JNTU HYDERABAD

DATA MINING

B.TECH./CSE & IT/R16 & R18

SYLLABUS

UNIT – I

Data Mining: Data – Types of Data – Data Mining Functionalities – Interestingness Patterns – Classification of Data Mining systems – Data mining Task primitives – Integration of Data mining system with a Data warehouse – Major issues in Data Mining – Data Pre-processing.

UNIT – II

Association Rule Mining: Mining Frequent Patterns – Associations and correlations – Mining Methods – Mining Various kinds of Association Rules – Correlation Analysis – Constraint based Association mining. Graph Pattern Mining, SPM.

UNIT – III

Classification: Classification and Prediction – Basic concepts – Decision tree induction – Bayesian classification, Rule-based classification, Lazy learner.

UNIT – IV

Clustering and Applications: Cluster analysis – Types of Data in Cluster Analysis – Categorization of Major Clustering Methods – Partitioning Methods, Hierarchical Methods – Density – Based Methods, Grid – Based Methods, Outlier Analysis.

UNIT – V

Advanced Concepts: Basic concepts in Mining data streams – Mining Time – series data – Mining sequence patterns in Transactional databases – Mining Object – Spatial – Multimedia – Text and Web data – Spatial Data mining – Multimedia Data mining – Text Mining – Mining the World Wide Web.

UNIT – I

DATA MINING

DATA MINING

Why Data Mining?

⇒ The major reason that data mining has attracted a great deal of attention in the information industry in recent years is due to the wide availability of huge amounts of data and need for turning such data into useful information and knowledge.

⇒ The information and knowledge gained can be used for applications ranging from business management, production control, and market analysis, to engineering design and science exploration.

⇒ Data mining can be viewed as a result of the natural evolution of information technology. It means, providing a path to extract the required data of an industry from warehousing machine. This is the witness of developing knowledge of an industry.

⇒ It includes data collection, database creation, data management (i.e data storage and retrieval, and database transaction processing) and data analysis and understanding (involving data warehousing and data mining).

Evolution of data mining and data warehousing: In the development of data mining, we should know the evolution of database. This includes,

1) **Data collection and Database creation:** In the 1960's, database and information technology began with file processing system. It is powerful database system. But it is providing inconsistency of data. It means, a user needs to maintain duplicate data of an industry.
2) **Database Management System:** In b/w 1970 – 1980, the progress of database is:
   - ⇒ Hierarchical and network database systems were developed.
   - ⇒ Relational database systems were developed.
   - ⇒ Data modeling tools were developed in early 1980s (such as E-R model etc.

⇒ Indexing and data organization techniques were developed. (such as B+ tree, hashing etc).

⇒ Query languages were developed. (such as SQL, PL/SQL).

⇒ User interfaces, forms and reports, query processing.

⇒ On-line transaction processing (OLTP).

3) **Advanced Database Systems:** In mid 1980s to till date, Advanced data models were developed. (such as extended relational, object-oriented, object-relational, spatial, temporal, multimedia, scientific databases etc.

4) **Data Warehousing and Data mining**: In late 1980 to till date,

⇒ Developed Data warehouse and OLAP technology.

⇒ Data mining and knowledge discovery were introduced.

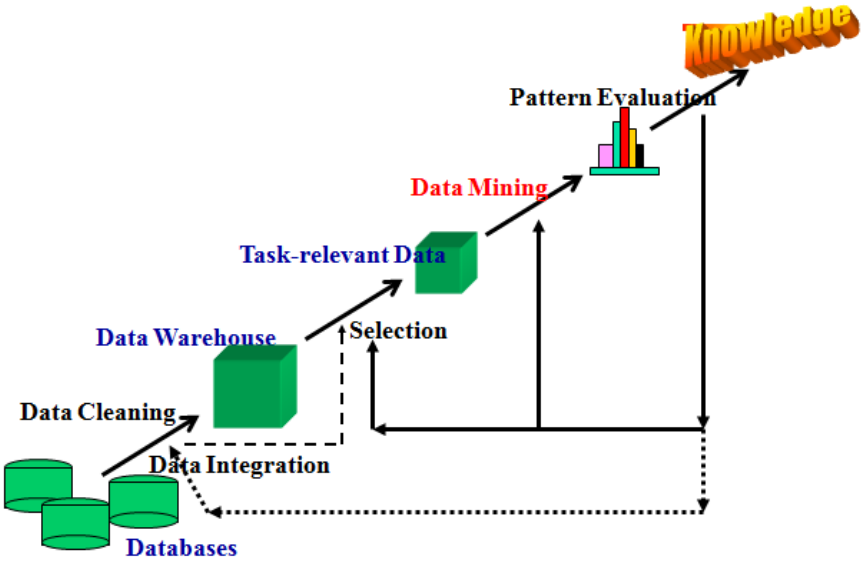
5) **Web-based Databases Systems**: In 1990 – till date, XML based database systems and web mining were developed.

6) **New Generation of Integrated Information Systems:** From 2000 onwards developed an integrated information system.

What is Data Mining?

The term Data Mining refers to extracting or "mining" knowledge from large amounts of data. The term mining is actually a misnomer (i.e. unstructured data). For example, mining of gold from rocks or sand is referred to as gold mining.

Data mining is the process of discovering meaningful new trends by storing the large amount of data in repository of database. It also uses pattern recognition techniques as well as statistical techniques.



Data mining steps in the knowledge discovery process (KDD): The Data mining is a step in the Knowledge Discovery in Databases (KDD). It has different stages, such as:

1) **Data Cleaning:** It is the process of removing noise and inconsistent data.
2) **Data Integrating:** It is the process of combining data from multiple sources.
3) **Data Selection:** It is the process of retrieving relevant data from database.
4) **Data Transformation:** In this process, data are transformed or consolidated into forms or reports by performing summary or aggregation operation.
5) **Data Mining:** It is an essential process to extracting data from raw data by using intelligent methods.
6) **Pattern Evaluation:** To identify the discovered data is in the knowledge based on some interestingness measures. (i.e identify the mined data is in the required format or not.).
7) **Knowledge presentation:** Visualization and knowledge representation techniques are used to present the mined data to the user.

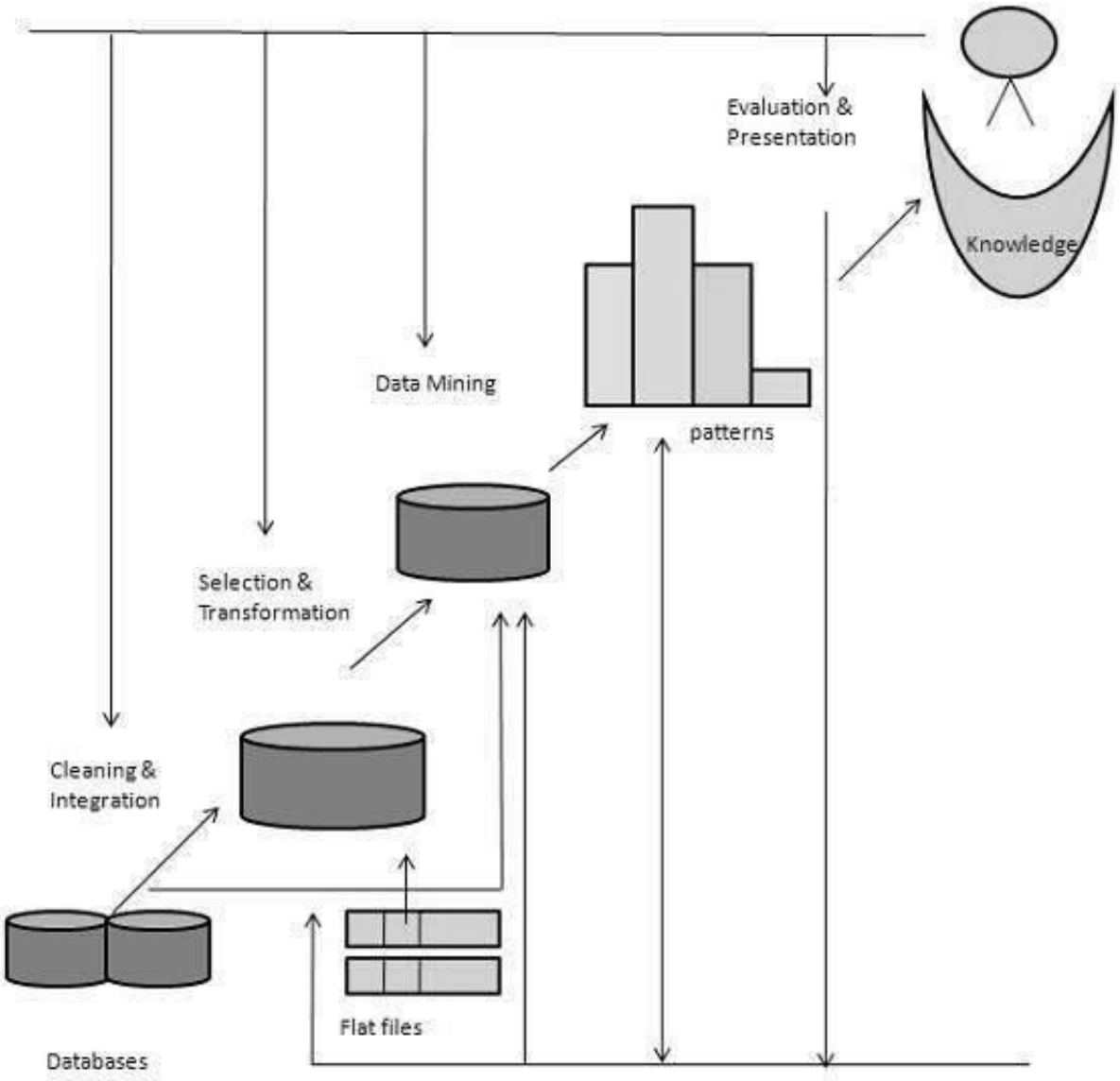
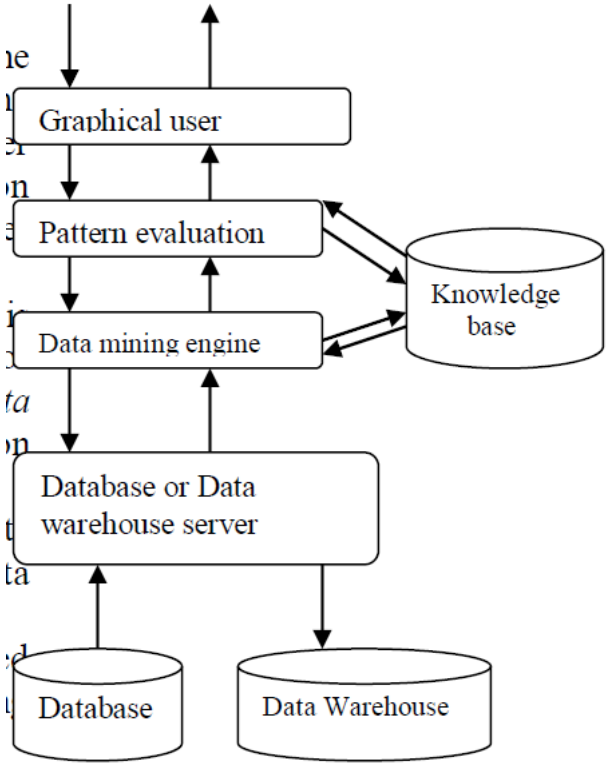


Fig: Knowledge Discovery in Databases Process

Architecture of Data Mining System: The architecture of data mining is the process of discovering the interesting knowledge from large amounts of data stored either in databases or in the data warehouse or information repositories. It has various stages to extract the data into user view from unstructured sources.

1) **Database, data warehouse, or information repositories:** This is single or set of databases, data warehouses, spreadsheets, or other kinds of information repositories. In this step, the Data Cleaning and Data Integration techniques may be performed on the data.

2) **Database or data warehouse server:** The database or data warehouse server is responsible for fetching the relevant data based on the user's data mining request.
3) **Knowledge base:** This is the domain knowledge that is used to guide the search or evaluate the interestingness of resulting patterns.
4) **Data mining engine:** This is essential to the data mining system and ideally consists of a set of functional modules for tasks such as characterization, association, classification, cluster analysis, and evolution and deviation analysis.
5) **pattern evaluation module:** This step providing measures, constraints(rules), methods etc to filter out the discovered patterns or data. This is most useful for efficient data mining.
6) **Graphical user interface:** This step provides the communication b/w user and data mining system. It allows the user to interact with the system by specifying a data mining query or task.



Which Kinds of Applications Are Targeted?

Data mining has seen great successes in many applications. Presentations of data mining in knowledge-intensive application domains, such as bioinformatics and software engineering.

1) Business intelligence (BI) technologies provide historical, current, and predictive views of business operations. Examples include reporting, online analytical processing, business performance management, competitive intelligence, benchmarking, and predictive analytics.

Data mining is the core of business intelligence. Online analytical processing tools in business intelligence depend on data warehousing and multidimensional data mining. Classification and prediction techniques are the core of predictive analytics in business intelligence, for which there are many applications in analyzing markets, supplies, and sales.

2) A Web search engine is a specialized computer server that searches for information on the Web. The search results of a user query are often returned as a list (sometimes called hits). The hits may consist of web pages, images, and other types of files.

Web search engines are essentially very large data mining applications. Various data mining techniques are used in all aspects of search engines, ranging from crawling (e.g., deciding which pages should be crawled and the crawling frequencies), indexing (e.g., selecting pages to be indexed and deciding to which extent the index should be constructed), and searching (e.g., deciding how pages should be ranked, which advertisements should be added, and how the search results can be personalized or made "context aware").

TYPES OF DATA MINING [OR] DATA MINING–ON WHAT KIND OF DATA? [OR] WHAT KIND OF DATA CAN BE MINED?

In this section, we examine a number of different data repositories on which mining can be performed. In principle, data mining should be applicable to any kind of data repository, as well as to transient data, such as data streams. Thus the scope of our examination of data repositories will include relational

databases, data warehouses, transactional databases, advanced database systems, flat files, data streams, and the World Wide Web. Advanced database systems include object-relational databases and specific application-oriented databases, such as spatial databases, time-series databases, text databases, and multimedia databases. The challenges and techniques of mining may differ for each of the repository systems.

1) Relational Databases: A database system, also called a database management system (DBMS), consists of a collection of interrelated data, known as a database, and a set of software programs to manage and access the data. The software programs involve mechanisms for the definition of database structures; for data storage; for concurrent, shared, or distributed data access; and for ensuring the consistency and security of the information stored, despite system crashes or attempts at unauthorized access.

A relational database is a collection of tables, each of which is assigned a unique name. Each table consists of a set of attributes (columns or fields) and usually stores a large set of tuples (records or rows). Each tuple in a relational table represents an object identified by a unique key and described by a set of attribute values. A semantic data model, such as an entity-relationship (ER) data model, is often constructed for relational databases. An ER data model represents the database as a set of entities and their relationships.

Consider the following example.

customer

cust_ID	*name*	*address*	*age*	*income*	*credit_info*	*category*	...
C1	Smith, Sandy	1223 Lake Ave., Chicago, IL	31	$78000	1	3	...
...	...	...	...	...	...	...	...

item

item_ID	*name*	*brand*	*category*	*type*	*price*	*place_made*	*supplier*	*cost*
I3	hi-res-TV	Toshiba	high resolution	TV	$988.00	Japan	NikoX	$600.00
I8	Laptop	Dell	laptop	computer	$1369.00	USA	Dell	$983.00
...	...	...	...	...	...	...	...	...

employee

empl_ID	*name*	*category*	*group*	*salary*	*commission*
E55	Jones, Jane	home entertainment	manager	$118,000	2%
...	...	...	...	...	...

branch

branch_ID	*name*	*address*
B1	City Square	396 Michigan Ave., Chicago, IL
...	...	...

purchases

trans_ID	*cust_ID*	*empl_ID*	*date*	*time*	*method_paid*	*amount*
T100	C1	E55	03/21/2005	15:45	Visa	$1357.00
...	...	...	...	...	...	...

items_sold

trans_ID	*item_ID*	*qty*
T100	I3	1
T100	I8	2
...	...	...

works_at

empl_ID	*branch_ID*
E55	B1
...	...

Figure 1.6 Fragments of relations from a relational database for *AllElectronics*.

2) Data Warehouses: Suppose that AllElectronics is a successful international company, with branches around the world. Each branch has its own set of databases. The president of AllElectronics has asked you to provide an analysis of the company's sales per item type per branch for the third quarter. This is a difficult task, particularly since the relevant data are spread out over several databases, physically located at numerous sites.

If AllElectronics had a data warehouse, this task would be easy. A data warehouse is a repository of information collected from multiple sources, stored under a unified schema, and that usually resides at a single site. Data warehouses are constructed via a process of data cleaning, data integration,

data transformation, data loading, and periodic data refreshing. Figure 1.7 shows the typical framework for construction and use of a data warehouse for AllElectronics.

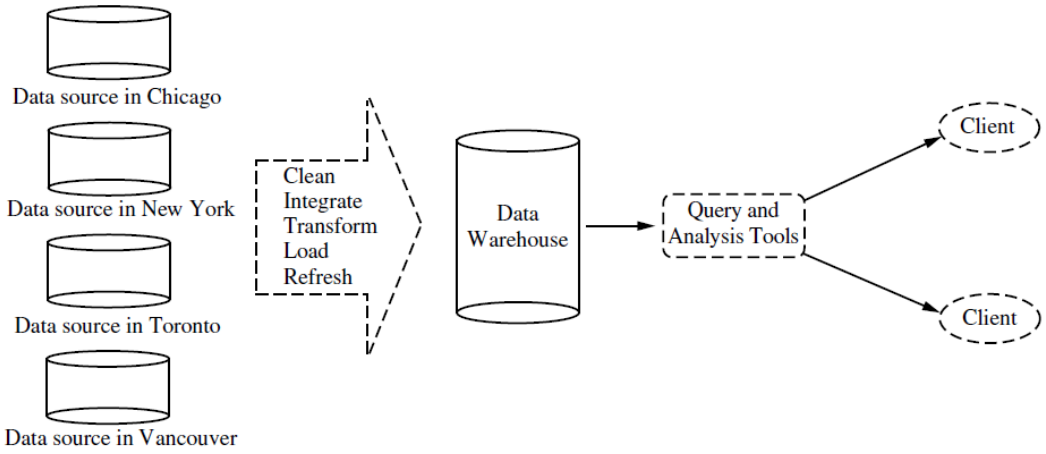


Figure 1.7 Typical framework of a data warehouse for *AllElectronics*.

To facilitate decision making, the data in a data warehouse are organized around major subjects, such as customer, item, supplier, and activity. The data are stored to provide information from a historical perspective (such as from the past 5–10 years) and are typically summarized. For example, rather than storing the details of each sales transaction, the data warehouse may store a summary of the transactions per item type for each store or, summarized to a higher level, for each sales region.

A data warehouse is usually modeled by a multidimensional database structure, where each dimension corresponds to an attribute or a set of attributes in the schema, and each cell stores the value of some aggregate measure, such as count or sales amount. The actual physical structure of a data warehouse may be a relational data store or a multidimensional data cube. A data cube provides a multidimensional view of data and allows the precomputation and fast accessing of summarized data.

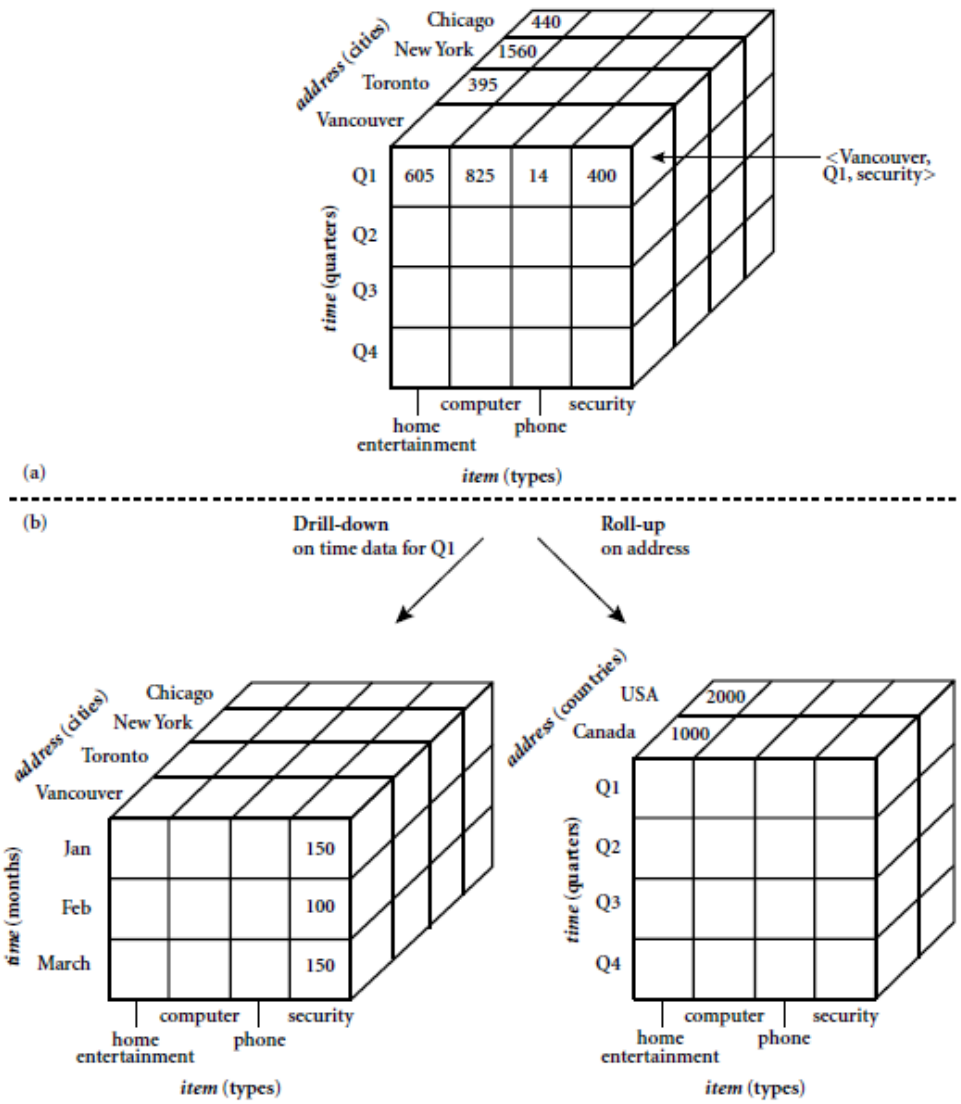


Figure 1.8 A multidimensional data cube, commonly used for data warehousing, (a) showing summarized data for *AllElectronics* and (b) showing summarized data resulting from drill-down and roll-up operations on the cube in (a). For improved readability, only some of the cube cell values are shown.

3) Transactional Databases: In general, a transactional database consists of a file where each record represents a transaction. A transaction typically includes a unique transaction identity number (trans ID) and a list of the items making up the transaction (such as items purchased in a store).

trans_ID	*list of item_IDs*
T100	I1, I3, I8, I16
T200	I2, I8
...	...

Figure 1.9 Fragment of a transactional database for sales at *AllElectronics*.

The transactional database may have additional tables associated with it, which contain other information regarding the sale, such as the date of the transaction, the customer ID number, the ID number of the salesperson and of the branch at which the sale occurred, and so on.

4) Advanced Data and Information Systems and Advanced Applications: Relational database systems have been widely used in business applications. With the progress of database technology, various kinds of advanced data and information systems have emerged and are undergoing development to address the requirements of new applications.

The new database applications include handling spatial data (such as maps), engineering design data (such as the design of buildings, system components, or integrated circuits), hypertext and multimedia data (including text, image, video, and audio data), time-related data (such as historical records or stock exchange data), stream data (such as video surveillance and sensor data, where data flow in and out like streams), and the World Wide Web (a huge, widely distributed information repository made available by the Internet). These applications require efficient data structures and scalable methods for handling complex object structures; variable-length records; semi structured or unstructured data; text, spatiotemporal, and multimedia data; and database schemas with complex structures and dynamic changes.

Object-Relational Databases: Object-relational databases are constructed based on an object-relational data model. This model extends the relational model by providing a rich data type for handling complex objects and object orientation. Because most sophisticated database applications need to handle

complex objects and structures, object-relational databases are becoming increasingly popular in industry and applications.

Temporal Databases, Sequence Databases, and Time-Series Databases: A temporal database typically stores relational data that include time-related attributes. These attributes may involve several timestamps, each having different semantics. A sequence database stores sequences of ordered events, with or without a concrete notion of time. Examples include customer shopping sequences, Web click streams, and biological sequences. A time-series database stores sequences of values or events obtained over repeated measurements of time (e.g., hourly, daily, weekly). Examples include data collected from the stock exchange, inventory control, and the observation of natural phenomena (like temperature and wind).

Spatial Databases and Spatiotemporal Databases: Spatial databases contain spatial-related information. Examples include geographic (map) databases, very large-scale integration (VLSI) or computed-aided design databases, and medical and satellite image databases. Spatial data may be represented in raster format, consisting of n-dimensional bit maps or pixel maps.

A spatial database that stores spatial objects that change with time is called a spatiotemporal database, from which interesting information can be mined. For example, we may be able to group the trends of moving objects and identify some strangely moving vehicles, or distinguish a bioterrorist attack from a normal outbreak of the flu based on the geographic spread of a disease with time.

Text Databases and Multimedia Databases: Text databases are databases that contain word descriptions for objects. These word descriptions are usually not simple keywords but rather long sentences or paragraphs, such as product specifications, error or bug reports, warning messages, summary reports, notes, or other documents. Text databases may be highly unstructured (such as some Web pages on the World Wide Web). Some text databases may be somewhat structured, that is, semi structured (such as e-mail messages and many HTML/XML Web pages), whereas others are

relatively well structured (such as library catalogue databases). Text databases with highly regular structures typically can be implemented using relational database systems.

Multimedia databases store image, audio, and video data. They are used in applications such as picture content-based retrieval, voice-mail systems, video-on-demand systems, the World Wide Web, and speech-based user interfaces that recognize spoken commands. Multimedia databases must support large objects, because data objects such as video can require gigabytes of storage. Specialized storage and search techniques are also required. Because video and audio data require real-time retrieval at a steady and predetermined rate in order to avoid picture or sound gaps and system buffer overflows, such data are referred to as continuous-media data.

Heterogeneous Databases and Legacy Databases: A heterogeneous database consists of a set of interconnected, autonomous component databases. The components communicate in order to exchange information and answer queries. Objects in one component database may differ greatly from objects in other component databases, making it difficult to assimilate their semantics into the overall heterogeneous database.

Many enterprises acquire legacy databases as a result of the long history of information technology development (including the application of different hardware and operating systems). A legacy database is a group of heterogeneous databases that combines different kinds of data systems, such as relational or object-oriented databases, hierarchical databases, network databases, spreadsheets, multimedia databases, or file systems. The heterogeneous databases in a legacy database may be connected by intra or inter-computer networks.

Data Streams: Many applications involve the generation and analysis of a new kind of data, called stream data, where data flow in and out of an observation platform (or window) dynamically.

The World Wide Web: The World Wide Web and its associated distributed information services, such as Yahoo!, Google, America Online, and AltaVista,

provide rich, worldwide, on-line information services, where data objects are linked together to facilitate interactive access.

DATA MINING FUNCTIONALITIES [OR] WHAT KINDS OF PATTERNS CAN BE MINED? [OR] DATA MINING FUNCTIONALITIES—WHAT KINDS OF PATTERNS CAN BE MINED?

Data mining functionalities are used to specify the kind of patterns to be found in data mining tasks. Data mining tasks are classified into two categories descriptive and predictive.

Descriptive mining tasks characterize the general properties of the data in the database.

Predictive mining tasks perform inference on the current data in order to make predictions.

1) Concept/Class Description: Descriptions of an individual classes or a concepts in summarized, concise and precise terms called class or concept descriptions. These descriptions can be divided into:

i. Data Characterization.
ii. Data Discrimination.

i. Data Characterization: It is summarization of the general characteristics of a target class of data (forms). The data corresponding to the user specified class are collected by a database query. The output of data characterization can be presented in various forms like pie charts, bar charts curves, multidimensional cubes, multidimensional tables etc. The resulting descriptions can be presented as generalized relations are called characteristic rules.

ii. Data Discriminations: Comparison of two target class data objects from one or set of contrasting (distinct) classes. The target and contrasting classes can be specified by the user, and the corresponding data objects are retrieved through database queries. For example, comparison of products whose sales increased by 10% in the last year with those whose sales decreased by 30% during the same period. This is called data discrimination.

2) Mining Frequent Patterns, Associations and Correlations:

i. Frequent Patterns: A frequent itemset typically refers to a set of items that often appear in a transactional data. For example, milk, and bread are frequently purchased by many customers. AllElectronics industry occurring the products which are frequently purchased by the customers. Generally, home needs are frequently used by the more customers.

ii. Association Analysis: "What is association analysis?"

Association analysis is the discovery of association rules showing attribute with value conditions that occur frequently together the given set of data. It is used for transaction data analysis. The Association rule of the form X ==> Y.

For example, In AllElectronics relational database, data mining system may find association rules like

buys(X, "computer") ==> buys(X, "software")

Here, who buys "computer", they buys "software".

age (X, "20 .. 29") & income (X, "20k .. 29k") ==> buys(X,

In this, the Association rule indicate that that indicates who employee of AllElectronics have the age b/w 20 to 29 and earning income b/w 20000 to 29000 are purchased CD player at AllElectronics Company.

3) Classification and Regressive prediction:

Classification is the process of finding a set of models that describes and distinguishes data classes or concepts. The derived model may be represented in various forms such as classification (IF-THEN) rules, decision trees, mathematical formulae or neural networks.

A decision tree is a flow-chart like tree structure. The decision trees can easily converted to classification rule. The neural networks are used for classification to provide connection b/w computers.

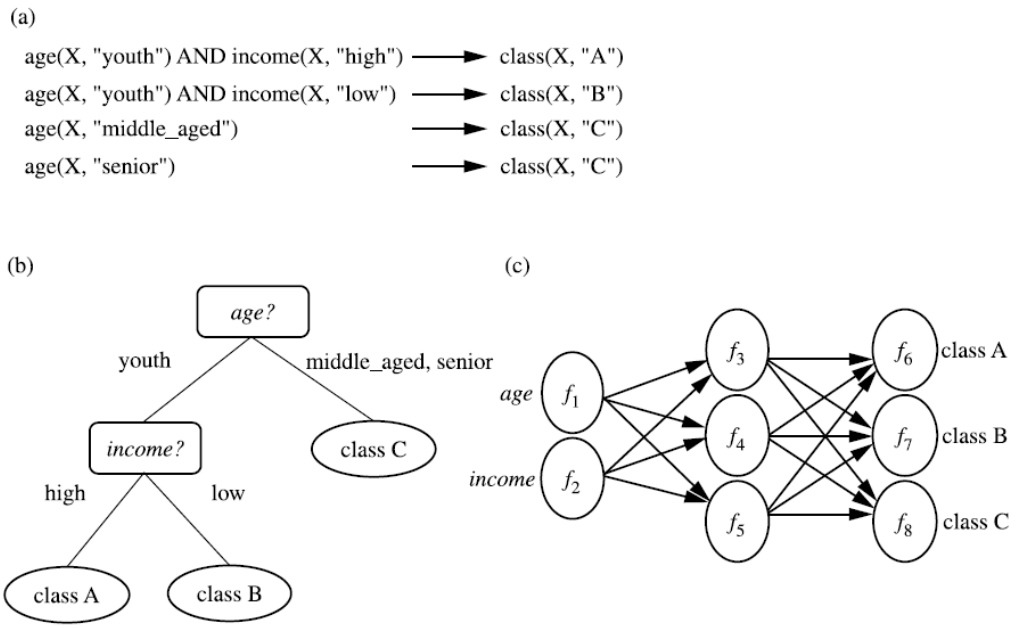


Figure 1.10 A classification model can be represented in various forms, such as (a) IF-THEN rules, (b) a decision tree, or a (c) neural network.

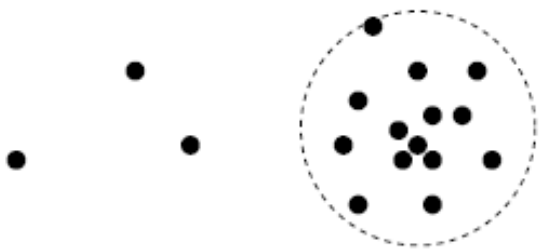
Regression for Predication is used to predict missing or unavailable data values rather than class labels. Prediction refers to both data value prediction and class label prediction. The predicted values are numerical data and are often referred to as prediction.

4) Cluster Analysis: ("What is cluster analysis?")

Clustering is a method of grouping data into different groups, so that in each group share similar trends and patterns. The objectives of clustering are:

i. To uncover natural groupings.
ii. To initiate hypothesis about the data.
iii. To find consistent and valid organization of data.

For example, Cluster analysis can be performed on AllElectronics customers. It means, to identify homogeneous (same group) customers. By this cluster may represent target groups for marketing to increase the sales.



5) Outlier Analysis: In this analysis, a database may contain data objects that do not do what someone wants. Most data mining methods discard outliers as noise or exceptions. Finding such type of applications are fraud detection is referred as outlier mining.

For example, Outlier analysis may uncover usage of credit cards by detecting purchases of large amount of products when comparing with regular purchase of large product customers.

6) Evolution Analysis: Data evolution analysis describes and models regularities or trends for objects whose behavior changes over time. Although this may include characterization, discrimination, association and correlation analysis, classification, prediction, or clustering of time related data, distinct features of such an analysis include time-series data analysis, sequence or periodicity pattern matching, and similarity-based data analysis.

INTERESTINGNESS PATTERNS [OR] ARE ALL OF THE PATTERNS INTERESTING?

A data mining system has the potential to generate thousands or even millions of patterns, or rules.

"So," you may ask, "are all of the patterns interesting?" Typically not only a small fraction of the patterns potentially generated would actually be of interest to any given user.

This raises some serious questions for data mining. You may wonder, "What makes a pattern interesting? Can a data mining system generate all of the interesting patterns? Can a data mining system generate only interesting patterns?"

To answer the first question, a pattern is interesting if it is (1) easily understood by humans, (2) valid on new or test data with some degree of certainty, (3) potentially useful, and (4) novel. A pattern is also interesting if it validates a hypothesis that the user sought to confirm. An interesting pattern represents knowledge.

Several **objective measures of pattern interestingness** exist. These are based on the structure of discovered patterns and the statistics underlying them. An objective measure for association rules of the form $X \Rightarrow Y$ is rule **support**, representing the percentage of transactions from a transaction database that the given rule satisfies. This is taken to be the probability $P(X \cup Y)$, where $X \cup Y$ indicates that a transaction contains both X and Y, that is, the union of itemsets X and Y. Another objective measure for association rules is **confidence**, which assesses the degree of certainty of the detected association. This is taken to be the conditional probability $P(Y|X)$, that is, the probability that a transaction containing X also contains Y. More formally, support and confidence are defined as

$$support(X \Rightarrow Y) = P(X \cup Y).$$
$$confidence(X \Rightarrow Y) = P(Y|X).$$

In general, each interestingness measure is associated with a threshold, which may be controlled by the user. For example, rules that do not satisfy a confidence threshold of, say, 50% can be considered uninteresting. Rules below the threshold likely reflect noise, exceptions, or minority cases and are probably of less value.

Although objective measures help identify interesting patterns, they are insufficient unless combined with subjective measures that reflect the needs and interests of a particular user. For example, patterns describing the characteristics of customers who shop frequently at AllElectronics should interest the marketing manager, but may be of little interest to analysts studying the same database for patterns on employee performance.

Furthermore, many patterns that are interesting by objective standards may represent common knowledge and, therefore, are actually uninteresting. Subjective interestingness measures are based on user beliefs in the data. These measures find patterns interesting if they are unexpected (contradicting a user's belief) or offer strategic information on which the user can act. In the latter case, such patterns are referred to as actionable. Patterns that are

expected can be interesting if they confirm a hypothesis that the user wished to validate, or resemble a user's hunch.

The second question—"Can a data mining system generate all of the interesting patterns?"—refers to the completeness of a data mining algorithm. It is often unrealistic and inefficient for data mining systems to generate all of the possible patterns. Instead, user-provided constraints and interestingness measures should be used to focus the search. For some mining tasks, such as association, this is often sufficient to ensure the completeness of the algorithm. Association rule mining is an example where the use of constraints and interestingness measures can ensure the completeness of mining.

Finally, the third question—"Can a data mining system generate only interesting patterns?"—is an optimization problem in data mining. It is highly desirable for data mining systems to generate only interesting patterns. This would be much more efficient for users and data mining systems, because neither would have to search through the patterns generated in order to identify the truly interesting ones. Progress has been made in this direction; however, such optimization remains a challenging issue in data mining.

Measures of pattern interestingness are essential for the efficient discovery of patterns of value to the given user. Such measures can be used after the data mining step in order to rank the discovered patterns according to their interestingness, filtering out the uninteresting ones. More importantly, such measures can be used to guide and constrain the discovery process, improving the search efficiency by pruning away subsets of the pattern space that do not satisfy prespecified interestingness constraints.

CLASSIFICATION OF DATA MINING SYSTEMS [OR] WHICH TECHNOLOGIES ARE USED?

Data mining is an interdisciplinary field, the confluence of a set of disciplines, including database systems, statistics, machine learning, visualization, and information science (Figure 1.12). Moreover, depending on the data mining approach used, techniques from other disciplines may be applied, such as

neural networks, fuzzy and/or rough set theory, knowledge representation, inductive logic programming, or high-performance computing.

Depending on the kinds of data to be mined or on the given data mining application, the data mining system may also integrate techniques from spatial data analysis, information retrieval, pattern recognition, image analysis, signal processing, computer graphics, Web technology, economics, business, bioinformatics, or psychology.

