1 Introduction

This book is an introduction to the young and fast-growing field of *data mining* (also known as *knowledge discovery from data*, or *KDD* for short). The book focuses on fundamental data mining concepts and techniques for discovering interesting patterns from data in various applications. In particular, we emphasize prominent techniques for developing effective, efficient, and scalable data mining tools.

This chapter is organized as follows. In Section 1.1, you will learn why data mining is in high demand and how it is part of the natural evolution of information technology. Section 1.2 defines data mining with respect to the knowledge discovery process. Next, you will learn about data mining from many aspects, such as the kinds of data that can be mined (Section 1.3), the kinds of knowledge to be mined (Section 1.4), the kinds of technologies to be used (Section 1.5), and targeted applications (Section 1.6). In this way, you will gain a multidimensional view of data mining. Finally, Section 1.7 outlines major data mining research and development issues.

1.1 Why Data Mining?

Necessity, who is the mother of invention. – Plato

We live in a world where vast amounts of data are collected daily. Analyzing such data is an important need. Section 1.1.1 looks at how data mining can meet this need by providing tools to discover knowledge from data. In Section 1.1.2, we observe how data mining can be viewed as a result of the natural evolution of information technology.

1.1.1 Moving toward the Information Age

"*We are living in the information age*" is a popular saying; however, *we are actually living in the data age*. Terabytes or petabytes[1] of data pour into our computer networks, the World Wide Web (WWW), and various data storage devices every day from business,

[1] A petabyte is a unit of information or computer storage equal to 1 quadrillion bytes, or a thousand terabytes, or 1 million gigabytes.

society, science and engineering, medicine, and almost every other aspect of daily life. This explosive growth of available data volume is a result of the computerization of our society and the fast development of powerful data collection and storage tools. Businesses worldwide generate gigantic data sets, including sales transactions, stock trading records, product descriptions, sales promotions, company profiles and performance, and customer feedback. For example, large stores, such as Wal-Mart, handle hundreds of millions of transactions per week at thousands of branches around the world. Scientific and engineering practices generate high orders of petabytes of data in a continuous manner, from remote sensing, process measuring, scientific experiments, system performance, engineering observations, and environment surveillance.

Global backbone telecommunication networks carry tens of petabytes of data traffic every day. The medical and health industry generates tremendous amounts of data from medical records, patient monitoring, and medical imaging. Billions of Web searches supported by search engines process tens of petabytes of data daily. Communities and social media have become increasingly important data sources, producing digital pictures and videos, blogs, Web communities, and various kinds of social networks. The list of sources that generate huge amounts of data is endless.

This explosively growing, widely available, and gigantic body of data makes our time truly the data age. Powerful and versatile tools are badly needed to automatically uncover valuable information from the tremendous amounts of data and to transform such data into organized knowledge. This necessity has led to the birth of data mining. The field is young, dynamic, and promising. Data mining has and will continue to make great strides in our journey from the data age toward the coming information age.

Example 1.1 **Data mining turns a large collection of data into knowledge.** A search engine (e.g., Google) receives hundreds of millions of queries every day. Each query can be viewed as a transaction where the user describes her or his information need. What novel and useful knowledge can a search engine learn from such a huge collection of queries collected from users over time? Interestingly, some patterns found in user search queries can disclose invaluable knowledge that cannot be obtained by reading individual data items alone. For example, Google's *Flu Trends* uses specific search terms as indicators of flu activity. It found a close relationship between the number of people who search for flu-related information and the number of people who actually have flu symptoms. A pattern emerges when all of the search queries related to flu are aggregated. Using aggregated Google search data, *Flu Trends* can estimate flu activity up to two weeks faster than traditional systems can.[2] This example shows how data mining can turn a large collection of data into knowledge that can help meet a current global challenge. ■

1.1.2 Data Mining as the Evolution of Information Technology

Data mining can be viewed as a result of the natural evolution of information technology. The database and data management industry evolved in the development of

[2]This is reported in [GMP+09].

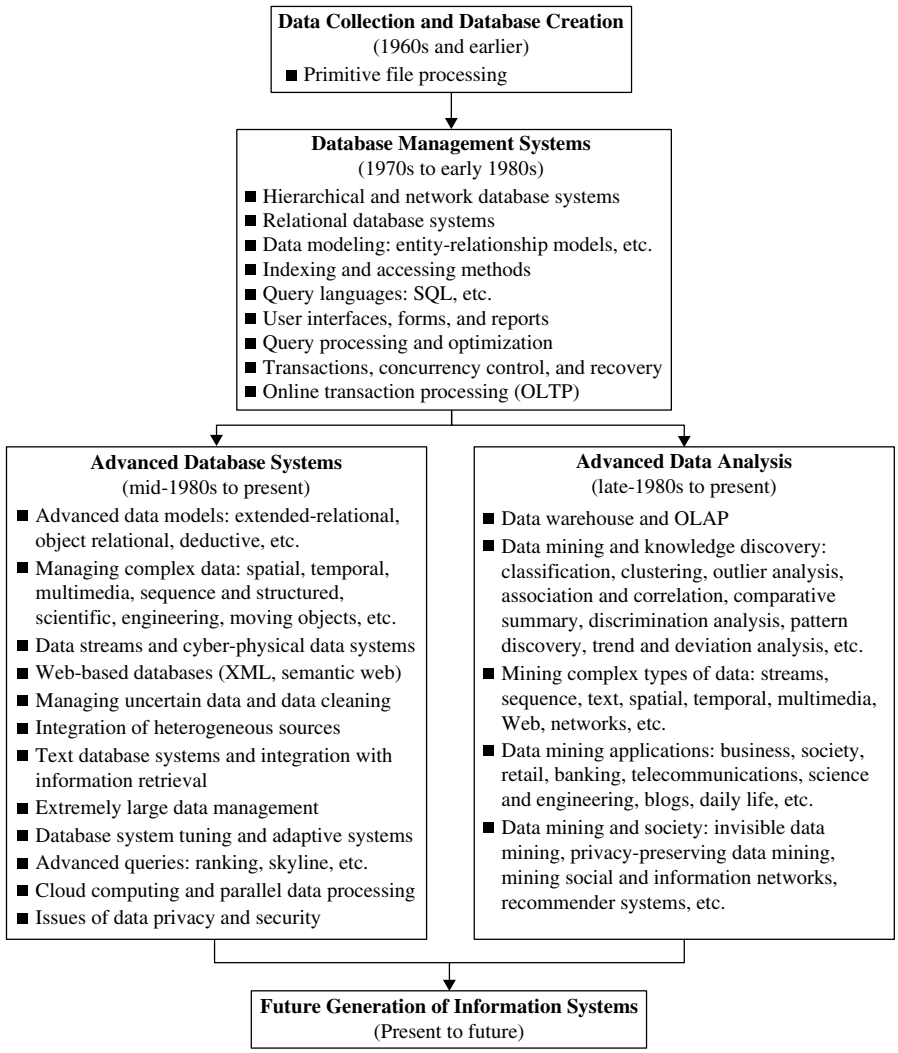


Figure 1.1 The evolution of database system technology.

several critical functionalities (Figure 1.1): *data collection and database creation, data management* (including data storage and retrieval and database transaction processing), and *advanced data analysis* (involving data warehousing and data mining). The early development of data collection and database creation mechanisms served as a prerequisite for the later development of effective mechanisms for data storage and retrieval, as well as query and transaction processing. Nowadays numerous database systems offer query and transaction processing as common practice. Advanced data analysis has naturally become the next step.

Since the 1960s, database and information technology has evolved systematically from primitive file processing systems to sophisticated and powerful database systems. The research and development in database systems since the 1970s progressed from early hierarchical and network database systems to relational database systems (where data are stored in relational table structures; see Section 1.3.1), data modeling tools, and indexing and accessing methods. In addition, users gained convenient and flexible data access through query languages, user interfaces, query optimization, and transaction management. Efficient methods for online transaction processing (OLTP), where a query is viewed as a read-only transaction, contributed substantially to the evolution and wide acceptance of relational technology as a major tool for efficient storage, retrieval, and management of large amounts of data.

After the establishment of database management systems, database technology moved toward the development of *advanced database systems*, *data warehousing*, and *data mining* for advanced data analysis and *web-based databases*. Advanced database systems, for example, resulted from an upsurge of research from the mid-1980s onward. These systems incorporate new and powerful data models such as extended-relational, object-oriented, object-relational, and deductive models. Application-oriented database systems have flourished, including spatial, temporal, multimedia, active, stream and sensor, scientific and engineering databases, knowledge bases, and office information bases. Issues related to the distribution, diversification, and sharing of data have been studied extensively.

Advanced data analysis sprang up from the late 1980s onward. The steady and dazzling progress of computer hardware technology in the past three decades led to large supplies of powerful and affordable computers, data collection equipment, and storage media. This technology provides a great boost to the database and information industry, and it enables a huge number of databases and information repositories to be available for transaction management, information retrieval, and data analysis. Data can now be stored in many different kinds of databases and information repositories.

One emerging data repository architecture is the **data warehouse** (Section 1.3.2). This is a repository of multiple heterogeneous data sources organized under a unified schema at a single site to facilitate management decision making. Data warehouse technology includes data cleaning, data integration, and online analytical processing (OLAP)—that is, analysis techniques with functionalities such as summarization, consolidation, and aggregation, as well as the ability to view information from different angles. Although OLAP tools support multidimensional analysis and decision making, additional data analysis tools are required for in-depth analysis—for example, data mining tools that provide data classification, clustering, outlier/anomaly detection, and the characterization of changes in data over time.

Huge volumes of data have been accumulated beyond databases and data warehouses. During the 1990s, the World Wide Web and web-based databases (e.g., XML databases) began to appear. Internet-based global information bases, such as the WWW and various kinds of interconnected, heterogeneous databases, have emerged and play a vital role in the information industry. The effective and efficient analysis of data from such different forms of data by integration of information retrieval, data mining, and information network analysis technologies is a challenging task.

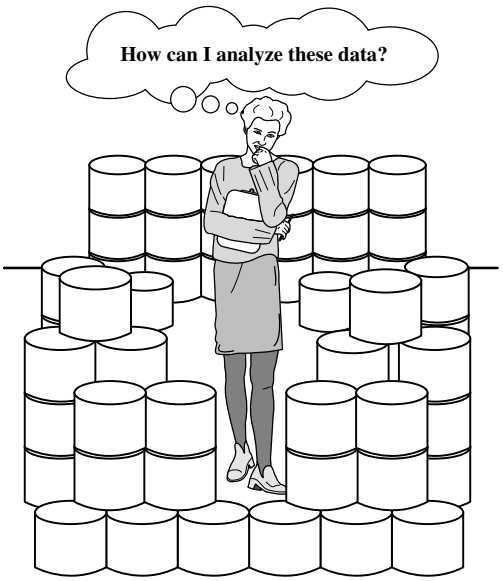


Figure 1.2 The world is data rich but information poor.

In summary, the abundance of data, coupled with the need for powerful data analysis tools, has been described as a *data rich but information poor* situation (Figure 1.2). The fast-growing, tremendous amount of data, collected and stored in large and numerous data repositories, has far exceeded our human ability for comprehension without powerful tools. As a result, data collected in large data repositories become "data tombs"—data archives that are seldom visited. Consequently, important decisions are often made based not on the information-rich data stored in data repositories but rather on a decision maker's intuition, simply because the decision maker does not have the tools to extract the valuable knowledge embedded in the vast amounts of data. Efforts have been made to develop expert system and knowledge-based technologies, which typically rely on users or domain experts to *manually* input knowledge into knowledge bases. Unfortunately, however, the manual knowledge input procedure is prone to biases and errors and is extremely costly and time consuming. The widening gap between data and information calls for the systematic development of *data mining tools* that can turn data tombs into "golden nuggets" of knowledge.

1.2 What Is Data Mining?

It is no surprise that data mining, as a truly interdisciplinary subject, can be defined in many different ways. Even the term *data mining* does not really present all the major components in the picture. To refer to the mining of gold from rocks or sand, we say *gold mining* instead of rock or sand mining. Analogously, data mining should have been more

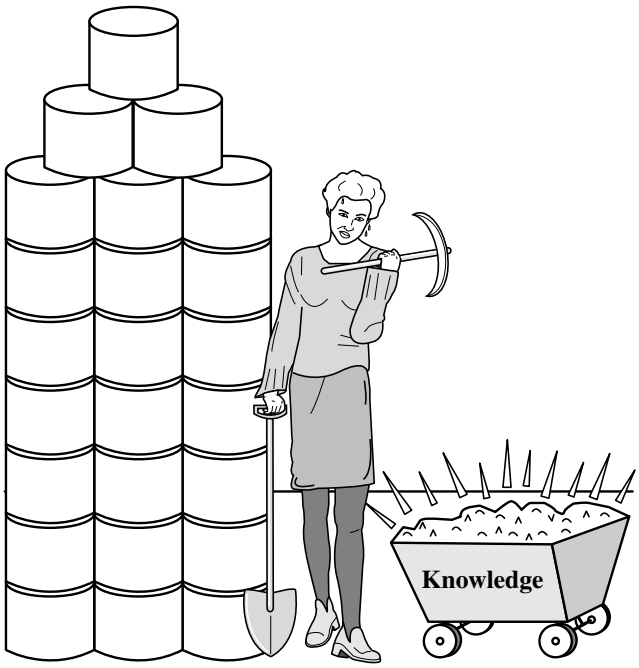


Figure 1.3 Data mining—searching for knowledge (interesting patterns) in data.

appropriately named "knowledge mining from data," which is unfortunately somewhat long. However, the shorter term, *knowledge mining* may not reflect the emphasis on mining from large amounts of data. Nevertheless, mining is a vivid term characterizing the process that finds a small set of precious nuggets from a great deal of raw material (Figure 1.3). Thus, such a misnomer carrying both "data" and "mining" became a popular choice. In addition, many other terms have a similar meaning to data mining—for example, *knowledge mining from data*, *knowledge extraction*, *data/pattern analysis*, *data archaeology*, and *data dredging*.

Many people treat data mining as a synonym for another popularly used term, **knowledge discovery from data**, or **KDD**, while others view data mining as merely an essential step in the process of knowledge discovery. The knowledge discovery process is shown in Figure 1.4 as an iterative sequence of the following steps:

1. **Data cleaning** (to remove noise and inconsistent data)
2. **Data integration** (where multiple data sources may be combined)[3]

[3]A popular trend in the information industry is to perform data cleaning and data integration as a preprocessing step, where the resulting data are stored in a data warehouse.

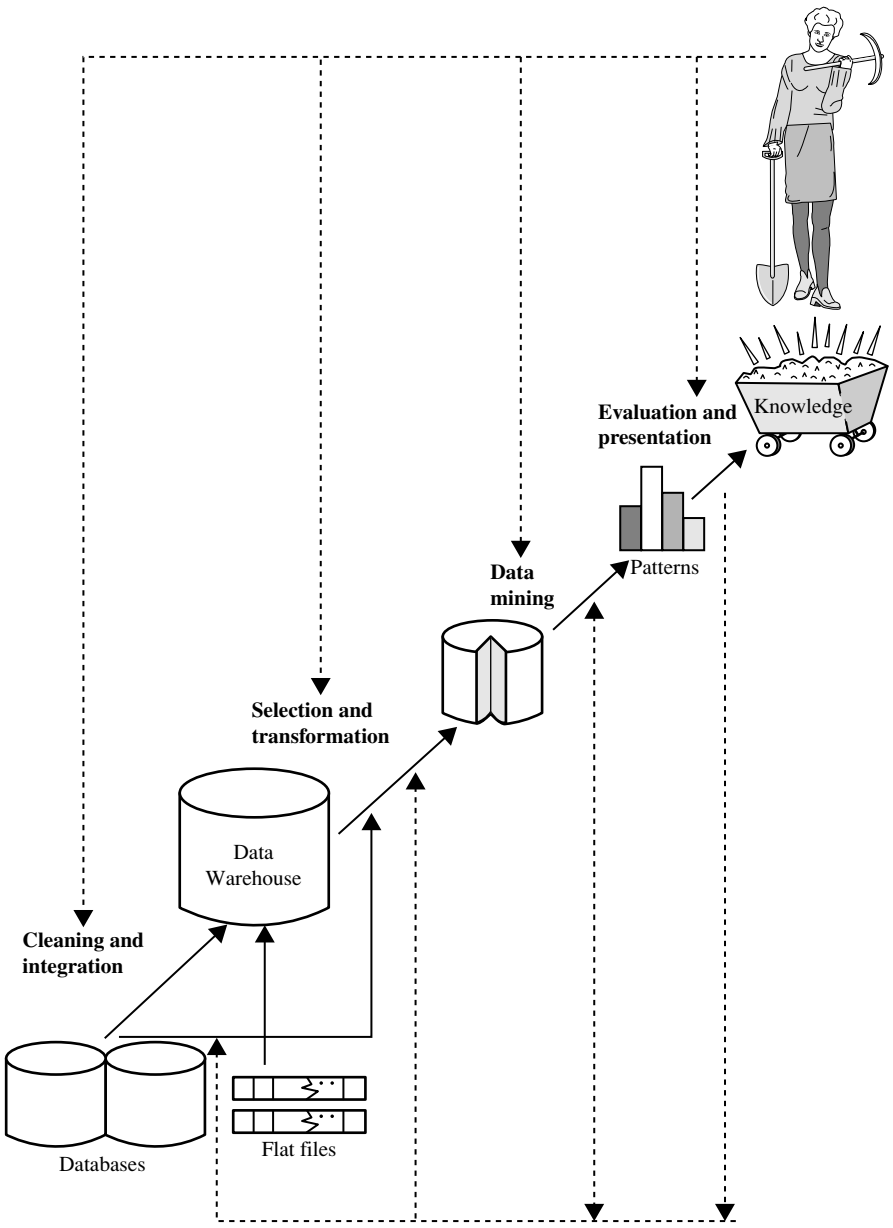


Figure 1.4 Data mining as a step in the process of knowledge discovery.

3. **Data selection** (where data relevant to the analysis task are retrieved from the database)
4. **Data transformation** (where data are transformed and consolidated into forms appropriate for mining by performing summary or aggregation operations)[4]
5. **Data mining** (an essential process where intelligent methods are applied to extract data patterns)
6. **Pattern evaluation** (to identify the truly interesting patterns representing knowledge based on *interestingness measures*—see Section 1.4.6)
7. **Knowledge presentation** (where visualization and knowledge representation techniques are used to present mined knowledge to users)

Steps 1 through 4 are different forms of data preprocessing, where data are prepared for mining. The data mining step may interact with the user or a knowledge base. The interesting patterns are presented to the user and may be stored as new knowledge in the knowledge base.

The preceding view shows data mining as one step in the knowledge discovery process, albeit an essential one because it uncovers hidden patterns for evaluation. However, in industry, in media, and in the research milieu, the term *data mining* is often used to refer to the entire knowledge discovery process (perhaps because the term is shorter than *knowledge discovery from data*). Therefore, we adopt a broad view of data mining functionality: **Data mining** is the *process* of discovering interesting patterns and knowledge from *large* amounts of data. The data sources can include databases, data warehouses, the Web, other information repositories, or data that are streamed into the system dynamically.

1.3 What Kinds of Data Can Be Mined?

As a general technology, data mining can be applied to any kind of data as long as the data are meaningful for a target application. The most basic forms of data for mining applications are database data (Section 1.3.1), data warehouse data (Section 1.3.2), and transactional data (Section 1.3.3). The concepts and techniques presented in this book focus on such data. Data mining can also be applied to other forms of data (e.g., data streams, ordered/sequence data, graph or networked data, spatial data, text data, multimedia data, and the WWW). We present an overview of such data in Section 1.3.4. Techniques for mining of these kinds of data are briefly introduced in Chapter 13. In-depth treatment is considered an advanced topic. Data mining will certainly continue to embrace new data types as they emerge.

[4]Sometimes data transformation and consolidation are performed before the data selection process, particularly in the case of data warehousing. *Data reduction* may also be performed to obtain a smaller representation of the original data without sacrificing its integrity.

1.3.1 Database Data

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 provide mechanisms for defining database structures and data storage; for specifying and managing concurrent, shared, or distributed data access; and for ensuring 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.

Example 1.2 A relational database for *AllElectronics*. The fictitious *AllElectronics* store is used to illustrate concepts throughout this book. The company is described by the following relation tables: *customer, item, employee*, and *branch*. The headers of the tables described here are shown in Figure 1.5. (A header is also called the *schema* of a relation.)

- The relation *customer* consists of a set of attributes describing the customer information, including a unique customer identity number (*cust_ID*), customer name, address, age, occupation, annual income, credit information, and category.
- Similarly, each of the relations *item*, *employee*, and *branch* consists of a set of attributes describing the properties of these entities.
- Tables can also be used to represent the relationships between or among multiple entities. In our example, these include *purchases* (customer purchases items, creating a sales transaction handled by an employee), *items_sold* (lists items sold in a given transaction), and *works_at* (employee works at a branch of *AllElectronics*). ■

customer	(*cust_ID, name, address, age, occupation, annual_income, credit_information, category*, ...)
item	(*item_ID, brand, category, type, price, place_made, supplier, cost*, ...)
employee	(*empl_ID, name, category, group, salary, commission*, ...)
branch	(*branch_ID, name, address*, ...)
purchases	(*trans_ID, cust_ID, empl_ID, date, time, method_paid, amount*)
items_sold	(*trans_ID, item_ID, qty*)
works_at	(*empl_ID, branch_ID*)

Figure 1.5 Relational schema for a relational database, *AllElectronics*.

Relational data can be accessed by **database queries** written in a relational query language (e.g., SQL) or with the assistance of graphical user interfaces. A given query is transformed into a set of relational operations, such as join, selection, and projection, and is then optimized for efficient processing. A query allows retrieval of specified subsets of the data. Suppose that your job is to analyze the *AllElectronics* data. Through the use of relational queries, you can ask things like, "*Show me a list of all items that were sold in the last quarter.*" Relational languages also use aggregate functions such as sum, avg (average), count, max (maximum), and min (minimum). Using aggregates allows you to ask: "*Show me the total sales of the last month, grouped by branch,*" or "*How many sales transactions occurred in the month of December?*" or "*Which salesperson had the highest sales?*"

When **mining relational databases**, we can go further by *searching for trends* or *data patterns*. For example, data mining systems can analyze customer data to predict the credit risk of new customers based on their income, age, and previous credit information. Data mining systems may also detect deviations—that is, items with sales that are far from those expected in comparison with the previous year. Such deviations can then be further investigated. For example, data mining may discover that there has been a change in packaging of an item or a significant increase in price.

Relational databases are one of the most commonly available and richest information repositories, and thus they are a major data form in the study of data mining.

1.3.2 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 usually residing at a single site. Data warehouses are constructed via a process of data cleaning, data integration, data transformation, data loading, and periodic data refreshing. This process is discussed in Chapters 3 and 4. Figure 1.6 shows the typical framework for construction and use of a data warehouse for *AllElectronics*.

To facilitate decision making, the data in a data warehouse are organized around *major subjects* (e.g., customer, item, supplier, and activity). The data are stored to provide information from a *historical perspective*, such as in the past 6 to 12 months, 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 data structure, called a **data cube**, in which 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*

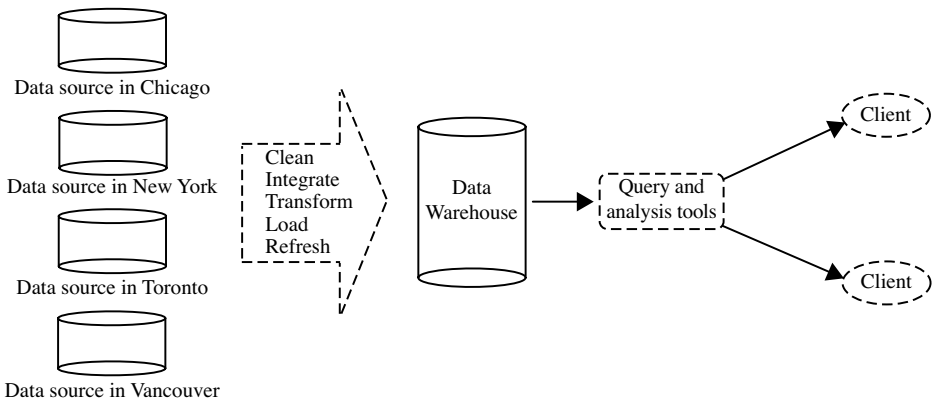


Figure 1.6 Typical framework of a data warehouse for *AllElectronics*.

or *sum*(*sales_amount*). A data cube provides a multidimensional view of data and allows the precomputation and fast access of summarized data.

Example 1.3 **A data cube for *AllElectronics*.** A data cube for summarized sales data of *AllElectronics* is presented in Figure 1.7(a). The cube has three dimensions: *address* (with city values *Chicago, New York, Toronto, Vancouver*), *time* (with quarter values *Q1, Q2, Q3, Q4*), and *item* (with item type values *home entertainment, computer, phone, security*). The aggregate value stored in each cell of the cube is *sales_amount* (in thousands). For example, the total sales for the first quarter, *Q1*, for the items related to security systems in Vancouver is $400,000, as stored in cell ⟨*Vancouver, Q1, security*⟩. Additional cubes may be used to store aggregate sums over each dimension, corresponding to the aggregate values obtained using different SQL group-bys (e.g., the total sales amount per city and quarter, or per city and item, or per quarter and item, or per each individual dimension). ■

By providing multidimensional data views and the precomputation of summarized data, data warehouse systems can provide inherent support for OLAP. Online analytical processing operations make use of background knowledge regarding the domain of the data being studied to allow the presentation of data at *different levels of abstraction*. Such operations accommodate different user viewpoints. Examples of OLAP operations include **drill-down** and **roll-up**, which allow the user to view the data at differing degrees of summarization, as illustrated in Figure 1.7(b). For instance, we can drill down on sales data summarized by *quarter* to see data summarized by *month*. Similarly, we can roll up on sales data summarized by *city* to view data summarized by *country*.

Although data warehouse tools help support data analysis, additional tools for data mining are often needed for in-depth analysis. **Multidimensional data mining** (also called **exploratory multidimensional data mining**) performs data mining in

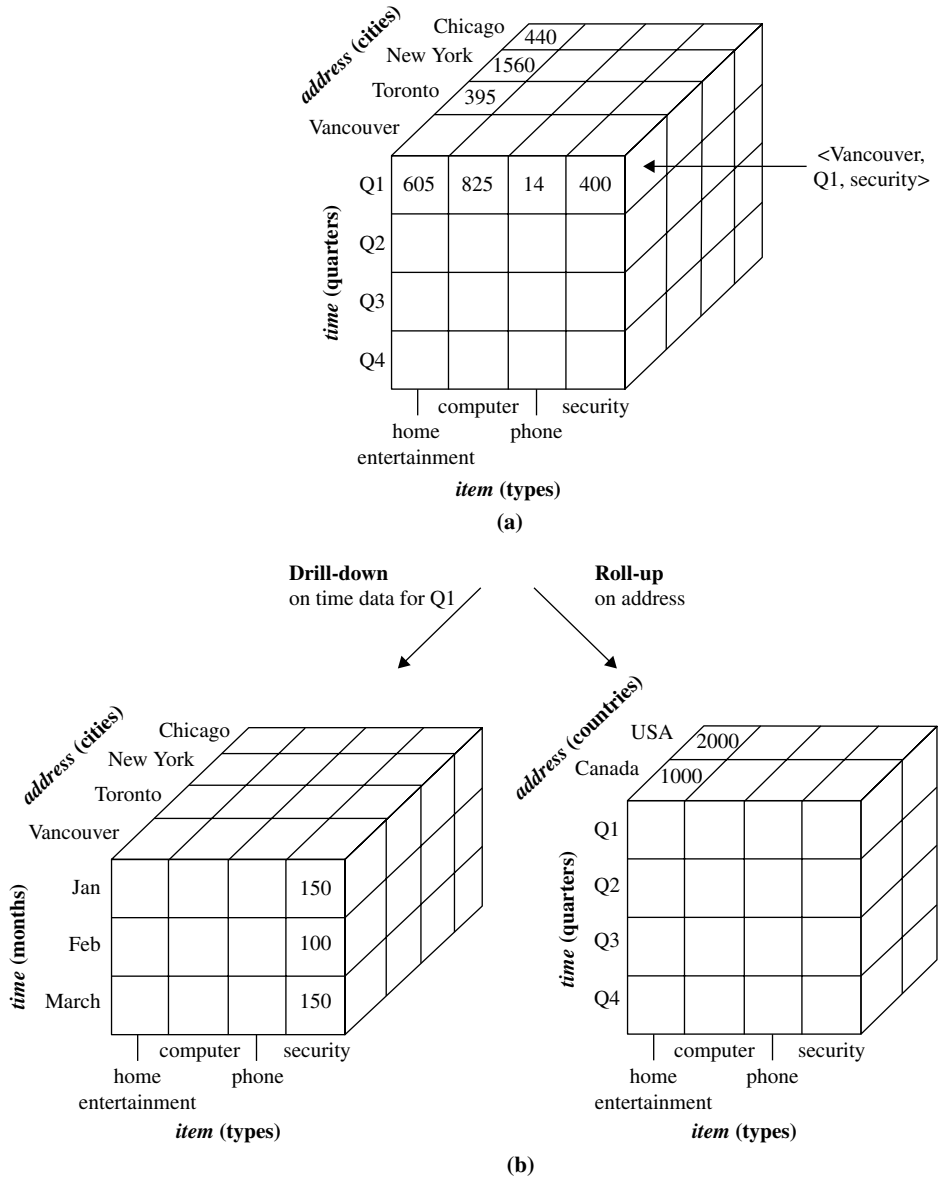


Figure 1.7 A multidimensional data cube, commonly used for data warehousing, (a) showing summarized data for *AllElectronics* and (b) showing summarized data resulting from drill-down and roll-up operations on the cube in (a). For improved readability, only some of the cube cell values are shown.

multidimensional space in an OLAP style. That is, it allows the exploration of multiple combinations of dimensions at varying levels of granularity in data mining, and thus has greater potential for discovering interesting patterns representing knowledge. An overview of data warehouse and OLAP technology is provided in Chapter 4. Advanced issues regarding data cube computation and multidimensional data mining are discussed in Chapter 5.

1.3.3 Transactional Data

In general, each record in a **transactional database** captures a transaction, such as a customer's purchase, a flight booking, or a user's clicks on a web page. A transaction typically includes a unique transaction identity number (*trans_ID*) and a list of the **items** making up the transaction, such as the items purchased in the transaction. A transactional database may have additional tables, which contain other information related to the transactions, such as item description, information about the salesperson or the branch, and so on.

Example 1.4 **A transactional database for *AllElectronics*.** Transactions can be stored in a table, with one record per transaction. A fragment of a transactional database for *AllElectronics* is shown in Figure 1.8. From the relational database point of view, the *sales* table in the figure is a nested relation because the attribute *list_of_item_IDs* contains a set of *items*. Because most relational database systems do not support nested relational structures, the transactional database is usually either stored in a flat file in a format similar to the table in Figure 1.8 or unfolded into a standard relation in a format similar to the *items_sold* table in Figure 1.5. ■

As an analyst of *AllElectronics*, you may ask, "*Which items sold well together?*" This kind of *market basket data analysis* would enable you to bundle groups of items together as a strategy for boosting sales. For example, given the knowledge that printers are commonly purchased together with computers, you could offer certain printers at a steep discount (or even for free) to customers buying selected computers, in the hopes of selling more computers (which are often more expensive than printers). A traditional database system is not able to perform market basket data analysis. Fortunately, data mining on transactional data can do so by mining *frequent itemsets*, that is, sets

trans_ID	*list_of_item_IDs*
T100	I1, I3, I8, I16
T200	I2, I8
...	...

Figure 1.8 Fragment of a transactional database for sales at *AllElectronics*.

of items that are frequently sold together. The mining of such frequent patterns from transactional data is discussed in Chapters 6 and 7.

1.3.4 Other Kinds of Data

Besides relational database data, data warehouse data, and transaction data, there are many other kinds of data that have versatile forms and structures and rather different semantic meanings. Such kinds of data can be seen in many applications: time-related or sequence data (e.g., historical records, stock exchange data, and time-series and biological sequence data), data streams (e.g., video surveillance and sensor data, which are continuously transmitted), spatial data (e.g., maps), engineering design data (e.g., the design of buildings, system components, or integrated circuits), hypertext and multimedia data (including text, image, video, and audio data), graph and networked data (e.g., social and information networks), and the Web (a huge, widely distributed information repository made available by the Internet). These applications bring about new challenges, like how to handle data carrying special structures (e.g., sequences, trees, graphs, and networks) and specific semantics (such as ordering, image, audio and video contents, and connectivity), and how to mine patterns that carry rich structures and semantics.

Various kinds of knowledge can be mined from these kinds of data. Here, we list just a few. Regarding temporal data, for instance, we can mine banking data for changing trends, which may aid in the scheduling of bank tellers according to the volume of customer traffic. Stock exchange data can be mined to uncover trends that could help you plan investment strategies (e.g., the best time to purchase *AllElectronics* stock). We could mine computer network data streams to detect intrusions based on the anomaly of message flows, which may be discovered by clustering, dynamic construction of stream models or by comparing the current frequent patterns with those at a previous time. With spatial data, we may look for patterns that describe changes in metropolitan poverty rates based on city distances from major highways. The relationships among a set of spatial objects can be examined in order to discover which subsets of objects are spatially autocorrelated or associated. By mining text data, such as literature on data mining from the past ten years, we can identify the evolution of hot topics in the field. By mining user comments on products (which are often submitted as short text messages), we can assess customer sentiments and understand how well a product is embraced by a market. From multimedia data, we can mine images to identify objects and classify them by assigning semantic labels or tags. By mining video data of a hockey game, we can detect video sequences corresponding to goals. *Web mining* can help us learn about the distribution of information on the WWW in general, characterize and classify web pages, and uncover web dynamics and the association and other relationships among different web pages, users, communities, and web-based activities.

It is important to keep in mind that, in many applications, multiple types of data are present. For example, in web mining, there often exist text data and multimedia data (e.g., pictures and videos) on web pages, graph data like web graphs, and map data on some web sites. In bioinformatics, genomic sequences, biological networks, and

3-D spatial structures of genomes may coexist for certain biological objects. Mining multiple data sources of complex data often leads to fruitful findings due to the mutual enhancement and consolidation of such multiple sources. On the other hand, it is also challenging because of the difficulties in data cleaning and data integration, as well as the complex interactions among the multiple sources of such data.

While such data require sophisticated facilities for efficient storage, retrieval, and updating, they also provide fertile ground and raise challenging research and implementation issues for data mining. Data mining on such data is an advanced topic. The methods involved are extensions of the basic techniques presented in this book.

1.4 What Kinds of Patterns Can Be Mined?

We have observed various types of data and information repositories on which data mining can be performed. Let us now examine the kinds of patterns that can be mined.

There are a number of ***data mining functionalities***. These include characterization and discrimination (Section 1.4.1); the mining of frequent patterns, associations, and correlations (Section 1.4.2); classification and regression (Section 1.4.3); clustering analysis (Section 1.4.4); and outlier analysis (Section 1.4.5). Data mining functionalities are used to specify the kinds of patterns to be found in data mining tasks. In general, such tasks can be classified into two categories: **descriptive** and **predictive**. Descriptive mining tasks characterize properties of the data in a target data set. Predictive mining tasks perform induction on the current data in order to make predictions.

Data mining functionalities, and the kinds of patterns they can discover, are described below. In addition, Section 1.4.6 looks at what makes a pattern interesting. Interesting patterns represent *knowledge*.

1.4.1 Class/Concept Description: Characterization and Discrimination

Data entries can be associated with classes or concepts. For example, in the *AllElectronics* store, classes of items for sale include *computers* and *printers*, and concepts of customers include *bigSpenders* and *budgetSpenders*. It can be useful to describe individual classes and concepts in summarized, concise, and yet precise terms. Such descriptions of a class or a concept are called **class/concept descriptions**. These descriptions can be derived using (1) *data characterization*, by summarizing the data of the class under study (often called the **target class**) in general terms, or (2) *data discrimination*, by comparison of the target class with one or a set of comparative classes (often called the **contrasting classes**), or (3) both data characterization and discrimination.

Data characterization is a summarization of the general characteristics or features of a target class of data. The data corresponding to the user-specified class are typically collected by a query. For example, to study the characteristics of software products with sales that increased by 10% in the previous year, the data related to such products can be collected by executing an SQL query on the sales database.

There are several methods for effective data summarization and characterization. Simple data summaries based on statistical measures and plots are described in Chapter 2. The data cube-based OLAP roll-up operation (Section 1.3.2) can be used to perform user-controlled data summarization along a specified dimension. This process is further detailed in Chapters 4 and 5, which discuss data warehousing. An *attribute-oriented induction* technique can be used to perform data generalization and characterization without step-by-step user interaction. This technique is also described in Chapter 4.

The output of data characterization can be presented in various forms. Examples include **pie charts**, **bar charts**, **curves**, **multidimensional data cubes**, and **multidimensional tables**, including crosstabs. The resulting descriptions can also be presented as **generalized relations** or in rule form (called **characteristic rules**).

Example 1.5 **Data characterization.** A customer relationship manager at *AllElectronics* may order the following data mining task: *Summarize the characteristics of customers who spend more than $5000 a year at AllElectronics.* The result is a general profile of these customers, such as that they are 40 to 50 years old, employed, and have excellent credit ratings. The data mining system should allow the customer relationship manager to drill down on any dimension, such as on *occupation* to view these customers according to their type of employment. ■

Data discrimination is a comparison of the general features of the target class data objects against the general features of objects from one or multiple contrasting classes. The target and contrasting classes can be specified by a user, and the corresponding data objects can be retrieved through database queries. For example, a user may want to compare the general features of software products with sales that increased by 10% last year against those with sales that decreased by at least 30% during the same period. The methods used for data discrimination are similar to those used for data characterization.

"*How are discrimination descriptions output?*" The forms of output presentation are similar to those for characteristic descriptions, although discrimination descriptions should include comparative measures that help to distinguish between the target and contrasting classes. Discrimination descriptions expressed in the form of rules are referred to as **discriminant rules**.

Example 1.6 **Data discrimination.** A customer relationship manager at *AllElectronics* may want to compare two groups of customers—those who shop for computer products regularly (e.g., more than twice a month) and those who rarely shop for such products (e.g., less than three times a year). The resulting description provides a general comparative profile of these customers, such as that 80% of the customers who frequently purchase computer products are between 20 and 40 years old and have a university education, whereas 60% of the customers who infrequently buy such products are either seniors or youths, and have no university degree. Drilling down on a dimension like *occupation*, or adding a new dimension like *income_level*, may help to find even more discriminative features between the two classes. ■

Concept description, including characterization and discrimination, is described in Chapter 4.

1.4.2 Mining Frequent Patterns, Associations, and Correlations

Frequent patterns, as the name suggests, are patterns that occur frequently in data. There are many kinds of frequent patterns, including frequent itemsets, frequent subsequences (also known as sequential patterns), and frequent substructures. A *frequent itemset* typically refers to a set of items that often appear together in a transactional data set—for example, milk and bread, which are frequently bought together in grocery stores by many customers. A frequently occurring subsequence, such as the pattern that customers, tend to purchase first a laptop, followed by a digital camera, and then a memory card, is a (*frequent*) *sequential pattern*. A substructure can refer to different structural forms (e.g., graphs, trees, or lattices) that may be combined with itemsets or subsequences. If a substructure occurs frequently, it is called a (*frequent*) *structured pattern*. Mining frequent patterns leads to the discovery of interesting associations and correlations within data.

Example 1.7 **Association analysis.** Suppose that, as a marketing manager at *AllElectronics*, you want to know which items are frequently purchased together (i.e., within the same transaction). An example of such a rule, mined from the *AllElectronics* transactional database, is

$$buys(X, \text{“}computer\text{”}) \Rightarrow buys(X, \text{“}software\text{”}) \; [support = 1\%, confidence = 50\%],$$

where X is a variable representing a customer. A **confidence**, or certainty, of 50% means that if a customer buys a computer, there is a 50% chance that she will buy software as well. A 1% **support** means that 1% of all the transactions under analysis show that computer and software are purchased together. This association rule involves a single attribute or predicate (i.e., *buys*) that repeats. Association rules that contain a single predicate are referred to as **single-dimensional association rules**. Dropping the predicate notation, the rule can be written simply as “*computer* $\Rightarrow$ *software* [1%, 50%].”

Suppose, instead, that we are given the *AllElectronics* relational database related to purchases. A data mining system may find association rules like

$$age(X, \text{“20..29”}) \wedge income(X, \text{“40K..49K”}) \Rightarrow buys(X, \text{“laptop”})$$
$$[support = 2\%, confidence = 60\%].$$

The rule indicates that of the *AllElectronics* customers under study, 2% are 20 to 29 years old with an income of $40,000 to $49,000 and have purchased a laptop (computer) at *AllElectronics*. There is a 60% probability that a customer in this age and income group will purchase a laptop. Note that this is an association involving more than one attribute or predicate (i.e., *age*, *income*, and *buys*). Adopting the terminology used in multidimensional databases, where each attribute is referred to as a dimension, the above rule can be referred to as a **multidimensional association rule**. ■

Typically, association rules are discarded as uninteresting if they do not satisfy both a **minimum support threshold** and a **minimum confidence threshold**. Additional analysis can be performed to uncover interesting statistical **correlations** between associated attribute–value pairs.

Frequent itemset mining is a fundamental form of frequent pattern mining. The mining of frequent patterns, associations, and correlations is discussed in Chapters 6 and 7, where particular emphasis is placed on efficient algorithms for frequent itemset mining. Sequential pattern mining and structured pattern mining are considered advanced topics.

1.4.3 Classification and Regression for Predictive Analysis

Classification is the process of finding a **model** (or function) that describes and distinguishes data classes or concepts. The model are derived based on the analysis of a set of **training data** (i.e., data objects for which the class labels are known). The model is used to predict the class label of objects for which the the class label is unknown.

