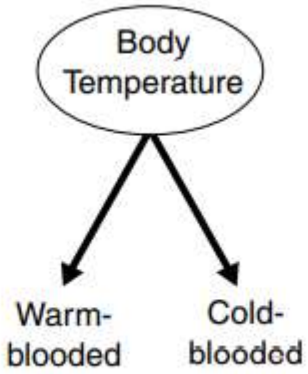
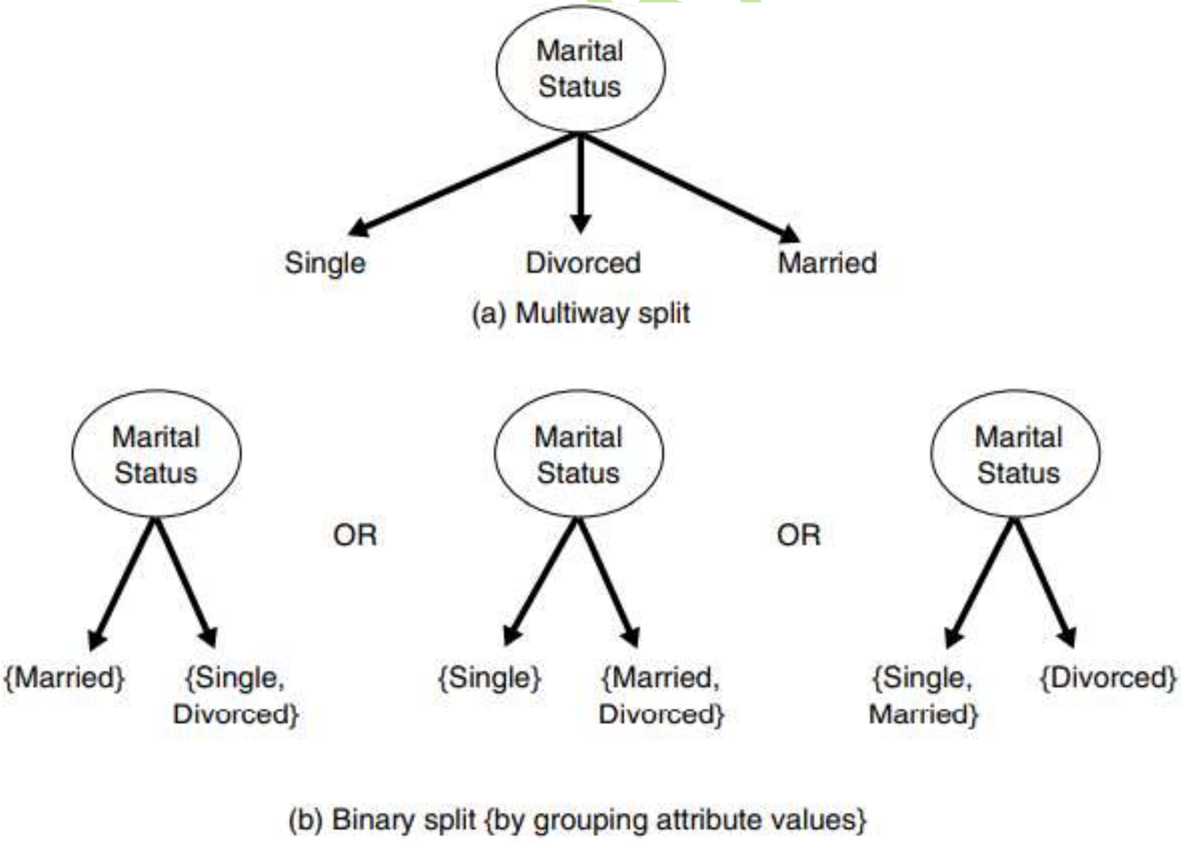


Figure 4.8. Test condition for binary attributes.

Nominal Attributes Since a nominal attribute can have many values, its test condition can be expressed in two ways, as shown in Figure 4.9. For a multiway split (Figure 4.9(a)), the number of outcomes depends on the number of distinct values for the corresponding attribute. For example, if an attribute such as marital status has three distinct values—single, married, or divorced—its test condition will produce a three-way split.



(a) Multiway split

(b) Binary split {by grouping attribute values}

Figure 4.9. Test conditions for nominal attributes.

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

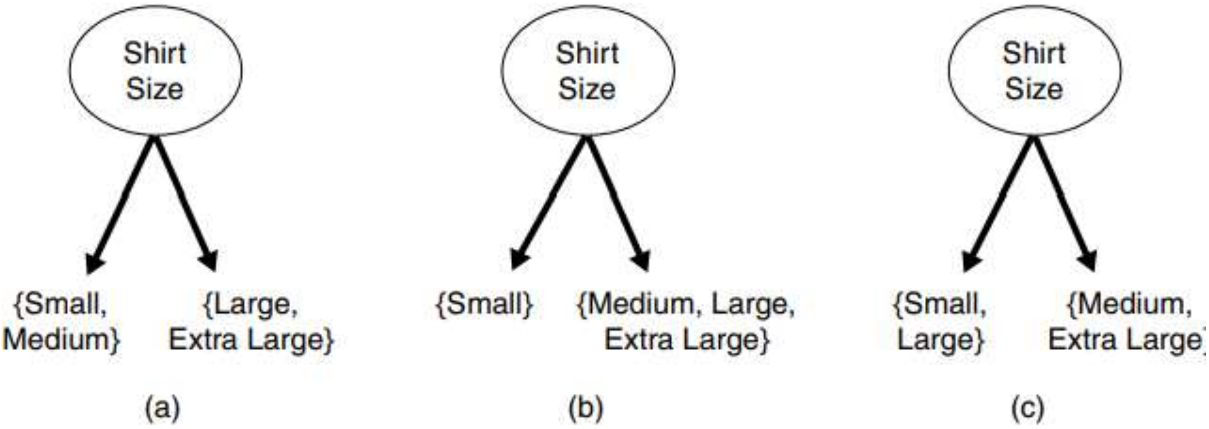


Figure 4.10. Different ways of grouping ordinal attribute values.

Continuous Attributes For continuous attributes, the test condition can be expressed as a comparison test $(A < v)$ or $(A \geq v)$ with binary outcomes, or a range query with outcomes of the form $v_i \leq A < v_{i+1}$, for $i = 1, \ldots, k$. The difference between these approaches is shown in Figure 4.11. For the binary case, the decision tree algorithm must consider all possible split positions v, and it selects the one that produces the best partition. For the multiway split, the algorithm must consider all possible ranges of continuous values.

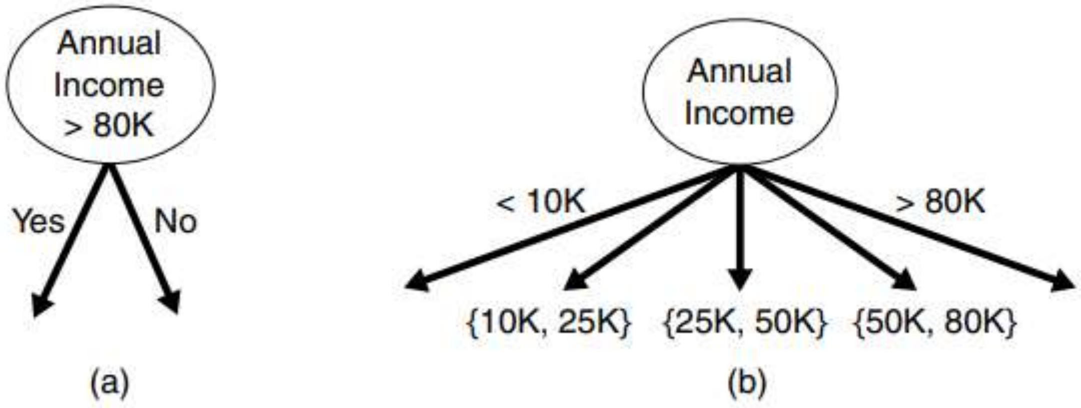


Figure 4.11. Test condition for continuous attributes.

Measures for Selecting the Best Split

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

Let p(i|t) denote the fraction of records belonging to class i at a given node t. We sometimes omit the reference to node t and express the fraction as pi. In a two-class problem, the class distribution at any node can be written as (p0, p1), where p1 = 1 − p0. To illustrate, consider the test conditions shown in Figure 4.12. The class distribution before splitting is (0.5, 0.5) because there are an equal number of records from each class.

If we split the data using the Gender attribute, then the class distributions of the child nodes are (0.6, 0.4) and (0.4, 0.6), respectively. Although the classes are no longer evenly distributed, the child nodes still contain records from both classes. Splitting on the second attribute, Car Type, will result in purer partitions.

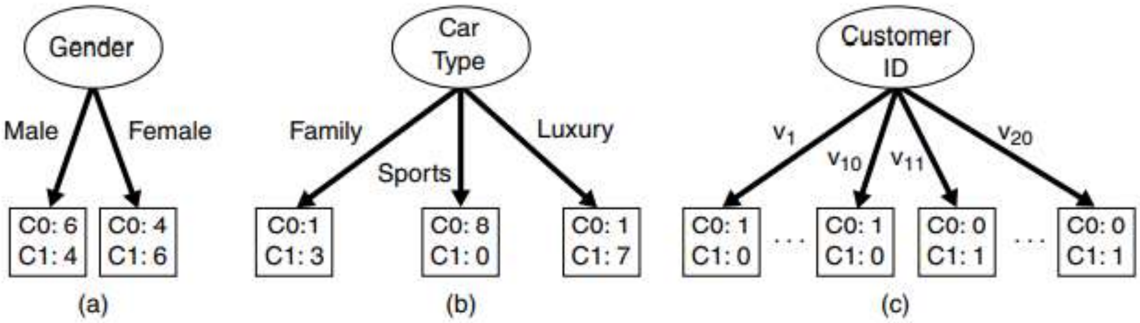


Figure 4.12. Multiway versus binary splits.

For example, a node with class distribution (0, 1) has zero impurity, whereas a node with uniform class distribution (0.5, 0.5) has the highest impurity. Examples of impurity measures include:-

$$\begin{aligned} \text{Entropy}(t) &= -\sum_{i=0}^{c-1} p(i|t)\log_2 p(i|t), \\ \text{Gini}(t) &= 1-\sum_{i=0}^{c-1}[p(i|t)]^2, \\ \text{Classification error}(t) &= 1-\max_i[p(i|t)], \end{aligned}$$

where c is the number of classes and 0 log2 0 = 0 in entropy calculations.

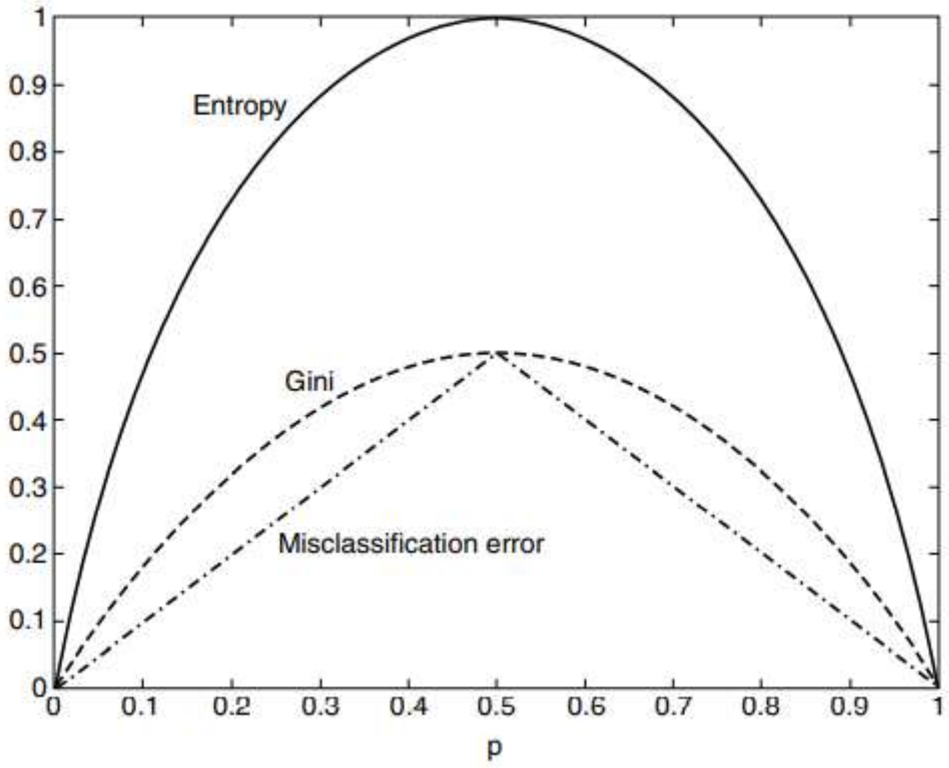
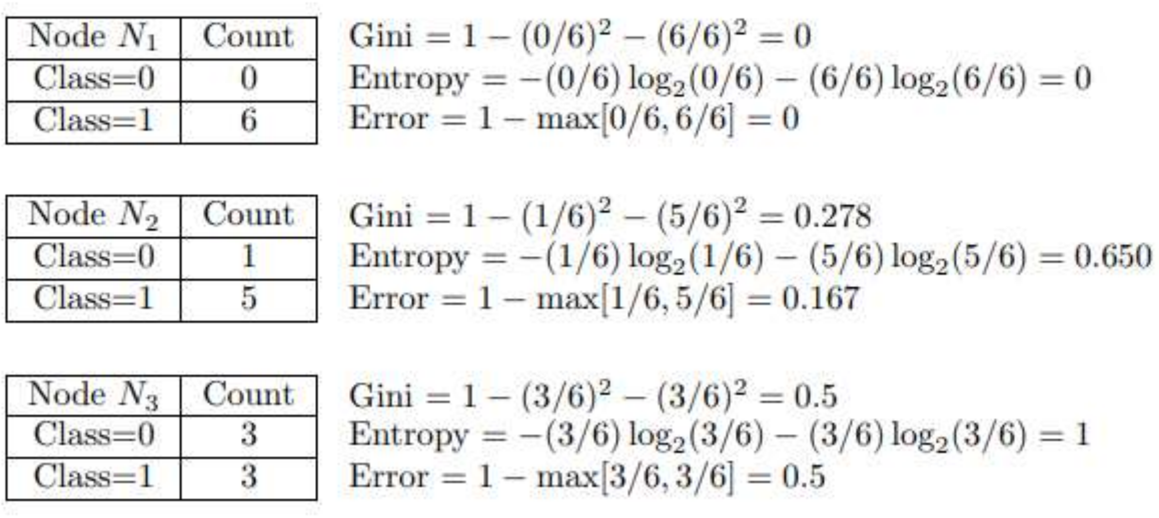


Figure 4.13. Comparison among the impurity measures for binary classification problems.

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



Node N_1	Count
Class=0	0
Class=1	6

Gini $= 1 - (0/6)^2 - (6/6)^2 = 0$
Entropy $= -(0/6)\log_2(0/6) - (6/6)\log_2(6/6) = 0$
Error $= 1 - \max[0/6, 6/6] = 0$

Node N_2	Count
Class=0	1
Class=1	5

Gini $= 1 - (1/6)^2 - (5/6)^2 = 0.278$
Entropy $= -(1/6)\log_2(1/6) - (5/6)\log_2(5/6) = 0.650$
Error $= 1 - \max[1/6, 5/6] = 0.167$

Node N_3	Count
Class=0	3
Class=1	3

Gini $= 1 - (3/6)^2 - (3/6)^2 = 0.5$
Entropy $= -(3/6)\log_2(3/6) - (3/6)\log_2(3/6) = 1$
Error $= 1 - \max[3/6, 3/6] = 0.5$

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

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

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

Splitting of Binary Attributes

Consider the diagram shown in Figure 4.14. Suppose there are two ways to split the data into smaller subsets. Before splitting, the Gini index is 0.5 since there are an equal number of records from both classes. If attribute A is chosen to split the data, the Gini index for node N1 is 0.4898, and for node N2, it is 0.480. The weighted average of the Gini index for the descendent nodes is (7/12) × 0.4898 + (5/12) × 0.480 = 0.486. Similarly, we can show that the weighted average of the Gini index for attribute B is 0.375. Since the subsets for attribute B have a smaller Gini index, it is preferred over attribute A.

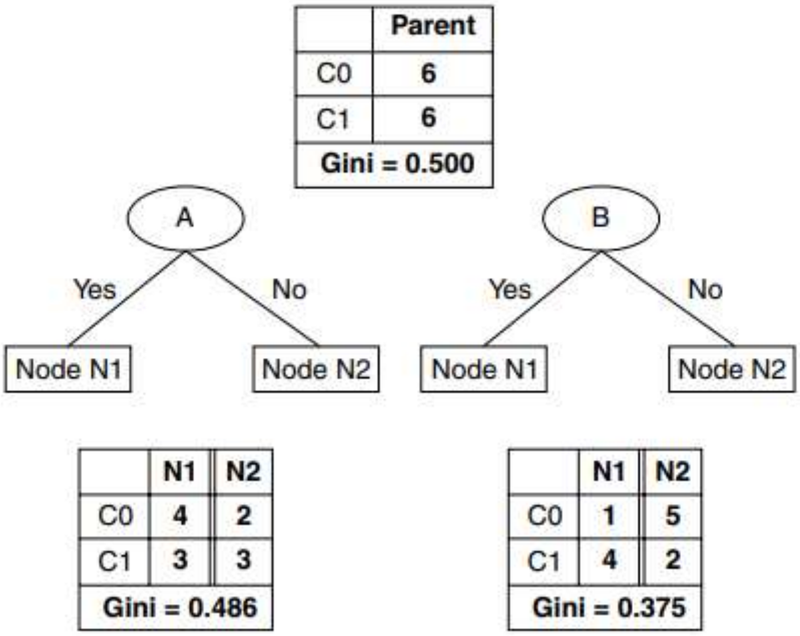


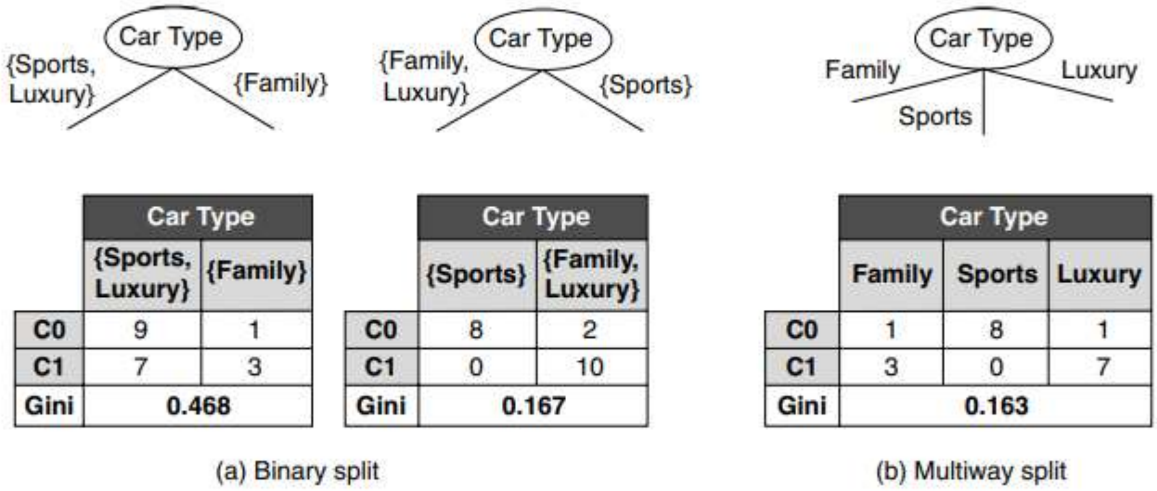
Figure 4.14. Splitting binary attributes.

Splitting of Nominal Attributes

As previously noted, a nominal attribute can produce either binary or multiway splits, as shown in Figure 4.15. The computation of the Gini index for a binary split is similar to that shown for determining binary attributes. For the first binary grouping of the Car Type attribute, the Gini index of {Sports, Luxury} is 0.4922 and the Gini index of {Family} is 0.3750. The weighted average Gini index for the grouping is equal to

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

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



	Car Type	
	{Sports, Luxury}	{Family}
C0	9	1
C1	7	3
Gini	0.468	

	Car Type	
	{Sports}	{Family, Luxury}
C0	8	2
C1	0	10
Gini	0.167	

	Car Type		
	Family	Sports	Luxury
C0	1	8	1
C1	3	0	7
Gini	0.163		

(a) Binary split

(b) Multiway split

Figure 4.15. Splitting nominal attributes.