Because of the diversity of disciplines contributing to data mining, datamining research is expected to generate a large variety of data mining systems. Therefore, it is necessary to provide a clear classification of data mining systems, which may help potential users distinguish between such systems and identify those that best match their needs. Data mining systems can be categorized according to various criteria, as follows:

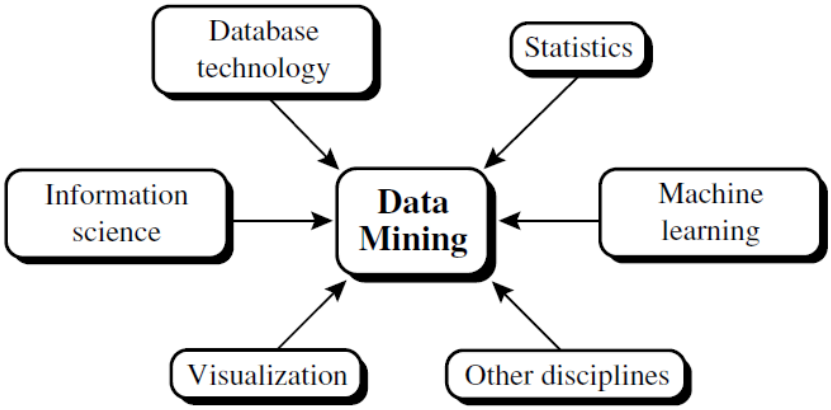


Figure 1.12 Data mining as a confluence of multiple disciplines.

Classification according to the kinds of databases mined: A data mining system can be classified according to the kinds of databases mined. Database systems can be classified according to different criteria (such as data models, or the types of data or applications involved), each of which may require its own data mining technique. Data mining systems can therefore be classified accordingly.

For instance, if classifying according to data models, we may have a relational, transactional, object-relational, or data warehouse mining system. If classifying according to the special types of data handled, we may have a

spatial, time-series, text, stream data, multimedia data mining system, or a World Wide Web mining system.

Classification according to the kinds of knowledge mined: Data mining systems can be categorized according to the kinds of knowledge they mine, that is, based on data mining functionalities, such as characterization, discrimination, association and correlation analysis, classification, prediction, clustering, outlier analysis, and evolution analysis. A comprehensive data mining system usually provides multiple and/or integrated data mining functionalities.

Moreover, data mining systems can be distinguished based on the granularity or levels of abstraction of the knowledge mined, including generalized knowledge (at a high level of abstraction), primitive-level knowledge (at a raw data level), or knowledge at multiple levels (considering several levels of abstraction). An advanced data mining system should facilitate the discovery of knowledge at multiple levels of abstraction.

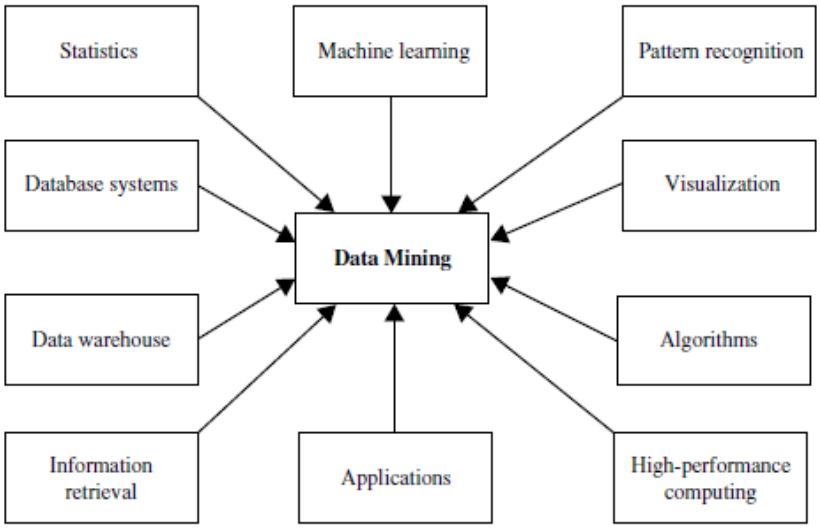
Data mining systems can also be categorized as those that mine data regularities (commonly occurring patterns) versus those that mine data irregularities (such as exceptions, or outliers). In general, concept description, association and correlation analysis, classification, prediction, and clustering mine data regularities, rejecting outliers as noise. These methods may also help detect outliers.

Classification according to the kinds of techniques utilized: Data mining systems can be categorized according to the underlying data mining techniques employed. These techniques can be described according to the degree of user interaction involved (e.g., autonomous systems, interactive exploratory systems, query-driven systems) or the methods of data analysis employed (e.g., database-oriented or data warehouse–oriented techniques, machine learning, statistics, visualization, pattern recognition, neural networks, and so on). A sophisticated data mining system will often adopt multiple data mining techniques or work out an effective, integrated technique that combines the merits of a few individual approaches.

Classification according to the applications adapted: Data mining systems can also be categorized according to the applications they adapt. For example, data mining systems may be tailored specifically for finance, telecommunications, DNA, stock markets, e-mail, and so on. Different applications often require the integration of application-specific methods. Therefore, a generic, all-purpose data mining system may not fit domain-specific mining tasks.

Alternate Answer

Data mining is classified with many techniques. Such as statistics, machine learning, pattern recognition, database and data warehouse systems, information retrieval, visualization, algorithms, high performance computing, and many application domains (Shown in Figure). Data mining system can be categorized according to various criteria.



1) Statistics: A statistical model is a set of mathematical functions that describe the behavior of the objects in a target class in terms of random variables and their associated probability distributions. Statistical models are widely used to model data and data classes. For example, in data mining tasks like data characterization and classification, statistical models of target classes can be built.

2) Machine Learning: Machine learning investigates how computers can learn (or improve their performance) based on data. A main research area is

for computer programs to automatically learn to recognize complex patterns and make intelligent decisions based on data. Machine learning is a fast-growing discipline.

i. **Supervised learning** is basically a synonym for classification. The supervision in the learning comes from the labeled examples in the training data set. For example, in the postal code recognition problem, a set of handwritten postal code images and their corresponding machine-readable translations are used as the training examples, which supervise the learning of the classification model.
ii. **Unsupervised learning** is essentially a synonym for clustering. The learning process is unsupervised since the input examples are not class labeled. For example, an unsupervised learning method can take, as input, a set of images of handwritten digits. Suppose that it finds 10 clusters of data. These clusters may correspond to the 10 distinct digits of 0 to 9, respectively.
iii. **Semi-supervised** learning is a class of machine learning techniques that make use of both labeled and unlabeled examples when learning a model. For a two-class problem, one class as the positive examples and the other class as the negative examples.
iv. **Active learning** is a machine learning approach that lets users play an active role in the learning process.

3) Database systems can focuses on the creation, maintenance, and use of databases for organizations and end-users. Particularly, database systems principles in data models, query languages, query processing and optimization methods, data storage, and indexing and accessing methods. Many data mining tasks need to handle large data sets or even real-time, fast streaming data. Recent database systems have built systematic data analysis capabilities on database data using data warehousing and data mining facilities.

4) Data warehouse integrates data from multiple sources and various timeframes. It provides OLAP facilities in multidimensional databases to

promote multidimensional data mining. It maintain recent data, previous data and historical data in database.

5) Information Retrieval (IR) is the science of searching for documents or information in documents. The typical approaches in information retrieval adopt probabilistic models. For example, a text document can be observing as a container of words, that is, a multi set of words appearing in the document.

6) Pattern recognition is the process of recognizing patterns by using machine learning algorithm. Pattern recognition can be defined as the classification of data based on knowledge already gained or on statistical information extracted from patterns and/or their representation. One of the important aspects of the pattern recognition is its application potential.

7) Data visualization is a general term that describes any effort to help people understand the significance of data by placing it in a visual context. Patterns, trends and correlations that might go undetected in text-based data can be exposed and recognized easier with data visualization software.

8) An algorithm in data mining (or machine learning) is a set of heuristics and calculations that creates a model from data. To create a model, the algorithm first analyzes the data you provide, looking for specific types of patterns or trends.

9) High Performance Computing (HPC) framework which can abstract the increased complexity in current computing systems and at the same time provide performance benefits by exploiting multiple forms of parallelism in Data Mining algorithms.

10) Data Mining Applications: The list of areas where data mining is widely used – Financial Data Analysis, Retail Industry, Telecommunication Industry, Biological Data Analysis, Other Scientific Applications, Intrusion Detection.

DATA MINING TASK PRIMITIVES

Each user will have a data mining task in mind, that is, some form of data analysis that he or she would like to have performed. A data mining task can be specified in the form of a data mining query, which is input to the data mining system. A data mining query is defined in terms of data mining task

primitives. These primitives allow the user to interactively communicate with the data mining system during discovery in order to direct the mining process, or examine the findings from different angles or depths. The data mining primitives specify the following, as illustrated in Figure 1.13.

1) **The set of task-relevant data to be mined:** This specifies the portions of the database or the set of data in which the user is interested. This includes the database attributes or data warehouse dimensions of interest (referred to as the relevant attributes or dimensions).
2) **The kind of knowledge to be mined:** This specifies the data mining functions to be performed, such as characterization, discrimination, association or correlation analysis, classification, prediction, clustering, outlier analysis, or evolution analysis.
3) **The background knowledge to be used in the discovery process:** This knowledge about the domain to be mined is useful for guiding the knowledge discovery process and for evaluating the patterns found. Concept hierarchies are a popular form of background knowledge, which allow data to be mined at multiple levels of abstraction.
   An example of a concept hierarchy for the attribute (or dimension) age is shown in Figure 1.14. User beliefs regarding relationships in the data are another form of background knowledge.
4) **The interestingness measures and thresholds for pattern evaluation:** They may be used to guide the mining process or, after discovery, to evaluate the discovered patterns. Different kinds of knowledge may have different interestingness measures. For example, interestingness measures for association rules include support and confidence. Rules whose support and confidence values are below user-specified thresholds are considered uninteresting.
5) **The expected representation for visualizing the discovered patterns:** This refers to the form in which discovered patterns are to be displayed, which may include rules, tables, charts, graphs, decision trees, and cubes.

A data mining query language can be designed to incorporate these primitives, allowing users to flexibly interact with data mining systems. Having a data mining query language provides a foundation on which user-friendly graphical interfaces can be built.

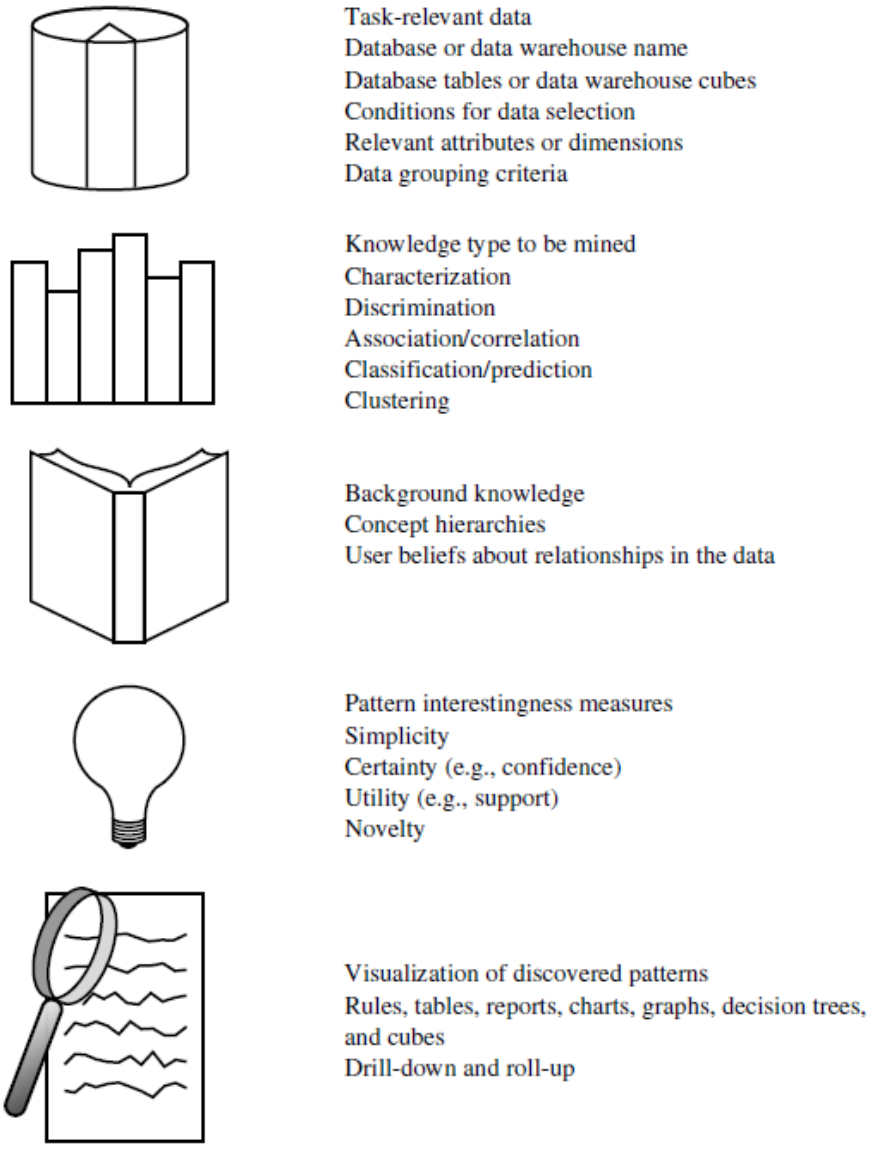


Figure 1.13 Primitives for specifying a data mining task.

This facilitates a data mining system's communication with other information systems and its integration with the overall information processing environment. Designing a comprehensive data mining language is challenging because data mining covers a wide spectrum of tasks, from data characterization to evolution analysis. Each task has different requirements. The design of an effective data mining query language requires a deep

understanding of the power, limitation, and underlying mechanisms of the various kinds of data mining tasks.

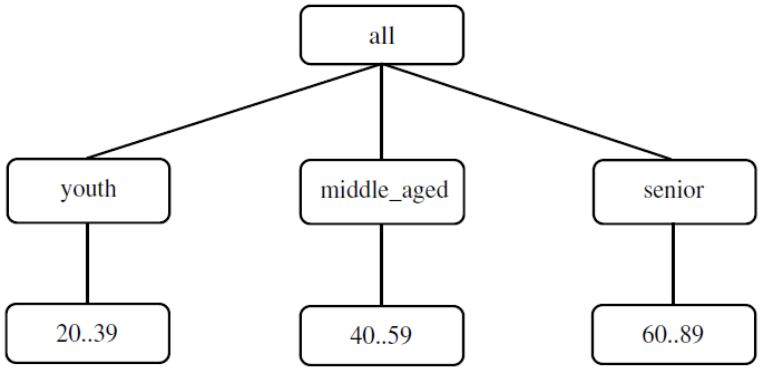


Figure 1.14 A concept hierarchy for the attribute (or dimension) *age*. The root node represents the most general abstraction level, denoted as all.

INTEGRATION OF A DATA MINING SYSTEM WITH A DATABASE OR DATA WAREHOUSE SYSTEM

A critical question in the design of a data mining (DM) system is how to integrate or couple the DM system with a database (DB) system and/or a data warehouse (DW) system. If a DM system works as a stand-alone system or is embedded in an application program, there are no DB or DW systems with which it has to communicate. This simple scheme is called no coupling, where the main focus of the DM design rests on developing effective and efficient algorithms for mining the available data sets. However, when a DM system works in an environment that requires it to communicate with other information system components, such as DB and DW systems, possible integration schemes include no coupling, loose coupling, semi-tight coupling, and tight coupling. We examine each of these schemes, as follows:

1) **No coupling:** No coupling means that a DM system will not utilize any function of a DB or DW system. It may fetch data from a particular source (such as a file system), process data using some data mining algorithms, and then store the mining results in another file.

   Such a system, though simple, suffers from several drawbacks. First, a DB system provides a great deal of flexibility and efficiency at storing, organizing, accessing, and processing data. Without using a DB/DW system, a DM system may spend a substantial amount of time finding,

collecting, cleaning, and transforming data. In DB and/or DW systems, data tend to be well organized, indexed, cleaned, integrated, or consolidated, so that finding the task-relevant, high-quality data becomes an easy task. Second, there are many tested, scalable algorithms and data structures implemented in DB and DW systems. It is feasible to realize efficient, scalable implementations using such systems. Moreover, most data have been or will be stored in DB/DW systems. Without any coupling of such systems, a DM system will need to use other tools to extract data, making it difficult to integrate such a system into an information processing environment. Thus, no coupling represents a poor design.

2) **Loose coupling:** Loose coupling means that a DM system will use some facilities of a DB or DW system, fetching data from a data repository managed by these systems, performing data mining, and then storing the mining results either in a file or in a designated place in a database or data warehouse.

   Loose coupling is better than no coupling because it can fetch any portion of data stored in databases or data warehouses by using query processing, indexing, and other system facilities. It incurs some advantages of the flexibility, efficiency, and other features provided by such systems. However, many loosely coupled mining systems are main memory-based. Because mining does not explore data structures and query optimization methods provided by DB or DW systems, it is difficult for loose coupling to achieve high scalability and good performance with large data sets.

3) **Semi-tight coupling:** Semi-tight coupling means that besides linking a DM system to a DB/DW system, efficient implementations of a few essential data mining primitives (identified by the analysis of frequently encountered data mining functions) can be provided in the DB/DW system. These primitives can include sorting, indexing, aggregation, histogram analysis, multiway join, and precomputation of some essential statistical measures, such as sum, count, max, min, standard deviation, and so on. Moreover, some frequently used intermediate

mining results can be precomputed and stored in the DB/DW system. Because these intermediate mining results are either precomputed or can be computed efficiently, this design will enhance the performance of a DM system.

4) **Tight coupling:** Tight coupling means that a DM system is smoothly integrated into the DB/DW system. The data mining subsystem is treated as one functional component of an information system. Data mining queries and functions are optimized based on mining query analysis, data structures, indexing schemes, and query processing methods of a DB or DW system. With further technology advances, DM, DB, and DW systems will evolve and integrate together as one information system with multiple functionalities. This will provide a uniform information processing environment.

   This approach is highly desirable because it facilitates efficient implementations of data mining functions, high system performance, and an integrated information processing environment.

With this analysis, it is easy to see that a data mining system should be coupled with a DB/DW system. Loose coupling, though not efficient, is better than no coupling because it uses both data and system facilities of a DB/DW system. Tight coupling is highly desirable, but its implementation is nontrivial and more research is needed in this area. Semi-tight coupling is a compromise between loose and tight coupling. It is important to identify commonly used data mining primitives and provide efficient implementations of such primitives in DB or DW systems.

MAJOR ISSUES IN DATA MINING

The scope of this book addresses major issues in data mining regarding mining methodology, user interaction, performance, and diverse data types. These issues are introduced below:

1) Mining methodology and user interaction issues: These reflect the kinds of knowledge mined, the ability to mine knowledge at multiple granularities, the use of domain knowledge, ad hoc mining, and knowledge visualization.

i. **Mining different kinds of knowledge in databases:** Because different users can be interested in different kinds of knowledge, data mining should cover a wide spectrum of data analysis and knowledge discovery tasks, including data characterization, discrimination, association and correlation analysis, classification, prediction, clustering, outlier analysis, and evolution analysis (which includes trend and similarity analysis). These tasks may use the same database in different ways and require the development of numerous data mining techniques.

ii. **Interactive mining of knowledge at multiple levels of abstraction:** Because it is difficult to know exactly what can be discovered within a database, the data mining process should be interactive. For databases containing a huge amount of data, appropriate sampling techniques can first be applied to facilitate interactive data exploration. Interactive mining allows users to focus the search for patterns, providing and refining data mining requests based on returned results. Specifically, knowledge should be mined by drilling down, rolling up, and pivoting through the data space and knowledge space interactively, similar to what OLAP can do on data cubes. In this way, the user can interact with the data mining system to view data and discovered patterns at multiple granularities and from different angles.

iii. **Incorporation of background knowledge:** Background knowledge, or information regarding the domain under study, may be used to guide the discovery process and allow discovered patterns to be expressed in concise terms and at different levels of abstraction. Domain knowledge related to databases, such as integrity constraints and deduction rules, can help focus and speed up a data mining process, or judge the interestingness of discovered patterns.

iv. **Data mining query languages and ad hoc data mining:** Relational query languages (such as SQL) allow users to pose ad hoc queries for data retrieval. In a similar vein, high-level data mining query languages need to be developed to allow users to describe ad hoc data mining tasks by facilitating the specification of the relevant sets of data for analysis, the domain knowledge, the kinds of knowledge to be mined, and the

conditions and constraints to be enforced on the discovered patterns. Such a language should be integrated with a database or data warehouse query language and optimized for efficient and flexible data mining.

v. **Presentation and visualization of data mining results:** Discovered knowledge should be expressed in high-level languages, visual representations, or other expressive forms so that the knowledge can be easily understood and directly usable by humans. This is especially crucial if the data mining system is to be interactive. This requires the system to adopt expressive knowledge representation techniques, such as trees, tables, rules, graphs, charts, crosstabs, matrices, or curves.

vi. **Handling noisy or incomplete data:** The data stored in a database may reflect noise, exceptional cases, or incomplete data objects. When mining data regularities, these objects may confuse the process, causing the knowledge model constructed to overfit the data. As a result, the accuracy of the discovered patterns can be poor. Data cleaning methods and data analysis methods that can handle noise are required, as well as outlier mining methods for the discovery and analysis of exceptional cases.

vii. **Pattern evaluation—the interestingness problem:** A data mining system can uncover thousands of patterns. Many of the patterns discovered may be uninteresting to the given user, either because they represent common knowledge or lack novelty. Several challenges remain regarding the development of techniques to assess the interestingness of discovered patterns, particularly with regard to subjective measures that estimate the value of patterns with respect to a given user class, based on user beliefs or expectations. The use of interestingness measures or user-specified constraints to guide the discovery process and reduce the search space is another active area of research.

2) Performance issues: These include efficiency, scalability, and parallelization of data mining algorithms.

i. **Efficiency and scalability of data mining algorithms:** To effectively extract information from a huge amount of data in databases, data mining algorithms must be efficient and scalable. In other words, the running time of a data mining algorithm must be predictable and acceptable in large databases. From a database perspective on knowledge discovery, efficiency and scalability are key issues in the implementation of data mining systems. Many of the issues discussed above under mining methodology and user interaction must also consider efficiency and scalability.

ii. **Parallel, distributed, and incremental mining algorithms:** The huge size of many databases, the wide distribution of data, and the computational complexity of some data mining methods are factors motivating the development of parallel and distributed data mining algorithms. Such algorithms divide the data into partitions, which are processed in parallel. The results from the partitions are then merged. Moreover, the high cost of some data mining processes promotes the need for incremental data mining algorithms that incorporate database updates without having to mine the entire data again "from scratch." Such algorithms perform knowledge modification incrementally to amend and strengthen what was previously discovered.

3) Issues relating to the diversity of database types:

i. **Handling of relational and complex types of data:** Because relational databases and data warehouses are widely used, the development of efficient and effective data mining systems for such data is important. However, other databases may contain complex data objects, hypertext and multimedia data, spatial data, temporal data, or transaction data. It is unrealistic to expect one system to mine all kinds of data, given the diversity of data types and different goals of data mining. Specific data mining systems should be constructed for mining specific kinds of data. Therefore, one may expect to have different data mining systems for different kinds of data.

ii. **Mining information from heterogeneous databases and global information systems:** Local- and wide-area computer networks (such as the Internet) connect many sources of data, forming huge, distributed, and heterogeneous databases. The discovery of knowledge from different sources of structured, semi structured, or unstructured data with diverse data semantics poses great challenges to data mining. Data mining may help disclose high-level data regularities in multiple heterogeneous databases that are unlikely to be discovered by simple query systems and may improve information exchange and interoperability in heterogeneous databases. Web mining, which uncovers interesting knowledge about Web contents, Web structures, Web usage, and Web dynamics, becomes a very challenging and fast-evolving field in data mining.

The above issues are considered major requirements and challenges for the further evolution of data mining technology. Some of the challenges have been addressed in recent data mining research and development, to a certain extent, and are now considered requirements, while others are still at the research stage. The issues, however, continue to stimulate further investigation and improvement.

Alternate Answer

Data mining is a dynamic and fast-expanding field with great strengths. Major issues in data mining research, partitioning them into five groups: mining methodology, user interaction, efficiency and scalability, diversity of data types, and data mining and society.

1) **Mining methodology:** In this methodology the user interaction on different issues such as:
   i. Mining various and new kinds of knowledge.
   ii. Mining knowledge in multidimensional space.
   iii. Data mining—an interdisciplinary effort.
   iv. Boosting the power of discovery in a networked environment.
   v. Handling uncertainty, noise, or incompleteness of data.
   vi. Pattern evaluation and pattern or constraint-guided mining.

2) **User Interaction:** Interesting areas of research include how to interact with a data mining system, how to incorporate a user's background knowledge in mining, and how to visualize and comprehend data mining results.
   i. Interactive mining.
   ii. Incorporation of background knowledge.
   iii. Ad hoc data mining and data mining query language.
   iv. Presentation and visualization of data mining results.
3) **Efficiency and Scalability:**
   i. Efficiency and scalability of data mining algorithms.
   ii. Parallel, distributed, and incremental mining algorithms.
   iii. Cloud computing and cluster computing.
4) **Diversity of Database Types:**
   i. Handling complex types of data.
   ii. Mining dynamic, networked, and global data repositories.
5) **Data Mining and Society:**
   i. Social impacts of data mining.
   ii. Privacy-preserving data mining.
   iii. Invisible data mining.

DATA PRE-PROCESSING

Data pre-processing is a data mining technique that involves transforming raw data into an understandable format. Real-world data is often incomplete, inconsistent, and/or lacking in certain behaviors or trends, and is likely to contain many errors. Data pre-processing is a proven method of resolving such issues. Data pre-processing prepares raw data for further processing.

Data pre-processing is used database-driven applications such as customer relationship management and rule-based applications (like neural networks).

Data Pre-processing Techniques: Data goes through a series of steps during pre-processing:

1) **Data Cleaning:** Data is cleansed through processes such as filling in missing values, smoothing the noisy data, or resolving the inconsistencies in the data.
2) **Data Integration:** Data with different representations are put together and conflicts within the data are resolved.
3) **Data Transformation:** Data is normalized, aggregated and generalized.
4) **Data Reduction:** This step aims to present a reduced representation of the data in a data warehouse.
5) **Data Discretization:** Involves the reduction of a number of values of a continuous attribute by dividing the range of attribute intervals.

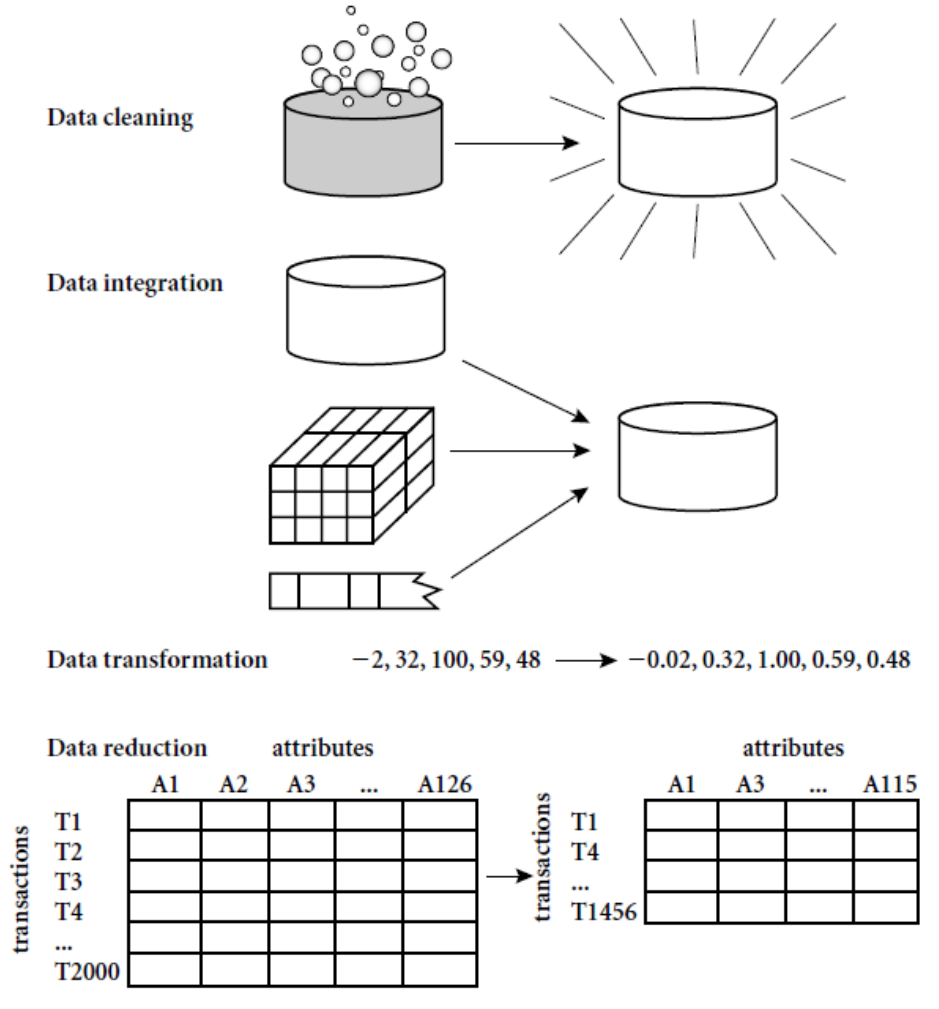


Figure 2.1 Forms of data preprocessing.

Need for pre-processing:

1) Incomplete, noisy and inconsistent data are common place properties of large real world databases and data warehouses.
2) Incomplete data can occur for a number of reasons:
    i. Attributes of interest may not always be available.
    ii. Relevant data may not be recorded due to misunderstanding, or because of equipment malfunctions.
    iii. Data that were inconsistent with other recorded data may have been deleted.
    iv. Missing data, particularly for tuples with missing values for some attributes, may need to be inferred.
    v. The data collection instruments used may be faulty.
    vi. There may have been human or computer errors occurring at data entry.
    vii. Errors in data transmission can also occur.
    viii. There may be technology limitations, such as limited buffer size for coordinating synchronized data transfer and consumption.
    ix. Data cleaning routines work to clean the data by filling in missing values, smoothing noisy data, identifying or removing outliers, and resolving inconsistencies.
    x. Data integration is the process of integrating multiple databases cubes or files. Yet some attributes representing a given may have different names in different databases, causing inconsistencies and redundancies.
    xi. Data transformation is a kind of operations, such as normalization and aggregation, are additional data pre-processing procedures that would contribute toward the success of the mining process.
    xii. Data reduction obtains a reduced representation of data set that is much smaller in volume, yet produces the same(or almost the same) analytical results.

*****

UNIT – II

ASSOCIATION RULE MINING

MINING FREQUENT PATTERNS

Frequent patterns are patterns (such as itemsets, sub-sequences, or substructures) that appear in a data set frequently. For example, a set of items, such as milk and bread, that appear frequently together in a transaction data set is a frequent itemset. A subsequence, such as buying first a PC, then a digital camera, and then a memory card, if it occurs frequently in a shopping history database, is a (frequent) sequential pattern.

A substructure can refer to different structural forms, such as subgraphs, subtrees, or sublattices, which may be combined with item sets or sub-sequences. If a substructure occurs frequently, it is called a (frequent) structured pattern. Finding such frequent patterns plays an essential role in mining associations, correlations, and many other interesting relationships among data. Moreover, it helps in data classification, clustering, and other data mining tasks as well. Thus, frequent pattern mining has become an important data mining task and a focused theme in data mining research.

Basic Concepts:

Market Basket Analysis: Frequent itemset mining leads to the discovery of associations and correlations among items in large transactional or relational data sets. With massive amounts of data continuously being collected and stored, many industries are becoming interested in mining such patterns from their databases. The discovery of interesting correlation relationships among huge amounts of business transaction records can help in many business decision-making processes, such as catalog design, cross-marketing, and customer shopping behavior analysis.

A typical example of frequent itemset mining is market basket analysis. This process analyzes customer buying habits by finding associations between the different items that customers place in their “shopping baskets” (Figure 5.1). The discovery of such associations can help retailers develop marketing strategies by gaining insight into which items are frequently purchased

together by customers. For instance, if customers are buying milk, how likely are they to also buy bread (and what kind of bread) on the same trip to the supermarket? Such information can lead to increased sales by helping retailers do selective marketing and plan their shelf space.

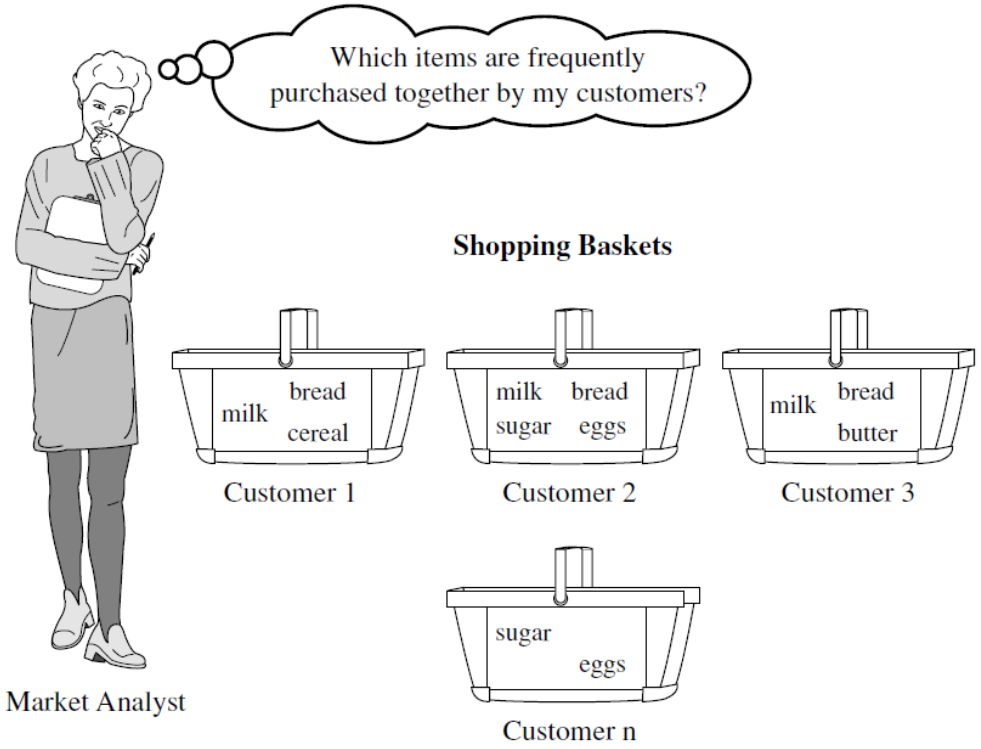


Figure 5.1 Market basket analysis.

If we think of the universe as the set of items available at the store, then each item has a Boolean variable representing the presence or absence of that item. Each basket can then be represented by a Boolean vector of values assigned to these variables. The Boolean vectors can be analyzed for buying patterns that reflect items that are frequently associated or purchased together. These patterns can be represented in the form of association rules. For example, the information that customers who purchase computers also tend to buy antivirus software at the same time is represented in Association Rule (5.1) below:

$$computer \Rightarrow antivirus\_software \ [support = 2\%, confidence = 60\%] \qquad (5.1)$$

Rule support and confidence are two measures of rule interestingness. They respectively reflect the usefulness and certainty of discovered rules. A support

of 2% for Association Rule (5.1) means that 2% of all the transactions under analysis show that computer and antivirus software are purchased together. A confidence of 60% means that 60% of the customers who purchased a computer also bought the software. Typically, association rules are considered interesting if they satisfy both a minimum support threshold and a minimum confidence threshold. Such thresholds can be set by users or domain experts. Additional analysis can be performed to uncover interesting statistical correlations between associated items.

Frequent Itemsets, Closed Itemsets, and Association Rules:

Let $I = \{I_1, I_2, \ldots, I_m\}$ be a set of items. Let D, the task-relevant data, be a set of database transactions where each transaction T is a set of items such that $T \subseteq I$. Each transaction is associated with an identifier, called TID. Let A be a set of items. A transaction T is said to contain A if and only if $A \subseteq T$. An association rule is an implication of the form $A \Rightarrow B$, where $A \subset I$, $B \subset I$, and $A \cap B = \phi$. The rule $A \Rightarrow B$ holds in the transaction set D with **support** s, where s is the percentage of transactions in D that contain $A \cup B$ (i.e., the *union* of sets A and B, or say, both A and B). This is taken to be the probability, $P(A \cup B)$.[1] The rule $A \Rightarrow B$ has **confidence** c in the transaction set D, where c is the percentage of transactions in D containing A that also contain B. This is taken to be the conditional probability, $P(B|A)$. That is,

$$support(A \Rightarrow B) = P(A \cup B) \quad (5.2)$$

$$confidence(A \Rightarrow B) = P(B|A). \quad (5.3)$$

Rules that satisfy both a minimum support threshold (*min_sup*) and a minimum confidence threshold (*min_conf*) are called **strong**. By convention, we write support and confidence values so as to occur between 0% and 100%, rather than 0 to 1.0.