"*How is the derived model presented?*" The derived model may be represented in various forms, such as *classification rules* (i.e., *IF-THEN rules*), *decision trees*, *mathematical formulae*, or *neural networks* (Figure 1.9). A **decision tree** is a flowchart-like tree structure, where each node denotes a test on an attribute value, each branch represents an outcome of the test, and tree leaves represent classes or class distributions. Decision trees can easily

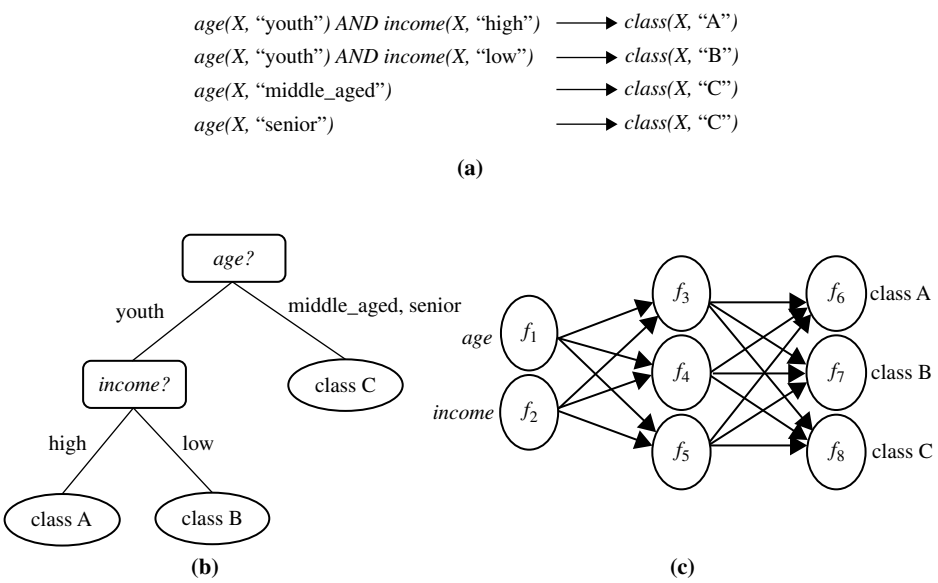


Figure 1.9 A classification model can be represented in various forms: (a) IF-THEN rules, (b) a decision tree, or (c) a neural network.

be converted to classification rules. A **neural network**, when used for classification, is typically a collection of neuron-like processing units with weighted connections between the units. There are many other methods for constructing classification models, such as naïve Bayesian classification, support vector machines, and *k*-nearest-neighbor classification.

Whereas classification predicts categorical (discrete, unordered) labels, **regression** models continuous-valued functions. That is, regression is used to predict missing or unavailable *numerical data values* rather than (discrete) class labels. The term *prediction* refers to both numeric prediction and class label prediction. **Regression analysis** is a statistical methodology that is most often used for numeric prediction, although other methods exist as well. Regression also encompasses the identification of distribution *trends* based on the available data.

Classification and regression may need to be preceded by **relevance analysis**, which attempts to identify attributes that are significantly relevant to the classification and regression process. Such attributes will be selected for the classification and regression process. Other attributes, which are irrelevant, can then be excluded from consideration.

Example 1.8 **Classification and regression.** Suppose as a sales manager of *AllElectronics* you want to classify a large set of items in the store, based on three kinds of responses to a sales campaign: *good response*, *mild response* and *no response*. You want to derive a model for each of these three classes based on the descriptive features of the items, such as *price*, *brand*, *place_made*, *type*, and *category*. The resulting classification should maximally distinguish each class from the others, presenting an organized picture of the data set.

Suppose that the resulting classification is expressed as a decision tree. The decision tree, for instance, may identify *price* as being the single factor that best distinguishes the three classes. The tree may reveal that, in addition to *price*, other features that help to further distinguish objects of each class from one another include *brand* and *place_made*. Such a decision tree may help you understand the impact of the given sales campaign and design a more effective campaign in the future.

Suppose instead, that rather than predicting categorical response labels for each store item, you would like to predict the amount of revenue that each item will generate during an upcoming sale at *AllElectronics*, based on the previous sales data. This is an example of regression analysis because the regression model constructed will predict a continuous function (or ordered value.) ■

Chapters 8 and 9 discuss classification in further detail. Regression analysis is beyond the scope of this book. Sources for further information are given in the bibliographic notes.

1.4.4 Cluster Analysis

Unlike classification and regression, which analyze class-labeled (training) data sets, **clustering** analyzes data objects without consulting class labels. In many cases, class-labeled data may simply not exist at the beginning. Clustering can be used to generate

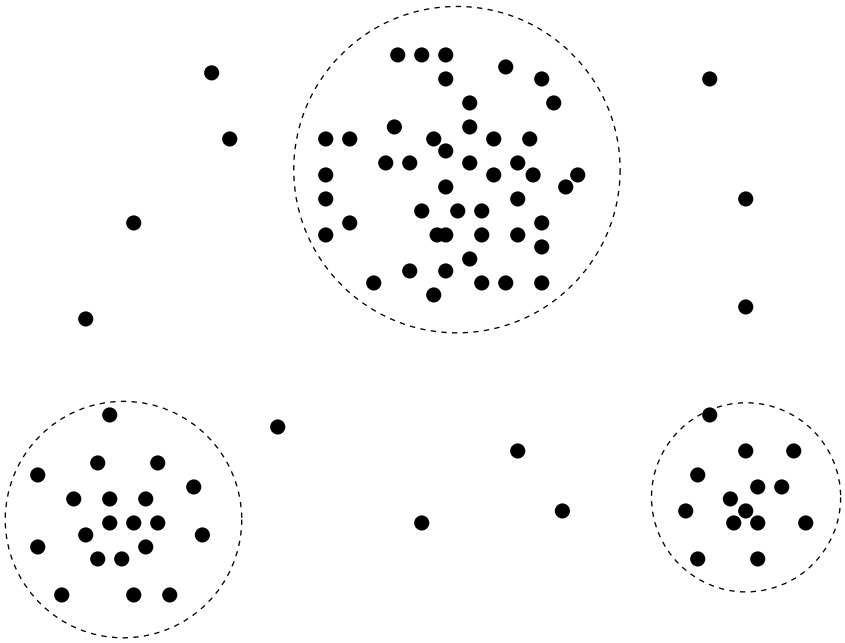


Figure 1.10 A 2-D plot of customer data with respect to customer locations in a city, showing three data clusters.

class labels for a group of data. The objects are clustered or grouped based on the principle of *maximizing the intraclass similarity and minimizing the interclass similarity*. That is, clusters of objects are formed so that objects within a cluster have high similarity in comparison to one another, but are rather dissimilar to objects in other clusters. Each cluster so formed can be viewed as a class of objects, from which rules can be derived. Clustering can also facilitate **taxonomy formation**, that is, the organization of observations into a hierarchy of classes that group similar events together.

Example 1.9 **Cluster analysis.** Cluster analysis can be performed on *AllElectronics* customer data to identify homogeneous subpopulations of customers. These clusters may represent individual target groups for marketing. Figure 1.10 shows a 2-D plot of customers with respect to customer locations in a city. Three clusters of data points are evident. ■

Cluster analysis forms the topic of Chapters 10 and 11.

1.4.5 Outlier Analysis

A data set may contain objects that do not comply with the general behavior or model of the data. These data objects are **outliers**. Many data mining methods discard outliers as noise or exceptions. However, in some applications (e.g., fraud detection) the rare

events can be more interesting than the more regularly occurring ones. The analysis of outlier data is referred to as **outlier analysis** or **anomaly mining**.

Outliers may be detected using statistical tests that assume a distribution or probability model for the data, or using distance measures where objects that are remote from any other cluster are considered outliers. Rather than using statistical or distance measures, density-based methods may identify outliers in a local region, although they look normal from a global statistical distribution view.

Example 1.10 **Outlier analysis.** Outlier analysis may uncover fraudulent usage of credit cards by detecting purchases of unusually large amounts for a given account number in comparison to regular charges incurred by the same account. Outlier values may also be detected with respect to the locations and types of purchase, or the purchase frequency. ■

Outlier analysis is discussed in Chapter 12.

1.4.6 Are All Patterns Interesting?

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

You may ask, *"Are all of the patterns interesting?"* Typically, the answer is no—only a small fraction of the patterns potentially generated would actually be of interest to a 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? Or, Can the system generate only the interesting ones?"*

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.

Other objective interestingness measures include *accuracy* and *coverage* for classification (IF-THEN) rules. In general terms, accuracy tells us the percentage of data that are correctly classified by a rule. Coverage is similar to support, in that it tells us the percentage of data to which a rule applies. Regarding understandability, we may use simple objective measures that assess the complexity or length in bits of the patterns mined.

Although objective measures help identify interesting patterns, they are often insufficient unless combined with subjective measures that reflect a particular user's needs and interests. For example, patterns describing the characteristics of customers who shop frequently at *AllElectronics* should be interesting to the marketing manager, but may be of little interest to other analysts studying the same database for patterns on employee performance. Furthermore, many patterns that are interesting by objective standards may represent common sense and, therefore, are actually uninteresting.

Subjective interestingness measures are based on user beliefs in the data. These measures find patterns interesting if the patterns 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**. For example, patterns like "a large earthquake often follows a cluster of small quakes" may be highly actionable if users can act on the information to save lives. Patterns that are **expected** can be interesting if they confirm a hypothesis that the user wishes to validate or they 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 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. The methods involved are examined in detail in Chapter 6.

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 efficient for users and data mining systems because neither would have to search through the patterns generated 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 by target users. Such measures can be used after the data mining step to rank the discovered patterns according to their interestingness, filtering out the uninteresting ones. More important, 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. Examples of such a constraint-based mining process are described in Chapter 7 (with respect to pattern discovery) and Chapter 11 (with respect to clustering).

Methods to assess pattern interestingness, and their use to improve data mining efficiency, are discussed throughout the book with respect to each kind of pattern that can be mined.

1.5 Which Technologies Are Used?

As a highly application-driven domain, data mining has incorporated many techniques from other domains such as statistics, machine learning, pattern recognition, database and data warehouse systems, information retrieval, visualization, algorithms, high-performance computing, and many application domains (Figure 1.11). The interdisciplinary nature of data mining research and development contributes significantly to the success of data mining and its extensive applications. In this section, we give examples of several disciplines that strongly influence the development of data mining methods.

1.5.1 Statistics

Statistics studies the collection, analysis, interpretation or explanation, and presentation of data. Data mining has an inherent connection with 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

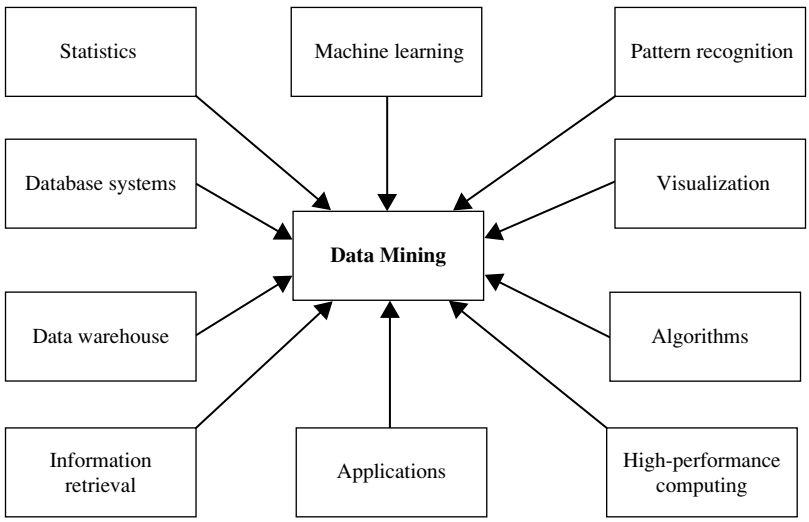


Figure 1.11 Data mining adopts techniques from many domains.

models of target classes can be built. In other words, such statistical models can be the outcome of a data mining task. Alternatively, data mining tasks can be built on top of statistical models. For example, we can use statistics to model noise and missing data values. Then, when mining patterns in a large data set, the data mining process can use the model to help identify and handle noisy or missing values in the data.

Statistics research develops tools for prediction and forecasting using data and statistical models. Statistical methods can be used to summarize or describe a collection of data. Basic **statistical descriptions** of data are introduced in Chapter 2. Statistics is useful for mining various patterns from data as well as for understanding the underlying mechanisms generating and affecting the patterns. **Inferential statistics** (or **predictive statistics**) models data in a way that accounts for randomness and uncertainty in the observations and is used to draw inferences about the process or population under investigation.

Statistical methods can also be used to verify data mining results. For example, after a classification or prediction model is mined, the model should be verified by statistical hypothesis testing. A **statistical hypothesis test** (sometimes called *confirmatory data analysis*) makes statistical decisions using experimental data. A result is called *statistically significant* if it is unlikely to have occurred by chance. If the classification or prediction model holds true, then the descriptive statistics of the model increases the soundness of the model.

Applying statistical methods in data mining is far from trivial. Often, a serious challenge is how to scale up a statistical method over a large data set. Many statistical methods have high complexity in computation. When such methods are applied on large data sets that are also distributed on multiple logical or physical sites, algorithms should be carefully designed and tuned to reduce the computational cost. This challenge becomes even tougher for online applications, such as online query suggestions in search engines, where data mining is required to continuously handle fast, real-time data streams.

1.5.2 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. For example, a typical machine learning problem is to program a computer so that it can automatically recognize handwritten postal codes on mail after learning from a set of examples.

Machine learning is a fast-growing discipline. Here, we illustrate classic problems in machine learning that are highly related to data mining.

- **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.

- **Unsupervised learning** is essentially a synonym for clustering. The learning process is unsupervised since the input examples are not class labeled. Typically, we may use clustering to discover classes within the data. 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. However, since the training data are not labeled, the learned model cannot tell us the semantic meaning of the clusters found.
- **Semi-supervised learning** is a class of machine learning techniques that make use of both labeled and unlabeled examples when learning a model. In one approach, labeled examples are used to learn class models and unlabeled examples are used to refine the boundaries between classes. For a two-class problem, we can think of the set of examples belonging to one class as the *positive examples* and those belonging to the other class as the *negative examples*. In Figure 1.12, if we do not consider the unlabeled examples, the dashed line is the decision boundary that best partitions the positive examples from the negative examples. Using the unlabeled examples, we can refine the decision boundary to the solid line. Moreover, we can detect that the two positive examples at the top right corner, though labeled, are likely noise or outliers.
- **Active learning** is a machine learning approach that lets users play an active role in the learning process. An active learning approach can ask a user (e.g., a domain expert) to label an example, which may be from a set of unlabeled examples or synthesized by the learning program. The goal is to optimize the model quality by actively acquiring knowledge from human users, given a constraint on how many examples they can be asked to label.

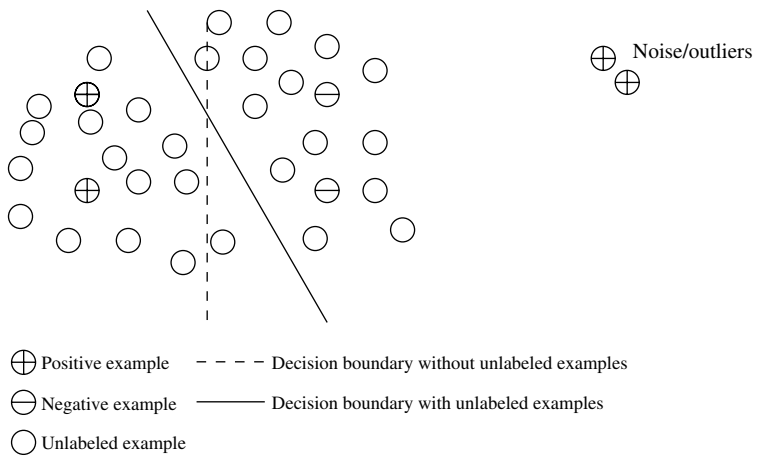


Figure 1.12 Semi-supervised learning.

You can see there are many similarities between data mining and machine learning. For classification and clustering tasks, machine learning research often focuses on the accuracy of the model. In addition to accuracy, data mining research places strong emphasis on the efficiency and scalability of mining methods on large data sets, as well as on ways to handle complex types of data and explore new, alternative methods.

1.5.3 Database Systems and Data Warehouses

Database systems research focuses on the creation, maintenance, and use of databases for organizations and end-users. Particularly, database systems researchers have established highly recognized principles in data models, query languages, query processing and optimization methods, data storage, and indexing and accessing methods. Database systems are often well known for their high scalability in processing very large, relatively structured data sets.

Many data mining tasks need to handle large data sets or even real-time, fast streaming data. Therefore, data mining can make good use of scalable database technologies to achieve high efficiency and scalability on large data sets. Moreover, data mining tasks can be used to extend the capability of existing database systems to satisfy advanced users' sophisticated data analysis requirements.

Recent database systems have built systematic data analysis capabilities on database data using data warehousing and data mining facilities. A **data warehouse** integrates data originating from multiple sources and various timeframes. It consolidates data in multidimensional space to form partially materialized data cubes. The data cube model not only facilitates OLAP in multidimensional databases but also promotes *multidimensional data mining* (see Section 1.3.2).

1.5.4 Information Retrieval

Information retrieval (**IR**) is the science of searching for documents or information in documents. Documents can be text or multimedia, and may reside on the Web. The differences between traditional information retrieval and database systems are twofold: Information retrieval assumes that (1) the data under search are unstructured; and (2) the queries are formed mainly by keywords, which do not have complex structures (unlike SQL queries in database systems).

The typical approaches in information retrieval adopt probabilistic models. For example, a text document can be regarded as a bag of words, that is, a multiset of words appearing in the document. The document's **language model** is the probability density function that generates the bag of words in the document. The similarity between two documents can be measured by the similarity between their corresponding language models.

Furthermore, a topic in a set of text documents can be modeled as a probability distribution over the vocabulary, which is called a **topic model**. A text document, which may involve one or multiple topics, can be regarded as a mixture of multiple topic models. By integrating information retrieval models and data mining techniques, we can find

the major topics in a collection of documents and, for each document in the collection, the major topics involved.

Increasingly large amounts of text and multimedia data have been accumulated and made available online due to the fast growth of the Web and applications such as digital libraries, digital governments, and health care information systems. Their effective search and analysis have raised many challenging issues in data mining. Therefore, text mining and multimedia data mining, integrated with information retrieval methods, have become increasingly important.

1.6 Which Kinds of Applications Are Targeted?

Where there are data, there are data mining applications

As a highly application-driven discipline, data mining has seen great successes in many applications. It is impossible to enumerate all applications where data mining plays a critical role. Presentations of data mining in knowledge-intensive application domains, such as bioinformatics and software engineering, require more in-depth treatment and are beyond the scope of this book. To demonstrate the importance of applications as a major dimension in data mining research and development, we briefly discuss two highly successful and popular application examples of data mining: *business intelligence* and *search engines*.

1.6.1 Business Intelligence

It is critical for businesses to acquire a better understanding of the commercial context of their organization, such as their customers, the market, supply and resources, and competitors. **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.

"*How important is business intelligence?*" Without data mining, many businesses may not be able to perform effective market analysis, compare customer feedback on similar products, discover the strengths and weaknesses of their competitors, retain highly valuable customers, and make smart business decisions.

Clearly, data mining is the core of business intelligence. Online analytical processing tools in business intelligence rely 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. Moreover, clustering plays a central role in customer relationship management, which groups customers based on their similarities. Using characterization mining techniques, we can better understand features of each customer group and develop customized customer reward programs.

1.6.2 Web Search Engines

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. Some search engines also search and return data available in public databases or open directories. Search engines differ from **web directories** in that web directories are maintained by human editors whereas search engines operate algorithmically or by a mixture of algorithmic and human input.

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*[5] (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").

Search engines pose grand challenges to data mining. First, they have to handle a huge and ever-growing amount of data. Typically, such data cannot be processed using one or a few machines. Instead, search engines often need to use *computer clouds*, which consist of thousands or even hundreds of thousands of computers that collaboratively mine the huge amount of data. Scaling up data mining methods over computer clouds and large distributed data sets is an area for further research.

Second, Web search engines often have to deal with online data. A search engine may be able to afford constructing a model offline on huge data sets. To do this, it may construct a query classifier that assigns a search query to predefined categories based on the query topic (i.e., whether the search query "apple" is meant to retrieve information about a fruit or a brand of computers). Whether a model is constructed offline, the application of the model online must be fast enough to answer user queries in real time.

Another challenge is maintaining and incrementally updating a model on fast-growing data streams. For example, a query classifier may need to be incrementally maintained continuously since new queries keep emerging and predefined categories and the data distribution may change. Most of the existing model training methods are offline and static and thus cannot be used in such a scenario.

Third, Web search engines often have to deal with queries that are asked only a very small number of times. Suppose a search engine wants to provide *context-aware* query recommendations. That is, when a user poses a query, the search engine tries to infer the context of the query using the user's profile and his query history in order to return more customized answers within a small fraction of a second. However, although the total number of queries asked can be huge, most of the queries may be asked only once or a few times. Such severely skewed data are challenging for many data mining and machine learning methods.

[5] A Web crawler is a computer program that browses the Web in a methodical, automated manner.

1.7 Major Issues in Data Mining

Life is short but art is long. – Hippocrates

Data mining is a dynamic and fast-expanding field with great strengths. In this section, we briefly outline the 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*. Many of these issues have been addressed in recent data mining research and development *to a certain extent* and are now considered *data mining requirements*; others are still at the research stage. The issues continue to stimulate further investigation and improvement in data mining.

1.7.1 Mining Methodology

Researchers have been vigorously developing new data mining methodologies. This involves the investigation of new kinds of knowledge, mining in multidimensional space, integrating methods from other disciplines, and the consideration of semantic ties among data objects. In addition, mining methodologies should consider issues such as data uncertainty, noise, and incompleteness. Some mining methods explore how user-specified measures can be used to assess the interestingness of discovered patterns as well as guide the discovery process. Let's have a look at these various aspects of mining methodology.

- *Mining various and new kinds of knowledge:* Data mining covers a wide spectrum of data analysis and knowledge discovery tasks, from data characterization and discrimination to association and correlation analysis, classification, regression, clustering, outlier analysis, sequence analysis, and trend and evolution analysis. These tasks may use the same database in different ways and require the development of numerous data mining techniques. Due to the diversity of applications, new mining tasks continue to emerge, making data mining a dynamic and fast-growing field. For example, for effective knowledge discovery in information networks, integrated clustering and ranking may lead to the discovery of high-quality clusters and object ranks in large networks.
- *Mining knowledge in multidimensional space:* When searching for knowledge in large data sets, we can explore the data in multidimensional space. That is, we can search for interesting patterns among combinations of dimensions (attributes) at varying levels of abstraction. Such mining is known as *(exploratory) multidimensional data mining*. In many cases, data can be aggregated or viewed as a multidimensional data cube. Mining knowledge in cube space can substantially enhance the power and flexibility of data mining.
- *Data mining—an interdisciplinary effort:* The power of data mining can be substantially enhanced by integrating new methods from multiple disciplines. For example,

to mine data with natural language text, it makes sense to fuse data mining methods with methods of information retrieval and natural language processing. As another example, consider the mining of software bugs in large programs. This form of mining, known as *bug mining*, benefits from the incorporation of software engineering knowledge into the data mining process.

- *Boosting the power of discovery in a networked environment:* Most data objects reside in a linked or interconnected environment, whether it be the Web, database relations, files, or documents. Semantic links across multiple data objects can be used to advantage in data mining. Knowledge derived in one set of objects can be used to boost the discovery of knowledge in a "related" or semantically linked set of objects.
- *Handling uncertainty, noise, or incompleteness of data:* Data often contain noise, errors, exceptions, or uncertainty, or are incomplete. Errors and noise may confuse the data mining process, leading to the derivation of erroneous patterns. Data cleaning, data preprocessing, outlier detection and removal, and uncertainty reasoning are examples of techniques that need to be integrated with the data mining process.
- *Pattern evaluation and pattern- or constraint-guided mining:* Not all the patterns generated by data mining processes are interesting. What makes a pattern interesting may vary from user to user. Therefore, techniques are needed to assess the interestingness of discovered patterns based on subjective measures. These estimate the value of patterns with respect to a given user class, based on user beliefs or expectations. Moreover, by using interestingness measures or user-specified constraints to *guide* the discovery process, we may generate more interesting patterns and reduce the search space.

1.7.2 User Interaction

The user plays an important role in the data mining process. 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*. We introduce each of these here.

- *Interactive mining:* The data mining process should be highly *interactive*. Thus, it is important to build flexible user interfaces and an exploratory mining environment, facilitating the user's interaction with the system. A user may like to first sample a set of data, explore general characteristics of the data, and estimate potential mining results. Interactive mining should allow users to dynamically change the focus of a search, to refine mining requests based on returned results, and to drill, dice, and pivot through the data and knowledge space interactively, dynamically exploring "cube space" while mining.
- *Incorporation of background knowledge:* Background knowledge, constraints, rules, and other information regarding the domain under study should be incorporated

into the knowledge discovery process. Such knowledge can be used for pattern evaluation as well as to guide the search toward interesting patterns.

- *Ad hoc data mining and data mining query languages:* Query languages (e.g., SQL) have played an important role in flexible searching because they allow users to pose ad hoc queries. Similarly, high-level data mining query languages or other high-level flexible user interfaces will give users the freedom to define ad hoc data mining tasks. This should facilitate 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. Optimization of the processing of such flexible mining requests is another promising area of study.
- *Presentation and visualization of data mining results:* How can a data mining system present data mining results, vividly and flexibly, so that the discovered knowledge can be easily understood and directly usable by humans? This is especially crucial if the data mining process is interactive. It requires the system to adopt expressive knowledge representations, user-friendly interfaces, and visualization techniques.

1.7.3 Efficiency and Scalability

Efficiency and scalability are always considered when comparing data mining algorithms. As data amounts continue to multiply, these two factors are especially critical.

- *Efficiency and scalability of data mining algorithms:* Data mining algorithms must be efficient and scalable in order to effectively extract information from huge amounts of data in many data repositories or in dynamic data streams. In other words, the running time of a data mining algorithm must be predictable, short, and acceptable by applications. *Efficiency, scalability, performance, optimization*, and the ability to execute in *real time* are key criteria that drive the development of many new data mining algorithms.
- *Parallel, distributed, and incremental mining algorithms:* The humongous size of many data sets, the wide distribution of data, and the computational complexity of some data mining methods are factors that motivate the development of **parallel and distributed data-intensive mining algorithms**. Such algorithms first partition the data into "pieces." Each piece is processed, in parallel, by searching for patterns. The parallel processes may interact with one another. The patterns from each partition are eventually merged.

  Cloud computing and *cluster computing*, which use computers in a distributed and collaborative way to tackle very large-scale computational tasks, are also active research themes in parallel data mining. In addition, the high cost of some data mining processes and the incremental nature of input promote **incremental** data mining, which incorporates new data updates without having to mine the entire data "from scratch." Such methods perform knowledge modification incrementally to amend and strengthen what was previously discovered.

1.7.4 Diversity of Database Types

The wide diversity of database types brings about challenges to data mining. These include

- *Handling complex types of data:* Diverse applications generate a wide spectrum of new data types, from structured data such as relational and data warehouse data to semi-structured and unstructured data; from stable data repositories to dynamic data streams; from simple data objects to temporal data, biological sequences, sensor data, spatial data, hypertext data, multimedia data, software program code, Web data, and social network data. It is unrealistic to expect one data mining system to mine all kinds of data, given the diversity of data types and the different goals of data mining. Domain- or application-dedicated data mining systems are being constructed for in-depth mining of specific kinds of data. The construction of effective and efficient data mining tools for diverse applications remains a challenging and active area of research.
- *Mining dynamic, networked, and global data repositories:* Multiple sources of data are connected by the Internet and various kinds of networks, forming gigantic, distributed, and heterogeneous global information systems and networks. The discovery of knowledge from different sources of structured, semi-structured, or unstructured yet interconnected data with diverse data semantics poses great challenges to data mining. Mining such gigantic, interconnected information networks may help disclose many more patterns and knowledge in heterogeneous data sets than can be discovered from a small set of isolated data repositories. Web mining, multisource data mining, and information network mining have become challenging and fast-evolving data mining fields.

1.7.5 Data Mining and Society

How does data mining impact society? What steps can data mining take to preserve the privacy of individuals? Do we use data mining in our daily lives without even knowing that we do? These questions raise the following issues:

- *Social impacts of data mining:* With data mining penetrating our everyday lives, it is important to study the impact of data mining on society. How can we use data mining technology to benefit society? How can we guard against its misuse? The improper disclosure or use of data and the potential violation of individual privacy and data protection rights are areas of concern that need to be addressed.
- *Privacy-preserving data mining:* Data mining will help scientific discovery, business management, economy recovery, and security protection (e.g., the real-time discovery of intruders and cyberattacks). However, it poses the risk of disclosing an individual's personal information. Studies on privacy-preserving data publishing and data mining are ongoing. The philosophy is to observe data sensitivity and preserve people's privacy while performing successful data mining.

- *Invisible data mining:* We cannot expect everyone in society to learn and master data mining techniques. More and more systems should have data mining functions built within so that people can perform data mining or use data mining results simply by mouse clicking, without any knowledge of data mining algorithms. Intelligent search engines and Internet-based stores perform such *invisible data mining* by incorporating data mining into their components to improve their functionality and performance. This is done often unbeknownst to the user. For example, when purchasing items online, users may be unaware that the store is likely collecting data on the buying patterns of its customers, which may be used to recommend other items for purchase in the future.

These issues and many additional ones relating to the research, development, and application of data mining are discussed throughout the book.

1.8 Summary

- *Necessity is the mother of invention.* With the mounting growth of data in every application, data mining meets the imminent need for effective, scalable, and flexible data analysis in our society. Data mining can be considered as a natural evolution of information technology and a confluence of several related disciplines and application domains.
- **Data mining** is the process of discovering interesting patterns from massive amounts of data. As a *knowledge discovery process*, it typically involves data cleaning, data integration, data selection, data transformation, pattern discovery, pattern evaluation, and knowledge presentation.
- A pattern is *interesting* if it is valid on test data with some degree of certainty, novel, potentially useful (e.g., can be acted on or validates a hunch about which the user was curious), and easily understood by humans. Interesting patterns represent **knowledge**. Measures of **pattern interestingness**, either *objective* or *subjective*, can be used to guide the discovery process.
- We present a **multidimensional view** of data mining. The major dimensions are **data**, **knowledge**, **technologies**, and **applications**.
- Data mining can be conducted on any kind of **data** as long as the data are meaningful for a target application, such as database data, data warehouse data, transactional data, and advanced data types. Advanced data types include time-related or sequence data, data streams, spatial and spatiotemporal data, text and multimedia data, graph and networked data, and Web data.
- A **data warehouse** is a repository for long-term storage of data from multiple sources, organized so as to facilitate management decision making. The data are stored under a unified schema and are typically summarized. Data warehouse systems provide multidimensional data analysis capabilities, collectively referred to as **online analytical processing**.

- **Multidimensional data mining** (also called **exploratory multidimensional data mining**) integrates core data mining techniques with OLAP-based multidimensional analysis. It searches for interesting patterns among multiple combinations of dimensions (attributes) at varying levels of abstraction, thereby exploring multidimensional data space.
- **Data mining functionalities** are used to specify the kinds of patterns or **knowledge** to be found in data mining tasks. The functionalities include characterization and discrimination; the mining of frequent patterns, associations, and correlations; classification and regression; cluster analysis; and outlier detection. As new types of data, new applications, and new analysis demands continue to emerge, there is no doubt we will see more and more novel data mining tasks in the future.
- Data mining, as a highly application-driven domain, has incorporated **technologies** from many other domains. These include statistics, machine learning, database and data warehouse systems, and information retrieval. The **interdisciplinary nature of data mining research and development** contributes significantly to the success of data mining and its extensive applications.
- Data mining has many successful **applications**, such as business intelligence, Web search, bioinformatics, health informatics, finance, digital libraries, and digital governments.
- There are many challenging **issues in data mining research**. Areas include mining methodology, user interaction, efficiency and scalability, and dealing with diverse data types. Data mining research has strongly impacted society and will continue to do so in the future.

1.9 Exercises

1.1 What is *data mining*? In your answer, address the following:

(a) Is it another hype?

(b) Is it a simple transformation or application of technology developed from *databases*, *statistics*, *machine learning*, and *pattern recognition*?

(c) We have presented a view that data mining is the result of the evolution of *database technology*. Do you think that data mining is also the result of the evolution of *machine learning research*? Can you present such views based on the historical progress of this discipline? Address the same for the fields of *statistics* and *pattern recognition*.

(d) Describe the steps involved in data mining when viewed as a process of knowledge discovery.

1.2 How is a *data warehouse* different from a *database*? How are they similar?

1.3 Define each of the following *data mining functionalities*: characterization, discrimination, association and correlation analysis, classification, regression, clustering, and

outlier analysis. Give examples of each data mining functionality, using a real-life database that you are familiar with.

1.4 Present an example where data mining is crucial to the success of a business. What *data mining functionalities* does this business need (e.g., think of the kinds of patterns that could be mined)? Can such patterns be generated alternatively by data query processing or simple statistical analysis?

1.5 Explain the difference and similarity between discrimination and classification, between characterization and clustering, and between classification and regression.

1.6 Based on your observations, describe another possible kind of knowledge that needs to be discovered by data mining methods but has not been listed in this chapter. Does it require a mining methodology that is quite different from those outlined in this chapter?

1.7 *Outliers* are often discarded as noise. However, one person's garbage could be another's treasure. For example, exceptions in credit card transactions can help us detect the fraudulent use of credit cards. Using fraudulence detection as an example, propose two methods that can be used to detect outliers and discuss which one is more reliable.

1.8 Describe three challenges to data mining regarding *data mining methodology* and *user interaction issues.*

1.9 What are the major challenges of mining a huge amount of data (e.g., billions of tuples) in comparison with mining a small amount of data (e.g., data set of a few hundred tuple)?

1.10 Outline the major research challenges of data mining in one specific application domain, such as stream/sensor data analysis, spatiotemporal data analysis, or bioinformatics.

1.10 Bibliographic Notes

The book *Knowledge Discovery in Databases*, edited by Piatetsky-Shapiro and Frawley [P-SF91], is an early collection of research papers on knowledge discovery from data. The book *Advances in Knowledge Discovery and Data Mining*, edited by Fayyad, Piatetsky-Shapiro, Smyth, and Uthurusamy [FPSS+96], is a collection of later research results on knowledge discovery and data mining. There have been many data mining books published in recent years, including *The Elements of Statistical Learning* by Hastie, Tibshirani, and Friedman [HTF09]; *Introduction to Data Mining* by Tan, Steinbach, and Kumar [TSK05]; *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations* by Witten, Frank, and Hall [WFH11]; *Predictive Data Mining* by Weiss and Indurkhya [WI98]; *Mastering Data Mining: The Art and Science of Customer Relationship Management* by Berry and Linoff [BL99]; *Principles of Data Mining (Adaptive Computation and Machine Learning)* by Hand, Mannila, and Smyth [HMS01]; *Mining the Web: Discovering Knowledge from Hypertext Data* by Chakrabarti [Cha03a]; *Web Data Mining: Exploring Hyperlinks, Contents, and Usage*

Data by Liu [Liu06]; *Data Mining: Introductory and Advanced Topics* by Dunham [Dun03]; and *Data Mining: Multimedia, Soft Computing, and Bioinformatics* by Mitra and Acharya [MA03].

There are also books that contain collections of papers or chapters on particular aspects of knowledge discovery—for example, *Relational Data Mining* edited by Dzeroski and Lavrac [De01]; *Mining Graph Data* edited by Cook and Holder [CH07]; *Data Streams: Models and Algorithms* edited by Aggarwal [Agg06]; *Next Generation of Data Mining* edited by Kargupta, Han, Yu, et al. [KHY+08]; *Multimedia Data Mining: A Systematic Introduction to Concepts and Theory* edited by Z. Zhang and R. Zhang [ZZ09]; *Geographic Data Mining and Knowledge Discovery* edited by Miller and Han [MH09]; and *Link Mining: Models, Algorithms and Applications* edited by Yu, Han, and Faloutsos [YHF10]. There are many tutorial notes on data mining in major databases, data mining, machine learning, statistics, and Web technology conferences.

KDNuggets is a regular electronic newsletter containing information relevant to knowledge discovery and data mining, moderated by Piatetsky-Shapiro since 1991. The Internet site *KDNuggets* (*www.kdnuggets.com*) contains a good collection of KDD-related information.

The data mining community started its first international conference on knowledge discovery and data mining in 1995. The conference evolved from the four international workshops on knowledge discovery in databases, held from 1989 to 1994. ACM-SIGKDD, a Special Interest Group on Knowledge Discovery in Databases was set up under ACM in 1998 and has been organizing the international conferences on knowledge discovery and data mining since 1999. IEEE Computer Science Society has organized its annual data mining conference, International Conference on Data Mining (ICDM), since 2001. SIAM (Society on Industrial and Applied Mathematics) has organized its annual data mining conference, SIAM Data Mining Conference (SDM), since 2002. A dedicated journal, *Data Mining and Knowledge Discovery*, published by Kluwers Publishers, has been available since 1997. An ACM journal, *ACM Transactions on Knowledge Discovery from Data*, published its first volume in 2007.

ACM-SIGKDD also publishes a bi-annual newsletter, *SIGKDD Explorations*. There are a few other international or regional conferences on data mining, such as the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD), the Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), and the International Conference on Data Warehousing and Knowledge Discovery (DaWaK).

Research in data mining has also been published in books, conferences, and journals on databases, statistics, machine learning, and data visualization. References to such sources are listed at the end of the book.

Popular textbooks on database systems include *Database Systems: The Complete Book* by Garcia-Molina, Ullman, and Widom [GMUW08]; *Database Management Systems* by Ramakrishnan and Gehrke [RG03]; *Database System Concepts* by Silberschatz, Korth, and Sudarshan [SKS10]; and *Fundamentals of Database Systems* by Elmasri and Navathe [EN10]. For an edited collection of seminal articles on database systems, see *Readings in Database Systems* by Hellerstein and Stonebraker [HS05].

There are also many books on data warehouse technology, systems, and applications, such as *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling* by Kimball and Ross [KR02]; *The Data Warehouse Lifecycle Toolkit* by Kimball, Ross, Thornthwaite, and Mundy [KRTM08]; *Mastering Data Warehouse Design: Relational and Dimensional Techniques* by Imhoff, Galemmo, and Geiger [IGG03]; and *Building the Data Warehouse* by Inmon [Inm96]. A set of research papers on materialized views and data warehouse implementations were collected in *Materialized Views: Techniques, Implementations, and Applications* by Gupta and Mumick [GM99]. Chaudhuri and Dayal [CD97] present an early comprehensive overview of data warehouse technology.

Research results relating to data mining and data warehousing have been published in the proceedings of many international database conferences, including the ACM-SIGMOD International Conference on Management of Data (SIGMOD), the International Conference on Very Large Data Bases (VLDB), the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS), the International Conference on Data Engineering (ICDE), the International Conference on Extending Database Technology (EDBT), the International Conference on Database Theory (ICDT), the International Conference on Information and Knowledge Management (CIKM), the International Conference on Database and Expert Systems Applications (DEXA), and the International Symposium on Database Systems for Advanced Applications (DASFAA). Research in data mining is also published in major database journals, such as *IEEE Transactions on Knowledge and Data Engineering (TKDE), ACM Transactions on Database Systems (TODS), Information Systems, The VLDB Journal, Data and Knowledge Engineering, International Journal of Intelligent Information Systems (JIIS)*, and *Knowledge and Information Systems (KAIS)*.

Many effective data mining methods have been developed by statisticians and introduced in a rich set of textbooks. An overview of classification from a statistical pattern recognition perspective can be found in *Pattern Classification* by Duda, Hart, and Stork [DHS01]. There are also many textbooks covering regression and other topics in statistical analysis, such as *Mathematical Statistics: Basic Ideas and Selected Topics* by Bickel and Doksum [BD01]; *The Statistical Sleuth: A Course in Methods of Data Analysis* by Ramsey and Schafer [RS01]; *Applied Linear Statistical Models* by Neter, Kutner, Nachtsheim, and Wasserman [NKNW96]; *An Introduction to Generalized Linear Models* by Dobson [Dob90]; *Applied Statistical Time Series Analysis* by Shumway [Shu88]; and *Applied Multivariate Statistical Analysis* by Johnson and Wichern [JW92].

Research in statistics is published in the proceedings of several major statistical conferences, including Joint Statistical Meetings, International Conference of the Royal Statistical Society and Symposium on the Interface: Computing Science and Statistics. Other sources of publication include the *Journal of the Royal Statistical Society*, *The Annals of Statistics*, the *Journal of American Statistical Association*, *Technometrics*, and *Biometrika*.

Textbooks and reference books on machine learning and pattern recognition include *Machine Learning* by Mitchell [Mit97]; *Pattern Recognition and Machine Learning* by Bishop [Bis06]; *Pattern Recognition* by Theodoridis and Koutroumbas [TK08]; *Introduction to Machine Learning* by Alpaydin [Alp11]; *Probabilistic Graphical Models: Principles*

and Techniques by Koller and Friedman [KF09]; and *Machine Learning: An Algorithmic Perspective* by Marsland [Mar09]. For an edited collection of seminal articles on machine learning, see *Machine Learning, An Artificial Intelligence Approach*, Volumes 1 through 4, edited by Michalski et al. [MCM83, MCM86, KM90, MT94], and *Readings in Machine Learning* by Shavlik and Dietterich [SD90].

Machine learning and pattern recognition research is published in the proceedings of several major machine learning, artificial intelligence, and pattern recognition conferences, including the International Conference on Machine Learning (ML), the ACM Conference on Computational Learning Theory (COLT), the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), the International Conference on Pattern Recognition (ICPR), the International Joint Conference on Artificial Intelligence (IJCAI), and the American Association of Artificial Intelligence Conference (AAAI). Other sources of publication include major machine learning, artificial intelligence, pattern recognition, and knowledge system journals, some of which have been mentioned before. Others include *Machine Learning (ML), Pattern Recognition (PR), Artificial Intelligence Journal (AI)*, *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, and *Cognitive Science*.

Textbooks and reference books on information retrieval include *Introduction to Information Retrieval* by Manning, Raghavan, and Schutz [MRS08]; *Information Retrieval: Implementing and Evaluating Search Engines* by Büttcher, Clarke, and Cormack [BCC10]; *Search Engines: Information Retrieval in Practice* by Croft, Metzler, and Strohman [CMS09]; *Modern Information Retrieval: The Concepts and Technology Behind Search* by Baeza-Yates and Ribeiro-Neto [BYRN11]; and *Information Retrieval: Algorithms and Heuristics* by Grossman and Frieder [GR04].