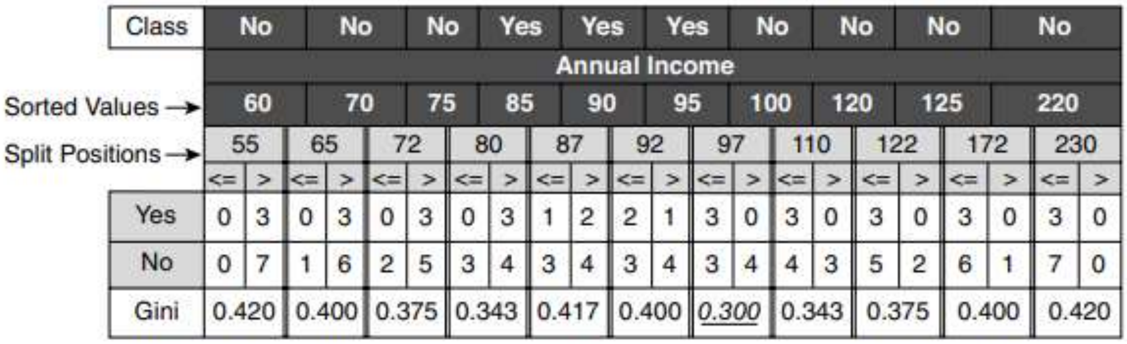
For the multiway split, the Gini index is computed for every attribute value. Since Gini({Family}) = 0.375, Gini({Sports}) = 0, and Gini({Luxury}) = 0.219, the overall Gini index for the multiway split is equal to

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

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

Splitting of Continuous Attributes

Consider the example shown in Figure 4.16, in which the test condition Annual Income ≤ v is used to split the training records for the loan default classification problem. A brute-force method for finding v is to consider every value of the attribute in the N records as a candidate split position. For each candidate v, the data set is scanned once to count the number of records with annual income less than or greater than v. We then compute the Gini index for each candidate and choose the one that gives the lowest value. This approach is computationally expensive because it requires O(N) operations to compute the Gini index at each candidate split position. Since there are N candidates, the overall complexity of this task is O(N2). To reduce the complexity, the training records are sorted based on their annual income, a computation that requires O(N log N) time. Candidate split positions are identified by taking the midpoints between two adjacent sorted values: 55, 65, 72, and so on. However, unlike the brute-force approach, we do not have to examine all N records when evaluating the Gini index of a candidate split position. For the first candidate, v = 55, none of the records has annual income less than \$55K. As a result, the Gini index for the descendent node with Annual Income < \$55K is zero. On the other hand, the number of records with annual income greater than or equal to \$55K is 3 (for class Yes) and 7 (for class No), respectively. Thus, the Gini index for this node is 0.420. The overall Gini index for this candidate split position is equal to 0 × 0+1 × 0.420 = 0.420. For the second candidate, v = 65, we can determine its class distribution by updating the distribution of the previous candidate. More specifically, the new distribution is obtained by examining the class label of the record with the lowest annual income (i.e., \$60K). Since the class label for this record is No, the count for class No is increased from 0 to 1 (for Annual Income ≤ \$65K) and is decreased from 7 to 6 (for Annual Income > \$65K). The distribution for class Yes remains unchanged. The new weighted-average Gini index for this candidate split position is 0.400. This procedure is repeated until the Gini index values for all candidates are computed, as shown in Figure 4.16. The best split position corresponds to the one that produces the smallest Gini index, i.e., v = 97. This procedure is less expensive because it requires a constant amount of time to update the class distribution at each candidate split position. It can be further optimized by considering only candidate split positions located between two adjacent records with different class labels. For example, because the first three sorted records (with annual incomes \$60K, \$70K, and \$75K) have identical class labels, the best split position should not reside between \$60K and \$75K. Therefore, the candidate split positions at v = \$55K, \$65K, \$72K, \$87K, \$92K, \$110K, \$122K, \$172K, and \$230K are ignored because they are located between two adjacent records with the same class labels. This approach allows us to reduce the number of candidate split positions from 11 to 2.



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

Figure 4.16. Splitting continuous attributes.

Gain Ratio

In the decision tree algorithm, a splitting criterion known as **gain ratio** is used to determine the goodness of a split. This criterion is defined as follows:

$$\text{Gain ratio} = \frac{\Delta_{\text{info}}}{\text{Split Info}}.$$

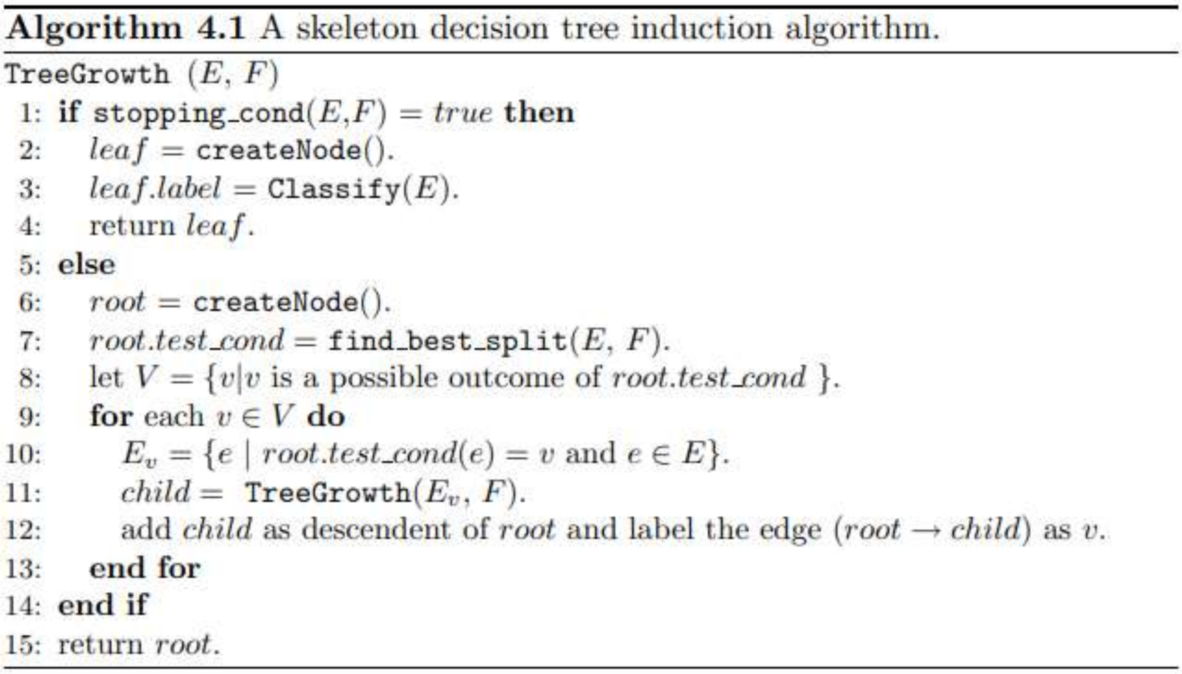
Here,

$$\text{Split Info} = -\sum_{i=1}^{k} P(v_i)\log_2 P(v_i)$$

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

Algorithm for Decision Tree Induction

A skeleton decision tree induction algorithm called TreeGrowth is shown in Algorithm 4.1. The input to this algorithm consists of the training records E and the attribute set F. The algorithm works by recursively selecting the best attribute to split the data (Step 7) and expanding the leaf nodes of the tree (Steps 11 and 12) until the stopping criterion is met (Step 1).



Algorithm 4.1 A skeleton decision tree induction algorithm.

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

The details of this algorithm are explained below:

1. The **createNode()** function extends the decision tree by creating a new node. A node in the decision tree has either a test condition, denoted as node.test cond, or a class label, denoted as node.label.

2. The **find_best_split()** function determines which attribute should be selected as the test condition for splitting the training records. As previously noted, the choice of test condition depends on which impurity measure is used to determine the goodness of a split. Some widely used measures include entropy, the Gini index, and the χ2 statistic.

3. The **Classify()** function determines the class label to be assigned to a leaf node. For each leaf node t, let p(i|t) denote the fraction of training records from class i associated with the node t. In most cases, the leaf node is assigned to the class that has the majority number of training records:

$$leaf.label = \operatorname*{argmax}_{i} \; p(i|t),$$

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

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

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

Evaluating the Performance of a Classifier

The estimated error helps the learning algorithm to do model selection; i.e., to find a model of the right complexity that is not susceptible to overfitting. Once the model has been constructed, it can be applied to the test set to predict the class labels of previously unseen records.

It is often useful to measure the performance of the model on the test set because such a measure provides an unbiased estimate of its generalization error. The accuracy or error rate computed from the test set can also be used to compare the relative performance of different classifiers on the same domain. However, in order to do this, the class labels of the test records must be known.

Some of the methods commonly used to evaluate the performance of a classifier

Holdout Method

In the holdout method, the original data with labeled examples is partitioned into two disjoint sets, called the **training** and the **test sets**, respectively.

A classification model is then induced from the training set and its performance is evaluated on the test set. The proportion of data reserved for training and for testing is typically at the discretion of the analysts (e.g., 50-50 or two thirds for training and one-third for testing).

The accuracy of the classifier can be estimated based on the accuracy of the induced model on the test set. The holdout method has several well-known limitations. **First**, fewer labeled examples are available for training because some of the records are with held for testing. As a result, the induced model may not be as good as when all the labeled examples are used for training. **Second**, the model may be highly dependent on the composition of the training and test sets. The smaller the training set size, the larger the variance of the model.

On the other hand, if the training set is too large, then the estimated accuracy computed from the smaller test set is less reliable. Such an estimate is said to have a wide confidence interval. Finally, the training and test sets are no longer independent of each other. Because the training and test sets are subsets of the original data, a class that is overrepresented in one subset will be underrepresented in the other, and vice versa.

Random Subsampling

The holdout method can be repeated several times to improve the estimation of a classifier’s performance. This approach is known as random subsampling. Let acc$_i$ be the model accuracy during the ith iteration. The overall accuracy is given by

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

Random subsampling still encounters some of the problems associated with the holdout method because it does not utilize as much data as possible for training. It also has no control over the number of times each record is used for testing and training. Consequently, some records might be used for training more often than others.

Cross-Validation

An alternative to random subsampling is cross-validation. In this approach, each record is used the same number of times for training and exactly once for testing.

To illustrate this method, suppose we partition the data into two equal-sized subsets. First, we choose one of the subsets for training and the other for testing. We then swap the roles of the subsets so that the previous training set becomes the test set and vice versa. This approach is called a **two fold cross-validation**. The total error is obtained by summing up the errors for both runs. In this example, each record is used exactly once for training and once for testing.

The **k-fold cross-validation** method generalizes this approach by segmenting the data into k equal-sized partitions. During each run, one of the partitions is chosen for testing, while the rest of them are used for training. This procedure is repeated k times so that each partition is used for testing exactly once. Again, the total error is found by summing up the errors for all k runs.

A special case of the k-fold cross-validation method sets k = N, the size of the data set. In this so-called **leave-one-out** approach, each test set contains only one record. This approach has the advantage of utilizing as much data as possible for training.

In addition, the test sets are **mutually exclusive** and they effectively cover the entire data set.

The drawback of this approach is that it is computationally expensive to **repeat** the procedure N times.

Furthermore, since each test set contains only **one** record, the variance of the estimated performance metric tends to be high.

Bootstrap

The methods presented so far assume that the training records are sampled without replacement. As a result, there are no duplicate records in the training and test sets.

In the bootstrap approach, the training records are sampled with replacement; i.e., a record already chosen for training is put back into the original pool of records so that it is equally likely to be redrawn. If the original data has N records, it can be shown that, on average, a bootstrap sample of size N contains about 63.2% of the records in the original data.

This approximation follows from the fact that the probability a record is chosen by a bootstrap sample is $1 - (1 - 1/N)^N$. When N is sufficiently large, the probability asymptotically approaches $1 – e^{-1} = 0.632$. Records that are not included in the bootstrap sample become part of the test set. The model induced from the training set is then applied to the test set to obtain an estimate of the accuracy of the bootstrap sample, i.

The sampling procedure is then repeated b times to generate b bootstrap samples. There are several variations to the bootstrap sampling approach in terms of how the overall accuracy of the classifier is computed. One of the more widely used approaches is the .632 bootstrap, which computes the overall accuracy by combining the accuracies of each bootstrap sample (e_i) with the accuracy computed from a training set that contains all the labeled examples in the original data (acc_s):

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

Chapter – 5 : Classification: Alternative Techniques

Alternative techniques for building classification models—from simple techniques such as **rule-based** and **nearest-neighbour classifiers** to more advanced techniques such as support **vector machines** and **ensemble methods**.

Rule-Based Classifier

A rule-based classifier is a technique for classifying records using a collection of "if ...then..." rules. Table 5.1 shows an example of a model generated by a rule-based classifier for the vertebrate classification problem. The rules for the model are represented in a disjunctive normal form, R = (r_1 V r_2 V ... r_k), where R is known as the rule set and r_i's are the classification rules or disjuncts.

Table 5.1. Example of a rule set for the vertebrate classification problem.

r_1:	(Gives Birth = no) ∧ (Aerial Creature = yes) ⟶ Birds
r_2:	(Gives Birth = no) ∧ (Aquatic Creature = yes) ⟶ Fishes
r_3:	(Gives Birth = yes) ∧ (Body Temperature = warm-blooded) ⟶ Mammals
r_4:	(Gives Birth = no) ∧ (Aerial Creature = no) ⟶ Reptiles
r_5:	(Aquatic Creature = semi) ⟶ Amphibians

Each classification rule can be expressed in the following way:

$$r_i : \quad (Condition_i) \longrightarrow y_i.$$

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

$$Condition_i = (A_1 \ op \ v_1) \wedge (A_2 \ op \ v_2) \wedge \ldots (A_k \ op \ v_k),$$

where (Aj , vj) is an attribute-value pair and op is a logical operator chosen from the set {=, !=, , ≤, ≥}. Each attribute test (Aj op vj) is known as a conjunct. The right-hand side of the rule is called the **rule consequent**, which contains the predicted class yi.

A rule r covers a record x if the precondition of r matches the attributes of x. r is also said to be fired or triggered whenever it covers a given record. For an illustration, consider the rule r1 given in Table 5.1 and the following attributes for two vertebrates: hawk and grizzly bear.

Name	Body Temperature	Skin Cover	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hiber-nates
hawk	warm-blooded	feather	no	no	yes	yes	no
grizzly bear	warm-blooded	fur	yes	no	no	yes	yes

r1 covers the first vertebrate because its precondition is satisfied by the hawk's attributes. The rule does not cover the second vertebrate because grizzly bears give birth to their young and cannot fly, thus violating the precondition of r1.

The quality of a classification rule can be evaluated using measures such as coverage and accuracy. Given a data set D and a classification rule r : A −→ y, the coverage of the rule is defined as the fraction of records in D that trigger the rule r. On the other hand, its accuracy or confidence factor is defined as the fraction of records triggered by r whose class labels are equal to y. The formal definitions of these measures are

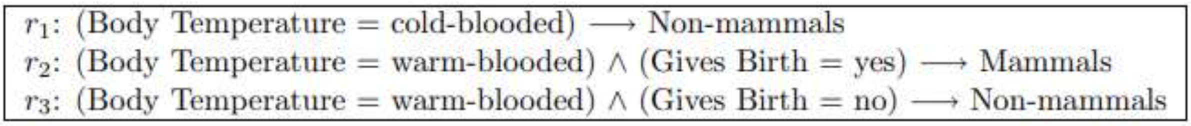
$$\begin{aligned} \text{Coverage}(r) &= \frac{|A|}{|D|} \\ \text{Accuracy}(r) &= \frac{|A \cap y|}{|A|}, \end{aligned}$$

where |A| is the number of records that satisfy the rule antecedent, |A ∩ y| is the number of records that satisfy both the antecedent and consequent, and |D| is the total number of records.

How a Rule-Based Classifier Works

Mutually Exclusive Rules The rules in a rule set R are mutually exclusive if no two rules in R are triggered by the same record. This property ensures that every record is covered by at most one rule in R. An example of a mutually exclusive rule set is shown in Table 5.3.

Table 5.3. Example of a mutually exclusive and exhaustive rule set.



r_1: (Body Temperature = cold-blooded) $\longrightarrow$ Non-mammals
r_2: (Body Temperature = warm-blooded) $\wedge$ (Gives Birth = yes) $\longrightarrow$ Mammals
r_3: (Body Temperature = warm-blooded) $\wedge$ (Gives Birth = no) $\longrightarrow$ Non-mammals

Exhaustive Rules A rule set R has exhaustive coverage if there is a rule for each combination of attribute values. This property ensures that every record is covered by at least one rule in R. Assuming that Body Temperature and Gives Birth are binary variables, the rule set shown in Table 5.3 has exhaustive coverage.

Together, these properties ensure that every record is covered by exactly one rule. Unfortunately, many rule-based classifiers, including the one shown in Table 5.1, do not have such properties. If the rule set is not exhaustive, then a default rule, r_d : () –→ y_d, must be added to cover the remaining cases.

A default rule has an empty antecedent and is triggered when all other rules have failed. Y_d is known as the default class and is typically assigned to the majority class of training records not covered by the existing rules. If the rule set is not mutually exclusive, then a record can be covered by several rules, some of which may predict conflicting classes.

There are two ways to overcome this problem.

Ordered Rules In this approach, the rules in a rule set are ordered in decreasing order of their priority, which can be defined in many ways (e.g., based on accuracy, coverage, total description length, or the order in which the rules are generated). An ordered rule set is also known as a decision list. When a test record is presented, it is classified by the highest-ranked rule that covers the record. This avoids the problem of having conflicting classes predicted by multiple classification rules.

Unordered Rules This approach allows a test record to trigger multiple classification rules and considers the consequent of each rule as a vote for a particular class. The votes are then tallied to determine the class label of the test record. The record is usually assigned to the class that receives the highest number of votes.

In some cases, the vote may be weighted by the rule's accuracy. Using unordered rules to build a rule-based classifier has both advantages and disadvantages. Unordered rules are less susceptible to errors caused by the wrong rule being selected to classify a test record (unlike classifiers based on ordered rules, which are sensitive to the choice of rule ordering criteria).

Model building is also less expensive because the rules do not have to be kept in sorted order. Nevertheless, classifying a test record can be quite an expensive task because the attributes of the test record must be compared against the precondition of every rule in the rule set.

Rule-Ordering Schemes

Rule ordering can be implemented on a **rule-by-rule** basis or on a **class-by-class** basis.

Rule-Based Ordering Scheme This approach orders the individual rules by some rule quality measure. This ordering scheme ensures that every test record is classified by the "best" rule covering it. A potential drawback of this scheme is that lower-ranked rules are much harder to interpret because they assume the negation of the rules preceding them.

Class-Based Ordering Scheme In this approach, rules that belong to the same class appear together in the rule set R. The rules are then collectively sorted on the basis of their class information. The relative ordering among the rules from the same class is not important; as long as one of the rules fires, the class will be assigned to the test record. This makes rule interpretation slightly easier. However, it is possible for a high-quality rule to be overlooked in favour of an inferior rule that happens to predict the higher-ranked class.

How to Build a Rule-Based Classifier

To build a rule-based classifier, we need to extract a set of rules that identifies key relationships between the attributes of a data set and the class label.

There are two broad classes of methods for extracting classification rules:

(1) **direct methods**, which extract classification rules directly from data, and

(2) **indirect methods**, which extract classification rules from other classification models, such as decision trees and neural networks.