A set of items is referred to as an **itemset**.[2] An itemset that contains k items is a k-**itemset**. The set {*computer, antivirus_software*} is a 2-itemset. The **occurrence frequency of an itemset** is the number of transactions that contain the itemset. This is also known, simply, as the **frequency**, **support count**, or **count** of the itemset. Note that the itemset support defined in Equation (5.2) is sometimes referred to as *relative support*, whereas the occurrence frequency is called the **absolute support**. If the relative support of an itemset I satisfies a prespecified **minimum support threshold** (i.e., the absolute support of I satisfies the corresponding **minimum support count threshold**), then I is a **frequent** itemset.[3] The set of frequent k-itemsets is commonly denoted by L_k.[4]

From Equation (5.3), we have

$$confidence(A \Rightarrow B) = P(B|A) = \frac{support(A \cup B)}{support(A)} = \frac{support\_count(A \cup B)}{support\_count(A)}. \quad (5.4)$$

Equation (5.4) shows that the confidence of rule $A \Rightarrow B$ can be easily derived from the support counts of A and $A \cup B$. That is, once the support counts of A, B, and $A \cup B$ are

found, it is straightforward to derive the corresponding association rules $A \Rightarrow B$ and $B \Rightarrow A$ and check whether they are strong. Thus the problem of mining association rules can be reduced to that of mining frequent itemsets.

In general, association rule mining can be viewed as a two-step process:

1. **Find all frequent itemsets:** By definition, each of these itemsets will occur at least as frequently as a predetermined minimum support count, *min_sup*.
2. **Generate strong association rules from the frequent itemsets:** By definition, these rules must satisfy minimum support and minimum confidence.

Additional interestingness measures can be applied for the discovery of correlation relationships between associated items, as will be discussed in Section 5.4. Because the second step is much less costly than the first, the overall performance of mining association rules is determined by the first step.

A major challenge in mining frequent itemsets from a large data set is the fact that such mining often generates a huge number of itemsets satisfying the minimum support (*min_sup*) threshold, especially when *min_sup* is set low. This is because if an itemset is frequent, each of its subsets is frequent as well. A long itemset will contain a combinatorial number of shorter, frequent sub-itemsets. For example, a frequent itemset of length 100, such as $\{a_1, a_2, \ldots, a_{100}\}$, contains $\binom{100}{1} = 100$ frequent 1-itemsets: $a_1, a_2, \ldots, a_{100}$, $\binom{100}{2}$ frequent 2-itemsets: $(a_1, a_2), (a_1, a_3), \ldots, (a_{99}, a_{100})$, and so on. The total number of frequent itemsets that it contains is thus,

$$\binom{100}{1} + \binom{100}{2} + \cdots + \binom{100}{100} = 2^{100} - 1 \approx 1.27 \times 10^{30}. \quad (5.5)$$

This is too huge a number of itemsets for any computer to compute or store. To overcome this difficulty, we introduce the concepts of *closed frequent itemset* and *maximal frequent itemset*.

An itemset X is **closed** in a data set S if there exists no proper super-itemset[5] Y such that Y has the same support count as X in S. An itemset X is a **closed frequent itemset** in set S if X is both closed and frequent in S. An itemset X is a **maximal frequent itemset** (or **max-itemset**) in set S if X is frequent, and there exists no super-itemset Y such that $X \subset Y$ and Y is frequent in S.

Let C be the set of closed frequent itemsets for a data set S satisfying a minimum support threshold, *min_sup*. Let $\mathcal{M}$ be the set of maximal frequent itemsets for S satisfying *min_sup*. Suppose that we have the support count of each itemset in C and $\mathcal{M}$. Notice that C and its count information can be used to derive the whole set of frequent itemsets. Thus we say that C contains complete information regarding its corresponding frequent itemsets. On the other hand, $\mathcal{M}$ registers only the support of the maximal itemsets.

Frequent Pattern Mining: A Road Map: Market basket analysis is just one form of frequent pattern mining. In fact, there are many kinds of frequent patterns, association rules, and correlation relationships. Frequent pattern mining can be classified in various ways, based on the following criteria:

1) Based on the completeness of patterns to be mined: We can mine the complete set of frequent itemsets, the closed frequent itemsets, and the maximal frequent itemsets, given a minimum support threshold. We can also mine constrained frequent itemsets (i.e., those that satisfy a set of user-defined constraints), approximate frequent itemsets (i.e., those that derive only approximate support counts for the mined frequent itemsets), near-match frequent itemsets (i.e., those that tally the support count of the near or almost matching itemsets), top-k frequent itemsets (i.e., the k most frequent itemsets for a user-specified value, k), and so on.

Different applications may have different requirements regarding the completeness of the patterns to be mined, which in turn can lead to different evaluation and optimization methods.

2) Based on the levels of abstraction involved in the rule set: Some methods for association rule mining can find rules at differing levels of abstraction. For example, suppose that a set of association rules mined includes the following rules where X is a variable representing a customer:

$$buys(X, \text{“}computer\text{”}) \Rightarrow buys(X, \text{“}HP\_printer\text{”}) \qquad (5.6)$$

$$buys(X, \text{“}laptop\_computer\text{”}) \Rightarrow buys(X, \text{“}HP\_printer\text{”}) \qquad (5.7)$$

In Rules (5.6) and (5.7), the items bought are referenced at different levels of abstraction (e.g., “computer” is a higher-level abstraction of “laptop computer”). We refer to the rule set mined as consisting of multilevel

association rules. If, instead, the rules within a given set do not reference items or attributes at different levels of abstraction, then the set contains single-level association rules.

3) Based on the number of data dimensions involved in the rule: If the items or attributes in an association rule reference only one dimension, then it is a single-dimensional association rule. Note that Rule (5.1), for example, could be rewritten as Rule (5.8):

$$buys(X, \text{“}computer\text{”}) \Rightarrow buys(X, \text{“}antivirus\_software\text{”}) \qquad (5.8)$$

Rules (5.6), (5.7), and (5.8) are single-dimensional association rules because they each refer to only one dimension, buys.

If a rule references two or more dimensions, such as the dimensions age, income, and buys, then it is a multidimensional association rule. The following rule is an example of a multidimensional rule:

$$age(X, \text{“}30\ldots39\text{”}) \wedge income(X, \text{“}42K\ldots48K\text{”}) \Rightarrow buys(X, \text{“}high\ resolution\ TV\text{”}). \qquad (5.9)$$

4) Based on the types of values handled in the rule: If a rule involves associations between the presence or absence of items, it is a Boolean association rule. For example, Rules (5.1), (5.6), and (5.7) are Boolean association rules obtained from market basket analysis.

If a rule describes associations between quantitative items or attributes, then it is a quantitative association rule. In these rules, quantitative values for items or attributes are partitioned into intervals. Rule (5.9) is also considered a quantitative association rule. Note that the quantitative attributes, age and income, have been discretized.

5) Based on the kinds of rules to be mined: Frequent pattern analysis can generate various kinds of rules and other interesting relationships. Association rules are the most popular kind of rules generated from frequent patterns. Typically, such mining can generate a large number of rules, many of which are redundant or do not indicate a correlation relationship among

itemsets. Thus, the discovered associations can be further analyzed to uncover statistical correlations, leading to correlation rules.

We can also mine strong gradient relationships among itemsets, where a gradient is the ratio of the measure of an item when compared with that of its parent (a generalized itemset), its child (a specialized itemset), or its sibling (a comparable itemset). One such example is: "The average sales from Sony_Digital_Camera increase over 16% when sold together with Sony_Laptop_Computer": both Sony_Digital_Camera and Sony Laptop Computer are siblings, where the parent itemset is Sony.

6) Based on the kinds of patterns to be mined: Many kinds of frequent patterns can be mined from different kinds of data sets. For this chapter, our focus is on frequent itemset mining, that is, the mining of frequent itemsets (sets of items) from transactional or relational data sets. However, other kinds of frequent patterns can be found from other kinds of data sets. Sequential pattern mining searches for frequent sub-sequences in a sequence data set, where a sequence records an ordering of events. For example, with sequential pattern mining, we can study the order in which items are frequently purchased. For instance, customers may tend to first buy a PC, followed by a digital camera, and then a memory card. Structured pattern mining searches for frequent substructures in a structured data set. Notice that structure is a general concept that covers many different kinds of structural forms, such as graphs, lattices, trees, sequences, sets, single items, or combinations of such structures. Single items are the simplest form of structure. Each element of an itemset may contain a subsequence, a subtree, and so on, and such containment relationships can be defined recursively. Therefore, structured pattern mining can be considered as the most general form of frequent pattern mining.

MINING METHODS [OR] EFFICIENT AND SCALABLE FREQUENT ITEMSET MINING METHODS

1) The Apriori Algorithm: Finding Frequent Itemsets Using Candidate Generation: Apriori is a seminal algorithm proposed by R. Agrawal and R. Srikant in 1994 for mining frequent itemsets for Boolean association rules. The name of the algorithm is based on

the fact that the algorithm uses *prior knowledge* of frequent itemset properties, as we shall see following. Apriori employs an iterative approach known as a *level-wise* search, where k-itemsets are used to explore $(k+1)$-itemsets. First, the set of frequent 1-itemsets is found by scanning the database to accumulate the count for each item, and collecting those items that satisfy minimum support. The resulting set is denoted L_1. Next, L_1 is used to find L_2, the set of frequent 2-itemsets, which is used to find L_3, and so on, until no more frequent k-itemsets can be found. The finding of each L_k requires one full scan of the database.

To improve the efficiency of the level-wise generation of frequent itemsets, an important property called the **Apriori property**, presented below, is used to reduce the search space. We will first describe this property, and then show an example illustrating its use.

Apriori property: *All nonempty subsets of a frequent itemset must also be frequent.*

The Apriori property is based on the following observation. By definition, if an itemset I does not satisfy the minimum support threshold, *min_sup*, then I is not frequent; that is, $P(I) < min\_sup$. If an item A is added to the itemset I, then the resulting itemset (i.e., $I \cup A$) cannot occur more frequently than I. Therefore, $I \cup A$ is not frequent either; that is, $P(I \cup A) < min\_sup$.

This property belongs to a special category of properties called **antimonotone** in the sense that *if a set cannot pass a test, all of its supersets will fail the same test as well.* It is called *antimonotone* because the property is monotonic in the context of failing a test.[7]

"*How is the Apriori property used in the algorithm?*" To understand this, let us look at how L_{k-1} is used to find L_k for $k \geq 2$. A two-step process is followed, consisting of **join** and **prune** actions.

1. **The join step:** To find L_k, a set of **candidate** k-itemsets is generated by joining L_{k-1} with itself. This set of candidates is denoted C_k. Let l_1 and l_2 be itemsets in L_{k-1}. The notation $l_i[j]$ refers to the jth item in l_i (e.g., $l_1[k-2]$ refers to the second to the last item in l_1). By convention, Apriori assumes that items within a transaction or itemset are sorted in lexicographic order. For the $(k-1)$-itemset, l_i, this means that the items are sorted such that $l_i[1] < l_i[2] < \ldots < l_i[k-1]$. The join, $L_{k-1} \bowtie L_{k-1}$, is performed, where members of L_{k-1} are joinable if their first $(k-2)$ items are in common. That is, members l_1 and l_2 of L_{k-1} are joined if $(l_1[1] = l_2[1]) \wedge (l_1[2] = l_2[2]) \wedge \ldots \wedge (l_1[k-2] = l_2[k-2]) \wedge (l_1[k-1] < l_2[k-1])$. The condition $l_1[k-1] < l_2[k-1]$ simply ensures that no duplicates are generated. The resulting itemset formed by joining l_1 and l_2 is $l_1[1], l_1[2], \ldots, l_1[k-2], l_1[k-1], l_2[k-1]$.

2. The prune step: C_k is a superset of L_k, that is, its members may or may not be frequent, but all of the frequent k-itemsets are included in C_k. A scan of the database to determine the count of each candidate in C_k would result in the determination of L_k (i.e., all candidates having a count no less than the minimum support count are frequent by definition, and therefore belong to L_k). C_k, however, can be huge, and so this could

involve heavy computation. To reduce the size of C_k, the Apriori property is used as follows. Any $(k-1)$-itemset that is not frequent cannot be a subset of a frequent k-itemset. Hence, if any $(k-1)$-subset of a candidate k-itemset is not in L_{k-1}, then the candidate cannot be frequent either and so can be removed from C_k. This **subset testing** can be done quickly by maintaining a hash tree of all frequent itemsets.

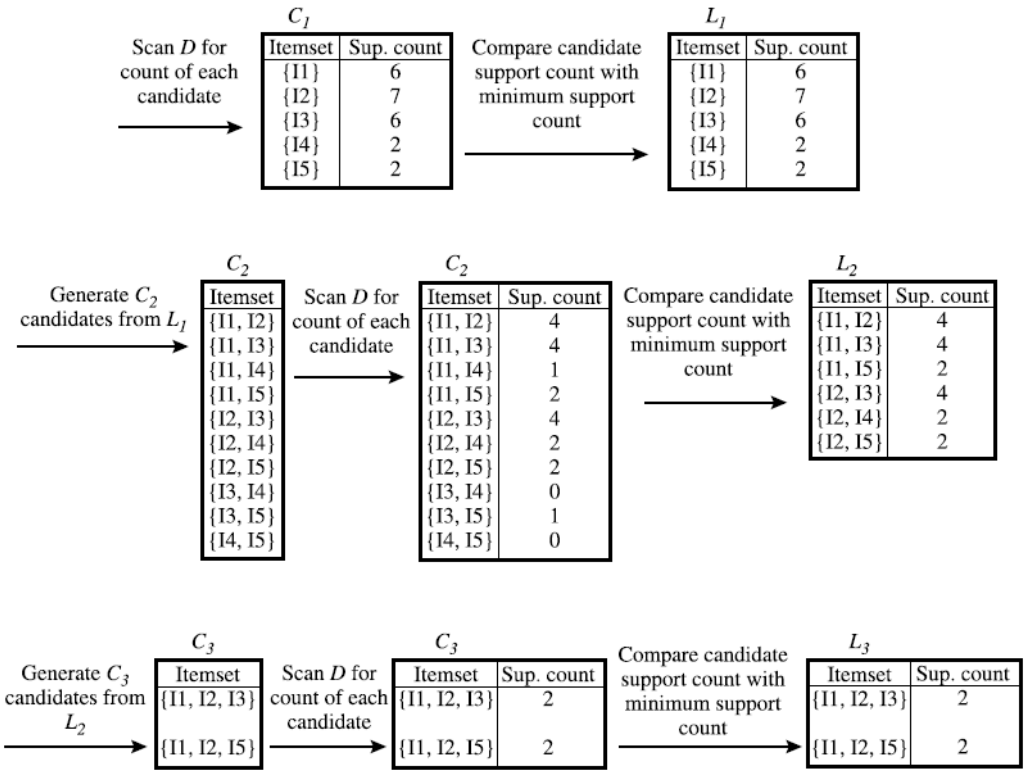


Figure 5.2 Generation of candidate itemsets and frequent itemsets, where the minimum support count is 2.

Algorithm: Apriori. Find frequent itemsets using an iterative level-wise approach based on candidate generation.

Input:

- D, a database of transactions;
- min_sup, the minimum support count threshold.

Output: L, frequent itemsets in D.

Method:

```
(1)   L1 = find_frequent_1-itemsets(D);
(2)   for (k = 2; Lk−1 ≠ ϕ; k++) {
(3)       Ck = apriori_gen(Lk−1);
(4)       for each transaction t ∈ D { // scan D for counts
(5)           Ct = subset(Ck, t); // get the subsets of t that are candidates
(6)           for each candidate c ∈ Ct
(7)               c.count++;
(8)       }
(9)       Lk = {c ∈ Ck | c.count ≥ min_sup}
(10)  }
(11)  return L = ∪k Lk;

procedure apriori_gen(Lk−1:frequent (k − 1)-itemsets)
(1)   for each itemset l1 ∈ Lk−1
(2)       for each itemset l2 ∈ Lk−1
(3)           if (l1[1] = l2[1]) ∧ (l1[2] = l2[2]) ∧ ... ∧ (l1[k − 2] = l2[k − 2]) ∧ (l1[k − 1] < l2[k − 1]) then {
(4)               c = l1 ⋈ l2; // join step: generate candidates
(5)               if has_infrequent_subset(c, Lk−1) then
(6)                   delete c; // prune step: remove unfruitful candidate
(7)               else add c to Ck;
(8)           }
(9)   return Ck;

procedure has_infrequent_subset(c: candidate k-itemset;
          Lk−1: frequent (k − 1)-itemsets); // use prior knowledge
(1)   for each (k − 1)-subset s of c
(2)       if s ∉ Lk−1 then
(3)           return TRUE;
(4)   return FALSE;
```

Figure 5.4 The Apriori algorithm for discovering frequent itemsets for mining Boolean association rules.

2) Generating Association Rules from Frequent Itemsets: Once the frequent itemsets from transactions in a database D have been found, it is straightforward to generate strong association rules from them (where strong association rules satisfy both minimum support and minimum confidence). This can be done using Equation (5.4) for confidence, which we show again here for completeness:

$$confidence(A \Rightarrow B) = P(B|A) = \frac{support\_count(A \cup B)}{support\_count(A)}.$$

The conditional probability is expressed in terms of itemset support count, where $support\_count(A \cup B)$ is the number of transactions containing the itemsets $A \cup B$, and $support\_count(A)$ is the number of transactions containing the itemset A. Based on this equation, association rules can be generated as follows:

- For each frequent itemset l, generate all nonempty subsets of l.
- For every nonempty subset s of l, output the rule "$s \Rightarrow (l - s)$" if $\frac{support\_count(l)}{support\_count(s)} \geq min\_conf$, where min_conf is the minimum confidence threshold.

Because the rules are generated from frequent itemsets, each one automatically satisfies minimum support. Frequent itemsets can be stored ahead of time in hash tables along with their counts so that they can be accessed quickly.

3) Improving the Efficiency of Apriori: "How can we further improve the efficiency of Apriori-based mining?" Many variations of the Apriori algorithm have been proposed that focus on improving the efficiency of the original algorithm. Several of these variations are summarized as follows:

Hash-based technique (hashing itemsets into corresponding buckets): A hash-based technique can be used to reduce the size of the candidate k-itemsets, C_k, for $k > 1$. For example, when scanning each transaction in the database to generate the frequent 1-itemsets, L_1, from the candidate 1-itemsets in C_1, we can generate all of the 2-itemsets for each transaction, hash (i.e., map) them into the different *buckets* of a *hash table* structure, and increase the corresponding bucket counts (Figure 5.5). A 2-itemset whose corresponding bucket count in the hash table is below the support

threshold cannot be frequent and thus should be removed from the candidate set. Such a hash-based technique may substantially reduce the number of the candidate k-itemsets examined (especially when $k = 2$).

Create hash table H_2 using hash function $h(x, y) = ((order\ of\ x) \times 10 + (order\ of\ y))\ mod\ 7$ →

H_2

bucket address	0	1	2	3	4	5	6
bucket count	2	2	4	2	2	4	4
bucket contents	{I1, I4} {I3, I5}	{I1, I5} {I1, I5}	{I2, I3} {I2, I3} {I2, I3} {I2, I3}	{I2, I4} {I2, I4}	{I2, I5} {I2, I5}	{I1, I2} {I1, I2} {I1, I2} {I1, I2}	{I1, I3} {I1, I3} {I1, I3} {I1, I3}

Figure 5.5 Hash table, H_2, for candidate 2-itemsets: This hash table was generated by scanning the transactions of Table 5.1 while determining L_1 from C_1. If the minimum support count is, say, 3, then the itemsets in buckets 0, 1, 3, and 4 cannot be frequent and so they should not be included in C_2.

Transaction reduction (reducing the number of transactions scanned in future iterations): A transaction that does not contain any frequent k-itemsets cannot contain any frequent $(k+1)$-itemsets. Therefore, such a transaction can be marked or removed from further consideration because subsequent scans of the database for j-itemsets, where $j > k$, will not require it.

Partitioning (partitioning the data to find candidate itemsets): A partitioning technique can be used that requires just two database scans to mine the frequent itemsets (Figure 5.6). It consists of two phases. In Phase I, the algorithm subdivides the transactions of D into n nonoverlapping partitions. If the minimum support threshold for transactions in D is *min_sup*, then the minimum support count for a partition is *min_sup* $\times$ *the number of transactions in that partition*. For each partition, all frequent itemsets within the partition are found. These are referred to as **local frequent itemsets**. The procedure employs a special data structure that, for each itemset, records the TIDs of the transactions containing the items in the itemset. This allows it to find all of the local frequent k-itemsets, for $k = 1, 2, \ldots$, in just one scan of the database.

A local frequent itemset may or may not be frequent with respect to the entire database, D. *Any itemset that is potentially frequent with respect to D must occur as a frequent itemset in at least one of the partitions.* Therefore, all local frequent itemsets are candidate itemsets with respect to D. The collection of frequent itemsets from all partitions forms the **global candidate itemsets** with respect to D. In Phase II, a second scan of D is conducted in which the actual support of each candidate is assessed in order to determine the global frequent itemsets. Partition size and the number of partitions are set so that each partition can fit into main memory and therefore be read only once in each phase.

Sampling (mining on a subset of the given data): The basic idea of the sampling approach is to pick a random sample S of the given data D, and then search for frequent itemsets in S instead of D. In this way, we trade off some degree of accuracy against efficiency. The sample size of S is such that the search for frequent itemsets in S can be done in main memory, and so only one scan of the transactions in S is required overall. Because we are searching for frequent itemsets in S rather than in D, it is possible that we will miss some of the global frequent itemsets. To lessen this possibility, we use a lower support threshold than minimum support to find the frequent itemsets local to S (denoted L^S). The rest of the database is then used to compute the actual frequencies of each itemset in L^S. A mechanism is used to determine whether all of the global frequent itemsets are included in L^S. If L^S actually contains all of the frequent itemsets in D, then only one scan of D is required. Otherwise, a second pass can be done in order to find the frequent itemsets that were missed in the first pass. The sampling approach is especially beneficial when efficiency is of utmost importance, such as in computationally intensive applications that must be run frequently.

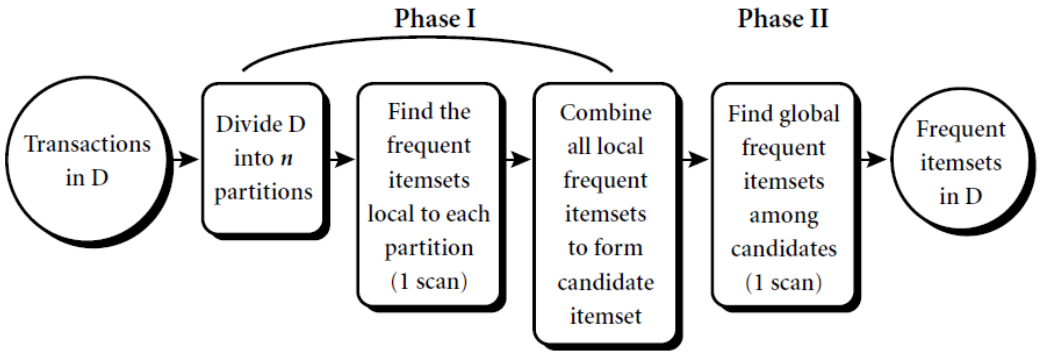


Figure 5.6 Mining by partitioning the data.

Dynamic itemset counting (adding candidate itemsets at different points during a scan): A dynamic itemset counting technique was proposed in which the database is partitioned into blocks marked by start points. In this variation, new candidate itemsets can be added at any start point, unlike in Apriori, which determines new candidate itemsets only immediately before each complete database scan. The technique is dynamic in that it estimates the support of all of the itemsets that have been counted so far, adding new candidate itemsets if all of their subsets are estimated to be frequent. The resulting algorithm requires fewer database scans than Apriori.

4) Mining Frequent Itemsets without Candidate Generation: As we have seen, in many cases the Apriori candidate generate-and-test method significantly reduces the size of candidate sets, leading to good performance gain. However, it can suffer from two nontrivial costs:

- *It may need to generate a huge number of candidate sets.* For example, if there are 10^4 frequent 1-itemsets, the Apriori algorithm will need to generate more than 10^7 candidate 2-itemsets. Moreover, to discover a frequent pattern of size 100, such as $\{a_1, \ldots, a_{100}\}$, it has to generate at least $2^{100} - 1 \approx 10^{30}$ candidates in total.
- *It may need to repeatedly scan the database and check a large set of candidates by pattern matching.* It is costly to go over each transaction in the database to determine the support of the candidate itemsets.

"Can we design a method that mines the complete set of frequent itemsets without candidate generation?" An interesting method in this attempt is called frequent-pattern growth, or simply FP-growth, which adopts a divide-and-conquer strategy as follows. First, it compresses the database representing frequent items into a frequent-pattern tree, or FP-tree, which retains the itemset association information. It then divides the compressed database into

a set of conditional databases (a special kind of projected database), each associated with one frequent item or "pattern fragment," and mines each such database separately.

The FP-growth method transforms the problem of finding long frequent patterns to searching for shorter ones recursively and then concatenating the suffix. It uses the least frequent items as a suffix, offering good selectivity. The method substantially reduces the search costs.

When the database is large, it is sometimes unrealistic to construct a main memory based FP-tree. An interesting alternative is to first partition the database into a set of projected databases, and then construct an FP-tree and mine it in each projected database. Such a process can be recursively applied to any projected database if its FP-tree still cannot fit in main memory.

A study on the performance of the FP-growth method shows that it is efficient and scalable for mining both long and short frequent patterns, and is about an order of magnitude faster than the Apriori algorithm. It is also faster than a Tree-Projection algorithm, which recursively projects a database into a tree of projected databases.

5) Mining Frequent Itemsets Using Vertical Data Format:

Both the Apriori and FP-growth methods mine frequent patterns from a set of transactions in *TID-itemset* format (that is, $\{TID : itemset\}$), where *TID* is a transaction-id and *itemset* is the set of items bought in transaction *TID*. This data format is known as **horizontal data format**. Alternatively, data can also be presented in *item-TID_set* format (that is, $\{item : TID\_set\}$), where *item* is an item name, and *TID_set* is the set of transaction identifiers containing the item. This format is known as **vertical data format**.

In this section, we look at how frequent itemsets can also be mined efficiently using vertical data format, which is the essence of the **ECLAT** (Equivalence CLASS Transformation) algorithm developed by Zaki [Zak00].

Algorithm: FP_growth. Mine frequent itemsets using an FP-tree by pattern fragment growth.

Input:

- D, a transaction database;
- min_sup, the minimum support count threshold.

Output: The complete set of frequent patterns.

Method:

1. The FP-tree is constructed in the following steps:
   (a) Scan the transaction database D once. Collect F, the set of frequent items, and their support counts. Sort F in support count descending order as L, the *list* of frequent items.
   (b) Create the root of an FP-tree, and label it as "null." For each transaction *Trans* in D do the following. Select and sort the frequent items in *Trans* according to the order of L. Let the sorted frequent item list in *Trans* be $[p|P]$, where p is the first element and P is the remaining list. Call insert_tree$([p|P], T)$, which is performed as follows. If T has a child N such that $N.item\text{-}name = p.item\text{-}name$, then increment N's count by 1; else create a new node N, and let its count be 1, its parent link be linked to T, and its node-link to the nodes with the same *item-name* via the node-link structure. If P is nonempty, call insert_tree(P, N) recursively.
2. The FP-tree is mined by calling FP_growth$(FP_tree, null)$, which is implemented as follows.

```
procedure FP_growth(Tree, α)
(1)  if Tree contains a single path P then
(2)      for each combination (denoted as β) of the nodes in the path P
(3)          generate pattern β ∪ α with support_count = minimum support count of nodes in β;
(4)  else for each a_i in the header of Tree {
(5)      generate pattern β = a_i ∪ α with support_count = a_i.support_count;
(6)      construct β's conditional pattern base and then β's conditional FP_tree Tree_β;
(7)      if Tree_β ≠ ∅ then
(8)          call FP_growth(Tree_β, β); }
```

Figure 5.9 The FP-growth algorithm for discovering frequent itemsets without candidate generation.

MINING VARIOUS KINDS OF ASSOCIATION RULES

We have studied efficient methods for mining frequent itemsets and association rules. In this section, we consider additional application requirements by extending our scope to include mining multilevel association rules, multidimensional association rules, and quantitative association rules in transactional and/or relational databases and data warehouses. Multilevel association rules involve concepts at different levels of abstraction. Multidimensional association rules involve more than one dimension or predicate (e.g., rules relating what a customer buys as well as the customer's age.) Quantitative association rules involve numeric attributes that have an implicit ordering among values (e.g., age).

Mining Multilevel Association Rules: For many applications, it is difficult to find strong associations among data items at low or primitive levels of abstraction due to the sparsity of data at those levels. Strong associations

discovered at high levels of abstraction may represent common sense knowledge. Moreover, what may represent common sense to one user may seem novel to another. Therefore, data mining systems should provide capabilities for mining association rules at multiple levels of abstraction, with sufficient flexibility for easy traversal among different abstraction spaces.

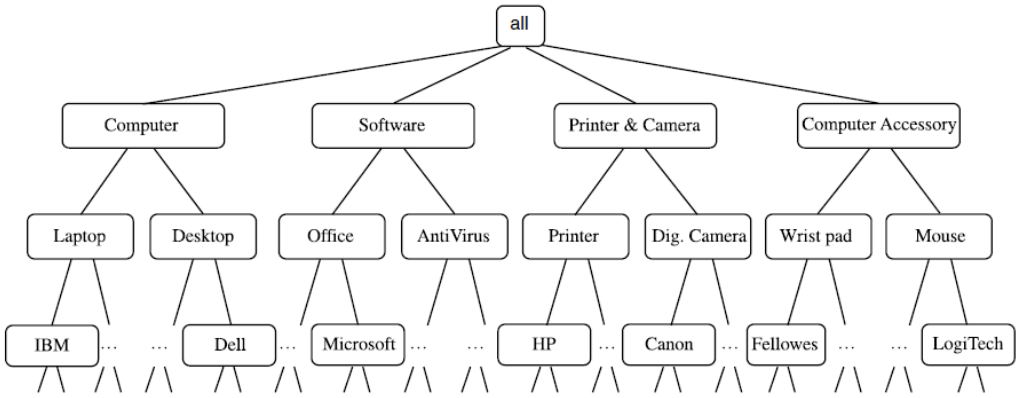


Figure 5.10 A concept hierarchy for *AllElectronics* computer items.

Association rules generated from mining data at multiple levels of abstraction are called multiple-level or multilevel association rules. Multilevel association rules can be mined efficiently using concept hierarchies under a support-confidence framework. In general, a top-down strategy is employed, where counts are accumulated for the calculation of frequent itemsets at each concept level, starting at the concept level 1 and working downward in the hierarchy toward the more specific concept levels, until no more frequent itemsets can be found. For each level, any algorithm for discovering frequent itemsets may be used, such as Apriori or its variations. A number of variations to this approach are described below, where each variation involves "playing" with the support threshold in a slightly different way. The variations are illustrated in Figures 5.11 and 5.12, where nodes indicate an item or itemset that has been examined, and nodes with thick borders indicate that an examined item or itemset is frequent.

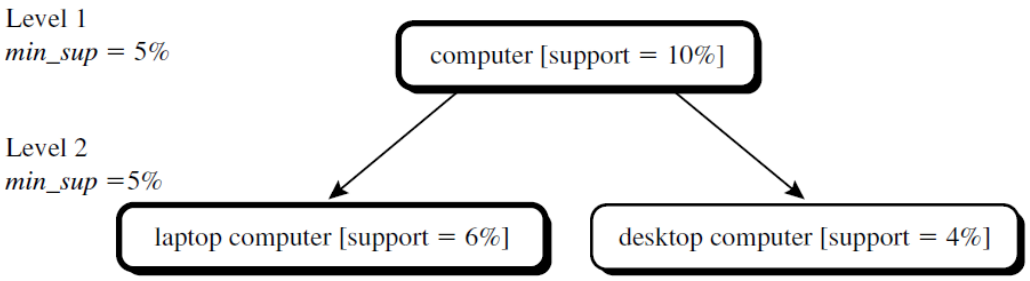


Figure 5.11 Multilevel mining with uniform support.

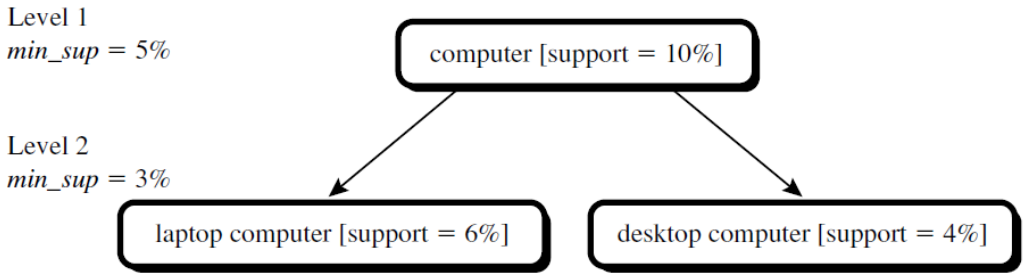


Figure 5.12 Multilevel mining with reduced support.

⇒ **Using uniform minimum support for all levels (referred to as uniform support):** The same minimum support threshold is used when mining at each level of abstraction. For example, in Figure 5.11, a minimum support threshold of 5% is used throughout (e.g., for mining from "computer" down to "laptop computer"). Both "computer" and "laptop computer" are found to be frequent, while "desktop computer" is not.

When a uniform minimum support threshold is used, the search procedure is simplified. The method is also simple in that users are required to specify only one minimum support threshold. An Apriori-like optimization technique can be adopted, based on the knowledge that an ancestor is a superset of its descendants: The search avoids examining itemsets containing any item whose ancestors do not have minimum support.

The uniform support approach, however, has some difficulties. It is unlikely that items at lower levels of abstraction will occur as frequently as those at higher levels of abstraction. If the minimum support

threshold is set too high, it could miss some meaningful associations occurring at low abstraction levels. If the threshold is set too low, it may generate many uninteresting associations occurring at high abstraction levels. This provides the motivation for the following approach.

⇒ **Using reduced minimum support at lower levels (referred to as reduced support):** Each level of abstraction has its own minimum support threshold. The deeper the level of abstraction, the smaller the corresponding threshold is. For example, in Figure 5.12, the minimum support thresholds for levels 1 and 2 are 5% and 3%, respectively. In this way, "computer," "laptop computer," and "desktop computer" are all considered frequent.

⇒ **Using item or group-based minimum support (referred to as group-based support):** Because users or experts often have insight as to which groups are more important than others, it is sometimes more desirable to set up user-specific, item, or group based minimal support thresholds when mining multilevel rules. For example, a user could set up the minimum support thresholds based on product price, or on items of interest, such as by setting particularly low support thresholds for laptop computers and flash drives in order to pay particular attention to the association patterns containing items in these categories.

Notice that the Apriori property may not always hold uniformly across all of the items when mining under reduced support and group-based support. However, efficient methods can be developed based on the extension of the property. The details are left as an exercise for interested readers.

A serious side effect of mining multilevel association rules is its generation of many redundant rules across multiple levels of abstraction due to the "ancestor" relationships among items. For example, consider the following rules where "laptop computer" is an ancestor of "IBM laptop computer" based on the concept hierarchy of Figure 5.10, and where X is a variable representing customers who purchased items in AllElectronics transactions.

$$buys(X, \text{“}laptop\ computer\text{”}) \Rightarrow buys(X, \text{“}HP\ printer\text{”})$$
$$[support = 8\%, confidence = 70\%] \quad (5.10)$$

$$buys(X, \text{“}IBM\ laptop\ computer\text{”}) \Rightarrow buys(X, \text{“}HP\ printer\text{”})$$
$$[support = 2\%, confidence = 72\%] \quad (5.11)$$

"If Rules (5.10) and (5.11) are both mined, then how useful is the latter rule?" you may wonder. *"Does it really provide any novel information?"* If the latter, less general rule does not provide new information, then it should be removed. Let's look at how this may be determined. A rule $R1$ is an **ancestor** of a rule $R2$, if $R1$ can be obtained by replacing the items in $R2$ by their ancestors in a concept hierarchy. For example, Rule (5.10) is an ancestor of Rule (5.11) because *"laptop computer"* is an ancestor of *"IBM laptop computer."* Based on this definition, a rule can be considered redundant if its support and confidence are close to their "expected" values, based on an ancestor of the rule. As an illustration, suppose that Rule (5.10) has a 70% confidence and 8% support, and that about one-quarter of all *"laptop computer"* sales are for *"IBM laptop computers."* We may expect Rule (5.11) to have a confidence of around 70% (since all data samples of *"IBM laptop computer"* are also samples of *"laptop computer"*) and a support of around 2% (i.e., $8\% \times \frac{1}{4}$). If this is indeed the case, then Rule (5.11) is not interesting because it does not offer any additional information and is less general than Rule (5.10).

Mining Multidimensional Association Rules from Relational Databases and Data Warehouses: So far in this chapter, we have studied association rules that imply a single predicate, that is, the predicate buys. For instance, in mining our AllElectronics database, we may discover the Boolean association rule

$$buys(X, \text{“}digital\ camera\text{”}) \Rightarrow buys(X, \text{“}HP\ printer\text{”}). \quad (5.12)$$

Following the terminology used in multidimensional databases, we refer to each distinct predicate in a rule as a dimension. Hence, we can refer to Rule (5.12) as a single dimensional or intra dimensional association rule because it contains a single distinct predicate (e.g., buys) with multiple occurrences (i.e., the predicate occurs more than once within the rule).

Suppose, however, that rather than using a transactional database, sales and related information are stored in a relational database or data warehouse. Such data stores are multidimensional, by definition. For instance, in

addition to keeping track of the items purchased in sales transactions, a relational database may record other attributes associated with the items, such as the quantity purchased or the price, or the branch location of the sale. Additional relational information regarding the customers who purchased the items, such as customer age, occupation, credit rating, income, and address, may also be stored. Considering each database attribute or warehouse dimension as a predicate, we can therefore mine association rules containing multiple predicates, such as

$$age(X, \text{“20...29”}) \wedge occupation(X, \text{“student”}) \Rightarrow buys(X, \text{“laptop”}). \quad (5.13)$$

Association rules that involve two or more dimensions or predicates can be referred to as **multidimensional association rules**. Rule (5.13) contains three predicates (*age, occupation*, and *buys*), each of which occurs *only once* in the rule. Hence, we say that it has **no repeated predicates**. Multidimensional association rules with no repeated predicates are called **interdimensional association rules**. We can also mine multidimensional association rules with repeated predicates, which contain multiple occurrences of some predicates. These rules are called **hybrid-dimensional association rules**. An example of such a rule is the following, where the predicate *buys* is repeated:

$$age(X, \text{“20...29”}) \wedge buys(X, \text{“laptop”}) \Rightarrow buys(X, \text{“HP printer”}) \quad (5.14)$$

Note that database attributes can be categorical or quantitative. **Categorical** attributes have a finite number of possible values, with no ordering among the values (e.g., *occupation, brand, color*). Categorical attributes are also called **nominal** attributes, because their values are “names of things.” **Quantitative** attributes are numeric and have an implicit ordering among values (e.g., *age, income, price*). Techniques for mining multidimensional association rules can be categorized into two basic approaches regarding the treatment of quantitative attributes.