Information retrieval research is published in the proceedings of several information retrieval and Web search and mining conferences, including the International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR), the International World Wide Web Conference (WWW), the ACM International Conference on Web Search and Data Mining (WSDM), the ACM Conference on Information and Knowledge Management (CIKM), the European Conference on Information Retrieval (ECIR), the Text Retrieval Conference (TREC), and the ACM/IEEE Joint Conference on Digital Libraries (JCDL). Other sources of publication include major information retrieval, information systems, and Web journals, such as *Journal of Information Retrieval, ACM Transactions on Information Systems (TOIS)*, *Information Processing and Management*, *Knowledge and Information Systems (KAIS)*, and *IEEE Transactions on Knowledge and Data Engineering (TKDE)*.

2 Getting to Know Your Data

It's tempting to jump straight into mining, but first, we need to get the data ready. This involves having a closer look at attributes and data values. Real-world data are typically noisy, enormous in volume (often several gigabytes or more), and may originate from a hodgepodge of heterogenous sources. This chapter is about getting familiar with your data. Knowledge about your data is useful for data preprocessing (see Chapter 3), the first major task of the data mining process. You will want to know the following: What are the types of *attributes* or fields that make up your data? What kind of values does each attribute have? Which attributes are discrete, and which are continuous-valued? What do the data *look like*? How are the values distributed? Are there ways we can visualize the data to get a better sense of it all? Can we spot any outliers? Can we measure the similarity of some data objects with respect to others? Gaining such insight into the data will help with the subsequent analysis.

"*So what can we learn about our data that's helpful in data preprocessing?*" We begin in Section 2.1 by studying the various attribute types. These include nominal attributes, binary attributes, ordinal attributes, and numeric attributes. Basic *statistical descriptions* can be used to learn more about each attribute's values, as described in Section 2.2. Given a *temperature* attribute, for example, we can determine its **mean** (average value), **median** (middle value), and **mode** (most common value). These are **measures of central tendency**, which give us an idea of the "middle" or center of distribution.

Knowing such basic statistics regarding each attribute makes it easier to fill in missing values, smooth noisy values, and spot outliers during data preprocessing. Knowledge of the attributes and attribute values can also help in fixing inconsistencies incurred during data integration. Plotting the measures of central tendency shows us if the data are symmetric or skewed. Quantile plots, histograms, and scatter plots are other graphic displays of basic statistical descriptions. These can all be useful during data preprocessing and can provide insight into areas for mining.

The field of data visualization provides many additional techniques for viewing data through graphical means. These can help identify relations, trends, and biases "hidden" in unstructured data sets. Techniques may be as simple as scatter-plot matrices (where

two attributes are mapped onto a 2-D grid) to more sophisticated methods such as treemaps (where a hierarchical partitioning of the screen is displayed based on the attribute values). Data visualization techniques are described in Section 2.3.

Finally, we may want to examine how similar (or dissimilar) data objects are. For example, suppose we have a database where the data objects are patients, described by their symptoms. We may want to find the similarity or dissimilarity between individual patients. Such information can allow us to find clusters of like patients within the data set. The similarity/dissimilarity between objects may also be used to detect outliers in the data, or to perform nearest-neighbor classification. (Clustering is the topic of Chapters 10 and 11, while nearest-neighbor classification is discussed in Chapter 9.) There are many measures for assessing similarity and dissimilarity. In general, such measures are referred to as proximity measures. Think of the proximity of two objects as a function of the *distance* between their attribute values, although proximity can also be calculated based on probabilities rather than actual distance. Measures of data proximity are described in Section 2.4.

In summary, by the end of this chapter, you will know the different attribute types and basic statistical measures to describe the central tendency and dispersion (spread) of attribute data. You will also know techniques to visualize attribute distributions and how to compute the similarity or dissimilarity between objects.

2.1 Data Objects and Attribute Types

Data sets are made up of data objects. A **data object** represents an entity—in a sales database, the objects may be customers, store items, and sales; in a medical database, the objects may be patients; in a university database, the objects may be students, professors, and courses. Data objects are typically described by attributes. Data objects can also be referred to as *samples, examples, instances, data points*, or *objects*. If the data objects are stored in a database, they are *data tuples*. That is, the rows of a database correspond to the data objects, and the columns correspond to the attributes. In this section, we define attributes and look at the various attribute types.

2.1.1 What Is an Attribute?

An **attribute** is a data field, representing a characteristic or feature of a data object. The nouns *attribute*, *dimension*, *feature*, and *variable* are often used interchangeably in the literature. The term *dimension* is commonly used in data warehousing. Machine learning literature tends to use the term *feature*, while statisticians prefer the term *variable*. Data mining and database professionals commonly use the term *attribute*, and we do here as well. Attributes describing a customer object can include, for example, *customer_ID*, *name*, and *address*. Observed values for a given attribute are known as *observations*. A set of attributes used to describe a given object is called an *attribute vector* (or *feature vector*). The distribution of data involving one attribute (or variable) is called *univariate*. A *bivariate* distribution involves two attributes, and so on.

The **type** of an attribute is determined by the set of possible values—nominal, binary, ordinal, or numeric—the attribute can have. In the following subsections, we introduce each type.

2.1.2 Nominal Attributes

Nominal means "relating to names." The values of a **nominal attribute** are symbols or *names of things*. Each value represents some kind of category, code, or state, and so nominal attributes are also referred to as **categorical**. The values do not have any meaningful order. In computer science, the values are also known as *enumerations*.

Example 2.1 **Nominal attributes.** Suppose that *hair_color* and *marital_status* are two attributes describing *person* objects. In our application, possible values for *hair_color* are *black*, *brown*, *blond*, *red*, *auburn*, *gray*, and *white*. The attribute *marital_status* can take on the values *single, married, divorced*, and *widowed*. Both *hair_color* and *marital_status* are nominal attributes. Another example of a nominal attribute is *occupation*, with the values *teacher, dentist, programmer, farmer*, and so on. ■

Although we said that the values of a nominal attribute are symbols or "names of things," it is possible to represent such symbols or "names" with numbers. With *hair_color*, for instance, we can assign a code of 0 for *black*, 1 for *brown*, and so on. Another example is *customor_ID*, with possible values that are all numeric. However, in such cases, the numbers are not intended to be used quantitatively. That is, mathematical operations on values of nominal attributes are not meaningful. It makes no sense to subtract one customer ID number from another, unlike, say, subtracting an age value from another (where *age* is a numeric attribute). Even though a nominal attribute may have integers as values, it is not considered a numeric attribute because the integers are not meant to be used quantitatively. We will say more on numeric attributes in Section 2.1.5.

Because nominal attribute values do not have any meaningful order about them and are not quantitative, it makes no sense to find the mean (average) value or median (middle) value for such an attribute, given a set of objects. One thing that is of interest, however, is the attribute's most commonly occurring value. This value, known as the *mode*, is one of the measures of central tendency. You will learn about measures of central tendency in Section 2.2.

2.1.3 Binary Attributes

A **binary attribute** is a nominal attribute with only two categories or states: 0 or 1, where 0 typically means that the attribute is absent, and 1 means that it is present. Binary attributes are referred to as **Boolean** if the two states correspond to *true* and *false*.

Example 2.2 **Binary attributes.** Given the attribute *smoker* describing a *patient* object, 1 indicates that the patient smokes, while 0 indicates that the patient does not. Similarly, suppose

the patient undergoes a medical test that has two possible outcomes. The attribute *medical_test* is binary, where a value of 1 means the result of the test for the patient is positive, while 0 means the result is negative. ■

A binary attribute 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*.

A binary attribute is **asymmetric** if the outcomes of the states are not equally important, such as the *positive* and *negative* outcomes of a medical test for HIV. By convention, we 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*).

2.1.4 Ordinal Attributes

An **ordinal attribute** is an attribute with possible values that have a meaningful order or *ranking* among them, but the magnitude between successive values is not known.

Example 2.3 **Ordinal attributes.** Suppose that *drink_size* corresponds to the size of drinks available at a fast-food restaurant. This nominal attribute has three possible values: *small, medium*, and *large*. The values have a meaningful sequence (which corresponds to increasing drink size); however, we cannot tell from the values *how much* bigger, say, a medium is than a large. Other examples of ordinal attributes include *grade* (e.g., *A+*, *A*, *A−*, *B+*, and so on) and *professional_rank*. Professional ranks can be enumerated in a sequential order: for example, *assistant*, *associate*, and *full* for professors, and *private, private first class, specialist, corporal, and sergeant* for army ranks.

Ordinal attributes are useful for registering subjective assessments of qualities that cannot be measured objectively; thus ordinal attributes are often used in surveys for ratings. In one survey, participants were asked to rate how satisfied they were as customers. Customer satisfaction had the following ordinal categories: *0: very dissatisfied, 1: somewhat dissatisfied, 2: neutral, 3: satisfied*, and *4: very satisfied.* ■

Ordinal attributes may also be obtained from the discretization of numeric quantities by splitting the value range into a finite number of ordered categories as described in Chapter 3 on data reduction.

The central tendency of an ordinal attribute can be represented by its mode and its median (the middle value in an ordered sequence), but the mean cannot be defined.

Note that nominal, binary, and ordinal attributes are *qualitative*. That is, they *describe* a feature of an object without giving an actual size or quantity. The values of such qualitative attributes are typically words representing categories. If integers are used, they represent computer codes for the categories, as opposed to measurable quantities (e.g., 0 for *small* drink size, 1 for *medium*, and 2 for *large*). In the following subsection we look at numeric attributes, which provide *quantitative* measurements of an object.

2.1.5 Numeric Attributes

A **numeric attribute** is *quantitative*; that is, it is a measurable quantity, represented in integer or real values. Numeric attributes can be *interval-scaled* or *ratio-scaled*.

Interval-Scaled Attributes

Interval-scaled attributes are measured on a scale of equal-size units. The values of interval-scaled attributes have order and can be positive, 0, or negative. Thus, in addition to providing a ranking of values, such attributes allow us to compare and quantify the *difference* between values.

Example 2.4 **Interval-scaled attributes.** A *temperature* attribute is interval-scaled. Suppose that we have the outdoor *temperature* value for a number of different days, where each day is an object. By ordering the values, we obtain a ranking of the objects with respect to *temperature*. In addition, we can quantify the difference between values. For example, a temperature of 20°C is five degrees higher than a temperature of 15°C. Calendar dates are another example. For instance, the years 2002 and 2010 are eight years apart. ■

Temperatures in Celsius and Fahrenheit do not have a true zero-point, that is, neither 0°C nor 0°F indicates "no temperature." (On the Celsius scale, for example, the unit of measurement is 1/100 of the difference between the melting temperature and the boiling temperature of water in atmospheric pressure.) Although we can compute the *difference* between temperature values, we cannot talk of one temperature value as being a *multiple* of another. Without a true zero, we cannot say, for instance, that 10°C is twice as warm as 5°C. That is, we cannot speak of the values in terms of ratios. Similarly, there is no true zero-point for calendar dates. (The year 0 does not correspond to the beginning of time.) This brings us to ratio-scaled attributes, for which a true zero-point exits.

Because interval-scaled attributes are numeric, we can compute their mean value, in addition to the median and mode measures of central tendency.

Ratio-Scaled Attributes

A **ratio-scaled attribute** is a numeric attribute with an inherent zero-point. That is, if a measurement is ratio-scaled, we can speak of a value as being a multiple (or ratio) of another value. In addition, the values are ordered, and we can also compute the difference between values, as well as the mean, median, and mode.

Example 2.5 **Ratio-scaled attributes.** Unlike temperatures in Celsius and Fahrenheit, the Kelvin (K) temperature scale has what is considered a true zero-point ($0°K = -273.15°C$): It is the point at which the particles that comprise matter have zero kinetic energy. Other examples of ratio-scaled attributes include *count* attributes such as *years_of_experience* (e.g., the objects are employees) and *number_of_words* (e.g., the objects are documents).

Additional examples include attributes to measure weight, height, and speed, and monetary quantities (e.g., you are 100 times richer with $100 than with $1). ■

2.1.6 Discrete versus Continuous Attributes

In our presentation, we have organized attributes into nominal, binary, ordinal, and numeric types. There are many ways to organize attribute types. The types are not mutually exclusive.

Classification algorithms developed from the field of machine learning often talk of attributes as being either *discrete* or *continuous*. Each type may be processed differently. A **discrete attribute** has a finite or countably infinite set of values, which may or may not be represented as integers. The attributes *hair_color*, *smoker*, *medical_test*, and *drink_size* each have a finite number of values, and so are discrete. Note that discrete attributes may have numeric values, such as 0 and 1 for binary attributes or, the values 0 to 110 for the attribute *age*. An attribute is *countably infinite* if the set of possible values is infinite but the values can be put in a one-to-one correspondence with natural numbers. For example, the attribute *customer_ID* is countably infinite. The number of customers can grow to infinity, but in reality, the actual set of values is countable (where the values can be put in one-to-one correspondence with the set of integers). Zip codes are another example.

If an attribute is not discrete, it is **continuous**. The terms *numeric attribute* and *continuous attribute* are often used interchangeably in the literature. (This can be confusing because, in the classic sense, continuous values are real numbers, whereas numeric values can be either integers or real numbers.) In practice, real values are represented using a finite number of digits. Continuous attributes are typically represented as floating-point variables.

2.2 Basic Statistical Descriptions of Data

For data preprocessing to be successful, it is essential to have an overall picture of your data. Basic statistical descriptions can be used to identify properties of the data and highlight which data values should be treated as noise or outliers.

This section discusses three areas of basic statistical descriptions. We start with *measures of central tendency* (Section 2.2.1), which measure the location of the middle or center of a data distribution. Intuitively speaking, given an attribute, where do most of its values fall? In particular, we discuss the mean, median, mode, and midrange.

In addition to assessing the central tendency of our data set, we also would like to have an idea of the *dispersion of the data*. That is, how are the data spread out? The most common data dispersion measures are the *range*, *quartiles*, and *interquartile range*; the *five-number summary* and *boxplots*; and the *variance* and *standard deviation* of the data. These measures are useful for identifying outliers and are described in Section 2.2.2.

Finally, we can use many graphic displays of basic statistical descriptions to visually inspect our data (Section 2.2.3). Most statistical or graphical data presentation software

packages include bar charts, pie charts, and line graphs. Other popular displays of data summaries and distributions include *quantile plots*, *quantile–quantile plots*, *histograms*, and *scatter plots*.

2.2.1 Measuring the Central Tendency: Mean, Median, and Mode

In this section, we look at various ways to measure the central tendency of data. Suppose that we have some attribute X, like *salary*, which has been recorded for a set of objects. Let $x_1, x_2, \ldots, x_N$ be the set of N observed values or *observations* for X. Here, these values may also be referred to as the data set (for X). If we were to plot the observations for *salary*, where would most of the values fall? This gives us an idea of the central tendency of the data. Measures of central tendency include the mean, median, mode, and midrange.

The most common and effective numeric measure of the "center" of a set of data is the *(arithmetic) mean*. Let $x_1, x_2, \ldots, x_N$ be a set of N values or *observations*, such as for some numeric attribute X, like *salary*. The **mean** of this set of values is

$$\bar{x} = \frac{\sum_{i=1}^{N} x_i}{N} = \frac{x_1 + x_2 + \cdots + x_N}{N}. \tag{2.1}$$

This corresponds to the built-in aggregate function, *average* (**avg()** in SQL), provided in relational database systems.

Example 2.6 **Mean.** Suppose we have the following values for *salary* (in thousands of dollars), shown in increasing order: 30, 36, 47, 50, 52, 52, 56, 60, 63, 70, 70, 110. Using Eq. (2.1), we have

$$\bar{x} = \frac{30 + 36 + 47 + 50 + 52 + 52 + 56 + 60 + 63 + 70 + 70 + 110}{12}$$
$$= \frac{696}{12} = 58.$$

Thus, the mean salary is $58,000. ■

Sometimes, each value x_i in a set may be associated with a weight w_i for $i = 1, \ldots, N$. The weights reflect the significance, importance, or occurrence frequency attached to their respective values. In this case, we can compute

$$\bar{x} = \frac{\sum_{i=1}^{N} w_i x_i}{\sum_{i=1}^{N} w_i} = \frac{w_1 x_1 + w_2 x_2 + \cdots + w_N x_N}{w_1 + w_2 + \cdots + w_N}. \tag{2.2}$$

This is called the **weighted arithmetic mean** or the **weighted average**.

Although the mean is the singlemost useful quantity for describing a data set, it is not always the best way of measuring the center of the data. A major problem with the mean is its sensitivity to extreme (e.g., outlier) values. Even a small number of extreme values can corrupt the mean. For example, the mean salary at a company may be substantially pushed up by that of a few highly paid managers. Similarly, the mean score of a class in an exam could be pulled down quite a bit by a few very low scores. To offset the effect caused by a small number of extreme values, we can instead use the **trimmed mean**, which is the mean obtained after chopping off values at the high and low extremes. For example, we can sort the values observed for *salary* and remove the top and bottom 2% before computing the mean. We should avoid trimming too large a portion (such as 20%) at both ends, as this can result in the loss of valuable information.

For skewed (asymmetric) data, a better measure of the center of data is the **median**, which is the middle value in a set of ordered data values. It is the value that separates the higher half of a data set from the lower half.

In probability and statistics, the median generally applies to numeric data; however, we may extend the concept to ordinal data. Suppose that a given data set of N values for an attribute X is sorted in increasing order. If N is odd, then the median is the *middle value* of the ordered set. If N is even, then the median is not unique; it is the two middlemost values and any value in between. If X is a numeric attribute in this case, by convention, the median is taken as the average of the two middlemost values.

Example 2.7 **Median.** Let's find the median of the data from Example 2.6. The data are already sorted in increasing order. There is an even number of observations (i.e., 12); therefore, the median is not unique. It can be any value within the two middlemost values of 52 and 56 (that is, within the sixth and seventh values in the list). By convention, we assign the average of the two middlemost values as the median; that is, $\frac{52+56}{2} = \frac{108}{2} = 54$. Thus, the median is \$54,000.

Suppose that we had only the first 11 values in the list. Given an odd number of values, the median is the middlemost value. This is the sixth value in this list, which has a value of \$52,000. ■

The median is expensive to compute when we have a large number of observations. For numeric attributes, however, we can easily *approximate* the value. Assume that data are grouped in intervals according to their x_i data values and that the frequency (i.e., number of data values) of each interval is known. For example, employees may be grouped according to their annual salary in intervals such as \$10–20,000, \$20–30,000, and so on. Let the interval that contains the median frequency be the *median interval*. We can approximate the median of the entire data set (e.g., the median salary) by interpolation using the formula

$$median = L_1 + \left(\frac{N/2 - \left(\sum freq\right)_l}{freq_{median}} \right) width, \tag{2.3}$$

where L_1 is the lower boundary of the median interval, N is the number of values in the entire data set, $\left(\sum freq\right)_l$ is the sum of the frequencies of all of the intervals that are

lower than the median interval, $freq_{median}$ is the frequency of the median interval, and *width* is the width of the median interval.

The *mode* is another measure of central tendency. The **mode** for a set of data is the value that occurs most frequently in the set. Therefore, it can be determined for qualitative and quantitative attributes. It is possible for the greatest frequency to correspond to several different values, which results in more than one mode. Data sets with one, two, or three modes are respectively called **unimodal**, **bimodal**, and **trimodal**. In general, a data set with two or more modes is **multimodal**. At the other extreme, if each data value occurs only once, then there is no mode.

Example 2.8 **Mode.** The data from Example 2.6 are bimodal. The two modes are \$52,000 and \$70,000. ■

For unimodal numeric data that are moderately skewed (asymmetrical), we have the following empirical relation:

$$mean - mode \approx 3 \times (mean - median). \tag{2.4}$$

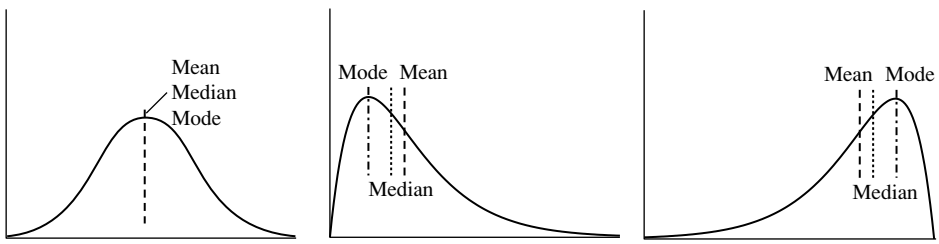
This implies that the mode for unimodal frequency curves that are moderately skewed can easily be approximated if the mean and median values are known.

The **midrange** can also be used to assess the central tendency of a numeric data set. It is the average of the largest and smallest values in the set. This measure is easy to compute using the SQL aggregate functions, max() and min().

Example 2.9 **Midrange.** The midrange of the data of Example 2.6 is $\frac{30{,}000+110{,}000}{2} = \$70{,}000$. ■

In a unimodal frequency curve with perfect **symmetric** data distribution, the mean, median, and mode are all at the same center value, as shown in Figure 2.1(a).

Data in most real applications are not symmetric. They may instead be either **positively skewed**, where the mode occurs at a value that is smaller than the median (Figure 2.1b), or **negatively skewed**, where the mode occurs at a value greater than the median (Figure 2.1c).



(a) Symmetric data **(b)** Positively skewed data **(c)** Negatively skewed data

Figure 2.1 Mean, median, and mode of symmetric versus positively and negatively skewed data.

2.2.2 Measuring the Dispersion of Data: Range, Quartiles, Variance, Standard Deviation, and Interquartile Range

We now look at measures to assess the dispersion or spread of numeric data. The measures include range, quantiles, quartiles, percentiles, and the interquartile range. The five-number summary, which can be displayed as a boxplot, is useful in identifying outliers. Variance and standard deviation also indicate the spread of a data distribution.

Range, Quartiles, and Interquartile Range

To start off, let's study the *range*, *quantiles*, *quartiles*, *percentiles*, and the *interquartile range* as measures of data dispersion.

Let $x_1, x_2, \ldots, x_N$ be a set of observations for some numeric attribute, X. The **range** of the set is the difference between the largest (max()) and smallest (min()) values.

Suppose that the data for attribute X are sorted in increasing numeric order. Imagine that we can pick certain data points so as to split the data distribution into equal-size consecutive sets, as in Figure 2.2. These data points are called *quantiles*. **Quantiles** are points taken at regular intervals of a data distribution, dividing it into essentially equal-size consecutive sets. (We say "essentially" because there may not be data values of X that divide the data into exactly equal-sized subsets. For readability, we will refer to them as equal.) The kth *q-quantile* for a given data distribution is the value x such that at most k/q of the data values are less than x and at most $(q-k)/q$ of the data values are more than x, where k is an integer such that $0 < k < q$. There are $q-1$ q-quantiles.

The 2-quantile is the data point dividing the lower and upper halves of the data distribution. It corresponds to the median. The 4-quantiles are the three data points that split the data distribution into four equal parts; each part represents one-fourth of the data distribution. They are more commonly referred to as **quartiles**. The 100-quantiles are more commonly referred to as **percentiles**; they divide the data distribution into 100 equal-sized consecutive sets. The median, quartiles, and percentiles are the most widely used forms of quantiles.

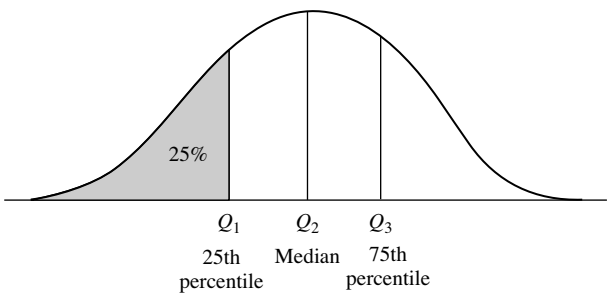


Figure 2.2 A plot of the data distribution for some attribute X. The quantiles plotted are quartiles. The three quartiles divide the distribution into four equal-size consecutive subsets. The second quartile corresponds to the median.

The quartiles give an indication of a distribution's center, spread, and shape. The **first quartile**, denoted by Q_1, is the 25th percentile. It cuts off the lowest 25% of the data. The **third quartile**, denoted by Q_3, is the 75th percentile—it cuts off the lowest 75% (or highest 25%) of the data. The second quartile is the 50th percentile. As the median, it gives the center of the data distribution.

The distance between the first and third quartiles is a simple measure of spread that gives the range covered by the middle half of the data. This distance is called the **interquartile range** (**IQR**) and is defined as

$$IQR = Q_3 - Q_1. \tag{2.5}$$

Example 2.10 **Interquartile range.** The quartiles are the three values that split the sorted data set into four equal parts. The data of Example 2.6 contain 12 observations, already sorted in increasing order. Thus, the quartiles for this data are the third, sixth, and ninth values, respectively, in the sorted list. Therefore, $Q_1 = \$47{,}000$ and Q_3 is \$63,000. Thus, the interquartile range is $IQR = 63 - 47 = \$16{,}000$. (Note that the sixth value is a median, \$52,000, although this data set has two medians since the number of data values is even.) ■

Five-Number Summary, Boxplots, and Outliers

No single numeric measure of spread (e.g., *IQR*) is very useful for describing skewed distributions. Have a look at the symmetric and skewed data distributions of Figure 2.1. In the symmetric distribution, the median (and other measures of central tendency) splits the data into equal-size halves. This does not occur for skewed distributions. Therefore, it is more informative to also provide the two quartiles Q_1 and Q_3, along with the median. A common rule of thumb for identifying suspected **outliers** is to single out values falling at least $1.5 \times IQR$ above the third quartile or below the first quartile.

Because Q_1, the median, and Q_3 together contain no information about the endpoints (e.g., tails) of the data, a fuller summary of the shape of a distribution can be obtained by providing the lowest and highest data values as well. This is known as the *five-number summary*. The **five-number summary** of a distribution consists of the median (Q_2), the quartiles Q_1 and Q_3, and the smallest and largest individual observations, written in the order of *Minimum*, Q_1, *Median*, Q_3, *Maximum*.

Boxplots are a popular way of visualizing a distribution. A boxplot incorporates the five-number summary as follows:

- Typically, the ends of the box are at the quartiles so that the box length is the interquartile range.
- The median is marked by a line within the box.
- Two lines (called *whiskers*) outside the box extend to the smallest (*Minimum*) and largest (*Maximum*) observations.

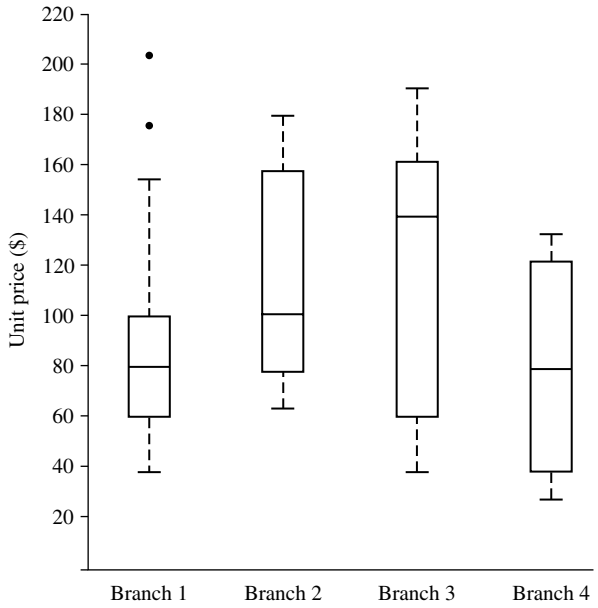


Figure 2.3 Boxplot for the unit price data for items sold at four branches of *AllElectronics* during a given time period.

When dealing with a moderate number of observations, it is worthwhile to plot potential outliers individually. To do this in a boxplot, the whiskers are extended to the extreme low and high observations *only if* these values are less than $1.5 \times IQR$ beyond the quartiles. Otherwise, the whiskers terminate at the most extreme observations occurring within $1.5 \times IQR$ of the quartiles. The remaining cases are plotted individually. Boxplots can be used in the comparisons of several sets of compatible data.

Example 2.11 **Boxplot.** Figure 2.3 shows boxplots for unit price data for items sold at four branches of *AllElectronics* during a given time period. For branch 1, we see that the median price of items sold is \$80, Q_1 is \$60, and Q_3 is \$100. Notice that two outlying observations for this branch were plotted individually, as their values of 175 and 202 are more than 1.5 times the IQR here of 40. ■

Boxplots can be computed in $O(n \log n)$ time. Approximate boxplots can be computed in linear or sublinear time depending on the quality guarantee required.

Variance and Standard Deviation

Variance and standard deviation are measures of data dispersion. They indicate how spread out a data distribution is. A low standard deviation means that the data observations tend to be very close to the mean, while a high standard deviation indicates that the data are spread out over a large range of values.

The **variance** of N observations, $x_1, x_2, \ldots, x_N$, for a numeric attribute X is

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2 = \left(\frac{1}{N}\sum_{i=1}^{N}x_i^2\right) - \bar{x}^2, \tag{2.6}$$

where $\bar{x}$ is the mean value of the observations, as defined in Eq. (2.1). The **standard deviation**, σ, of the observations is the square root of the variance, σ^2.

Example 2.12 **Variance and standard deviation.** In Example 2.6, we found $\bar{x} = \$58{,}000$ using Eq. (2.1) for the mean. To determine the variance and standard deviation of the data from that example, we set $N = 12$ and use Eq. (2.6) to obtain

$$\begin{aligned}\sigma^2 &= \frac{1}{12}(30^2 + 36^2 + 47^2 \ldots + 110^2) - 58^2 \\ &\approx 379.17 \\ \sigma &\approx \sqrt{379.17} \approx 19.47.\end{aligned}$$

■

The basic properties of the standard deviation, σ, as a measure of spread are as follows:

- σ measures spread about the mean and should be considered only when the mean is chosen as the measure of center.
- $\sigma = 0$ only when there is no spread, that is, when all observations have the same value. Otherwise, $\sigma > 0$.

Importantly, an observation is unlikely to be more than several standard deviations away from the mean. Mathematically, using Chebyshev's inequality, it can be shown that at least $\left(1 - \frac{1}{k^2}\right) \times 100\%$ of the observations are no more than k standard deviations from the mean. Therefore, the standard deviation is a good indicator of the spread of a data set.

The computation of the variance and standard deviation is scalable in large databases.

2.2.3 Graphic Displays of Basic Statistical Descriptions of Data

In this section, we study graphic displays of basic statistical descriptions. These include *quantile plots*, *quantile–quantile plots*, *histograms*, and *scatter plots*. Such graphs are helpful for the visual inspection of data, which is useful for data preprocessing. The first three of these show univariate distributions (i.e., data for one attribute), while scatter plots show bivariate distributions (i.e., involving two attributes).

Quantile Plot

In this and the following subsections, we cover common graphic displays of data distributions. A **quantile plot** is a simple and effective way to have a first look at a univariate data distribution. First, it displays all of the data for the given attribute (allowing the user

to assess both the overall behavior and unusual occurrences). Second, it plots quantile information (see Section 2.2.2). Let x_i, for $i = 1$ to N, be the data sorted in increasing order so that x_1 is the smallest observation and x_N is the largest for some ordinal or numeric attribute X. Each observation, x_i, is paired with a percentage, f_i, which indicates that approximately $f_i \times 100\%$ of the data are below the value, x_i. We say "approximately" because there may not be a value with exactly a fraction, f_i, of the data below x_i. Note that the 0.25 percentile corresponds to quartile Q_1, the 0.50 percentile is the median, and the 0.75 percentile is Q_3.

Let

$$f_i = \frac{i - 0.5}{N}. \tag{2.7}$$

These numbers increase in equal steps of $1/N$, ranging from $\frac{1}{2N}$ (which is slightly above 0) to $1 - \frac{1}{2N}$ (which is slightly below 1). On a quantile plot, x_i is graphed against f_i. This allows us to compare different distributions based on their quantiles. For example, given the quantile plots of sales data for two different time periods, we can compare their Q_1, median, Q_3, and other f_i values at a glance.

Example 2.13 **Quantile plot.** Figure 2.4 shows a quantile plot for the *unit price* data of Table 2.1. ■

Quantile–Quantile Plot

A **quantile–quantile plot**, or **q-q plot**, graphs the quantiles of one univariate distribution against the corresponding quantiles of another. It is a powerful visualization tool in that it allows the user to view whether there is a shift in going from one distribution to another.

Suppose that we have two sets of observations for the attribute or variable *unit price*, taken from two different branch locations. Let $x_1, \ldots, x_N$ be the data from the first branch, and $y_1, \ldots, y_M$ be the data from the second, where each data set is sorted in increasing order. If $M = N$ (i.e., the number of points in each set is the same), then we simply plot y_i against x_i, where y_i and x_i are both $(i - 0.5)/N$ quantiles of their respective data sets. If $M < N$ (i.e., the second branch has fewer observations than the first), there can be only M points on the q-q plot. Here, y_i is the $(i - 0.5)/M$ quantile of the y

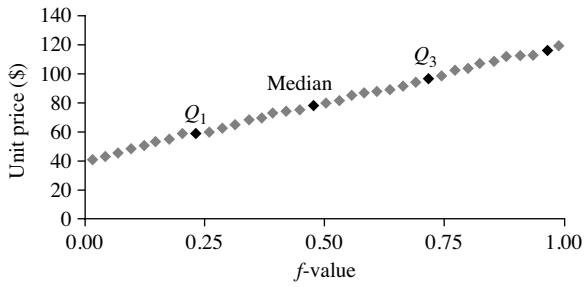


Figure 2.4 A quantile plot for the unit price data of Table 2.1.

Table 2.1 A Set of Unit Price Data for Items Sold at a Branch of *AllElectronics*

Unit price ($)	*Count of items sold*
40	275
43	300
47	250
—	—
74	360
75	515
78	540
—	—
115	320
117	270
120	350

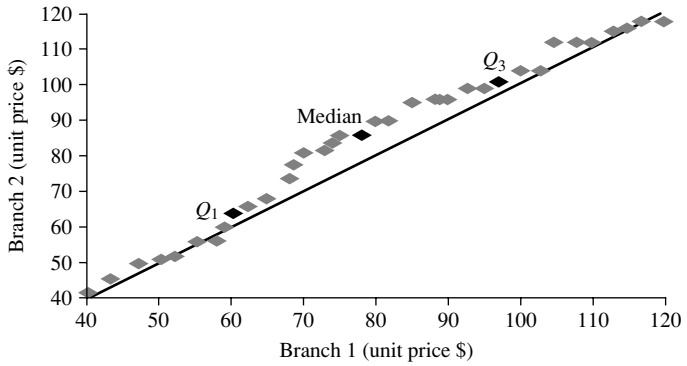


Figure 2.5 A q-q plot for unit price data from two *AllElectronics* branches.

data, which is plotted against the $(i - 0.5)/M$ quantile of the x data. This computation typically involves interpolation.

Example 2.14 **Quantile–quantile plot.** Figure 2.5 shows a quantile–quantile plot for *unit price* data of items sold at two branches of *AllElectronics* during a given time period. Each point corresponds to the same quantile for each data set and shows the unit price of items sold at branch 1 versus branch 2 for that quantile. (To aid in comparison, the straight line represents the case where, for each given quantile, the unit price at each branch is the same. The darker points correspond to the data for Q_1, the median, and Q_3, respectively.)

We see, for example, that at Q_1, the unit price of items sold at branch 1 was slightly less than that at branch 2. In other words, 25% of items sold at branch 1 were less than or

equal to \$60, while 25% of items sold at branch 2 were less than or equal to \$64. At the 50th percentile (marked by the median, which is also Q_2), we see that 50% of items sold at branch 1 were less than \$78, while 50% of items at branch 2 were less than \$85. In general, we note that there is a shift in the distribution of branch 1 with respect to branch 2 in that the unit prices of items sold at branch 1 tend to be lower than those at branch 2. ■

Histograms

Histograms (or **frequency histograms**) are at least a century old and are widely used. "Histos" means pole or mast, and "gram" means chart, so a histogram is a chart of poles. Plotting histograms is a graphical method for summarizing the distribution of a given attribute, X. If X is nominal, such as *automobile_model* or *item_type*, then a pole or vertical bar is drawn for each known value of X. The height of the bar indicates the frequency (i.e., count) of that X value. The resulting graph is more commonly known as a **bar chart**.

If X is numeric, the term *histogram* is preferred. The range of values for X is partitioned into disjoint consecutive subranges. The subranges, referred to as *buckets* or *bins*, are disjoint subsets of the data distribution for X. The range of a bucket is known as the **width**. Typically, the buckets are of equal width. For example, a *price* attribute with a value range of \$1 to \$200 (rounded up to the nearest dollar) can be partitioned into subranges 1 to 20, 21 to 40, 41 to 60, and so on. For each subrange, a bar is drawn with a height that represents the total count of items observed within the subrange. Histograms and partitioning rules are further discussed in Chapter 3 on data reduction.

Example 2.15 **Histogram.** Figure 2.6 shows a histogram for the data set of Table 2.1, where buckets (or bins) are defined by equal-width ranges representing \$20 increments and the frequency is the count of items sold. ■

Although histograms are widely used, they may not be as effective as the quantile plot, q-q plot, and boxplot methods in comparing groups of univariate observations.

Scatter Plots and Data Correlation

A **scatter plot** is one of the most effective graphical methods for determining if there appears to be a relationship, pattern, or trend between two numeric attributes. To construct a scatter plot, each pair of values is treated as a pair of coordinates in an algebraic sense and plotted as points in the plane. Figure 2.7 shows a scatter plot for the set of data in Table 2.1.

The scatter plot is a useful method for providing a first look at bivariate data to see clusters of points and outliers, or to explore the possibility of correlation relationships. Two attributes, X, and Y, are **correlated** if one attribute implies the other. Correlations can be positive, negative, or null (uncorrelated). Figure 2.8 shows examples of positive and negative correlations between two attributes. If the plotted points pattern slopes

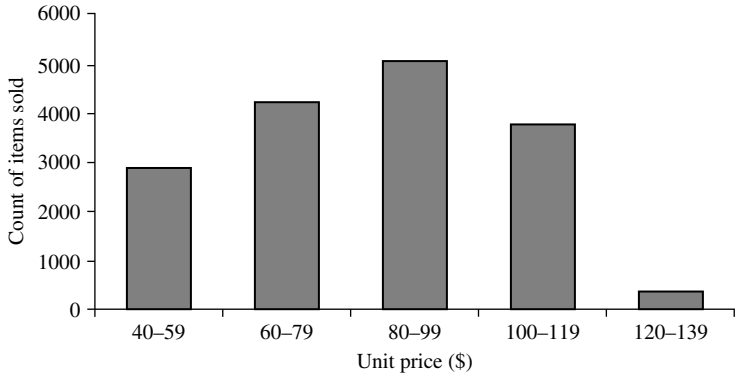


Figure 2.6 A histogram for the Table 2.1 data set.

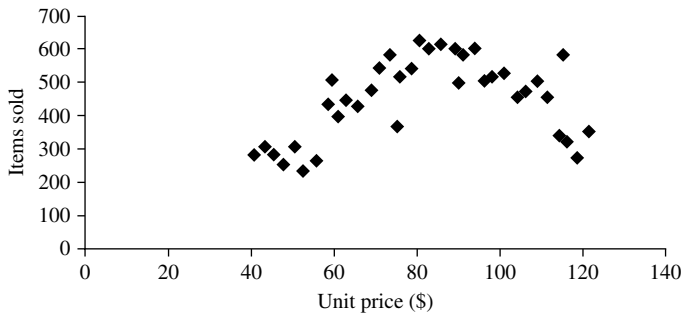


Figure 2.7 A scatter plot for the Table 2.1 data set.

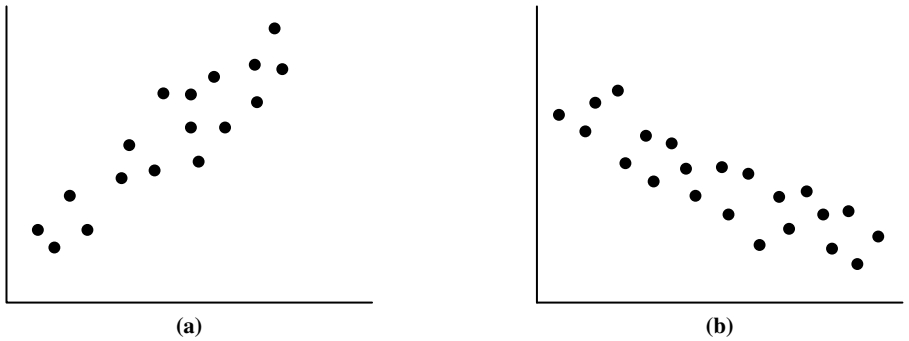


Figure 2.8 Scatter plots can be used to find (a) positive or (b) negative correlations between attributes.

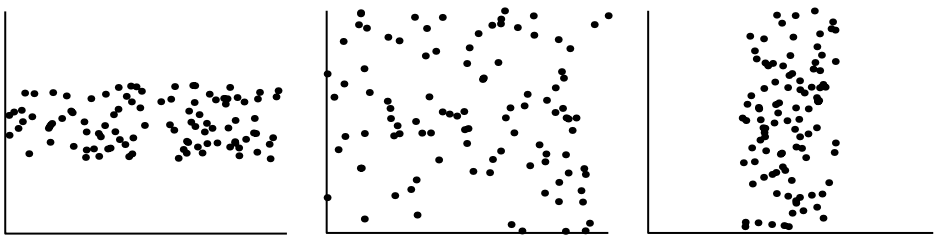


Figure 2.9 Three cases where there is no observed correlation between the two plotted attributes in each of the data sets.

from lower left to upper right, this means that the values of X increase as the values of Y increase, suggesting a *positive correlation* (Figure 2.8a). If the pattern of plotted points slopes from upper left to lower right, the values of X increase as the values of Y decrease, suggesting a *negative correlation* (Figure 2.8b). A line of best fit can be drawn to study the correlation between the variables. Statistical tests for correlation are given in Chapter 3 on data integration (Eq. (3.3)). Figure 2.9 shows three cases for which there is no correlation relationship between the two attributes in each of the given data sets. Section 2.3.2 shows how scatter plots can be extended to n attributes, resulting in a *scatter-plot matrix*.

In conclusion, basic data descriptions (e.g., measures of central tendency and measures of dispersion) and graphic statistical displays (e.g., quantile plots, histograms, and scatter plots) provide valuable insight into the overall behavior of your data. By helping to identify noise and outliers, they are especially useful for data cleaning.

2.3 Data Visualization

How can we convey data to users effectively? **Data visualization** aims to communicate data clearly and effectively through graphical representation. Data visualization has been used extensively in many applications—for example, at work for reporting, managing business operations, and tracking progress of tasks. More popularly, we can take advantage of visualization techniques to discover data relationships that are otherwise not easily observable by looking at the raw data. Nowadays, people also use data visualization to create fun and interesting graphics.