Direct Methods for Rule Extraction

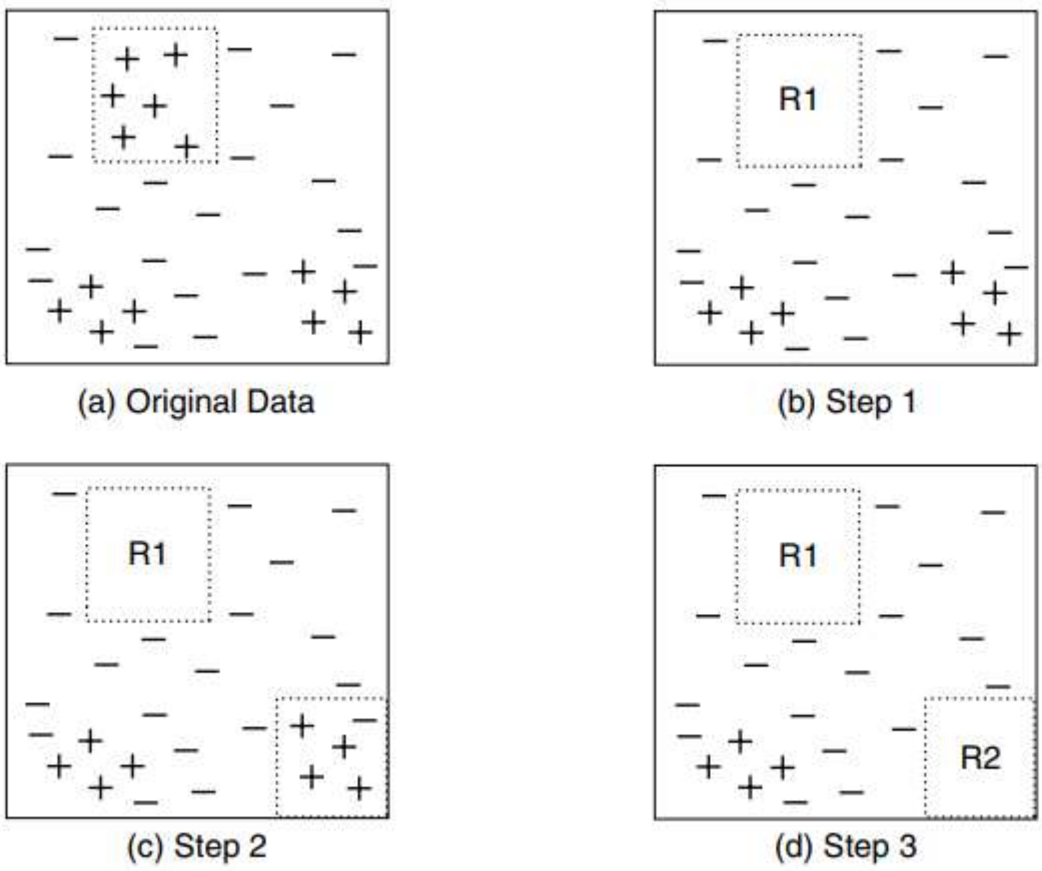
The **sequential covering algorithm** is often used to extract rules directly from data. Rules are grown in a greedy fashion based on a certain evaluation measure.

The algorithm starts with an empty decision list, R. The **Learn-One-Rule** function is then used to extract the best rule for class y that covers the current set of training records. During rule extraction, all training records for class y are considered to be positive examples, while those that belong to other classes are considered to be negative examples. A rule is desirable if it covers most of the positive examples and none (or very few) of the negative examples. Once such a rule is found, the training records covered by the rule are eliminated. The new rule is added to the bottom of the decision list R. This procedure is repeated until the stopping criterion is met. The algorithm then proceeds to generate rules for the next class.

Algorithm 5.1 Sequential covering algorithm.

```
Let $E$ be the training records and $A$ be the set of attribute-value pairs, $\{(A_j, v_j)\}$.
Let $Y_o$ be an ordered set of classes $\{y_1, y_2, \ldots, y_k\}$.
Let $R = \{\ \}$ be the initial rule list.
for each class $y \in Y_o - \{y_k\}$ do
  while stopping condition is not met do
    $r \leftarrow$ Learn-One-Rule $(E,\ A,\ y)$.
    Remove training records from $E$ that are covered by $r$.
    Add $r$ to the bottom of the rule list: $R \longrightarrow R \vee r$.
  end while
end for
Insert the default rule, $\{\} \longrightarrow y_k$, to the bottom of the rule list $R$.
```

Figure 5.2 demonstrates how the sequential covering algorithm works for a data set that contains a collection of positive and negative examples. The rule R1, whose coverage is shown in Figure 5.2(b), is extracted first because it covers the largest fraction of positive examples. All the training records covered by R1 are subsequently removed and the algorithm proceeds to look for the next best rule, which is R2.



(a) Original Data

(b) Step 1

(c) Step 2

(d) Step 3

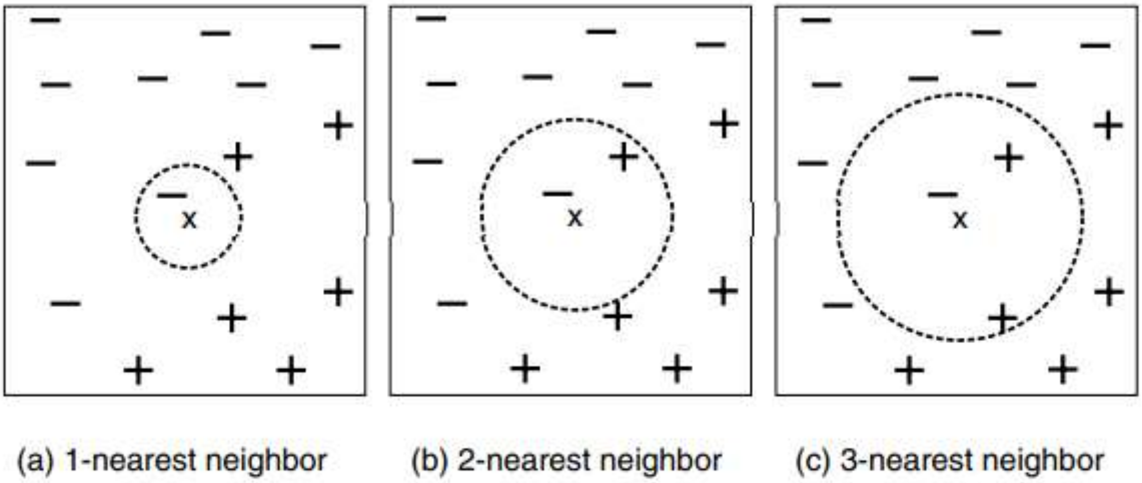
Figure 5.2. An example of the sequential covering algorithm.

Nearest-Neighbour classifiers

The classification framework shown in Figure 4.3 involves a two-step process: (1) an inductive step for constructing a classification model from data, and

(2) a deductive step for applying the model to test examples.

Decision tree and rule-based classifiers are examples of **eager learners** because they are designed to learn a model that maps the input attributes to the class label as soon as the training data becomes available. An opposite strategy would be to delay the process of modeling the training data until it is needed to classify the test examples. Techniques that employ this strategy are known as **lazy learners**. An example of a lazy learner is the **Rote classifier**, which memorizes the entire training data and performs classification only if the attributes of a test instance match one of the training examples exactly.



(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

Figure 5.7. The 1-, 2-, and 3-nearest neighbors of an instance.

An obvious drawback of this approach is that some test records may not be classified because they do not match any training example.

One way to make this approach more flexible is to find all the training examples that are relatively similar to the attributes of the test example. These examples, which are known as **nearest neighbors**, can be used to determine the class label of the test example. The justification for using nearest neighbors is best exemplified by the following saying: "If it walks like a duck, quacks like a duck, and looks like a duck, then it's probably a duck." A nearest-neighbor classifier represents each example as a data point in a d-dimensional space, where d is the number of attributes. The k-nearest neighbors of a given example z refer to the k points that are closest to z.

Figure 5.7 illustrates the 1-, 2-, and 3-nearest neighbors of a data point located at the center of each circle. The data point is classified based on the class labels of its neighbors. In the case where the neighbors have more than one label, the data point is assigned to the majority class of its nearest neighbors. In Figure 5.7(a), the 1-nearest neighbor of the data point is a negative example. Therefore the data point is assigned to the negative class. If the number of nearest neighbors is three, as shown in Figure 5.7(c), then the neighborhood contains two positive examples and one negative example. Using the majority voting scheme, the data point is assigned to the positive class. In the case where there is a tie between the classes (see Figure 5.7(b)), we may randomly choose one of them to classify the data point.

The preceding discussion underscores the importance of choosing the right value for k. If k is too small, then the nearest-neighbor classifier may be susceptible to overfitting because of noise in the training data. On the other hand, if k is too large, the nearest-neighbor classifier may misclassify the test instance because its list of nearest neighbors may include data points that are located far away from its neighborhood (see Figure 5.8).

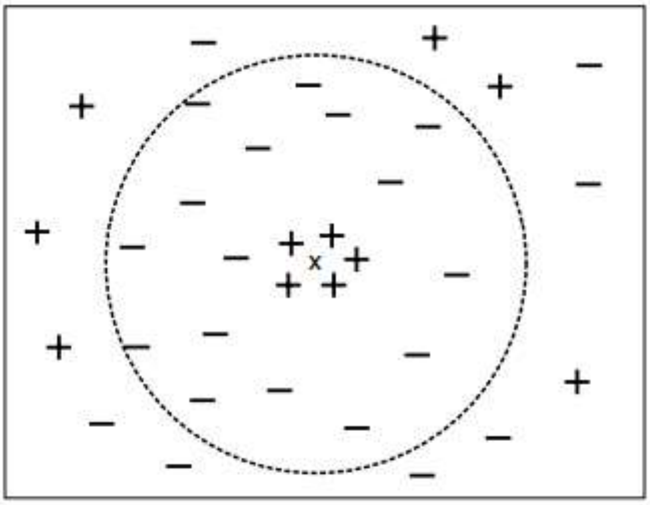
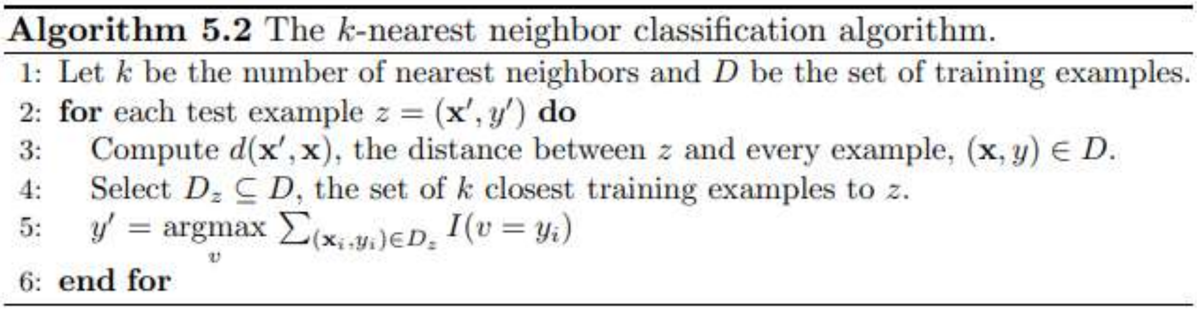


Figure 5.8. k-nearest neighbor classification with large k.

Algorithm

The algorithm computes the distance (or similarity) between each test example z = (x' , y') and all the training examples (x, y) ∈ D to determine its nearest-neighbor list, Dz.

Such computation can be costly if the number of training examples is large. However, efficient indexing techniques are available to reduce the amount of computations needed to find the nearest neighbors of a test example.



Algorithm 5.2 The k-nearest neighbor classification algorithm.

```
Let $k$ be the number of nearest neighbors and $D$ be the set of training examples.
for each test example $z = (\mathbf{x}', y')$ do
    Compute $d(\mathbf{x}', \mathbf{x})$, the distance between $z$ and every example, $(\mathbf{x}, y) \in D$.
    Select $D_z \subseteq D$, the set of $k$ closest training examples to $z$.
    $y' = \underset{v}{\operatorname{argmax}} \sum_{(\mathbf{x}_i, y_i) \in D_z} I(v = y_i)$
end for
```

Once the nearest-neighbor list is obtained, the test example is classified based on the majority class of its nearest neighbors:

$$\text{Majority Voting:} \quad y' = \underset{v}{\operatorname{argmax}} \sum_{(\mathbf{x}_i, y_i) \in D_z} I(v = y_i).$$

where v is a class label, yi is the class label for one of the nearest neighbors, and I(·) is an indicator function that returns the value 1 if its argument is true and 0 otherwise.

Using the distance-weighted voting scheme, the class label can be determined as follows:

$$\text{Distance-Weighted Voting:} \quad y' = \underset{v}{\operatorname{argmax}} \sum_{(\mathbf{x}_i, y_i) \in D_z} w_i \times I(v = y_i).$$

Characteristics of Nearest-Neighbor Classifiers

The characteristics of the nearest-neighbor classifier are summarized below:

- Nearest-neighbor classification is part of a more general technique known as instance-based learning, which uses specific training instances to make predictions without having to maintain an abstraction (or model) derived from data. Instance-based learning algorithms require a proximity measure to determine the similarity or distance between instances and a classification function that returns the predicted class of a test instance based on its proximity to other instances.

- Lazy learners such as nearest-neighbor classifiers do not require model building. However, classifying a test example can be quite expensive because we need to compute the proximity values individually between the test and training examples. In contrast, eager learners often spend the bulk of their computing resources for model building. Once a model has been built, classifying a test example is extremely fast.

- Nearest-neighbor classifiers make their predictions based on local information, whereas decision tree and rule-based classifiers attempt to find 226 5.3 Bayesian Classifiers a global model that fits the entire input space. Because the classification decisions are made locally, nearest-neighbor classifiers (with small values of k) are quite susceptible to noise.

- Nearest-neighbor classifiers can produce arbitrarily shaped decision boundaries. Such boundaries provide a more flexible model representation compared to decision tree and rule-based classifiers that are often constrained to rectilinear decision boundaries. The decision boundaries of nearest-neighbor classifiers also have high variability because they depend on the composition of training examples. Increasing the number of nearest neighbors may reduce such variability.

- Nearest-neighbor classifiers can produce wrong predictions unless the appropriate proximity measure and data preprocessing steps are taken. For example, suppose we want to classify a group of people based on attributes such as height (measured in meters) and weight (measured in pounds). The height attribute has a low variability, ranging from 1.5 m to 1.85 m, whereas the weight attribute may vary from 90 lb. to 250 lb. If the scale of the attributes are not taken into consideration, the proximity measure may be dominated by differences in the weights of a person.

Bayesian Classifiers

In many applications the relationship between the attribute set and the class variable is non-deterministic. In other words, the class label of a test record cannot be predicted with certainty even though its attribute set is identical to some of the training examples.

Bayes Theorem

Consider a football game between two rival teams: Team 0 and Team 1. Suppose Team 0 wins 65% of the time and Team 1 wins the remaining matches. Among the games won by Team 0, only 30% of them come from playing on Team 1's *football field. On the other hand, 75% of the victories for Team 1 are obtained while playing at home. If Team 1 is to host the next match between the two teams, which team will most likely emerge as the winner?*

Let X and Y be a pair of random variables. Their **joint probability**, **P(X = x, Y = y)**, refers to the probability that variable X will take on the value x and variable Y will take on the value y.

A **conditional probability** is the probability that a random variable will take on a particular value given that the outcome for another random variable is known.

For example, the **conditional probability P(Y = y|X = x)** refers to the probability that the variable Y will take on the value y, given that the variable X is observed to have the value x.

The joint and conditional probabilities for X and Y are related in the following way:

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

Rearranging the last two expressions in above Equation leads to the following formula, known as the **Bayes theorem**:

$$P(Y|X) = P(X|Y)P(Y)\ P(X)$$

The Bayes theorem can be used to solve the prediction problem stated at the beginning of this section. For notational convenience, let X be the random variable that represents the team hosting the match and Y be the random variable that represents the winner of the match. Both X and Y can take on values from the set {0, 1}.

We can summarize the information given in the problem as follows:

Probability Team 0 wins is P(Y = 0) = 0.65.

Probability Team 1 wins is P(Y = 1) = 1 − P(Y = 0) = 0.35.

Probability Team 1 hosted the match it won is P(X = 1|Y = 1) = 0.75.

Probability Team 1 hosted the match won by Team 0 is P(X = 1|Y = 0) = 0.3.

Our objective is to compute P(Y = 1|X = 1), which is the conditional probability that Team 1 wins the next match it will be hosting, and compares it against P(Y = 0|X = 1). Using the Bayes theorem, we obtain

$$\begin{aligned}
P(Y=1|X=1) &= \frac{P(X=1|Y=1)\times P(Y=1)}{P(X=1)} \\
&= \frac{P(X=1|Y=1)\times P(Y=1)}{P(X=1,Y=1)+P(X=1,Y=0)} \\
&= \frac{P(X=1|Y=1)\times P(Y=1)}{P(X=1|Y=1)P(Y=1)+P(X=1|Y=0)P(Y=0)} \\
&= \frac{0.75\times 0.35}{0.75\times 0.35+0.3\times 0.65} \\
&= 0.5738,
\end{aligned}$$

where the law of total probability (see Equation C.5 on page 722) was applied in the second line. Furthermore, P(Y = 0|X = 1) = 1 − P(Y = 1|X = 1) = 0.4262. Since P(Y = 1|X = 1) > P(Y = 0|X = 1), Team 1 has a better chance than Team 0 of winning the next match.

Let **X** denote the attribute set and **Y** denote the class variable. If the class variable has a non-deterministic relationship with the attributes, then we can treat X and Y as random variables and capture their relationship probabilistically using P(Y|X). This conditional probability is also known as the **posterior probability** for Y , as opposed to its **prior probability**, P(Y).

The Bayes theorem is useful because it allows us to express the posterior probability in terms of the prior probability P(Y), the **class-conditional probability P(X|Y)**, and the evidence, P(X):

$$P(Y|\mathbf{X}) = \frac{P(\mathbf{X}|Y)\times P(Y)}{P(\mathbf{X})}.$$

When comparing the posterior probabilities for different values of Y , the denominator term, **P(X)**, is always constant, and thus, can be ignored. The Bayesian Classifiers prior probability **P(Y)** can be easily estimated from the training set by computing the fraction of training records that belong to each class.

To estimate the class-conditional probabilities P(X|Y), we present two implementations of Bayesian classification methods: the naive Bayes classifier and the Bayesian belief network.

Naive Bayes Classifier

A naive Bayes classifier estimates the class-conditional probability by assuming that the attributes are conditionally independent, given the class label y. The conditional independence assumption can be formally stated as follows:

$$P(\mathbf{X}|Y = y) = \prod_{i=1}^{d} P(X_i|Y = y),$$

where each attribute set X = {X1, X2,...,Xd} consists of d attributes.

Conditional Independence

Let X, Y, and Z denote three sets of random variables. The variables in X are said to be conditionally independent of Y, given Z, if the following condition holds:

$$P(\mathbf{X}|\mathbf{Y}, \mathbf{Z}) = P(\mathbf{X}|\mathbf{Z}).$$

The conditional independence between X and Y can also be written into a form that looks similar to conditional probability:

$$\begin{aligned} P(\mathbf{X}, \mathbf{Y}|\mathbf{Z}) &= \frac{P(\mathbf{X}, \mathbf{Y}, \mathbf{Z})}{P(\mathbf{Z})} \\ &= \frac{P(\mathbf{X}, \mathbf{Y}, \mathbf{Z})}{P(\mathbf{Y}, \mathbf{Z})} \times \frac{P(\mathbf{Y}, \mathbf{Z})}{P(\mathbf{Z})} \\ &= P(\mathbf{X}|\mathbf{Y}, \mathbf{Z}) \times P(\mathbf{Y}|\mathbf{Z}) \\ &= P(\mathbf{X}|\mathbf{Z}) \times P(\mathbf{Y}|\mathbf{Z}), \end{aligned}$$

How a Naive Bayes Classifier Works

To classify a test record, the naive Bayes classifier computes the posterior probability for each class Y :

$$P(Y|\mathbf{X}) = \frac{P(Y)\prod_{i=1}^{d} P(X_i|Y)}{P(\mathbf{X})}.$$

Since P(X) is fixed for every Y , it is sufficient to choose the class that maximizes the numerator term,

$$P(Y)\prod_{i=1}^{d} P(X_i|Y)$$

Estimating Conditional Probabilities for Categorical Attributes

For a categorical attribute Xi, the conditional probability P(Xi = xi|Y = y) is estimated according to the fraction of training instances in class y that take on a particular attribute value xi.

Estimating Conditional Probabilities for Continuous Attributes

A Gaussian distribution is usually chosen to represent the class-conditional probability for continuous attributes. The distribution is characterized by two parameters, its mean, μ, and variance, σ^2 For each class yj , the class-conditional probability for attribute Xi is

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

The parameter μij can be estimated based on the sample mean of Xi(x) for all training records that belong to the class yj . Similarly, σ^2_{ij} can be estimated from the sample variance (s^2) of such training records.