In the first approach, *quantitative attributes are discretized using predefined concept hierarchies*. This discretization occurs before mining. For instance, a concept hierarchy for *income* may be used to replace the original numeric values of this attribute by interval labels, such as “0…20K”, “21K …30K”, “31K …40K”, and so on. Here, discretization is *static* and predetermined. Chapter 2 on data preprocessing gave several techniques for discretizing numeric attributes. The discretized numeric attributes, with their interval labels, can then be treated as categorical attributes (where each interval is considered a category). We refer to this as **mining multidimensional association rules using static discretization of quantitative attributes.**

In the second approach, *quantitative attributes are discretized or clustered into "bins" based on the distribution of the data.* These bins may be further combined during the mining process. The discretization process is *dynamic* and established so as to satisfy some mining criteria, such as maximizing the confidence of the rules mined. Because this strategy treats the numeric attribute values as quantities rather than as predefined ranges or categories, association rules mined from this approach are also referred to as **(dynamic) quantitative association rules.**

Let's study each of these approaches for mining multidimensional association rules. For simplicity, we confine our discussion to interdimensional association rules. Note that rather than searching for frequent itemsets (as is done for single-dimensional association rule mining), in multidimensional association rule mining we search for frequent *predicate sets.* A ***k*-predicate set** is a set containing *k* conjunctive predicates. For instance, the set of predicates {*age, occupation, buys*} from Rule (5.13) is a 3-predicate set. Similar to the notation used for itemsets, we use the notation L_k to refer to the set of frequent *k*-predicate sets.

Mining Multidimensional Association Rules Using Static Discretization of Quantitative Attributes: Quantitative attributes, in this case, are discretized before mining using predefined concept hierarchies or data discretization techniques, where numeric values are replaced by interval labels. Categorical attributes may also be generalized to higher conceptual levels if desired. If the resulting task-relevant data are stored in a relational table, then any of the frequent itemset mining algorithms we have discussed can be modified easily so as to find all frequent predicate sets rather than frequent itemsets. In particular, instead of searching on only one attribute like buys, we need to search through all of the relevant attributes, treating each attribute-value pair as an itemset.

Alternatively, the transformed multidimensional data may be used to construct a data cube. Data cubes are well suited for the mining of multidimensional association rules: They store aggregates (such as counts), in multidimensional space, which is essential for computing the support and confidence of multidimensional association rules. Figure 5.13 shows the lattice of cuboids defining a data cube for the dimensions age, income, and buys. The cells of an n-dimensional cuboid can be used to store the support counts of the corresponding n-predicate sets. The base cuboid aggregates the task-relevant data by age, income, and buys; the 2-D cuboid, (age, income),

aggregates by age and income, and so on; the 0-D (apex) cuboid contains the total number of transactions in the task-relevant data.

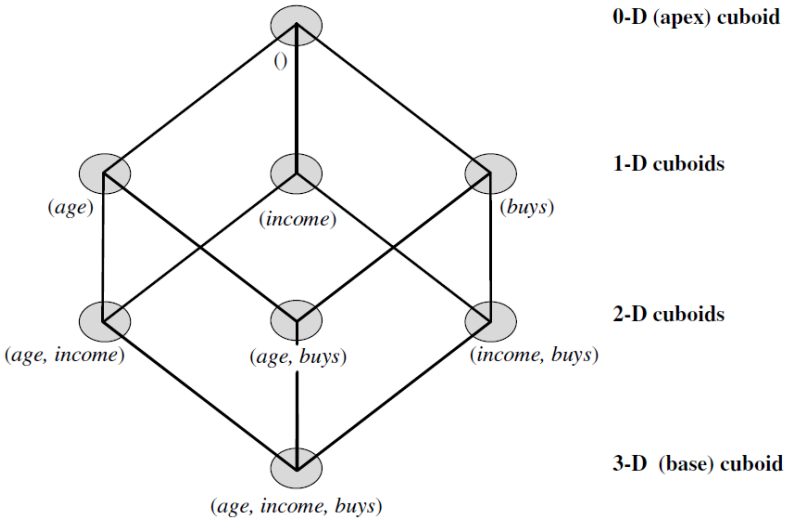


Figure 5.13 Lattice of cuboids, making up a 3-D data cube. Each cuboid represents a different group-by. The base cuboid contains the three predicates *age*, *income*, and *buys*.

Mining Quantitative Association Rules: Quantitative association rules are multidimensional association rules in which the numeric attributes are dynamically discretized during the mining process so as to satisfy some mining criteria, such as maximizing the confidence or compactness of the rules mined. In this section, we focus specifically on how to mine quantitative association rules having two quantitative attributes on the left-hand side of the rule and one categorical attribute on the right-hand side of the rule. That is,

$$A_{quan1} \wedge A_{quan2} \Rightarrow A_{cat}$$

where A_{quan1} and A_{quan2} are tests on quantitative attribute intervals (where the intervals are dynamically determined), and A_{cat} tests a categorical attribute from the task-relevant data. Such rules have been referred to as **two-dimensional quantitative association rules**, because they contain two quantitative dimensions. For instance, suppose you are curious about the association relationship between pairs of quantitative attributes, like customer age and income, and the type of television (such as *high-definition TV*, i.e., *HDTV*) that customers like to buy. An example of such a 2-D quantitative association rule is

$$age(X, \text{“30...39”}) \wedge income(X, \text{“42}K\text{...48}K\text{”}) \Rightarrow buys(X, \text{“}HDTV\text{”}) \quad (5.15)$$

“How can we find such rules?” Let’s look at an approach used in a system called ARCS (Association Rule Clustering System), which borrows ideas from image processing.

FROM ASSOCIATION MINING TO CORRELATION ANALYSIS

Most association rule mining algorithms employ a support-confidence framework. Often, many interesting rules can be found using low support thresholds. Although minimum support and confidence thresholds help weed out or exclude the exploration of a good number of uninteresting rules, many rules so generated are still not interesting to the users. Unfortunately, this is especially true when mining at low support thresholds or mining for long patterns. This has been one of the major bottlenecks for successful application of association rule mining.

Strong Rules Are Not Necessarily Interesting: Whether or not a rule is interesting can be assessed either subjectively or objectively. Ultimately, only the user can judge if a given rule is interesting, and this judgment, being subjective, may differ from one user to another. However, objective interestingness measures, based on the statistics "behind" the data, can be used as one step toward the goal of weeding out uninteresting rules from presentation to the user.

From Association Analysis to Correlation Analysis:

As we have seen above, the support and confidence measures are insufficient at filtering out uninteresting association rules. To tackle this weakness, a correlation measure can be used to augment the support-confidence framework for association rules. This leads to *correlation rules* of the form

$$A \Rightarrow B \ [support, confidence. correlation]. \tag{5.22}$$

That is, a correlation rule is measured not only by its support and confidence but also by the correlation between itemsets A and B. There are many different correlation measures from which to choose. In this section, we study various correlation measures to determine which would be good for mining large data sets.

Lift is a simple correlation measure that is given as follows. The occurrence of itemset A is **independent** of the occurrence of itemset B if $P(A \cup B) = P(A)P(B)$; otherwise, itemsets A and B are **dependent** and **correlated** as events. This definition can easily be extended to more than two itemsets. The **lift** between the occurrence of A and B can be measured by computing

$$lift(A, B) = \frac{P(A \cup B)}{P(A)P(B)}. \tag{5.23}$$

If the resulting value of Equation (5.23) is less than 1, then the occurrence of A is *negatively correlated with* the occurrence of B. If the resulting value is greater than 1, then A and B are *positively correlated*, meaning that the occurrence of one implies the occurrence of the other. If the resulting value is equal to 1, then A and B are *independent* and there is no correlation between them.

Equation (5.23) is equivalent to $P(B|A)/P(B)$, or $conf(A \Rightarrow B)/sup(B)$, which is also referred as the *lift* of the association (or correlation) rule $A \Rightarrow B$. In other words, it assesses the degree to which the occurrence of one "lifts" the occurrence of the other. For example, if A corresponds to the sale of computer games and B corresponds to the sale of videos, then given the current market conditions, the sale of games is said to increase or "lift" the likelihood of the sale of videos by a factor of the value returned by Equation (5.23).

Let's examine two other correlation measures, all confidence and cosine, as defined below.

Given an itemset $X = \{i_1, i_2, \ldots, i_k\}$, the **all_confidence** of X is defined as

$$all\_conf(X) = \frac{sup(X)}{max\_item\_sup(X)} = \frac{sup(X)}{max\{sup(i_j)|\forall i_j \in X\}}, \tag{5.24}$$

where $max\{sup(i_j)|\forall i_j \in X\}$ is the maximum (single) item support of all the items in X, and hence is called the **max_item_sup** of the itemset X. The *all_confidence* of X is the minimal confidence among the set of rules $i_j \rightarrow X - i_j$, where $i_j \in X$.

Given two itemsets A and B, the **cosine** measure of A and B is defined as

$$cosine(A, B) = \frac{P(A \cup B)}{\sqrt{P(A) \times P(B)}} = \frac{sup(A \cup B)}{\sqrt{sup(A) \times sup(B)}}. \tag{5.25}$$

The *cosine* measure can be viewed as a harmonized *lift* measure: the two formulae are similar except that for cosine, the *square root* is taken on the product of the probabilities of A and B. This is an important difference, however, because by taking the square root, the cosine value is only influenced by the supports of A, B, and $A \cup B$, and not by the total number of transactions.

CONSTRAINT-BASED ASSOCIATION MINING

A data mining process may uncover thousands of rules from a given set of data, most of which end up being unrelated or uninteresting to the users. Often, users have a good sense of which "direction" of mining may lead to interesting patterns and the "form" of the patterns or rules they would like to find. Thus, a good heuristic is to have the users specify such intuition or expectations as constraints to confine the search space. This strategy is known as constraint-based mining. The constraints can include the following:

1) **Knowledge type constraints:** These specify the type of knowledge to be mined, such as association or correlation.
2) **Data constraints:** These specify the set of task-relevant data.
3) **Dimension/level constraints:** These specify the desired dimensions (or attributes) of the data, or levels of the concept hierarchies, to be used in mining.
4) **Interestingness constraints:** These specify thresholds on statistical measures of rule interestingness, such as support, confidence, and correlation.
5) **Rule constraints:** These specify the form of rules to be mined. Such constraints may be expressed as metarules (rule templates), as the maximum or minimum number of predicates that can occur in the rule antecedent or consequent, or as relationships among attributes, attribute values, and/or aggregates.

The above constraints can be specified using a high-level declarative data mining query language and user interface.

Metarule-Guided Mining of Association Rules: "How are metarules useful?" Metarules allow users to specify the syntactic form of rules that they are interested in mining. The rule forms can be used as constraints to help improve the efficiency of the mining process. Metarules may be based on the analyst's experience, expectations, or intuition regarding the data or may be automatically generated based on the database schema.

"*How can metarules be used to guide the mining process?*" Let's examine this problem closely. Suppose that we wish to mine interdimensional association rules, such as in the example above. A metarule is a rule template of the form

$$P_1 \wedge P_2 \wedge \cdots \wedge P_l \Rightarrow Q_1 \wedge Q_2 \wedge \cdots \wedge Q_r, \quad (5.28)$$

where P_i $(i = 1, \ldots, l)$ and Q_j $(j = 1, \ldots, r)$ are either instantiated predicates or predicate variables. Let the number of predicates in the metarule be $p = l + r$. In order to find interdimensional association rules satisfying the template,

- We need to find all frequent p-predicate sets, L_p.
- We must also have the support or count of the l-predicate subsets of L_p in order to compute the confidence of rules derived from L_p.

Constraint Pushing: Mining Guided by Rule Constraints: Rule constraints specify expected set/subset relationships of the variables in the mined rules, constant initiation of variables, and aggregate functions. Users typically employ their knowledge of the application or data to specify rule constraints for the mining task. These rule constraints may be used together with, or as an alternative to, metarule-guided mining.

Dimension/level constraints and interestingness constraints can be applied after mining to filter out discovered rules, although it is generally more efficient and less expensive to use them during mining, to help prune the search space.

GRAPH PATTERN MINING (GPM)

Graph pattern mining is the mining of frequent subgraphs (also called (sub) graph patterns) in one or a set of graphs. Methods for mining graph patterns can be categorized into Apriori-based and pattern growth–based approaches. Alternatively, we can mine the set of closed graphs where a graph g is closed if there exists no proper super graph g′ that carries the same support count as g. Moreover, there are many variant graph patterns, including approximate frequent graphs, coherent graphs, and dense graphs. User-specified constraints can be pushed deep into the graph pattern mining process to improve mining efficiency.

Graph pattern mining has many interesting applications. For example, it can be used to generate compact and effective graph index structures based on the concept of frequent and discriminative graph patterns. Approximate structure similarity search can be achieved by exploring graph index structures and multiple graph features. Moreover, classification of graphs can also be performed effectively using frequent and discriminative subgraphs as features.

SEQUENTIAL PATTERN MINING (SPM)

Sequential pattern mining is a topic of data mining concerned with finding statistically relevant patterns between data examples where the values are delivered in a sequence. It is usually presumed that the values are discrete, and thus time series mining is closely related, but usually considered a different activity. Sequential pattern mining is a special case of structured data mining.

There are several key traditional computational problems addressed within this field. These include building efficient databases and indexes for sequence information, extracting the frequently occurring patterns, comparing sequences for similarity, and recovering missing sequence members. In general, sequence mining problems can be classified as string mining which is typically based on string processing algorithms and itemset mining which is typically based on association rule learning. Local process models extend sequential pattern mining to more complex patterns that can include (exclusive) choices, loops, and concurrency constructs in addition to the sequential ordering construct.

*****

UNIT – III

CLASSIFICATION

CLASSIFICATION & PREDICTION

What is classification?

Following are the examples of cases where the data analysis task is Classification:

1) A bank loan officer wants to analyze the data in order to know which customer (loan applicant) are risky or which are safe.
2) A marketing manager at a company needs to analyze a customer with a given profile, who will buy a new computer.

In both of the above examples, a model or classifier is constructed to predict the categorical labels. These labels are risky or safe for loan application data and yes or no for marketing data.

What is prediction?

Following are the examples of cases where the data analysis task is Prediction:

Suppose the marketing manager needs to predict how much a given customer will spend during a sale at his company. In this example we are bothered to predict a numeric value. Therefore the data analysis task is an example of numeric prediction. In this case, a model or a predictor will be constructed that predicts a continuous-valued-function or ordered value.

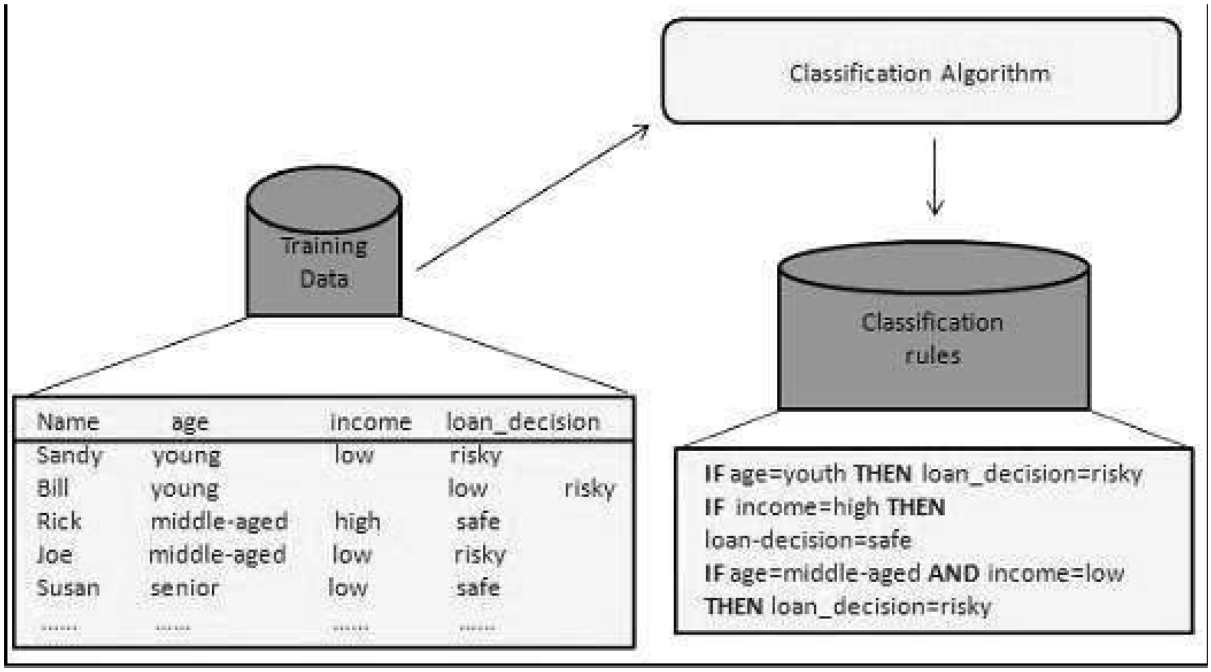
How Does Classification Works?

With the help of the bank loan application that we have discussed above, let us understand the working of classification. The Data Classification process includes two steps:

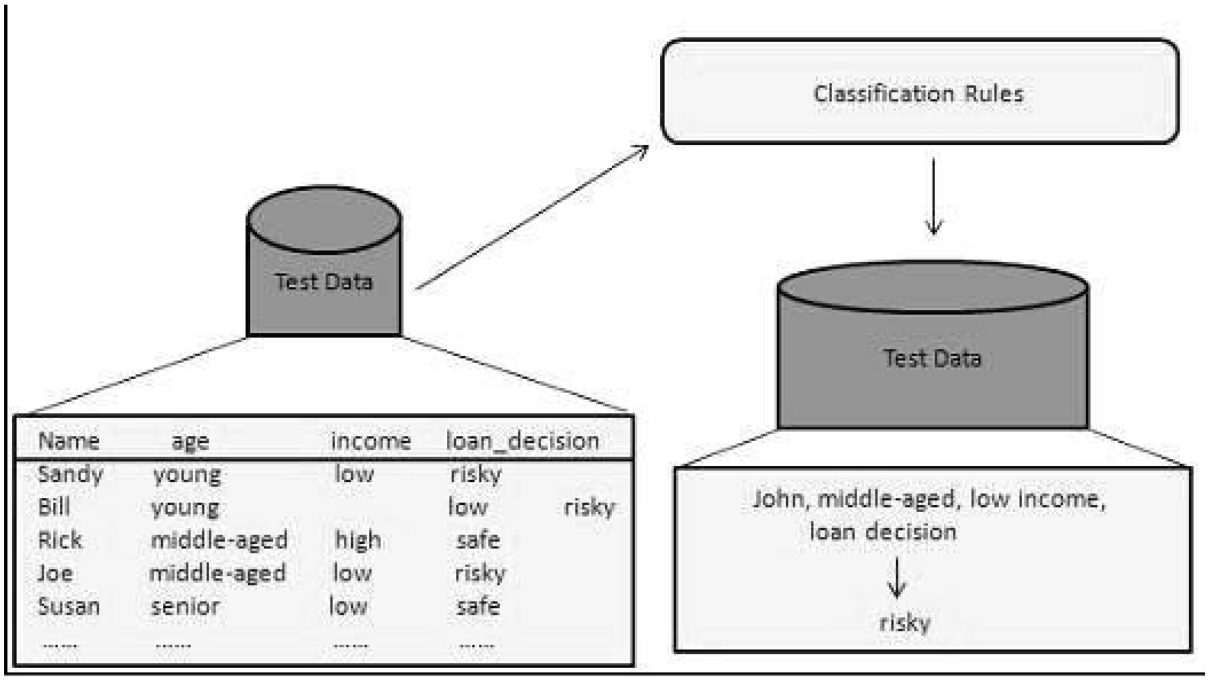
1) Building the Classifier or Model.
2) Using Classifier for Classification.

1) Building the Classifier or Model: This step is the learning step or the learning phase. In this step the classification algorithms build the classifier. The classifier is built from the training set made up of database tuples and

their associated class labels. Each tuple that constitutes the training set is referred to as a category or class. These tuples can also be referred to as sample, object or data points.



2) Using Classifier for Classification: In this step, the classifier is used for classification. Here the test data is used to estimate the accuracy of classification rules. The classification rules can be applied to the new data tuples if the accuracy is considered acceptable.



Classification and Prediction Issues/Issues Regarding Classification and Prediction: The major issue is preparing the data for Classification and Prediction. Preparing the data involves the following activities:

1) **Data Cleaning:** Data cleaning involves removing the noise and treatment of missing values. The noise is removed by applying smoothing techniques and the problem of missing values is solved by replacing a missing value with most commonly occurring value for that attribute.
2) **Relevance Analysis:** Database may also have the irrelevant attributes. Correlation analysis is used to know whether any two given attributes are related.
3) **Data Transformation and reduction:** The data can be transformed by any of the following methods.
4) **Normalization:** The data is transformed using normalization. Normalization involves scaling all values for given attribute in order to make them fall within a small specified range. Normalization is used when in the learning step, the neural networks or the methods involving measurements are used.
5) **Generalization:** The data can also be transformed by generalizing it to the higher concept. For this purpose we can use the concept hierarchies.

Comparing Classification and Prediction Methods: Classification and prediction methods can be compared and evaluated according to the following criteria:

1) **Accuracy:** The accuracy of a classifier refers to the ability of a given classifier to correctly predict the class label of new or previously unseen data (i.e., tuples without class label information). Similarly, the accuracy of a predictor refers to how well a given predictor can guess the value of the predicted attribute for new or previously unseen data. Accuracy can be estimated using one or more test sets that are independent of the training set. Because the accuracy computed is only

an estimate of how well the classifier or predictor will do on new data tuples, confidence limits can be computed to help gauge this estimate.

2) **Speed:** This refers to the computational costs involved in generating and using the given classifier or predictor.
3) **Robustness:** This is the ability of the classifier or predictor to make correct predictions given noisy data or data with missing values.
4) **Scalability:** This refers to the ability to construct the classifier or predictor efficiently given large amounts of data.
5) **Interpretability:** This refers to the level of understanding and insight that is provided by the classifier or predictor. Interpretability is subjective and therefore more difficult to assess.

Evaluation of Classifier-classification techniques: 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.

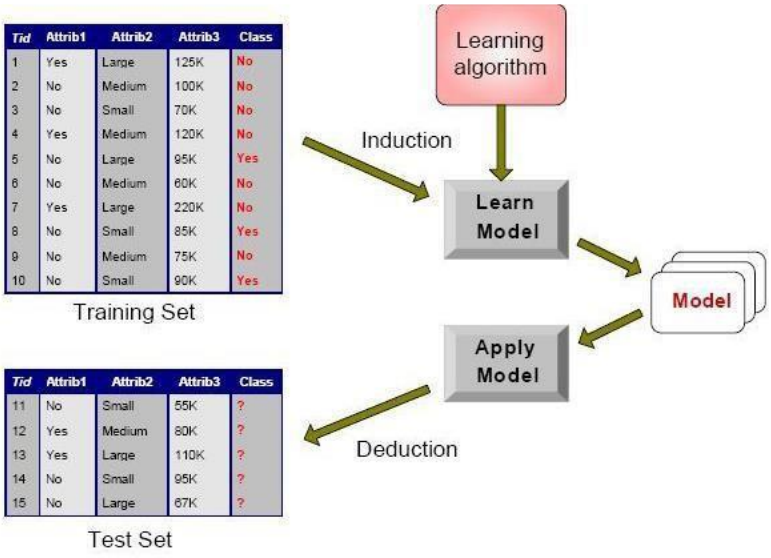


Fig: Classification model generation

CLASSIFICATION BY DECISION TREE INDUCTION

Decision tree induction is the learning of decision trees from class-labeled training tuples. A decision tree is a flowchart-like tree structure, where each internal node (nonleaf node) denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (or terminal node) holds a class label. The topmost node in a tree is the root node.

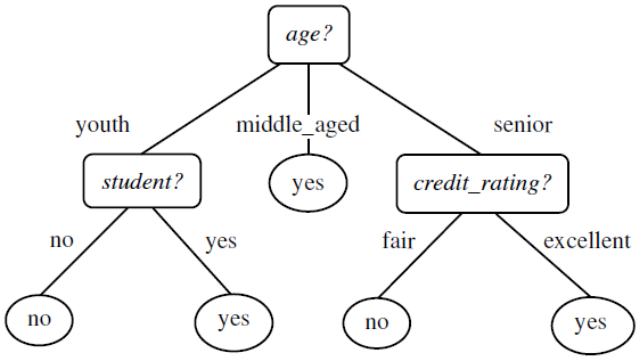


Figure 6.2 A decision tree for the concept *buys_computer*, indicating whether a customer at *AllElectronics* is likely to purchase a computer. Each internal (nonleaf) node represents a test on an attribute. Each leaf node represents a class (either *buys_computer = yes* or *buys_computer = no*).

A typical decision tree is shown in Figure 6.2. It represents the concept buys _computer, that is, it predicts whether a customer at AllElectronics is likely to purchase a computer. Internal nodes are denoted by rectangles, and leaf nodes are denoted by ovals. Some decision tree algorithms produce only binary trees (where each internal node branches to exactly two other nodes), whereas others can produce nonbinary trees.

"How are decision trees used for classification?"

Given a tuple, X, for which the associated class label is unknown, the attribute values of the tuple are tested against the decision tree. A path is traced from the root to a leaf node, which holds the class prediction for that tuple. Decision trees can easily be converted to classification rules.

"Why are decision tree classifiers so popular?"

The construction of decision tree classifiers does not require any domain knowledge or parameter setting, and therefore is appropriate for exploratory knowledge discovery. Decision trees can handle high dimensional data. Their

representation of acquired knowledge in tree form is intuitive and generally easy to assimilate by humans. The learning and classification steps of decision tree induction are simple and fast. In general, decision tree classifiers have good accuracy. However, successful use may depend on the data at hand. Decision tree induction algorithms have been used for classification in many application areas, such as medicine, manufacturing and production, financial analysis, astronomy, and molecular biology. Decision trees are the basis of several commercial rule induction systems.

Decision Tree Induction: During the late 1970s and early 1980s, J. Ross Quinlan, a researcher in machine learning, developed a decision tree algorithm known as ID3 (Iterative Dichotomiser). This work expanded on earlier work on concept learning systems, described by E. B. Hunt, J. Marin, and P. T. Stone. Quinlan later presented C4.5 (a successor of ID3), which became a benchmark to which newer supervised learning algorithms are often compared. In 1984, a group of statisticians (L. Breiman, J. Friedman, R. Olshen, and C. Stone) published the book Classification and Regression Trees (CART), which described the generation of binary decision trees. ID3 and CART were invented independently of one another at around the same time, yet follow a similar approach for learning decision trees from training tuples. These two cornerstone algorithms spawned a flurry of work on decision tree induction.

ID3, C4.5, and CART adopt a greedy (i.e., nonbacktracking) approach in which decision trees are constructed in a top-down recursive divide-and-conquer manner. Most algorithms for decision tree induction also follow such a top-down approach, which starts with a training set of tuples and their associated class labels.

Algorithm: Generate_decision_tree. Generate a decision tree from the training tuples of data partition D.

Input:

- Data partition, D, which is a set of training tuples and their associated class labels;
- *attribute_list*, the set of candidate attributes;
- *Attribute_selection_method*, a procedure to determine the splitting criterion that "best" partitions the data tuples into individual classes. This criterion consists of a *splitting_attribute* and, possibly, either a *split point* or *splitting subset*.

Output: A decision tree.

Method:

```
(1)  create a node N;
(2)  if tuples in D are all of the same class, C then
(3)      return N as a leaf node labeled with the class C;
(4)  if attribute_list is empty then
(5)      return N as a leaf node labeled with the majority class in D; // majority voting
(6)  apply Attribute_selection_method(D, attribute_list) to find the "best" splitting_criterion;
(7)  label node N with splitting_criterion;
(8)  if splitting_attribute is discrete-valued and
         multiway splits allowed then // not restricted to binary trees
(9)      attribute_list ← attribute_list − splitting_attribute; // remove splitting_attribute
(10) for each outcome j of splitting_criterion
     // partition the tuples and grow subtrees for each partition
(11)     let Dj be the set of data tuples in D satisfying outcome j; // a partition
(12)     if Dj is empty then
(13)         attach a leaf labeled with the majority class in D to node N;
(14)     else attach the node returned by Generate_decision_tree(Dj, attribute_list) to node N;
     endfor
(15) return N;
```

Figure 6.3 Basic algorithm for inducing a decision tree from training tuples.

The training set is recursively partitioned into smaller subsets as the tree is being built. A basic decision tree algorithm is summarized in Figure 6.3. At first glance, the algorithm may appear long, but fear not! It is quite straightforward. The strategy is as follows.

- The algorithm is called with three parameters: D, *attribute_list*, and *Attribute_selection_method*. We refer to D as a data partition. Initially, it is the complete set of training tuples and their associated class labels. The parameter *attribute_list* is a list of attributes describing the tuples. *Attribute_selection_method* specifies a heuristic procedure for selecting the attribute that "best" discriminates the given tuples according

to class. This procedure employs an attribute selection measure, such as information gain or the gini index. Whether the tree is strictly binary is generally driven by the attribute selection measure. Some attribute selection measures, such as the gini index, enforce the resulting tree to be binary. Others, like information gain, do not, therein allowing multiway splits (i.e., two or more branches to be grown from a node).

- The tree starts as a single node, N, representing the training tuples in D (step 1).[5]
- If the tuples in D are all of the same class, then node N becomes a leaf and is labeled with that class (steps 2 and 3). Note that steps 4 and 5 are terminating conditions. All of the terminating conditions are explained at the end of the algorithm.
- Otherwise, the algorithm calls *Attribute_selection_method* to determine the **splitting criterion**. The splitting criterion tells us which attribute to test at node N by determining the "best" way to separate or partition the tuples in D into individual classes (step 6). The splitting criterion also tells us which branches to grow from node N with respect to the outcomes of the chosen test. More specifically, the splitting criterion indicates the **splitting attribute** and may also indicate either a **split-point** or a **splitting subset**. The splitting criterion is determined so that, ideally, the resulting partitions at each branch are as "pure" as possible. A partition is **pure** if all of the tuples in it belong to the same class. In other words, if we were to split up the tuples in D according to the mutually exclusive outcomes of the splitting criterion, we hope for the resulting partitions to be as pure as possible.

- The node N is labeled with the splitting criterion, which serves as a test at the node (step 7). A branch is grown from node N for each of the outcomes of the splitting criterion. The tuples in D are partitioned accordingly (steps 10 to 11). There are three possible scenarios, as illustrated in Figure 6.4. Let A be the splitting attribute. A has v distinct values, $\{a_1, a_2, \ldots, a_v\}$, based on the training data.

  1. *A is discrete-valued*: In this case, the outcomes of the test at node N correspond directly to the known values of A. A branch is created for each known value, a_j, of A and labeled with that value (Figure 6.4(a)). Partition D_j is the subset of class-labeled tuples in D having value a_j of A. Because all of the tuples in a given partition have the same value for A, then A need not be considered in any future partitioning of the tuples. Therefore, it is removed from *attribute_list* (steps 8 to 9).
  2. *A is continuous-valued*: In this case, the test at node N has two possible outcomes, corresponding to the conditions $A \leq$ *split_point* and $A >$ *split_point*, respectively,

where *split_point* is the split-point returned by *Attribute_selection_method* as part of the splitting criterion. (In practice, the split-point, a, is often taken as the midpoint of two known adjacent values of A and therefore may not actually be a pre-existing value of A from the training data.) Two branches are grown from N and labeled according to the above outcomes (Figure 6.4(b)). The tuples are partitioned such that D_1 holds the subset of class-labeled tuples in D for which $A \leq split\_point$, while D_2 holds the rest.

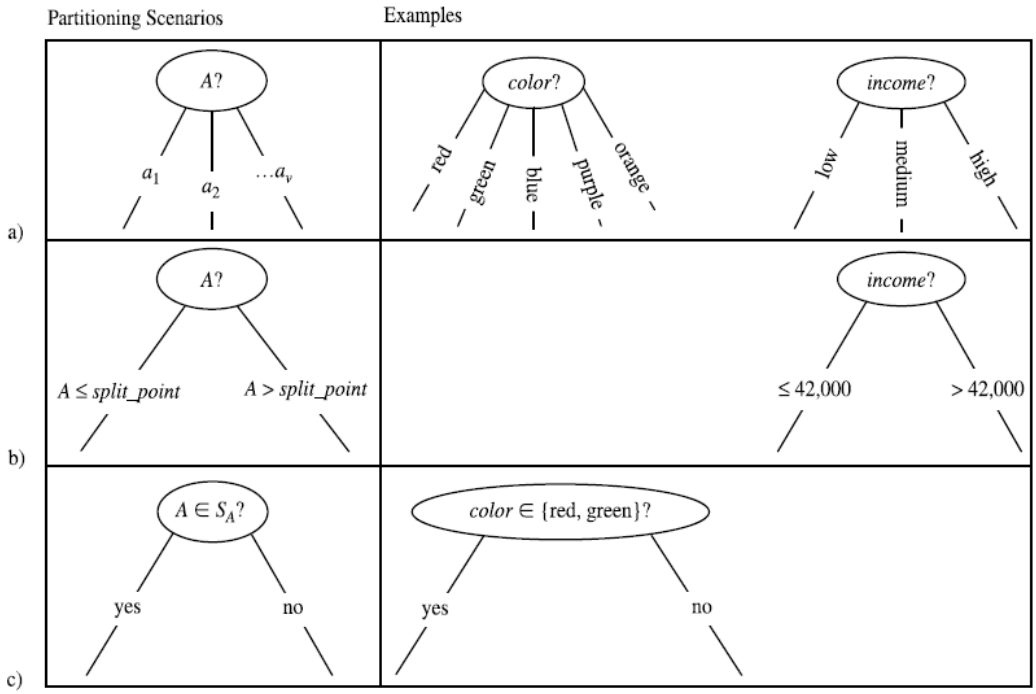


Figure 6.4 Three possibilities for partitioning tuples based on the splitting criterion, shown with examples. Let A be the splitting attribute. (a) If A is discrete-valued, then one branch is grown for each known value of A. (b) If A is continuous-valued, then two branches are grown, corresponding to $A \leq split\_point$ and $A > split\_point$. (c) If A is discrete-valued and a binary tree must be produced, then the test is of the form $A \in S_A$, where S_A is the splitting subset for A.

3. *A is discrete-valued* and a *binary tree* must be produced (as dictated by the attribute selection measure or algorithm being used): The test at node N is of the form "$A \in S_A$?". S_A is the splitting subset for A, returned by *Attribute_selection_method* as part of the splitting criterion. It is a subset of the known values of A. If a given tuple has value a_j of A and if $a_j \in S_A$, then the test at node N is satisfied. Two branches are grown from N (Figure 6.4(c)). By convention, the left branch out of N is labeled *yes* so that D_1 corresponds to the subset of class-labeled tuples in D

that satisfy the test. The right branch out of N is labeled *no* so that D_2 corresponds to the subset of class-labeled tuples from D that do not satisfy the test.

- The algorithm uses the same process recursively to form a decision tree for the tuples at each resulting partition, D_j, of D (step 14).
- The recursive partitioning stops only when any one of the following terminating conditions is true:

  1. All of the tuples in partition D (represented at node N) belong to the same class (steps 2 and 3), or
  2. There are no remaining attributes on which the tuples may be further partitioned (step 4). In this case, **majority voting** is employed (step 5). This involves converting node N into a leaf and labeling it with the most common class in D. Alternatively, the class distribution of the node tuples may be stored.
  3. There are no tuples for a given branch, that is, a partition D_j is empty (step 12). In this case, a leaf is created with the majority class in D (step 13).

- The resulting decision tree is returned (step 15).

The computational complexity of the algorithm given training set D is $O(n \times |D| \times log(|D|))$, where n is the number of attributes describing the tuples in D and $|D|$ is the number of training tuples in D. This means that the computational cost of growing a tree grows at most $n \times |D| \times log(|D|)$ with $|D|$ tuples. The proof is left as an exercise for the reader.

Incremental versions of decision tree induction have also been proposed. When given new training data, these restructure the decision tree acquired from learning on previous training data, rather than relearning a new tree from scratch.

Differences in decision tree algorithms include how the attributes are selected in creating the tree (Section 6.3.2) and the mechanisms used for pruning (Section 6.3.3). The basic algorithm described above requires one pass over the training tuples in D for each level of the tree. This can lead to long training times and lack of available memory when dealing with large databases. Improvements regarding the scalability of decision tree induction are discussed in Section 6.3.4. A discussion of strategies for extracting rules from decision trees is given in Section 6.5.2 regarding rule-based classification.

Attribute Selection Measures: An attribute selection measure is a heuristic for selecting the splitting criterion that "best" separates a given data partition, D, of class-labeled training tuples into individual classes. If we were to split D into smaller partitions according to the outcomes of the splitting criterion, ideally each partition would be pure (i.e., all of the tuples that fall into a given partition would belong to the same class). Conceptually, the "best" splitting

criterion is the one that most closely results in such a scenario. Attribute selection measures are also known as splitting rules because they determine how the tuples at a given node are to be split. The attribute selection measure provides a ranking for each attribute describing the given training tuples. The attribute having the best score for the measure is chosen as the splitting attribute for the given tuples. If the splitting attribute is continuous-valued or if we are restricted to binary trees then, respectively, either a split point or a splitting subset must also be determined as part of the splitting criterion.