In this section, we briefly introduce the basic concepts of data visualization. We start with multidimensional data such as those stored in relational databases. We discuss several representative approaches, including pixel-oriented techniques, geometric projection techniques, icon-based techniques, and hierarchical and graph-based techniques. We then discuss the visualization of complex data and relations.

2.3.1 Pixel-Oriented Visualization Techniques

A simple way to visualize the value of a dimension is to use a pixel where the color of the pixel reflects the dimension's value. For a data set of m dimensions, **pixel-oriented techniques** create m windows on the screen, one for each dimension. The m dimension values of a record are mapped to m pixels at the corresponding positions in the windows. The colors of the pixels reflect the corresponding values.

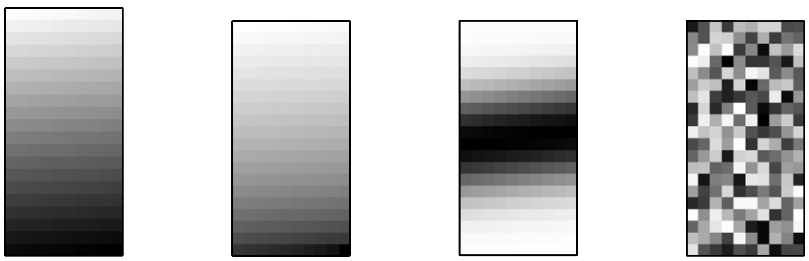
Inside a window, the data values are arranged in some global order shared by all windows. The global order may be obtained by sorting all data records in a way that's meaningful for the task at hand.

Example 2.16 **Pixel-oriented visualization.** *AllElectronics* maintains a customer information table, which consists of four dimensions: *income*, *credit_limit*, *transaction_volume*, and *age*. Can we analyze the correlation between *income* and the other attributes by visualization?

We can sort all customers in income-ascending order, and use this order to lay out the customer data in the four visualization windows, as shown in Figure 2.10. The pixel colors are chosen so that the smaller the value, the lighter the shading. Using pixel-based visualization, we can easily observe the following: *credit_limit* increases as *income* increases; customers whose income is in the middle range are more likely to purchase more from *AllElectronics*; there is no clear correlation between *income* and *age*. ■

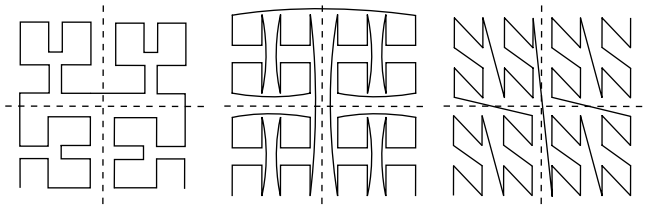
In pixel-oriented techniques, data records can also be ordered in a query-dependent way. For example, given a point query, we can sort all records in descending order of similarity to the point query.

Filling a window by laying out the data records in a linear way may not work well for a wide window. The first pixel in a row is far away from the last pixel in the previous row, though they are next to each other in the global order. Moreover, a pixel is next to the one above it in the window, even though the two are not next to each other in the global order. To solve this problem, we can lay out the data records in a space-filling curve



(a) *income* **(b)** *credit_limit* **(c)** *transaction_volume* **(d)** *age*

Figure 2.10 Pixel-oriented visualization of four attributes by sorting all customers in *income* ascending order.



(a) Hilbert curve **(b)** Gray code **(c)** Z-curve

Figure 2.11 Some frequently used 2-D space-filling curves.

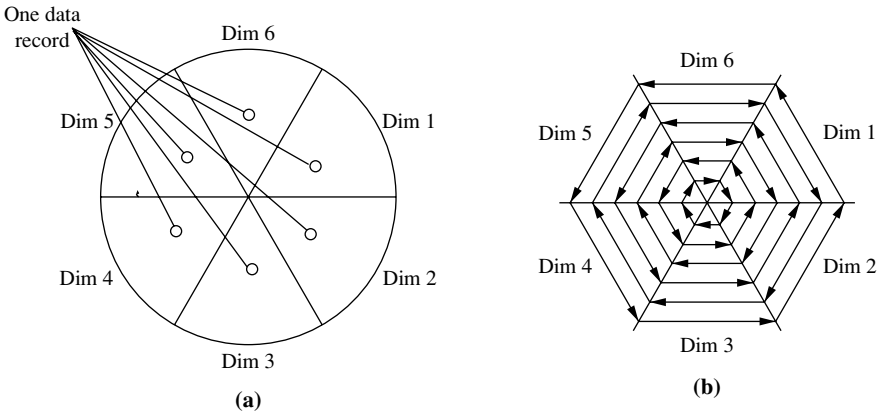


Figure 2.12 The circle segment technique. (a) Representing a data record in circle segments. (b) Laying out pixels in circle segments.

to fill the windows. A *space-filling curve* is a curve with a range that covers the entire n-dimensional unit hypercube. Since the visualization windows are 2-D, we can use any 2-D space-filling curve. Figure 2.11 shows some frequently used 2-D space-filling curves.

Note that the windows do not have to be rectangular. For example, the *circle segment technique* uses windows in the shape of segments of a circle, as illustrated in Figure 2.12. This technique can ease the comparison of dimensions because the dimension windows are located side by side and form a circle.

2.3.2 Geometric Projection Visualization Techniques

A drawback of pixel-oriented visualization techniques is that they cannot help us much in understanding the distribution of data in a multidimensional space. For example, they do not show whether there is a dense area in a multidimensional subspace. **Geometric**

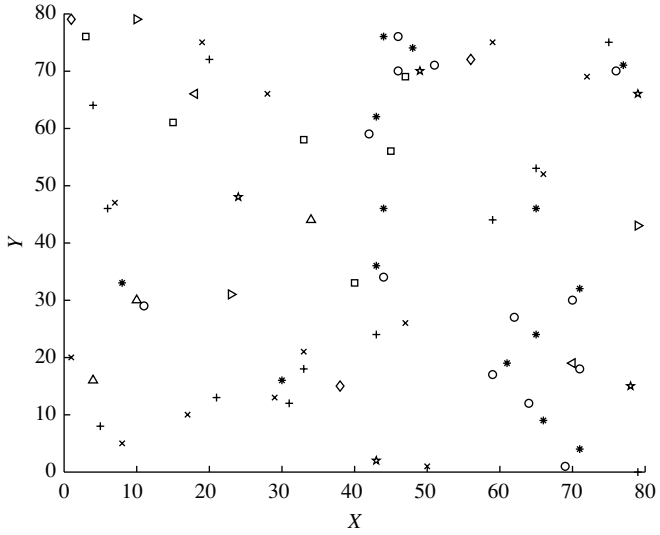


Figure 2.13 Visualization of a 2-D data set using a scatter plot. *Source: www.cs.sfu.ca/jpei/publications/rareevent-geoinformatica06.pdf*.

projection techniques help users find interesting projections of multidimensional data sets. The central challenge the geometric projection techniques try to address is how to visualize a high-dimensional space on a 2-D display.

A **scatter plot** displays 2-D data points using Cartesian coordinates. A third dimension can be added using different colors or shapes to represent different data points. Figure 2.13 shows an example, where X and Y are two spatial attributes and the third dimension is represented by different shapes. Through this visualization, we can see that points of types "+" and "×" tend to be colocated.

A 3-D scatter plot uses three axes in a Cartesian coordinate system. If it also uses color, it can display up to 4-D data points (Figure 2.14).

For data sets with more than four dimensions, scatter plots are usually ineffective. The **scatter-plot matrix** technique is a useful extension to the scatter plot. For an n-dimensional data set, a scatter-plot matrix is an $n \times n$ grid of 2-D scatter plots that provides a visualization of each dimension with every other dimension. Figure 2.15 shows an example, which visualizes the Iris data set. The data set consists of 450 samples from each of three species of Iris flowers. There are five dimensions in the data set: length and width of sepal and petal, and species.

The scatter-plot matrix becomes less effective as the dimensionality increases. Another popular technique, called parallel coordinates, can handle higher dimensionality. To visualize n-dimensional data points, the **parallel coordinates** technique draws n equally spaced axes, one for each dimension, parallel to one of the display axes.

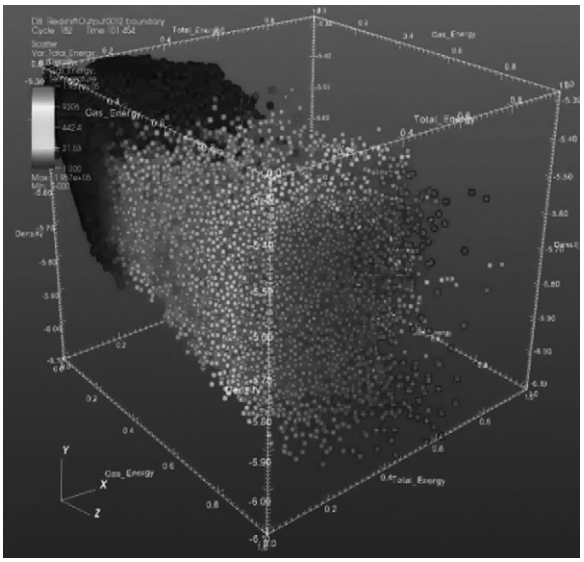


Figure 2.14 Visualization of a 3-D data set using a scatter plot. *Source: http://upload.wikimedia.org/wikipedia/commons/c/c4/Scatter_plot.jpg.*

A data record is represented by a polygonal line that intersects each axis at the point corresponding to the associated dimension value (Figure 2.16).

A major limitation of the parallel coordinates technique is that it cannot effectively show a data set of many records. Even for a data set of several thousand records, visual clutter and overlap often reduce the readability of the visualization and make the patterns hard to find.

2.3.3 Icon-Based Visualization Techniques

Icon-based visualization techniques use small icons to represent multidimensional data values. We look at two popular icon-based techniques: *Chernoff faces* and *stick figures*.

Chernoff faces were introduced in 1973 by statistician Herman Chernoff. They display multidimensional data of up to 18 variables (or dimensions) as a cartoon human face (Figure 2.17). Chernoff faces help reveal trends in the data. Components of the face, such as the eyes, ears, mouth, and nose, represent values of the dimensions by their shape, size, placement, and orientation. For example, dimensions can be mapped to the following facial characteristics: eye size, eye spacing, nose length, nose width, mouth curvature, mouth width, mouth openness, pupil size, eyebrow slant, eye eccentricity, and head eccentricity.

Chernoff faces make use of the ability of the human mind to recognize small differences in facial characteristics and to assimilate many facial characteristics at once.

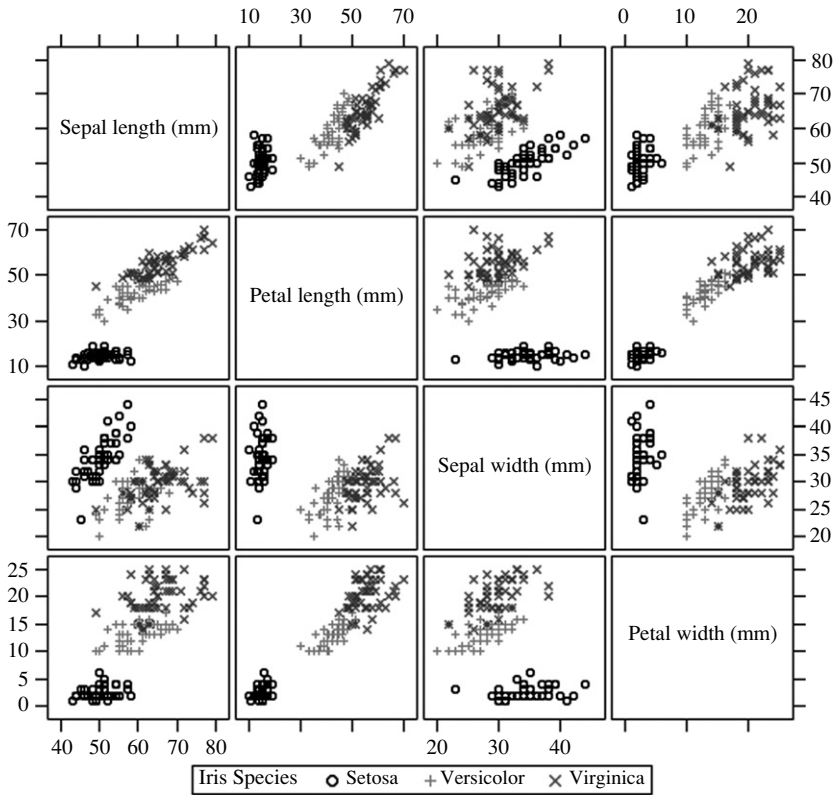


Figure 2.15 Visualization of the Iris data set using a scatter-plot matrix. *Source: http://support.sas.com/documentation/cdl/en/grstatproc/61948/HTML/default/images/gsgscmat.gif*.

Viewing large tables of data can be tedious. By condensing the data, Chernoff faces make the data easier for users to digest. In this way, they facilitate visualization of regularities and irregularities present in the data, although their power in relating multiple relationships is limited. Another limitation is that specific data values are not shown. Furthermore, facial features vary in perceived importance. This means that the similarity of two faces (representing two multidimensional data points) can vary depending on the order in which dimensions are assigned to facial characteristics. Therefore, this mapping should be carefully chosen. Eye size and eyebrow slant have been found to be important.

Asymmetrical Chernoff faces were proposed as an extension to the original technique. Since a face has vertical symmetry (along the y-axis), the left and right side of a face are identical, which wastes space. Asymmetrical Chernoff faces double the number of facial characteristics, thus allowing up to 36 dimensions to be displayed.

The **stick figure** visualization technique maps multidimensional data to five-piece stick figures, where each figure has four limbs and a body. Two dimensions are mapped to the display (x and y) axes and the remaining dimensions are mapped to the angle

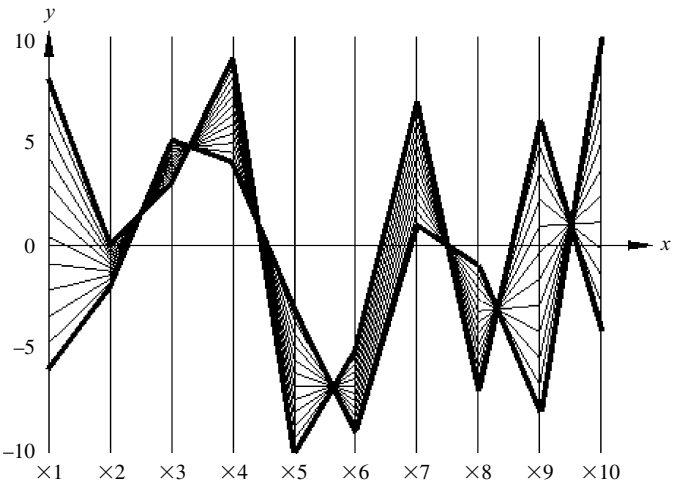


Figure 2.16 Here is a visualization that uses parallel coordinates. *Source: www.stat.columbia.edu/∼cook/movabletype/archives/2007/10/parallel_coordi.thml.*

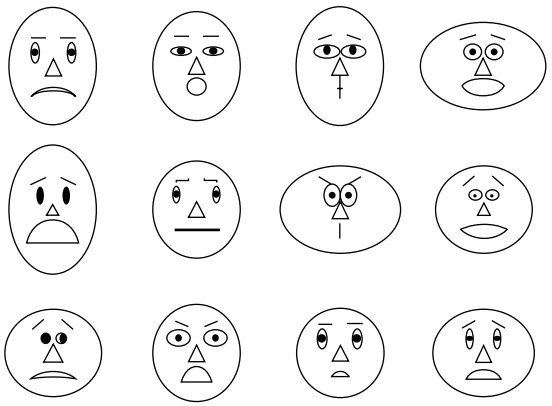


Figure 2.17 Chernoff faces. Each face represents an n-dimensional data point ($n \leq 18$).

and/or length of the limbs. Figure 2.18 shows census data, where *age* and *income* are mapped to the display axes, and the remaining dimensions (*gender, education*, and so on) are mapped to stick figures. If the data items are relatively dense with respect to the two display dimensions, the resulting visualization shows texture patterns, reflecting data trends.

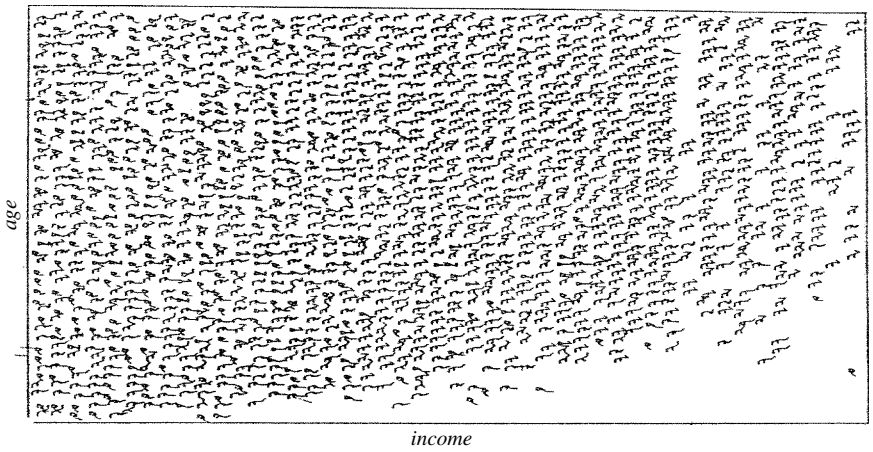


Figure 2.18 Census data represented using stick figures. *Source:* Professor G. Grinstein, Department of Computer Science, University of Massachusetts at Lowell.

2.3.4 Hierarchical Visualization Techniques

The visualization techniques discussed so far focus on visualizing multiple dimensions simultaneously. However, for a large data set of high dimensionality, it would be difficult to visualize all dimensions at the same time. **Hierarchical visualization techniques** partition all dimensions into subsets (i.e., subspaces). The subspaces are visualized in a hierarchical manner.

"Worlds-within-Worlds," also known as *n*-Vision, is a representative hierarchical visualization method. Suppose we want to visualize a 6-D data set, where the dimensions are $F, X_1, \ldots, X_5$. We want to observe how dimension F changes with respect to the other dimensions. We can first fix the values of dimensions X_3, X_4, X_5 to some selected values, say, c_3, c_4, c_5. We can then visualize F, X_1, X_2 using a 3-D plot, called a *world*, as shown in Figure 2.19. The position of the origin of the inner world is located at the point (c_3, c_4, c_5) in the outer world, which is another 3-D plot using dimensions X_3, X_4, X_5. A user can interactively change, in the outer world, the location of the origin of the inner world. The user then views the resulting changes of the inner world. Moreover, a user can vary the dimensions used in the inner world and the outer world. Given more dimensions, more levels of worlds can be used, which is why the method is called "worlds-within-worlds."

As another example of hierarchical visualization methods, **tree-maps** display hierarchical data as a set of nested rectangles. For example, Figure 2.20 shows a tree-map visualizing Google news stories. All news stories are organized into seven categories, each shown in a large rectangle of a unique color. Within each category (i.e., each rectangle at the top level), the news stories are further partitioned into smaller subcategories.

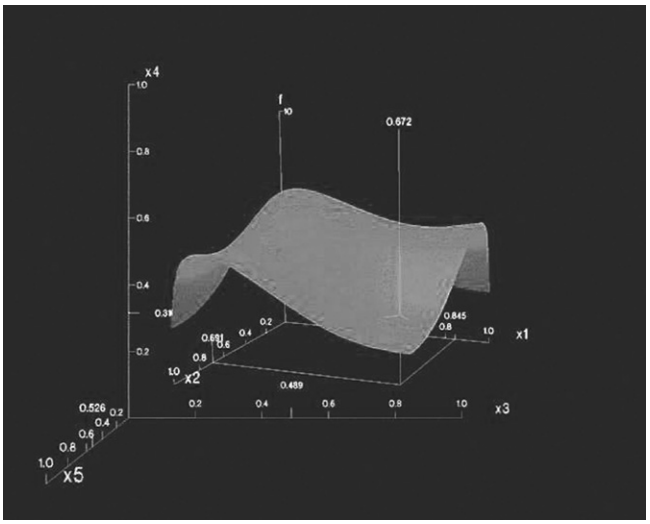


Figure 2.19 "Worlds-within-Worlds" (also known as *n*-Vision). *Source: http://graphics.cs.columbia.edu/projects/AutoVisual/images/1.dipstick.5.gif.*

2.3.5 Visualizing Complex Data and Relations

In early days, visualization techniques were mainly for numeric data. Recently, more and more non-numeric data, such as text and social networks, have become available. Visualizing and analyzing such data attracts a lot of interest.

There are many new visualization techniques dedicated to these kinds of data. For example, many people on the Web tag various objects such as pictures, blog entries, and product reviews. A **tag cloud** is a visualization of statistics of user-generated tags. Often, in a tag cloud, tags are listed alphabetically or in a user-preferred order. The importance of a tag is indicated by font size or color. Figure 2.21 shows a tag cloud for visualizing the popular tags used in a Web site.

Tag clouds are often used in two ways. First, in a tag cloud for a single item, we can use the size of a tag to represent the number of times that the tag is applied to this item by different users. Second, when visualizing the tag statistics on multiple items, we can use the size of a tag to represent the number of items that the tag has been applied to, that is, the popularity of the tag.

In addition to complex data, complex relations among data entries also raise challenges for visualization. For example, Figure 2.22 uses a disease influence graph to visualize the correlations between diseases. The nodes in the graph are diseases, and the size of each node is proportional to the prevalence of the corresponding disease. Two nodes are linked by an edge if the corresponding diseases have a strong correlation. The width of an edge is proportional to the strength of the correlation pattern of the two corresponding diseases.

Figure 2.20 Newsmap: Use of tree-maps to visualize Google news headline stories. *Source: www.cs.umd.edu/class/spring2005/cmsc838s/viz4all/ss/newsmap.png.*

In summary, visualization provides effective tools to explore data. We have introduced several popular methods and the essential ideas behind them. There are many existing tools and methods. Moreover, visualization can be used in data mining in various aspects. In addition to visualizing data, visualization can be used to represent the data mining process, the patterns obtained from a mining method, and user interaction with the data. Visual data mining is an important research and development direction.

2.4 Measuring Data Similarity and Dissimilarity

In data mining applications, such as clustering, outlier analysis, and nearest-neighbor classification, we need ways to assess how alike or unalike objects are in comparison to one another. For example, a store may want to search for clusters of *customer* objects, resulting in groups of customers with similar characteristics (e.g., similar income, area of residence, and age). Such information can then be used for marketing. A **cluster** is

animals architecture art asia australia autumn baby band barcelona beach berlin bike bird birds birthday black blackandwhite blue bw california canada canon car cat chicago china christmas church city clouds color concert cute dance day de dog england europe fall family fashion festival film florida flower flowers food football france friends fun garden geotagged germany girl girls graffiti green halloween hawaii holiday home house india iphone ireland island italia italy japan july kids la lake landscape light live london love macro me mexico model mountain mountains museum music nature new newyork newyorkcity night nikon nyc ocean old paris park party people photo photography photos portrait red river rock san sanfrancisco scotland sea seattle show sky snow spain spring street summer sun sunset taiwan texas thailand tokyo toronto tour travel tree trees trip uk urban usa vacation washington water wedding white winter yellow york zoo

Figure 2.21 Using a tag cloud to visualize popular Web site tags. *Source:* A snapshot of *www.flickr.com/photos/tags/*, January 23, 2010.

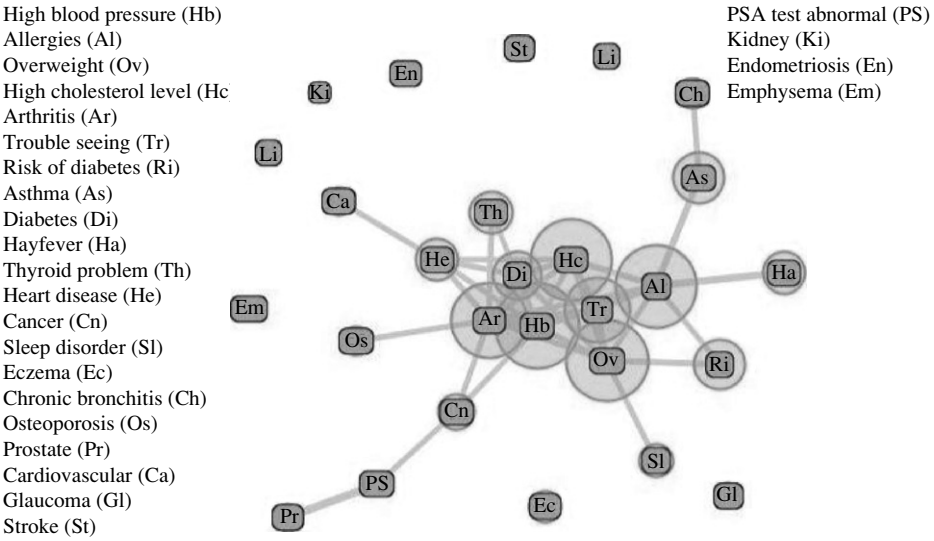


Figure 2.22 Disease influence graph of people at least 20 years old in the NHANES data set.

a collection of data objects such that the objects within a cluster are *similar* to one another and *dissimilar* to the objects in other clusters. Outlier analysis also employs clustering-based techniques to identify potential outliers as objects that are highly dissimilar to others. Knowledge of object similarities can also be used in nearest-neighbor classification schemes where a given object (e.g., a *patient*) is assigned a class label (relating to, say, a *diagnosis*) based on its similarity toward other objects in the model.

This section presents similarity and dissimilarity measures, which are referred to as measures of *proximity*. Similarity and dissimilarity are related. A similarity measure for two objects, i and j, will typically return the value 0 if the objects are unalike. The higher the similarity value, the greater the similarity between objects. (Typically, a value of 1 indicates complete similarity, that is, the objects are identical.) A dissimilarity measure works the opposite way. It returns a value of 0 if the objects are the same (and therefore, far from being dissimilar). The higher the dissimilarity value, the more dissimilar the two objects are.

In Section 2.4.1 we present two data structures that are commonly used in the above types of applications: the *data matrix* (used to store the data objects) and the *dissimilarity matrix* (used to store dissimilarity values for pairs of objects). We also switch to a different notation for data objects than previously used in this chapter since now we are dealing with objects described by more than one attribute. We then discuss how object dissimilarity can be computed for objects described by *nominal* attributes (Section 2.4.2), by *binary* attributes (Section 2.4.3), by *numeric* attributes (Section 2.4.4), by *ordinal* attributes (Section 2.4.5), or by combinations of these attribute types (Section 2.4.6). Section 2.4.7 provides similarity measures for very long and sparse data vectors, such as term-frequency vectors representing documents in information retrieval. Knowing how to compute dissimilarity is useful in studying attributes and will also be referenced in later topics on clustering (Chapters 10 and 11), outlier analysis (Chapter 12), and nearest-neighbor classification (Chapter 9).

2.4.1 Data Matrix versus Dissimilarity Matrix

In Section 2.2, we looked at ways of studying the central tendency, dispersion, and spread of observed values for some attribute X. Our objects there were one-dimensional, that is, described by a single attribute. In this section, we talk about objects described by *multiple* attributes. Therefore, we need a change in notation. Suppose that we have n objects (e.g., persons, items, or courses) described by p attributes (also called *measurements* or *features*, such as age, height, weight, or gender). The objects are $x_1 = (x_{11}, x_{12}, \ldots, x_{1p})$, $x_2 = (x_{21}, x_{22}, \ldots, x_{2p})$, and so on, where x_{ij} is the value for object x_i of the jth attribute. For brevity, we hereafter refer to object x_i as object i. The objects may be tuples in a relational database, and are also referred to as *data samples* or *feature vectors*.

Main memory-based clustering and nearest-neighbor algorithms typically operate on either of the following two data structures:

- **Data matrix** (or *object-by-attribute structure*): This structure stores the n data objects in the form of a relational table, or n-by-p matrix (n objects $\times p$ attributes):

$$\begin{bmatrix} x_{11} & \cdots & x_{1f} & \cdots & x_{1p} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{i1} & \cdots & x_{if} & \cdots & x_{ip} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{n1} & \cdots & x_{nf} & \cdots & x_{np} \end{bmatrix}. \tag{2.8}$$

Each row corresponds to an object. As part of our notation, we may use f to index through the p attributes.

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

$$\begin{bmatrix} 0 & & & & \\ d(2,1) & 0 & & & \\ d(3,1) & d(3,2) & 0 & & \\ \vdots & \vdots & \vdots & & \\ d(n,1) & d(n,2) & \cdots & \cdots & 0 \end{bmatrix}, \tag{2.9}$$

where $d(i, j)$ is the measured **dissimilarity** or "difference" between objects i and j. In general, $d(i, j)$ is a non-negative 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. Note that $d(i, i) = 0$; that is, the difference between an object and itself is 0. Furthermore, $d(i, j) = d(j, i)$. (For readability, we do not show the $d(j, i)$ entries; the matrix is symmetric.) Measures of dissimilarity are discussed throughout the remainder of this chapter.

Measures of similarity can often be expressed as a function of measures of dissimilarity. For example, for nominal data,

$$sim(i, j) = 1 - d(i, j), \tag{2.10}$$

where $sim(i, j)$ is the similarity between objects i and j. Throughout the rest of this chapter, we will also comment on measures of similarity.

A data matrix is made up of two entities or "things," namely rows (for objects) and columns (for attributes). Therefore, the data matrix is often called a **two-mode** matrix. The dissimilarity matrix contains one kind of entity (dissimilarities) and so is called a **one-mode** matrix. Many clustering and nearest-neighbor algorithms operate on a dissimilarity matrix. Data in the form of a data matrix can be transformed into a dissimilarity matrix before applying such algorithms.

2.4.2 Proximity Measures for Nominal Attributes

A nominal attribute can take on two or more states (Section 2.1.2). For example, *map_color* is a nominal attribute that may have, say, five states: *red, yellow, green, pink*, and *blue*.

Let the number of states of a nominal attribute 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 nominal attributes?" The dissimilarity between two objects i and j can be computed based on the ratio of mismatches:

$$d(i, j) = \frac{p - m}{p}, \tag{2.11}$$

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

Example 2.17 **Dissimilarity between nominal attributes.** Suppose that we have the sample data of Table 2.2, except that only the *object-identifier* and the attribute *test-1* are available, where *test-1* is nominal. (We will use *test-2* and *test-3* in later examples.) Let's compute the dissimilarity matrix (Eq. 2.9), that is,

$$\begin{bmatrix} 0 & & & \\ d(2,1) & 0 & & \\ d(3,1) & d(3,2) & 0 & \\ d(4,1) & d(4,2) & d(4,3) & 0 \end{bmatrix}.$$

Since here we have one nominal attribute, *test-1*, we set $p = 1$ in Eq. (2.11) so that $d(i, j)$ evaluates to 0 if objects i and j match, and 1 if the objects differ. Thus, we get

$$\begin{bmatrix} 0 & & & \\ 1 & 0 & & \\ 1 & 1 & 0 & \\ 0 & 1 & 1 & 0 \end{bmatrix}.$$

From this, we see that all objects are dissimilar except objects 1 and 4 (i.e., $d(4,1) = 0$). ■

Table 2.2 A Sample Data Table Containing Attributes of Mixed Type

Object Identifier	*test-1 (nominal)*	*test-2 (ordinal)*	*test-3 (numeric)*
1	code A	excellent	45
2	code B	fair	22
3	code C	good	64
4	code A	excellent	28

Alternatively, similarity can be computed as

$$sim(i, j) = 1 - d(i, j) = \frac{m}{p}. \tag{2.12}$$

Proximity between objects described by nominal attributes can be computed using an alternative encoding scheme. Nominal attributes can be encoded using asymmetric binary attributes by creating a new binary attribute for each of the M states. For an object with a given state value, the binary attribute representing that state is set to 1, while the remaining binary attributes are set to 0. For example, to encode the nominal attribute *map_color*, a binary attribute can be created for each of the five colors previously listed. For an object having the color *yellow*, the *yellow* attribute is set to 1, while the remaining four attributes are set to 0. Proximity measures for this form of encoding can be calculated using the methods discussed in the next subsection.

2.4.3 Proximity Measures for Binary Attributes

Let's look at dissimilarity and similarity measures for objects described by either *symmetric* or *asymmetric binary attributes*.

Recall that a binary attribute has only one of two states: 0 and 1, where 0 means that the attribute is absent, and 1 means that it is present (Section 2.1.3). Given the attribute *smoker* describing a patient, for instance, 1 indicates that the patient smokes, while 0 indicates that the patient does not. Treating binary attributes as if they are numeric can be misleading. Therefore, methods specific to binary data are necessary for computing dissimilarity.

"*So, how can we compute the dissimilarity between two binary attributes?*" One approach involves computing a dissimilarity matrix from the given binary data. If all binary attributes are thought of as having the same weight, we have the 2×2 contingency table of Table 2.3, where q is the number of attributes that equal 1 for both objects i and j, r is the number of attributes that equal 1 for object i but equal 0 for object j, s is the number of attributes that equal 0 for object i but equal 1 for object j, and t is the number of attributes that equal 0 for both objects i and j. The total number of attributes is p, where $p = q + r + s + t$.

Recall that for symmetric binary attributes, each state is equally valuable. Dissimilarity that is based on symmetric binary attributes is called **symmetric binary dissimilarity**. If objects i and j are described by symmetric binary attributes, then the

Table 2.3 Contingency Table for Binary Attributes

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

dissimilarity between i and j is

$$d(i, j) = \frac{r + s}{q + r + s + t}. \tag{2.13}$$

For asymmetric binary attributes, the two states are not equally important, such as the *positive* (1) and *negative* (0) outcomes of a disease test. Given two asymmetric binary attributes, the agreement of two 1s (a positive match) is then considered more significant than that of two 0s (a negative match). Therefore, such binary attributes are often considered "monary" (having one state). The dissimilarity based on these attributes is called **asymmetric binary dissimilarity**, where the number of negative matches, t, is considered unimportant and is thus ignored in the following computation:

$$d(i, j) = \frac{r + s}{q + r + s}. \tag{2.14}$$

Complementarily, we can measure the difference between two binary attributes based on the notion of similarity instead of dissimilarity. For example, the **asymmetric binary similarity** between the objects i and j can be computed as

$$sim(i, j) = \frac{q}{q + r + s} = 1 - d(i, j). \tag{2.15}$$

The coefficient $sim(i, j)$ of Eq. (2.15) is called the **Jaccard coefficient** and is popularly referenced in the literature.

When both symmetric and asymmetric binary attributes occur in the same data set, the mixed attributes approach described in Section 2.4.6 can be applied.

Example 2.18 **Dissimilarity between binary attributes.** Suppose that a patient record table (Table 2.4) contains the attributes *name, gender, fever, cough, test-1, test-2, test-3*, and *test-4*, where *name* is an object identifier, *gender* is a symmetric attribute, and the remaining attributes are asymmetric binary.

For asymmetric attribute values, let the values Y (*yes*) and P (*positive*) be set to 1, and the value N (*no* or *negative*) be set to 0. Suppose that the distance between objects

Table 2.4 Relational Table Where Patients Are Described by Binary Attributes

name	*gender*	*fever*	*cough*	*test-1*	*test-2*	*test-3*	*test-4*
Jack	M	Y	N	P	N	N	N
Jim	M	Y	Y	N	N	N	N
Mary	F	Y	N	P	N	P	N
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

(patients) is computed based only on the asymmetric attributes. According to Eq. (2.14), the distance between each pair of the three patients—Jack, Mary, and Jim—is

$$d(Jack, Jim) = \frac{1+1}{1+1+1} = 0.67,$$

$$d(Jack, Mary) = \frac{0+1}{2+0+1} = 0.33,$$

$$d(Jim, Mary) = \frac{1+2}{1+1+2} = 0.75.$$

These measurements suggest that Jim and Mary are unlikely to have a similar disease because they have the highest dissimilarity value among the three pairs. Of the three patients, Jack and Mary are the most likely to have a similar disease. ■

2.4.4 Dissimilarity of Numeric Data: Minkowski Distance

In this section, we describe distance measures that are commonly used for computing the dissimilarity of objects described by numeric attributes. These measures include the *Euclidean, Manhattan*, and *Minkowski distances.*

In some cases, the data are normalized before applying distance calculations. This involves transforming the data to fall within a smaller or common range, such as $[-1, 1]$ or $[0.0, 1.0]$. Consider a *height* attribute, for example, which could be measured in either meters or inches. In general, expressing an attribute in smaller units will lead to a larger range for that attribute, and thus tend to give such attributes greater effect or "weight." Normalizing the data attempts to give all attributes an equal weight. It may or may not be useful in a particular application. Methods for normalizing data are discussed in detail in Chapter 3 on data preprocessing.

The most popular distance measure is **Euclidean distance** (i.e., straight line or "as the crow flies"). Let $i = (x_{i1}, x_{i2}, \ldots, x_{ip})$ and $j = (x_{j1}, x_{j2}, \ldots, x_{jp})$ be two objects described by p numeric attributes. The Euclidean distance between objects i and j is defined as

$$d(i, j) = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \cdots + (x_{ip} - x_{jp})^2}. \tag{2.16}$$

Another well-known measure is the **Manhattan (or city block) distance**, named so because it is the distance in blocks between any two points in a city (such as 2 blocks down and 3 blocks over for a total of 5 blocks). It is defined as

$$d(i, j) = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + \cdots + |x_{ip} - x_{jp}|. \tag{2.17}$$

Both the Euclidean and the Manhattan distance satisfy the following mathematical properties:

Non-negativity: $d(i, j) \geq 0$: Distance is a non-negative number.

Identity of indiscernibles: $d(i, i) = 0$: The distance of an object to itself is 0.

Symmetry: $d(i, j) = d(j, i)$: Distance is a symmetric function.

Triangle inequality: $d(i, j) \leq d(i, k) + d(k, j)$: Going directly from object i to object j in space is no more than making a detour over any other object k.

A measure that satisfies these conditions is known as **metric**. Please note that the non-negativity property is implied by the other three properties.

Example 2.19 Euclidean distance and Manhattan distance. Let $\boldsymbol{x_1} = (1, 2)$ and $\boldsymbol{x_2} = (3, 5)$ represent two objects as shown in Figure 2.23. The Euclidean distance between the two is $\sqrt{2^2 + 3^2} = 3.61$. The Manhattan distance between the two is $2 + 3 = 5$. ■

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

$$d(i, j) = \sqrt[h]{|x_{i1} - x_{j1}|^h + |x_{i2} - x_{j2}|^h + \cdots + |x_{ip} - x_{jp}|^h}, \tag{2.18}$$

where h is a real number such that $h \geq 1$. (Such a distance is also called L_p **norm** in some literature, where the symbol p refers to our notation of h. We have kept p as the number of attributes to be consistent with the rest of this chapter.) It represents the Manhattan distance when $h = 1$ (i.e., L_1 norm) and Euclidean distance when $h = 2$ (i.e., L_2 norm).

The **supremum distance** (also referred to as $\boldsymbol{L_{max}}$, $\boldsymbol{L_\infty}$ **norm** and as the **Chebyshev distance**) is a generalization of the Minkowski distance for $h \to \infty$. To compute it, we find the attribute f that gives the maximum difference in values between the two objects. This difference is the supremum distance, defined more formally as:

$$d(i, j) = \lim_{h \to \infty} \left(\sum_{f=1}^{p} |x_{if} - x_{jf}|^h \right)^{\frac{1}{h}} = \max_{f}^{p} |x_{if} - x_{jf}|. \tag{2.19}$$

The L^∞ norm is also known as the *uniform norm*.

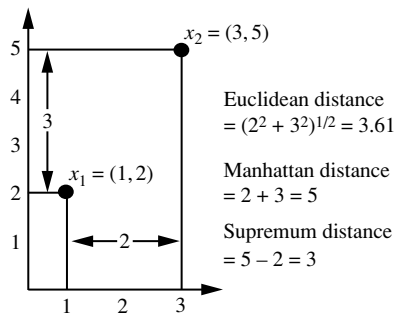


Figure 2.23 Euclidean, Manhattan, and supremum distances between two objects.

Example 2.20 **Supremum distance.** Let's use the same two objects, $\boldsymbol{x_1} = (1, 2)$ and $\boldsymbol{x_2} = (3, 5)$, as in Figure 2.23. The second attribute gives the greatest difference between values for the objects, which is $5 - 2 = 3$. This is the supremum distance between both objects. ■

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

$$d(i, j) = \sqrt{w_1|x_{i1} - x_{j1}|^2 + w_2|x_{i2} - x_{j2}|^2 + \cdots + w_m|x_{ip} - x_{jp}|^2}. \tag{2.20}$$

Weighting can also be applied to other distance measures as well.

2.4.5 Proximity Measures for Ordinal Attributes

The values of an ordinal attribute have a meaningful order or ranking about them, yet the magnitude between successive values is unknown (Section 2.1.4). An example includes the sequence *small*, *medium*, *large* for a *size* attribute. Ordinal attributes may also be obtained from the discretization of numeric attributes by splitting the value range into a finite number of categories. These categories are organized into ranks. That is, the range of a numeric attribute can be mapped to an ordinal attribute f having M_f states. For example, the range of the interval-scaled attribute *temperature* (in Celsius) can be organized into the following states: -30 to -10, -10 to 10, 10 to 30, representing the categories *cold temperature*, *moderate temperature*, and *warm temperature*, respectively. Let M represent the number of possible states that an ordinal attribute can have. These ordered states define the ranking $1, \ldots, M_f$.

"*How are ordinal attributes handled?*" The treatment of ordinal attributes is quite similar to that of numeric attributes when computing dissimilarity between objects. Suppose that f is an attribute from a set of ordinal attributes 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 attribute can have a different number of states, it is often necessary to map the range of each attribute onto [0.0, 1.0] so that each attribute has equal weight. We perform such data normalization by replacing the rank r_{if} of the ith object in the fth attribute by

$$z_{if} = \frac{r_{if} - 1}{M_f - 1}. \tag{2.21}$$

3. Dissimilarity can then be computed using any of the distance measures described in Section 2.4.4 for numeric attributes, using z_{if} to represent the f value for the ith object.