The conditional probability that Xi lies within some interval, xi and xi + , where is a small constant:

$$\begin{aligned} P(x_i \le X_i \le x_i + \epsilon|Y = y_j) &= \int_{x_i}^{x_i+\epsilon} f(X_i; \mu_{ij}, \sigma_{ij})dX_i \\ &\approx f(x_i; \mu_{ij}, \sigma_{ij}) \times \epsilon. \end{aligned}$$

Since appears as a constant multiplicative factor for each class, it cancels out when we normalize the posterior probability for P(Y|X).

To predict the class label of a test record X = (Home Owner=No, Marital Status = Married, Income = $120K), we need to compute the posterior probabilities P(No|X) and P(Yes|X).

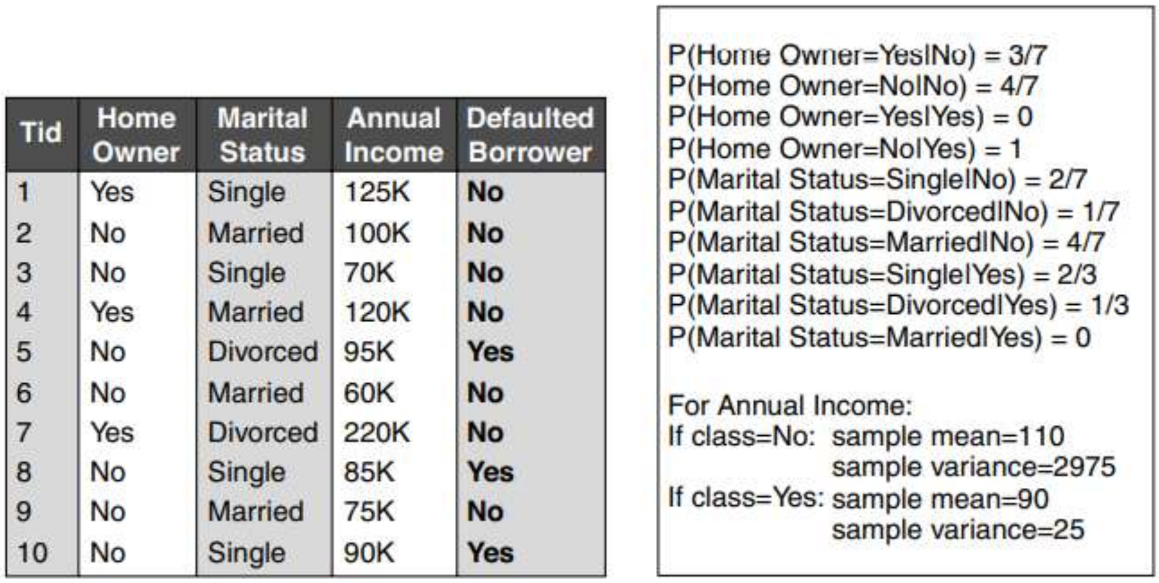
$$\overline{x} = \frac{125 + 100 + 70 + \ldots + 75}{7} = 110$$

$$s^2 = \frac{(125-110)^2 + (100-110)^2 + \ldots + (75-110)^2}{7(6)} = 2975$$

$$s = \sqrt{2975} = 54.54.$$

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

The prior probabilities of each class can be estimated by calculating the fraction of training records that belong to each class. Since there are three records that belong to the class Yes and seven records that belong to the class No, P(Yes)=0.3 and P(No)=0.7. Using the information provided in Figure 5.10(b), the class-conditional probabilities can be computed as follows:



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

P(Home Owner=Yes|No) = 3/7
P(Home Owner=No|No) = 4/7
P(Home Owner=Yes|Yes) = 0
P(Home Owner=No|Yes) = 1
P(Marital Status=Single|No) = 2/7
P(Marital Status=Divorced|No) = 1/7
P(Marital Status=Married|No) = 4/7
P(Marital Status=Single|Yes) = 2/3
P(Marital Status=Divorced|Yes) = 1/3
P(Marital Status=Married|Yes) = 0

For Annual Income:
If class=No: sample mean=110
sample variance=2975
If class=Yes: sample mean=90
sample variance=25

(a)

(b)

Figure 5.10. The naïve Bayes classifier for the loan classification problem.

$$
\begin{aligned}
P(\mathbf{X}|\texttt{No}) &= P(\texttt{Home Owner} = \texttt{No}|\texttt{No}) \times P(\texttt{Status} = \texttt{Married}|\texttt{No}) \\
&\quad \times P(\texttt{Annual Income} = \$120\text{K}|\texttt{No}) \\
&= 4/7 \times 4/7 \times 0.0072 = 0.0024.
\end{aligned}
$$

$$
\begin{aligned}
P(\mathbf{X}|\texttt{Yes}) &= P(\texttt{Home Owner} = \texttt{No}|\texttt{Yes}) \times P(\texttt{Status} = \texttt{Married}|\texttt{Yes}) \\
&\quad \times P(\texttt{Annual Income} = \$120\text{K}|\texttt{Yes}) \\
&= 1 \times 0 \times 1.2 \times 10^{-9} = 0.
\end{aligned}
$$

Putting them together, the posterior probability for class No is P(No|X) = α × 7/10 × 0.0024 = 0.0016α, where α = 1/P(X) is a constant term.

Alternative Metrics

A **confusion matrix** that summarizes the number of instances predicted correctly or incorrectly by a classification model is shown in Table 5.6.

The following terminology is often used when referring to the counts tabulated in a confusion matrix:

- True positive (TP) or f++, which corresponds to the number of positive examples correctly predicted by the classification model.

- False negative (FN) or f+−, which corresponds to the number of positive examples wrongly predicted as negative by the classification model.

- False positive (FP) or f−+, which corresponds to the number of negative examples wrongly predicted as positive by the classification model.

- True negative (TN) or f−−, which corresponds to the number of negative examples correctly predicted by the classification model.

The counts in a confusion matrix can also be expressed in terms of percentages. The **true positive rate** (TPR) or sensitivity is defined as the fraction of positive examples predicted correctly by the model, i.e.,

$$TPR = TP/(T P + F N)$$

Similarly, the **true negative rate** (TNR) or specificity is defined as the fraction of negative examples predicted correctly by the model, i.e.,

$$TNR = TN/(T N + F P)$$

Finally, the **false positive rate** (FPR) is the fraction of negative examples predicted as a positive class, i.e.,

FPR = FP/(T N + F P)

while the **false negative rate** (FNR) is the fraction of positive examples predicted as a negative class, i.e.,

FNR = FN/(T P + F N)

Recall and precision are two widely used metrics employed in applications where successful detection of one of the classes is considered more significant than detection of the other classes. A formal definition of these metrics is given below.

$$\text{Precision, } p = \frac{TP}{TP + FP}$$
$$\text{Recall, } r = \frac{TP}{TP + FN}$$

Precision determines the fraction of records that actually turns out to be positive in the group the classifier has declared as a positive class. The higher the precision is, the lower the number of false positive errors committed by the classifier.

Recall measures the fraction of positive examples correctly predicted by the classifier. Classifiers with large recall have very few positive examples misclassified as the negative class.

F1 measure represents a harmonic mean between recall and precision, i.e.,

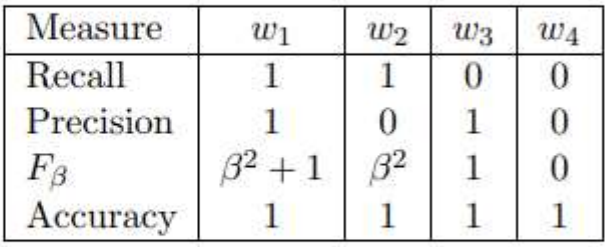
$$F_1 = \frac{2}{\frac{1}{r} + \frac{1}{p}}.$$

The harmonic mean of two numbers x and y tends to be closer to the smaller of the two numbers. Hence, a high value of F1-measure ensures that both precision and recall are reasonably high.

Weighted accuracy measure, which is defined by the following equation:

$$\text{Weighted accuracy} = \frac{w_1TP + w_4TN}{w_1TP + w_2FP + w_3FN + w_4TN}.$$

The relationship between weighted accuracy and other performance metrics is summarized in the following table:



Measure	w_1	w_2	w_3	w_4
Recall	1	1	0	0
Precision	1	0	1	0
F_β	$\beta^2 + 1$	β^2	1	0
Accuracy	1	1	1	1

CHAPTER – 6 : Association Analysis: Basic Concepts and Algorithms

A methodology known as **association analysis**, which is useful for discovering interesting relationships hidden in large data sets. The uncovered relationships can be represented in the form of **association rules** or sets of frequent items.

Itemset and Support Count

Let I = {i1,i2,...,id} be the set of all items in a market basket data and T = {t1, t2,...,tN } be the set of all transactions. Each transaction ti contains a subset of items chosen from I.

In association analysis, a collection of zero or more items is termed an itemset. If an itemset contains k items, it is called a k-itemset.

The null (or empty) set is an itemset that does not contain any items.

The transaction width is defined as the number of items present in a transaction. A transaction tj is said to contain an itemset X if X is a subset of tj .

An important property of an itemset is its support count, which refers to the number of transactions that contain a particular itemset. Mathematically, the support count, σ(X), for an itemset X can be stated as follows:

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

where the symbol |·| denote the number of elements in a set.

Association Rule An association rule is an implication expression of the form X −→ Y , where X and Y are disjoint itemsets, i.e., X ∩ Y = ∅. The strength of an association rule can be measured in terms of its **support** and **confidence**.

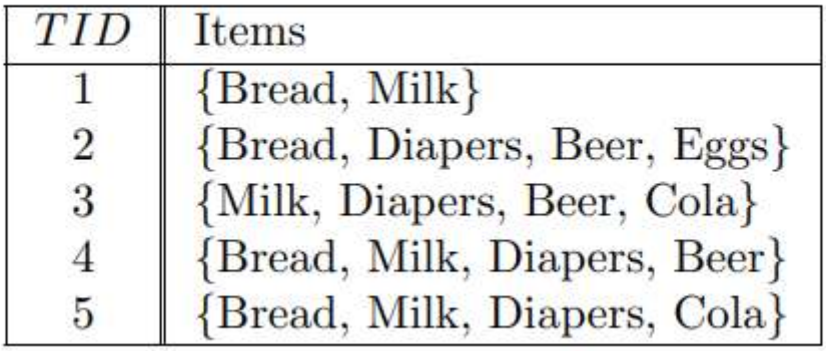
Support determines how often a rule is applicable to a given data set, while **confidence** determines how frequently items in Y appear in transactions that contain X. The formal definitions of these metrics are:

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

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

Example :-

Table 6.1. An example of market basket transactions.



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

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

Why Use Support and Confidence?

Support is an important measure because a rule that has very low support may occur simply by chance. A low support rule is also likely to be uninteresting from a business perspective because it may not be profitable to promote items that customers seldom buy together.

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

Formulation of Association Rule Mining Problem

The association rule mining problem can be formally stated as follows:

Definition 6.1 (Association Rule Discovery)

Given a set of transactions T, find all the rules having **support ≥ minsup** and **confidence ≥ minconf**, where minsup and minconf are the corresponding support and confidence thresholds.

More specifically, the total number of possible rules extracted from a data set that contains d items is:

$$R = 3^d - 2^{d+1} + 1.$$

This approach requires us to compute the support and confidence for $3^6 - 2^7 + 1 = 602$ rules. More than 80% of the rules are discarded after applying minsup = 20% and minconf = 50%, thus making most of the computations become wasted.

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

1. **Frequent Itemset Generation**, whose objective is to find all the itemsets that satisfy the minsup threshold. These itemsets are called frequent itemsets.

2. **Rule Generation**, whose objective is to extract all the high-confidence rules from the frequent itemsets found in the previous step. These rules are called strong rules.

Frequent Itemset Generation

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

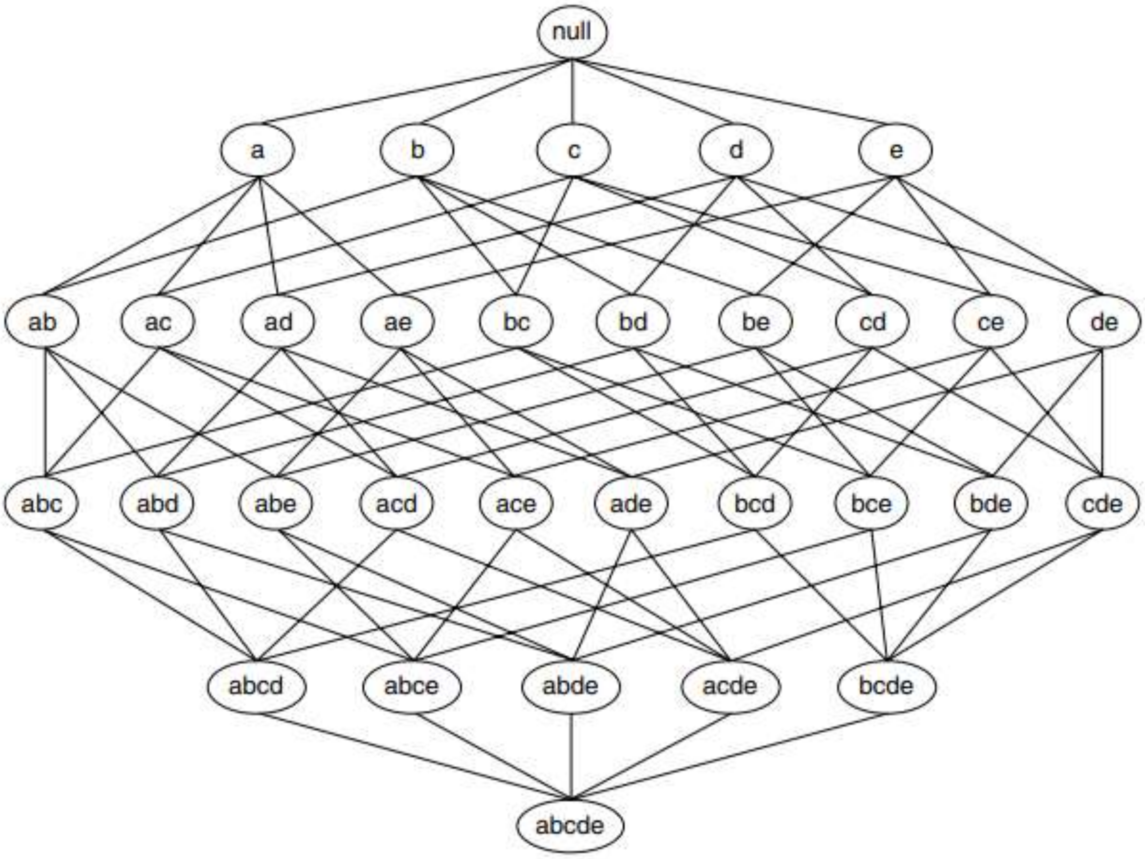
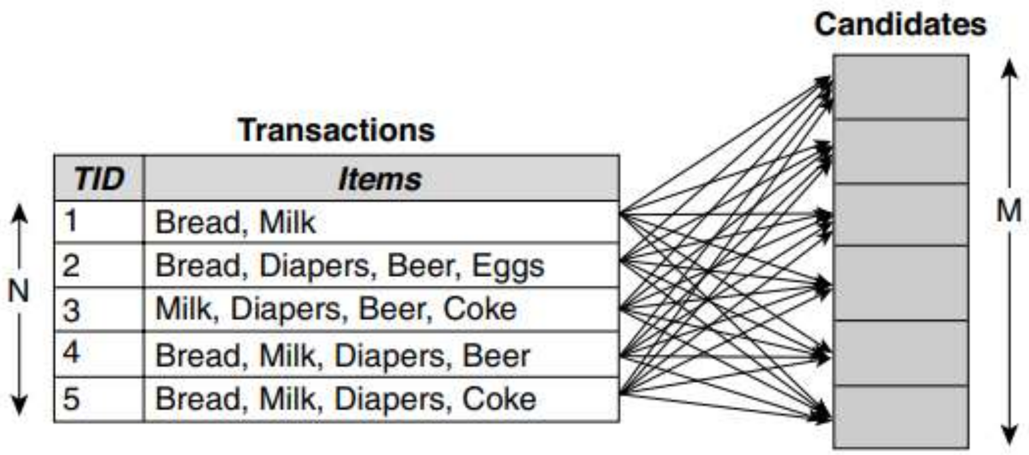


Figure 6.1. An itemset lattice.

A brute-force approach for finding frequent itemsets is to determine the support count for every candidate itemset in the lattice structure. To do this, we need to compare each candidate against every transaction, an operation that is shown in Figure 6.2. If the candidate is contained in a transaction, its support count will be incremented.

For example, the support for {Bread, Milk} is incremented three times because the itemset is contained in transactions 1, 4, and 5.

Such an approach can be very expensive because it requires **O(NMw)** comparisons, where **N** is the number of transactions, **M = 2^K – 1** is the number of candidate itemsets, and w is the maximum transaction width.



TID	Items
1	Bread, Milk
2	Bread, Diapers, Beer, Eggs
3	Milk, Diapers, Beer, Coke
4	Bread, Milk, Diapers, Beer
5	Bread, Milk, Diapers, Coke

Figure 6.2. Counting the support of candidate itemsets.

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

1. **Reduce the number of candidate itemsets (M)**

   The Apriori principle, described in the next section, is an effective way to eliminate some of the candidate itemsets without counting their support values.

2. **Reduce the number of comparisons**

   Instead of matching each candidate itemset against every transaction, we can reduce the number of comparisons by using more advanced data structures, either to store the candidate itemsets or to compress the data set.

The Apriori Principle

If an itemset is frequent, then all of its subsets must also be frequent.

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

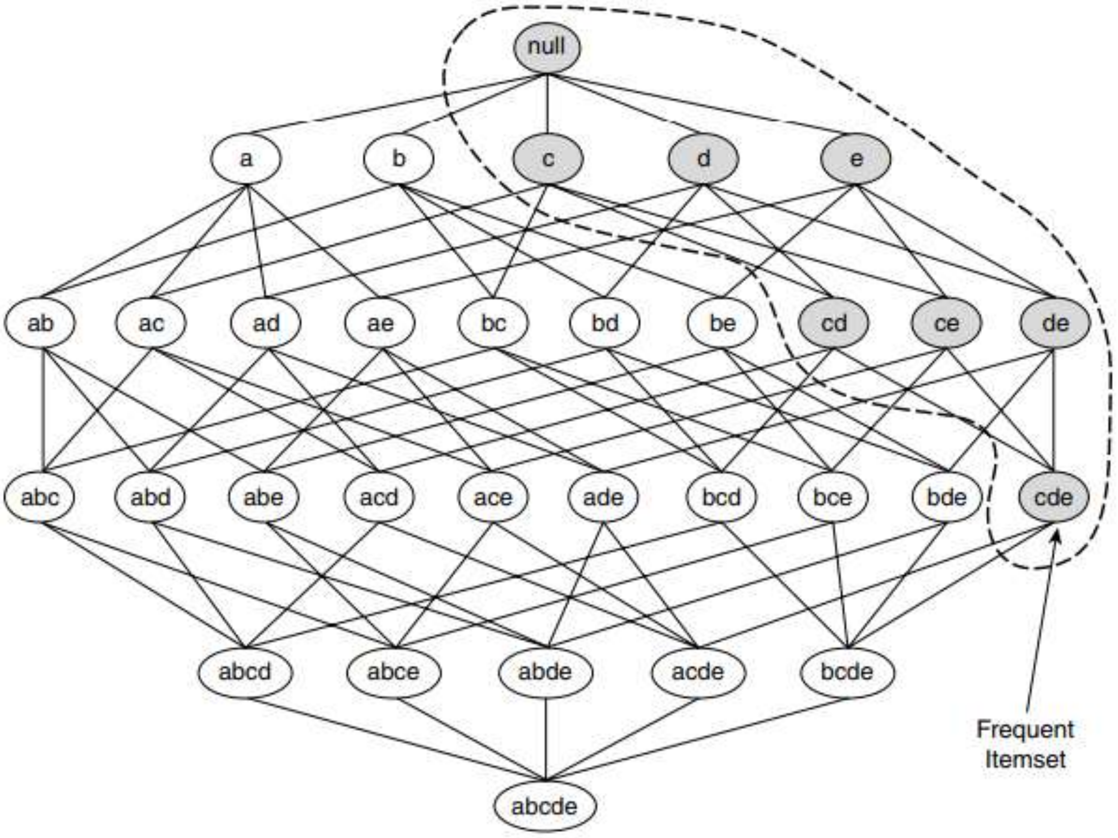


Figure 6.3. An illustration of the *Apriori* principle. If $\{c, d, e\}$ is frequent, then all subsets of this itemset are frequent.