The tree node created for partition D is labeled with the splitting criterion, branches are grown for each outcome of the criterion, and the tuples are partitioned accordingly. This section describes three popular attribute selection measures—information gain, gain ratio, and gini index.

The notation used herein is as follows. Let D, the data partition, be a training set of class-labeled tuples. Suppose the class label attribute has m distinct values defining m distinct classes, C_i (for $i = 1, \ldots, m$). Let $C_{i,D}$ be the set of tuples of class C_i in D. Let $|D|$ and $|C_{i,D}|$ denote the number of tuples in D and $C_{i,D}$, respectively.

Information gain:

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i),$$

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j).$$

$$Gain(A) = Info(D) - Info_A(D).$$

Gain ratio:

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \log_2\left(\frac{|D_j|}{|D|}\right).$$

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo(A)}.$$

Gini index:

$$Gini(D) = 1 - \sum_{i=1}^{m} p_i^2,$$

$$Gini_A(D) = \frac{|D_1|}{|D|} Gini(D_1) + \frac{|D_2|}{|D|} Gini(D_2).$$

$$\Delta Gini(A) = Gini(D) - Gini_A(D).$$

Tree Pruning: When a decision tree is built, many of the branches will reflect anomalies in the training data due to noise or outliers. Tree pruning methods address this problem of overfitting the data. Such methods typically use statistical measures to remove the least reliable branches. An unpruned tree and a pruned version of it are shown in Figure 6.6. Pruned trees tend to be smaller and less complex and, thus, easier to comprehend. They are usually faster and better at correctly classifying independent test data (i.e., of previously unseen tuples) than unpruned trees.

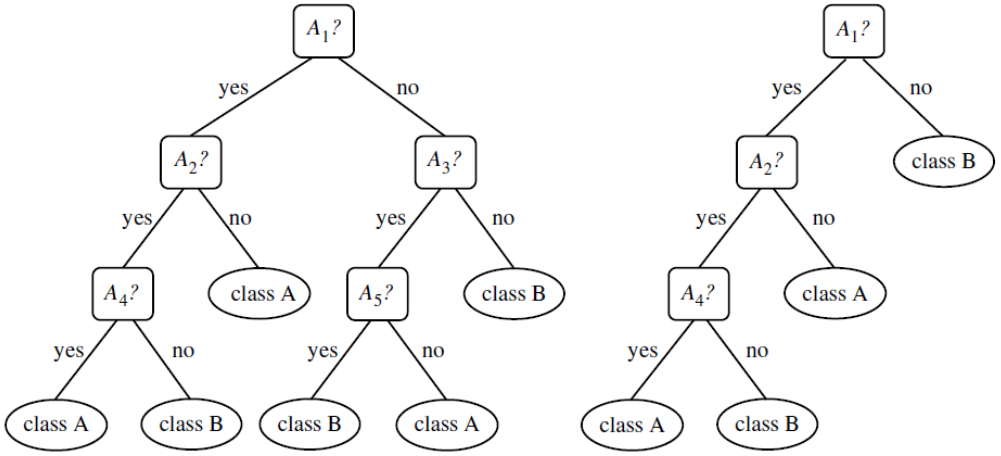


Figure 6.6 An unpruned decision tree and a pruned version of it.

BAYESIAN CLASSIFICATION

"What are Bayesian classifiers?"

Bayesian classifiers are statistical classifiers. They can predict class membership probabilities, such as the probability that a given tuple belongs to a particular class.

Bayesian classification is based on Bayes' theorem, described below. Studies comparing classification algorithms have found a simple Bayesian classifier known as the naïve Bayesian classifier to be comparable in performance with decision tree and selected neural network classifiers. Bayesian classifiers have also exhibited high accuracy and speed when applied to large databases.

Naïve Bayesian classifiers assume that the effect of an attribute value on a given class is independent of the values of the other attributes. This assumption is called class conditional independence. It is made to simplify the computations involved and, in this sense, is considered "naïve." Bayesian belief networks are graphical models, which unlike naïve Bayesian classifiers, allow the representation of dependencies among subsets of attributes. Bayesian belief networks can also be used for classification.

Bayes' Theorem:

Bayes' theorem is named after Thomas Bayes, a nonconformist English clergyman who did early work in probability and decision theory during the 18th century. Let $\boldsymbol{X}$ be a data tuple. In Bayesian terms, $\boldsymbol{X}$ is considered "evidence." As usual, it is described by measurements made on a set of n attributes. Let H be some hypothesis, such as that the data tuple $\boldsymbol{X}$ belongs to a specified class C. For classification problems, we want to determine $P(H|\boldsymbol{X})$, the probability that the hypothesis H holds given the "evidence" or observed data tuple $\boldsymbol{X}$. In other words, we are looking for the probability that tuple $\boldsymbol{X}$ belongs to class C, given that we know the attribute description of $\boldsymbol{X}$.

$P(H|\boldsymbol{X})$ is the **posterior probability**, or *a posteriori probability*, of H conditioned on $\boldsymbol{X}$. For example, suppose our world of data tuples is confined to customers described by the attributes *age* and *income*, respectively, and that $\boldsymbol{X}$ is a 35-year-old customer with an income of $40,000. Suppose that H is the hypothesis that our customer will buy a computer. Then $P(H|\boldsymbol{X})$ reflects the probability that customer $\boldsymbol{X}$ will buy a computer given that we know the customer's age and income.

In contrast, $P(H)$ is the **prior probability**, or *a priori probability,* of H. For our example, this is the probability that any given customer will buy a computer, regardless of age, income, or any other information, for that matter. The posterior probability, $P(H|\boldsymbol{X})$, is based on more information (e.g., customer information) than the prior probability, $P(H)$, which is independent of $\boldsymbol{X}$.

Similarly, $P(\boldsymbol{X}|H)$ is the posterior probability of $\boldsymbol{X}$ conditioned on H. That is, it is the probability that a customer, $\boldsymbol{X}$, is 35 years old and earns $40,000, given that we know the customer will buy a computer.

$P(\boldsymbol{X})$ is the prior probability of $\boldsymbol{X}$. Using our example, it is the probability that a person from our set of customers is 35 years old and earns $40,000.

"*How are these probabilities estimated?*" $P(H)$, $P(\boldsymbol{X}|H)$, and $P(\boldsymbol{X})$ may be estimated from the given data, as we shall see below. **Bayes' theorem** is useful in that it provides a way of calculating the posterior probability, $P(H|\boldsymbol{X})$, from $P(H)$, $P(\boldsymbol{X}|H)$, and $P(\boldsymbol{X})$. Bayes' theorem is

$$P(H|\boldsymbol{X}) = \frac{P(\boldsymbol{X}|H)P(H)}{P(\boldsymbol{X})}. \tag{6.10}$$

Naïve Bayesian Classification: The naïve Bayesian classifier, or simple Bayesian classifier, works as follows:

1. Let D be a training set of tuples and their associated class labels. As usual, each tuple is represented by an n-dimensional attribute vector, $\boldsymbol{X} = (x_1, x_2, \ldots, x_n)$, depicting n measurements made on the tuple from n attributes, respectively, $A_1, A_2, \ldots, A_n$.
2. Suppose that there are m classes, $C_1, C_2, \ldots, C_m$. Given a tuple, $\boldsymbol{X}$, the classifier will predict that $\boldsymbol{X}$ belongs to the class having the highest posterior probability, conditioned on $\boldsymbol{X}$. That is, the naïve Bayesian classifier predicts that tuple $\boldsymbol{X}$ belongs to the class C_i if and only if

   $$P(C_i|\boldsymbol{X}) > P(C_j|\boldsymbol{X}) \quad \text{for } 1 \leq j \leq m, j \neq i.$$

   Thus we maximize $P(C_i|\boldsymbol{X})$. The class C_i for which $P(C_i|\boldsymbol{X})$ is maximized is called the *maximum posteriori hypothesis*. By Bayes' theorem (Equation (6.10)),

   $$P(C_i|\boldsymbol{X}) = \frac{P(\boldsymbol{X}|C_i)P(C_i)}{P(\boldsymbol{X})}. \tag{6.11}$$

3. As $P(\boldsymbol{X})$ is constant for all classes, only $P(\boldsymbol{X}|C_i)P(C_i)$ need be maximized. If the class prior probabilities are not known, then it is commonly assumed that the classes are equally likely, that is, $P(C_1) = P(C_2) = \cdots = P(C_m)$, and we would therefore maximize $P(\boldsymbol{X}|C_i)$. Otherwise, we maximize $P(\boldsymbol{X}|C_i)P(C_i)$. Note that the class prior probabilities may be estimated by $P(C_i) = |C_{i,D}|/|D|$, where $|C_{i,D}|$ is the number of training tuples of class C_i in D.
4. Given data sets with many attributes, it would be extremely computationally expensive to compute $P(\boldsymbol{X}|C_i)$. In order to reduce computation in evaluating $P(\boldsymbol{X}|C_i)$, the naive assumption of **class conditional independence** is made. This presumes that the values of the attributes are conditionally independent of one another, given the class label of the tuple (i.e., that there are no dependence relationships among the attributes). Thus,

$$P(X|C_i) = \prod_{k=1}^{n} P(x_k|C_i) \quad (6.12)$$
$$= P(x_1|C_i) \times P(x_2|C_i) \times \cdots \times P(x_n|C_i).$$

We can easily estimate the probabilities $P(x_1|C_i), P(x_2|C_i), \ldots, P(x_n|C_i)$ from the training tuples. Recall that here x_k refers to the value of attribute A_k for tuple $\boldsymbol{X}$. For each attribute, we look at whether the attribute is categorical or continuous-valued. For instance, to compute $P(\boldsymbol{X}|C_i)$, we consider the following:

(a) If A_k is categorical, then $P(x_k|C_i)$ is the number of tuples of class C_i in D having the value x_k for A_k, divided by $|C_{i,D}|$, the number of tuples of class C_i in D.

(b) If A_k is continuous-valued, then we need to do a bit more work, but the calculation is pretty straightforward. A continuous-valued attribute is typically assumed to have a Gaussian distribution with a mean μ and standard deviation σ, defined by

$$g(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \quad (6.13)$$

so that

$$P(x_k|C_i) = g(x_k, \mu_{C_i}, \sigma_{C_i}). \quad (6.14)$$

These equations may appear daunting, but hold on! We need to compute μ_{C_i} and σ_{C_i}, which are the mean (i.e., average) and standard deviation, respectively, of the values of attribute A_k for training tuples of class C_i. We then plug these two quantities into Equation (6.13), together with x_k, in order to estimate $P(x_k|C_i)$. For example, let $\boldsymbol{X} = (35, \$40{,}000)$, where A_1 and A_2 are the attributes *age* and *income*, respectively. Let the class label attribute be *buys_computer*. The associated class label for $\boldsymbol{X}$ is *yes* (i.e., *buys_computer* = *yes*). Let's suppose that *age* has not been discretized and therefore exists as a continuous-valued attribute. Suppose that from the training set, we find that customers in D who buy a computer are 38 ± 12 years of age. In other words, for attribute *age* and this class, we have $\mu = 38$ years and $\sigma = 12$. We can plug these quantities, along with $x_1 = 35$ for our tuple $\boldsymbol{X}$ into Equation (6.13) in order to estimate $P(age = 35|buys\_computer = yes)$. For a quick review of mean and standard deviation calculations, please see Section 2.2.

5. In order to predict the class label of $\boldsymbol{X}$, $P(\boldsymbol{X}|C_i)P(C_i)$ is evaluated for each class C_i. The classifier predicts that the class label of tuple $\boldsymbol{X}$ is the class C_i if and only if

$$P(\boldsymbol{X}|C_i)P(C_i) > P(\boldsymbol{X}|C_j)P(C_j) \quad \text{for } 1 \leq j \leq m, j \neq i. \quad (6.15)$$

In other words, the predicted class label is the class C_i for which $P(\boldsymbol{X}|C_i)P(C_i)$ is the maximum.

"How effective are Bayesian classifiers?"

Various empirical studies of this classifier in comparison to decision tree and neural network classifiers have found it to be comparable in some domains. In theory, Bayesian classifiers have the minimum error rate in comparison to all other classifiers. However, in practice this is not always the case, owing to inaccuracies in the assumptions made for its use, such as class conditional independence, and the lack of available probability data.

Bayesian classifiers are also useful in that they provide a theoretical justification for other classifiers that do not explicitly use Bayes' theorem. For example, under certain assumptions, it can be shown that many neural network and curve-fitting algorithms output the maximum posteriori hypothesis, as does the naïve Bayesian classifier.

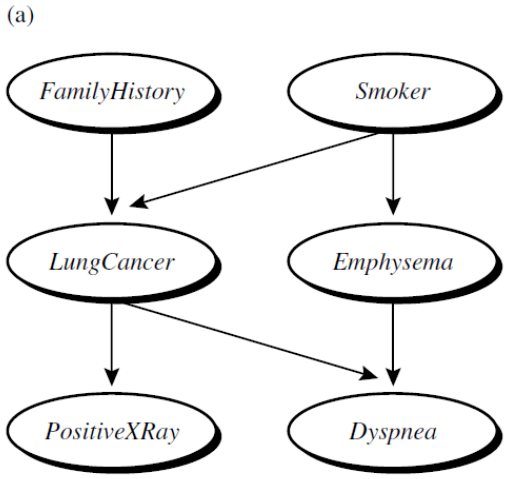
"*What if I encounter probability values of zero?*" Recall that in Equation (6.12), we estimate $P(\boldsymbol{X}|C_i)$ as the product of the probabilities $P(x_1|C_i), P(x_2|C_i), \ldots, P(x_n|C_i)$, based on the assumption of class conditional independence. These probabilities can be estimated from the training tuples (step 4). We need to compute $P(\boldsymbol{X}|C_i)$ for *each* class ($i = 1, 2, \ldots, m$) in order to find the class C_i for which $P(\boldsymbol{X}|C_i)P(C_i)$ is the maximum (step 5). Let's consider this calculation. For each attribute-value pair (i.e., $A_k = x_k$, for $k = 1, 2, \ldots, n$) in tuple $\boldsymbol{X}$, we need to count the number of tuples having that attribute-value pair, per class (i.e., per C_i, for $i = 1, \ldots, m$). In Example 6.4, we have two classes ($m = 2$), namely *buys_computer* = *yes* and *buys_computer* = *no*. Therefore, for the attribute-value pair *student* = *yes* of $\boldsymbol{X}$, say, we need two counts—the number of customers who are students and for which *buys_computer* = *yes* (which contributes to $P(\boldsymbol{X}|buys\_computer = yes)$) and the number of customers who are students and for which *buys_computer* = *no* (which contributes to $P(\boldsymbol{X}|buys\_computer = no)$). But what if, say, there are no training tuples representing students for the class *buys_computer* = *no*, resulting in $P(student = yes|buys\_computer = no) = 0$? In other words, what happens if we should end up with a probability value of zero for some $P(x_k|C_i)$? Plugging this zero value into Equation (6.12) would return a zero probability for $P(\boldsymbol{X}|C_i)$, even though, without the zero probability, we may have ended up with a high probability, suggesting that $\boldsymbol{X}$ belonged to class C_i! A zero probability cancels the effects of all of the other (posteriori) probabilities (on C_i) involved in the product.

Bayesian Belief Networks: The naïve Bayesian classifier makes the assumption of class conditional independence, that is, given the class label of a tuple, the values of the attributes are assumed to be conditionally independent of one another. This simplifies computation. When the assumption holds true, then the naïve Bayesian classifier is the most accurate

in comparison with all other classifiers. In practice, however, dependencies can exist between variables.

Bayesian belief networks specify joint conditional probability distributions. They allow class conditional independencies to be defined between subsets of variables. They provide a graphical model of causal relationships, on which learning can be performed. Trained Bayesian belief networks can be used for classification. Bayesian belief networks are also known as belief networks, Bayesian networks, and probabilistic networks. For brevity, we will refer to them as belief networks.

A belief network is defined by two components—a directed acyclic graph and a set of conditional probability tables (Figure 6.11). Each node in the directed acyclic graph represents a random variable. The variables may be discrete or continuous-valued. They may correspond to actual attributes given in the data or to "hidden variables" believed to form a relationship (e.g., in the case of medical data, a hidden variable may indicate a syndrome, representing a number of symptoms that, together, characterize a specific disease). Each arc represents a probabilistic dependence. If an arc is drawn from a node Y to a node Z, then Y is a parent or immediate predecessor of Z, and Z is a descendant of Y. Each variable is conditionally independent of its non-descendants in the graph, given its parents.



(b)

	FH, S	*FH, ~S*	*~FH, S*	*~FH, ~S*
LC	0.8	0.5	0.7	0.1
~LC	0.2	0.5	0.3	0.9

Figure 6.11 A simple Bayesian belief network: (a) A proposed causal model, represented by a directed acyclic graph. (b) The conditional probability table for the values of the variable *LungCancer (LC)* showing each possible combination of the values of its parent nodes, *FamilyHistory (FH)* and *Smoker (S)*. Figure is adapted from [RBKK95].

Figure 6.11 is a simple belief network, adapted from [RBKK95] for six Boolean variables. The arcs in Figure 6.11(a) allow a representation of causal knowledge. For example, having lung cancer is influenced by a person's family history of lung cancer, as well as whether or not the person is a smoker. Note that the variable Positive X-Ray is independent of whether the patient has a family history of lung cancer or is a smoker, given that we know the patient has lung cancer. In other words, once we know the outcome of the variable Lung Cancer, then the variables Family History and Smoker do not provide any additional information regarding Positive X-Ray. The arcs also show that the variable Lung Cancer is conditionally independent of Emphysema, given its parents, Family History and Smoker.

A belief network has one **conditional probability table** (**CPT**) for each variable. The CPT for a variable Y specifies the conditional distribution $P(Y|Parents(Y))$, where $Parents(Y)$ are the parents of Y. Figure 6.11(b) shows a CPT for the variable *LungCancer*. The conditional probability for each known value of *LungCancer* is given for each possible combination of values of its parents. For instance, from the upper leftmost and bottom rightmost entries, respectively, we see that

$$P(LungCancer = yes \mid FamilyHistory = yes, Smoker = yes) = 0.8$$
$$P(LungCancer = no \mid FamilyHistory = no, Smoker = no) = 0.9$$

Let $\boldsymbol{X} = (x_1, \ldots, x_n)$ be a data tuple described by the variables or attributes $Y_1, \ldots, Y_n$, respectively. Recall that each variable is conditionally independent of its nondescendants in the network graph, given its parents. This allows the network to provide a complete representation of the existing joint probability distribution with the following equation:

$$P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i|Parents(Y_i)), \qquad (6.16)$$

where $P(x_1, \ldots, x_n)$ is the probability of a particular combination of values of $\boldsymbol{X}$, and the values for $P(x_i|Parents(Y_i))$ correspond to the entries in the CPT for Y_i.

A node within the network can be selected as an "output" node, representing a class label attribute. There may be more than one output node. Various algorithms for learning can be applied to the network. Rather than returning a single class label, the classification process can return a probability distribution that gives the probability of each class.

RULE-BASED CLASSIFICATION

In this section, we look at rule-based classifiers, where the learned model is represented as a set of IF-THEN rules. We first examine how such rules are used for classification.

We then study ways in which they can be generated, either from a decision tree or directly from the training data using a sequential covering algorithm.

Using IF-THEN Rules for Classification: Rules are a good way of representing information or bits of knowledge. A rule-based classifier uses a set of IF-THEN rules for classification. An IF-THEN rule is an expression of the form

IF *condition* THEN *conclusion*.

An example is rule *R*1,

R1: IF *age* = *youth* AND *student* = *yes* THEN *buys_computer* = *yes*.

The "IF"-part (or left-hand side) of a rule is known as the **rule antecedent** or **precondition**. The "THEN"-part (or right-hand side) is the **rule consequent**. In the rule antecedent, the condition consists of one or more *attribute tests* (such as *age* = *youth*, and *student* = *yes*) that are logically ANDed. The rule's consequent contains a class prediction (in this case, we are predicting whether a customer will buy a computer). *R*1 can also be written as

R1: (*age* = *youth*) ∧ (*student* = *yes*) ⇒ (*buys_computer* = *yes*).

If the condition (that is, all of the attribute tests) in a rule antecedent holds true for a given tuple, we say that the rule antecedent is **satisfied** (or simply, that the rule is satisfied) and that the rule **covers** the tuple.

A rule *R* can be assessed by its coverage and accuracy. Given a tuple, ***X***, from a class-labeled data set, *D*, let n_{covers} be the number of tuples covered by *R*; $n_{correct}$ be the number of tuples correctly classified by *R*; and |*D*| be the number of tuples in *D*. We can define the **coverage** and **accuracy** of *R* as

$$coverage(R) = \frac{n_{covers}}{|D|} \tag{6.19}$$

$$accuracy(R) = \frac{n_{correct}}{n_{covers}}. \tag{6.20}$$

That is, a rule's coverage is the percentage of tuples that are covered by the rule (i.e., whose attribute values hold true for the rule's antecedent). For a rule's accuracy, we look at the tuples that it covers and see what percentage of them the rule can correctly classify.

Let's see how we can use rule-based classification to predict the class label of a given tuple, $\boldsymbol{X}$. If a rule is satisfied by $\boldsymbol{X}$, the rule is said to be **triggered**. For example, suppose we have

$$\boldsymbol{X}= (age = youth, income = medium, student = yes, credit\_rating = fair).$$

We would like to classify $\boldsymbol{X}$ according to *buys_computer*. $\boldsymbol{X}$ satisfies $R1$, which triggers the rule.

If $R1$ is the only rule satisfied, then the rule **fires** by returning the class prediction for $\boldsymbol{X}$. Note that triggering does not always mean firing because there may be more than one rule that is satisfied! If more than one rule is triggered, we have a potential problem. What if they each specify a different class? Or what if no rule is satisfied by $\boldsymbol{X}$?

We tackle the first question. If more than one rule is triggered, we need a **conflict resolution strategy** to figure out which rule gets to fire and assign its class prediction to $\boldsymbol{X}$. There are many possible strategies. We look at two, namely *size ordering* and *rule ordering*.

The **size ordering** scheme assigns the highest priority to the triggering rule that has the "toughest" requirements, where toughness is measured by the rule antecedent *size*. That is, the triggering rule with the most attribute tests is fired.

The **rule ordering** scheme prioritizes the rules beforehand. The ordering may be *class-based* or *rule-based*. With **class-based ordering**, the classes are sorted in order of decreasing "importance," such as by decreasing *order of prevalence*. That is, all of the rules for the most prevalent (or most frequent) class come first, the rules for the next prevalent class come next, and so on. Alternatively, they may be sorted based on the misclassification cost per class. Within each class, the rules are not ordered—they don't have to be because they all predict the same class (and so there can be no class conflict!). With **rule-based ordering**, the rules are organized into one long priority list, according to some measure of rule quality such as accuracy, coverage, or size (number of attribute tests in the rule antecedent), or based on advice from domain experts. When rule ordering is used, the rule set is known as a **decision list**. With rule ordering, the triggering rule that appears earliest in the list has highest priority, and so it gets to fire its class prediction. Any other rule that satisfies $\boldsymbol{X}$ is ignored. Most rule-based classification systems use a class-based rule-ordering strategy.

Note that in the first strategy, overall the rules are *unordered*. They can be applied in any order when classifying a tuple. That is, a disjunction (logical OR) is implied between each of the rules. Each rule represents a stand-alone nugget or piece of knowledge. This is in contrast to the rule-ordering (decision list) scheme for which rules must be applied in the prescribed order so as to avoid conflicts. Each rule in a decision list implies the negation of the rules that come before it in the list. Hence, rules in a decision list are more difficult to interpret.

Now that we have seen how we can handle conflicts, let's go back to the scenario where there is no rule satisfied by X. How, then, can we determine the class label of X? In this case, a fallback or **default rule** can be set up to specify a default class, based on a training set. This may be the class in majority or the majority class of the tuples that were not covered by any rule. The default rule is evaluated at the end, if and only if no other rule covers X. The condition in the default rule is empty. In this way, the rule fires when no other rule is satisfied.

Rule Extraction from a Decision Tree: Decision tree classifiers are a popular method of classification—it is easy to understand how decision trees work and they are known for their accuracy. Decision trees can become large and difficult to interpret. In this subsection, we look at how to build a rulebased classifier by extracting IF-THEN rules from a decision tree. In comparison with a decision tree, the IF-THEN rules may be easier for humans to understand, particularly if the decision tree is very large.

To extract rules from a decision tree, one rule is created for each path from the root to a leaf node. Each splitting criterion along a given path is logically ANDed to form the rule antecedent ("IF" part). The leaf node holds the class prediction, forming the rule consequent ("THEN" part).

A disjunction (logical OR) is implied between each of the extracted rules. Because the rules are extracted directly from the tree, they are mutually exclusive and exhaustive. By mutually exclusive, this means that we cannot have rule conflicts here because no two rules will be triggered for the same tuple. (We have one rule per leaf, and any tuple can map to only one leaf.) By exhaustive, there is one rule for each possible attribute-value combination, so that this set of rules does not require a default rule. Therefore, the order of the rules does not matter—they are unordered.

Since we end up with one rule per leaf, the set of extracted rules is not much simpler than the corresponding decision tree! The extracted rules may be even more difficult to interpret than the original trees in some cases. As an example, Figure 6.7 showed decision trees that suffer from subtree repetition and replication. The resulting set of rules extracted can be large and difficult to follow, because some of the attribute tests may be irrelevant or redundant.

So, the plot thickens. Although it is easy to extract rules from a decision tree, we may need to do some more work by pruning the resulting rule set.

"How can we prune the rule set?" For a given rule antecedent, any condition that does not improve the estimated accuracy of the rule can be pruned (i.e., removed), thereby generalizing the rule. C4.5 extracts rules from an unpruned tree, and then prunes the rules using a pessimistic approach similar to its tree pruning method. The training tuples and their associated class labels are used to estimate rule accuracy. However, because this would result in an optimistic estimate, alternatively, the estimate is adjusted to compensate for the bias, resulting in a pessimistic estimate. In addition, any rule that does not contribute to the overall accuracy of the entire rule set can also be pruned.

Other problems arise during rule pruning, however, as the rules will no longer be mutually exclusive and exhaustive. For conflict resolution, C4.5 adopts a class-based ordering scheme. It groups all rules for a single class together, and then determines a ranking of these class rule sets. Within a rule set, the rules are not ordered. C4.5 orders the class rule sets so as to minimize the number of false-positive errors (i.e., where a rule predicts a class, C, but the actual class is not C). The class rule set with the least number of false positives is examined first. Once pruning is complete, a final check is done to remove any duplicates. When choosing a default class, C4.5 does not choose the majority class, because this class will likely have many rules for its tuples. Instead, it selects the class that contains the most training tuples that were not covered by any rule.

Rule Induction Using a Sequential Covering Algorithm: IF-THEN rules can be extracted directly from the training data (i.e., without having to generate a decision tree first) using a sequential covering algorithm. The name comes from the notion that the rules are learned sequentially (one at a time), where each rule for a given class will ideally cover many of the tuples of that class (and hopefully none of the tuples of other classes). Sequential covering algorithms are the most widely used approach to mining disjunctive sets of classification rules, and form the topic of this subsection. Note that in a newer alternative approach, classification rules can be generated using associative

classification algorithms, which search for attribute-value pairs that occur frequently in the data. These pairs may form association rules, which can be analyzed and used in classification.

There are many sequential covering algorithms. Popular variations include AQ, CN2, and the more recent, RIPPER. The general strategy is as follows. Rules are learned one at a time. Each time a rule is learned, the tuples covered by the rule are removed, and the process repeats on the remaining tuples. This sequential learning of rules is in contrast to decision tree induction. Because the path to each leaf in a decision tree corresponds to a rule, we can consider decision tree induction as learning a set of rules simultaneously.

A basic sequential covering algorithm is shown in Figure 6.12. Here, rules are learned for one class at a time. Ideally, when learning a rule for a class, C_i, we would like the rule to cover all (or many) of the training tuples of class C and none (or few) of the tuples from other classes. In this way, the rules learned should be of high accuracy. The rules need not necessarily be of high coverage. This is because we can have more than one rule for a class, so that different rules may cover different tuples within the same class.

The process continues until the terminating condition is met, such as when there are no more training tuples or the quality of a rule returned is below a user-specified threshold. The Learn_One_Rule procedure finds the "best" rule for the current class, given the current set of training tuples.

Algorithm: Sequential covering. Learn a set of IF-THEN rules for classification.

Input:

- *D*, a data set class-labeled tuples;
- *Att_vals*, the set of all attributes and their possible values.

Output: A set of IF-THEN rules.

Method:

```
(1)  Rule_set = {}; // initial set of rules learned is empty
(2)  for each class c do
(3)      repeat
(4)          Rule = Learn_One_Rule(D, Att_vals, c);
```

```
(5)             remove tuples covered by Rule from D;
(6)         until terminating condition;
(7)         Rule_set = Rule_set + Rule; // add new rule to rule set
(8)     endfor
(9)     return Rule_Set;
```

Figure 6.12 Basic sequential covering algorithm.

"How are rules learned?" Typically, rules are grown in a *general-to-specific* manner (Figure 6.13). We can think of this as a beam search, where we start off with an empty rule and then gradually keep appending attribute tests to it. We append by adding the attribute test as a logical conjunct to the existing condition of the rule antecedent. Suppose our training set, D, consists of loan application data. Attributes regarding each applicant include their age, income, education level, residence, credit rating, and the term of the loan. The classifying attribute is *loan_decision*, which indicates whether a loan is accepted (considered safe) or rejected (considered risky). To learn a rule for the class "accept," we start off with the most general rule possible, that is, the condition of the rule antecedent is empty. The rule is:

IF THEN *loan_decision* = *accept*.

We then consider each possible attribute test that may be added to the rule. These can be derived from the parameter *Att_vals*, which contains a list of attributes with their associated values. For example, for an attribute-value pair (att, val), we can consider

attribute tests such as $att = val$, $att \leq val$, $att > val$, and so on. Typically, the training data will contain many attributes, each of which may have several possible values. Finding an optimal rule set becomes computationally explosive. Instead, *Learn_One_Rule* adopts a greedy depth-first strategy. Each time it is faced with adding a new attribute test (conjunct) to the current rule, it picks the one that most improves the rule quality, based on the training samples. We will say more about rule quality measures in a minute. For the moment, let's say we use rule accuracy as our quality measure. Getting back to our example with Figure 6.13, suppose *Learn_One_Rule* finds that the attribute test *income* = *high* best improves the accuracy of our current (empty) rule. We append it to the condition, so that the current rule becomes

IF *income* = *high* THEN *loan_decision* = *accept*.

Each time we add an attribute test to a rule, the resulting rule should cover more of the "accept" tuples. During the next iteration, we again consider the possible attribute tests and end up selecting *credit_rating* = *excellent*. Our current rule grows to become

IF *income* = *high* AND *credit_rating* = *excellent* THEN *loan_decision* = *accept*.

The process repeats, where at each step, we continue to greedily grow rules until the resulting rule meets an acceptable quality level.

Greedy search does not allow for backtracking. At each step, we heuristically add what appears to be the best choice at the moment. What if we unknowingly made a poor choice along the way? To lessen the chance of this happening, instead of selecting the best attribute test to append to the current rule, we can select the best k attribute tests. In this way, we perform a beam search of width k wherein we maintain the k best candidates overall at each step, rather than a single best candidate.

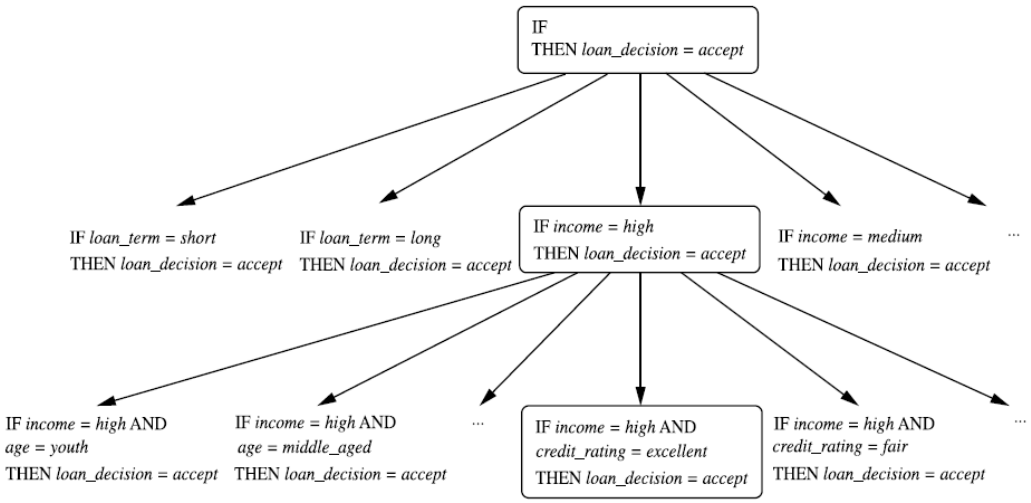


Figure 6.13 A general-to-specific search through rule space.

Rule Quality Measures: Learn_One_Rule needs a measure of rule quality. Every time it considers an attribute test, it must check to see if appending such a test to the current rule's condition will result in an improved rule.

Coverage on its own is not useful either—for a given class we could have a rule that covers many tuples, most of which belong to other classes! Thus, we seek other measures for evaluating rule quality, which may integrate aspects of accuracy and coverage. Here we will look at a few, namely entropy, another based on information gain, and a statistical test that considers coverage. For our discussion, suppose we are learning rules for the class c. Our current rule is R: IF condition THEN class = c. We want to see if logically ANDing a given

attribute test to condition would result in a better rule. We call the new condition, condition0, where R0: IF condition0 THEN class = c is our potential new rule. In other words, we want to see if R0 is any better than R.

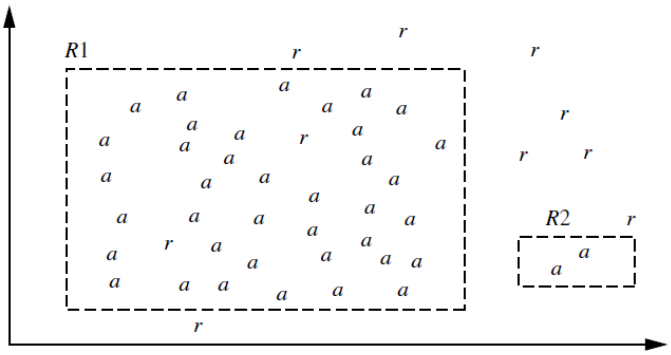


Figure 6.14 Rules for the class *loan_decision = accept*, showing *accept* (*a*) and *reject* (*r*) tuples.

Another measure is based on information gain and was proposed in FOIL (First Order Inductive Learner), a sequential covering algorithm that learns first-order logic rules. Learning first-order rules is more complex because such rules contain variables, whereas the rules we are concerned with in this section are propositional (i.e., variable-free).[7] In machine learning, the tuples of the class for which we are learning rules are called *positive* tuples, while the remaining tuples are *negative*. Let *pos* (*neg*) be the number of positive (negative) tuples covered by R. Let pos' (neg') be the number of positive (negative) tuples covered by R'. FOIL assesses the information gained by extending *condition* as

$$FOIL\_Gain = pos' \times \left(\log_2 \frac{pos'}{pos' + neg'} - log_2 \frac{pos}{pos + neg} \right). \tag{6.21}$$

It favors rules that have high accuracy and cover many positive tuples.

We can also use a statistical test of significance to determine if the apparent effect of a rule is not attributed to chance but instead indicates a genuine correlation between attribute values and classes. The test compares the observed distribution among classes of tuples covered by a rule with the expected distribution that would result if the rule made predictions at random. We want to assess whether any observed differences between these two distributions may be attributed to chance. We can use the **likelihood ratio statistic**,

$$Likelihood\_Ratio = 2 \sum_{i=1}^{m} f_i \log \left(\frac{f_i}{e_i} \right), \tag{6.22}$$

where m is the number of classes. For tuples satisfying the rule, f_i is the observed frequency of each class i among the tuples. e_i is what we would expect the frequency of each class i to be if the rule made random predictions. The statistic has a χ^2 distribution with $m-1$ degrees of freedom. The higher the likelihood ratio is, the more likely that there is a *significant* difference in the number of correct predictions made by our rule in comparison with a "random guessor." That is, the performance of our rule is not due to chance. The ratio helps identify rules with insignificant coverage.

CN2 uses entropy together with the likelihood ratio test, while FOIL's information gain is used by RIPPER.

Rule Pruning: Learn_One_Rule does not employ a test set when evaluating rules. Assessments of rule quality as described above are made with tuples from the original training data.

Such assessment is optimistic because the rules will likely overfit the data. That is, the rules may perform well on the training data, but less well on subsequent data. To compensate for this, we can prune the rules. A rule is pruned by removing a conjunct (attribute test). We choose to prune a rule, R, if the pruned version of R has greater quality, as assessed on an independent set of tuples. As in decision tree pruning, we refer to this set as a pruning set. Various pruning strategies can be used, such as the pessimistic pruning approach described in the previous section. FOIL uses a simple yet effective method. Given a rule, R,

$$FOIL\_Prune(R) = \frac{pos - neg}{pos + neg}, \qquad (6.23)$$

where pos and neg are the number of positive and negative tuples covered by R, respectively. This value will increase with the accuracy of R on a pruning set. Therefore, if the FOIL_Prune value is higher for the pruned version of R, then we prune R. By convention, RIPPER starts with the most recently added conjunct when considering pruning. Conjuncts are pruned one at a time as long as this results in an improvement.