Example 2.21 **Dissimilarity between ordinal attributes.** Suppose that we have the sample data shown earlier in Table 2.2, except that this time only the *object-identifier* and the continuous ordinal attribute, *test-2*, are available. There are three states for *test-2*: *fair*, *good*, and *excellent*, that is, $M_f = 3$. For step 1, if we replace each value for *test-2* by its rank, the four objects are assigned the ranks 3, 1, 2, and 3, respectively. Step 2 normalizes the ranking by mapping rank 1 to 0.0, rank 2 to 0.5, and rank 3 to 1.0. For step 3, we can use, say, the Euclidean distance (Eq. 2.16), which results in the following dissimilarity matrix:

$$\begin{bmatrix} 0 & & & \\ 1.0 & 0 & & \\ 0.5 & 0.5 & 0 & \\ 0 & 1.0 & 0.5 & 0 \end{bmatrix}.$$

Therefore, objects 1 and 2 are the most dissimilar, as are objects 2 and 4 (i.e., $d(2,1) = 1.0$ and $d(4,2) = 1.0$). This makes intuitive sense since objects 1 and 4 are both *excellent*. Object 2 is *fair*, which is at the opposite end of the range of values for *test-2*. ■

Similarity values for ordinal attributes can be interpreted from dissimilarity as $sim(i,j) = 1 - d(i,j)$.

2.4.6 Dissimilarity for Attributes of Mixed Types

Sections 2.4.2 through 2.4.5 discussed how to compute the dissimilarity between objects described by attributes of the same type, where these types may be either *nominal*, *symmetric binary*, *asymmetric binary*, *numeric*, or *ordinal*. However, in many real databases, objects are described by a *mixture* of attribute types. In general, a database can contain all of these attribute types.

"So, how can we compute the dissimilarity between objects of mixed attribute types?" One approach is to group each type of attribute together, performing separate data mining (e.g., clustering) analysis for each type. This is feasible if these analyses derive compatible results. However, in real applications, it is unlikely that a separate analysis per attribute type will generate compatible results.

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

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

$$d(i,j) = \frac{\sum_{f=1}^{p} \delta_{ij}^{(f)} d_{ij}^{(f)}}{\sum_{f=1}^{p} \delta_{ij}^{(f)}}, \tag{2.22}$$

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

- If f is numeric: $d_{ij}^{(f)} = \frac{|x_{if} - x_{jf}|}{max_h x_{hf} - min_h x_{hf}}$, where h runs over all nonmissing objects for attribute f.
- If f is nominal or binary: $d_{ij}^{(f)} = 0$ 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 numeric.

These steps are identical to what we have already seen for each of the individual attribute types. The only difference is for numeric attributes, where 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 attributes describing the objects are of different types.

Example 2.22 **Dissimilarity between attributes of mixed type.** Let's compute a dissimilarity matrix for the objects in Table 2.2. Now we will consider *all* of the attributes, which are of different types. In Examples 2.17 and 2.21, we worked out the dissimilarity matrices for each of the individual attributes. The procedures we followed for *test-1* (which is nominal) and *test-2* (which is ordinal) are the same as outlined earlier for processing attributes of mixed types. Therefore, we can use the dissimilarity matrices obtained for *test-1* and *test-2* later when we compute Eq. (2.22). First, however, we need to compute the dissimilarity matrix for the third attribute, *test-3* (which is numeric). That is, we must compute $d_{ij}^{(3)}$. Following the case for numeric attributes, we let $max_h x_h = 64$ and $min_h x_h = 22$. The difference between the two is used in Eq. (2.22) to normalize the values of the dissimilarity matrix. The resulting dissimilarity matrix for *test-3* is

$$\begin{bmatrix} 0 & & & \\ 0.55 & 0 & & \\ 0.45 & 1.00 & 0 & \\ 0.40 & 0.14 & 0.86 & 0 \end{bmatrix}.$$

We can now use the dissimilarity matrices for the three attributes in our computation of Eq. (2.22). The indicator $\delta_{ij}^{(f)} = 1$ for each of the three attributes, f. We get, for example, $d(3, 1) = \frac{1(1) + 1(0.50) + 1(0.45)}{3} = 0.65$. The resulting dissimilarity matrix obtained for the

data described by the three attributes of mixed types is:

$$\begin{bmatrix} 0 & & & \\ 0.85 & 0 & & \\ 0.65 & 0.83 & 0 & \\ 0.13 & 0.71 & 0.79 & 0 \end{bmatrix}.$$

From Table 2.2, we can intuitively guess that objects 1 and 4 are the most similar, based on their values for *test*-1 and *test*-2. This is confirmed by the dissimilarity matrix, where $d(4, 1)$ is the lowest value for any pair of different objects. Similarly, the matrix indicates that objects 1 and 2 are the least similar. ■

2.4.7 Cosine Similarity

Cosine similarity measures the similarity between two vectors of an inner product space. It is measured by the cosine of the angle between two vectors and determines whether two vectors are pointing in roughly the same direction. It is often used to measure document similarity in text analysis.

A document can be represented by thousands of attributes, each recording the frequency of a particular word (such as a keyword) or phrase in the document. Thus, each document is an object represented by what is called a *term-frequency vector*. For example, in Table 2.5, we see that *Document1* contains five instances of the word *team*, while *hockey* occurs three times. The word *coach* is absent from the entire document, as indicated by a count value of 0. Such data can be highly asymmetric.

Term-frequency vectors are typically very long and **sparse** (i.e., they have many 0 values). Applications using such structures include information retrieval, text document clustering, biological taxonomy, and gene feature mapping. The traditional distance measures that we have studied in this chapter do not work well for such sparse numeric data. For example, two term-frequency vectors may have many 0 values in common, meaning that the corresponding documents do not share many words, but this does not make them similar. We need a measure that will focus on the words that the two documents *do* have in common, and the occurrence frequency of such words. In other words, we need a measure for numeric data that ignores zero-matches.

Table 2.5 Document Vector or Term-Frequency Vector

Document	*team*	*coach*	*hockey*	*baseball*	*soccer*	*penalty*	*score*	*win*	*loss*	*season*
Document1	5	0	3	0	2	0	0	2	0	0
Document2	3	0	2	0	1	1	0	1	0	1
Document3	0	7	0	2	1	0	0	3	0	0
Document4	0	1	0	0	1	2	2	0	3	0

Cosine similarity is a measure of similarity that can be used to compare documents or, say, give a ranking of documents with respect to a given vector of query words. Let $\boldsymbol{x}$ and $\boldsymbol{y}$ be two vectors for comparison. Using the cosine measure as a similarity function, we have

$$sim(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{||\boldsymbol{x}||||\boldsymbol{y}||}, \tag{2.23}$$

where $||\boldsymbol{x}||$ is the Euclidean norm of vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_p)$, defined as $\sqrt{x_1^2 + x_2^2 + \cdots + x_p^2}$. Conceptually, it is the length of the vector. Similarly, $||\boldsymbol{y}||$ is the Euclidean norm of vector $\boldsymbol{y}$. The measure computes the cosine of the angle between vectors $\boldsymbol{x}$ and $\boldsymbol{y}$. A cosine value of 0 means that the two vectors are at 90 degrees to each other (orthogonal) and have no match. The closer the cosine value to 1, the smaller the angle and the greater the match between vectors. Note that because the cosine similarity measure does not obey all of the properties of Section 2.4.4 defining metric measures, it is referred to as a *nonmetric measure*.

Example 2.23 **Cosine similarity between two term-frequency vectors.** Suppose that $\boldsymbol{x}$ and $\boldsymbol{y}$ are the first two term-frequency vectors in Table 2.5. That is, $\boldsymbol{x} = (5, 0, 3, 0, 2, 0, 0, 2, 0, 0)$ and $\boldsymbol{y} = (3, 0, 2, 0, 1, 1, 0, 1, 0, 1)$. How similar are $\boldsymbol{x}$ and $\boldsymbol{y}$? Using Eq. (2.23) to compute the cosine similarity between the two vectors, we get:

$$\begin{aligned}
\boldsymbol{x}^t \cdot \boldsymbol{y} &= 5 \times 3 + 0 \times 0 + 3 \times 2 + 0 \times 0 + 2 \times 1 + 0 \times 1 + 0 \times 0 + 2 \times 1 \\
&\quad + 0 \times 0 + 0 \times 1 = 25 \\
||\boldsymbol{x}|| &= \sqrt{5^2 + 0^2 + 3^2 + 0^2 + 2^2 + 0^2 + 0^2 + 2^2 + 0^2 + 0^2} = 6.48 \\
||\boldsymbol{y}|| &= \sqrt{3^2 + 0^2 + 2^2 + 0^2 + 1^2 + 1^2 + 0^2 + 1^2 + 0^2 + 1^2} = 4.12 \\
sim(\boldsymbol{x}, \boldsymbol{y}) &= 0.94
\end{aligned}$$

Therefore, if we were using the cosine similarity measure to compare these documents, they would be considered quite similar. ■

When attributes are binary-valued, the cosine similarity function can be interpreted in terms of shared features or 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 (i.e., shared) 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, $sim(\boldsymbol{x}, \boldsymbol{y})$ is a measure of relative possession of common attributes.

A simple variation of cosine similarity for the preceding scenario is

$$sim(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{\boldsymbol{x} \cdot \boldsymbol{x} + \boldsymbol{y} \cdot \boldsymbol{y} - \boldsymbol{x} \cdot \boldsymbol{y}}, \tag{2.24}$$

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.

2.5 Summary

- Data sets are made up of data objects. A **data object** represents an entity. Data objects are described by attributes. Attributes can be nominal, binary, ordinal, or numeric.
- The values of a **nominal** (or **categorical**) **attribute** are symbols or names of things, where each value represents some kind of category, code, or state.
- **Binary attributes** are nominal attributes with only two possible states (such as 1 and 0 or true and false). If the two states are equally important, the attribute is *symmetric*; otherwise it is *asymmetric*.
- An **ordinal attribute** is an attribute with possible values that have a meaningful order or ranking among them, but the magnitude between successive values is not known.
- A **numeric attribute** is *quantitative* (i.e., it is a measurable quantity) represented in integer or real values. Numeric attribute types can be *interval-scaled* or *ratio-scaled*. The values of an **interval-scaled attribute** are measured in fixed and equal units. **Ratio-scaled attributes** are numeric attributes with an inherent zero-point. Measurements are ratio-scaled in that we can speak of values as being an order of magnitude larger than the unit of measurement.
- **Basic statistical descriptions** provide the analytical foundation for data preprocessing. The basic statistical measures for data summarization include *mean, weighted mean, median*, and *mode* for measuring the central tendency of data; and *range, quantiles, quartiles, interquartile range, variance*, and *standard deviation* for measuring the dispersion of data. Graphical representations (e.g., *boxplots, quantile plots, quantile–quantile plots, histograms*, and *scatter plots*) facilitate visual inspection of the data and are thus useful for data preprocessing and mining.
- **Data visualization** techniques may be *pixel-oriented, geometric-based, icon-based*, or *hierarchical*. These methods apply to multidimensional relational data. Additional techniques have been proposed for the visualization of complex data, such as text and social networks.
- Measures of object **similarity** and **dissimilarity** are used in data mining applications such as clustering, outlier analysis, and nearest-neighbor classification. Such measures of *proximity* can be computed for each attribute type studied in this chapter, or for combinations of such attributes. Examples include the *Jaccard coefficient* for asymmetric binary attributes and *Euclidean*, *Manhattan*, *Minkowski*, and *supremum* distances for numeric attributes. For applications involving sparse numeric data vectors, such as term-frequency vectors, the *cosine measure* and the *Tanimoto coefficient* are often used in the assessment of similarity.

2.6 Exercises

2.1 Give three additional commonly used statistical measures that are not already illustrated in this chapter for the characterization of *data dispersion*. Discuss how they can be computed efficiently in large databases.

2.2 Suppose that the data for analysis includes the attribute *age*. The *age* values for the data tuples are (in increasing order) 13, 15, 16, 16, 19, 20, 20, 21, 22, 22, 25, 25, 25, 25, 30, 33, 33, 35, 35, 35, 35, 36, 40, 45, 46, 52, 70.

(a) What is the *mean* of the data? What is the *median*?

(b) What is the *mode* of the data? Comment on the data's modality (i.e., bimodal, trimodal, etc.).

(c) What is the *midrange* of the data?

(d) Can you find (roughly) the first quartile (Q_1) and the third quartile (Q_3) of the data?

(e) Give the *five-number summary* of the data.

(f) Show a *boxplot* of the data.

(g) How is a *quantile–quantile plot* different from a *quantile plot*?

2.3 Suppose that the values for a given set of data are grouped into intervals. The intervals and corresponding frequencies are as follows:

age	*frequency*
1–5	200
6–15	450
16–20	300
21–50	1500
51–80	700
81–110	44

Compute an *approximate median* value for the data.

2.4 Suppose that a hospital tested the age and body fat data for 18 randomly selected adults with the following results:

age	23	23	27	27	39	41	47	49	50
%fat	9.5	26.5	7.8	17.8	31.4	25.9	27.4	27.2	31.2
age	52	54	54	56	57	58	58	60	61
%fat	34.6	42.5	28.8	33.4	30.2	34.1	32.9	41.2	35.7

(a) Calculate the mean, median, and standard deviation of *age* and *%fat*.

(b) Draw the boxplots for *age* and *%fat*.

(c) Draw a *scatter plot* and a *q-q plot* based on these two variables.

2.5 Briefly outline how to compute the dissimilarity between objects described by the following:

(a) Nominal attributes

(b) Asymmetric binary attributes

(c) Numeric attributes
(d) Term-frequency vectors

2.6 Given two objects represented by the tuples (22, 1, 42, 10) and (20, 0, 36, 8):

(a) Compute the *Euclidean distance* between the two objects.
(b) Compute the *Manhattan distance* between the two objects.
(c) Compute the *Minkowski distance* between the two objects, using $q = 3$.
(d) Compute the *supremum distance* between the two objects.

2.7 The *median* is one of the most important holistic measures in data analysis. Propose several methods for median approximation. Analyze their respective complexity under different parameter settings and decide to what extent the real value can be approximated. Moreover, suggest a heuristic strategy to balance between accuracy and complexity and then apply it to all methods you have given.

2.8 It is important to define or select similarity measures in data analysis. However, there is no commonly accepted subjective similarity measure. Results can vary depending on the similarity measures used. Nonetheless, seemingly different similarity measures may be equivalent after some transformation.

Suppose we have the following 2-D data set:

	A_1	A_2
$\boldsymbol{x_1}$	1.5	1.7
$\boldsymbol{x_2}$	2	1.9
$\boldsymbol{x_3}$	1.6	1.8
$\boldsymbol{x_4}$	1.2	1.5
$\boldsymbol{x_5}$	1.5	1.0

(a) Consider the data as 2-D data points. Given a new data point, $\boldsymbol{x} = (1.4, 1.6)$ as a query, rank the database points based on similarity with the query using Euclidean distance, Manhattan distance, supremum distance, and cosine similarity.
(b) Normalize the data set to make the norm of each data point equal to 1. Use Euclidean distance on the transformed data to rank the data points.

2.7 Bibliographic Notes

Methods for descriptive data summarization have been studied in the statistics literature long before the onset of computers. Good summaries of statistical descriptive data mining methods include Freedman, Pisani, and Purves [FPP07] and Devore [Dev95]. For

statistics-based visualization of data using boxplots, quantile plots, quantile–quantile plots, scatter plots, and loess curves, see Cleveland [Cle93].

Pioneering work on data visualization techniques is described in *The Visual Display of Quantitative Information* [Tuf83], *Envisioning Information* [Tuf90], and *Visual Explanations: Images and Quantities, Evidence and Narrative* [Tuf97], all by Tufte, in addition to *Graphics and Graphic Information Processing* by Bertin [Ber81], *Visualizing Data* by Cleveland [Cle93], and *Information Visualization in Data Mining and Knowledge Discovery* edited by Fayyad, Grinstein, and Wierse [FGW01].

Major conferences and symposiums on visualization include *ACM Human Factors in Computing Systems (CHI)*, *Visualization*, and the *International Symposium on Information Visualization*. Research on visualization is also published in *Transactions on Visualization and Computer Graphics*, *Journal of Computational and Graphical Statistics*, and *IEEE Computer Graphics and Applications*.

Many graphical user interfaces and visualization tools have been developed and can be found in various data mining products. Several books on data mining (e.g., *Data Mining Solutions* by Westphal and Blaxton [WB98]) present many good examples and visual snapshots. For a survey of visualization techniques, see "Visual techniques for exploring databases" by Keim [Kei97].

Similarity and distance measures among various variables have been introduced in many textbooks that study cluster analysis, including Hartigan [Har75]; Jain and Dubes [JD88]; Kaufman and Rousseeuw [KR90]; and Arabie, Hubert, and de Soete [AHS96]. Methods for combining attributes of different types into a single dissimilarity matrix were introduced by Kaufman and Rousseeuw [KR90].

3 Data Preprocessing

Today's real-world databases are highly susceptible to noisy, missing, and inconsistent data due to their typically huge size (often several gigabytes or more) and their likely origin from multiple, heterogenous sources. Low-quality data will lead to low-quality mining results. "*How can the data be preprocessed in order to help improve the quality of the data and, consequently, of the mining results? How can the data be preprocessed so as to improve the efficiency and ease of the mining process?*"

There are several data preprocessing techniques. *Data cleaning* can be applied to remove noise and correct inconsistencies in data. *Data integration* merges data from multiple sources into a coherent data store such as a data warehouse. *Data reduction* can reduce data size by, for instance, aggregating, eliminating redundant features, or clustering. *Data transformations* (e.g., normalization) may be applied, where data are scaled to fall within a smaller range like 0.0 to 1.0. This can improve the accuracy and efficiency of mining algorithms involving distance measurements. These techniques are not mutually exclusive; they may work together. For example, data cleaning can involve transformations to correct wrong data, such as by transforming all entries for a *date* field to a common format.

In Chapter 2, we learned about the different attribute types and how to use basic statistical descriptions to study data characteristics. These can help identify erroneous values and outliers, which will be useful in the data cleaning and integration steps. Data processing techniques, when applied before mining, can substantially improve the overall quality of the patterns mined and/or the time required for the actual mining.

In this chapter, we introduce the basic concepts of data preprocessing in Section 3.1. The methods for data preprocessing are organized into the following categories: data cleaning (Section 3.2), data integration (Section 3.3), data reduction (Section 3.4), and data transformation (Section 3.5).

3.1 Data Preprocessing: An Overview

This section presents an overview of data preprocessing. Section 3.1.1 illustrates the many elements defining data quality. This provides the incentive behind data preprocessing. Section 3.1.2 outlines the major tasks in data preprocessing.

3.1.1 Data Quality: Why Preprocess the Data?

Data have quality if they satisfy the requirements of the intended use. There are many factors comprising **data quality**, including *accuracy, completeness, consistency, timeliness, believability*, and *interpretability*.

Imagine that you are a manager at *AllElectronics* and have been charged with analyzing the company's data with respect to your branch's sales. You immediately set out to perform this task. You carefully inspect the company's database and data warehouse, identifying and selecting the attributes or dimensions (e.g., *item*, *price*, and *units_sold*) to be included in your analysis. Alas! You notice that several of the attributes for various tuples have no recorded value. For your analysis, you would like to include information as to whether each item purchased was advertised as on sale, yet you discover that this information has not been recorded. Furthermore, users of your database system have reported errors, unusual values, and inconsistencies in the data recorded for some transactions. In other words, the data you wish to analyze by data mining techniques are *incomplete* (lacking attribute values or certain attributes of interest, or containing only aggregate data); *inaccurate* or *noisy* (containing errors, or values that deviate from the expected); and *inconsistent* (e.g., containing discrepancies in the department codes used to categorize items). Welcome to the real world!

This scenario illustrates three of the elements defining data quality: **accuracy**, **completeness**, and **consistency**. Inaccurate, incomplete, and inconsistent data are commonplace properties of large real-world databases and data warehouses. There are many possible reasons for inaccurate data (i.e., having incorrect attribute values). The data collection instruments used may be faulty. There may have been human or computer errors occurring at data entry. Users may purposely submit incorrect data values for mandatory fields when they do not wish to submit personal information (e.g., by choosing the default value "January 1" displayed for birthday). This is known as *disguised missing data*. Errors in data transmission can also occur. There may be technology limitations such as limited buffer size for coordinating synchronized data transfer and consumption. Incorrect data may also result from inconsistencies in naming conventions or data codes, or inconsistent formats for input fields (e.g., *date*). Duplicate tuples also require data cleaning.

Incomplete data can occur for a number of reasons. Attributes of interest may not always be available, such as customer information for sales transaction data. Other data may not be included simply because they were not considered important at the time of entry. Relevant data may not be recorded due to a misunderstanding or because of equipment malfunctions. Data that were inconsistent with other recorded data may

have been deleted. Furthermore, the recording of the data history or modifications may have been overlooked. Missing data, particularly for tuples with missing values for some attributes, may need to be inferred.

Recall that data quality depends on the intended use of the data. Two different users may have very different assessments of the quality of a given database. For example, a marketing analyst may need to access the database mentioned before for a list of customer addresses. Some of the addresses are outdated or incorrect, yet overall, 80% of the addresses are accurate. The marketing analyst considers this to be a large customer database for target marketing purposes and is pleased with the database's accuracy, although, as sales manager, you found the data inaccurate.

Timeliness also affects data quality. Suppose that you are overseeing the distribution of monthly sales bonuses to the top sales representatives at *AllElectronics*. Several sales representatives, however, fail to submit their sales records on time at the end of the month. There are also a number of corrections and adjustments that flow in after the month's end. For a period of time following each month, the data stored in the database are incomplete. However, once all of the data are received, it is correct. The fact that the month-end data are not updated in a timely fashion has a negative impact on the data quality.

Two other factors affecting data quality are believability and interpretability. **Believability** reflects how much the data are trusted by users, while **interpretability** reflects how easy the data are understood. Suppose that a database, at one point, had several errors, all of which have since been corrected. The past errors, however, had caused many problems for sales department users, and so they no longer trust the data. The data also use many accounting codes, which the sales department does not know how to interpret. Even though the database is now accurate, complete, consistent, and timely, sales department users may regard it as of low quality due to poor believability and interpretability.

3.1.2 Major Tasks in Data Preprocessing

In this section, we look at the major steps involved in data preprocessing, namely, data cleaning, data integration, data reduction, and data transformation.

Data cleaning routines work to "clean" the data by filling in missing values, smoothing noisy data, identifying or removing outliers, and resolving inconsistencies. If users believe the data are dirty, they are unlikely to trust the results of any data mining that has been applied. Furthermore, dirty data can cause confusion for the mining procedure, resulting in unreliable output. Although most mining routines have some procedures for dealing with incomplete or noisy data, they are not always robust. Instead, they may concentrate on avoiding overfitting the data to the function being modeled. Therefore, a useful preprocessing step is to run your data through some data cleaning routines. Section 3.2 discusses methods for data cleaning.

Getting back to your task at *AllElectronics*, suppose that you would like to include data from multiple sources in your analysis. This would involve integrating multiple databases, data cubes, or files (i.e., **data integration**). Yet some attributes representing a

given concept may have different names in different databases, causing inconsistencies and redundancies. For example, the attribute for customer identification may be referred to as *customer_id* in one data store and *cust_id* in another. Naming inconsistencies may also occur for attribute values. For example, the same first name could be registered as "Bill" in one database, "William" in another, and "B." in a third. Furthermore, you suspect that some attributes may be inferred from others (e.g., annual revenue). Having a large amount of redundant data may slow down or confuse the knowledge discovery process. Clearly, in addition to data cleaning, steps must be taken to help avoid redundancies during data integration. Typically, data cleaning and data integration are performed as a preprocessing step when preparing data for a data warehouse. Additional data cleaning can be performed to detect and remove redundancies that may have resulted from data integration.

"*Hmmm,*" you wonder, as you consider your data even further. "*The data set I have selected for analysis is HUGE, which is sure to slow down the mining process. Is there a way I can reduce the size of my data set without jeopardizing the data mining results?*" **Data reduction** obtains a reduced representation of the data set that is much smaller in volume, yet produces the same (or almost the same) analytical results. Data reduction strategies include *dimensionality reduction* and *numerosity reduction*.

In **dimensionality reduction**, data encoding schemes are applied so as to obtain a reduced or "compressed" representation of the original data. Examples include data compression techniques (e.g., *wavelet transforms* and *principal components analysis*), *attribute subset selection* (e.g., removing irrelevant attributes), and *attribute construction* (e.g., where a small set of more useful attributes is derived from the original set).

In **numerosity reduction**, the data are replaced by alternative, smaller representations using parametric models (e.g., *regression* or *log-linear models*) or nonparametric models (e.g., *histograms, clusters*, *sampling*, or *data aggregation*). Data reduction is the topic of Section 3.4.

Getting back to your data, you have decided, say, that you would like to use a distance-based mining algorithm for your analysis, such as neural networks, nearest-neighbor classifiers, or clustering.[1] Such methods provide better results if the data to be analyzed have been *normalized*, that is, scaled to a smaller range such as [0.0, 1.0]. Your customer data, for example, contain the attributes *age* and *annual salary*. The *annual salary* attribute usually takes much larger values than *age*. Therefore, if the attributes are left unnormalized, the distance measurements taken on *annual salary* will generally outweigh distance measurements taken on *age*. *Discretization* and *concept hierarchy generation* can also be useful, where raw data values for attributes are replaced by ranges or higher conceptual levels. For example, raw values for *age* may be replaced by higher-level concepts, such as *youth*, *adult*, or *senior*.

Discretization and concept hierarchy generation are powerful tools for data mining in that they allow data mining at multiple abstraction levels. Normalization, data

[1] Neural networks and nearest-neighbor classifiers are described in Chapter 9, and clustering is discussed in Chapters 10 and 11.

discretization, and concept hierarchy generation are forms of **data transformation**. You soon realize such data transformation operations are additional data preprocessing procedures that would contribute toward the success of the mining process. Data integration and data discretization are discussed in Sections 3.5.

Figure 3.1 summarizes the data preprocessing steps described here. Note that the previous categorization is not mutually exclusive. For example, the removal of redundant data may be seen as a form of data cleaning, as well as data reduction.

In summary, real-world data tend to be dirty, incomplete, and inconsistent. Data preprocessing techniques can improve data quality, thereby helping to improve the accuracy and efficiency of the subsequent mining process. Data preprocessing is an important step in the knowledge discovery process, because quality decisions must be based on quality data. Detecting data anomalies, rectifying them early, and reducing the data to be analyzed can lead to huge payoffs for decision making.

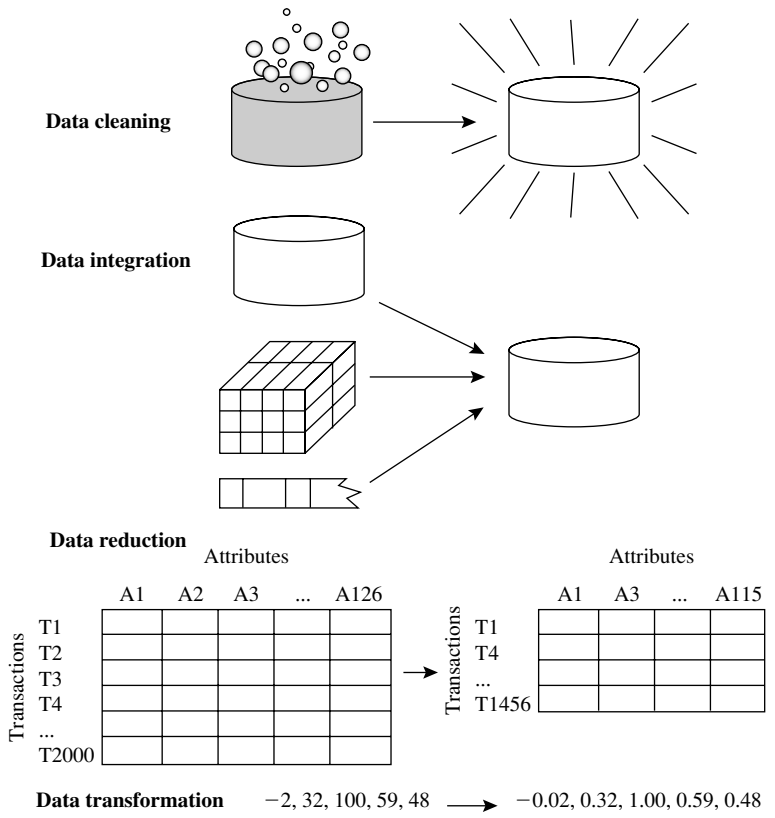


Figure 3.1 Forms of data preprocessing.

3.2 Data Cleaning

Real-world data tend to be incomplete, noisy, and inconsistent. *Data cleaning* (or *data cleansing*) routines attempt to fill in missing values, smooth out noise while identifying outliers, and correct inconsistencies in the data. In this section, you will study basic methods for data cleaning. Section 3.2.1 looks at ways of handling missing values. Section 3.2.2 explains data smoothing techniques. Section 3.2.3 discusses approaches to data cleaning as a process.

3.2.1 Missing Values

Imagine that you need to analyze *AllElectronics* sales and customer data. You note that many tuples have no recorded value for several attributes such as customer *income*. How can you go about filling in the missing values for this attribute? Let's look at the following methods.

1. **Ignore the tuple**: This is usually done when the class label is missing (assuming the mining task involves classification). This method is not very effective, unless the tuple contains several attributes with missing values. It is especially poor when the percentage of missing values per attribute varies considerably. By ignoring the tuple, we do not make use of the remaining attributes' values in the tuple. Such data could have been useful to the task at hand.
2. **Fill in the missing value manually**: In general, this approach is time consuming and may not be feasible given a large data set with many missing values.
3. **Use a global constant to fill in the missing value**: Replace all missing attribute values by the same constant such as a label like "*Unknown*" or $-\infty$. If missing values are replaced by, say, "*Unknown*," then the mining program may mistakenly think that they form an interesting concept, since they all have a value in common—that of "*Unknown*." Hence, although this method is simple, it is not foolproof.
4. **Use a measure of central tendency for the attribute (e.g., the mean or median) to fill in the missing value**: Chapter 2 discussed measures of central tendency, which indicate the "middle" value of a data distribution. For normal (symmetric) data distributions, the mean can be used, while skewed data distribution should employ the median (Section 2.2). For example, suppose that the data distribution regarding the income of *AllElectronics* customers is symmetric and that the mean income is \$56,000. Use this value to replace the missing value for *income*.
5. **Use the attribute mean or median for all samples belonging to the same class as the given tuple**: For example, if classifying customers according to *credit_risk*, we may replace the missing value with the mean *income* value for customers in the same credit risk category as that of the given tuple. If the data distribution for a given class is skewed, the median value is a better choice.
6. **Use the most probable value to fill in the missing value**: This may be determined with regression, inference-based tools using a Bayesian formalism, or decision tree

induction. For example, using the other customer attributes in your data set, you may construct a decision tree to predict the missing values for *income*. Decision trees and Bayesian inference are described in detail in Chapters 8 and 9, respectively, while regression is introduced in Section 3.4.5.

Methods 3 through 6 bias the data—the filled-in value may not be correct. Method 6, however, is a popular strategy. In comparison to the other methods, it uses the most information from the present data to predict missing values. By considering the other attributes' values in its estimation of the missing value for *income*, there is a greater chance that the relationships between *income* and the other attributes are preserved.

It is important to note that, in some cases, a missing value may not imply an error in the data! For example, when applying for a credit card, candidates may be asked to supply their driver's license number. Candidates who do not have a driver's license may naturally leave this field blank. Forms should allow respondents to specify values such as "not applicable." Software routines may also be used to uncover other null values (e.g., "don't know," "?" or "none"). Ideally, each attribute should have one or more rules regarding the *null* condition. The rules may specify whether or not nulls are allowed and/or how such values should be handled or transformed. Fields may also be intentionally left blank if they are to be provided in a later step of the business process. Hence, although we can try our best to clean the data after it is seized, good database and data entry procedure design should help minimize the number of missing values or errors in the first place.

3.2.2 Noisy Data

"*What is noise?*" **Noise** is a random error or variance in a measured variable. In Chapter 2, we saw how some basic statistical description techniques (e.g., boxplots and scatter plots), and methods of data visualization can be used to identify outliers, which may represent noise. Given a numeric attribute such as, say, *price*, how can we "smooth" out the data to remove the noise? Let's look at the following data smoothing techniques.

Binning: Binning methods smooth a sorted data value by consulting its "neighborhood," that is, the values around it. The sorted values are distributed into a number of "buckets," or *bins*. Because binning methods consult the neighborhood of values, they perform *local* smoothing. Figure 3.2 illustrates some binning techniques. In this example, the data for *price* are first sorted and then partitioned into *equal-frequency* bins of size 3 (i.e., each bin contains three values). In **smoothing by bin means**, each value in a bin is replaced by the mean value of the bin. For example, the mean of the values 4, 8, and 15 in Bin 1 is 9. Therefore, each original value in this bin is replaced by the value 9.

Similarly, **smoothing by bin medians** can be employed, in which each bin value is replaced by the bin median. In **smoothing by bin boundaries**, the minimum and maximum values in a given bin are identified as the *bin boundaries*. Each bin value is then replaced by the closest boundary value. In general, the larger the width, the

Sorted data for *price* (in dollars): 4, 8, 15, 21, 21, 24, 25, 28, 34

Partition into (equal-frequency) bins:

Bin 1: 4, 8, 15
Bin 2: 21, 21, 24
Bin 3: 25, 28, 34

Smoothing by bin means:

Bin 1: 9, 9, 9
Bin 2: 22, 22, 22
Bin 3: 29, 29, 29

Smoothing by bin boundaries:

Bin 1: 4, 4, 15
Bin 2: 21, 21, 24
Bin 3: 25, 25, 34

Figure 3.2 Binning methods for data smoothing.

greater the effect of the smoothing. Alternatively, bins may be *equal width*, where the interval range of values in each bin is constant. Binning is also used as a discretization technique and is further discussed in Section 3.5.

Regression: Data smoothing can also be done by regression, a technique that conforms data values to a function. *Linear regression* involves finding the "best" line to fit two attributes (or variables) so that one attribute can be used to predict the other. *Multiple linear regression* is an extension of linear regression, where more than two attributes are involved and the data are fit to a multidimensional surface. Regression is further described in Section 3.4.5.

Outlier analysis: Outliers may be detected by clustering, for example, where similar values are organized into groups, or "clusters." Intuitively, values that fall outside of the set of clusters may be considered outliers (Figure 3.3). Chapter 12 is dedicated to the topic of outlier analysis.

Many data smoothing methods are also used for data discretization (a form of data transformation) and data reduction. For example, the binning techniques described before reduce the number of distinct values per attribute. This acts as a form of data reduction for logic-based data mining methods, such as decision tree induction, which repeatedly makes value comparisons on sorted data. Concept hierarchies are a form of data discretization that can also be used for data smoothing. A concept hierarchy for *price*, for example, may map real *price* values into *inexpensive*, *moderately_priced*, and *expensive*, thereby reducing the number of data values to be handled by the mining

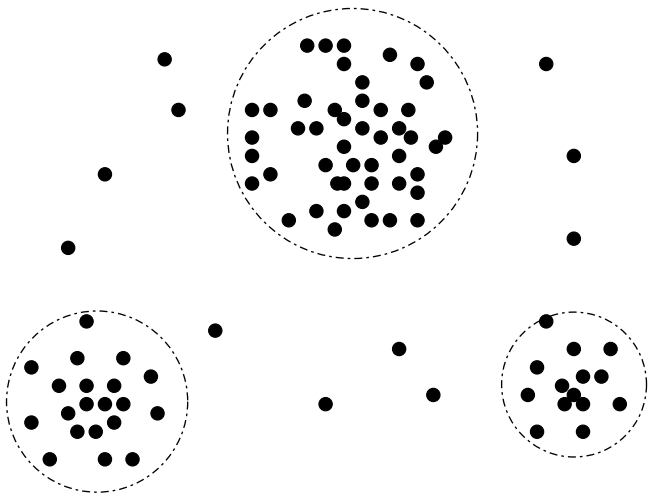


Figure 3.3 A 2-D customer data plot with respect to customer locations in a city, showing three data clusters. Outliers may be detected as values that fall outside of the cluster sets.

process. Data discretization is discussed in Section 3.5. Some methods of classification (e.g., neural networks) have built-in data smoothing mechanisms. Classification is the topic of Chapters 8 and 9.

3.2.3 Data Cleaning as a Process

Missing values, noise, and inconsistencies contribute to inaccurate data. So far, we have looked at techniques for handling missing data and for smoothing data. "*But data cleaning is a big job. What about data cleaning as a process? How exactly does one proceed in tackling this task? Are there any tools out there to help?*"

The first step in data cleaning as a process is *discrepancy detection*. Discrepancies can be caused by several factors, including poorly designed data entry forms that have many optional fields, human error in data entry, deliberate errors (e.g., respondents not wanting to divulge information about themselves), and data decay (e.g., outdated addresses). Discrepancies may also arise from inconsistent data representations and inconsistent use of codes. Other sources of discrepancies include errors in instrumentation devices that record data and system errors. Errors can also occur when the data are (inadequately) used for purposes other than originally intended. There may also be inconsistencies due to data integration (e.g., where a given attribute can have different names in different databases).[2]

[2]Data integration and the removal of redundant data that can result from such integration are further described in Section 3.3.

"*So, how can we proceed with discrepancy detection?*" As a starting point, use any knowledge you may already have regarding properties of the data. Such knowledge or "data about data" is referred to as **metadata.** This is where we can make use of the knowledge we gained about our data in Chapter 2. For example, what are the data type and domain of each attribute? What are the acceptable values for each attribute? The basic statistical data descriptions discussed in Section 2.2 are useful here to grasp data trends and identify anomalies. For example, find the mean, median, and mode values. Are the data symmetric or skewed? What is the range of values? Do all values fall within the expected range? What is the standard deviation of each attribute? Values that are more than two standard deviations away from the mean for a given attribute may be flagged as potential outliers. Are there any known dependencies between attributes? In this step, you may write your own scripts and/or use some of the tools that we discuss further later. From this, you may find noise, outliers, and unusual values that need investigation.

As a data analyst, you should be on the lookout for the inconsistent use of codes and any inconsistent data representations (e.g., "2010/12/25" and "25/12/2010" for *date*). **Field overloading** is another error source that typically results when developers squeeze new attribute definitions into unused (bit) portions of already defined attributes (e.g., an unused bit of an attribute that has a value range that uses only, say, 31 out of 32 bits).

The data should also be examined regarding unique rules, consecutive rules, and null rules. A **unique rule** says that each value of the given attribute must be different from all other values for that attribute. A **consecutive rule** says that there can be no missing values between the lowest and highest values for the attribute, and that all values must also be unique (e.g., as in check numbers). A **null rule** specifies the use of blanks, question marks, special characters, or other strings that may indicate the null condition (e.g., where a value for a given attribute is not available), and how such values should be handled. As mentioned in Section 3.2.1, reasons for missing values may include (1) the person originally asked to provide a value for the attribute refuses and/or finds that the information requested is not applicable (e.g., a *license_number* attribute left blank by nondrivers); (2) the data entry person does not know the correct value; or (3) the value is to be provided by a later step of the process. The null rule should specify how to record the null condition, for example, such as to store zero for numeric attributes, a blank for character attributes, or any other conventions that may be in use (e.g., entries like "don't know" or "?" should be transformed to blank).

There are a number of different commercial tools that can aid in the discrepancy detection step. **Data scrubbing tools** use simple domain knowledge (e.g., knowledge of postal addresses and spell-checking) to detect errors and make corrections in the data. These tools rely on parsing and fuzzy matching techniques when cleaning data from multiple sources. **Data auditing tools** find discrepancies by analyzing the data to discover rules and relationships, and detecting data that violate such conditions. They are variants of data mining tools. For example, they may employ statistical analysis to find correlations, or clustering to identify outliers. They may also use the basic statistical data descriptions presented in Section 2.2.

Some data inconsistencies may be corrected manually using external references. For example, errors made at data entry may be corrected by performing a paper

trace. Most errors, however, will require *data transformations*. That is, once we find discrepancies, we typically need to define and apply (a series of) transformations to correct them.

Commercial tools can assist in the data transformation step. **Data migration tools** allow simple transformations to be specified such as to replace the string *"gender"* by *"sex."* **ETL (extraction/transformation/loading) tools** allow users to specify transforms through a graphical user interface (GUI). These tools typically support only a restricted set of transforms so that, often, we may also choose to write custom scripts for this step of the data cleaning process.

The two-step process of discrepancy detection and data transformation (to correct discrepancies) iterates. This process, however, is error-prone and time consuming. Some transformations may introduce more discrepancies. Some *nested discrepancies* may only be detected after others have been fixed. For example, a typo such as "20010" in a year field may only surface once all date values have been converted to a uniform format. Transformations are often done as a batch process while the user waits without feedback. Only after the transformation is complete can the user go back and check that no new anomalies have been mistakenly created. Typically, numerous iterations are required before the user is satisfied. Any tuples that cannot be automatically handled by a given transformation are typically written to a file without any explanation regarding the reasoning behind their failure. As a result, the entire data cleaning process also suffers from a lack of interactivity.

New approaches to data cleaning emphasize increased interactivity. Potter's Wheel, for example, is a publicly available data cleaning tool that integrates discrepancy detection and transformation. Users gradually build a series of transformations by composing and debugging individual transformations, one step at a time, on a spreadsheet-like interface. The transformations can be specified graphically or by providing examples. Results are shown immediately on the records that are visible on the screen. The user can choose to undo the transformations, so that transformations that introduced additional errors can be "erased." The tool automatically performs discrepancy checking in the background on the latest transformed view of the data. Users can gradually develop and refine transformations as discrepancies are found, leading to more effective and efficient data cleaning.

Another approach to increased interactivity in data cleaning is the development of declarative languages for the specification of data transformation operators. Such work focuses on defining powerful extensions to SQL and algorithms that enable users to express data cleaning specifications efficiently.

As we discover more about the data, it is important to keep updating the metadata to reflect this knowledge. This will help speed up data cleaning on future versions of the same data store.

3.3 Data Integration

Data mining often requires data integration—the merging of data from multiple data stores. Careful integration can help reduce and avoid redundancies and inconsistencies

in the resulting data set. This can help improve the accuracy and speed of the subsequent data mining process.

The semantic heterogeneity and structure of data pose great challenges in data integration. How can we match schema and objects from different sources? This is the essence of the *entity identification problem*, described in Section 3.3.1. Are any attributes correlated? Section 3.3.2 presents correlation tests for numeric and nominal data. Tuple duplication is described in Section 3.3.3. Finally, Section 3.3.4 touches on the detection and resolution of data value conflicts.