Conversely, if an itemset such as {a, b} is infrequent, then all of its supersets must be infrequent too. As illustrated in Figure 6.4, the entire subgraph containing the supersets of {a, b} can be pruned immediately once {a, b} is found to be infrequent.

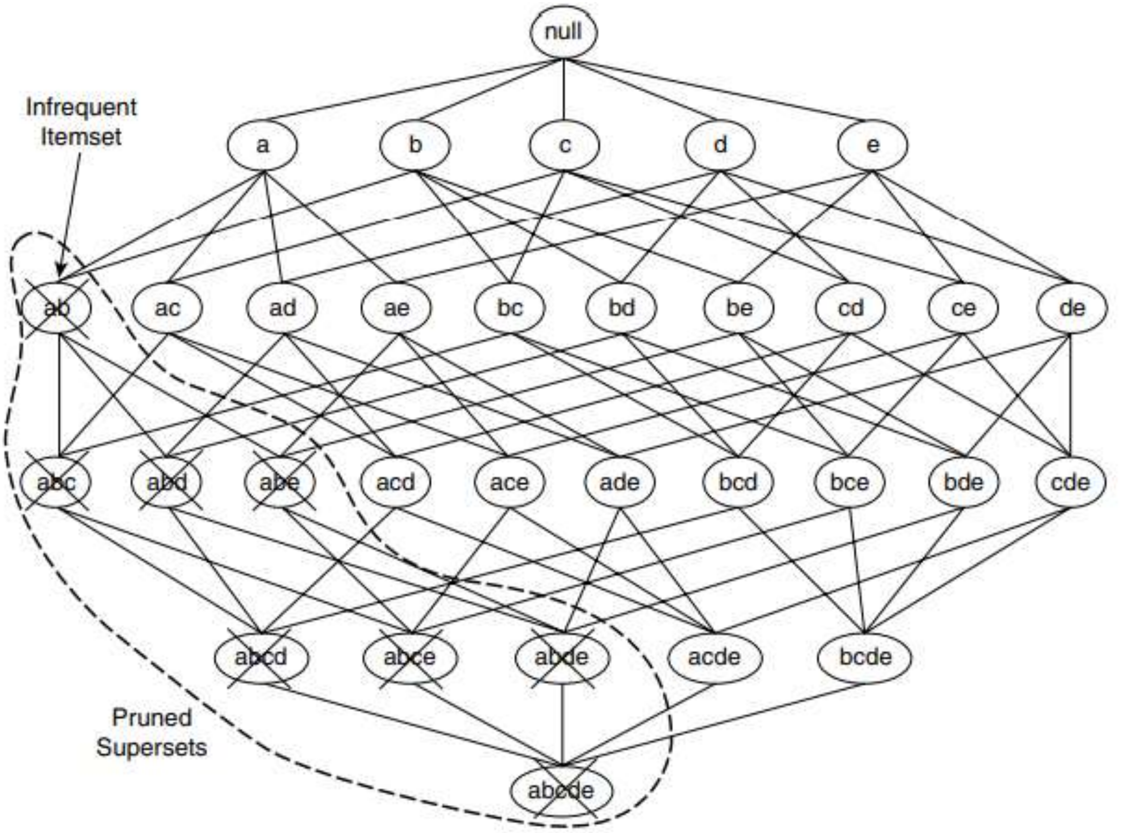


Figure 6.4. An illustration of support-based pruning. If $\{a, b\}$ is infrequent, then all supersets of $\{a, b\}$ are infrequent.

This strategy of trimming the exponential search space based on the support measure is known as **support-based pruning**. Such a pruning strategy is made possible by a key property of the support measure, namely, that the support for an itemset never exceeds the support for its subsets. This property is also known as the **anti-monotone property** of the support measure.

Definition 6.2 (Monotonicity Property)

Let I be a set of items, and J = 2I be the power set of I. A measure f is monotone (or upward closed) if which means that if X is a subset of Y , then f(X) must not exceed f(Y).

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

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

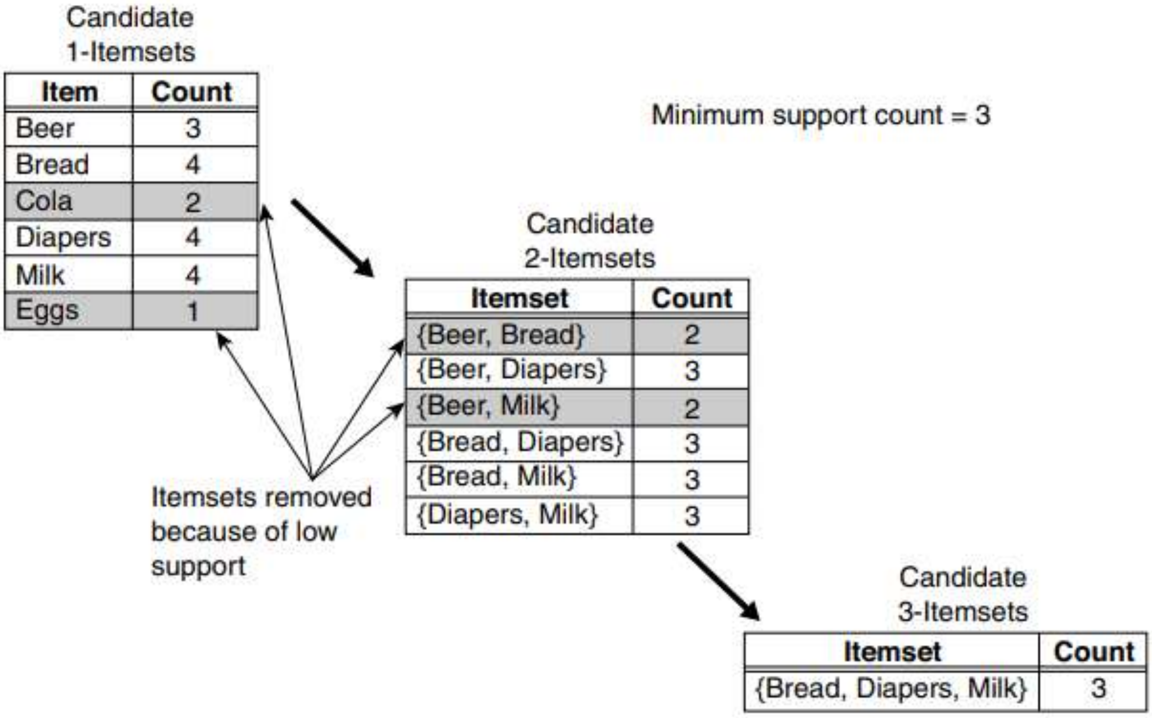
$$\forall X, Y \in J : \ (X \subseteq Y) \longrightarrow f(Y) \leq f(X),$$

Frequent Itemset Generation in the Apriori Algorithm

Apriori is the first association rule mining algorithm that pioneered the use of support-based pruning to systematically control the exponential growth of candidate itemsets.

Figure 6.5 provides a high-level illustration of the frequent itemset generation part of the Apriori algorithm for the transactions shown in Table 6.1.

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



Item	Count
Beer	3
Bread	4
Cola	2
Diapers	4
Milk	4
Eggs	1

Itemset	Count
{Beer, Bread}	2
{Beer, Diapers}	3
{Beer, Milk}	2
{Bread, Diapers}	3
{Bread, Milk}	3
{Diapers, Milk}	3

Itemset	Count
{Bread, Diapers, Milk}	3

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

The effectiveness of the Apriori pruning strategy can be shown by counting the number of candidate itemsets generated. A **brute-force** strategy of enumerating all itemsets (up to size 3) as candidates will produce

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

candidates. With the Apriori principle, this number decreases to

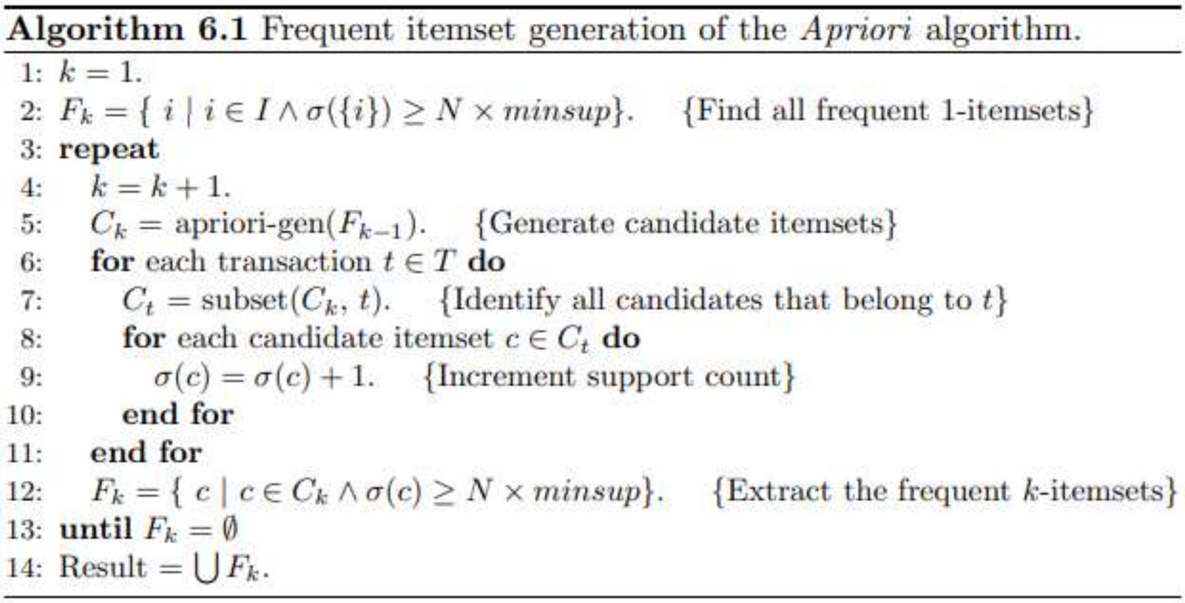
$$\binom{6}{1} + \binom{4}{2} + 1 = 6 + 6 + 1 = 13$$

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

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

- The algorithm initially makes a single pass over the data set to determine the support of each item. Upon completion of this step, the set of all frequent 1-itemsets, F_1, will be known (steps 1 and 2).

- Next, the algorithm will iteratively generate new candidate k-itemsets using the frequent (k − 1)-itemsets found in the previous iteration (step 5). Candidate generation is implemented using a function called apriorigen, which is described in Section 6.2.3.



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

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

- To count the support of the candidates, the algorithm needs to make an additional pass over the data set (steps 6–10). The subset function is used to determine all the candidate itemsets in Ck that are contained in each transaction t. The implementation of this function is described in Section 6.2.4.

- After counting their supports, the algorithm eliminates all candidate itemsets whose support counts are less than minsup (step 12).

- The algorithm terminates when there are no new frequent itemsets generated, i.e., Fk = Ø (step 13).

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

Candidate Generation and Pruning

1. **Candidate Generation :** This operation generates new candidate kitemsets based on the frequent (k − 1)-itemsets found in the previous iteration.

2. **Candidate Pruning :** This operation eliminates some of the candidate k-itemsets using the support-based pruning strategy.

To illustrate the candidate pruning operation, consider a candidate k-itemset, X = {i1, i2,...,ik}. The algorithm must determine whether all of its proper subsets, X − {ij} (∀j = 1, 2,...,k), are frequent. If one of them is infrequent, then X is immediately pruned. This approach can effectively reduce the number of candidate itemsets considered during support counting.

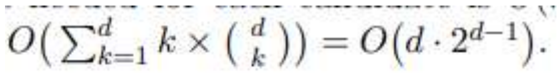
In principle, there are many ways to generate candidate itemsets. The following is a list of requirements for an effective candidate generation procedure:

1. It should avoid generating too many unnecessary candidates. A candidate itemset is unnecessary if at least one of its subsets is infrequent. Such a candidate is guaranteed to be infrequent according to the antimonotone property of support.

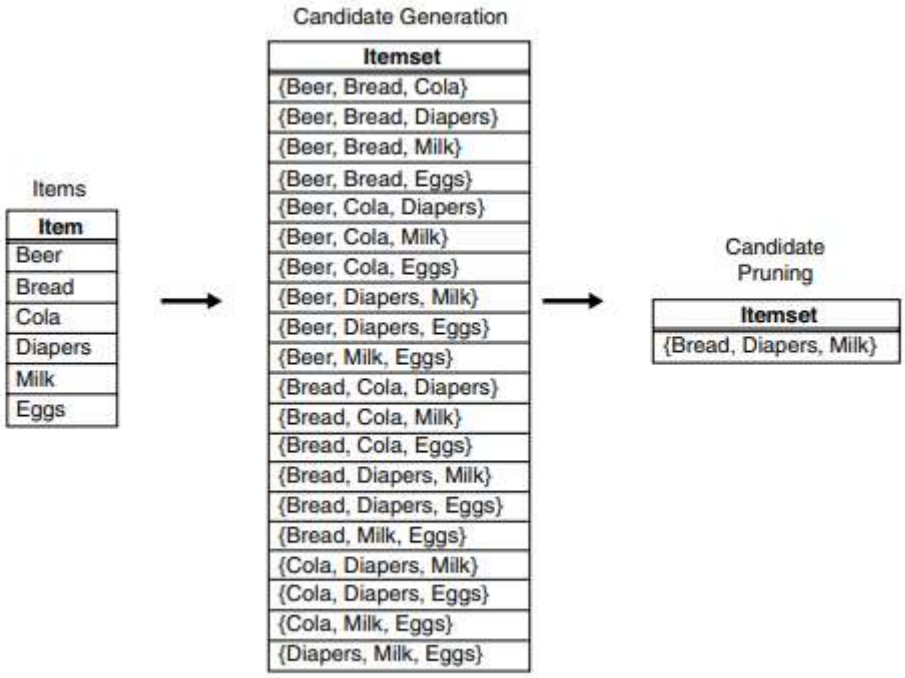
2. It must ensure that the candidate set is complete, i.e., no frequent itemsets are left out by the candidate generation procedure. To ensure completeness, the set of candidate itemsets must subsume the set of all frequent itemsets, i.e., ∀k : Fk ⊆ Ck.

3. It should not generate the same candidate itemset more than once. For example, the candidate itemset {a, b, c, d} can be generated in many ways—by merging {a, b, c} with {d}, {b, d} with {a, c}, {c} with {a, b, d}, etc. Generation of duplicate candidates leads to wasted computations and thus should be avoided for efficiency reasons.

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



$$O\left(\sum_{k=1}^{d} k \times \binom{d}{k}\right) = O(d \cdot 2^{d-1}).$$



Items

Item
Beer
Bread
Cola
Diapers
Milk
Eggs

→

Candidate Generation

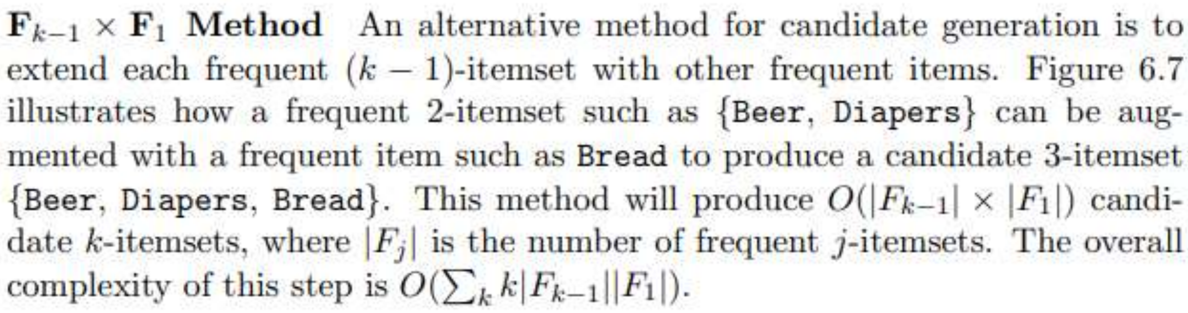
Itemset
{Beer, Bread, Cola}
{Beer, Bread, Diapers}
{Beer, Bread, Milk}
{Beer, Bread, Eggs}
{Beer, Cola, Diapers}
{Beer, Cola, Milk}
{Beer, Cola, Eggs}
{Beer, Diapers, Milk}
{Beer, Diapers, Eggs}
{Beer, Milk, Eggs}
{Bread, Cola, Diapers}
{Bread, Cola, Milk}
{Bread, Cola, Eggs}
{Bread, Diapers, Milk}
{Bread, Diapers, Eggs}
{Bread, Milk, Eggs}
{Cola, Diapers, Milk}
{Cola, Diapers, Eggs}
{Cola, Milk, Eggs}
{Diapers, Milk, Eggs}

→

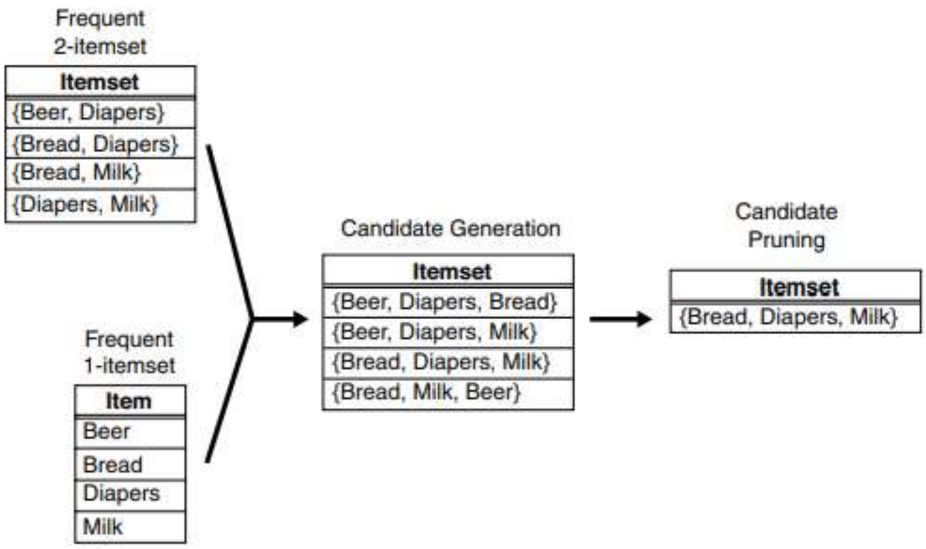
Candidate Pruning

Itemset
{Bread, Diapers, Milk}

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



$\mathbf{F}_{k-1} \times \mathbf{F}_1$ Method An alternative method for candidate generation is to extend each frequent $(k-1)$-itemset with other frequent items. Figure 6.7 illustrates how a frequent 2-itemset such as {Beer, Diapers} can be augmented with a frequent item such as Bread to produce a candidate 3-itemset {Beer, Diapers, Bread}. This method will produce $O(|F_{k-1}| \times |F_1|)$ candidate k-itemsets, where $|F_j|$ is the number of frequent j-itemsets. The overall complexity of this step is $O(\sum_k k|F_{k-1}||F_1|)$.