LAZY LEARNER [OR] LEARNING FROM YOUR NEIGHBORS

The classification methods discussed so far in this chapter—decision tree induction, Bayesian classification, rule-based classification, classification by backpropagation, support vector machines, and classification based on

association rule mining—are all examples of eager learners. Eager learners, when given a set of training tuples, will construct a generalization (i.e., classification) model before receiving new (e.g., test) tuples to classify. We can think of the learned model as being ready and eager to classify previously unseen tuples.

Imagine a contrasting lazy approach, in which the learner instead waits until the last minute before doing any model construction in order to classify a given test tuple. That is, when given a training tuple, a lazy learner simply stores it (or does only a little minor processing) and waits until it is given a test tuple. Only when it sees the test tuple does it perform generalization in order to classify the tuple based on its similarity to the stored training tuples. Unlike eager learning methods, lazy learners do less work when a training tuple is presented and more work when making a classification or prediction. Because lazy learners store the training tuples or "instances," they are also referred to as instance based learners, even though all learning is essentially based on instances.

When making a classification or prediction, lazy learners can be computationally expensive. They require efficient storage techniques and are well-suited to implementation on parallel hardware. They offer little explanation or insight into the structure of the data. Lazy learners, however, naturally support incremental learning. They are able to model complex decision spaces having hyperpolygonal shapes that may not be as easily describable by other learning algorithms (such as hyper-rectangular shapes modelled by decision trees). In this section, we look at two examples of lazy learners: k-nearest neighbor classifiers and case-based reasoning classifiers.

k-Nearest-Neighbor Classifiers: The k-nearest-neighbor method was first described in the early 1950s. The method is labor intensive when given large training sets, and did not gain popularity until the 1960s when increased computing power became available. It has since been widely used in the area of pattern recognition.

Nearest-neighbor classifiers are based on learning by analogy, that is, by comparing a given test tuple with training tuples that are similar to it. The

training tuples are described by n attributes. Each tuple represents a point in an n-dimensional space. In this way, all of the training tuples are stored in an n-dimensional pattern space. When given an unknown tuple, a k-nearest-neighbor classifier searches the pattern space for the k training tuples that are closest to the unknown tuple. These k training tuples are the k "nearest neighbors" of the unknown tuple.

"Closeness" is defined in terms of a distance metric, such as Euclidean distance. The Euclidean distance between two points or tuples, say, $\boldsymbol{X}_1 = (x_{11}, x_{12}, \ldots, x_{1n})$ and $\boldsymbol{X}_2 = (x_{21}, x_{22}, \ldots, x_{2n})$, is

$$dist(\boldsymbol{X_1}, \boldsymbol{X_2}) = \sqrt{\sum_{i=1}^{n} (x_{1i} - x_{2i})^2}. \tag{6.45}$$

In other words, for each numeric attribute, we take the difference between the corresponding values of that attribute in tuple $\boldsymbol{X_1}$ and in tuple $\boldsymbol{X_2}$, square this difference, and accumulate it. The square root is taken of the total accumulated distance count. Typically, we normalize the values of each attribute before using Equation (6.45). This helps prevent attributes with initially large ranges (such as *income*) from outweighing attributes with initially smaller ranges (such as binary attributes). Min-max normalization, for example, can be used to transform a value v of a numeric attribute A to v' in the range [0, 1] by computing

$$v' = \frac{v - min_A}{max_A - min_A}, \tag{6.46}$$

where min_A and max_A are the minimum and maximum values of attribute A. Chapter 2 describes other methods for data normalization as a form of data transformation.

For k-nearest-neighbor classification, the unknown tuple is assigned the most common class among its k nearest neighbors. When $k = 1$, the unknown tuple is assigned the class of the training tuple that is closest to it in pattern space. Nearest-neighbor classifiers can also be used for prediction, that is, to return a real-valued prediction for a given unknown tuple. In this case, the classifier returns the average value of the real-valued labels associated with the k nearest neighbors of the unknown tuple.

"But how can distance be computed for attributes that not numeric, but categorical, such as color?" The above discussion assumes that the attributes used to describe the tuples are all numeric. For categorical attributes, a simple method is to compare the corresponding value of the attribute in tuple $\boldsymbol{X_1}$ with that in tuple $\boldsymbol{X_2}$. If the two are identical (e.g., tuples $\boldsymbol{X_1}$ and $\boldsymbol{X_2}$ both have the color blue), then the difference between the two is taken as 0. If the two are different (e.g., tuple $\boldsymbol{X_1}$ is blue but tuple $\boldsymbol{X_2}$ is red), then the difference is considered to be 1. Other methods may incorporate more sophisticated schemes for differential grading (e.g., where a larger difference score is assigned, say, for blue and white than for blue and black).

"What about missing values?" In general, if the value of a given attribute A is missing in tuple $\boldsymbol{X_1}$ and/or in tuple $\boldsymbol{X_2}$, we assume the maximum possible difference. Suppose that each of the attributes have been mapped to the range [0, 1]. For categorical attributes, we take the difference value to be 1 if either one or both of the corresponding values of A are missing. If A is numeric and missing from both tuples $\boldsymbol{X_1}$ and $\boldsymbol{X_2}$, then the difference is also taken to be 1. If only one value is missing and the other (which we'll call v') is present and normalized, then we can take the difference to be either $|1 - v'|$ or $|0 - v'|$ (i.e., $1 - v'$ or v'), whichever is greater.

"How can I determine a good value for k, the number of neighbors?" This can be determined experimentally. Starting with $k = 1$, we use a test set to estimate the error rate of the classifier. This process can be repeated each time by incrementing k to allow for one more neighbor. The k value that gives the minimum error rate may be selected. In general, the larger the number of training tuples is, the larger the value of k will be (so that classification and prediction decisions can be based on a larger portion of the stored tuples). As the number of training tuples approaches infinity and $k = 1$, the error rate can be no worse then twice the Bayes error rate (the latter being the theoretical minimum). If k also approaches infinity, the error rate approaches the Bayes error rate.

Nearest-neighbor classifiers use distance-based comparisons that intrinsically assign equal weight to each attribute. They therefore can suffer from poor accuracy when given noisy or irrelevant attributes. The method, however, has been modified to incorporate attribute weighting and the pruning of noisy data tuples. The choice of a distance metric can be critical. The Manhattan (city block) distance (Section 7.2.1), or other distance measurements, may also be used.

Nearest-neighbor classifiers can be extremely slow when classifying test tuples. If D is a training database of $|D|$ tuples and $k = 1$, then $O(|D|)$ comparisons are required in order to classify a given test tuple. By presorting and arranging the stored tuples

into search trees, the number of comparisons can be reduced to $O(log(|D|))$. Parallel implementation can reduce the running time to a constant, that is $O(1)$, which is independent of $|D|$. Other techniques to speed up classification time include the use of *partial distance* calculations and *editing* the stored tuples. In the **partial distance** method, we compute the distance based on a subset of the n attributes. If this distance exceeds a threshold, then further computation for the given stored tuple is halted, and the process moves on to the next stored tuple. The **editing** method removes training tuples that prove useless. This method is also referred to as **pruning** or **condensing** because it reduces the total number of tuples stored.

Case-Based Reasoning (CBR): Case-based reasoning (CBR) classifiers use a database of problem solutions to solve new problems. Unlike nearest-neighbor classifiers, which store training tuples as points in Euclidean space, CBR stores the tuples or "cases" for problem solving as complex symbolic descriptions. Business applications of CBR include problem resolution for

customer service help desks, where cases describe product-related diagnostic problems. CBR has also been applied to areas such as engineering and law, where cases are either technical designs or legal rulings, respectively. Medical education is another area for CBR, where patient case histories and treatments are used to help diagnose and treat new patients.

When given a new case to classify, a case-based reasoner will first check if an identical training case exists. If one is found, then the accompanying solution to that case is returned. If no identical case is found, then the case-based reasoner will search for training cases having components that are similar to those of the new case. Conceptually, these training cases may be considered as neighbors of the new case. If cases are represented as graphs, this involves searching for subgraphs that are similar to subgraphs within the new case. The case-based reasoner tries to combine the solutions of the neighboring training cases in order to propose a solution for the new case. If incompatibilities arise with the individual solutions, then backtracking to search for other solutions may be necessary. The case-based reasoner may employ background knowledge and problem-solving strategies in order to propose a feasible combined solution.

Challenges in case-based reasoning include finding a good similarity metric (e.g., for matching subgraphs) and suitable methods for combining solutions. Other challenges include the selection of salient features for indexing training cases and the development of efficient indexing techniques. A trade-off between accuracy and efficiency evolves as the number of stored cases becomes very large. As this number increases, the case-based reasoner becomes more intelligent. After a certain point, however, the efficiency of the system will suffer as the time required to search for and process relevant cases increases. As with nearest-neighbor classifiers, one solution is to edit the training database. Cases that are redundant or that have not proved useful may be discarded for the sake of improved performance. These decisions, however, are not clear-cut and their automation remains an active area of research.

*****

UNIT – IV

CLUSTERING AND APPLICATIONS

CLUSTER ANALYSIS

What is Cluster Analysis?

The process of grouping a set of physical or abstract objects into classes of similar objects is called clustering. A cluster is a collection of data objects that are similar to one another within the same cluster and are dissimilar to the objects in other clusters. A cluster of data objects can be treated collectively as one group and so may be considered as a form of data compression.

Explanation/Overview: The process of grouping a set of physical or abstract objects into classes of similar objects is called clustering. A cluster is a collection of data objects that are similar to one another within the same cluster and are dissimilar to the objects in other clusters. A cluster of data objects can be treated collectively as one group and so may be considered as a form of data compression. Although classification is an effective means for distinguishing groups or classes of objects, it requires the often costly collection and labeling of a large set of training tuples or patterns, which the classifier uses to model each group. It is often more desirable to proceed in the reverse direction: First partition the set of data into groups based on data similarity (e.g., using clustering), and then assign labels to the relatively small number of groups. Additional advantages of such a clustering-based process are that it is adaptable to changes and helps single out useful features that distinguish different groups.

Cluster analysis is an important human activity. Early in childhood, we learn how to distinguish between cats and dogs, or between animals and plants, by continuously improving subconscious clustering schemes. By automated clustering, we can identify dense and sparse regions in object space and, therefore, discover overall distribution patterns and interesting correlations among data attributes. Cluster analysis has been widely used in numerous applications, including market research, pattern recognition, data analysis, and image processing. In business, clustering can help marketers discover

distinct groups in their customer bases and characterize customer groups based on purchasing patterns. In biology, it can be used to derive plant and animal taxonomies, categorize genes with similar functionality, and gain insight into structures inherent in populations. Clustering may also help in the identification of areas of similar land use in an earth observation database and in the identification of groups of houses in a city according to house type, value, and geographic location, as well as the identification of groups of automobile insurance policy holders with a high average claim cost. It can also be used to help classify documents on the Web for information discovery.

Clustering is also called data segmentation in some applications because clustering partitions large data sets into groups according to their similarity. Clustering can also be used for outlier detection, where outliers (values that are “far away” from any cluster) may be more interesting than common cases. Applications of outlier detection include the detection of credit card fraud and the monitoring of criminal activities in electronic commerce. For example, exceptional cases in credit card transactions, such as very expensive and frequent purchases, may be of interest as possible fraudulent activity. As a data mining function, cluster analysis can be used as a stand-alone tool to gain insight into the distribution of data, to observe the characteristics of each cluster, and to focus on a particular set of clusters for further analysis. Alternatively, it may serve as a pre-processing step for other algorithms, such as characterization, attribute subset selection, and classification, which would then operate on the detected clusters and the selected attributes or features.

Data clustering is under vigorous development. Contributing areas of research include data mining, statistics, machine learning, spatial database technology, biology, and marketing. Owing to the huge amounts of data collected in databases, cluster analysis has recently become a highly active topic in data mining research.

As a branch of statistics, cluster analysis has been extensively studied for many years, focusing mainly on distance-based cluster analysis. Cluster analysis tools based on k-means, k-medoids, and several other methods have also been built into many statistical analysis software packages or systems, such as S-Plus, SPSS, and SAS. In machine learning, clustering is an example of unsupervised learning. Unlike classification, clustering and unsupervised learning do not rely on predefined classes and class-labeled training examples. For this reason, clustering is a form of learning by observation, rather than learning by examples. In data mining, efforts have focused on finding methods for efficient and effective cluster analysis in large databases. Active themes of research focus on the scalability of clustering methods, the effectiveness of methods for clustering complex shapes and types of data, high-dimensional clustering techniques, and methods for clustering mixed numerical and categorical data in large databases.

Points to Remember:

1) A cluster of data objects can be treated as one group.
2) While doing cluster analysis, we first partition the set of data into groups based on data similarity and then assign the labels to the groups.
3) The main advantage of clustering over classification is that, it is adaptable to changes and helps single out useful features that distinguish different groups.

Requirements of Clustering in Data Mining: Clustering is a challenging field of research in which its potential applications pose their own special requirements. The following are typical requirements of clustering in data mining:

1) **Scalability:** Many clustering algorithms work well on small data sets containing fewer than several hundred data objects; however, a large database may contain millions of objects. Clustering on a sample of a given large data set may lead to biased results. Highly scalable clustering algorithms are needed.

2) **Ability to deal with different types of attributes:** Many algorithms are designed to cluster interval-based (numerical) data. However, applications may require clustering other types of data, such as binary, categorical (nominal), and ordinal data, or mixtures of these data types.
3) **Discovery of clusters with arbitrary shape:** Many clustering algorithms determine clusters based on Euclidean or Manhattan distance measures. Algorithms based on such distance measures tend to find spherical clusters with similar size and density. However, a cluster could be of any shape. It is important to develop algorithms that can detect clusters of arbitrary shape.
4) **Minimal requirements for domain knowledge to determine input parameters:** Many clustering algorithms require users to input certain parameters in cluster analysis (such as the number of desired clusters). The clustering results can be quite sensitive to input parameters. Parameters are often difficult to determine, especially for data sets containing high-dimensional objects. This not only burdens users, but it also makes the quality of clustering difficult to control.
5) **Ability to deal with noisy data:** Most real-world databases contain outliers or missing, unknown, or erroneous data. Some clustering algorithms are sensitive to such data and may lead to clusters of poor quality.
6) **Incremental clustering and insensitivity to the order of input records:** Some clustering algorithms cannot incorporate newly inserted data (i.e., database updates) into existing clustering structures and, instead, must determine a new clustering from scratch. Some clustering algorithms are sensitive to the order of input data. That is, given a set of data objects, such an algorithm may return dramatically different clustering's depending on the order of presentation of the input objects. It is important to develop incremental clustering algorithms and algorithms that are insensitive to the order of input.
7) **High dimensionality:** A database or a data warehouse can contain several dimensions or attributes. Many clustering algorithms are good at handling low-dimensional data, involving only two to three

dimensions. Human eyes are good at judging the quality of clustering for up to three dimensions. Finding clusters of data objects in high dimensional space is challenging, especially considering that such data can be sparse and highly skewed.

8) **Constraint-based clustering:** Real-world applications may need to perform clustering under various kinds of constraints. Suppose that your job is to choose the locations for a given number of new automatic banking machines (ATMs) in a city. To decide upon this, you may cluster households while considering constraints such as the city's rivers and highway networks, and the type and number of customers per cluster. A challenging task is to find groups of data with good clustering behavior that satisfy specified constraints.
9) **Interpretability and usability:** Users expect clustering results to be interpretable, comprehensible, and usable. That is, clustering may need to be tied to specific semantic interpretations and applications. It is important to study how an application goal may influence the selection of clustering features and methods.

With these requirements in mind, our study of cluster analysis proceeds as follows. First, we study different types of data and how they can influence clustering methods. Second, we present a general categorization of clustering methods. We then study each clustering method in detail, including partitioning methods, hierarchical methods, density-based methods, grid-based methods, and model-based methods. We also examine clustering in high-dimensional space, constraint-based clustering, and outlier analysis.

Applications of Cluster Analysis:

1) Clustering analysis is broadly used in many applications such as market research, pattern recognition, data analysis, and image processing.
2) Clustering can also help marketers discover distinct groups in their customer base. And they can characterize their customer groups based on the purchasing patterns.

3) In the field of biology, it can be used to derive plant and animal taxonomies, categorize genes with similar functionalities and gain insight into structures inherent to populations.
4) Clustering also helps in identification of areas of similar land use in an earth observation database. It also helps in the identification of groups of houses in a city according to house type, value, and geographic location.
5) Clustering also helps in classifying documents on the web for information discovery.
6) Clustering is also used in outlier detection applications such as detection of credit card fraud.
7) As a data mining function, cluster analysis serves as a tool to gain insight into the distribution of data to observe characteristics of each cluster.

TYPES OF DATA IN CLUSTER ANALYSIS

In this section, we study the types of data that often occur in cluster analysis and how to pre-process them for such an analysis. Suppose that a data set to be clustered contains n objects, which may represent persons, houses, documents, countries, and so on. Main memory-based clustering algorithms typically operate on either of the following two data structures.

- **Data matrix** (or *object-by-variable structure*): This represents n objects, such as persons, with p **variables** (also called *measurements* or *attributes*), such as age, height, weight, gender, and so on. The structure is in the form of a relational table, or n-by-p matrix (n objects $\times p$ variables):

$$\begin{bmatrix} x_{11} & \cdots & x_{1f} & \cdots & x_{1p} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{i1} & \cdots & x_{if} & \cdots & x_{ip} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{n1} & \cdots & x_{nf} & \cdots & x_{np} \end{bmatrix} \quad (7.1)$$

- **Dissimilarity matrix** (or *object-by-object structure*): This stores a collection of proximities that are available for all pairs of n objects. It is often represented by an n-by-n table:

$$\begin{bmatrix} 0 & & & & \\ d(2,1) & 0 & & & \\ d(3,1) & d(3,2) & 0 & & \\ \vdots & \vdots & \vdots & & \\ d(n,1) & d(n,2) & \cdots & \cdots & 0 \end{bmatrix} \quad (7.2)$$

where $d(i, j)$ is the measured **difference** or **dissimilarity** between objects i and j. In general, $d(i, j)$ is a nonnegative number that is close to 0 when objects i and j are highly similar or "near" each other, and becomes larger the more they differ. Since $d(i, j) = d(j, i)$, and $d(i, i) = 0$, we have the matrix in (7.2). Measures of dissimilarity are discussed throughout this section.

The rows and columns of the data matrix represent different entities, while those of the dissimilarity matrix represent the same entity. Thus, the data matrix is often called a two-mode matrix, whereas the dissimilarity matrix is called a one-mode matrix. Many clustering algorithms operate on a dissimilarity matrix. If the data are presented in the form of a data matrix, it can first be transformed into a dissimilarity matrix before applying such clustering algorithms.

1) Interval-Scaled Variables:

"What are interval-scaled variables?"

Interval-scaled variables are continuous measurements of a roughly linear scale. Typical examples include weight and height, latitude and longitude coordinates (e.g., when clustering houses), and weather temperature. The measurement unit used can affect the clustering analysis. For example, changing measurement units from meters to inches for height, or from kilograms to pounds for weight, may lead to a very different clustering structure. In general, expressing a variable in smaller units will lead to a larger range for that variable, and thus a larger effect on the resulting clustering structure. To help avoid dependence on the choice of measurement units, the data should be standardized. Standardizing measurements attempts to give all variables an equal weight. This is particularly useful when given no prior knowledge of the data. However, in some applications, users

may intentionally want to give more weight to a certain set of variables than to others. For example, when clustering basketball player candidates, we may prefer to give more weight to the variable height.

"How can the data for a variable be standardized?"

To standardize measurements, one choice is to convert the original measurements to unitless variables. Given measurements for a variable f, this can be performed as follows:

1. Calculate the **mean absolute deviation**, s_f:

$$s_f = \frac{1}{n}(|x_{1f} - m_f| + |x_{2f} - m_f| + \cdots + |x_{nf} - m_f|), \tag{7.3}$$

where $x_{1f}, \ldots, x_{nf}$ are n measurements of f, and m_f is the *mean* value of f, that is, $m_f = \frac{1}{n}(x_{1f} + x_{2f} + \cdots + x_{nf})$.

2. Calculate the **standardized measurement**, or **z-score**:

$$z_{if} = \frac{x_{if} - m_f}{s_f}. \tag{7.4}$$

The mean absolute deviation, s_f, is more robust to outliers than the standard deviation, σ_f. When computing the mean absolute deviation, the deviations from the mean (i.e., $|x_{if} - m_f|$) are not squared; hence, the effect of outliers is somewhat reduced. There are more robust measures of dispersion, such as the *median absolute deviation*. However, the advantage of using the mean absolute deviation is that the z-scores of outliers do not become too small; hence, the outliers remain detectable.

Standardization may or may not be useful in a particular application. Thus the choice of whether and how to perform standardization should be left to the user. After standardization, or without standardization in certain applications, the dissimilarity (or similarity) between the objects described by interval-scaled variables is typically computed based on the distance between each pair of objects. The most popular distance measure is Euclidean distance, which is defined as:

$$d(i, j) = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \cdots + (x_{in} - x_{jn})^2}, \quad (7.5)$$

where $i = (x_{i1}, x_{i2}, \ldots, x_{in})$ and $j = (x_{j1}, x_{j2}, \ldots, x_{jn})$ are two n-dimensional data objects.

Another well-known metric is **Manhattan (or city block) distance**, defined as

$$d(i, j) = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + \cdots + |x_{in} - x_{jn}|. \quad (7.6)$$

Both the Euclidean distance and Manhattan distance satisfy the following mathematic requirements of a distance function:

1. $d(i, j) \geq 0$: Distance is a nonnegative number.
2. $d(i, i) = 0$: The distance of an object to itself is 0.
3. $d(i, j) = d(j, i)$: Distance is a symmetric function.
4. $d(i, j) \leq d(i, h) + d(h, j)$: Going directly from object i to object j in space is no more than making a detour over any other object h (*triangular inequality*).

Minkowski distance is a generalization of both Euclidean distance and Manhattan distance. It is defined as

$$d(i, j) = (|x_{i1} - x_{j1}|^p + |x_{i2} - x_{j2}|^p + \cdots + |x_{in} - x_{jn}|^p)^{1/p}, \quad (7.7)$$

where p is a positive integer. Such a distance is also called L_p norm, in some literature. It represents the Manhattan distance when $p = 1$ (i.e., L_1 norm) and Euclidean distance when $p = 2$ (i.e., L_2 norm).

If each variable is assigned a weight according to its perceived importance, the **weighted Euclidean distance** can be computed as

$$d(i, j) = \sqrt{w_1|x_{i1} - x_{j1}|^2 + w_2|x_{i2} - x_{j2}|^2 + \cdots + w_m|x_{in} - x_{jn}|^2}. \quad (7.8)$$

Weighting can also be applied to the Manhattan and Minkowski distances.

2) Binary Variables: Let us see how to compute the dissimilarity between objects described by either symmetric or asymmetric binary variables.

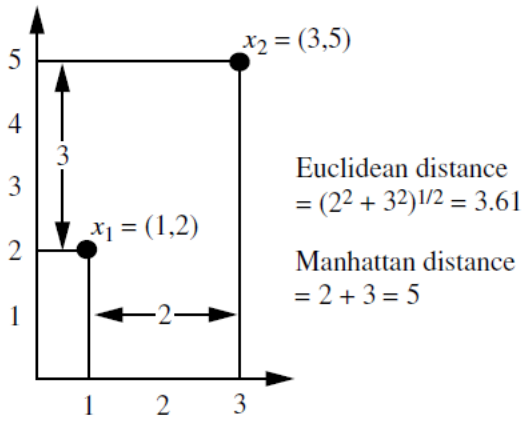


Figure 7.1 Euclidean and Manhattan distances between two objects.

"What is binary variables?"

A binary variable has only two states: 0 or 1, where 0 means that the variable is absent, and 1 means that it is present. Given the variable smoker describing a patient, for instance, 1 indicates that the patient smokes, while 0 indicates that the patient does not. Treating binary variables as if they are interval-scaled can lead to misleading clustering results. Therefore, methods specific to binary data are necessary for computing dissimilarities.

"So, how can we compute the dissimilarity between two binary variables?"

One approach involves computing a dissimilarity matrix from the given binary data. If all binary variables are thought of as having the same weight, we have the 2-by-2 contingency table of Table 7.1, where q is the number of variables that equal 1 for both objects i and j, r is the number of variables that equal 1 for object i but that are 0 for object j, s is the number of variables that equal 0 for object i but equal 1 for object j, and t is the number of variables that equal 0 for both objects i and j. The total number of variables is p, where p = q + r + s + t.

"What is the difference between symmetric and asymmetric binary variables?"

A binary variable is symmetric if both of its states are equally valuable and carry the same weight; that is, there is no preference on which outcome

should be coded as 0 or 1. One such example could be the attribute gender having the states male and female. Dissimilarity that is based on symmetric binary variables is called symmetric binary dissimilarity. Its dissimilarity (or distance) measure, defined in Equation (7.9), can be used to assess the dissimilarity between objects i and j.

$$d(i, j) = \frac{r+s}{q+r+s+t}. \tag{7.9}$$

A binary variable is asymmetric if the outcomes of the states are not equally important, such as the positive and negative outcomes of a disease test. By convention, we shall code the most important outcome, which is usually the rarest one, by 1 (e.g., HIV positive) and the other by 0 (e.g., HIV negative). Given two asymmetric binary variables, the agreement of two 1s (a positive match) is then considered more significant than that of two 0s (a negative match). Therefore, such binary variables are often considered "monary" (as if having one state). The dissimilarity based on such variables is called asymmetric binary dissimilarity, where the number of negative matches, t, is considered unimportant and thus is ignored in the computation, as shown in Equation (7.10).

$$d(i, j) = \frac{r+s}{q+r+s}. \tag{7.10}$$

Complementarily, we can measure the distance between two binary variables based on the notion of *similarity* instead of *dissimilarity*. For example, the **asymmetric binary similarity** between the objects i and j, or $sim(i, j)$, can be computed as,

$$sim(i, j) = \frac{q}{q+r+s} = 1 - d(i, j). \tag{7.11}$$

The coefficient sim(i, j) is called the Jaccard coefficient, which is popularly referenced in the literature.

Table 7.1 A contingency table for binary variables.

		object j		
		1	0	sum
	1	q	r	$q+r$
object i	0	s	t	$s+t$
	sum	$q+s$	$r+t$	p

3) Categorical, Ordinal, and Ratio-Scaled Variables: "How can we compute the dissimilarity between objects described by categorical, ordinal, and ratio-scaled variables?"

Categorical Variables

A **categorical variable** is a generalization of the binary variable in that it can take on more than two states. For example, *map_color* is a categorical variable that may have, say, five states: *red, yellow, green, pink*, and *blue*.

Let the number of states of a categorical variable be M. The states can be denoted by letters, symbols, or a set of integers, such as $1, 2, \ldots, M$. Notice that such integers are used just for data handling and do not represent any specific ordering.

"*How is dissimilarity computed between objects described by categorical variables?*" The dissimilarity between two objects i and j can be computed based on the ratio of mismatches:

$$d(i, j) = \frac{p - m}{p}, \tag{7.12}$$

where m is the number of *matches* (i.e., the number of variables for which i and j are in the same state), and p is the total number of variables. Weights can be assigned to increase the effect of m or to assign greater weight to the matches in variables having a larger number of states.

Categorical variables can be encoded by asymmetric binary variables by creating a new binary variable for each of the M states. For an object with a given state value, the binary variable representing that state is set to 1, while the remaining binary variables are set to 0. For example, to encode the categorical variable map color, a binary variable can be created for each of the five colors listed above. For an object having the color yellow, the yellow variable is set to 1, while the remaining four variables are set to 0.

Ordinal Variables

A **discrete ordinal variable** resembles a categorical variable, except that the M states of the ordinal value are ordered in a meaningful sequence. Ordinal variables are very useful for registering subjective assessments of qualities that cannot be measured objectively. For example, professional ranks are often enumerated in a sequential order, such as *assistant*, *associate*, and *full* for professors. A **continuous ordinal variable** looks like a set of continuous data of an unknown scale; that is, the relative ordering of the values is essential but their actual magnitude is not. For example, the relative ranking in a particular sport (e.g., gold, silver, bronze) is often more essential than the actual values of a particular measure. Ordinal variables may also be obtained from the discretization of interval-scaled quantities by splitting the value range into a finite number of classes. The values of an ordinal variable can be mapped to *ranks*. For example, suppose that an ordinal variable f has M_f states. These ordered states define the ranking $1, \ldots, M_f$.

"*How are ordinal variables handled?*" The treatment of ordinal variables is quite similar to that of interval-scaled variables when computing the dissimilarity between objects. Suppose that f is a variable from a set of ordinal variables describing n objects. The dissimilarity computation with respect to f involves the following steps:

1. The value of f for the ith object is x_{if}, and f has M_f ordered states, representing the ranking $1, \ldots, M_f$. Replace each x_{if} by its corresponding rank, $r_{if} \in \{1, \ldots, M_f\}$.
2. Since each ordinal variable can have a different number of states, it is often necessary to map the range of each variable onto [0.0,1.0] so that each variable has equal weight. This can be achieved by replacing the rank r_{if} of the ith object in the fth variable by

$$z_{if} = \frac{r_{if} - 1}{M_f - 1}. \tag{7.13}$$

3. Dissimilarity can then be computed using any of the distance measures described in Section 7.2.1 for interval-scaled variables, using z_{if} to represent the f value for the ith object.

Ratio-Scaled Variables

A **ratio-scaled variable** makes a positive measurement on a nonlinear scale, such as an exponential scale, approximately following the formula

$$Ae^{Bt} \quad \text{or} \quad Ae^{-Bt} \tag{7.14}$$

where A and B are positive constants, and t typically represents time. Common examples include the growth of a bacteria population or the decay of a radioactive element.

"How can I compute the dissimilarity between objects described by ratio-scaled variables?" There are three methods to handle ratio-scaled variables for computing the dissimilarity between objects.

- Treat ratio-scaled variables like interval-scaled variables. This, however, is not usually a good choice since it is likely that the scale may be distorted.
- Apply **logarithmic transformation** to a ratio-scaled variable f having value x_{if} for object i by using the formula $y_{if} = \log(x_{if})$. The y_{if} values can be treated as interval-valued, as described in Section 7.2.1. Notice that for some ratio-scaled variables, log-log or other transformations may be applied, depending on the variable's definition and the application.
- Treat x_{if} as continuous ordinal data and treat their ranks as interval-valued.

The latter two methods are the most effective, although the choice of method used may depend on the given application.

4) Variables of Mixed Types: Previously we discussed how to compute the dissimilarity between objects described by variables of the same type, where these types may be either interval-scaled, symmetric binary, asymmetric binary, categorical, ordinal, or ratio-scaled. However, in many real databases, objects are described by a mixture of variable types. In general, a database can contain all of the six variable types listed above.

"So, how can we compute the dissimilarity between objects of mixed variable types?"

One approach is to group each kind of variable together, performing a separate cluster analysis for each variable type. This is feasible if these analyses derive compatible results. However, in real applications, it is unlikely that a separate cluster analysis per variable type will generate compatible results.

A more preferable approach is to process all variable types together, performing a single cluster analysis. One such technique combines the different variables into a single dissimilarity matrix, bringing all of the meaningful variables onto a common scale of the interval [0.0,1.0].

Suppose that the data set contains p variables of mixed type. The dissimilarity $d(i, j)$ between objects i and j is defined as

$$d(i, j) = \frac{\sum_{f=1}^{p} \delta_{ij}^{(f)} d_{ij}^{(f)}}{\sum_{f=1}^{p} \delta_{ij}^{(f)}}, \tag{7.15}$$

where the indicator $\delta_{ij}^{(f)} = 0$ if either (1) x_{if} or x_{jf} is missing (i.e., there is no measurement of variable f for object i or object j), or (2) $x_{if} = x_{jf} = 0$ and variable f is asymmetric binary; otherwise, $\delta_{ij}^{(f)} = 1$. The contribution of variable f to the dissimilarity between i and j, that is, $d_{ij}^{(f)}$, is computed dependent on its type:

- If f is interval-based: $d_{ij}^{(f)} = \frac{|x_{if} - x_{jf}|}{max_h x_{hf} - min_h x_{hf}}$, where h runs over all nonmissing objects for variable f.
- If f is binary or categorical: $d_{ij}^{(f)} = 0$ if $x_{if} = x_{jf}$; otherwise $d_{ij}^{(f)} = 1$.
- If f is ordinal: compute the ranks r_{if} and $z_{if} = \frac{r_{if} - 1}{M_f - 1}$, and treat z_{if} as interval-scaled.
- If f is ratio-scaled: either perform logarithmic transformation and treat the transformed data as interval-scaled; or treat f as continuous ordinal data, compute r_{if} and z_{if}, and then treat z_{if} as interval-scaled.

The above steps are identical to what we have already seen for each of the individual variable types. The only difference is for interval-based variables, where here we normalize so that the values map to the interval [0.0,1.0]. Thus, the dissimilarity between objects can be computed even when the variables describing the objects are of different types.

5) Vector Objects: In some applications, such as information retrieval, text document clustering, and biological taxonomy, we need to compare and cluster complex objects (such as documents) containing a large number of symbolic entities (such as keywords and phrases). To measure the distance between complex objects, it is often desirable to abandon traditional metric

distance computation and introduce a nonmetric similarity function. There are several ways to define such a similarity function, s(x, y), to compare two vectors x and y. One popular way is to define the similarity function as a cosine measure as follows:

$$s(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x}^t \cdot \boldsymbol{y}}{||\boldsymbol{x}||||\boldsymbol{y}||}, \tag{7.16}$$

where $\boldsymbol{x}^t$ is a transposition of vector $\boldsymbol{x}$, $||\boldsymbol{x}||$ is the Euclidean norm of vector $\boldsymbol{x}$,[1] $||\boldsymbol{y}||$ is the Euclidean norm of vector $\boldsymbol{y}$, and s is essentially the cosine of the angle between vectors $\boldsymbol{x}$ and $\boldsymbol{y}$. This value is invariant to rotation and dilation, but it is not invariant to translation and general linear transformation.

When variables are binary-valued (0 or 1), the above similarity function can be interpreted in terms of shared features and attributes. Suppose an object $\boldsymbol{x}$ possesses the ith attribute if $x_i = 1$. Then $\boldsymbol{x}^t \cdot \boldsymbol{y}$ is the number of attributes possessed by both $\boldsymbol{x}$ and $\boldsymbol{y}$, and $|\boldsymbol{x}||\boldsymbol{y}|$ is the geometric mean of the number of attributes possessed by $\boldsymbol{x}$ and the number possessed by $\boldsymbol{y}$. Thus $s(\boldsymbol{x}, \boldsymbol{y})$ is a measure of relative possession of common attributes.

A simple variation of the above measure is

$$s(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x}^t \cdot \boldsymbol{y}}{\boldsymbol{x}^t \cdot \boldsymbol{x} + \boldsymbol{y}^t \cdot \boldsymbol{y} - \boldsymbol{x}^t \cdot \boldsymbol{y}} \tag{7.17}$$

which is the ratio of the number of attributes shared by $\boldsymbol{x}$ and $\boldsymbol{y}$ to the number of attributes possessed by $\boldsymbol{x}$ or $\boldsymbol{y}$. This function, known as the **Tanimoto coefficient** or **Tanimoto distance**, is frequently used in information retrieval and biology taxonomy.

Notice that there are many ways to select a particular similarity (or distance) function or normalize the data for cluster analysis. There is no universal standard to guide such selection. The appropriate selection of such measures will heavily depend on the given application. One should bear this in mind and refine the selection of such measures to ensure that the clusters generated are meaningful and useful for the application at hand.

CLUSTERING METHODS [OR] CATEGORIZATION OF MAJOR CLUSTERING METHODS [OR] EVALUATION OF CLUSTERING ALGORITHMS

Many clustering algorithms exist in the literature. It is difficult to provide a crisp categorization of clustering methods because these categories may overlap, so that a method may have features from several categories.

Nevertheless, it is useful to present a relatively organized picture of the different clustering methods.

In general, the major clustering methods can be classified into the following categories:

1) Partitioning Method.
2) Hierarchical Method.
3) Density-based Method.
4) Grid-Based Method.
5) Model-Based Method.
6) Constraint-based Method.

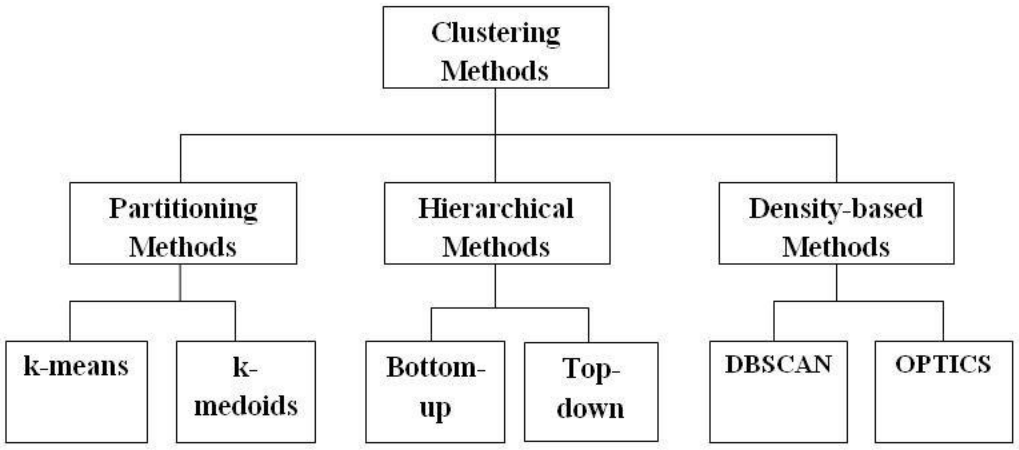


Fig: Clustering methods

1) Partitioning methods: Given a database of n objects or data tuples, a partitioning method constructs k partitions of the data, where each partition represents a cluster and $k \leq n$. That is, it classifies the data into k groups, which together satisfy the following requirements: (1) each group must contain at least one object, and (2) each object must belong to exactly one group. Notice that the second requirement can be relaxed in some fuzzy partitioning techniques.