3.3.1 Entity Identification Problem

It is likely that your data analysis task will involve *data integration*, which combines data from multiple sources into a coherent data store, as in data warehousing. These sources may include multiple databases, data cubes, or flat files.

There are a number of issues to consider during data integration. *Schema integration* and *object matching* can be tricky. How can equivalent real-world entities from multiple data sources be matched up? This is referred to as the **entity identification problem**. For example, how can the data analyst or the computer be sure that *customer_id* in one database and *cust_number* in another refer to the same attribute? Examples of metadata for each attribute include the name, meaning, data type, and range of values permitted for the attribute, and null rules for handling blank, zero, or null values (Section 3.2). Such metadata can be used to help avoid errors in schema integration. The metadata may also be used to help transform the data (e.g., where data codes for *pay_type* in one database may be "*H*" and "*S*" but *1* and *2* in another). Hence, this step also relates to data cleaning, as described earlier.

When matching attributes from one database to another during integration, special attention must be paid to the *structure* of the data. This is to ensure that any attribute functional dependencies and referential constraints in the source system match those in the target system. For example, in one system, a *discount* may be applied to the order, whereas in another system it is applied to each individual line item within the order. If this is not caught before integration, items in the target system may be improperly discounted.

3.3.2 Redundancy and Correlation Analysis

Redundancy is another important issue in data integration. An attribute (such as *annual revenue*, for instance) may be redundant if it can be "derived" from another attribute or set of attributes. Inconsistencies in attribute or dimension naming can also cause redundancies in the resulting data set.

Some redundancies can be detected by **correlation analysis**. Given two attributes, such analysis can measure how strongly one attribute implies the other, based on the available data. For nominal data, we use the χ^2 (*chi-square*) test. For numeric attributes, we can use the *correlation coefficient* and *covariance*, both of which access how one attribute's values vary from those of another.

χ^2 Correlation Test for Nominal Data

For nominal data, a correlation relationship between two attributes, A and B, can be discovered by a χ^2 (**chi-square**) test. Suppose A has c distinct values, namely $a_1, a_2, \ldots a_c$. B has r distinct values, namely $b_1, b_2, \ldots b_r$. The data tuples described by A and B can be shown as a **contingency table**, with the c values of A making up the columns and the r values of B making up the rows. Let (A_i, B_j) denote the joint event that attribute A takes on value a_i and attribute B takes on value b_j, that is, where $(A = a_i, B = b_j)$. Each and every possible (A_i, B_j) joint event has its own cell (or slot) in the table. The χ^2 value (also known as the *Pearson* χ^2 *statistic*) is computed as

$$\chi^2 = \sum_{i=1}^{c} \sum_{j=1}^{r} \frac{(o_{ij} - e_{ij})^2}{e_{ij}}, \tag{3.1}$$

where o_{ij} is the *observed frequency* (i.e., actual count) of the joint event (A_i, B_j) and e_{ij} is the *expected frequency* of (A_i, B_j), which can be computed as

$$e_{ij} = \frac{count(A = a_i) \times count(B = b_j)}{n}, \tag{3.2}$$

where n is the number of data tuples, $count(A = a_i)$ is the number of tuples having value a_i for A, and $count(B = b_j)$ is the number of tuples having value b_j for B. The sum in Eq. (3.1) is computed over all of the $r \times c$ cells. Note that the cells that contribute the most to the χ^2 value are those for which the actual count is very different from that expected.

The χ^2 statistic tests the hypothesis that A and B are *independent*, that is, there is no correlation between them. The test is based on a significance level, with $(r - 1) \times (c - 1)$ degrees of freedom. We illustrate the use of this statistic in Example 3.1. If the hypothesis can be rejected, then we say that A and B are statistically correlated.

Example 3.1 Correlation analysis of nominal attributes using χ^2. Suppose that a group of 1500 people was surveyed. The gender of each person was noted. Each person was polled as to whether his or her preferred type of reading material was fiction or nonfiction. Thus, we have two attributes, *gender* and *preferred_reading*. The observed frequency (or count) of each possible joint event is summarized in the contingency table shown in Table 3.1, where the numbers in parentheses are the expected frequencies. The expected frequencies are calculated based on the data distribution for both attributes using Eq. (3.2).

Using Eq. (3.2), we can verify the expected frequencies for each cell. For example, the expected frequency for the cell (*male, fiction*) is

$$e_{11} = \frac{count(male) \times count(fiction)}{n} = \frac{300 \times 450}{1500} = 90,$$

and so on. Notice that in any row, the sum of the expected frequencies must equal the total observed frequency for that row, and the sum of the expected frequencies in any column must also equal the total observed frequency for that column.

Table 3.1 Example 2.1's 2 × 2 Contingency Table Data

	male	*female*	*Total*
fiction	250 (90)	200 (360)	450
non_fiction	50 (210)	1000 (840)	1050
Total	300	1200	1500

Note: Are *gender* and *preferred_reading* correlated?

Using Eq. (3.1) for χ^2 computation, we get

$$\chi^2 = \frac{(250-90)^2}{90} + \frac{(50-210)^2}{210} + \frac{(200-360)^2}{360} + \frac{(1000-840)^2}{840}$$
$$= 284.44 + 121.90 + 71.11 + 30.48 = 507.93.$$

For this 2 × 2 table, the degrees of freedom are $(2-1)(2-1) = 1$. For 1 degree of freedom, the χ^2 value needed to reject the hypothesis at the 0.001 significance level is 10.828 (taken from the table of upper percentage points of the χ^2 distribution, typically available from any textbook on statistics). Since our computed value is above this, we can reject the hypothesis that *gender* and *preferred_reading* are independent and conclude that the two attributes are (strongly) correlated for the given group of people. ■

Correlation Coefficient for Numeric Data

For numeric attributes, we can evaluate the correlation between two attributes, A and B, by computing the **correlation coefficient** (also known as **Pearson's product moment coefficient**, named after its inventer, Karl Pearson). This is

$$r_{A,B} = \frac{\sum_{i=1}^{n}(a_i - \bar{A})(b_i - \bar{B})}{n\sigma_A\sigma_B} = \frac{\sum_{i=1}^{n}(a_i b_i) - n\bar{A}\bar{B}}{n\sigma_A\sigma_B}, \tag{3.3}$$

where n is the number of tuples, a_i and b_i are the respective values of A and B in tuple i, $\bar{A}$ and $\bar{B}$ are the respective mean values of A and B, σ_A and σ_B are the respective standard deviations of A and B (as defined in Section 2.2.2), and $\Sigma(a_i b_i)$ is the sum of the AB cross-product (i.e., for each tuple, the value for A is multiplied by the value for B in that tuple). Note that $-1 \le r_{A,B} \le +1$. If $r_{A,B}$ is greater than 0, then A and B are *positively correlated*, meaning that the values of A increase as the values of B increase. The higher the value, the stronger the correlation (i.e., the more each attribute implies the other). Hence, a higher value may indicate that A (or B) may be removed as a redundancy.

If the resulting value is equal to 0, then A and B are *independent* and there is no correlation between them. If the resulting value is less than 0, then A and B are *negatively correlated*, where the values of one attribute increase as the values of the other attribute decrease. This means that each attribute discourages the other. Scatter plots can also be used to view correlations between attributes (Section 2.2.3). For example, Figure 2.8's

scatter plots respectively show positively correlated data and negatively correlated data, while Figure 2.9 displays uncorrelated data.

Note that correlation does not imply causality. That is, if A and B are correlated, this does not necessarily imply that A causes B or that B causes A. For example, in analyzing a demographic database, we may find that attributes representing the number of hospitals and the number of car thefts in a region are correlated. This does not mean that one causes the other. Both are actually causally linked to a third attribute, namely, *population*.

Covariance of Numeric Data

In probability theory and statistics, correlation and covariance are two similar measures for assessing how much two attributes change together. Consider two numeric attributes A and B, and a set of n observations $\{(a_1, b_1), \ldots, (a_n, b_n)\}$. The mean values of A and B, respectively, are also known as the **expected values** on A and B, that is,

$$E(A) = \bar{A} = \frac{\sum_{i=1}^{n} a_i}{n}$$

and

$$E(B) = \bar{B} = \frac{\sum_{i=1}^{n} b_i}{n}.$$

The **covariance** between A and B is defined as

$$Cov(A, B) = E((A - \bar{A})(B - \bar{B})) = \frac{\sum_{i=1}^{n} (a_i - \bar{A})(b_i - \bar{B})}{n}. \tag{3.4}$$

If we compare Eq. (3.3) for $r_{A,B}$ (correlation coefficient) with Eq. (3.4) for covariance, we see that

$$r_{A,B} = \frac{Cov(A, B)}{\sigma_A \sigma_B}, \tag{3.5}$$

where σ_A and σ_B are the standard deviations of A and B, respectively. It can also be shown that

$$Cov(A, B) = E(A \cdot B) - \bar{A}\bar{B}. \tag{3.6}$$

This equation may simplify calculations.

For two attributes A and B that tend to change together, if A is larger than $\bar{A}$ (the expected value of A), then B is likely to be larger than $\bar{B}$ (the expected value of B). Therefore, the covariance between A and B is *positive*. On the other hand, if one of the attributes tends to be above its expected value when the other attribute is below its expected value, then the covariance of A and B is *negative*.

If A and B are *independent* (i.e., they do not have correlation), then $E(A \cdot B) = E(A) \cdot E(B)$. Therefore, the covariance is $Cov(A, B) = E(A \cdot B) - \bar{A}\bar{B} = E(A) \cdot E(B) - \bar{A}\bar{B} = 0$. However, the converse is not true. Some pairs of random variables (attributes) may have a covariance of 0 but are not independent. Only under some additional assumptions

Table 3.2 Stock Prices for *AllElectronics* and *HighTech*

Time point	*AllElectronics*	*HighTech*
t1	6	20
t2	5	10
t3	4	14
t4	3	5
t5	2	5

(e.g., the data follow multivariate normal distributions) does a covariance of 0 imply independence.

Example 3.2 **Covariance analysis of numeric attributes.** Consider Table 3.2, which presents a simplified example of stock prices observed at five time points for *AllElectronics* and *HighTech*, a high-tech company. If the stocks are affected by the same industry trends, will their prices rise or fall together?

$$E(AllElectronics) = \frac{6+5+4+3+2}{5} = \frac{20}{5} = \$4$$

and

$$E(HighTech) = \frac{20+10+14+5+5}{5} = \frac{54}{5} = \$10.80.$$

Thus, using Eq. (3.4), we compute

$$Cov(AllElectroncis, HighTech) = \frac{6\times 20+5\times 10+4\times 14+3\times 5+2\times 5}{5} - 4\times 10.80$$
$$= 50.2 - 43.2 = 7.$$

Therefore, given the positive covariance we can say that stock prices for both companies rise together. ■

Variance is a special case of covariance, where the two attributes are identical (i.e., the covariance of an attribute with itself). Variance was discussed in Chapter 2.

3.3.3 Tuple Duplication

In addition to detecting redundancies between attributes, duplication should also be detected at the tuple level (e.g., where there are two or more identical tuples for a given unique data entry case). The use of denormalized tables (often done to improve performance by avoiding joins) is another source of data redundancy. Inconsistencies often arise between various duplicates, due to inaccurate data entry or updating some but not all data occurrences. For example, if a purchase order database contains attributes for

the purchaser's name and address instead of a key to this information in a purchaser database, discrepancies can occur, such as the same purchaser's name appearing with different addresses within the purchase order database.

3.3.4 Data Value Conflict Detection and Resolution

Data integration also involves the *detection and resolution of data value conflicts*. For example, for the same real-world entity, attribute values from different sources may differ. This may be due to differences in representation, scaling, or encoding. For instance, a *weight* attribute may be stored in metric units in one system and British imperial units in another. For a hotel chain, the *price* of rooms in different cities may involve not only different currencies but also different services (e.g., free breakfast) and taxes. When exchanging information between schools, for example, each school may have its own curriculum and grading scheme. One university may adopt a quarter system, offer three courses on database systems, and assign grades from A+ to F, whereas another may adopt a semester system, offer two courses on databases, and assign grades from 1 to 10. It is difficult to work out precise course-to-grade transformation rules between the two universities, making information exchange difficult.

Attributes may also differ on the abstraction level, where an attribute in one system is recorded at, say, a lower abstraction level than the "same" attribute in another. For example, the *total_sales* in one database may refer to one branch of *All_Electronics*, while an attribute of the same name in another database may refer to the total sales for *All_Electronics* stores in a given region. The topic of discrepancy detection is further described in Section 3.2.3 on data cleaning as a process.

3.4 Data Reduction

Imagine that you have selected data from the *AllElectronics* data warehouse for analysis. The data set will likely be huge! Complex data analysis and mining on huge amounts of data can take a long time, making such analysis impractical or infeasible.

Data reduction techniques can be applied to obtain a reduced representation of the data set that is much smaller in volume, yet closely maintains the integrity of the original data. That is, mining on the reduced data set should be more efficient yet produce the same (or almost the same) analytical results. In this section, we first present an overview of data reduction strategies, followed by a closer look at individual techniques.

3.4.1 Overview of Data Reduction Strategies

Data reduction strategies include *dimensionality reduction*, *numerosity reduction*, and *data compression*.

Dimensionality reduction is the process of reducing the number of random variables or attributes under consideration. Dimensionality reduction methods include *wavelet*

transforms (Section 3.4.2) and *principal components analysis* (Section 3.4.3), which transform or project the original data onto a smaller space. *Attribute subset selection* is a method of dimensionality reduction in which irrelevant, weakly relevant, or redundant attributes or dimensions are detected and removed (Section 3.4.4).

Numerosity reduction techniques replace the original data volume by alternative, smaller forms of data representation. These techniques may be parametric or nonparametric. For *parametric methods*, a model is used to estimate the data, so that typically only the data parameters need to be stored, instead of the actual data. (Outliers may also be stored.) Regression and log-linear models (Section 3.4.5) are examples. *Nonparametric methods* for storing reduced representations of the data include *histograms* (Section 3.4.6), *clustering* (Section 3.4.7), *sampling* (Section 3.4.8), and *data cube aggregation* (Section 3.4.9).

In **data compression**, transformations are applied so as to obtain a reduced or "compressed" representation of the original data. If the original data can be *reconstructed* from the compressed data without any information loss, the data reduction is called **lossless**. If, instead, we can reconstruct only an approximation of the original data, then the data reduction is called **lossy**. There are several lossless algorithms for string compression; however, they typically allow only limited data manipulation. Dimensionality reduction and numerosity reduction techniques can also be considered forms of data compression.

There are many other ways of organizing methods of data reduction. The computational time spent on data reduction should not outweigh or "erase" the time saved by mining on a reduced data set size.

3.4.2 Wavelet Transforms

The **discrete wavelet transform** (**DWT**) is a linear signal processing technique that, when applied to a data vector $\boldsymbol{X}$, transforms it to a numerically different vector, $\boldsymbol{X}'$, of **wavelet coefficients**. The two vectors are of the same length. When applying this technique to data reduction, we consider each tuple as an n-dimensional data vector, that is, $\boldsymbol{X} = (x_1, x_2, \ldots, x_n)$, depicting n measurements made on the tuple from n database attributes.[3]

"*How can this technique be useful for data reduction if the wavelet transformed data are of the same length as the original data?*" The usefulness lies in the fact that the wavelet transformed data can be truncated. A compressed approximation of the data can be retained by storing only a small fraction of the strongest of the wavelet coefficients. For example, all wavelet coefficients larger than some user-specified threshold can be retained. All other coefficients are set to 0. The resulting data representation is therefore very sparse, so that operations that can take advantage of data sparsity are computationally very fast if performed in wavelet space. The technique also works to remove noise without smoothing out the main features of the data, making it effective for data

[3] In our notation, any variable representing a vector is shown in bold italic font; measurements depicting the vector are shown in italic font.

cleaning as well. Given a set of coefficients, an approximation of the original data can be constructed by applying the *inverse* of the DWT used.

The DWT is closely related to the *discrete Fourier transform (DFT)*, a signal processing technique involving sines and cosines. In general, however, the DWT achieves better lossy compression. That is, if the same number of coefficients is retained for a DWT and a DFT of a given data vector, the DWT version will provide a more accurate approximation of the original data. Hence, for an equivalent approximation, the DWT requires less space than the DFT. Unlike the DFT, wavelets are quite localized in space, contributing to the conservation of local detail.

There is only one DFT, yet there are several families of DWTs. Figure 3.4 shows some wavelet families. Popular wavelet transforms include the Haar-2, Daubechies-4, and Daubechies-6. The general procedure for applying a discrete wavelet transform uses a hierarchical *pyramid algorithm* that halves the data at each iteration, resulting in fast computational speed. The method is as follows:

1. The length, L, of the input data vector must be an integer power of 2. This condition can be met by padding the data vector with zeros as necessary ($L \geq n$).
2. Each transform involves applying two functions. The first applies some data smoothing, such as a sum or weighted average. The second performs a weighted difference, which acts to bring out the detailed features of the data.
3. The two functions are applied to pairs of data points in $\boldsymbol{X}$, that is, to all pairs of measurements (x_{2i}, x_{2i+1}). This results in two data sets of length $L/2$. In general, these represent a smoothed or low-frequency version of the input data and the high-frequency content of it, respectively.
4. The two functions are recursively applied to the data sets obtained in the previous loop, until the resulting data sets obtained are of length 2.
5. Selected values from the data sets obtained in the previous iterations are designated the wavelet coefficients of the transformed data.

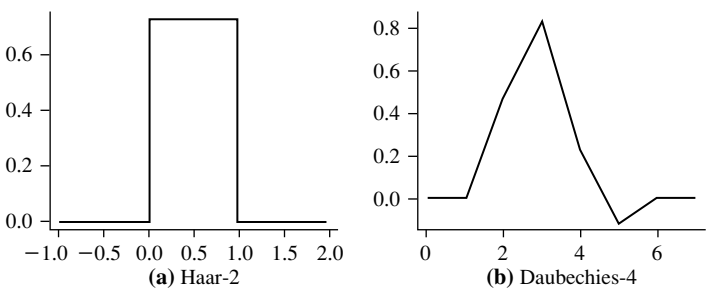


Figure 3.4 Examples of wavelet families. The number next to a wavelet name is the number of *vanishing moments* of the wavelet. This is a set of mathematical relationships that the coefficients must satisfy and is related to the number of coefficients.

Equivalently, a matrix multiplication can be applied to the input data in order to obtain the wavelet coefficients, where the matrix used depends on the given DWT. The matrix must be **orthonormal**, meaning that the columns are unit vectors and are mutually orthogonal, so that the matrix inverse is just its transpose. Although we do not have room to discuss it here, this property allows the reconstruction of the data from the smooth and smooth-difference data sets. By factoring the matrix used into a product of a few sparse matrices, the resulting "fast DWT" algorithm has a complexity of $O(n)$ for an input vector of length n.

Wavelet transforms can be applied to multidimensional data such as a data cube. This is done by first applying the transform to the first dimension, then to the second, and so on. The computational complexity involved is linear with respect to the number of cells in the cube. Wavelet transforms give good results on sparse or skewed data and on data with ordered attributes. Lossy compression by wavelets is reportedly better than JPEG compression, the current commercial standard. Wavelet transforms have many real-world applications, including the compression of fingerprint images, computer vision, analysis of time-series data, and data cleaning.

3.4.3 Principal Components Analysis

In this subsection we provide an intuitive introduction to principal components analysis as a method of dimesionality reduction. A detailed theoretical explanation is beyond the scope of this book. For additional references, please see the bibliographic notes (Section 3.8) at the end of this chapter.

Suppose that the data to be reduced consist of tuples or data vectors described by n attributes or dimensions. **Principal components analysis** (**PCA**; also called the Karhunen-Loeve, or K-L, method) searches for k n-dimensional orthogonal vectors that can best be used to represent the data, where $k \leq n$. The original data are thus projected onto a much smaller space, resulting in dimensionality reduction. Unlike attribute subset selection (Section 3.4.4), which reduces the attribute set size by retaining a subset of the initial set of attributes, PCA "combines" the essence of attributes by creating an alternative, smaller set of variables. The initial data can then be projected onto this smaller set. PCA often reveals relationships that were not previously suspected and thereby allows interpretations that would not ordinarily result.

The basic procedure is as follows:

1. The input data are normalized, so that each attribute falls within the same range. This step helps ensure that attributes with large domains will not dominate attributes with smaller domains.
2. PCA computes k orthonormal vectors that provide a basis for the normalized input data. These are unit vectors that each point in a direction perpendicular to the others. These vectors are referred to as the *principal components*. The input data are a linear combination of the principal components.
3. The principal components are sorted in order of decreasing "significance" or strength. The principal components essentially serve as a new set of axes for the data,

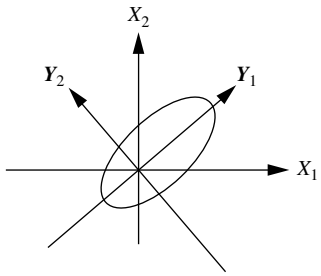


Figure 3.5 Principal components analysis. $\boldsymbol{Y}_1$ and $\boldsymbol{Y}_2$ are the first two principal components for the given data.

providing important information about variance. That is, the sorted axes are such that the first axis shows the most variance among the data, the second axis shows the next highest variance, and so on. For example, Figure 3.5 shows the first two principal components, $\boldsymbol{Y}_1$ and $\boldsymbol{Y}_2$, for the given set of data originally mapped to the axes X_1 and X_2. This information helps identify groups or patterns within the data.

4. Because the components are sorted in decreasing order of "significance," the data size can be reduced by eliminating the weaker components, that is, those with low variance. Using the strongest principal components, it should be possible to reconstruct a good approximation of the original data.

PCA can be applied to ordered and unordered attributes, and can handle sparse data and skewed data. Multidimensional data of more than two dimensions can be handled by reducing the problem to two dimensions. Principal components may be used as inputs to multiple regression and cluster analysis. In comparison with wavelet transforms, PCA tends to be better at handling sparse data, whereas wavelet transforms are more suitable for data of high dimensionality.

3.4.4 Attribute Subset Selection

Data sets for analysis may contain hundreds of attributes, many of which may be irrelevant to the mining task or redundant. For example, if the task is to classify customers based on whether or not they are likely to purchase a popular new CD at *AllElectronics* when notified of a sale, attributes such as the customer's telephone number are likely to be irrelevant, unlike attributes such as *age* or *music_taste*. Although it may be possible for a domain expert to pick out some of the useful attributes, this can be a difficult and time-consuming task, especially when the data's behavior is not well known. (Hence, a reason behind its analysis!) Leaving out relevant attributes or keeping irrelevant attributes may be detrimental, causing confusion for the mining algorithm employed. This can result in discovered patterns of poor quality. In addition, the added volume of irrelevant or redundant attributes can slow down the mining process.

Attribute subset selection[4] reduces the data set size by removing irrelevant or redundant attributes (or dimensions). The goal of attribute subset selection is to find a minimum set of attributes such that the resulting probability distribution of the data classes is as close as possible to the original distribution obtained using all attributes. Mining on a reduced set of attributes has an additional benefit: It reduces the number of attributes appearing in the discovered patterns, helping to make the patterns easier to understand.

"*How can we find a 'good' subset of the original attributes?*" For n attributes, there are 2^n possible subsets. An exhaustive search for the optimal subset of attributes can be prohibitively expensive, especially as n and the number of data classes increase. Therefore, heuristic methods that explore a reduced search space are commonly used for attribute subset selection. These methods are typically **greedy** in that, while searching through attribute space, they always make what looks to be the best choice at the time. Their strategy is to make a locally optimal choice in the hope that this will lead to a globally optimal solution. Such greedy methods are effective in practice and may come close to estimating an optimal solution.

The "best" (and "worst") attributes are typically determined using tests of statistical significance, which assume that the attributes are independent of one another. Many other attribute evaluation measures can be used such as the *information gain* measure used in building decision trees for classification.[5]

Basic heuristic methods of attribute subset selection include the techniques that follow, some of which are illustrated in Figure 3.6.

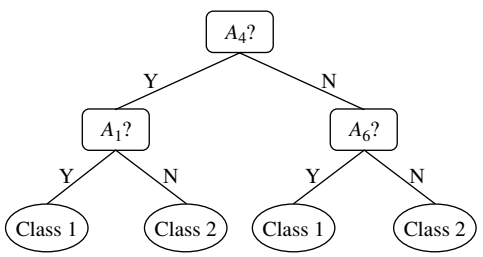
Forward selection	Backward elimination	Decision tree induction
Initial attribute set: $\{A_1, A_2, A_3, A_4, A_5, A_6\}$ Initial reduced set: $\{\}$ $\Rightarrow \{A_1\}$ $\Rightarrow \{A_1, A_4\}$ => Reduced attribute set: $\{A_1, A_4, A_6\}$	Initial attribute set: $\{A_1, A_2, A_3, A_4, A_5, A_6\}$ $\Rightarrow \{A_1, A_3, A_4, A_5, A_6\}$ $\Rightarrow \{A_1, A_4, A_5, A_6\}$ => Reduced attribute set: $\{A_1, A_4, A_6\}$	Initial attribute set: $\{A_1, A_2, A_3, A_4, A_5, A_6\}$  => Reduced attribute set: $\{A_1, A_4, A_6\}$

Figure 3.6 Greedy (heuristic) methods for attribute subset selection.

[4]In machine learning, attribute subset selection is known as *feature subset selection.*

[5]The information gain measure is described in detail in Chapter 8.

1. **Stepwise forward selection**: The procedure starts with an empty set of attributes as the reduced set. The best of the original attributes is determined and added to the reduced set. At each subsequent iteration or step, the best of the remaining original attributes is added to the set.
2. **Stepwise backward elimination**: The procedure starts with the full set of attributes. At each step, it removes the worst attribute remaining in the set.
3. **Combination of forward selection and backward elimination**: The stepwise forward selection and backward elimination methods can be combined so that, at each step, the procedure selects the best attribute and removes the worst from among the remaining attributes.
4. **Decision tree induction**: Decision tree algorithms (e.g., ID3, C4.5, and CART) were originally intended for classification. Decision tree induction constructs a flowchart-like structure where each internal (nonleaf) node denotes a test on an attribute, each branch corresponds to an outcome of the test, and each external (leaf) node denotes a class prediction. At each node, the algorithm chooses the "best" attribute to partition the data into individual classes.

   When decision tree induction is used for attribute subset selection, a tree is constructed from the given data. All attributes that do not appear in the tree are assumed to be irrelevant. The set of attributes appearing in the tree form the reduced subset of attributes.

The stopping criteria for the methods may vary. The procedure may employ a threshold on the measure used to determine when to stop the attribute selection process.

In some cases, we may want to create new attributes based on others. Such **attribute construction**[6] can help improve accuracy and understanding of structure in high-dimensional data. For example, we may wish to add the attribute *area* based on the attributes *height* and *width*. By combining attributes, attribute construction can discover missing information about the relationships between data attributes that can be useful for knowledge discovery.

3.4.5 Regression and Log-Linear Models: Parametric Data Reduction

Regression and log-linear models can be used to approximate the given data. In (simple) **linear regression**, the data are modeled to fit a straight line. For example, a random variable, y (called a *response variable*), can be modeled as a linear function of another random variable, x (called a *predictor variable*), with the equation

$$y = wx + b, \tag{3.7}$$

where the variance of y is assumed to be constant. In the context of data mining, x and y are numeric database attributes. The coefficients, w and b (called *regression coefficients*),

[6] In the machine learning literature, attribute construction is known as *feature construction*.

specify the slope of the line and the *y*-intercept, respectively. These coefficients can be solved for by the *method of least squares*, which minimizes the error between the actual line separating the data and the estimate of the line. **Multiple linear regression** is an extension of (simple) linear regression, which allows a response variable, *y*, to be modeled as a linear function of two or more predictor variables.

Log-linear models approximate discrete multidimensional probability distributions. Given a set of tuples in *n* dimensions (e.g., described by *n* attributes), we can consider each tuple as a point in an *n*-dimensional space. Log-linear models can be used to estimate the probability of each point in a multidimensional space for a set of discretized attributes, based on a smaller subset of dimensional combinations. This allows a higher-dimensional data space to be constructed from lower-dimensional spaces. Log-linear models are therefore also useful for dimensionality reduction (since the lower-dimensional points together typically occupy less space than the original data points) and data smoothing (since aggregate estimates in the lower-dimensional space are less subject to sampling variations than the estimates in the higher-dimensional space).

Regression and log-linear models can both be used on sparse data, although their application may be limited. While both methods can handle skewed data, regression does exceptionally well. Regression can be computationally intensive when applied to high-dimensional data, whereas log-linear models show good scalability for up to 10 or so dimensions.

Several software packages exist to solve regression problems. Examples include SAS (*www.sas.com*), SPSS (*www.spss.com*), and S-Plus (*www.insightful.com*). Another useful resource is the book *Numerical Recipes in C*, by Press, Teukolsky, Vetterling, and Flannery [PTVF07], and its associated source code.

3.4.6 Histograms

Histograms use binning to approximate data distributions and are a popular form of data reduction. Histograms were introduced in Section 2.2.3. A **histogram** for an attribute, *A*, partitions the data distribution of *A* into disjoint subsets, referred to as *buckets* or *bins*. If each bucket represents only a single attribute–value/frequency pair, the buckets are called *singleton buckets*. Often, buckets instead represent continuous ranges for the given attribute.

Example 3.3 **Histograms.** The following data are a list of *AllElectronics* prices for commonly sold items (rounded to the nearest dollar). The numbers have been sorted: 1, 1, 5, 5, 5, 5, 5, 8, 8, 10, 10, 10, 10, 12, 14, 14, 14, 15, 15, 15, 15, 15, 15, 18, 18, 18, 18, 18, 18, 18, 18, 20, 20, 20, 20, 20, 20, 20, 21, 21, 21, 21, 25, 25, 25, 25, 25, 28, 28, 30, 30, 30.

Figure 3.7 shows a histogram for the data using singleton buckets. To further reduce the data, it is common to have each bucket denote a continuous value range for the given attribute. In Figure 3.8, each bucket represents a different $10 range for *price*. ■

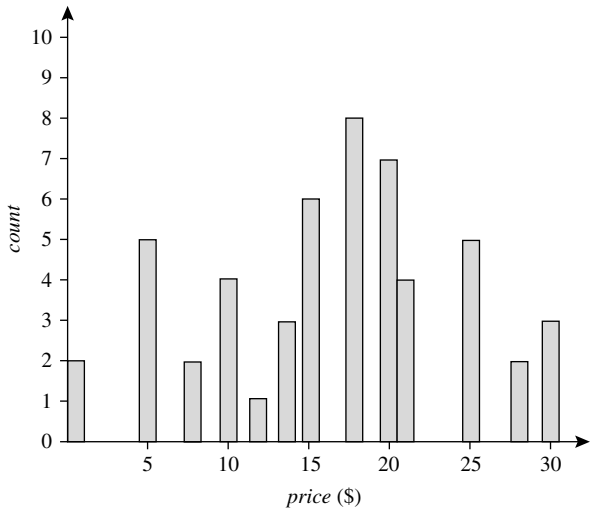


Figure 3.7 A histogram for *price* using singleton buckets—each bucket represents one price–value/frequency pair.

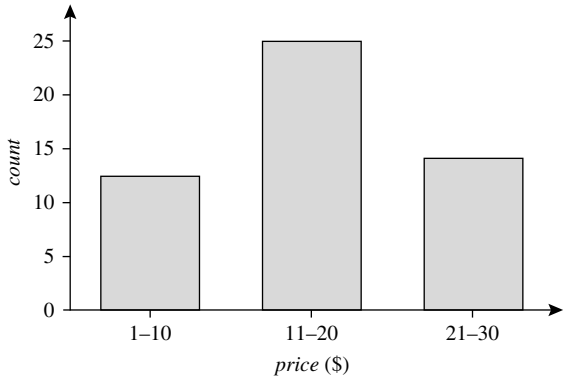


Figure 3.8 An equal-width histogram for *price*, where values are aggregated so that each bucket has a uniform width of $10.

"*How are the buckets determined and the attribute values partitioned?*" There are several partitioning rules, including the following:

- **Equal-width**: In an equal-width histogram, the width of each bucket range is uniform (e.g., the width of $10 for the buckets in Figure 3.8).
- **Equal-frequency** (or equal-depth): In an equal-frequency histogram, the buckets are created so that, roughly, the frequency of each bucket is constant (i.e., each bucket contains roughly the same number of contiguous data samples).

Histograms are highly effective at approximating both sparse and dense data, as well as highly skewed and uniform data. The histograms described before for single attributes can be extended for multiple attributes. *Multidimensional histograms* can capture dependencies between attributes. These histograms have been found effective in approximating data with up to five attributes. More studies are needed regarding the effectiveness of multidimensional histograms for high dimensionalities.

Singleton buckets are useful for storing high-frequency outliers.

3.4.7 Clustering

Clustering techniques consider data tuples as objects. They partition the objects into groups, or *clusters*, so that objects within a cluster are "similar" to one another and "dissimilar" to objects in other clusters. Similarity is commonly defined in terms of how "close" the objects are in space, based on a distance function. The "quality" of a cluster may be represented by its *diameter*, the maximum distance between any two objects in the cluster. **Centroid distance** is an alternative measure of cluster quality and is defined as the average distance of each cluster object from the cluster centroid (denoting the "average object," or average point in space for the cluster). Figure 3.3 showed a 2-D plot of customer data with respect to customer locations in a city. Three data clusters are visible.

In data reduction, the cluster representations of the data are used to replace the actual data. The effectiveness of this technique depends on the data's nature. It is much more effective for data that can be organized into distinct clusters than for smeared data.

There are many measures for defining clusters and cluster quality. Clustering methods are further described in Chapters 10 and 11.

3.4.8 Sampling

Sampling can be used as a data reduction technique because it allows a large data set to be represented by a much smaller random data sample (or subset). Suppose that a large data set, D, contains N tuples. Let's look at the most common ways that we could sample D for data reduction, as illustrated in Figure 3.9.

- **Simple random sample without replacement (SRSWOR) of size** s: This is created by drawing s of the N tuples from D ($s < N$), where the probability of drawing any tuple in D is $1/N$, that is, all tuples are equally likely to be sampled.
- **Simple random sample with replacement (SRSWR) of size** s: This is similar to SRSWOR, except that each time a tuple is drawn from D, it is recorded and then *replaced*. That is, after a tuple is drawn, it is placed back in D so that it may be drawn again.
- **Cluster sample**: If the tuples in D are grouped into M mutually disjoint "clusters," then an SRS of s clusters can be obtained, where $s < M$. For example, tuples in a database are usually retrieved a page at a time, so that each page can be considered

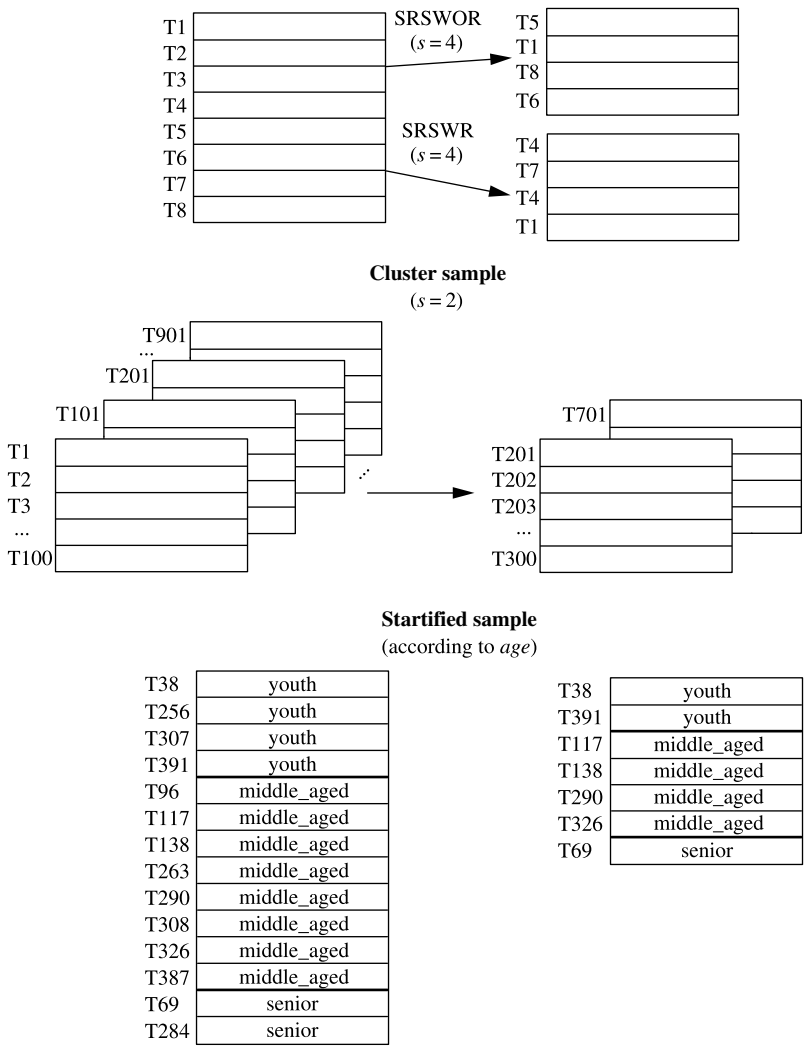


Figure 3.9 Sampling can be used for data reduction.

a cluster. A reduced data representation can be obtained by applying, say, SRSWOR to the pages, resulting in a cluster sample of the tuples. Other clustering criteria conveying rich semantics can also be explored. For example, in a spatial database, we may choose to define clusters geographically based on how closely different areas are located.

- **Stratified sample:** If D is divided into mutually disjoint parts called *strata,* a stratified sample of D is generated by obtaining an SRS at each stratum. This helps ensure a

representative sample, especially when the data are skewed. For example, a stratified sample may be obtained from customer data, where a stratum is created for each customer age group. In this way, the age group having the smallest number of customers will be sure to be represented.

An advantage of sampling for data reduction is that the cost of obtaining a sample *is proportional to the size of the sample*, *s*, as opposed to *N*, the data set size. Hence, sampling complexity is potentially *sublinear* to the size of the data. Other data reduction techniques can require at least one complete pass through *D*. For a fixed sample size, sampling complexity increases only linearly as the number of data dimensions, *n*, increases, whereas techniques using histograms, for example, increase exponentially in *n*.

When applied to data reduction, sampling is most commonly used to estimate the answer to an aggregate query. It is possible (using the central limit theorem) to determine a sufficient sample size for estimating a given function within a specified degree of error. This sample size, *s*, may be extremely small in comparison to *N*. Sampling is a natural choice for the progressive refinement of a reduced data set. Such a set can be further refined by simply increasing the sample size.

3.4.9 Data Cube Aggregation

Imagine that you have collected the data for your analysis. These data consist of the *AllElectronics* sales per quarter, for the years 2008 to 2010. You are, however, interested in the annual sales (total per year), rather than the total per quarter. Thus, the data can be *aggregated* so that the resulting data summarize the total sales per year instead of per quarter. This aggregation is illustrated in Figure 3.10. The resulting data set is smaller in volume, without loss of information necessary for the analysis task.

Data cubes are discussed in detail in Chapter 4 on data warehousing and Chapter 5 on data cube technology. We briefly introduce some concepts here. Data cubes store

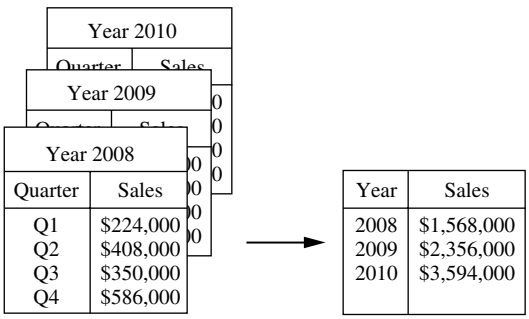


Figure 3.10 Sales data for a given branch of *AllElectronics* for the years 2008 through 2010. On the *left*, the sales are shown per quarter. On the *right*, the data are aggregated to provide the annual sales.

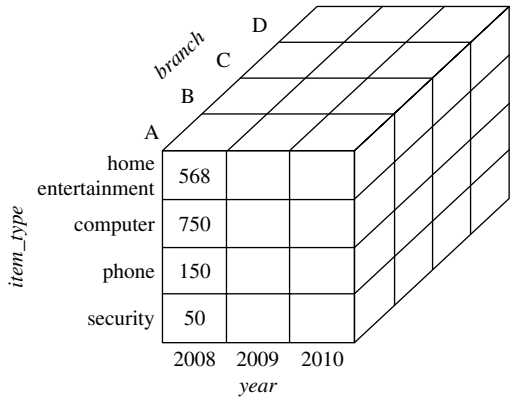


Figure 3.11 A data cube for sales at *AllElectronics*.

multidimensional aggregated information. For example, Figure 3.11 shows a data cube for multidimensional analysis of sales data with respect to annual sales per item type for each *AllElectronics* branch. Each cell holds an aggregate data value, corresponding to the data point in multidimensional space. (For readability, only some cell values are shown.) *Concept hierarchies* may exist for each attribute, allowing the analysis of data at multiple abstraction levels. For example, a hierarchy for *branch* could allow branches to be grouped into regions, based on their address. Data cubes provide fast access to precomputed, summarized data, thereby benefiting online analytical processing as well as data mining.

The cube created at the lowest abstraction level is referred to as the **base cuboid**. The base cuboid should correspond to an individual entity of interest such as *sales* or *customer*. In other words, the lowest level should be usable, or useful for the analysis. A cube at the highest level of abstraction is the **apex cuboid**. For the sales data in Figure 3.11, the apex cuboid would give one total—the total *sales* for all three years, for all item types, and for all branches. Data cubes created for varying levels of abstraction are often referred to as *cuboids*, so that a data cube may instead refer to a *lattice of cuboids*. Each higher abstraction level further reduces the resulting data size. When replying to data mining requests, the *smallest* available cuboid relevant to the given task should be used. This issue is also addressed in Chapter 4.

3.5 Data Transformation and Data Discretization

This section presents methods of data transformation. In this preprocessing step, the data are transformed or consolidated so that the resulting mining process may be more efficient, and the patterns found may be easier to understand. Data discretization, a form of data transformation, is also discussed.