Itemset
{Beer, Diapers}
{Bread, Diapers}
{Bread, Milk}
{Diapers, Milk}

Item
Beer
Bread
Diapers
Milk

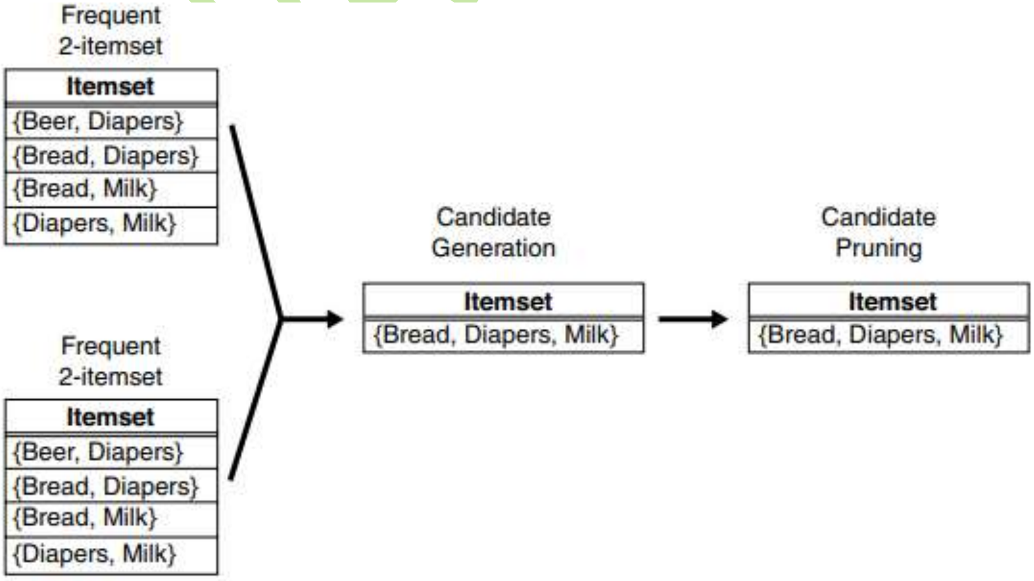
Itemset
{Beer, Diapers, Bread}
{Beer, Diapers, Milk}
{Bread, Diapers, Milk}
{Bread, Milk, Beer}

Itemset
{Bread, Diapers, Milk}

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

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

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



Itemset
{Beer, Diapers}
{Bread, Diapers}
{Bread, Milk}
{Diapers, Milk}

Itemset
{Beer, Diapers}
{Bread, Diapers}
{Bread, Milk}
{Diapers, Milk}

Itemset
{Bread, Diapers, Milk}

Itemset
{Bread, Diapers, Milk}

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

Support Counting

Support counting is the process of determining the frequency of occurrence for every candidate itemset that survives the candidate pruning step of the apriori-gen function.

An alternative approach is to enumerate the itemsets contained in each transaction and use them to update the support counts of their respective candidate itemsets.

To illustrate, consider a transaction t that contains five items, {1, 2, 3, 5, 6}. There are $^{5}C_{3}$ = 10 itemsets of size 3 contained in this transaction. Some of the itemsets may correspond to the candidate 3-itemsets under investigation, in which case, their support counts are incremented. Other subsets of t that do not correspond to any candidates can be ignored.

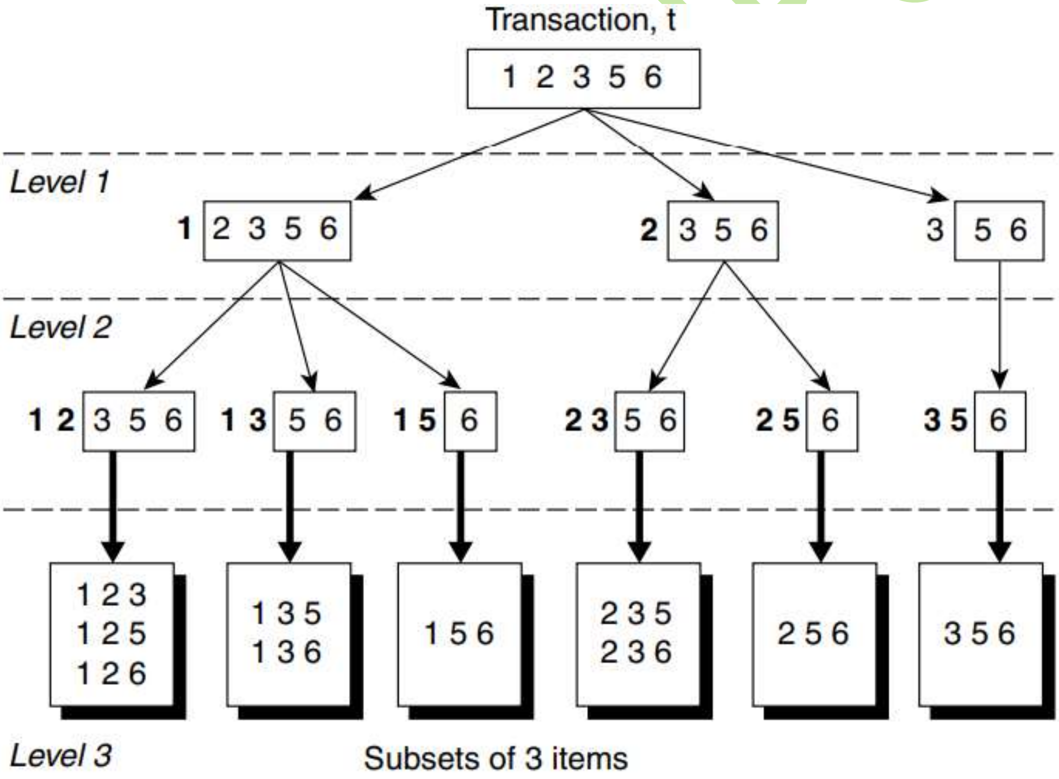
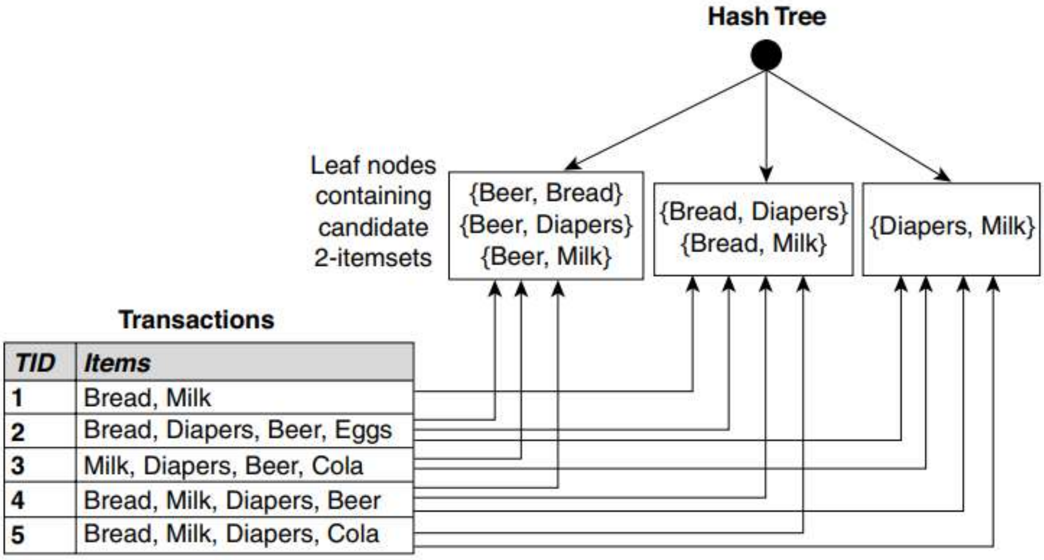


Figure 6.9. Enumerating subsets of three items from a transaction *t*.

Support Counting Using a Hash Tree

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



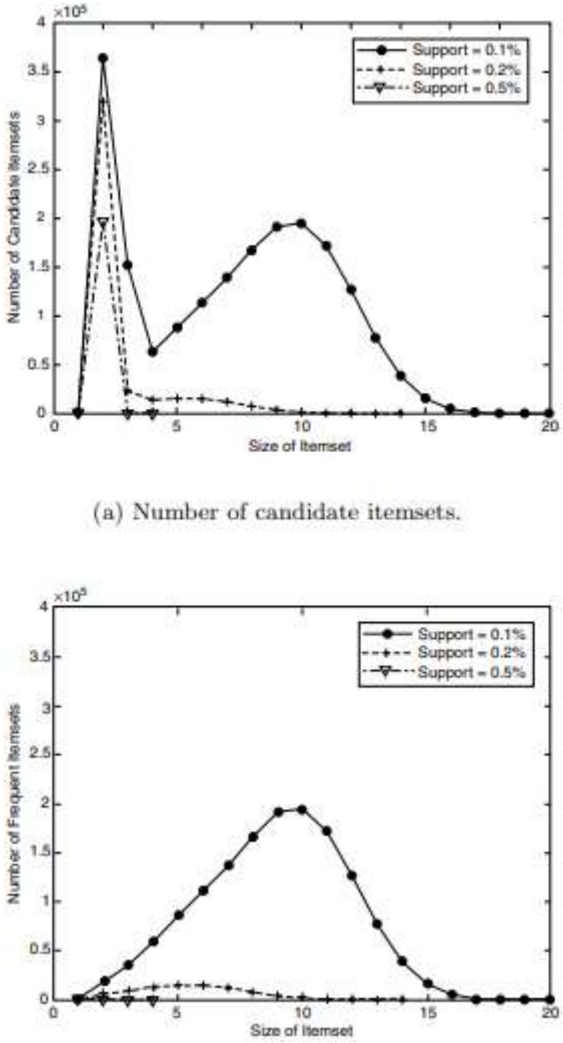
TID	Items
1	Bread, Milk
2	Bread, Diapers, Beer, Eggs
3	Milk, Diapers, Beer, Cola
4	Bread, Milk, Diapers, Beer
5	Bread, Milk, Diapers, Cola

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

Computational Complexity

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

Support Threshold Lowering the support threshold often results in more itemsets being declared as frequent. This has an adverse effect on the computational complexity of the algorithm because more candidate itemsets must be generated and counted, as shown in Figure 6.13. The maximum size of frequent itemsets also tends to increase with lower support thresholds. As the maximum size of the frequent itemsets increases, the algorithm will need to make more passes over the data set.



(a) Number of candidate itemsets.

(b) Number of frequent itemsets.

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

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

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

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

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

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

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

A hash tree is also constructed during candidate generation to store the candidate itemsets. Because the maximum depth of the tree is k, the cost for populating the hash tree with candidate itemsets is

$$O\left(\sum_{k=2}^{w} k|C_k|\right).$$

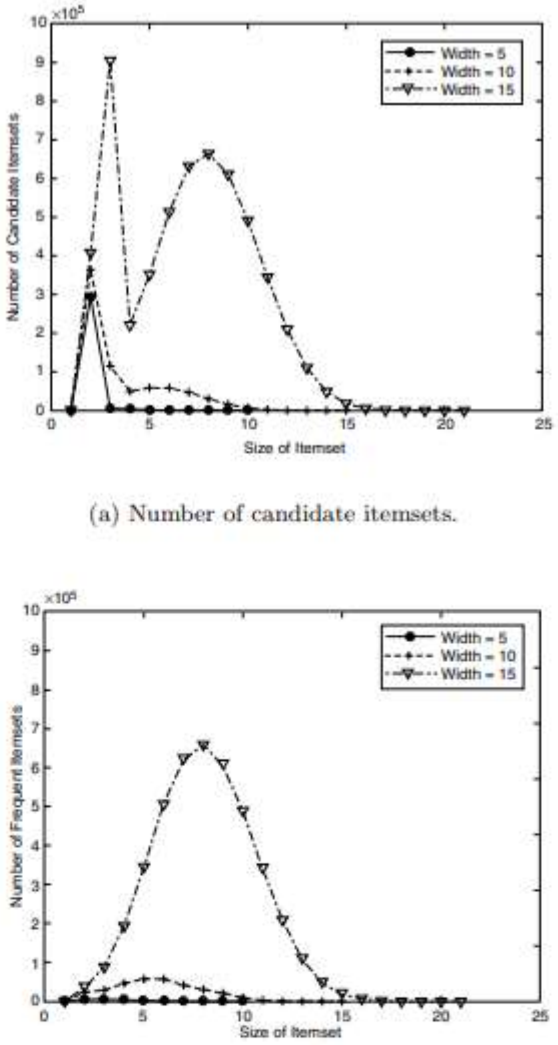
During candidate pruning, we need to verify that the k − 2 subsets of every candidate k-itemset are frequent. Since the cost for looking up a candidate in a hash tree is O(k), the candidate pruning step requires

$$O\left(\sum_{k=2}^{w} k(k-2)|C_k|\right)$$

Support counting Each transaction of length |t| produces $\binom{|t|}{k}$ itemsets of size k. This is also the effective number of hash tree traversals performed for each transaction. The cost for support counting is

$$O\left(N\sum_{k}\binom{w}{k}\alpha_k\right).$$

where w is the maximum transaction width and α_k is the cost for updating the support count of a candidate k-itemset in the hash tree.



(a) Number of candidate itemsets.

(b) Number of Frequent Itemsets.

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

Rule Generation

Each frequent k-itemset, Y , can produce up to 2^k–2 association rules, ignoring rules that have empty antecedents or consequents (Ø –-> Y or Y –-> Ø). An association rule can be extracted by partitioning the itemset Y into two non-empty subsets, X and Y −X, such that X –-> Y −X satisfies the confidence threshold. Note that all such rules must have already met the support threshold because they are generated from a frequent itemset.

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

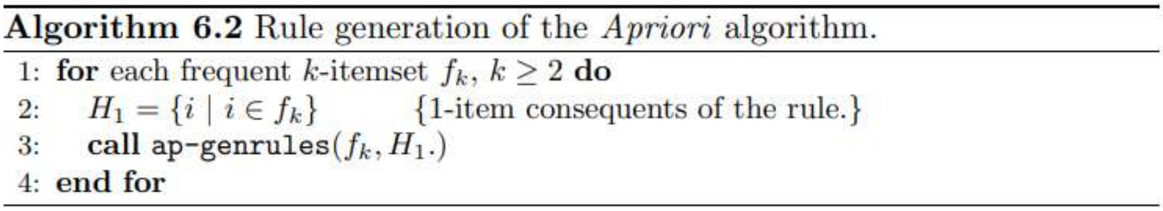
Confidence-Based Pruning

Unlike the support measure, confidence does not have any monotone property. For example, the confidence for X –→ Y can be larger, smaller, or equal to the confidence for another rule X˜ –→ Y˜ , where X˜ ⊆ X and Y˜ ⊆ Y.

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

Rule Generation in Apriori Algorithm

The Apriori algorithm uses a level-wise approach for generating association rules, where each level corresponds to the number of items that belong to the rule consequent. Initially, all the high-confidence rules that have only one item in the rule consequent are extracted. These rules are then used to generate new candidate rules. For example, if {acd} –→ {b} and {abd} –→ {c} are high-confidence rules, then the candidate rule {ad} –→ {bc} is generated by merging the consequents of both rules. Figure 6.15 shows a lattice structure for the association rules generated from the frequent itemset {a, b, c, d}. If any node in the lattice has low confidence, then according to Theorem 6.2, the entire subgraph spanned by the node can be pruned immediately. Suppose the confidence for {bcd} –→ {a} is low. All the rules containing item a in its consequent, including {cd} –→ {ab}, {bd} –→ {ac}, {bc} –→ {ad}, and {d} –→ {abc} can be discarded.



Algorithm 6.2 Rule generation of the *Apriori* algorithm.

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

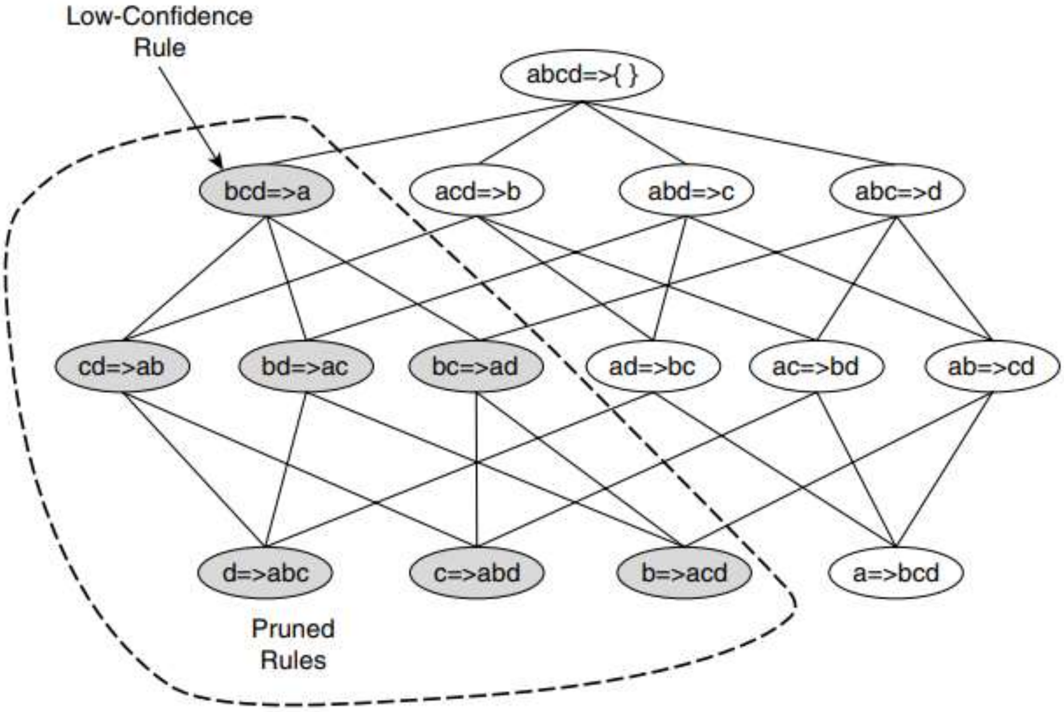


Figure 6.15. Pruning of association rules using the confidence measure.

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

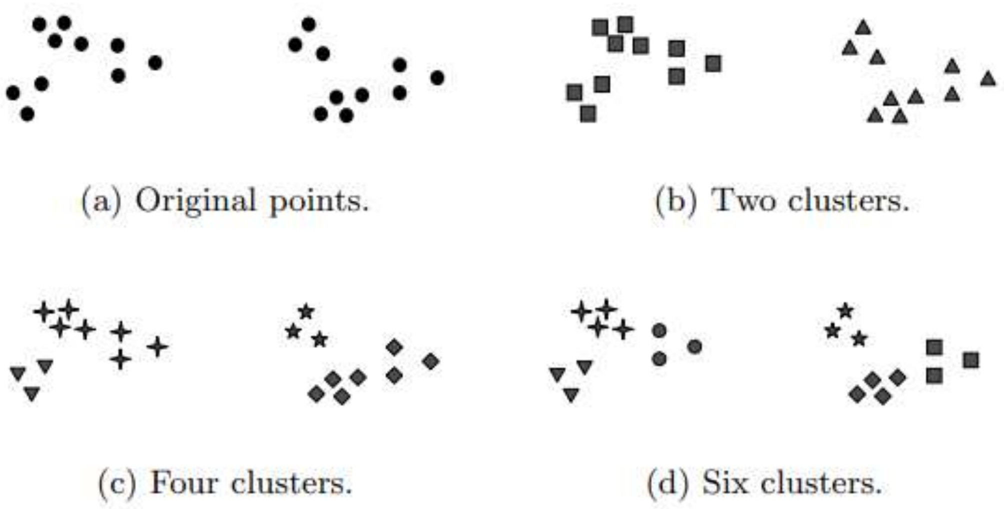
```
$k = |f_k|$   {size of frequent itemset.}
$m = |H_m|$   {size of rule consequent.}
if $k > m + 1$ then
  $H_{m+1} = $ apriori-gen($H_m$).
  for each $h_{m+1} \in H_{m+1}$ do
    $conf = \sigma(f_k)/\sigma(f_k - h_{m+1})$.
    if $conf \geq minconf$ then
      output the rule $(f_k - h_{m+1}) \longrightarrow h_{m+1}$.
    else
      delete $h_{m+1}$ from $H_{m+1}$.
    end if
  end for
  call ap-genrules($f_k, H_{m+1}$.)
end if
```

CHAPTER – 8 : Cluster Analysis: Basic Concepts and Algorithms

What Is Cluster Analysis?

Cluster analysis groups data objects based only on information found in the data that describes the objects and their relationships. The goal is that the objects within a group be similar (or related) to one another and different from (or unrelated to) the objects in other groups.

The greater the similarity (or homogeneity) within a group and the greater the difference between groups, the better or more distinct the clustering.



(a) Original points.

(b) Two clusters.

(c) Four clusters.

(d) Six clusters.

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

Cluster analysis is related to other techniques that are used to divide data objects into groups. For instance, clustering can be regarded as a form of classification in that it creates a labeling of objects with class (cluster) labels. However, it derives these labels only from the data. In contrast, classification in the sense of Chapter 4 is **supervised classification**; i.e., new, unlabeled objects are assigned a class label using a model developed from objects with known class labels. For this reason, cluster analysis is sometimes referred to as **unsupervised classification**. When the term classification is used without any qualification within data mining, it typically refers to **supervised classification**.