Given k, the number of partitions to construct, a partitioning method creates an initial partitioning. It then uses an iterative relocation technique that attempts to improve the partitioning by moving objects from one group to another. The general criterion of a good partitioning is that objects in the same cluster are “close” or related to each other, whereas objects of different

clusters are "far apart" or very different. There are various kinds of other criteria for judging the quality of partitions.

To achieve global optimality in partitioning-based clustering would require the exhaustive enumeration of all of the possible partitions. Instead, most applications adopt one of a few popular heuristic methods, such as (1) the k-means algorithm, where each cluster is represented by the mean value of the objects in the cluster, and (2) the k-medoids algorithm, where each cluster is represented by one of the objects located near the center of the cluster. These heuristic clustering methods work well for finding spherical-shaped clusters in small to medium-sized databases. To find clusters with complex shapes and for clustering very large data sets, partitioning-based methods need to be extended.

Classical Partitioning Methods: k-Means and k-Medoids: The most well-known and commonly used partitioning methods are k-means, k-medoids, and their variations.

Centroid-Based Technique: The k-Means Method:

The *k*-**means algorithm** takes the input parameter, *k*, and partitions a set of *n* objects into *k* clusters so that the resulting intracluster similarity is high but the intercluster similarity is low. Cluster similarity is measured in regard to the *mean* value of the objects in a cluster, which can be viewed as the cluster's *centroid* or *center of gravity*.

"*How does the k-means algorithm work?*" The *k*-means algorithm proceeds as follows. First, it randomly selects *k* of the objects, each of which initially represents a cluster mean or center. For each of the remaining objects, an object is assigned to the cluster to which it is the most similar, based on the distance between the object and the cluster mean. It then computes the new mean for each cluster. This process iterates until the criterion function converges. Typically, the **square-error criterion** is used, defined as

$$E = \sum_{i=1}^{k} \sum_{p \in C_i} |p - m_i|^2, \tag{7.18}$$

where E is the sum of the square error for all objects in the data set; p is the point in space representing a given object; and m_i is the mean of cluster C_i (both p and m_i are multidimensional). In other words, for each object in each cluster, the distance from the object to its cluster center is squared, and the distances are summed. This criterion tries to make the resulting k clusters as compact and as separate as possible. The *k*-means procedure is summarized in Figure 7.2.

Algorithm: *k*-means. The *k*-means algorithm for partitioning, where each cluster's center is represented by the mean value of the objects in the cluster.

Input:

- *k*: the number of clusters,
- *D*: a data set containing *n* objects.

Output: A set of *k* clusters.

Method:

(1) arbitrarily choose *k* objects from *D* as the initial cluster centers;
(2) **repeat**
(3) (re)assign each object to the cluster to which the object is the most similar, based on the mean value of the objects in the cluster;
(4) update the cluster means, i.e., calculate the mean value of the objects for each cluster;
(5) **until** no change;

Figure 7.2 The *k*-means partitioning algorithm.

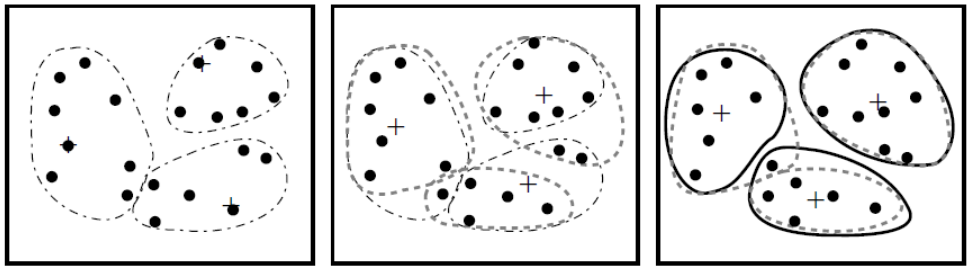


Figure 7.3 Clustering of a set of objects based on the *k*-means method. (The mean of each cluster is marked by a "+".)

Representative Object-Based Technique: The k-Medoids Method:

The *k*-means algorithm is sensitive to outliers because an object with an extremely large value may substantially distort the distribution of data. This effect is particularly exacerbated due to the use of the *square*-error function (Equation (7.18)).

"*How might the algorithm be modified to diminish such sensitivity?*" Instead of taking the mean value of the objects in a cluster as a reference point, we can pick actual objects to represent the clusters, using one representative object per cluster. Each remaining object is clustered with the representative object to which it is the most similar. The partitioning method is then performed based on the principle of minimizing the sum of

the dissimilarities between each object and its corresponding reference point. That is, an **absolute-error criterion** is used, defined as

$$E = \sum_{j=1}^{k} \sum_{p \in C_j} |p - o_j|, \tag{7.19}$$

where E is the sum of the absolute error for all objects in the data set; p is the point in space representing a given object in cluster C_j; and o_j is the representative object of C_j. In general, the algorithm iterates until, eventually, each representative object is actually the **medoid**, or most centrally located object, of its cluster. This is the basis of the *k*-**medoids method** for grouping n objects into k clusters.

Let's look closer at *k*-medoids clustering. The initial representative objects (or seeds) are chosen arbitrarily. The iterative process of replacing representative objects by nonrepresentative objects continues as long as the quality of the resulting clustering is improved. This quality is estimated using a cost function that measures the average dissimilarity between an object and the representative object of its cluster. To determine whether a nonrepresentative object, o_{random}, is a good replacement for a current representative object, o_j, the following four cases are examined for each of the nonrepresentative objects, p, as illustrated in Figure 7.4.

- Case 1: p currently belongs to representative object, o_j. If o_j is replaced by o_{random} as a representative object and p is closest to one of the other representative objects, o_i, $i \neq j$, then p is reassigned to o_i.
- Case 2: p currently belongs to representative object, o_j. If o_j is replaced by o_{random} as a representative object and p is closest to o_{random}, then p is reassigned to o_{random}.
- Case 3: p currently belongs to representative object, o_i, $i \neq j$. If o_j is replaced by o_{random} as a representative object and p is still closest to o_i, then the assignment does not change.
- Case 4: p currently belongs to representative object, o_i, $i \neq j$. If o_j is replaced by o_{random} as a representative object and p is closest to o_{random}, then p is reassigned to o_{random}.

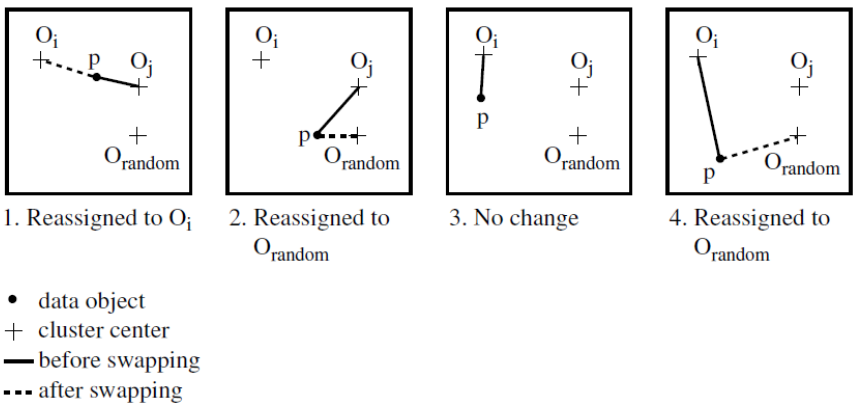


Figure 7.4 Four cases of the cost function for *k*-medoids clustering.

Algorithm: *k*-medoids. PAM, a *k*-medoids algorithm for partitioning based on medoid or central objects.

Input:

- *k*: the number of clusters,
- *D*: a data set containing *n* objects.

Output: A set of *k* clusters.

Method:

(1) arbitrarily choose *k* objects in *D* as the initial representative objects or seeds;
(2) **repeat**
(3) assign each remaining object to the cluster with the nearest representative object;
(4) randomly select a nonrepresentative object, o_{random};
(5) compute the total cost, *S*, of swapping representative object, o_j, with o_{random};
(6) **if** $S < 0$ **then** swap o_j with o_{random} to form the new set of *k* representative objects;
(7) **until** no change;

Figure 7.5 PAM, a *k*-medoids partitioning algorithm.

Partitioning Methods in Large Databases: From k-Medoids to CLARANS:

"*How efficient is the k-medoids algorithm on large data sets?*" A typical *k*-medoids partitioning algorithm like PAM works effectively for small data sets, but does not scale well for large data sets. To deal with larger data sets, a *sampling*-based method, called **CLARA** (Clustering LARge Applications), can be used.

The idea behind CLARA is as follows: Instead of taking the whole set of data into consideration, a small portion of the actual data is chosen as a representative of the data. Medoids are then chosen from this sample using PAM. If the sample is selected in a fairly random manner, it should closely represent the original data set. The representative objects (medoids) chosen will likely be similar to those that would have been chosen from the whole data set. CLARA draws multiple samples of the data set, applies PAM on each sample, and returns its best clustering as the output. As expected, CLARA can deal with larger data sets than PAM. The complexity of each iteration now becomes $O(ks^2 + k(n-k))$, where *s* is the size of the sample, *k* is the number of clusters, and *n* is the total number of objects.

The effectiveness of CLARA depends on the sample size. Notice that PAM searches for the best *k* medoids among a given data set, whereas CLARA searches for the best *k* medoids among the *selected sample* of the data set. CLARA cannot find the best clustering if any of the best sampled medoids is not among the best *k* medoids. That is, if an object o_i is one of the best *k* medoids but is not selected during sampling, CLARA will never find the best clustering. This is, therefore, a trade-off for efficiency. A good clustering based on sampling will not necessarily represent a good clustering of the whole data set if the sample is biased.

2) Hierarchical methods: A hierarchical method creates a hierarchical decomposition of the given set of data objects. A hierarchical method can be classified as being either agglomerative or divisive, based on how the hierarchical decomposition is formed. The agglomerative approach, also called the bottom-up approach, starts with each object forming a separate group. It successively merges the objects or groups that are close to one another, until all of the groups are merged into one (the topmost level of the hierarchy), or until a termination condition holds. The divisive approach, also called the top-down approach, starts with all of the objects in the same cluster. In each successive iteration, a cluster is split up into smaller clusters, until eventually each object is in one cluster, or until a termination condition holds.

Hierarchical methods suffer from the fact that once a step (merge or split) is done, it can never be undone. This rigidity is useful in that it leads to smaller computation costs by not having to worry about a combinatorial number of different choices. However, such techniques cannot correct erroneous decisions. There are two approaches to improving the quality of hierarchical clustering: (1) perform careful analysis of object "linkages" at each hierarchical partitioning, such as in Chameleon, or (2) integrate hierarchical agglomeration and other approaches by first using a hierarchical agglomerative algorithm to group objects into microclusters, and then performing macroclustering on the microclusters using another clustering method such as iterative relocation, as in BIRCH.

Agglomerative and Divisive Hierarchical Clustering: In general, there are two types of hierarchical clustering methods:

1) **Agglomerative hierarchical clustering:** This bottom-up strategy starts by placing each object in its own cluster and then merges these atomic clusters into larger and larger clusters, until all of the objects are in a single cluster or until certain termination conditions are satisfied. Most hierarchical clustering methods belong to this category. They differ only in their definition of intercluster similarity.
2) **Divisive hierarchical clustering:** This top-down strategy does the reverse of agglomerative hierarchical clustering by starting with all

objects in one cluster. It subdivides the cluster into smaller and smaller pieces, until each object forms a cluster on its own or until it satisfies certain termination conditions, such as a desired number of clusters is obtained or the diameter of each cluster is within a certain threshold.

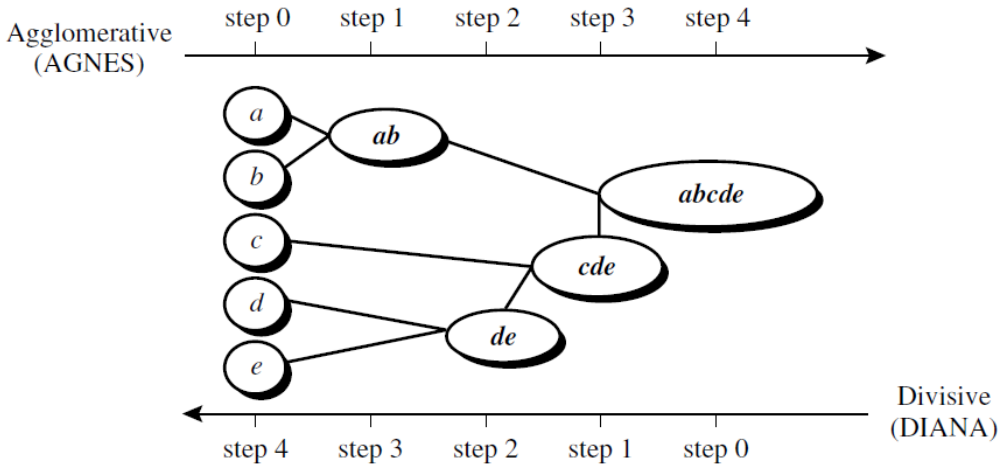


Figure 7.6 Agglomerative and divisive hierarchical clustering on data objects $\{a, b, c, d, e\}$.

In either agglomerative or divisive hierarchical clustering, the user can specify the desired number of clusters as a termination condition.

A tree structure called a **dendrogram** is commonly used to represent the process of hierarchical clustering. It shows how objects are grouped together step by step. Figure 7.7 shows a dendrogram for the five objects presented in Figure 7.6, where $l = 0$ shows the five objects as singleton clusters at level 0. At $l = 1$, objects $\boldsymbol{a}$ and $\boldsymbol{b}$ are grouped together to form the first cluster, and they stay together at all subsequent levels. We can also use a vertical axis to show the similarity scale between clusters. For example, when the similarity of two groups of objects, $\{\boldsymbol{a}, \boldsymbol{b}\}$ and $\{\boldsymbol{c}, \boldsymbol{d}, \boldsymbol{e}\}$, is roughly 0.16, they are merged together to form a single cluster.

Four widely used measures for distance between clusters are as follows, where $|\boldsymbol{p} - \boldsymbol{p}'|$ is the distance between two objects or points, $\boldsymbol{p}$ and $\boldsymbol{p}'$; $\boldsymbol{m_i}$ is the mean for cluster, C_i; and n_i is the number of objects in C_i.

$$\text{Minimum distance}: \quad d_{min}(C_i, C_j) = min_{\boldsymbol{p} \in C_i, \boldsymbol{p}' \in C_j} |\boldsymbol{p} - \boldsymbol{p}'| \tag{7.20}$$

$$\text{Maximum distance}: \quad d_{max}(C_i, C_j) = max_{\boldsymbol{p} \in C_i, \boldsymbol{p}' \in C_j} |\boldsymbol{p} - \boldsymbol{p}'| \tag{7.21}$$

$$\text{Mean distance}: \quad d_{mean}(C_i, C_j) = |\boldsymbol{m_i} - \boldsymbol{m_j}| \tag{7.22}$$

$$\text{Average distance}: \quad d_{avg}(C_i, C_j) = \frac{1}{n_i n_j} \sum_{\boldsymbol{p} \in C_i} \sum_{\boldsymbol{p}' \in C_j} |\boldsymbol{p} - \boldsymbol{p}'| \tag{7.23}$$

When an algorithm uses the *minimum distance*, $d_{min}(C_i, C_j)$, to measure the distance between clusters, it is sometimes called a **nearest-neighbor clustering algorithm**. Moreover, if the clustering process is terminated when the distance between nearest clusters exceeds an arbitrary threshold, it is called a **single-linkage algorithm**. If we view the data points as nodes of a graph, with edges forming a path between the nodes in a cluster, then the merging of two clusters, C_i and C_j, corresponds to adding an edge between the nearest pair of nodes in C_i and C_j. Because edges linking clusters always go between distinct clusters, the resulting graph will generate a tree. Thus, an agglomerative hierarchical clustering algorithm that uses the minimum distance measure is also called a **minimal spanning tree algorithm**.

When an algorithm uses the *maximum distance*, $d_{max}(C_i, C_j)$, to measure the distance between clusters, it is sometimes called a **farthest-neighbor clustering algorithm**. If the clustering process is terminated when the maximum distance between nearest clusters exceeds an arbitrary threshold, it is called a **complete-linkage algorithm**. By viewing data points as nodes of a graph, with edges linking nodes, we can think of each cluster as a *complete* subgraph, that is, with edges connecting all of the nodes in the clusters. The distance between two clusters is determined by the most distant nodes in the two clusters. Farthest-neighbor algorithms tend to minimize the increase in diameter of the clusters at each iteration as little as possible. If the true clusters are rather compact and approximately equal in size, the method will produce high-quality clusters. Otherwise, the clusters produced can be meaningless.

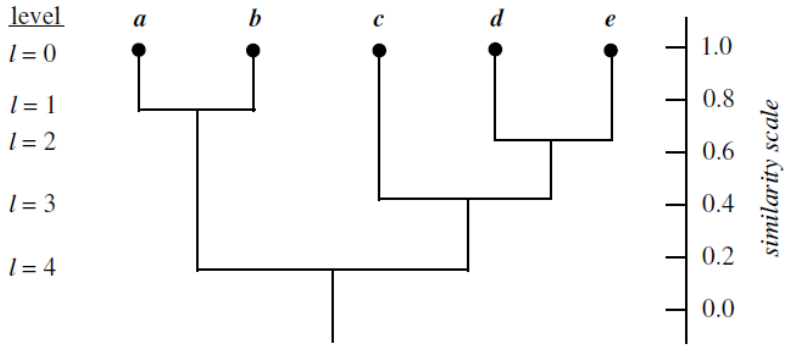


Figure 7.7 Dendrogram representation for hierarchical clustering of data objects $\{a, b, c, d, e\}$.

BIRCH: Balanced Iterative Reducing and Clustering Using Hierarchies:

BIRCH is designed for clustering a large amount of numerical data by integration of hierarchical clustering (at the initial *microclustering* stage) and other clustering methods such as iterative partitioning (at the later *macroclustering* stage). It overcomes the two difficulties of agglomerative clustering methods: (1) scalability and (2) the inability to undo what was done in the previous step.

BIRCH introduces two concepts, *clustering feature* and *clustering feature tree* (*CF tree*), which are used to summarize cluster representations. These structures help the clustering method achieve good speed and scalability in large databases and also make it effective for incremental and dynamic clustering of incoming objects.

Let's look closer at the above-mentioned structures. Given n d-dimensional data objects or points in a cluster, we can define the centroid $\boldsymbol{x_0}$, radius R, and diameter D of the cluster as follows:

$$\boldsymbol{x_0} = \frac{\sum_{i=1}^{n} \boldsymbol{x_i}}{n} \tag{7.24}$$

$$R = \sqrt{\frac{\sum_{i=1}^{n} (\boldsymbol{x_i} - \boldsymbol{x_0})^2}{n}} \tag{7.25}$$

$$D = \sqrt{\frac{\sum_{i=1}^{n} \sum_{j=1}^{n} (\boldsymbol{x_i} - \boldsymbol{x_j})^2}{n(n-1)}} \tag{7.26}$$

where R is the average distance from member objects to the centroid, and D is the average pairwise distance within a cluster. Both R and D reflect the tightness of the cluster around the centroid. A **clustering feature** (CF) is a three-dimensional vector summarizing information about clusters of objects. Given n d-dimensional objects or points in a cluster, $\{\boldsymbol{x_i}\}$, then the CF of the cluster is defined as

$$CF = \langle n, \boldsymbol{LS}, \boldsymbol{SS} \rangle, \tag{7.27}$$

where n is the number of points in the cluster, $\boldsymbol{LS}$ is the linear sum of the n points (i.e., $\sum_{i=1}^{n} \boldsymbol{x_i}$), and $\boldsymbol{SS}$ is the square sum of the data points (i.e., $\sum_{i=1}^{n} \boldsymbol{x_i}^2$).

A clustering feature is essentially a summary of the statistics for the given cluster: the zeroth, first, and second moments of the cluster from a statistical point of view. Clustering features are *additive*. For example, suppose that we have two disjoint clusters, C_1 and C_2, having the clustering features, CF_1 and CF_2, respectively. The clustering feature for the cluster that is formed by merging C_1 and C_2 is simply $CF_1 + CF_2$.

Clustering features are sufficient for calculating all of the measurements that are needed for making clustering decisions in BIRCH. BIRCH thus utilizes storage efficiently by employing the clustering features to summarize information about the clusters of objects, thereby bypassing the need to store all objects.

A **CF tree** is a height-balanced tree that stores the clustering features for a hierarchical clustering. An example is shown in Figure 7.8. By definition, a nonleaf node in a tree has descendants or "children." The nonleaf nodes store sums of the CFs of their children, and thus summarize clustering information about their children. A CF tree has two parameters: *branching factor*, B, and *threshold*, T. The branching factor specifies the maximum number of children per nonleaf node. The threshold parameter specifies the maximum diameter of subclusters stored at the leaf nodes of the tree. These two parameters influence the size of the resulting tree.

BIRCH tries to produce the best clusters with the available resources. Given a limited amount of main memory, an important consideration is to minimize the time required for I/O. BIRCH applies a *multiphase* clustering technique: a single scan of the data set yields a basic good clustering, and one or more additional scans can (optionally) be used to further improve the quality. The primary phases are:

- **Phase 1:** BIRCH scans the database to build an initial in-memory CF tree, which can be viewed as a multilevel compression of the data that tries to preserve the inherent clustering structure of the data.

- **Phase 2:** BIRCH applies a (selected) clustering algorithm to cluster the leaf nodes of the CF tree, which removes sparse clusters as outliers and groups dense clusters into larger ones.

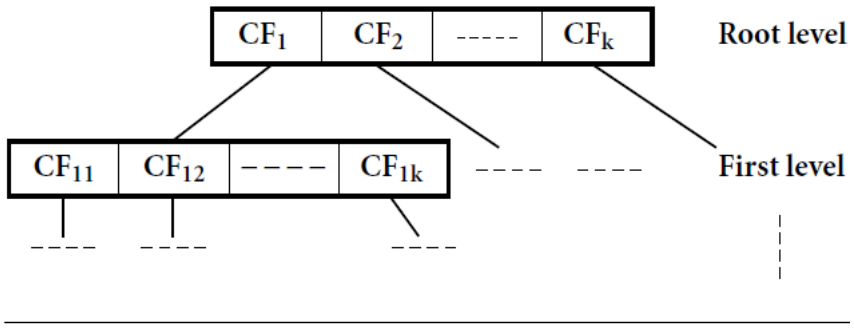


Figure 7.8 A CF tree structure.

ROCK: A Hierarchical Clustering Algorithm for Categorical Attributes:

ROCK (RObust Clustering using linKs) is a hierarchical clustering algorithm that explores the concept of links (the number of common neighbors between two objects) for data with categorical attributes. Traditional clustering algorithms for clustering data with Boolean and categorical attributes use distance functions. However, experiments show that such distance measures cannot lead to high-quality clusters when clustering categorical data. Furthermore, most clustering algorithms assess only the similarity between points when clustering; that is, at each step, points that are the most similar are merged into a single cluster. This "localized" approach is prone to errors. For example, two distinct clusters may have a few points or outliers that are close; therefore, relying on the similarity between points to make clustering decisions could cause the two clusters to be merged. ROCK takes a more global approach to clustering by considering the neighborhoods of individual pairs of points. If two similar points also have similar neighborhoods, then the two points likely belong to the same cluster and so can be merged.

$$sim(T_i, T_j) = \frac{|T_i \cap T_j|}{|T_i \cup T_j|}. \qquad (7.28)$$

Based on these ideas, ROCK first constructs a sparse graph from a given data similarity matrix using a similarity threshold and the concept of shared neighbors. It then performs agglomerative hierarchical clustering on the sparse graph. A goodness measure is used to evaluate the clustering. Random sampling is used for scaling up to large data sets. The worst-case time complexity of ROCK is $O(n^2 + nm_m m_a + n^2 \log n)$, where m_m and m_a are the maximum and average number of neighbors, respectively, and n is the number of objects.

In several real-life data sets, such as the congressional voting data set and the mushroom data set at UC-Irvine Machine Learning Repository, ROCK has demonstrated its power at deriving much more meaningful clusters than the traditional hierarchical clustering algorithms.

Chameleon: A Hierarchical Clustering Algorithm Using Dynamic Modeling:

Chameleon is a hierarchical clustering algorithm that uses dynamic modeling to determine the similarity between pairs of clusters. It was derived based on the observed weaknesses of two hierarchical clustering algorithms: ROCK and CURE. ROCK and related schemes emphasize cluster interconnectivity while ignoring information regarding cluster proximity. CURE and related schemes consider cluster proximity yet ignore cluster interconnectivity. In Chameleon, cluster similarity is assessed based on how well-connected objects are within a cluster *and* on the proximity of clusters. That is, two clusters are merged if their *interconnectivity* is high and they are *close together*. Thus, Chameleon does not depend on a static, user-supplied model and can automatically adapt to the internal characteristics of the clusters being merged. The merge process facilitates the discovery of natural and homogeneous clusters and applies to all types of data as long as a similarity function can be specified.

"*How does Chameleon work?*" The main approach of Chameleon is illustrated in Figure 7.9. Chameleon uses a k-nearest-neighbor graph approach to construct a sparse graph, where each vertex of the graph represents a data object, and there exists an edge between two vertices (objects) if one object is among the k-most-similar objects of the other. The edges are weighted to reflect the similarity between objects. Chameleon uses a graph partitioning algorithm to partition the k-nearest-neighbor graph into a large number of relatively small subclusters. It then uses an agglomerative hierarchical clustering algorithm that repeatedly merges subclusters based on their similarity. To determine the pairs of most similar subclusters, it takes into account both the interconnectivity as well as the closeness of the clusters. We will give a mathematical definition for these criteria shortly.

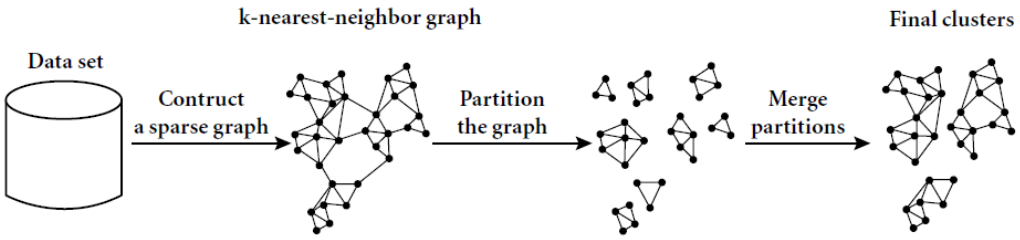


Figure 7.9 Chameleon: Hierarchical clustering based on k-nearest neighbors and dynamic modeling. Based on [KHK99].

The graph-partitioning algorithm partitions the k-nearest-neighbor graph such that it minimizes the **edge cut**. That is, a cluster C is partitioned into subclusters C_i and C_j so as to minimize the *weight of the edges* that would be cut should C be bisected into C_i and C_j. Edge cut is denoted $EC(C_i, C_j)$ and assesses the *absolute* interconnectivity between clusters C_i and C_j.

Chameleon determines the similarity between each pair of clusters C_i and C_j according to their *relative interconnectivity*, $RI(C_i, C_j)$, and their *relative closeness*, $RC(C_i, C_j)$:

- The **relative interconnectivity**, $RI(C_i, C_j)$, between two clusters, C_i and C_j, is defined as the absolute interconnectivity between C_i and C_j, normalized with respect to the internal interconnectivity of the two clusters, C_i and C_j. That is,

$$RI(C_i, C_j) = \frac{|EC_{\{C_i, C_j\}}|}{\frac{1}{2}(|EC_{C_i}| + |EC_{C_j}|)}, \tag{7.29}$$

  where $EC_{\{C_i, C_j\}}$ is the edge cut, defined as above, for a cluster containing both C_i and C_j. Similarly, EC_{C_i} (or EC_{C_j}) is the minimum sum of the cut edges that partition C_i (or C_j) into two roughly equal parts.

- The **relative closeness**, $RC(C_i, C_j)$, between a pair of clusters, C_i and C_j, is the absolute closeness between C_i and C_j, normalized with respect to the internal closeness of the two clusters, C_i and C_j. It is defined as

$$RC(C_i, C_j) = \frac{\overline{S}_{EC_{\{C_i, C_j\}}}}{\frac{|C_i|}{|C_i|+|C_j|}\overline{S}_{EC_{C_i}} + \frac{|C_j|}{|C_i|+|C_j|}\overline{S}_{EC_{C_j}}}, \tag{7.30}$$

  where $\overline{S}_{EC_{\{C_i, C_j\}}}$ is the average weight of the edges that connect vertices in C_i to vertices in C_j, and $\overline{S}_{EC_{C_i}}$ (or $\overline{S}_{EC_{C_j}}$) is the average weight of the edges that belong to the min-cut bisector of cluster C_i (or C_j).

Chameleon has been shown to have greater power at discovering arbitrarily shaped clusters of high quality than several well-known algorithms such as BIRCH and density-based DBSCAN. However, the processing cost for high-dimensional data may require $O(n^2)$ time for n objects in the worst case.

3) Density-based methods: Most partitioning methods cluster objects based on the distance between objects. Such methods can find only spherical-shaped clusters and encounter difficulty at discovering clusters of arbitrary shapes. Other clustering methods have been developed based on the notion of density. Their general idea is to continue growing the given cluster as long

as the density (number of objects or data points) in the "neighborhood" exceeds some threshold; that is, for each data point within a given cluster, the neighborhood of a given radius has to contain at least a minimum number of points. Such a method can be used to filter out noise (outliers) and discover clusters of arbitrary shape.

DBSCAN and its extension, OPTICS, are typical density-based methods that grow clusters according to a density-based connectivity analysis. DENCLUE is a method that clusters objects based on the analysis of the value distributions of density functions.

DBSCAN: A Density-Based Clustering Method Based on Connected Regions with Sufficiently High Density:

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a density-based clustering algorithm. The algorithm grows regions with sufficiently high density into clusters and discovers clusters of arbitrary shape in spatial databases with noise. It defines a cluster as a maximal set of *density-connected* points.

The basic ideas of density-based clustering involve a number of new definitions. We intuitively present these definitions, and then follow up with an example.

- The neighborhood within a radius ε of a given object is called the **ε-neighborhood** of the object.
- If the ε-neighborhood of an object contains at least a minimum number, *MinPts*, of objects, then the object is called a **core object**.
- Given a set of objects, D, we say that an object $\boldsymbol{p}$ is **directly density-reachable** from object $\boldsymbol{q}$ if $\boldsymbol{p}$ is within the ε-neighborhood of $\boldsymbol{q}$, and $\boldsymbol{q}$ is a core object.
- An object $\boldsymbol{p}$ is **density-reachable** from object $\boldsymbol{q}$ with respect to ε and *MinPts* in a set of objects, D, if there is a chain of objects $\boldsymbol{p_1}, \ldots, \boldsymbol{p_n}$, where $\boldsymbol{p_1} = \boldsymbol{q}$ and $\boldsymbol{p_n} = \boldsymbol{p}$ such that $\boldsymbol{p_{i+1}}$ is directly density-reachable from $\boldsymbol{p_i}$ with respect to ε and *MinPts*, for $1 \leq i \leq n$, $\boldsymbol{p_i} \in D$.
- An object $\boldsymbol{p}$ is **density-connected** to object $\boldsymbol{q}$ with respect to ε and *MinPts* in a set of objects, D, if there is an object $\mathbf{o} \in D$ such that both $\boldsymbol{p}$ and $\boldsymbol{q}$ are density-reachable from $\boldsymbol{o}$ with respect to ε and *MinPts*.

Density reachability is the transitive closure of direct density reachability, and this relationship is asymmetric. Only core objects are mutually density reachable. Density connectivity, however, is a symmetric relation.

A **density-based cluster** is a set of density-connected objects that is maximal with respect to density-reachability. Every object not contained in any cluster is considered to be *noise*.

"*How does DBSCAN find clusters?*" DBSCAN searches for clusters by checking the ε-neighborhood of each point in the database. If the ε-neighborhood of a point ***p*** contains more than *MinPts*, a new cluster with ***p*** as a core object is created. DBSCAN then iteratively collects directly density-reachable objects from these core objects, which may involve the merge of a few density-reachable clusters. The process terminates when no new point can be added to any cluster.

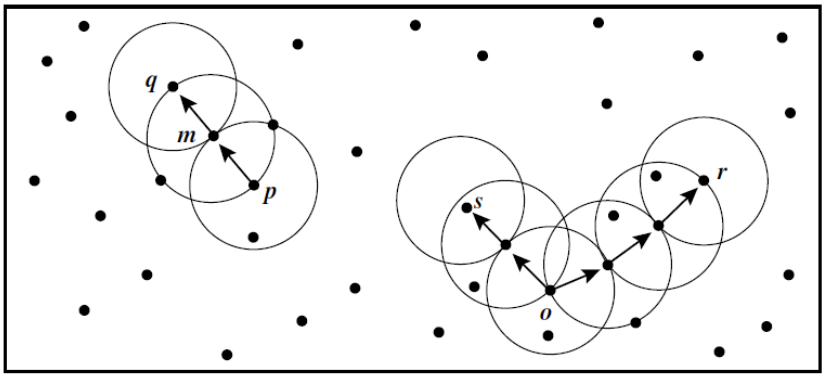


Figure 7.10 Density reachability and density connectivity in density-based clustering. Based on [EKSX96].

If a spatial index is used, the computational complexity of DBSCAN is $O(n \log n)$, where n is the number of database objects. Otherwise, it is $O(n^2)$. With appropriate settings of the user-defined parameters ε and *MinPts*, the algorithm is effective at finding arbitrary-shaped clusters.

OPTICS: Ordering Points to Identify the Clustering Structure:

Although DBSCAN can cluster objects given input parameters such as ε and *MinPts*, it still leaves the user with the responsibility of selecting parameter values that will lead to the discovery of acceptable clusters. Actually, this is a problem associated with many other clustering algorithms. Such parameter settings are usually empirically set and difficult to determine, especially for real-world, high-dimensional data sets. Most algorithms are very sensitive to such parameter values: slightly different settings may lead to very different clusterings of the data. Moreover, high-dimensional real data sets often have very skewed distributions, such that their intrinsic clustering structure may not be characterized by *global* density parameters.

To help overcome this difficulty, a cluster analysis method called **OPTICS** was proposed. Rather than produce a data set clustering explicitly, OPTICS computes an augmented *cluster ordering* for automatic and interactive cluster analysis. This ordering represents the density-based clustering structure of the data. It contains information that is equivalent to density-based clustering obtained from a wide range of parameter settings. The cluster ordering can be used to extract basic clustering information (such as cluster centers or arbitrary-shaped clusters) as well as provide the intrinsic clustering structure.

By examining DBSCAN, we can easily see that for a constant *MinPts* value, density-based clusters with respect to a higher density (i.e., a lower value for ε) are *completely contained* in density-connected sets obtained with respect to a lower density. Recall that the parameter ε is a distance—it is the neighborhood radius. Therefore, in order to produce a set or ordering of density-based clusters, we can extend the DBSCAN algorithm to process a set of distance parameter values at the same time. To construct the different clusterings simultaneously, the objects should be processed in a specific order. This order selects an object that is density-reachable with respect to the lowest ε value so that clusters with higher density (lower ε) will be finished first. Based on this idea, two values need to be stored for each object—*core-distance* and *reachability-distance*:

- The **core-distance** of an object $\boldsymbol{p}$ is the smallest ε' value that makes $\{p\}$ a core object. If $\boldsymbol{p}$ is not a core object, the core-distance of $\boldsymbol{p}$ is undefined.
- The **reachability-distance** of an object $\boldsymbol{q}$ with respect to another object $\boldsymbol{p}$ is the greater value of the core-distance of $\boldsymbol{p}$ and the Euclidean distance between $\boldsymbol{p}$ and $\boldsymbol{q}$. If $\boldsymbol{p}$ is not a core object, the reachability-distance between $\boldsymbol{p}$ and $\boldsymbol{q}$ is undefined.

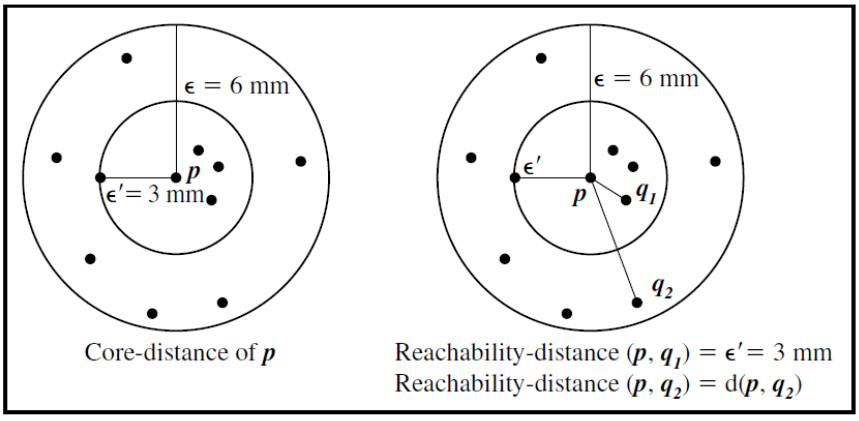


Figure 7.11 OPTICS terminology. Based on [ABKS99].

DENCLUE: Clustering Based on Density Distribution Functions:

DENCLUE (DENsity-based CLUstEring) is a clustering method based on a set of density distribution functions. The method is built on the following ideas: (1) the influence of each data point can be formally modeled using a mathematical function, called an *influence function*, which describes the impact of a data point within its neighborhood; (2) the overall density of the data space can be modeled analytically as the sum of the influence function applied to all data points; and (3) clusters can then be determined mathematically by identifying *density attractors*, where density attractors are local maxima of the overall density function.