3.5.1 Data Transformation Strategies Overview

In *data transformation*, the data are transformed or consolidated into forms appropriate for mining. Strategies for data transformation include the following:

1. **Smoothing**, which works to remove noise from the data. Techniques include binning, regression, and clustering.
2. **Attribute construction** (or *feature construction*), where new attributes are constructed and added from the given set of attributes to help the mining process.
3. **Aggregation**, where summary or aggregation operations are applied to the data. For example, the daily sales data may be aggregated so as to compute monthly and annual total amounts. This step is typically used in constructing a data cube for data analysis at multiple abstraction levels.
4. **Normalization**, where the attribute data are scaled so as to fall within a smaller range, such as −1.0 to 1.0, or 0.0 to 1.0.
5. **Discretization**, where the raw values of a numeric attribute (e.g., *age*) are replaced by interval labels (e.g., 0–10, 11–20, etc.) or conceptual labels (e.g., *youth, adult, senior*). The labels, in turn, can be recursively organized into higher-level concepts, resulting in a *concept hierarchy* for the numeric attribute. Figure 3.12 shows a concept hierarchy for the attribute *price*. More than one concept hierarchy can be defined for the same attribute to accommodate the needs of various users.
6. **Concept hierarchy generation for nominal data**, where attributes such as *street* can be generalized to higher-level concepts, like *city* or *country*. Many hierarchies for nominal attributes are implicit within the database schema and can be automatically defined at the schema definition level.

Recall that there is much overlap between the major data preprocessing tasks. The first three of these strategies were discussed earlier in this chapter. Smoothing is a form of

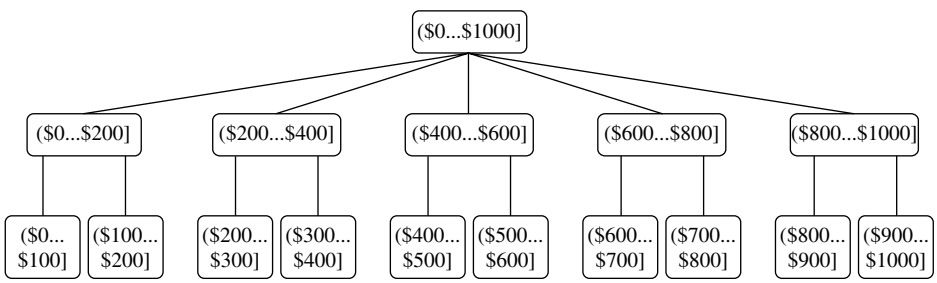


Figure 3.12 A concept hierarchy for the attribute *price*, where an interval ($\$X \ldots \$Y$] denotes the range from $\$X$ (exclusive) to $\$Y$ (inclusive).

data cleaning and was addressed in Section 3.2.2. Section 3.2.3 on the data cleaning process also discussed ETL tools, where users specify transformations to correct data inconsistencies. Attribute construction and aggregation were discussed in Section 3.4 on data reduction. In this section, we therefore concentrate on the latter three strategies.

Discretization techniques can be categorized based on how the discretization is performed, such as whether it uses class information or which direction it proceeds (i.e., top-down vs. bottom-up). If the discretization process uses class information, then we say it is *supervised discretization*. Otherwise, it is *unsupervised*. If the process starts by first finding one or a few points (called *split points* or *cut points*) to split the entire attribute range, and then repeats this recursively on the resulting intervals, it is called *top-down discretization* or *splitting*. This contrasts with *bottom-up discretization* or *merging*, which starts by considering all of the continuous values as potential split-points, removes some by merging neighborhood values to form intervals, and then recursively applies this process to the resulting intervals.

Data discretization and concept hierarchy generation are also forms of data reduction. The raw data are replaced by a smaller number of interval or concept labels. This simplifies the original data and makes the mining more efficient. The resulting patterns mined are typically easier to understand. Concept hierarchies are also useful for mining at multiple abstraction levels.

The rest of this section is organized as follows. First, normalization techniques are presented in Section 3.5.2. We then describe several techniques for data discretization, each of which can be used to generate concept hierarchies for numeric attributes. The techniques include *binning* (Section 3.5.3) and *histogram analysis* (Section 3.5.4), as well as *cluster analysis*, *decision tree analysis*, and *correlation analysis* (Section 3.5.5). Finally, Section 3.5.6 describes the automatic generation of concept hierarchies for nominal data.

3.5.2 Data Transformation by Normalization

The measurement unit used can affect the data analysis. For example, changing measurement units from meters to inches for *height*, or from kilograms to pounds for *weight*, may lead to very different results. In general, expressing an attribute in smaller units will lead to a larger range for that attribute, and thus tend to give such an attribute greater effect or "weight." To help avoid dependence on the choice of measurement units, the data should be *normalized* or *standardized*. This involves transforming the data to fall within a smaller or common range such as $[-1, 1]$ or $[0.0, 1.0]$. (The terms *standardize* and *normalize* are used interchangeably in data preprocessing, although in statistics, the latter term also has other connotations.)

Normalizing the data attempts to give all attributes an equal weight. Normalization is particularly useful for classification algorithms involving neural networks or distance measurements such as nearest-neighbor classification and clustering. If using the neural network backpropagation algorithm for classification mining (Chapter 9), normalizing the input values for each attribute measured in the training tuples will help speed up the learning phase. For distance-based methods, normalization helps prevent

attributes with initially large ranges (e.g., *income*) from outweighing attributes with initially smaller ranges (e.g., binary attributes). It is also useful when given no prior knowledge of the data.

There are many methods for data normalization. We study *min-max normalization, z-score normalization,* and *normalization by decimal scaling.* For our discussion, let A be a numeric attribute with n observed values, $v_1, v_2, \ldots, v_n$.

Min-max normalization performs a linear transformation on the original data. Suppose that min_A and max_A are the minimum and maximum values of an attribute, A. Min-max normalization maps a value, v_i, of A to v'_i in the range $[new\_min_A, new\_max_A]$ by computing

$$v'_i = \frac{v_i - min_A}{max_A - min_A}(new\_max_A - new\_min_A) + new\_min_A. \tag{3.8}$$

Min-max normalization preserves the relationships among the original data values. It will encounter an "out-of-bounds" error if a future input case for normalization falls outside of the original data range for A.

Example 3.4 **Min-max normalization.** Suppose that the minimum and maximum values for the attribute *income* are \$12,000 and \$98,000, respectively. We would like to map *income* to the range $[0.0, 1.0]$. By min-max normalization, a value of \$73,600 for *income* is transformed to $\frac{73{,}600 - 12{,}000}{98{,}000 - 12{,}000}(1.0 - 0) + 0 = 0.716$. ■

In ***z*-score normalization** (or *zero-mean normalization*), the values for an attribute, A, are normalized based on the mean (i.e., average) and standard deviation of A. A value, v_i, of A is normalized to v'_i by computing

$$v'_i = \frac{v_i - \bar{A}}{\sigma_A}, \tag{3.9}$$

where $\bar{A}$ and σ_A are the mean and standard deviation, respectively, of attribute A. The mean and standard deviation were discussed in Section 2.2, where $\bar{A} = \frac{1}{n}(v_1 + v_2 + \cdots + v_n)$ and σ_A is computed as the square root of the variance of A (see Eq. (2.6)). This method of normalization is useful when the actual minimum and maximum of attribute A are unknown, or when there are outliers that dominate the min-max normalization.

Example 3.5 **z-score normalization.** Suppose that the mean and standard deviation of the values for the attribute *income* are \$54,000 and \$16,000, respectively. With z-score normalization, a value of \$73,600 for *income* is transformed to $\frac{73{,}600 - 54{,}000}{16{,}000} = 1.225$. ■

A variation of this z-score normalization replaces the standard deviation of Eq. (3.9) by the *mean absolute deviation* of A. The *mean absolute deviation* of A, denoted s_A, is

$$s_A = \frac{1}{n}(|v_1 - \bar{A}| + |v_2 - \bar{A}| + \cdots + |v_n - \bar{A}|). \tag{3.10}$$

Thus, z-score normalization using the mean absolute deviation is

$$v_i' = \frac{v_i - \bar{A}}{s_A}. \tag{3.11}$$

The mean absolute deviation, s_A, is more robust to outliers than the standard deviation, σ_A. When computing the mean absolute deviation, the deviations from the mean (i.e., $|x_i - \bar{x}|$) are not squared; hence, the effect of outliers is somewhat reduced.

Normalization by decimal scaling normalizes by moving the decimal point of values of attribute A. The number of decimal points moved depends on the maximum absolute value of A. A value, v_i, of A is normalized to v_i' by computing

$$v_i' = \frac{v_i}{10^j}, \tag{3.12}$$

where j is the smallest integer such that $max(|v_i'|) < 1$.

Example 3.6 **Decimal scaling.** Suppose that the recorded values of A range from -986 to 917. The maximum absolute value of A is 986. To normalize by decimal scaling, we therefore divide each value by 1000 (i.e., $j = 3$) so that -986 normalizes to -0.986 and 917 normalizes to 0.917. ■

Note that normalization can change the original data quite a bit, especially when using z-score normalization or decimal scaling. It is also necessary to save the normalization parameters (e.g., the mean and standard deviation if using z-score normalization) so that future data can be normalized in a uniform manner.

3.5.3 Discretization by Binning

Binning is a top-down splitting technique based on a specified number of bins. Section 3.2.2 discussed binning methods for data smoothing. These methods are also used as discretization methods for data reduction and concept hierarchy generation. For example, attribute values can be discretized by applying equal-width or equal-frequency binning, and then replacing each bin value by the bin mean or median, as in *smoothing by bin means* or *smoothing by bin medians*, respectively. These techniques can be applied recursively to the resulting partitions to generate concept hierarchies.

Binning does not use class information and is therefore an unsupervised discretization technique. It is sensitive to the user-specified number of bins, as well as the presence of outliers.

3.5.4 Discretization by Histogram Analysis

Like binning, histogram analysis is an unsupervised discretization technique because it does not use class information. Histograms were introduced in Section 2.2.3. A histogram partitions the values of an attribute, A, into disjoint ranges called *buckets* or *bins*.

Various partitioning rules can be used to define histograms (Section 3.4.6). In an *equal-width* histogram, for example, the values are partitioned into equal-size partitions or ranges (e.g., earlier in Figure 3.8 for *price*, where each bucket has a width of $10). With an *equal-frequency* histogram, the values are partitioned so that, ideally, each partition contains the same number of data tuples. The histogram analysis algorithm can be applied recursively to each partition in order to automatically generate a multilevel concept hierarchy, with the procedure terminating once a prespecified number of concept levels has been reached. A *minimum interval size* can also be used per level to control the recursive procedure. This specifies the minimum width of a partition, or the minimum number of values for each partition at each level. Histograms can also be partitioned based on cluster analysis of the data distribution, as described next.

3.5.5 Discretization by Cluster, Decision Tree, and Correlation Analyses

Clustering, decision tree analysis, and correlation analysis can be used for data discretization. We briefly study each of these approaches.

Cluster analysis is a popular data discretization method. A clustering algorithm can be applied to discretize a numeric attribute, *A*, by partitioning the values of *A* into clusters or groups. Clustering takes the distribution of *A* into consideration, as well as the closeness of data points, and therefore is able to produce high-quality discretization results.

Clustering can be used to generate a concept hierarchy for *A* by following either a top-down splitting strategy or a bottom-up merging strategy, where each cluster forms a node of the concept hierarchy. In the former, each initial cluster or partition may be further decomposed into several subclusters, forming a lower level of the hierarchy. In the latter, clusters are formed by repeatedly grouping neighboring clusters in order to form higher-level concepts. Clustering methods for data mining are studied in Chapters 10 and 11.

Techniques to generate decision trees for classification (Chapter 8) can be applied to discretization. Such techniques employ a top-down splitting approach. Unlike the other methods mentioned so far, decision tree approaches to discretization are supervised, that is, they make use of class label information. For example, we may have a data set of patient symptoms (the attributes) where each patient has an associated *diagnosis* class label. Class distribution information is used in the calculation and determination of split-points (data values for partitioning an attribute range). Intuitively, the main idea is to select split-points so that a given resulting partition contains as many tuples of the same class as possible. *Entropy* is the most commonly used measure for this purpose. To discretize a numeric attribute, *A*, the method selects the value of *A* that has the minimum entropy as a split-point, and recursively partitions the resulting intervals to arrive at a hierarchical discretization. Such discretization forms a concept hierarchy for *A*.

Because decision tree–based discretization uses class information, it is more likely that the interval boundaries (split-points) are defined to occur in places that may help improve classification accuracy. Decision trees and the entropy measure are described in greater detail in Section 8.2.2.

Measures of correlation can be used for discretization. *ChiMerge* is a χ^2-based discretization method. The discretization methods that we have studied up to this point have all employed a top-down, splitting strategy. This contrasts with ChiMerge, which employs a bottom-up approach by finding the best neighboring intervals and then merging them to form larger intervals, recursively. As with decision tree analysis, ChiMerge is supervised in that it uses class information. The basic notion is that for accurate discretization, the relative class frequencies should be fairly consistent within an interval. Therefore, if two adjacent intervals have a very similar distribution of classes, then the intervals can be merged. Otherwise, they should remain separate.

ChiMerge proceeds as follows. Initially, each distinct value of a numeric attribute A is considered to be one interval. χ^2 tests are performed for every pair of adjacent intervals. Adjacent intervals with the least χ^2 values are merged together, because low χ^2 values for a pair indicate similar class distributions. This merging process proceeds recursively until a predefined stopping criterion is met.

3.5.6 Concept Hierarchy Generation for Nominal Data

We now look at data transformation for nominal data. In particular, we study concept hierarchy generation for nominal attributes. Nominal attributes have a finite (but possibly large) number of distinct values, with no ordering among the values. Examples include *geographic_location*, *job_category*, and *item_type*.

Manual definition of concept hierarchies can be a tedious and time-consuming task for a user or a domain expert. Fortunately, many hierarchies are implicit within the database schema and can be automatically defined at the schema definition level. The concept hierarchies can be used to transform the data into multiple levels of granularity. For example, data mining patterns regarding sales may be found relating to specific regions or countries, in addition to individual branch locations.

We study four methods for the generation of concept hierarchies for nominal data, as follows.

1. **Specification of a partial ordering of attributes explicitly at the schema level by users or experts:** Concept hierarchies for nominal attributes or dimensions typically involve a group of attributes. A user or expert can easily define a concept hierarchy by specifying a partial or total ordering of the attributes at the schema level. For example, suppose that a relational database contains the following group of attributes: *street, city, province_or_state*, and *country*. Similarly, a data warehouse *location* dimension may contain the same attributes. A hierarchy can be defined by specifying the total ordering among these attributes at the schema level such as *street* $<$ *city* $<$ *province_or_state* $<$ *country*.

2. **Specification of a portion of a hierarchy by explicit data grouping:** This is essentially the manual definition of a portion of a concept hierarchy. In a large database, it is unrealistic to define an entire concept hierarchy by explicit value enumeration. On the contrary, we can easily specify explicit groupings for a small portion of intermediate-level data. For example, after specifying that *province* and *country*

form a hierarchy at the schema level, a user could define some intermediate levels manually, such as "{*Alberta, Saskatchewan, Manitoba*} $\subset$ *prairies_Canada*" and "{*British Columbia, prairies_Canada*} $\subset$ *Western_Canada*."

3. **Specification of a *set of attributes*, but not of their partial ordering:** A user may specify a set of attributes forming a concept hierarchy, but omit to explicitly state their partial ordering. The system can then try to automatically generate the attribute ordering so as to construct a meaningful concept hierarchy.

   "*Without knowledge of data semantics, how can a hierarchical ordering for an arbitrary set of nominal attributes be found?*" Consider the observation that since higher-level concepts generally cover several subordinate lower-level concepts, an attribute defining a high concept level (e.g., *country*) will usually contain a smaller number of distinct values than an attribute defining a lower concept level (e.g., *street*). Based on this observation, a concept hierarchy can be automatically generated based on the number of distinct values per attribute in the given attribute set. The attribute with the most distinct values is placed at the lowest hierarchy level. The lower the number of distinct values an attribute has, the higher it is in the generated concept hierarchy. This heuristic rule works well in many cases. Some local-level swapping or adjustments may be applied by users or experts, when necessary, after examination of the generated hierarchy.

Let's examine an example of this third method.

Example 3.7 Concept hierarchy generation based on the number of distinct values per attribute. Suppose a user selects a set of location-oriented attributes—*street, country, province_or_state*, and *city*—from the *AllElectronics* database, but does not specify the hierarchical ordering among the attributes.

A concept hierarchy for *location* can be generated automatically, as illustrated in Figure 3.13. First, sort the attributes in ascending order based on the number of distinct values in each attribute. This results in the following (where the number of distinct values per attribute is shown in parentheses): *country* (15), *province_or_state* (365), *city* (3567), and *street* (674,339). Second, generate the hierarchy from the top down according to the sorted order, with the first attribute at the top level and the last attribute at the bottom level. Finally, the user can examine the generated hierarchy, and when necessary, modify it to reflect desired semantic relationships among the attributes. In this example, it is obvious that there is no need to modify the generated hierarchy. ■

Note that this heuristic rule is not foolproof. For example, a time dimension in a database may contain 20 distinct years, 12 distinct months, and 7 distinct days of the week. However, this does not suggest that the time hierarchy should be "*year* < *month* < *days_of_the_week*," with *days_of_the_week* at the top of the hierarchy.

4. **Specification of only a partial set of attributes:** Sometimes a user can be careless when defining a hierarchy, or have only a vague idea about what should be included in a hierarchy. Consequently, the user may have included only a small subset of the

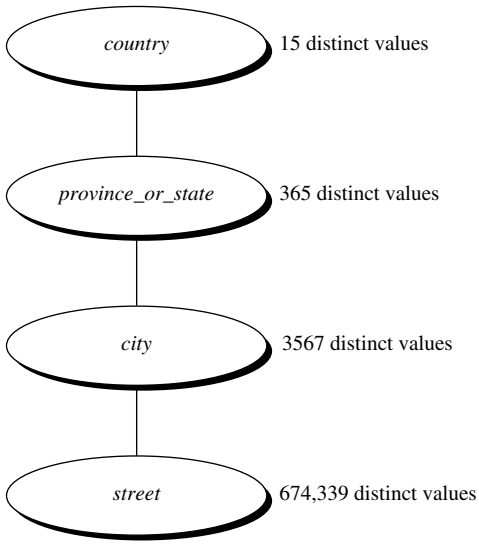


Figure 3.13 Automatic generation of a schema concept hierarchy based on the number of distinct attribute values.

relevant attributes in the hierarchy specification. For example, instead of including all of the hierarchically relevant attributes for *location*, the user may have specified only *street* and *city*. To handle such partially specified hierarchies, it is important to embed data semantics in the database schema so that attributes with tight semantic connections can be pinned together. In this way, the specification of one attribute may trigger a whole group of semantically tightly linked attributes to be "dragged in" to form a complete hierarchy. Users, however, should have the option to override this feature, as necessary.

Example 3.8 **Concept hierarchy generation using prespecified semantic connections.** Suppose that a data mining expert (serving as an administrator) has pinned together the five attributes *number, street, city, province_or_state*, and *country*, because they are closely linked semantically regarding the notion of *location*. If a user were to specify only the attribute *city* for a hierarchy defining *location*, the system can automatically drag in all five semantically related attributes to form a hierarchy. The user may choose to drop any of these attributes (e.g., *number* and *street*) from the hierarchy, keeping *city* as the lowest conceptual level. ■

In summary, information at the schema level and on attribute–value counts can be used to generate concept hierarchies for nominal data. Transforming nominal data with the use of concept hierarchies allows higher-level knowledge patterns to be found. It allows mining at multiple levels of abstraction, which is a common requirement for data mining applications.

3.6 Summary

- **Data quality** is defined in terms of *accuracy, completeness, consistency, timeliness, believability*, and *interpretabilty*. These qualities are assessed based on the intended use of the data.
- **Data cleaning** routines attempt to fill in missing values, smooth out noise while identifying outliers, and correct inconsistencies in the data. Data cleaning is usually performed as an iterative two-step process consisting of discrepancy detection and data transformation.
- **Data integration** combines data from multiple sources to form a coherent data store. The resolution of semantic heterogeneity, metadata, correlation analysis, tuple duplication detection, and data conflict detection contribute to smooth data integration.
- **Data reduction** techniques obtain a reduced representation of the data while minimizing the loss of information content. These include methods of *dimensionality reduction*, *numerosity reduction*, and *data compression*. **Dimensionality reduction** reduces the number of random variables or attributes under consideration. Methods include *wavelet transforms, principal components analysis, attribute subset selection*, and *attribute creation*. **Numerosity reduction** methods use parametric or nonparatmetric models to obtain smaller representations of the original data. Parametric models store only the model parameters instead of the actual data. Examples include regression and log-linear models. Nonparamteric methods include histograms, clustering, sampling, and data cube aggregation. **Data compression** methods apply transformations to obtain a reduced or "compressed" representation of the original data. The data reduction is *lossless* if the original data can be reconstructed from the compressed data without any loss of information; otherwise, it is *lossy*.
- **Data transformation** routines convert the data into appropriate forms for mining. For example, in **normalization**, attribute data are scaled so as to fall within a small range such as 0.0 to 1.0. Other examples are **data discretization** and **concept hierarchy generation**.
- **Data discretization** transforms numeric data by mapping values to interval or concept labels. Such methods can be used to automatically generate *concept hierarchies* for the data, which allows for mining at multiple levels of granularity. Discretization techniques include binning, histogram analysis, cluster analysis, decision tree analysis, and correlation analysis. For nominal data, **concept hierarchies** may be generated based on schema definitions as well as the number of distinct values per attribute.
- Although numerous methods of data preprocessing have been developed, data preprocessing remains an active area of research, due to the huge amount of inconsistent or dirty data and the complexity of the problem.

3.7 Exercises

3.1 *Data quality* can be assessed in terms of several issues, including accuracy, completeness, and consistency. For each of the above three issues, discuss how data quality assessment can depend on the *intended use* of the data, giving examples. Propose two other dimensions of data quality.

3.2 In real-world data, tuples with *missing values* for some attributes are a common occurrence. Describe various methods for handling this problem.

3.3 Exercise 2.2 gave the following data (in increasing order) for the attribute *age*: 13, 15, 16, 16, 19, 20, 20, 21, 22, 22, 25, 25, 25, 25, 30, 33, 33, 35, 35, 35, 35, 36, 40, 45, 46, 52, 70.

(a) Use *smoothing by bin means* to smooth these data, using a bin depth of 3. Illustrate your steps. Comment on the effect of this technique for the given data.

(b) How might you determine *outliers* in the data?

(c) What other methods are there for *data smoothing*?

3.4 Discuss issues to consider during *data integration*.

3.5 What are the value ranges of the following *normalization methods*?

(a) min-max normalization

(b) z-score normalization

(c) z-score normalization using the mean absolute deviation instead of standard deviation

(d) normalization by decimal scaling

3.6 Use these methods to *normalize* the following group of data:

$$200, 300, 400, 600, 1000$$

(a) min-max normalization by setting $min = 0$ and $max = 1$

(b) z-score normalization

(c) z-score normalization using the mean absolute deviation instead of standard deviation

(d) normalization by decimal scaling

3.7 Using the data for *age* given in Exercise 3.3, answer the following:

(a) Use min-max normalization to transform the value 35 for *age* onto the range $[0.0, 1.0]$.

(b) Use z-score normalization to transform the value 35 for *age*, where the standard deviation of *age* is 12.94 years.

(c) Use normalization by decimal scaling to transform the value 35 for *age*.

(d) Comment on which method you would prefer to use for the given data, giving reasons as to why.

3.8 Using the data for *age* and *body fat* given in Exercise 2.4, answer the following:

(a) Normalize the two attributes based on *z-score normalization*.

(b) Calculate the *correlation coefficient* (Pearson's product moment coefficient). Are these two attributes positively or negatively correlated? Compute their covariance.

3.9 Suppose a group of 12 *sales price* records has been sorted as follows:

$$5, 10, 11, 13, 15, 35, 50, 55, 72, 92, 204, 215.$$

Partition them into three bins by each of the following methods:

(a) equal-frequency (equal-depth) partitioning

(b) equal-width partitioning

(c) clustering

3.10 Use a flowchart to summarize the following procedures for *attribute subset selection*:

(a) stepwise forward selection

(b) stepwise backward elimination

(c) a combination of forward selection and backward elimination

3.11 Using the data for *age* given in Exercise 3.3,

(a) Plot an equal-width histogram of width 10.

(b) Sketch examples of each of the following sampling techniques: SRSWOR, SRSWR, cluster sampling, and stratified sampling. Use samples of size 5 and the strata "youth," "middle-aged," and "senior."

3.12 ChiMerge [Ker92] is a supervised, bottom-up (i.e., merge-based) *data discretization* method. It relies on χ^2 analysis: Adjacent intervals with the least χ^2 values are merged together until the chosen stopping criterion satisfies.

(a) Briefly describe how ChiMerge works.

(b) Take the IRIS data set, obtained from the University of California–Irvine Machine Learning Data Repository (*www.ics.uci.edu/∼mlearn/MLRepository.html*), as a data set to be discretized. Perform data discretization for each of the four numeric attributes using the ChiMerge method. (Let the stopping criteria be: *max-interval* = 6). You need to write a small program to do this to avoid clumsy numerical computation. Submit your simple analysis and your test results: split-points, final intervals, and the documented source program.

3.13 Propose an algorithm, in pseudocode or in your favorite programming language, for the following:

(a) The automatic generation of a concept hierarchy for nominal data based on the number of distinct values of attributes in the given schema.

(b) The automatic generation of a concept hierarchy for numeric data based on the *equal-width* partitioning rule.

(c) The automatic generation of a concept hierarchy for numeric data based on the *equal-frequency* partitioning rule.

3.14 Robust data loading poses a challenge in database systems because the input data are often dirty. In many cases, an input record may miss multiple values; some records could be *contaminated*, with some data values out of range or of a different data type than expected. Work out an automated *data cleaning and loading* algorithm so that the erroneous data will be marked and contaminated data will not be mistakenly inserted into the database during data loading.

3.8 Bibliographic Notes

Data preprocessing is discussed in a number of textbooks, including English [Eng99], Pyle [Pyl99], Loshin [Los01], Redman [Red01], and Dasu and Johnson [DJ03]. More specific references to individual preprocessing techniques are given later.

For discussion regarding data quality, see Redman [Red92]; Wang, Storey, and Firth [WSF95]; Wand and Wang [WW96]; Ballou and Tayi [BT99]; and Olson [Ols03]. Potter's Wheel (*control.cx.berkely.edu/abc*), the interactive data cleaning tool described in Section 3.2.3, is presented in Raman and Hellerstein [RH01]. An example of the development of declarative languages for the specification of data transformation operators is given in Galhardas et al. [GFS+01]. The handling of missing attribute values is discussed in Friedman [Fri77]; Breiman, Friedman, Olshen, and Stone [BFOS84]; and Quinlan [Qui89]. Hua and Pei [HP07] presented a heuristic approach to cleaning *disguised missing data*, where such data are captured when users falsely select default values on forms (e.g., "January 1" for *birthdate*) when they do not want to disclose personal information.

A method for the detection of outlier or "garbage" patterns in a handwritten character database is given in Guyon, Matic, and Vapnik [GMV96]. Binning and data normalization are treated in many texts, including Kennedy et al. [KLV+98], Weiss and Indurkhya [WI98], and Pyle [Pyl99]. Systems that include attribute (or feature) construction include BACON by Langley, Simon, Bradshaw, and Zytkow [LSBZ87]; Stagger by Schlimmer [Sch86]; FRINGE by Pagallo [Pag89]; and AQ17-DCI by Bloedorn and Michalski [BM98]. Attribute construction is also described in Liu and Motoda [LM98a, LM98b]. Dasu et al. built a BELLMAN system and proposed a set of interesting methods for building a data quality browser by mining database structures [DJMS02].

A good survey of data reduction techniques can be found in Barbará et al. [BDF+97]. For algorithms on data cubes and their precomputation, see Sarawagi and Stonebraker [SS94]; Agarwal et al. [AAD+96]; Harinarayan, Rajaraman, and Ullman [HRU96]; Ross and Srivastava [RS97]; and Zhao, Deshpande, and Naughton [ZDN97]. Attribute subset selection (or *feature subset selection*) is described in many texts such as Neter, Kutner, Nachtsheim, and Wasserman [NKNW96]; Dash and Liu [DL97]; and Liu and Motoda [LM98a, LM98b]. A combination forward selection and backward elimination method

was proposed in Siedlecki and Sklansky [SS88]. A wrapper approach to attribute selection is described in Kohavi and John [KJ97]. Unsupervised attribute subset selection is described in Dash, Liu, and Yao [DLY97].

For a description of wavelets for dimensionality reduction, see Press, Teukolosky, Vetterling, and Flannery [PTVF07]. A general account of wavelets can be found in Hubbard [Hub96]. For a list of wavelet software packages, see Bruce, Donoho, and Gao [BDG96]. Daubechies transforms are described in Daubechies [Dau92]. The book by Press et al. [PTVF07] includes an introduction to singular value decomposition for principal components analysis. Routines for PCA are included in most statistical software packages such as SAS (*www.sas.com/SASHome.html*).

An introduction to regression and log-linear models can be found in several textbooks such as James [Jam85]; Dobson [Dob90]; Johnson and Wichern [JW92]; Devore [Dev95]; and Neter, Kutner, Nachtsheim, and Wasserman [NKNW96]. For log-linear models (known as *multiplicative models* in the computer science literature), see Pearl [Pea88]. For a general introduction to histograms, see Barbará et al. [BDF+97] and Devore and Peck [DP97]. For extensions of single-attribute histograms to multiple attributes, see Muralikrishna and DeWitt [MD88] and Poosala and Ioannidis [PI97]. Several references to clustering algorithms are given in Chapters 10 and 11 of this book, which are devoted to the topic.

A survey of multidimensional indexing structures is given in Gaede and Günther [GG98]. The use of multidimensional index trees for data aggregation is discussed in Aoki [Aok98]. Index trees include R-trees (Guttman [Gut84]), quad-trees (Finkel and Bentley [FB74]), and their variations. For discussion on sampling and data mining, see Kivinen and Mannila [KM94] and John and Langley [JL96].

There are many methods for assessing attribute relevance. Each has its own bias. The information gain measure is biased toward attributes with many values. Many alternatives have been proposed, such as gain ratio (Quinlan [Qui93]), which considers the probability of each attribute value. Other relevance measures include the Gini index (Breiman, Friedman, Olshen, and Stone [BFOS84]), the χ^2 contingency table statistic, and the uncertainty coefficient (Johnson and Wichern [JW92]). For a comparison of attribute selection measures for decision tree induction, see Buntine and Niblett [BN92]. For additional methods, see Liu and Motoda [LM98a], Dash and Liu [DL97], and Almuallim and Dietterich [AD91].

Liu et al. [LHTD02] performed a comprehensive survey of data discretization methods. Entropy-based discretization with the C4.5 algorithm is described in Quinlan [Qui93]. In Catlett [Cat91], the D-2 system binarizes a numeric feature recursively. ChiMerge by Kerber [Ker92] and Chi2 by Liu and Setiono [LS95] are methods for the automatic discretization of numeric attributes that both employ the χ^2 statistic. Fayyad and Irani [FI93] apply the minimum description length principle to determine the number of intervals for numeric discretization. Concept hierarchies and their automatic generation from categorical data are described in Han and Fu [HF94].

4 Data Warehousing and Online Analytical Processing

Data warehouses generalize and consolidate data in multidimensional space. The construction of data warehouses involves data cleaning, data integration, and data transformation, and can be viewed as an important preprocessing step for data mining. Moreover, data warehouses provide *online analytical processing (OLAP)* tools for the interactive analysis of multidimensional data of varied granularities, which facilitates effective data generalization and data mining. Many other data mining functions, such as association, classification, prediction, and clustering, can be integrated with OLAP operations to enhance interactive mining of knowledge at multiple levels of abstraction. Hence, the data warehouse has become an increasingly important platform for data analysis and OLAP and will provide an effective platform for data mining. Therefore, data warehousing and OLAP form an essential step in the knowledge discovery process. This chapter presents an overview of data warehouse and OLAP technology. This overview is essential for understanding the overall data mining and knowledge discovery process.

In this chapter, we study a well-accepted definition of the data warehouse and see why more and more organizations are building data warehouses for the analysis of their data (Section 4.1). In particular, we study the *data cube*, a multidimensional data model for data warehouses and OLAP, as well as OLAP operations such as roll-up, drill-down, slicing, and dicing (Section 4.2). We also look at data warehouse design and usage (Section 4.3). In addition, we discuss *multidimensional data mining*, a powerful paradigm that integrates data warehouse and OLAP technology with that of data mining. An overview of data warehouse implementation examines general strategies for efficient data cube computation, OLAP data indexing, and OLAP query processing (Section 4.4). Finally, we study data generalization by attribute-oriented induction (Section 4.5). This method uses concept hierarchies to generalize data to multiple levels of abstraction.

4.1 Data Warehouse: Basic Concepts

This section gives an introduction to data warehouses. We begin with a definition of the data warehouse (Section 4.1.1). We outline the differences between operational database

systems and data warehouses (Section 4.1.2), then explain the need for using data warehouses for data analysis, rather than performing the analysis directly on traditional databases (Section 4.1.3). This is followed by a presentation of data warehouse architecture (Section 4.1.4). Next, we study three data warehouse models—an enterprise model, a data mart, and a virtual warehouse (Section 4.1.5). Section 4.1.6 describes back-end utilities for data warehousing, such as extraction, transformation, and loading. Finally, Section 4.1.7 presents the metadata repository, which stores data about data.

4.1.1 What Is a Data Warehouse?

Data warehousing provides architectures and tools for business executives to systematically organize, understand, and use their data to make strategic decisions. Data warehouse systems are valuable tools in today's competitive, fast-evolving world. In the last several years, many firms have spent millions of dollars in building enterprise-wide data warehouses. Many people feel that with competition mounting in every industry, data warehousing is the latest must-have marketing weapon—a way to retain customers by learning more about their needs.

"*Then, what exactly is a data warehouse?*" Data warehouses have been defined in many ways, making it difficult to formulate a rigorous definition. Loosely speaking, a data warehouse refers to a data repository that is maintained separately from an organization's operational databases. Data warehouse systems allow for integration of a variety of application systems. They support information processing by providing a solid platform of consolidated historic data for analysis.

According to William H. Inmon, a leading architect in the construction of data warehouse systems, "A data warehouse is a subject-oriented, integrated, time-variant, and nonvolatile collection of data in support of management's decision making process" [Inm96]. This short but comprehensive definition presents the major features of a data warehouse. The four keywords—*subject-oriented, integrated, time-variant*, and *nonvolatile*—distinguish data warehouses from other data repository systems, such as relational database systems, transaction processing systems, and file systems.

Let's take a closer look at each of these key features.

- **Subject-oriented**: A data warehouse is organized around major subjects such as customer, supplier, product, and sales. Rather than concentrating on the day-to-day operations and transaction processing of an organization, a data warehouse focuses on the modeling and analysis of data for decision makers. Hence, data warehouses typically provide a simple and concise view of particular subject issues by excluding data that are not useful in the decision support process.
- **Integrated**: A data warehouse is usually constructed by integrating multiple heterogeneous sources, such as relational databases, flat files, and online transaction records. Data cleaning and data integration techniques are applied to ensure consistency in naming conventions, encoding structures, attribute measures, and so on.

- **Time-variant**: Data are stored to provide information from an historic perspective (e.g., the past 5–10 years). Every key structure in the data warehouse contains, either implicitly or explicitly, a time element.
- **Nonvolatile**: A data warehouse is always a physically separate store of data transformed from the application data found in the operational environment. Due to this separation, a data warehouse does not require transaction processing, recovery, and concurrency control mechanisms. It usually requires only two operations in data accessing: *initial loading of data* and *access of data*.

In sum, a data warehouse is a semantically consistent data store that serves as a physical implementation of a decision support data model. It stores the information an enterprise needs to make strategic decisions. A data warehouse is also often viewed as an architecture, constructed by integrating data from multiple heterogeneous sources to support structured and/or ad hoc queries, analytical reporting, and decision making.

Based on this information, we view **data warehousing** as the process of constructing and using data warehouses. The construction of a data warehouse requires data cleaning, data integration, and data consolidation. The utilization of a data warehouse often necessitates a collection of *decision support* technologies. This allows "knowledge workers" (e.g., managers, analysts, and executives) to use the warehouse to quickly and conveniently obtain an overview of the data, and to make sound decisions based on information in the warehouse. Some authors use the term *data warehousing* to refer only to the process of data warehouse *construction*, while the term *warehouse DBMS* is used to refer to the *management and utilization* of data warehouses. We will not make this distinction here.

"*How are organizations using the information from data warehouses?*" Many organizations use this information to support business decision-making activities, including (1) increasing customer focus, which includes the analysis of customer buying patterns (such as buying preference, buying time, budget cycles, and appetites for spending); (2) repositioning products and managing product portfolios by comparing the performance of sales by quarter, by year, and by geographic regions in order to fine-tune production strategies; (3) analyzing operations and looking for sources of profit; and (4) managing customer relationships, making environmental corrections, and managing the cost of corporate assets.

Data warehousing is also very useful from the point of view of *heterogeneous database integration*. Organizations typically collect diverse kinds of data and maintain large databases from multiple, heterogeneous, autonomous, and distributed information sources. It is highly desirable, yet challenging, to integrate such data and provide easy and efficient access to it. Much effort has been spent in the database industry and research community toward achieving this goal.

The traditional database approach to heterogeneous database integration is to build **wrappers** and **integrators** (or **mediators**) on top of multiple, heterogeneous databases. When a query is posed to a client site, a metadata dictionary is used to translate the query into queries appropriate for the individual heterogeneous sites involved. These

queries are then mapped and sent to local query processors. The results returned from the different sites are integrated into a global answer set. This **query-driven approach** requires complex information filtering and integration processes, and competes with local sites for processing resources. It is inefficient and potentially expensive for frequent queries, especially queries requiring aggregations.

Data warehousing provides an interesting alternative to this traditional approach. Rather than using a query-driven approach, data warehousing employs an **update-driven** approach in which information from multiple, heterogeneous sources is integrated in advance and stored in a warehouse for direct querying and analysis. Unlike online transaction processing databases, data warehouses do not contain the most current information. However, a data warehouse brings high performance to the integrated heterogeneous database system because data are copied, preprocessed, integrated, annotated, summarized, and restructured into one semantic data store. Furthermore, query processing in data warehouses does not interfere with the processing at local sources. Moreover, data warehouses can store and integrate historic information and support complex multidimensional queries. As a result, data warehousing has become popular in industry.

4.1.2 Differences between Operational Database Systems and Data Warehouses

Because most people are familiar with commercial relational database systems, it is easy to understand what a data warehouse is by comparing these two kinds of systems.

The major task of online operational database systems is to perform online transaction and query processing. These systems are called **online transaction processing (OLTP)** systems. They cover most of the day-to-day operations of an organization such as purchasing, inventory, manufacturing, banking, payroll, registration, and accounting. Data warehouse systems, on the other hand, serve users or knowledge workers in the role of data analysis and decision making. Such systems can organize and present data in various formats in order to accommodate the diverse needs of different users. These systems are known as **online analytical processing (OLAP)** systems.

The major distinguishing features of OLTP and OLAP are summarized as follows:

- **Users and system orientation**: An OLTP system is *customer-oriented* and is used for transaction and query processing by clerks, clients, and information technology professionals. An OLAP system is *market-oriented* and is used for data analysis by knowledge workers, including managers, executives, and analysts.
- **Data contents**: An OLTP system manages current data that, typically, are too detailed to be easily used for decision making. An OLAP system manages large amounts of historic data, provides facilities for summarization and aggregation, and stores and manages information at different levels of granularity. These features make the data easier to use for informed decision making.

- **Database design**: An OLTP system usually adopts an entity-relationship (ER) data model and an application-oriented database design. An OLAP system typically adopts either a *star* or a *snowflake* model (see Section 4.2.2) and a subject-oriented database design.
- **View**: An OLTP system focuses mainly on the current data within an enterprise or department, without referring to historic data or data in different organizations. In contrast, an OLAP system often spans multiple versions of a database schema, due to the evolutionary process of an organization. OLAP systems also deal with information that originates from different organizations, integrating information from many data stores. Because of their huge volume, OLAP data are stored on multiple storage media.
- **Access patterns**: The access patterns of an OLTP system consist mainly of short, atomic transactions. Such a system requires concurrency control and recovery mechanisms. However, accesses to OLAP systems are mostly read-only operations (because most data warehouses store historic rather than up-to-date information), although many could be complex queries.

Other features that distinguish between OLTP and OLAP systems include database size, frequency of operations, and performance metrics. These are summarized in Table 4.1.

4.1.3 But, Why Have a Separate Data Warehouse?

Because operational databases store huge amounts of data, you may wonder, "*Why not perform online analytical processing directly on such databases instead of spending additional time and resources to construct a separate data warehouse?*" A major reason for such a separation is to help promote the *high performance of both systems*. An operational database is designed and tuned from known tasks and workloads like indexing and hashing using primary keys, searching for particular records, and optimizing "canned" queries. On the other hand, data warehouse queries are often complex. They involve the computation of large data groups at summarized levels, and may require the use of special data organization, access, and implementation methods based on multidimensional views. Processing OLAP queries in operational databases would substantially degrade the performance of operational tasks.

Moreover, an operational database supports the concurrent processing of multiple transactions. Concurrency control and recovery mechanisms (e.g., locking and logging) are required to ensure the consistency and robustness of transactions. An OLAP query often needs read-only access of data records for summarization and aggregation. Concurrency control and recovery mechanisms, if applied for such OLAP operations, may jeopardize the execution of concurrent transactions and thus substantially reduce the throughput of an OLTP system.

Finally, the separation of operational databases from data warehouses is based on the different structures, contents, and uses of the data in these two systems. Decision

Table 4.1 Comparison of OLTP and OLAP Systems

Feature	*OLTP*	*OLAP*
Characteristic	operational processing	informational processing
Orientation	transaction	analysis
User	clerk, DBA, database professional	knowledge worker (e.g., manager, executive, analyst)
Function	day-to-day operations	long-term informational requirements decision support
DB design	ER-based, application-oriented	star/snowflake, subject-oriented
Data	current, guaranteed up-to-date	historic, accuracy maintained over time
Summarization	primitive, highly detailed	summarized, consolidated
View	detailed, flat relational	summarized, multidimensional
Unit of work	short, simple transaction	complex query
Access	read/write	mostly read
Focus	data in	information out
Operations	index/hash on primary key	lots of scans
Number of records accessed	tens	millions
Number of users	thousands	hundreds
DB size	GB to high-order GB	$\geq$ TB
Priority	high performance, high availability	high flexibility, end-user autonomy
Metric	transaction throughput	query throughput, response time

Note: Table is partially based on Chaudhuri and Dayal [CD97].

support requires historic data, whereas operational databases do not typically maintain historic data. In this context, the data in operational databases, though abundant, are usually far from complete for decision making. Decision support requires consolidation (e.g., aggregation and summarization) of data from heterogeneous sources, resulting in high-quality, clean, integrated data. In contrast, operational databases contain only detailed raw data, such as transactions, which need to be consolidated before analysis. Because the two systems provide quite different functionalities and require different kinds of data, it is presently necessary to maintain separate databases. However, many vendors of operational relational database management systems are beginning to optimize such systems to support OLAP queries. As this trend continues, the separation between OLTP and OLAP systems is expected to decrease.