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

Different Types of Clusterings

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

Hierarchical versus Partitional The most commonly discussed distinction among different types of clusterings is whether the set of clusters is nested or unnested, or in more traditional terminology, hierarchical or partitional. A **partitional clustering** is simply a division of the set of data objects into non-overlapping subsets (clusters) such that each data object is in exactly one subset.

If we permit clusters to have subclusters, then we obtain a **hierarchical clustering**, which is a set of nested clusters that are organized as a tree. Each node (cluster) in the tree (except for the leaf nodes) is the union of its children (subclusters), and the root of the tree is the cluster containing all the objects.

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

In a **fuzzy clustering**, every object belongs to every cluster with a membership weight that is between 0 (absolutely doesn't belong) and 1 (absolutely belongs). In other words, clusters are treated as fuzzy sets. (Mathematically, a fuzzy set is one in which an object belongs to any set with a weight that is between 0 and 1.

Complete versus Partial A **complete clustering** assigns every object to a cluster, whereas a **partial clustering** does not. The motivation for a partial clustering is that some objects in a data set may not belong to well-defined groups. Many times objects in the data set may represent noise, outliers, or "uninteresting background."

Different Types of Clusters

Well-Separated A cluster is a set of objects in which each object is closer (or more similar) to every other object in the cluster than to any object not in the cluster. Sometimes a threshold is used to specify that all the objects in a cluster must be sufficiently close (or similar) to one another. This idealistic definition of a cluster is satisfied only when the data contains natural clusters that are quite far from each other. Well-separated clusters do not need to be globular, but can have any shape.

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

Graph-Based If the data is represented as a graph, where the nodes are objects and the links represent connections among objects, then a cluster can be defined as a **connected component**; i.e., a group of objects that are connected to one another, but that have no connection to objects outside the group. An important example of graph-based clusters are contiguity-based clusters, where two objects are connected only if they are within a specified distance of each other. This implies that each object in a **contiguity-based**

cluster is closer to some other object in the cluster than to any point in a different cluster.

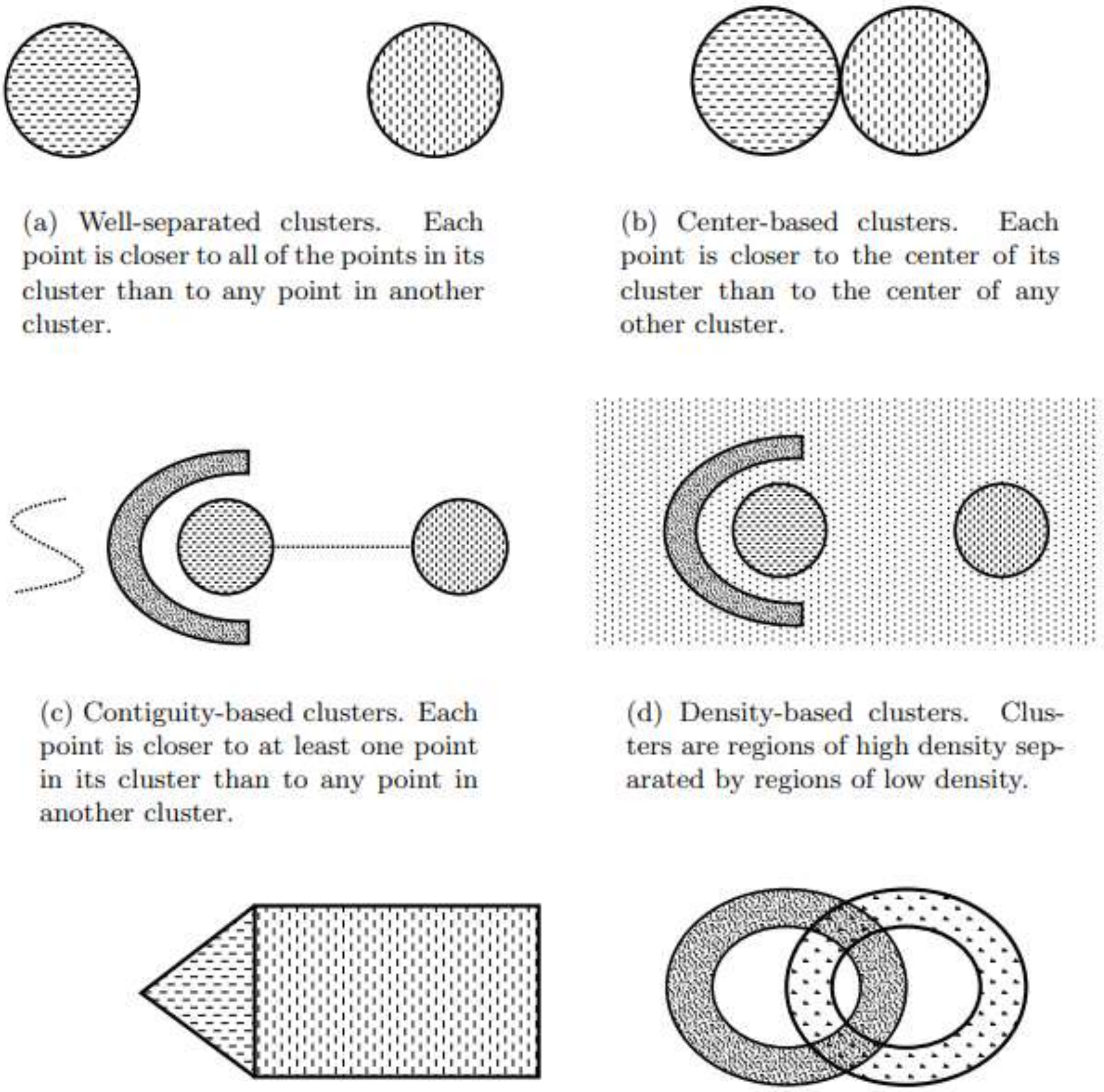
Other types of graph-based clusters are also possible. One such approach defines a cluster as a **clique**; i.e., a set of nodes in a graph that are completely connected to each other. Specifically, if we add connections between objects in the order of their distance from one another, a cluster is formed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

Density-Based A cluster is a dense region of objects that is surrounded by a region of low density. The two circular clusters are not merged, because the bridge between them fades into the noise. Likewise, the curve that is present in fades into the noise and does not form a cluster in Figure 8.2(d). A density-based definition of a cluster is often employed when the clusters are irregular or intertwined, and when noise and outliers are present. By contrast, a contiguity based definition of a cluster would not work well for the data of Figure 8.2(d) since the noise would tend to form bridges between clusters.

Shared-Property (Conceptual Clusters) More generally, we can define a cluster as a set of objects that share some property. This definition encompasses all the previous definitions of a cluster; e.g., objects in a center-based cluster share the property that they are all closest to the same centroid or medoid. However, the shared-property approach also includes new types of clusters.

Road Map

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



(a) Well-separated clusters. Each point is closer to all of the points in its cluster than to any point in another cluster.

(b) Center-based clusters. Each point is closer to the center of its cluster than to the center of any other cluster.

(c) Contiguity-based clusters. Each point is closer to at least one point in its cluster than to any point in another cluster.

(d) Density-based clusters. Clusters are regions of high density separated by regions of low density.

(e) Conceptual clusters. Points in a cluster share some general property that derives from the entire set of points. (Points in the intersection of the circles belong to both.)

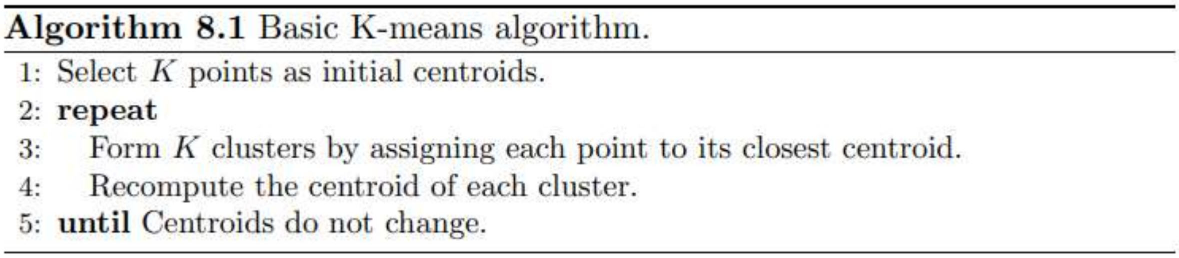
Figure 8.2. Different types of clusters as illustrated by sets of two-dimensional points.

K-means

Prototype-based clustering techniques create a one-level partitioning of the data objects. There are a number of such techniques, but two of the most prominent are K-means and K-medoid. K-means defines a prototype in terms of a centroid, which is usually the mean of a group of points, and is typically applied to objects in a continuous n-dimensional space. K-medoid defines a prototype in terms of a medoid, which is the most representative point for a group of points, and can be applied to a wide range of data since it requires only a proximity measure for a pair of objects. While a centroid almost never corresponds to an actual data point, a medoid, by its definition, must be an actual data point.

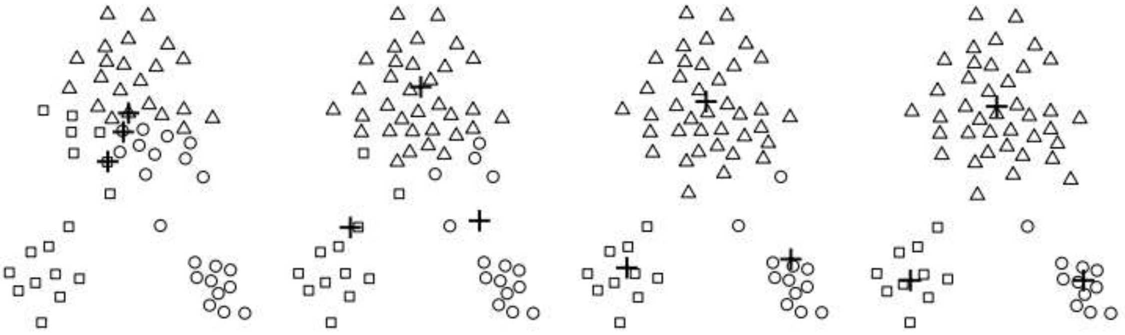
The Basic K-means Algorithm

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



Algorithm 8.1 Basic K-means algorithm.

```
Select $K$ points as initial centroids.
repeat
    Form $K$ clusters by assigning each point to its closest centroid.
    Recompute the centroid of each cluster.
until Centroids do not change.
```



(a) Iteration 1. (b) Iteration 2. (c) Iteration 3. (d) Iteration 4.

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

Assigning Points to the Closest Centroid

To assign a point to the closest centroid, we need a proximity measure that quantifies the notion of “closest” for the specific data under consideration. Euclidean (L2) distance is often used for data points in Euclidean space, while cosine similarity is more appropriate for documents.

Manhattan (L1) distance can be used for Euclidean data, while the Jaccard measure is often employed for documents.

Centroids and Objective Functions

Step 4 of the K-means algorithm was stated rather generally as “recompute the centroid of each cluster,” since the centroid can vary, depending on the proximity measure for the data and the goal of the clustering. The goal of the clustering is typically expressed by an objective function that depends on the proximities of the points to one another or to the cluster centroids; e.g., minimize the squared distance of each point to its closest centroid.

Data in Euclidean Space

Consider data whose proximity measure is Euclidean distance. For our objective function, which measures the quality of a clustering, we use the sum of the squared error (SSE), which is also known as scatter. In other words, we calculate the error of each data point, i.e., its Euclidean distance to the closest centroid, and then compute the total sum of the squared errors.

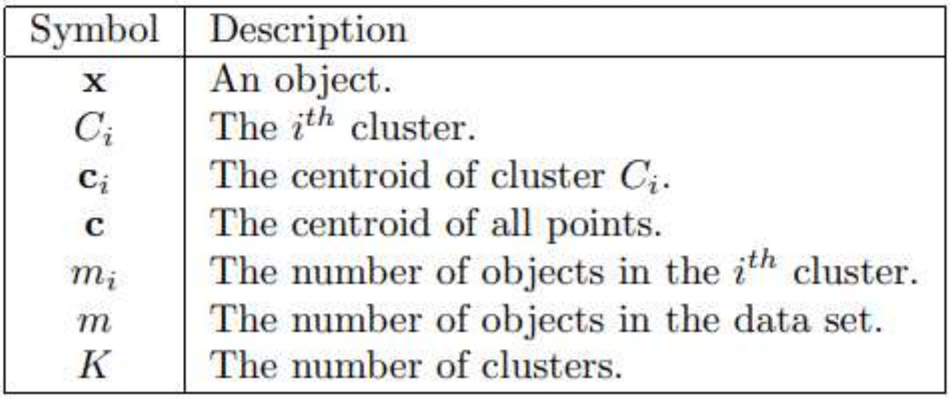
$$\text{SSE} = \sum_{i=1}^{K} \sum_{x \in C_i} dist(c_i, x)^2$$

where dist is the standard Euclidean (L2) distance between two objects in Euclidean space.

The centroid (mean) of the i th cluster is defined by

$$\mathbf{c}_i = \frac{1}{m_i} \sum_{\mathbf{x} \in C_i} \mathbf{x}$$

Table 8.1. Table of notation.



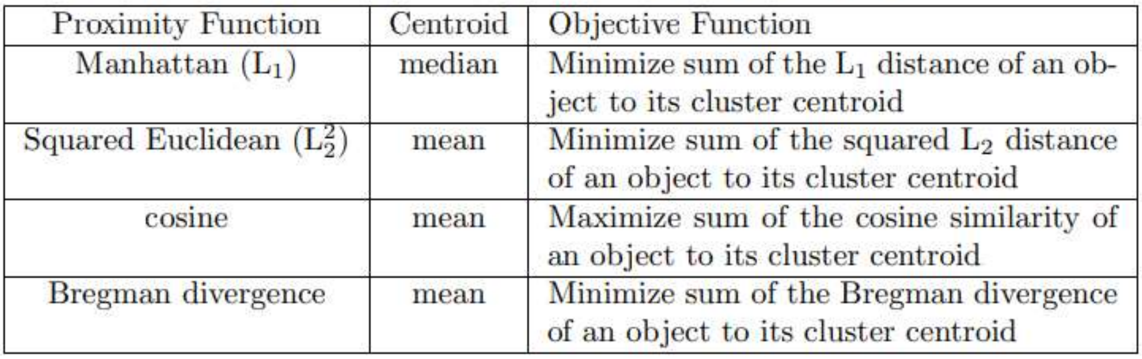
Symbol	Description
$\mathbf{x}$	An object.
C_i	The i^{th} cluster.
$\mathbf{c}_i$	The centroid of cluster C_i.
$\mathbf{c}$	The centroid of all points.
m_i	The number of objects in the i^{th} cluster.
m	The number of objects in the data set.
K	The number of clusters.

Document Data To illustrate that K-means is not restricted to data in Euclidean space, we consider document data and the cosine similarity measure. Here we assume that the document data is represented as a document-term matrix.

Our objective is to maximize the similarity of the documents in a cluster to the cluster centroid; this quantity is known as the cohesion of the cluster. For this objective it can be shown that the cluster centroid is, as for Euclidean data, the mean. The analogous quantity to the total SSE is the total cohesion, which is given by

$$\text{Total Cohesion} = \sum_{i=1}^{K} \sum_{\mathbf{x} \in C_i} cosine(\mathbf{x}, \mathbf{c}_i)$$

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

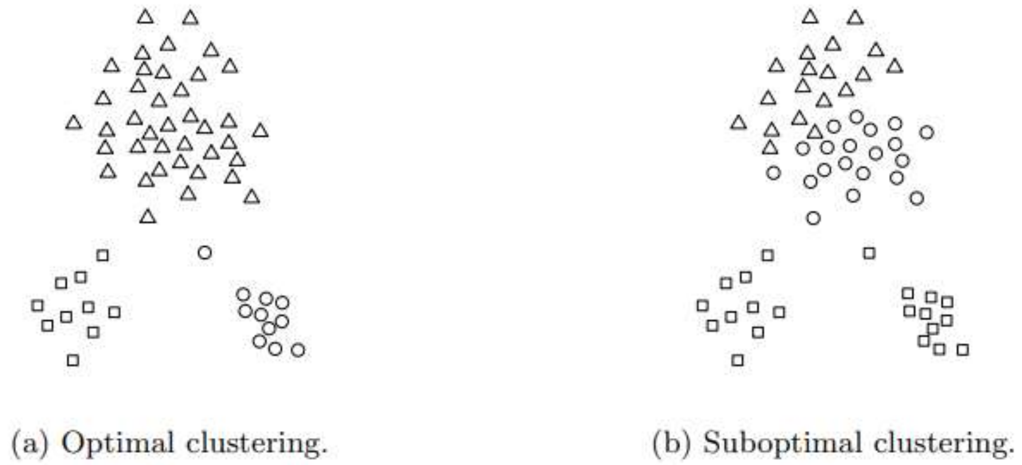


Proximity Function	Centroid	Objective Function
Manhattan (L_1)	median	Minimize sum of the L_1 distance of an object to its cluster centroid
Squared Euclidean (L_2^2)	mean	Minimize sum of the squared L_2 distance of an object to its cluster centroid
cosine	mean	Maximize sum of the cosine similarity of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

Choosing Initial Centroids

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

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



(a) Optimal clustering.

(b) Suboptimal clustering.

Figure 8.4. Three optimal and non-optimal clusters.

Example 8.1 (Poor Initial Centroids).

Example 8.2 (Limits of Random Initialization)

Time and Space Complexity

The **space requirements** for K-means are modest because only the data points and centroids are stored. Specifically, the storage required is O((m + K)n), where m is the number of points and n is the number of attributes. The time requirements for K-means are also modest—basically linear in the number of data points. In particular, the **time required** is O(I ∗K ∗m∗n), where I is the number of iterations required for convergence.

K-means: Additional Issues

Handling Empty Clusters

One of the problems with the basic K-means algorithm given earlier is that empty clusters can be obtained if no points are allocated to a cluster during the assignment step. If this happens, then a strategy is needed to choose a replacement centroid, since otherwise, the squared error will be larger than necessary.

One approach is to choose the point that is farthest away from any current centroid. If nothing else, this eliminates the point that currently contributes most to the total squared error. Another approach is to choose the replacement centroid from the cluster that has the highest SSE. This will typically split the cluster and reduce the overall SSE of the clustering. If there are several empty clusters, then this process can be repeated several times.

Outliers

Outliers can also be identified in a postprocessing step. For instance, we can keep track of the SSE contributed by each point, and eliminate those points with unusually high contributions, especially over multiple runs. Also, we may want to eliminate small clusters since they frequently represent groups of outliers.

Reducing the SSE with Postprocessing

An obvious way to reduce the SSE is to find more clusters, i.e., to use a larger K. However, in many cases, we would like to improve the SSE, but don't want to increase the number of clusters. This is often possible because K-means typically converges to a local minimum.

One commonly used approach is to use alternate cluster splitting and merging phases. During a splitting phase, clusters are divided, while during a merging phase, clusters are combined. In this way, it is often possible to escape local SSE minima and still produce a clustering solution with the desired number of clusters.

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

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

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

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

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

Merge two clusters: The clusters with the closest centroids are typically chosen, although another, perhaps better, approach is to merge the two clusters that result in the smallest increase in total SSE. These two merging strategies are the same ones that are used in the hierarchical clustering techniques known as the **centroid method** and **Ward's method,** respectively.

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.

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

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

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

Agglomerative Hierarchical Clustering

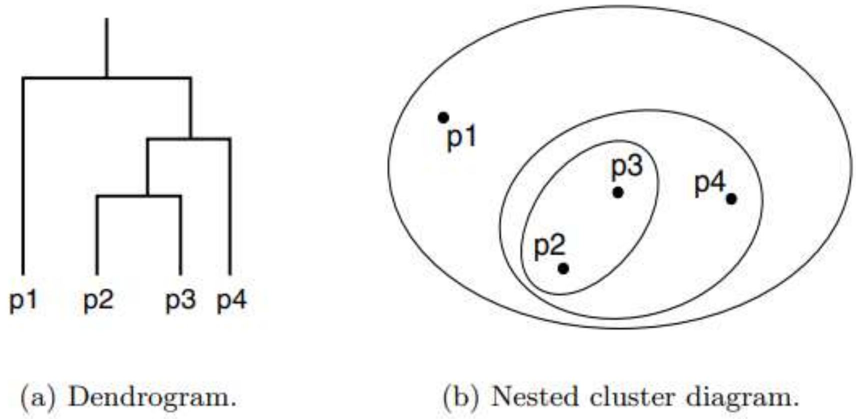
Hierarchical clustering techniques are a second important category of clustering methods. As with K-means, these approaches are relatively old compared to many clustering algorithms, but they still enjoy widespread use. There are two basic approaches for generating a hierarchical clustering:

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

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

Agglomerative hierarchical clustering techniques are by far the most common, and, in this section, we will focus exclusively on these methods.