Let $\boldsymbol{x}$ and $\boldsymbol{y}$ be objects or points in F^d, a d-dimensional input space. The **influence function** of data object $\boldsymbol{y}$ on $\boldsymbol{x}$ is a function, $f_B^{\boldsymbol{y}} : F^d \rightarrow R_0^+$, which is defined in terms of a basic influence function f_B:

$$f_B^{\boldsymbol{y}}(x) = f_B(\boldsymbol{x}, \boldsymbol{y}). \tag{7.31}$$

This reflects the impact of y on x. In principle, the influence function can be an arbitrary function that can be determined by the distance between two objects in a neighborhood. The distance function, d(x, y), should be reflexive and symmetric, such as the Euclidean distance function. It can be used to compute a square wave influence function,

$$f_{Square}(\boldsymbol{x}, \boldsymbol{y}) = \begin{cases} 0 & if\ d(\boldsymbol{x}, \boldsymbol{y}) > \sigma \\ 1 & otherwise, \end{cases} \tag{7.32}$$

or a *Gaussian influence function*,

$$f_{Gauss}(\boldsymbol{x}, \boldsymbol{y}) = e^{-\frac{d(\boldsymbol{x}, \boldsymbol{y})^2}{2\sigma^2}}. \tag{7.33}$$

To help understand the concept of influence function, the following example offers some additional insight.

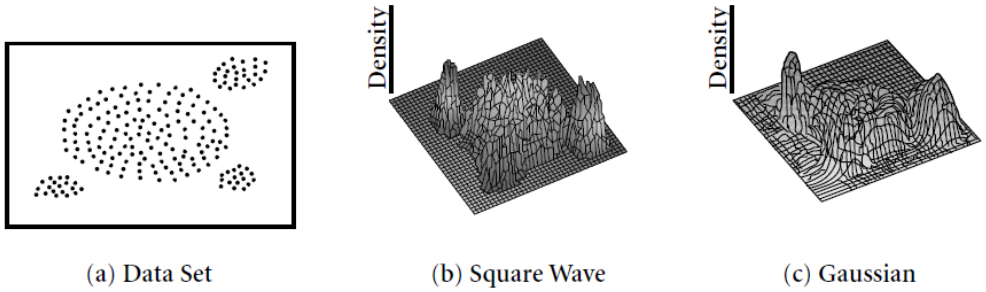
The **density function** at an object or point $\boldsymbol{x} \in F^d$ is defined as the sum of influence functions of all data points. That is, it is the total influence on $\boldsymbol{x}$ of all of the data points. Given n data objects, $D = \{\boldsymbol{x_1}, \ldots, \boldsymbol{x_n}\} \subset F^d$, the density function at $\boldsymbol{x}$ is defined as

$$f_B^D(\boldsymbol{x}) = \sum_{i=1}^{n} f_B^{\boldsymbol{x}_i}(\boldsymbol{x}) = f_B^{\boldsymbol{x}_1}(\boldsymbol{x}) + f_B^{\boldsymbol{x}_2}(\boldsymbol{x}) + \cdots + f_B^{\boldsymbol{x}_n}(\boldsymbol{x}). \tag{7.34}$$

For example, the density function that results from the Gaussian influence function (7.33) is

$$f_{Gauss}^D(\boldsymbol{x}) = \sum_{i=1}^{n} e^{-\frac{d(\boldsymbol{x}, \boldsymbol{x}_i)^2}{2\sigma^2}}. \tag{7.35}$$

Figure 7.13 shows a 2-D data set together with the corresponding overall density functions for a square wave and a Gaussian influence function.



(a) Data Set (b) Square Wave (c) Gaussian

Figure 7.13 Possible density functions for a 2-D data set. From [HK98].

4) Grid-based methods: Grid-based methods quantize the object space into a finite number of cells that form a grid structure. All of the clustering operations are performed on the grid structure (i.e., on the quantized space). The main advantage of this approach is its fast processing time, which is typically independent of the number of data objects and dependent only on the number of cells in each dimension in the quantized space.

STING is a typical example of a grid-based method. WaveCluster applies wavelet transformation for clustering analysis and is both grid-based and density-based.

STING: STatistical INformation Grid:

STING is a grid-based multiresolution clustering technique in which the spatial area is divided into rectangular cells. There are usually several levels of such rectangular cells corresponding to different levels of resolution, and these cells form a hierarchical structure: each cell at a high level is partitioned to form a number of cells at the next lower level. Statistical information regarding the attributes in each grid cell (such as the mean, maximum, and minimum values) is precomputed and stored. These statistical parameters are useful for query processing, as described below.

Figure 7.15 shows a hierarchical structure for STING clustering. Statistical parameters of higher-level cells can easily be computed from the parameters of the lower-level cells. These parameters include the following: the attribute-independent parameter, *count*; the attribute-dependent parameters, *mean*, *stdev* (standard deviation), *min* (minimum), *max* (maximum); and the type of *distribution* that the attribute value in the cell follows, such as *normal*, *uniform*, *exponential*, or *none* (if the distribution is unknown). When the data are loaded into the database, the parameters *count*, *mean*, *stdev*, *min*, and *max* of the bottom-level cells are calculated directly from the data. The value of *distribution* may either be assigned by the user if the distribution type is known beforehand or obtained by hypothesis tests such as the χ^2 test. The type of distribution of a higher-level cell can be computed based on the majority of distribution types of its corresponding lower-level cells in conjunction with a threshold filtering process. If the distributions of the lower-level cells disagree with each other and fail the threshold test, the distribution type of the high-level cell is set to *none*.

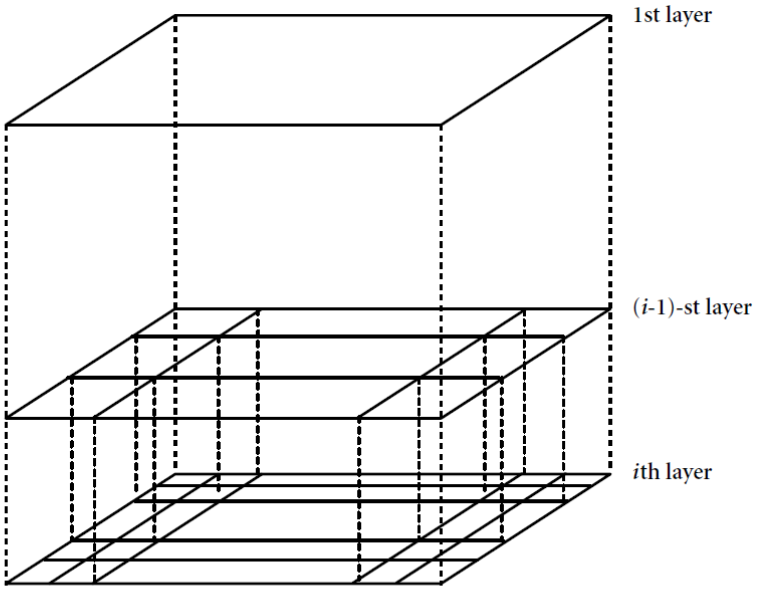


Figure 7.15 A hierarchical structure for STING clustering.

WaveCluster: Clustering Using Wavelet Transformation:

WaveCluster is a multiresolution clustering algorithm that first summarizes the data by imposing a multidimensional grid structure onto the data space. It then uses a *wavelet transformation* to transform the original feature space, finding dense regions in the transformed space.

In this approach, each grid cell summarizes the information of a group of points that map into the cell. This summary information typically fits into main memory for use by the multiresolution wavelet transform and the subsequent cluster analysis.

A **wavelet transform** is a signal processing technique that decomposes a signal into different frequency subbands. The wavelet model can be applied to d-dimensional signals by applying a one-dimensional wavelet transform d times. In applying a wavelet transform, data are transformed so as to preserve the relative distance between objects at different levels of resolution. This allows the natural clusters in the data to become more distinguishable. Clusters can then be identified by searching for dense regions in the new domain. Wavelet transforms are also discussed in Chapter 2, where they are used

"*Why is wavelet transformation useful for clustering?*" It offers the following advantages:

- *It provides unsupervised clustering.* It uses hat-shaped filters that emphasize regions where the points cluster, while suppressing weaker information outside of the cluster boundaries. Thus, dense regions in the original feature space act as attractors for nearby points and as inhibitors for points that are further away. This means that the clusters in the data automatically stand out and "clear" the regions around them. Thus, another advantage is that wavelet transformation can automatically result in the removal of outliers.
- *The multiresolution property of wavelet transformations can help detect clusters at varying levels of accuracy.* For example, Figure 7.16 shows a sample of two-dimensional feature space, where each point in the image represents the attribute or feature values of one object in the spatial data set. Figure 7.17 shows the resulting wavelet transformation at different resolutions, from a fine scale (scale 1) to a coarse scale (scale 3). At each level, the four subbands into which the original

  data are decomposed are shown. The subband shown in the upper-left quadrant emphasizes the average neighborhood around each data point. The subband in the upper-right quadrant emphasizes the horizontal edges of the data. The subband in the lower-left quadrant emphasizes the vertical edges, while the subband in the lower-right quadrant emphasizes the corners.
- *Wavelet-based clustering is very fast,* with a computational complexity of $O(n)$, where n is the number of objects in the database. The algorithm implementation can be made parallel.

WaveCluster is a grid-based and density-based algorithm. It conforms with many of the requirements of a good clustering algorithm: It handles large data sets efficiently, discovers clusters with arbitrary shape, successfully handles outliers, is insensitive to the order of input, and does not require the specification of input parameters such as the number of clusters or a neighborhood radius. In experimental studies, WaveCluster was found to outperform BIRCH, CLARANS, and DBSCAN in terms of both efficiency and clustering quality. The study also found WaveCluster capable of handling data with up to 20 dimensions.

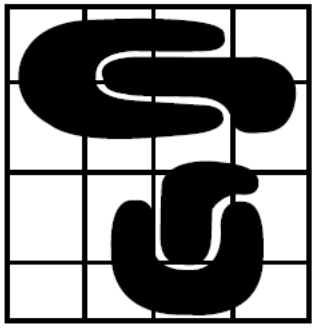


Figure 7.16 A sample of two-dimensional feature space. From [SCZ98].

(a) (b) (c)

Figure 7.17 Multiresolution of the feature space in Figure 7.16 at (a) scale 1 (high resolution); (b) scale 2 (medium resolution); and (c) scale 3 (low resolution). From [SCZ98].

5) Model-based methods: Model-based methods hypothesize a model for each of the clusters and find the best fit of the data to the given model. A model-based algorithm may locate clusters by constructing a density function that reflects the spatial distribution of the data points. It also leads to a way of automatically determining the number of clusters based on standard

statistics, taking "noise" or outliers into account and thus yielding robust clustering methods.

EM is an algorithm that performs expectation-maximization analysis based on statistical modeling. COBWEB is a conceptual learning algorithm that performs probability analysis and takes concepts as a model for clusters. SOM (or self-organizing feature map) is a neural network-based algorithm that clusters by mapping high dimensional data into a 2-D or 3-D feature map, which is also useful for data visualization.

6) Constraint-based clustering: Aside from the above categories of clustering methods, there are two classes of clustering tasks that require special attention. One is clustering high-dimensional data, and the other is constraint-based clustering.

Clustering high-dimensional data is a particularly important task in cluster analysis because many applications require the analysis of objects containing a large number of features or dimensions.

Constraint-based clustering is a clustering approach that performs clustering by incorporation of user-specified or application-oriented constraints. A constraint expresses a user's expectation or describes "properties" of the desired clustering results, and provides an effective means for communicating with the clustering process. Various kinds of constraints can be specified, either by a user or as per application requirements.

OUTLIER ANALYSIS

"What is an outlier?"

Very often, there exist data objects that do not comply with the general behavior or model of the data. Such data objects, which are grossly different from or inconsistent with the remaining set of data, are called outliers. Outliers can be caused by measurement or execution error.

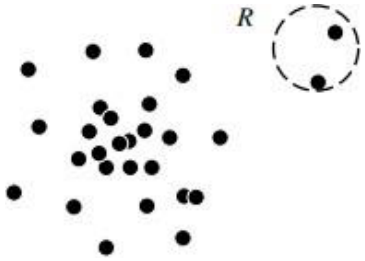


Fig: Outliers

Many data mining algorithms try to minimize the influence of outliers or eliminate them all together. This, however, could result in the loss of important hidden information because one person's noise could be another person's signal. In other words, the outliers may be of particular interest, such as in the case of fraud detection, where outliers may indicate fraudulent activity. Thus, outlier detection and analysis is an interesting data mining task, referred to as outlier mining.

Outlier mining has wide applications. As mentioned previously, it can be used in fraud detection, for example, by detecting unusual usage of credit cards or telecommunication services. In addition, it is useful in customized marketing for identifying the spending behavior of customers with extremely low or extremely high incomes, or in medical analysis for finding unusual responses to various medical treatments.

Outlier mining can be described as follows: Given a set of n data points or objects and k, the expected number of outliers, find the top k objects that are considerably dissimilar, exceptional, or inconsistent with respect to the remaining data. The outlier mining problem can be viewed as two subproblems: (1) define what data can be considered as inconsistent in a given data set, and (2) find an efficient method to mine the outliers so defined.

"What about using data visualization methods for outlier detection?"

This may seem like an obvious choice, since human eyes are very fast and effective at noticing data inconsistencies. However, this does not apply to data containing cyclic plots, where values that appear to be outliers could be perfectly valid values in reality. Data visualization methods are weak in detecting outliers in data with many categorical attributes or in data of high

dimensionality, since human eyes are good at visualizing numeric data of only two to three dimensions.

Procedures for detecting outliers: There are two basic types of procedures for detecting outliers:

1) **Block procedures:** In this case, either all of the suspect objects are treated as outliers or all of them are accepted as consistent.
2) **Consecutive (or sequential) procedures:** An example of such a procedure is the inside out procedure. Its main idea is that the object that is least "likely" to be an outlier is tested first. If it is found to be an outlier, then all of the more extreme values are also considered outliers; otherwise, the next most extreme object is tested, and so on. This procedure tends to be more effective than block procedures.

*****

UNIT – V

ADVANCED CONCEPTS

MINING DATA STREAMS

Tremendous and potentially infinite volumes of data streams are often generated by real-time surveillance systems, communication networks, Internet traffic, on-line transactions in the financial market or retail industry, electric power grids, industry production processes, scientific and engineering experiments, remote sensors, and other dynamic environments. Unlike traditional data sets, stream data flow in and out of a computer system continuously and with varying update rates. They are temporally ordered, fast changing, massive, and potentially infinite. It may be impossible to store an entire data stream or to scan through it multiple times due to its tremendous volume. Moreover, stream data tend to be of a rather low level of abstraction, whereas most analysts are interested in relatively high-level dynamic changes, such as trends and deviations. To discover knowledge or patterns from data streams, it is necessary to develop single-scan, on-line, multilevel, multidimensional stream processing and analysis methods.

Such single-scan, on-line data analysis methodology should not be confined to only stream data. It is also critically important for processing non-stream data that are massive. With data volumes mounting by terabytes or even petabytes, stream data nicely capture our data processing needs of today: even when the complete set of data is collected and can be stored in massive data storage devices, single scan (as in data stream systems) instead of random access (as in database systems) may still be the most realistic processing mode, because it is often too expensive to scan such a data set multiple times.

Frequent-Pattern Mining in Data Streams: Frequent-pattern mining finds a set of patterns that occur frequently in a data set, where a pattern can be a set of items (called an itemset), a subsequence, or a substructure. A pattern is considered frequent if its count satisfies a minimum support. Scalable methods for mining frequent patterns have been extensively studied for static

data sets. However, mining such patterns in dynamic data streams poses substantial new challenges. Many existing frequent-pattern mining algorithms require the system to scan the whole data set more than once, but this is unrealistic for infinite data streams. How can we perform incremental updates of frequent itemsets for stream data since an infrequent itemset can become frequent later on, and hence cannot be ignored? Moreover, a frequent itemset can become infrequent as well. The number of infrequent itemsets is exponential and so it is impossible to keep track of all of them.

To overcome this difficulty, there are two possible approaches. One is to keep track of only a predefined, limited set of items and itemsets. This method has very limited usage and expressive power because it requires the system to confine the scope of examination to only the set of predefined itemsets beforehand. The second approach is to derive an approximate set of answers. In practice, approximate answers are often sufficient. A number of approximate item or itemset counting algorithms have been developed in recent research. Here we introduce one such algorithm: the Lossy Counting algorithm. It approximates the frequency of items or itemsets within a user-specified error bound, ε.

MINING TIME – SERIES DATA

"What is a time-series database?"

A time-series database consists of sequences of values or events obtained over repeated measurements of time. The values are typically measured at equal time intervals (e.g., hourly, daily, weekly). Time-series databases are popular in many applications, such as stock market analysis, economic and sales forecasting, budgetary analysis, utility studies, inventory studies, yield projections, workload projections, process and quality control, observation of natural phenomena (such as atmosphere, temperature, wind, earthquake), scientific and engineering experiments, and medical treatments. A time-series database is also a sequence database. However, a sequence database is any database that consists of sequences of ordered events, with or without concrete notions of time. For example, Web page traversal sequences and

customer shopping transaction sequences are sequence data, but they may not be time-series data.

With the growing deployment of a large number of sensors, telemetry devices, and other on-line data collection tools, the amount of time-series data is increasing rapidly, often in the order of gigabytes per day (such as in stock trading) or even per minute (such as from NASA space programs). How can we find correlation relationships within time-series data? How can we analyze such huge numbers of time series to find similar or regular patterns, trends, bursts (such as sudden sharp changes), and outliers, with fast or even on-line real-time response? This has become an increasingly important and challenging problem.

Trend Analysis:

A time series involving a variable Y, representing, say, the daily closing price of a share in a stock market, can be viewed as a function of time t, that is, $Y = F(t)$. Such a function can be illustrated as a time-series graph, as shown in Figure 8.4, which describes a point moving with the passage of time.

"*How can we study time-series data?*" In general, there are two goals in time-series analysis: (1) *modeling time series* (i.e., to gain insight into the mechanisms or underlying forces that generate the time series), and (2) *forecasting time series* (i.e., to predict the future values of the time-series variables).

Trend analysis consists of the following four major **components** or **movements** for characterizing time-series data:

- **Trend or long-term movements**: These indicate the general direction in which a time-series graph is moving over a long interval of time. This movement is displayed by a **trend curve**, or a **trend line**. For example, the trend curve of Figure 8.4 is indicated by a dashed curve. Typical methods for determining a trend curve or trend line include the *weighted moving average method* and the *least squares method*, discussed later.
- **Cyclic movements or cyclic variations**: These refer to the *cycles*, that is, the long-term oscillations about a trend line or curve, which may or may not be periodic. That is, the cycles need not necessarily follow exactly similar patterns after equal intervals of time.

- **Seasonal movements or seasonal variations:** These are systematic or calendar related. Examples include events that recur annually, such as the sudden increase in sales of chocolates and flowers before Valentine's Day or of department store items before Christmas. The observed increase in water consumption in summer due to warm weather is another example. In these examples, seasonal movements are the identical or nearly identical patterns that a time series appears to follow during corresponding months of successive years.
- **Irregular or random movements:** These characterize the sporadic motion of time series due to random or chance events, such as labor disputes, floods, or announced personnel changes within companies.

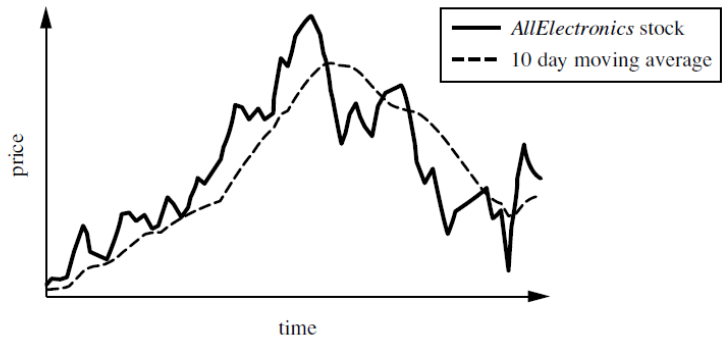


Figure 8.4 Time-series data of the stock price of *AllElectronics* over time. The *trend* is shown with a dashed curve, calculated by a moving average.

"How can we determine the trend of the data?"

A common method for determining trend is to calculate a moving average of order n as the following sequence of arithmetic means:

$$\frac{y_1 + y_2 + \cdots + y_n}{n}, \frac{y_2 + y_3 + \cdots + y_{n+1}}{n}, \frac{y_3 + y_4 + \cdots + y_{n+2}}{n}, \cdots \qquad (8.5)$$

A moving average tends to reduce the amount of variation present in the data set. Thus the process of replacing the time series by its moving average eliminates unwanted fluctuations and is therefore also referred to as the **smoothing of time series**. If weighted arithmetic means are used in Sequence (8.5), the resulting sequence is called a **weighted moving average of order** *n*.

Similarity Search in Time-Series Analysis: "What is a similarity search?" Unlike normal database queries, which find data that match the given query exactly, a similarity search finds data sequences that differ only slightly from the given query sequence. Given a set of time-series sequences, S, there are two types of similarity searches: subsequence matching and whole sequence matching. Subsequence matching finds the sequences in S that contain sub-

sequences that are similar to a given query sequence x, while whole sequence matching finds a set of sequences in S that are similar to each other (as a whole). Subsequence matching is a more frequently encountered problem in applications. Similarity search in time-series analysis is useful for financial market analysis (e.g., stock data analysis), medical diagnosis (e.g., cardiogram analysis), and in scientific or engineering databases (e.g., power consumption analysis).

MINING SEQUENCE PATTERNS IN TRANSACTIONAL DATABASES

"What is sequential pattern mining?"

Sequential pattern mining is the mining of frequently occurring ordered events or sub-sequences as patterns. An example of a sequential pattern is "Customers who buy a Canon digital camera are likely to buy an HP color printer within a month." For retail data, sequential patterns are useful for shelf placement and promotions. This industry, as well as telecommunications and other businesses, may also use sequential patterns for targeted marketing, customer retention, and many other tasks. Other areas in which sequential patterns can be applied include Web access pattern analysis, weather prediction, production processes, and network intrusion detection.

The sequential pattern mining problem was first introduced by Agrawal and Srikant in 1995 [AS95] based on their study of customer purchase sequences, as follows: "Given a set of sequences, where each sequence consists of a list of events (or elements) and each event consists of a set of items, and given a user-specified minimum support threshold of min sup, sequential pattern mining finds all frequent sub-sequences, that is, the sub-sequences whose occurrence frequency in the set of sequences is no less than min_sup."

Let's establish some vocabulary for our discussion of sequential pattern mining. Let $I = \{I_1, I_2, \ldots, I_p\}$ be the set of all *items*. An **itemset** is a nonempty set of items. A **sequence** is an ordered list of **events**. A sequence s is denoted $\langle e_1 e_2 e_3 \cdots e_l \rangle$, where event e_1 occurs before e_2, which occurs before e_3, and so on. Event e_j is also called an **element** of s. In the case of customer purchase data, an event refers to a shopping trip in which a customer bought items at a certain store. The event is thus an itemset, that is, an unordered list of items that the customer purchased during the trip. The itemset (or event) is denoted $(x_1 x_2 \cdots x_q)$, where x_k is an item. For brevity, the brackets are omitted if an element has only one item, that is, element (x) is written as x. Suppose that a customer made several shopping trips to the store. These ordered events form a sequence for the customer. That is, the customer first bought the items in s_1, then later bought the items in s_2, and so on. An item can occur at most once in an event of a sequence, but can occur multiple times in different events of a sequence. The number of instances of items in a sequence is called the **length** of the sequence. A sequence with length l is called an l-**sequence**. A sequence $\alpha = \langle a_1 a_2 \cdots a_n \rangle$ is called a **subsequence** of another sequence $\beta = \langle b_1 b_2 \cdots b_m \rangle$, and β is a **supersequence** of α, denoted as $\alpha \sqsubseteq \beta$, if there exist integers $1 \leq j_1 < j_2 < \cdots < j_n \leq m$ such that $a_1 \subseteq b_{j_1}, a_2 \subseteq b_{j_2}, \ldots, a_n \subseteq b_{j_n}$. For example, if $\alpha = \langle (ab), d \rangle$ and $\beta = \langle (abc), (de) \rangle$, where a, b, c, d, and e are items, then α is a subsequence of β and β is a supersequence of α.

A **sequence database**, S, is a set of tuples, $\langle SID, s \rangle$, where SID is a *sequence_ID* and s is a sequence. For our example, S contains sequences for all customers of the store. A tuple $\langle SID, s \rangle$ is said to **contain** a sequence α, if α is a subsequence of s. The **support** of a sequence α in a sequence database S is the number of tuples in the database containing α, that is, $support_S(\alpha) = |\{\langle SID, s \rangle | (\langle SID, s \rangle \in S) \wedge (\alpha \sqsubseteq s)\}|$. It can be denoted as $support(\alpha)$ if the sequence database is clear from the context. Given a positive integer min_sup as the **minimum support threshold**, a sequence α is **frequent** in sequence database S if $support_S(\alpha) \geq min\_sup$. That is, for sequence α to be frequent, it must occur at least min_sup times in S. *A frequent sequence is called a* **sequential pattern**. A sequential pattern with length l is called an l-**pattern**. The following example illustrates these concepts.

This model of sequential pattern mining is an abstraction of customer shopping sequence analysis.

MINING OBJECT – SPATIAL – MULTIMEDIA – TEXT AND WEB DATA

SPATIAL DATA MINING

A spatial database stores a large amount of space-related data, such as maps, pre-processed remote sensing or medical imaging data, and VLSI chip layout data. Spatial databases have many features distinguishing them from relational databases. They carry topological and/or distance information, usually organized by sophisticated, multidimensional spatial indexing

structures that are accessed by spatial data access methods and often require spatial reasoning, geometric computation, and spatial knowledge representation techniques.

Spatial data mining refers to the extraction of knowledge, spatial relationships, or other interesting patterns not explicitly stored in spatial databases. Such mining demands an integration of data mining with spatial database technologies. It can be used for understanding spatial data, discovering spatial relationships and relationships between spatial and nonspatial data, constructing spatial knowledge bases, reorganizing spatial databases, and optimizing spatial queries. It is expected to have wide applications in geographic information systems, geo marketing, remote sensing, image database exploration, medical imaging, navigation, traffic control, environmental studies, and many other areas where spatial data are used. A crucial challenge to spatial data mining is the exploration of efficient spatial data mining techniques due to the huge amount of spatial data and the complexity of spatial data types and spatial access methods.

“What about using statistical techniques for spatial data mining?”

Statistical spatial data analysis has been a popular approach to analyzing spatial data and exploring geographic information. The term geostatistics is often associated with continuous geographic space, whereas the term spatial statistics is often associated with discrete space. In a statistical model that handles nonspatial data, one usually assumes statistical independence among different portions of data. However, different from traditional data sets, there is no such independence among spatially distributed data because in reality, spatial objects are often interrelated, or more exactly spatially co-located, in the sense that the closer the two objects are located, the more likely they share similar properties. For example, nature resource, climate, temperature, and economic situations are likely to be similar in geographically closely located regions. People even consider this as the first law of geography: “Everything is related to everything else, but nearby things are more related than distant things.” Such a property of close interdependency across nearby space leads to the notion of spatial autocorrelation. Based on this notion,

spatial statistical modeling methods have been developed with good success. Spatial data mining will further develop spatial statistical analysis methods and extend them for huge amounts of spatial data, with more emphasis on efficiency, scalability, cooperation with database and data warehouse systems, improved user interaction, and the discovery of new types of knowledge.

MULTIMEDIA DATA MINING

"What is a multimedia database?"

A multimedia database system stores and manages a large collection of multimedia data, such as audio, video, image, graphics, speech, text, document, and hypertext data, which contain text, text markups, and linkages. Multimedia database systems are increasingly common owing to the popular use of audio video equipment, digital cameras, CD-ROMs, and the Internet. Typical multimedia database systems include NASA's EOS (Earth Observation System), various kinds of image and audio-video databases, and Internet databases.

Similarity Search in Multimedia Data: "When searching for similarities in multimedia data, can we search on either the data description or the data content?" That is correct. For similarity searching in multimedia data, we consider two main families of multimedia indexing and retrieval systems: (1) description-based retrieval systems, which build indices and perform object retrieval based on image descriptions, such as keywords, captions, size, and time of creation; and (2) content-based retrieval systems, which support retrieval based on the image content, such as color histogram, texture, pattern, image topology, and the shape of objects and their layouts and locations within the image. Description-based retrieval is labor-intensive if performed manually. If automated, the results are typically of poor quality. For example, the assignment of keywords to images can be a tricky and arbitrary task. Recent development of Web-based image clustering and classification methods has improved the quality of description-based Web image retrieval, because image surrounded text information as well as Web

linkage information can be used to extract proper description and group images describing a similar theme together.

Content-based retrieval uses visual features to index images and promotes object retrieval based on feature similarity, which is highly desirable in many applications. In a content-based image retrieval system, there are often two kinds of queries: image sample-based queries and image feature specification queries. Image-sample-based queries find all of the images that are similar to the given image sample. This search compares the feature vector (or signature) extracted from the sample with the feature vectors of images that have already been extracted and indexed in the image database. Based on this comparison, images that are close to the sample image are returned. Image feature specification queries specify or sketch image features like color, texture, or shape, which are translated into a feature vector to be matched with the feature vectors of the images in the database. Content-based retrieval has wide applications, including medical diagnosis, weather prediction, TV production, Web search engines for images, and e-commerce. Some systems, such as QBIC (Query By Image Content), support both sample-based and image feature specification queries. There are also systems that support both content based and description-based retrieval.

Several approaches have been proposed and studied for similarity-based retrieval in image databases, based on image signature:

1) **Color histogram–based signature:** In this approach, the signature of an image includes color histograms based on the color composition of an image regardless of its scale or orientation. This method does not contain any information about shape, image topology, or texture. Thus, two images with similar color composition but that contain very different shapes or textures may be identified as similar, although they could be completely unrelated semantically.
2) **Multi feature composed signature:** In this approach, the signature of an image includes a composition of multiple features: color histogram, shape, image topology, and texture. The extracted image features are stored as metadata, and images are indexed based on such metadata.

Often, separate distance functions can be defined for each feature and subsequently combined to derive the overall results. Multidimensional content-based search often uses one or a few probe features to search for images containing such (similar) features. It can therefore be used to search for similar images. This is the most popularly used approach in practice.

3) **Wavelet-based signature:** This approach uses the dominant wavelet coefficients of an image as its signature. Wavelets capture shape, texture, and image topology information in a single unified framework. This improves efficiency and reduces the need for providing multiple search primitives (unlike the second method above). However, since this method computes a single signature for an entire image, it may fail to identify images containing similar objects where the objects differ in location or size.
4) **Wavelet-based signature with region-based granularity:** In this approach, the computation and comparison of signatures are at the granularity of regions, not the entire image. This is based on the observation that similar images may contain similar regions, but a region in one image could be a translation or scaling of a matching region in the other. Therefore, a similarity measure between the query image Q and a target image T can be defined in terms of the fraction of the area of the two images covered by matching pairs of regions from Q and T. Such a region-based similarity search can find images containing similar objects, where these objects may be translated or scaled.

Mining Associations in Multimedia Data: "What kinds of associations can be mined in multimedia data?" Association rules involving multimedia objects can be mined in image and video databases. At least three categories can be observed:

1) **Associations between image content and nonimage content features:** A rule like "If at least 50% of the upper part of the picture is

blue, then it is likely to represent sky" belongs to this category since it links the image content to the keyword sky.

2) **Associations among image contents that are not related to spatial relationships:** A rule like "If a picture contains two blue squares, then it is likely to contain one red circle as well" belongs to this category since the associations are all regarding image contents.
3) **Associations among image contents related to spatial relationships:** A rule like "If a red triangle is between two yellow squares, then it is likely a big oval-shaped object is underneath" belongs to this category since it associates objects in the image with spatial relationships.

TEXT MINING

Most previous studies of data mining have focused on structured data, such as relational, transactional, and data warehouse data. However, in reality, a substantial portion of the available information is stored in text databases (or document databases), which consist of large collections of documents from various sources, such as news articles, research papers, books, digital libraries, e-mail messages, and Web pages. Text databases are rapidly growing due to the increasing amount of information available in electronic form, such as electronic publications, various kinds of electronic documents, e-mail, and the World Wide Web (which can also be viewed as a huge, interconnected, dynamic text database). Nowadays most of the information in government, industry, business, and other institutions are stored electronically, in the form of text databases.

Data stored in most text databases are semi-structured data in that they are neither completely unstructured nor completely structured. For example, a document may contain a few structured fields, such as title, authors, publication date, category, and so on, but also contain some largely unstructured text components, such as abstract and contents. There have been a great deal of studies on the modeling and implementation of semi-structured data in recent database research. Moreover, information retrieval

techniques, such as text indexing methods, have been developed to handle unstructured documents.

Traditional information retrieval techniques become inadequate for the increasingly vast amounts of text data. Typically, only a small fraction of the many available documents will be relevant to a given individual user. Without knowing what could be in the documents, it is difficult to formulate effective queries for analyzing and extracting useful information from the data. Users need tools to compare different documents, rank the importance and relevance of the documents, or find patterns and trends across multiple documents. Thus, text mining has become an increasingly popular and essential theme in data mining.

Text Mining Approaches: There are many approaches to text mining, which can be classified from different perspectives, based on the inputs taken in the text mining system and the data mining tasks to be performed. In general, the major approaches, based on the kinds of data they take as input, are: (1) the keyword-based approach, where the input is a set of keywords or terms in the documents, (2) the tagging approach, where the input is a set of tags, and (3) the information-extraction approach, which inputs semantic information, such as events, facts, or entities uncovered by information extraction. A simple keyword-based approach may only discover relationships at a relatively shallow level, such as rediscovery of compound nouns (e.g., “database” and “systems”) or co-occurring patterns with less significance (e.g., “terrorist” and “explosion”). It may not bring much deep understanding to the text. The tagging approach may rely on tags obtained by manual tagging (which is costly and is unfeasible for large collections of documents) or by some automated categorization algorithm (which may process a relatively small set of tags and require defining the categories before hand). The information-extraction approach is more advanced and may lead to the discovery of some deep knowledge, but it requires semantic analysis of text by natural language understanding and machine learning methods. This is a challenging knowledge discovery task.

MINING THE WORLD WIDE WEB (WWW)

The World Wide Web serves as a huge, widely distributed, global information service center for news, advertisements, consumer information, financial management, education, government, e-commerce, and many other information services. The Web also contains a rich and dynamic collection of hyperlink information and Web page access and usage information, providing rich sources for data mining. However, based on the following observations, the Web also poses great challenges for effective resource and knowledge discovery.

⇒ **The Web seems to be too huge for effective data warehousing and data mining.** The size of the Web is in the order of hundreds of terabytes and is still growing rapidly. Many organizations and societies place most of their public-accessible information on the Web. It is barely possible to set up a data warehouse to replicate, store, or integrate all of the data on the Web.

⇒ **The complexity of Web pages is far greater than that of any traditional text document collection.** Web pages lack a unifying structure. They contain far more authoring style and content variations than any set of books or other traditional text-based documents. The Web is considered a huge digital library; however, the tremendous number of documents in this library are not arranged according to any particular sorted order. There is no index by category, nor by title, author, cover page, table of contents, and so on. It can be very challenging to search for the information you desire in such a library!

⇒ **The Web is a highly dynamic information source.** Not only does the Web grow rapidly, but its information is also constantly updated. News, stock markets, weather, sports, shopping, company advertisements, and numerous other Web pages are updated regularly on the Web. Linkage information and access records are also updated frequently. The Web serves a broad diversity of user communities. The Internet currently connects more than 100 million workstations, and its user community is still rapidly expanding. Users may have very different

backgrounds, interests, and usage purposes. Most users may not have good knowledge of the structure of the information network and may not be aware of the heavy cost of a particular search. They can easily get lost by groping in the "darkness" of the network, or become bored by taking many access "hops" and waiting impatiently for a piece of information.

⇒ **Only a small portion of the information on the Web is truly relevant or useful.** It is said that 99% of the Web information is useless to 99% of Web users. Although this may not seem obvious, it is true that a particular person is generally interested in only a tiny portion of the Web, while the rest of the Web contains information that is uninteresting to the user and may swamp desired search results. How can the portion of the Web that is truly relevant to your interest be determined? How can we find high quality Web pages on a specified topic?

These challenges have promoted research into efficient and effective discovery and use of resources on the Internet.

There are many index-based Web search engines. These search the Web, index Web pages, and build and store huge keyword-based indices that help locate sets of Web pages containing certain keywords. With such search engines, an experienced user may be able to quickly locate documents by providing a set of tightly constrained keywords and phrases. However, a simple keyword-based search engine suffers from several deficiencies. First, a topic of any breadth can easily contain hundreds of thousands of documents. This can lead to a huge number of document entries returned by a search engine, many of which are only marginally relevant to the topic or may contain materials of poor quality. Second, many documents that are highly relevant to a topic may not contain keywords defining them. This is referred to as the polysemy problem. For example, the keyword Java may refer to the Java programming language, or an island in Indonesia, or brewed coffee. As another example, a search based on the keyword search engine may not find even the most popular Web search engines like Google, Yahoo!,

AltaVista, or America Online if these services do not claim to be search engines on their Web pages. This indicates that a simple keyword based Web search engine is not sufficient for Web resource discovery.

"If a keyword-based Web search engine is not sufficient for Web resource discovery, how can we even think of doing Web mining?" Compared with keyword-based Web search, Web mining is a more challenging task that searches for Web structures, ranks the importance of Web contents, discovers the regularity and dynamics of Web contents, and mines Web access patterns. However, Web mining can be used to substantially enhance the power of a Web search engine since Web mining may identify authoritative Web pages, classify Web documents, and resolve many ambiguities and subtleties raised in keyword-based Web search. In general, Web mining tasks can be classified into three categories: Web content mining, Web structure mining, and Web usage mining. Alternatively, Web structures can be treated as a part of Web contents so that Web mining can instead be simply classified into Web content mining and Web usage mining.

*****

Prepared By:

RIYAZ MOHAMMED