A hierarchical clustering is often displayed graphically using a tree-like diagram called a **dendrogram**, which displays both the cluster-subcluster relationships and the order in which the clusters were merged (agglomerative view) or split (divisive view). For sets of two-dimensional points, such as those that we will use as examples, a hierarchical clustering can also be graphically represented using a nested cluster diagram. Figure 8.13 shows an example of these two types of figures for a set of four two-dimensional points.



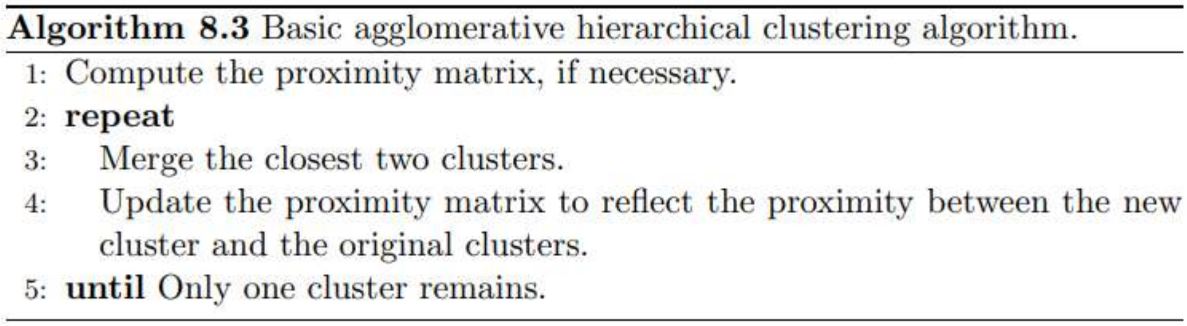
(a) Dendrogram.

(b) Nested cluster diagram.

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

Basic Agglomerative Hierarchical Clustering Algorithm

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



Algorithm 8.3 Basic agglomerative hierarchical clustering algorithm.

```
Compute the proximity matrix, if necessary.
repeat
    Merge the closest two clusters.
    Update the proximity matrix to reflect the proximity between the new
    cluster and the original clusters.
until Only one cluster remains.
```

Defining Proximity between Clusters

The key operation of Algorithm 8.3 is the computation of the proximity between two clusters, and it is the definition of cluster proximity that differentiates the various agglomerative hierarchical techniques that we will dis cuss. Cluster proximity is typically defined with a particular type of cluster in mind—see Section 8.1.2. For example, many agglomerative hierarchical clustering techniques, such as **MIN**, **MAX**, and **Group Average**, come from a graph-based view of clusters.

MIN defines cluster proximity as the proximity between the closest two points that are in different clusters, or using graph terms, the shortest edge between two nodes in different subsets of nodes. This yields contiguity-based clusters as shown in Figure 8.2(c).

Alternatively, MAX takes the proximity between the farthest two points in different clusters to be the cluster proximity, or using graph terms, the longest edge between two nodes in different subsets of nodes. (If our proximities are distances, then the names, MIN and MAX, are short and suggestive. For similarities, however, where higher values indicate closer points, the names seem reversed. For that reason, we usually prefer to use the alternative names, **single link** and **complete link, respectively**.)

Another graph-based approach, the group average technique, defines cluster proximity to be the average pairwise proximities (average length of edges) of all pairs of points from different clusters. Figure 8.14 illustrates these three approaches.

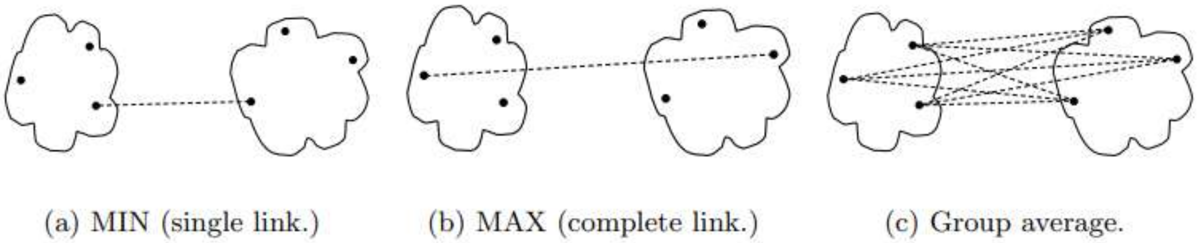


Figure 8.14. Graph-based definitions of cluster proximity

If, instead, we take a prototype-based view, in which each cluster is represented by a centroid, different definitions of cluster proximity are more natural. When using centroids, the cluster proximity is commonly defined as the proximity between cluster centroids. An alternative technique, **Ward's method**, also assumes that a cluster is represented by its centroid, but it measures the proximity between two clusters in terms of the increase in the SSE that results from merging the two clusters. Like K-means, Ward's method attempts to minimize the sum of the squared distances of points from their cluster centroids.

Time and Space Complexity

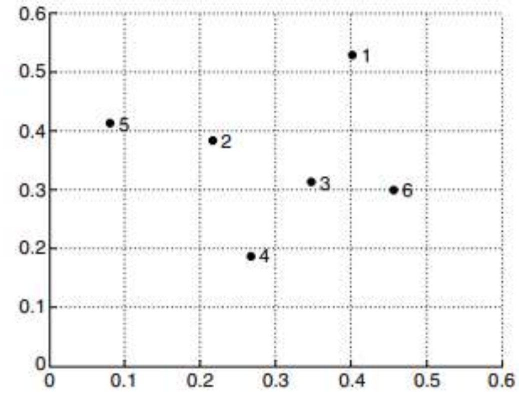
The basic agglomerative hierarchical clustering algorithm just presented uses a proximity matrix. This requires the storage of ½m^2 proximities (assuming the proximity matrix is symmetric) where m is the number of data points. The space needed to keep track of the clusters is proportional to the number of clusters, which is m−1, excluding singleton clusters. Hence, the total space complexity is O(m^2).

Without modification, this would yield a time complexity of O(m^3). If the distances from each cluster to all other clusters are stored as a sorted list (or heap), it is possible to reduce the cost of finding the two closest clusters to O(m − i + 1). However, because of the additional complexity of keeping data in a sorted list or heap, the overall time required for a hierarchical clustering based on Algorithm 8.3 is O(m^2 log m).

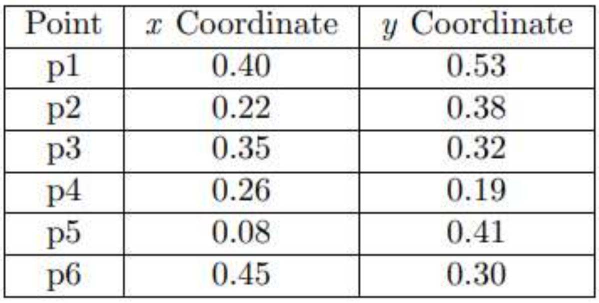
Specific Techniques

Sample Data

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

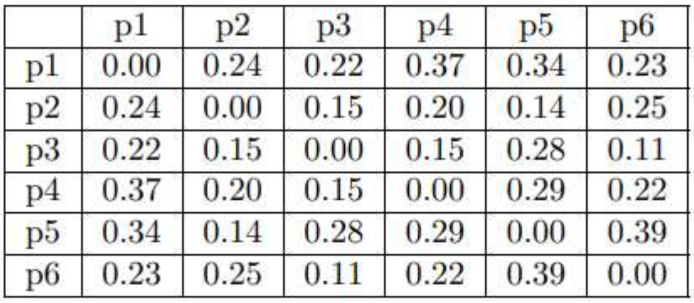


igure 8.15. Set of 6 two-dimensional points.



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

Table 8.3. xy coordinates of 6 points.



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

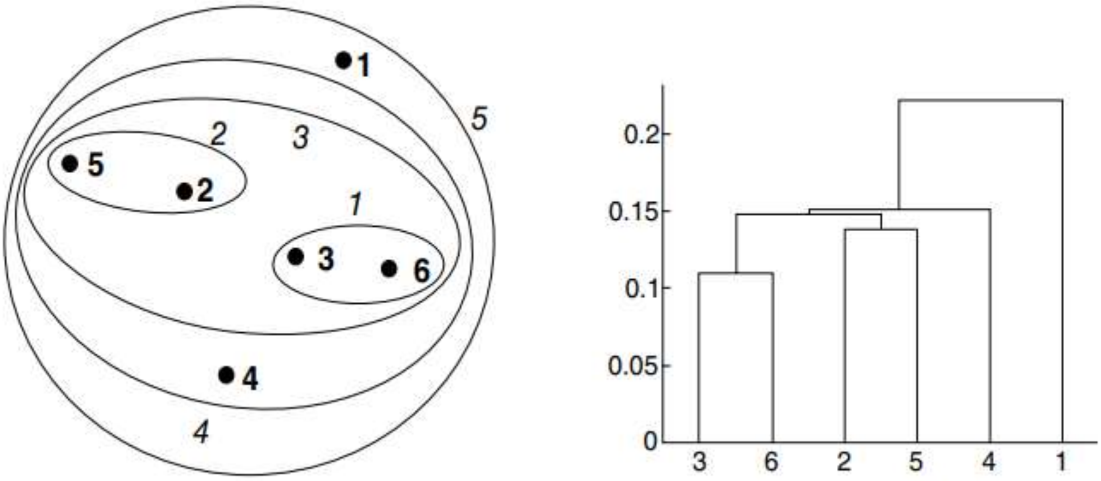
Table 8.4. Euclidean distance matrix for 6 points.

Single Link or MIN

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

Example 8.4 (Single Link). Figure 8.16 shows the result of applying the single link technique to our example data set of six points. Figure 8.16(a) shows the nested clusters as a sequence of nested ellipses, where the numbers associated with the ellipses indicate the order of the clustering. Figure 8.16(b) shows the same information, but as a dendrogram. The height at which two clusters are merged in the dendrogram reflects the distance of the two clusters. For instance, from Table 8.4, we see that the distance between points 3 and 6 is 0.11, and that is the height at which they are joined into one cluster in the dendrogram. As another example, the distance between clusters {3, 6} and {2, 5} is given by

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



(a) Single link clustering.

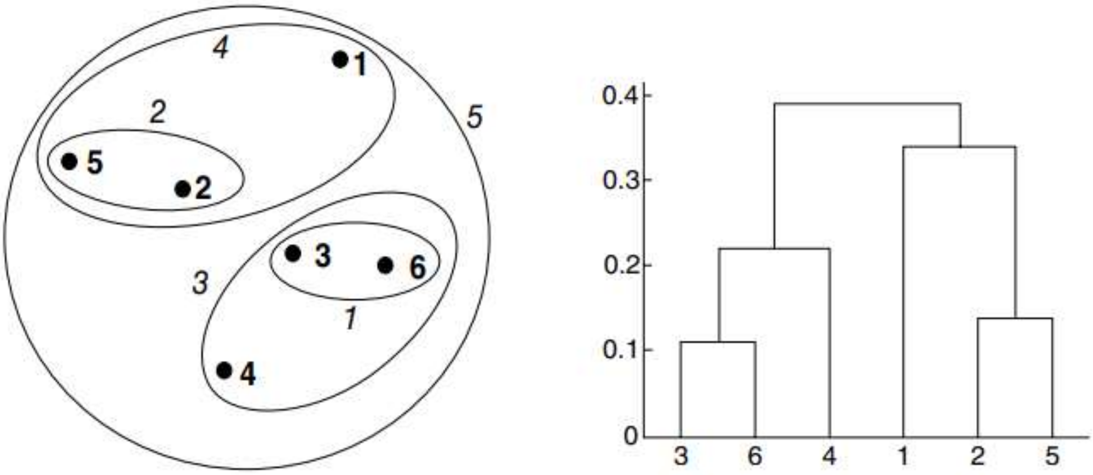
(b) Single link dendrogram.

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

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

Example 8.5 (Complete Link). Figure 8.17 shows the results of applying MAX to the sample data set of six points. As with single link, points 3 and 6 are merged first. However, {3, 6} is merged with {4}, instead of {2, 5} or {1} because

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

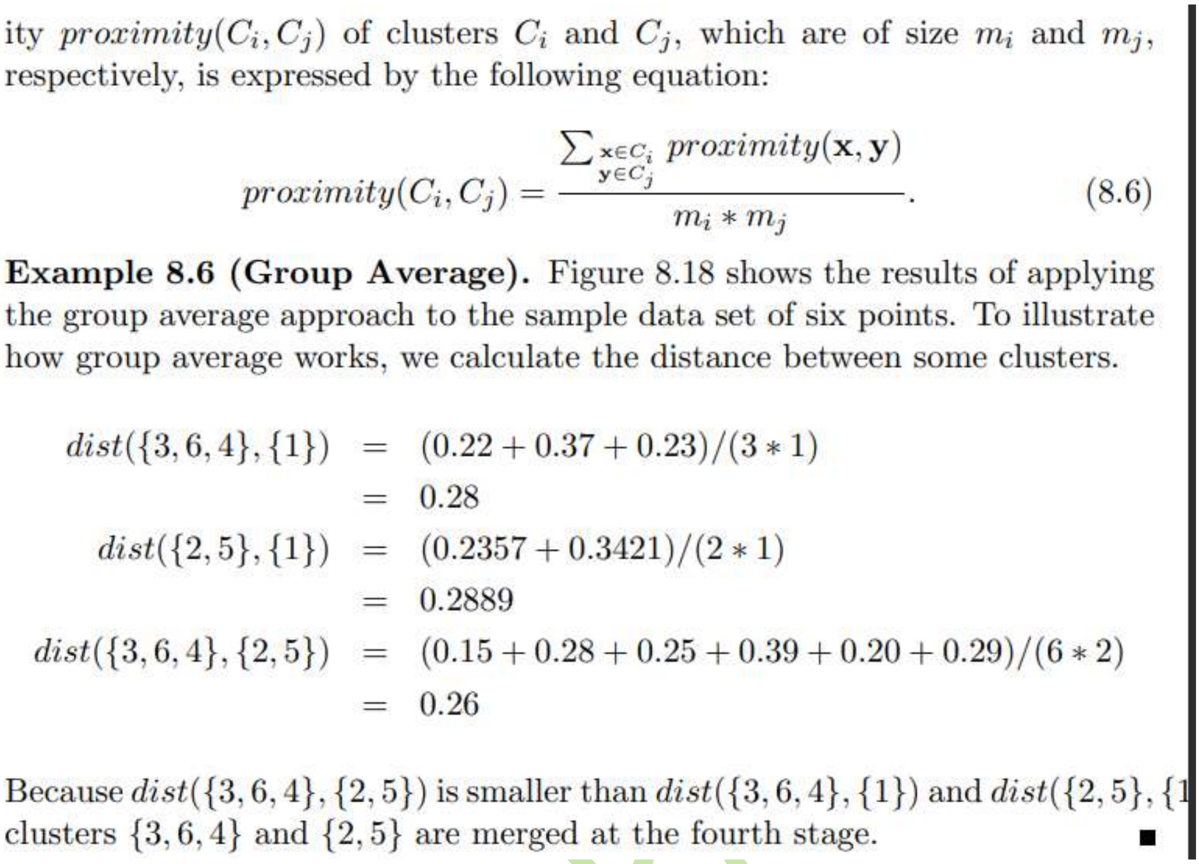


(a) Complete link clustering.

(b) Complete link dendrogram.

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

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



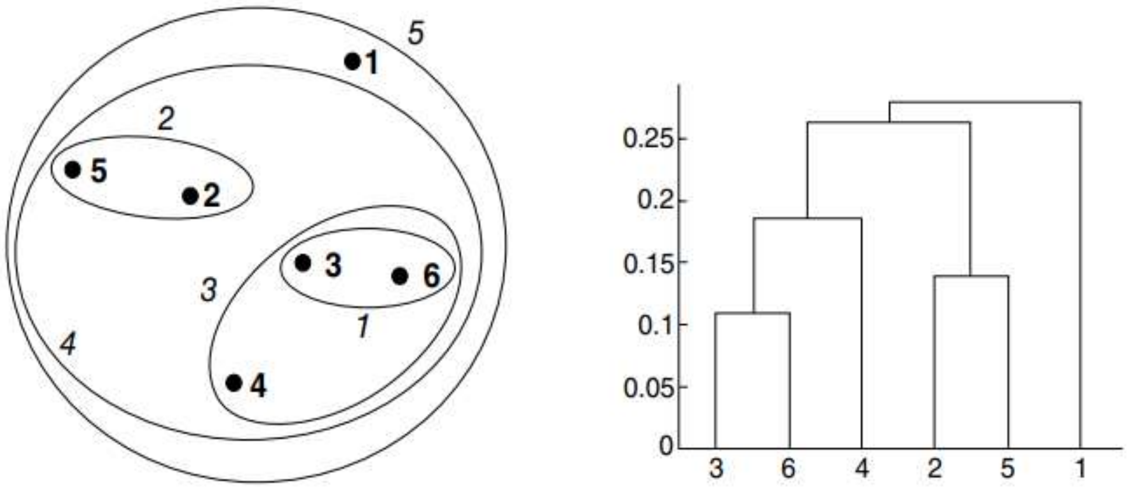
ity $proximity(C_i, C_j)$ of clusters C_i and C_j, which are of size m_i and m_j, respectively, is expressed by the following equation:

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

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

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

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



(a) Group average clustering.

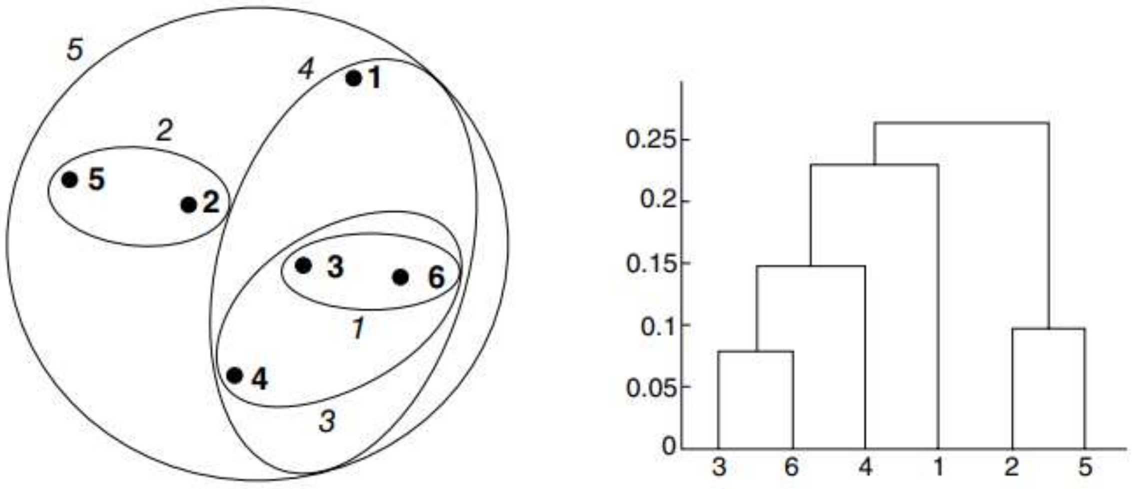
(b) Group average dendrogram.

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

Ward’s Method and Centroid Methods

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

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



(a) Ward's clustering.

(b) Ward's dendrogram.

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

Centroid methods calculate the proximity between two clusters by calculating the distance between the centroids of clusters. These techniques may seem similar to K-means, but as we have remarked, Ward's method is the correct hierarchical analog. Centroid methods also have a characteristic—often considered bad—that is not possessed by the other hierarchical clustering techniques that we have discussed: the possibility of inversions. Specifically, two clusters that are merged may be more similar (less distant) than the pair of clusters that were merged in a previous step. For the other methods, the distance between merged clusters monotonically increases (or is, at worst, non-increasing) as we proceed from singleton clusters to one all-inclusive cluster.

8.3.3 The Lance-Williams Formula for Cluster Proximity

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

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

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

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

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