4.1.4 Data Warehousing: A Multitiered Architecture

Data warehouses often adopt a three-tier architecture, as presented in Figure 4.1.

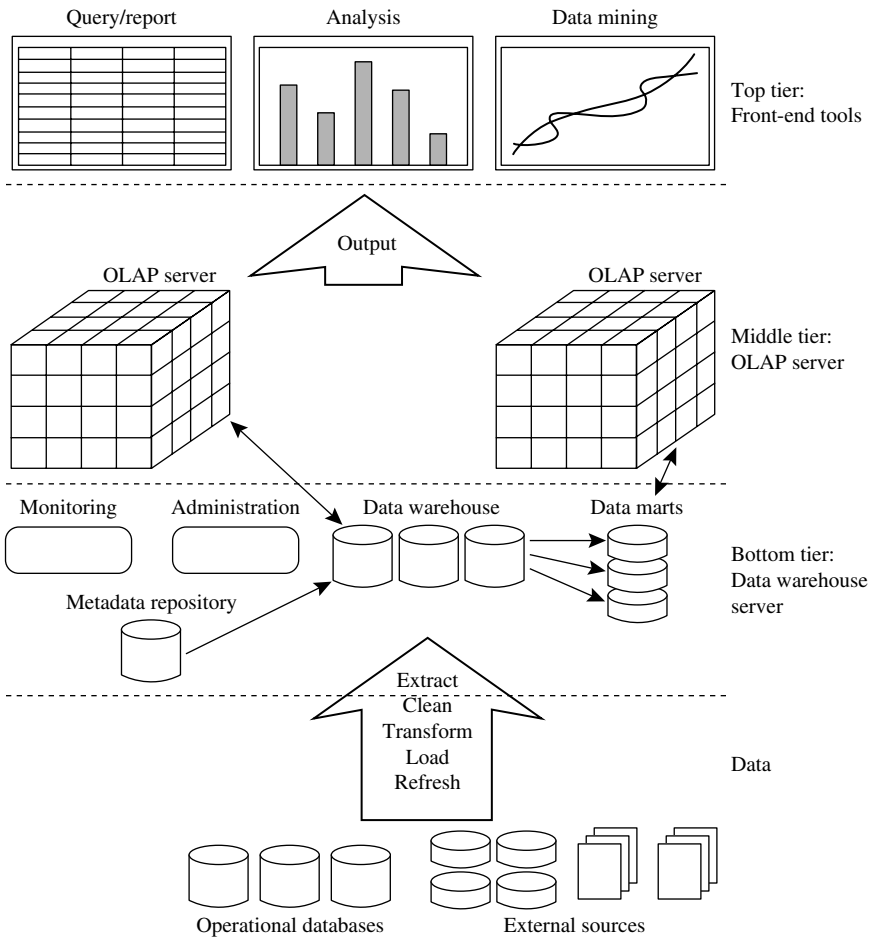


Figure 4.1 A three-tier data warehousing architecture.

1. The bottom tier is a **warehouse database server** that is almost always a relational database system. Back-end tools and utilities are used to feed data into the bottom tier from operational databases or other external sources (e.g., customer profile information provided by external consultants). These tools and utilities perform data extraction, cleaning, and transformation (e.g., to merge similar data from different sources into a unified format), as well as load and refresh functions to update the data warehouse (see Section 4.1.6). The data are extracted using application program interfaces known as **gateways**. A gateway is supported by the underlying DBMS and allows client programs to generate SQL code to be executed at a server. Examples of gateways include ODBC (Open Database Connection) and OLEDB (Object

Linking and Embedding Database) by Microsoft and JDBC (Java Database Connection). This tier also contains a metadata repository, which stores information about the data warehouse and its contents. The metadata repository is further described in Section 4.1.7.

2. The middle tier is an **OLAP server** that is typically implemented using either (1) a **relational OLAP (ROLAP)** model (i.e., an extended relational DBMS that maps operations on multidimensional data to standard relational operations); or (2) a **multidimensional OLAP (MOLAP)** model (i.e., a special-purpose server that directly implements multidimensional data and operations). OLAP servers are discussed in Section 4.4.4.
3. The top tier is a **front-end client layer**, which contains query and reporting tools, analysis tools, and/or data mining tools (e.g., trend analysis, prediction, and so on).

4.1.5 Data Warehouse Models: Enterprise Warehouse, Data Mart, and Virtual Warehouse

From the architecture point of view, there are three data warehouse models: the *enterprise warehouse*, the *data mart*, and the *virtual warehouse*.

Enterprise warehouse: An enterprise warehouse collects all of the information about subjects spanning the entire organization. It provides corporate-wide data integration, usually from one or more operational systems or external information providers, and is cross-functional in scope. It typically contains detailed data as well as summarized data, and can range in size from a few gigabytes to hundreds of gigabytes, terabytes, or beyond. An enterprise data warehouse may be implemented on traditional mainframes, computer superservers, or parallel architecture platforms. It requires extensive business modeling and may take years to design and build.

Data mart: A data mart contains a subset of corporate-wide data that is of value to a specific group of users. The scope is confined to specific selected subjects. For example, a marketing data mart may confine its subjects to customer, item, and sales. The data contained in data marts tend to be summarized.

Data marts are usually implemented on low-cost departmental servers that are Unix/Linux or Windows based. The implementation cycle of a data mart is more likely to be measured in weeks rather than months or years. However, it may involve complex integration in the long run if its design and planning were not enterprise-wide.

Depending on the source of data, data marts can be categorized as independent or dependent. *Independent* data marts are sourced from data captured from one or more operational systems or external information providers, or from data generated locally within a particular department or geographic area. *Dependent* data marts are sourced directly from enterprise data warehouses.

Virtual warehouse: A virtual warehouse is a set of views over operational databases. For efficient query processing, only some of the possible summary views may be materialized. A virtual warehouse is easy to build but requires excess capacity on operational database servers.

"*What are the pros and cons of the top-down and bottom-up approaches to data warehouse development?*" The top-down development of an enterprise warehouse serves as a systematic solution and minimizes integration problems. However, it is expensive, takes a long time to develop, and lacks flexibility due to the difficulty in achieving consistency and consensus for a common data model for the entire organization. The bottom-up approach to the design, development, and deployment of independent data marts provides flexibility, low cost, and rapid return of investment. It, however, can lead to problems when integrating various disparate data marts into a consistent enterprise data warehouse.

A recommended method for the development of data warehouse systems is to implement the warehouse in an incremental and evolutionary manner, as shown in Figure 4.2. First, a high-level corporate data model is defined within a reasonably short period (such as one or two months) that provides a corporate-wide, consistent, integrated view of data among different subjects and potential usages. This high-level model, although it will need to be refined in the further development of enterprise data warehouses and departmental data marts, will greatly reduce future integration problems. Second, independent data marts can be implemented in parallel with the enterprise

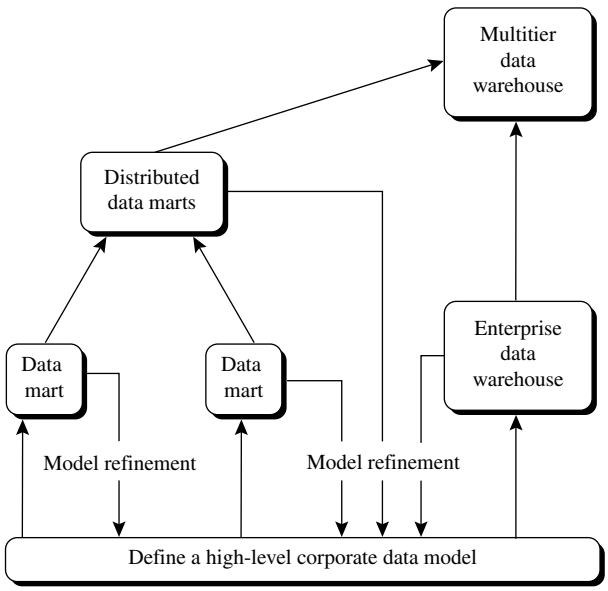


Figure 4.2 A recommended approach for data warehouse development.

warehouse based on the same corporate data model set noted before. Third, distributed data marts can be constructed to integrate different data marts via hub servers. Finally, a **multitier data warehouse** is constructed where the enterprise warehouse is the sole custodian of all warehouse data, which is then distributed to the various dependent data marts.

4.1.6 Extraction, Transformation, and Loading

Data warehouse systems use back-end tools and utilities to populate and refresh their data (Figure 4.1). These tools and utilities include the following functions:

- **Data extraction**, which typically gathers data from multiple, heterogeneous, and external sources.
- **Data cleaning**, which detects errors in the data and rectifies them when possible.
- **Data transformation**, which converts data from legacy or host format to warehouse format.
- **Load**, which sorts, summarizes, consolidates, computes views, checks integrity, and builds indices and partitions.
- **Refresh**, which propagates the updates from the data sources to the warehouse.

Besides cleaning, loading, refreshing, and metadata definition tools, data warehouse systems usually provide a good set of data warehouse management tools.

Data cleaning and data transformation are important steps in improving the data quality and, subsequently, the data mining results (see Chapter 3). Because we are mostly interested in the aspects of data warehousing technology related to data mining, we will not get into the details of the remaining tools, and recommend interested readers to consult books dedicated to data warehousing technology.

4.1.7 Metadata Repository

Metadata are data about data. When used in a data warehouse, metadata are the data that define warehouse objects. Figure 4.1 showed a metadata repository within the bottom tier of the data warehousing architecture. Metadata are created for the data names and definitions of the given warehouse. Additional metadata are created and captured for timestamping any extracted data, the source of the extracted data, and missing fields that have been added by data cleaning or integration processes.

A metadata repository should contain the following:

- A description of the *data warehouse structure*, which includes the warehouse schema, view, dimensions, hierarchies, and derived data definitions, as well as data mart locations and contents.

- *Operational metadata*, which include data lineage (history of migrated data and the sequence of transformations applied to it), currency of data (active, archived, or purged), and monitoring information (warehouse usage statistics, error reports, and audit trails).
- The *algorithms used for summarization*, which include measure and dimension definition algorithms, data on granularity, partitions, subject areas, aggregation, summarization, and predefined queries and reports.
- *Mapping from the operational environment to the data warehouse*, which includes source databases and their contents, gateway descriptions, data partitions, data extraction, cleaning, transformation rules and defaults, data refresh and purging rules, and security (user authorization and access control).
- *Data related to system performance*, which include indices and profiles that improve data access and retrieval performance, in addition to rules for the timing and scheduling of refresh, update, and replication cycles.
- *Business metadata*, which include business terms and definitions, data ownership information, and charging policies.

A data warehouse contains different levels of summarization, of which metadata is one. Other types include current detailed data (which are almost always on disk), older detailed data (which are usually on tertiary storage), lightly summarized data, and highly summarized data (which may or may not be physically housed).

Metadata play a very different role than other data warehouse data and are important for many reasons. For example, metadata are used as a directory to help the decision support system analyst locate the contents of the data warehouse, and as a guide to the data mapping when data are transformed from the operational environment to the data warehouse environment. Metadata also serve as a guide to the algorithms used for summarization between the current detailed data and the lightly summarized data, and between the lightly summarized data and the highly summarized data. Metadata should be stored and managed persistently (i.e., on disk).

4.2 Data Warehouse Modeling: Data Cube and OLAP

Data warehouses and OLAP tools are based on a **multidimensional data model**. This model views data in the form of a *data cube*. In this section, you will learn how data cubes model *n*-dimensional data (Section 4.2.1). In Section 4.2.2, various multidimensional models are shown: star schema, snowflake schema, and fact constellation. You will also learn about concept hierarchies (Section 4.2.3) and measures (Section 4.2.4) and how they can be used in basic OLAP operations to allow interactive mining at multiple levels of abstraction. Typical OLAP operations such as drill-down and roll-up are illustrated

(Section 4.2.5). Finally, the starnet model for querying multidimensional databases is presented (Section 4.2.6).

4.2.1 Data Cube: A Multidimensional Data Model

"*What is a data cube?*" A **data cube** allows data to be modeled and viewed in multiple dimensions. It is defined by dimensions and facts.

In general terms, **dimensions** are the perspectives or entities with respect to which an organization wants to keep records. For example, *AllElectronics* may create a *sales* data warehouse in order to keep records of the store's sales with respect to the dimensions *time*, *item*, *branch*, and *location*. These dimensions allow the store to keep track of things like monthly sales of items and the branches and locations at which the items were sold. Each dimension may have a table associated with it, called a **dimension table**, which further describes the dimension. For example, a dimension table for *item* may contain the attributes *item_name, brand*, and *type*. Dimension tables can be specified by users or experts, or automatically generated and adjusted based on data distributions.

A multidimensional data model is typically organized around a central theme, such as *sales*. This theme is represented by a fact table. **Facts** are numeric measures. Think of them as the quantities by which we want to analyze relationships between dimensions. Examples of facts for a sales data warehouse include *dollars_sold* (sales amount in dollars), *units_sold* (number of units sold), and *amount_budgeted*. The **fact table** contains the names of the *facts*, or measures, as well as keys to each of the related dimension tables. You will soon get a clearer picture of how this works when we look at multidimensional schemas.

Although we usually think of cubes as 3-D geometric structures, in data warehousing the data cube is *n*-dimensional. To gain a better understanding of data cubes and the multidimensional data model, let's start by looking at a simple 2-D data cube that is, in fact, a table or spreadsheet for sales data from *AllElectronics*. In particular, we will look at the *AllElectronics* sales data for items sold per quarter in the city of Vancouver. These data are shown in Table 4.2. In this 2-D representation, the sales for Vancouver are shown with respect to the *time* dimension (organized in quarters) and the *item* dimension (organized according to the types of items sold). The fact or measure displayed is *dollars_sold* (in thousands).

Now, suppose that we would like to view the sales data with a third dimension. For instance, suppose we would like to view the data according to *time* and *item*, as well as *location*, for the cities Chicago, New York, Toronto, and Vancouver. These 3-D data are shown in Table 4.3. The 3-D data in the table are represented as a series of 2-D tables. Conceptually, we may also represent the same data in the form of a 3-D data cube, as in Figure 4.3.

Suppose that we would now like to view our sales data with an additional fourth dimension such as *supplier*. Viewing things in 4-D becomes tricky. However, we can think of a 4-D cube as being a series of 3-D cubes, as shown in Figure 4.4. If we continue

Table 4.2 2-D View of Sales Data for *AllElectronics* According to *time* and *item*

	location = *"Vancouver"*			
	item *(type)*			
time *(quarter)*	*home entertainment*	*computer*	*phone*	*security*
Q1	605	825	14	400
Q2	680	952	31	512
Q3	812	1023	30	501
Q4	927	1038	38	580

Note: The sales are from branches located in the city of Vancouver. The measure displayed is *dollars_sold* (in thousands).

Table 4.3 3-D View of Sales Data for *AllElectronics* According to *time*, *item*, and *location*

	location = "Chicago"				***location*** = "New York"				***location*** = "Toronto"				***location*** = "Vancouver"			
	item				***item***				***item***				***item***			
time	*home ent.*	*comp.*	*phone*	*sec.*	*home ent.*	*comp.*	*phone*	*sec.*	*home ent.*	*comp.*	*phone*	*sec.*	*home ent.*	*comp.*	*phone*	*sec.*
Q1	854	882	89	623	1087	968	38	872	818	746	43	591	605	825	14	400
Q2	943	890	64	698	1130	1024	41	925	894	769	52	682	680	952	31	512
Q3	1032	924	59	789	1034	1048	45	1002	940	795	58	728	812	1023	30	501
Q4	1129	992	63	870	1142	1091	54	984	978	864	59	784	927	1038	38	580

Note: The measure displayed is *dollars_sold* (in thousands).

in this way, we may display any n-dimensional data as a series of $(n-1)$-dimensional "cubes." The data cube is a metaphor for multidimensional data storage. The actual physical storage of such data may differ from its logical representation. The important thing to remember is that data cubes are n-dimensional and do not confine data to 3-D.

Tables 4.2 and 4.3 show the data at different degrees of summarization. In the data warehousing research literature, a data cube like those shown in Figures 4.3 and 4.4 is often referred to as a **cuboid**. Given a set of dimensions, we can generate a cuboid for each of the possible subsets of the given dimensions. The result would form a *lattice* of cuboids, each showing the data at a different level of summarization, or group-by. The lattice of cuboids is then referred to as a data cube. Figure 4.5 shows a lattice of cuboids forming a data cube for the dimensions *time*, *item*, *location*, and *supplier*.

The cuboid that holds the lowest level of summarization is called the **base cuboid**. For example, the 4-D cuboid in Figure 4.4 is the base cuboid for the given *time*, *item*, *location*, and *supplier* dimensions. Figure 4.3 is a 3-D (nonbase) cuboid for *time*, *item*,

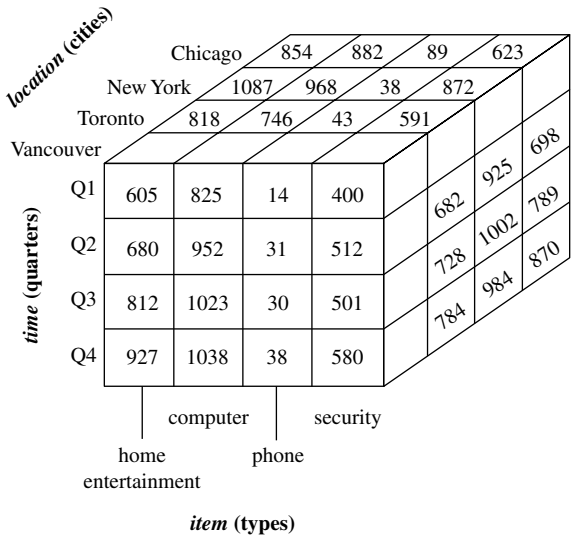


Figure 4.3 A 3-D data cube representation of the data in Table 4.3, according to *time*, *item*, and *location*. The measure displayed is *dollars_sold* (in thousands).

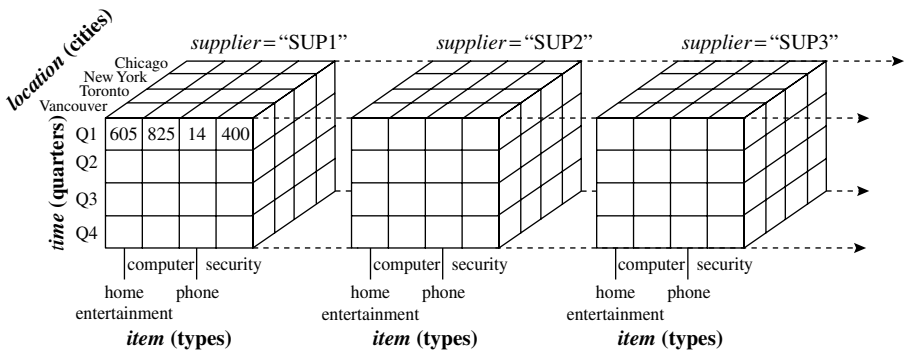


Figure 4.4 A 4-D data cube representation of sales data, according to *time*, *item*, *location*, and *supplier*. The measure displayed is *dollars_sold* (in thousands). For improved readability, only some of the cube values are shown.

and *location*, summarized for all suppliers. The 0-D cuboid, which holds the highest level of summarization, is called the **apex cuboid**. In our example, this is the total sales, or *dollars_sold*, summarized over all four dimensions. The apex cuboid is typically denoted by all.

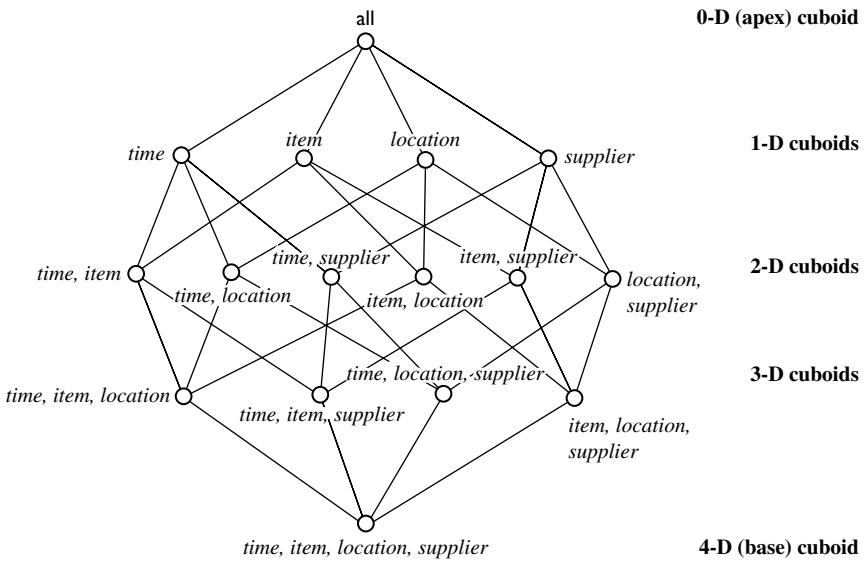


Figure 4.5 Lattice of cuboids, making up a 4-D data cube for *time*, *item*, *location*, and *supplier*. Each cuboid represents a different degree of summarization.

4.2.2 Stars, Snowflakes, and Fact Constellations: Schemas for Multidimensional Data Models

The entity-relationship data model is commonly used in the design of relational databases, where a database schema consists of a set of entities and the relationships between them. Such a data model is appropriate for online transaction processing. A data warehouse, however, requires a concise, subject-oriented schema that facilitates online data analysis.

The most popular data model for a data warehouse is a **multidimensional model**, which can exist in the form of a **star schema**, a **snowflake schema**, or a **fact constellation schema**. Let's look at each of these.

Star schema: The most common modeling paradigm is the star schema, in which the data warehouse contains (1) a large central table (**fact table**) containing the bulk of the data, with no redundancy, and (2) a set of smaller attendant tables (**dimension tables**), one for each dimension. The schema graph resembles a starburst, with the dimension tables displayed in a radial pattern around the central fact table.

Example 4.1 **Star schema.** A star schema for *AllElectronics* sales is shown in Figure 4.6. Sales are considered along four dimensions: *time, item, branch*, and *location*. The schema contains a central fact table for *sales* that contains keys to each of the four dimensions, along

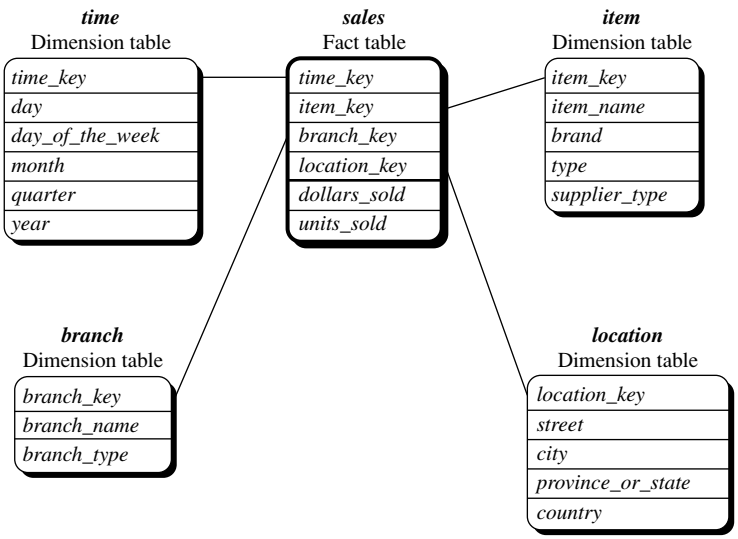


Figure 4.6 Star schema of *sales* data warehouse.

with two measures: *dollars_sold* and *units_sold*. To minimize the size of the fact table, dimension identifiers (e.g., *time_key* and *item_key*) are system-generated identifiers. ■

Notice that in the star schema, each dimension is represented by only one table, and each table contains a set of attributes. For example, the *location* dimension table contains the attribute set {*location_key, street, city, province_or_state, country*}. This constraint may introduce some redundancy. For example, "Urbana" and "Chicago" are both cities in the state of Illinois, USA. Entries for such cities in the *location* dimension table will create redundancy among the attributes *province_or_state* and *country*; that is, (..., Urbana, IL, USA) and (..., Chicago, IL, USA). Moreover, the attributes within a dimension table may form either a hierarchy (total order) or a lattice (partial order).

Snowflake schema: The snowflake schema is a variant of the star schema model, where some dimension tables are *normalized*, thereby further splitting the data into additional tables. The resulting schema graph forms a shape similar to a snowflake.

The major difference between the snowflake and star schema models is that the dimension tables of the snowflake model may be kept in normalized form to reduce redundancies. Such a table is easy to maintain and saves storage space. However, this space savings is negligible in comparison to the typical magnitude of the fact table. Furthermore, the snowflake structure can reduce the effectiveness of browsing, since more joins will be needed to execute a query. Consequently, the system performance may be adversely impacted. Hence, although the snowflake schema reduces redundancy, it is not as popular as the star schema in data warehouse design.

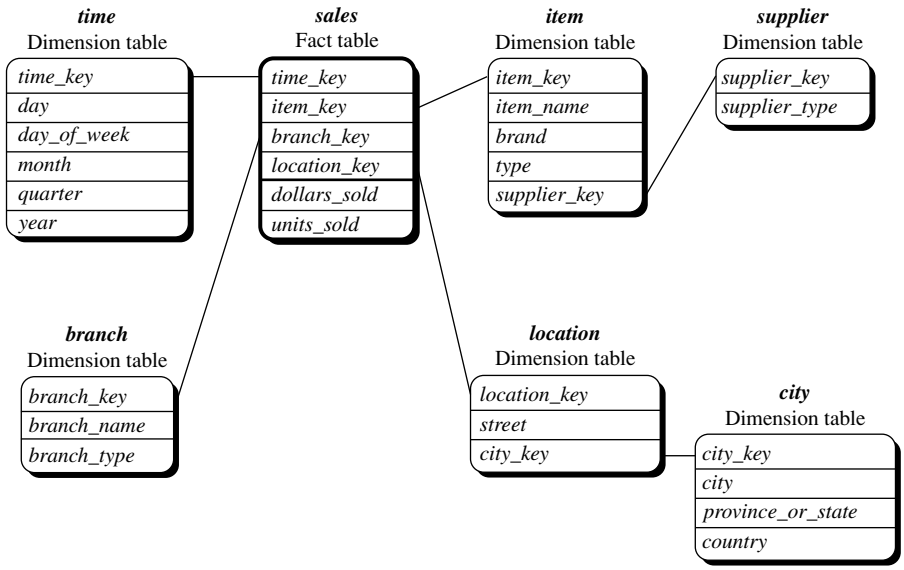


Figure 4.7 Snowflake schema of a *sales* data warehouse.

Example 4.2 **Snowflake schema.** A snowflake schema for *AllElectronics* sales is given in Figure 4.7. Here, the *sales* fact table is identical to that of the star schema in Figure 4.6. The main difference between the two schemas is in the definition of dimension tables. The single dimension table for *item* in the star schema is normalized in the snowflake schema, resulting in new *item* and *supplier* tables. For example, the *item* dimension table now contains the attributes *item_key, item_name, brand, type*, and *supplier_key*, where *supplier_key* is linked to the *supplier* dimension table, containing *supplier_key* and *supplier_type* information. Similarly, the single dimension table for *location* in the star schema can be normalized into two new tables: *location* and *city*. The *city_key* in the new *location* table links to the *city* dimension. Notice that, when desirable, further normalization can be performed on *province_or_state* and *country* in the snowflake schema shown in Figure 4.7. ■

Fact constellation: Sophisticated applications may require multiple fact tables to *share* dimension tables. This kind of schema can be viewed as a collection of stars, and hence is called a **galaxy schema** or a **fact constellation**.

Example 4.3 **Fact constellation.** A fact constellation schema is shown in Figure 4.8. This schema specifies two fact tables, *sales* and *shipping*. The *sales* table definition is identical to that of the star schema (Figure 4.6). The *shipping* table has five dimensions, or keys—*item_key, time_key, shipper_key, from_location*, and *to_location*—and two measures—*dollars_cost*

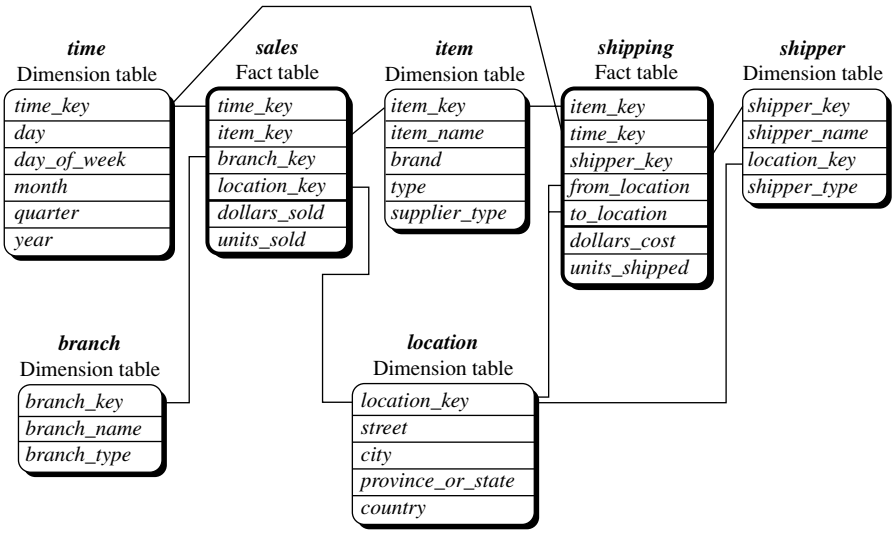


Figure 4.8 Fact constellation schema of a sales and shipping data warehouse.

and *units_shipped*. A fact constellation schema allows dimension tables to be shared between fact tables. For example, the dimensions tables for *time, item*, and *location* are shared between the *sales* and *shipping* fact tables. ■

In data warehousing, there is a distinction between a data warehouse and a data mart. A data warehouse collects information about subjects that span the *entire organization*, such as *customers, items, sales, assets*, and *personnel*, and thus its scope is *enterprise-wide*. For data warehouses, the fact constellation schema is commonly used, since it can model multiple, interrelated subjects. A **data mart**, on the other hand, is a department subset of the data warehouse that focuses on selected subjects, and thus its scope is *department-wide*. For data marts, the *star* or *snowflake* schema is commonly used, since both are geared toward modeling single subjects, although the star schema is more popular and efficient.

4.2.3 Dimensions: The Role of Concept Hierarchies

A **concept hierarchy** defines a sequence of mappings from a set of low-level concepts to higher-level, more general concepts. Consider a concept hierarchy for the dimension *location*. City values for *location* include Vancouver, Toronto, New York, and Chicago. Each city, however, can be mapped to the province or state to which it belongs. For example, Vancouver can be mapped to British Columbia, and Chicago to Illinois. The provinces and states can in turn be mapped to the country (e.g., Canada or the United States) to which they belong. These mappings form a concept hierarchy for the

dimension *location*, mapping a set of low-level concepts (i.e., cities) to higher-level, more general concepts (i.e., countries). This concept hierarchy is illustrated in Figure 4.9.

Many concept hierarchies are implicit within the database schema. For example, suppose that the dimension *location* is described by the attributes *number, street, city, province_or_state, zip_code*, and *country*. These attributes are related by a total order, forming a concept hierarchy such as "*street* < *city* < *province_or_state* < *country*." This hierarchy is shown in Figure 4.10(a). Alternatively, the attributes of a dimension may

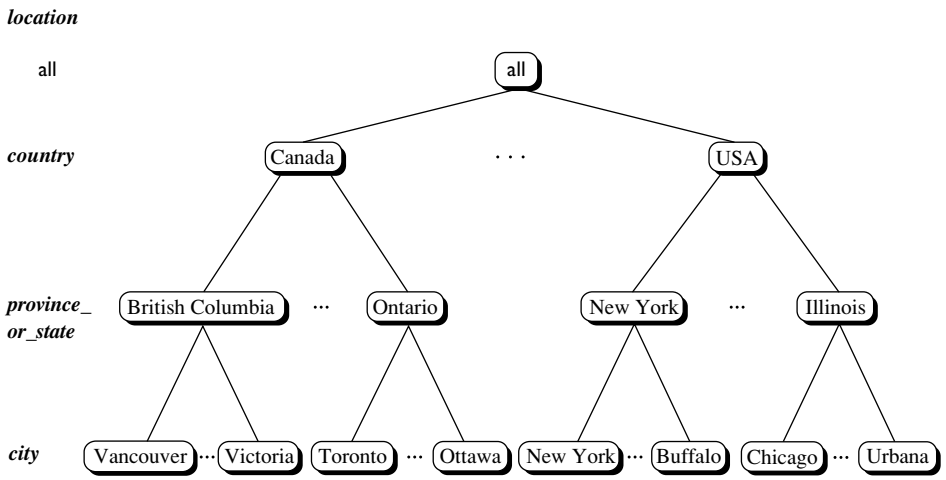


Figure 4.9 A concept hierarchy for *location*. Due to space limitations, not all of the hierarchy nodes are shown, indicated by ellipses between nodes.

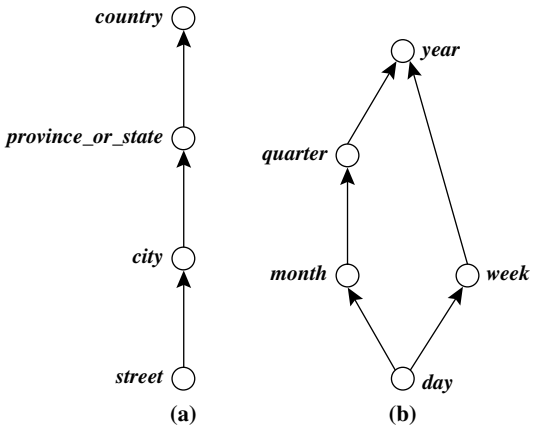


Figure 4.10 Hierarchical and lattice structures of attributes in warehouse dimensions: (a) a hierarchy for *location* and (b) a lattice for *time*.

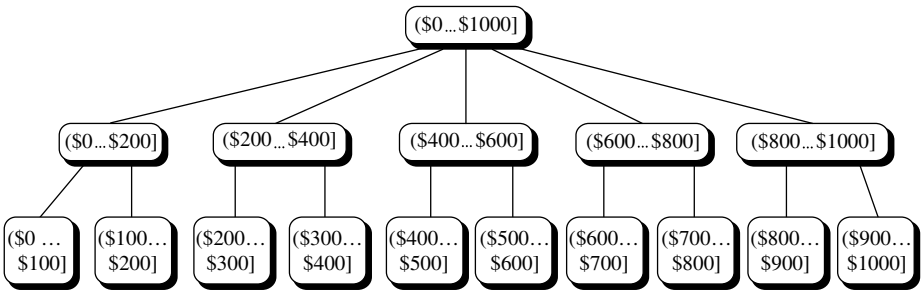


Figure 4.11 A concept hierarchy for *price*.

be organized in a partial order, forming a lattice. An example of a partial order for the *time* dimension based on the attributes *day, week, month, quarter*, and *year* is "*day* < {*month* < *quarter; week*} < *year*."[1] This lattice structure is shown in Figure 4.10(b). A concept hierarchy that is a total or partial order among attributes in a database schema is called a **schema hierarchy**. Concept hierarchies that are common to many applications (e.g., *for time*) may be predefined in the data mining system. Data mining systems should provide users with the flexibility to tailor predefined hierarchies according to their particular needs. For example, users may want to define a fiscal year starting on April 1 or an academic year starting on September 1.

Concept hierarchies may also be defined by discretizing or grouping values for a given dimension or attribute, resulting in a **set-grouping hierarchy**. A total or partial order can be defined among groups of values. An example of a set-grouping hierarchy is shown in Figure 4.11 for the dimension *price*, where an interval ($\$X \ldots \$Y$] denotes the range from $\$X$ (exclusive) to $\$Y$ (inclusive).

There may be more than one concept hierarchy for a given attribute or dimension, based on different user viewpoints. For instance, a user may prefer to organize *price* by defining ranges for *inexpensive, moderately_priced*, and *expensive*.

Concept hierarchies may be provided manually by system users, domain experts, or knowledge engineers, or may be automatically generated based on statistical analysis of the data distribution. The automatic generation of concept hierarchies is discussed in Chapter 3 as a preprocessing step in preparation for data mining.

Concept hierarchies allow data to be handled at varying levels of abstraction, as we will see in Section 4.2.4.

4.2.4 Measures: Their Categorization and Computation

"*How are measures computed?*" To answer this question, we first study how measures can be categorized. Note that a *multidimensional point* in the data cube space can be defined

[1]Since a *week* often crosses the boundary of two consecutive months, it is usually not treated as a lower abstraction of *month*. Instead, it is often treated as a lower abstraction of *year*, since a year contains approximately 52 weeks.

by a set of dimension–value pairs; for example, ⟨*time* = "Q1", *location* = "Vancouver", *item* = "computer"⟩. A data cube **measure** is a numeric function that can be evaluated at each point in the data cube space. A measure value is computed for a given point by aggregating the data corresponding to the respective dimension–value pairs defining the given point. We will look at concrete examples of this shortly.

Measures can be organized into three categories—distributive, algebraic, and holistic—based on the kind of aggregate functions used.

Distributive: An aggregate function is *distributive* if it can be computed in a distributed manner as follows. Suppose the data are partitioned into n sets. We apply the function to each partition, resulting in n aggregate values. If the result derived by applying the function to the n aggregate values is the same as that derived by applying the function to the entire data set (without partitioning), the function can be computed in a distributed manner. For example, sum() can be computed for a data cube by first partitioning the cube into a set of subcubes, computing sum() for each subcube, and then summing up the counts obtained for each subcube. Hence, sum() is a distributive aggregate function.

For the same reason, count(), min(), and max() are distributive aggregate functions. By treating the count value of each nonempty base cell as 1 by default, count() of any cell in a cube can be viewed as the sum of the count values of all of its corresponding child cells in its subcube. Thus, count() is distributive. A measure is *distributive* if it is obtained by applying a distributive aggregate function. Distributive measures can be computed efficiently because of the way the computation can be partitioned.

Algebraic: An aggregate function is *algebraic* if it can be computed by an algebraic function with M arguments (where M is a bounded positive integer), each of which is obtained by applying a distributive aggregate function. For example, avg() (average) can be computed by sum()/count(), where both sum() and count() are distributive aggregate functions. Similarly, it can be shown that min_N() and max_N() (which find the N minimum and N maximum values, respectively, in a given set) and standard_deviation() are algebraic aggregate functions. A measure is *algebraic* if it is obtained by applying an algebraic aggregate function.

Holistic: An aggregate function is *holistic* if there is no constant bound on the storage size needed to describe a subaggregate. That is, there does not exist an algebraic function with M arguments (where M is a constant) that characterizes the computation. Common examples of holistic functions include median(), mode(), and rank(). A measure is *holistic* if it is obtained by applying a holistic aggregate function.

Most large data cube applications require efficient computation of distributive and algebraic measures. Many efficient techniques for this exist. In contrast, it is difficult to compute holistic measures efficiently. Efficient techniques to *approximate* the computation of some holistic measures, however, do exist. For example, rather than computing the exact median(), Equation (2.3) of Chapter 2 can be used to estimate the approximate median value for a large data set. In many cases, such techniques are sufficient to overcome the difficulties of efficient computation of holistic measures.

Various methods for computing different measures in data cube construction are discussed in depth in Chapter 5. Notice that most of the current data cube technology confines the measures of multidimensional databases to *numeric data*. However, measures can also be applied to other kinds of data, such as spatial, multimedia, or text data.

4.2.5 Typical OLAP Operations

"How are concept hierarchies useful in OLAP?" In the multidimensional model, data are organized into multiple dimensions, and each dimension contains multiple levels of abstraction defined by concept hierarchies. This organization provides users with the flexibility to view data from different perspectives. A number of OLAP data cube operations exist to materialize these different views, allowing interactive querying and analysis of the data at hand. Hence, OLAP provides a user-friendly environment for interactive data analysis.

Example 4.4 **OLAP operations.** Let's look at some typical OLAP operations for multidimensional data. Each of the following operations described is illustrated in Figure 4.12. At the center of the figure is a data cube for *AllElectronics* sales. The cube contains the dimensions *location, time*, and *item*, where *location* is aggregated with respect to city values, *time* is aggregated with respect to quarters, and *item* is aggregated with respect to item types. To aid in our explanation, we refer to this cube as the central cube. The measure displayed is *dollars_sold* (in thousands). (For improved readability, only some of the cubes' cell values are shown.) The data examined are for the cities Chicago, New York, Toronto, and Vancouver.

Roll-up: The roll-up operation (also called the *drill-up* operation by some vendors) performs aggregation on a data cube, either by *climbing up a concept hierarchy* for a dimension or by *dimension reduction*. Figure 4.12 shows the result of a roll-up operation performed on the central cube by climbing up the concept hierarchy for *location* given in Figure 4.9. This hierarchy was defined as the total order "*street* < *city* < *province_or_state* < *country*." The roll-up operation shown aggregates the data by ascending the *location* hierarchy from the level of *city* to the level of *country*. In other words, rather than grouping the data by city, the resulting cube groups the data by country.

When roll-up is performed by dimension reduction, one or more dimensions are removed from the given cube. For example, consider a sales data cube containing only the *location* and *time* dimensions. Roll-up may be performed by removing, say, the *time* dimension, resulting in an aggregation of the total sales by location, rather than by location and by time.

Drill-down: Drill-down is the reverse of roll-up. It navigates from less detailed data to more detailed data. Drill-down can be realized by either *stepping down a concept hierarchy* for a dimension or *introducing additional dimensions*. Figure 4.12 shows the result of a drill-down operation performed on the central cube by stepping down a

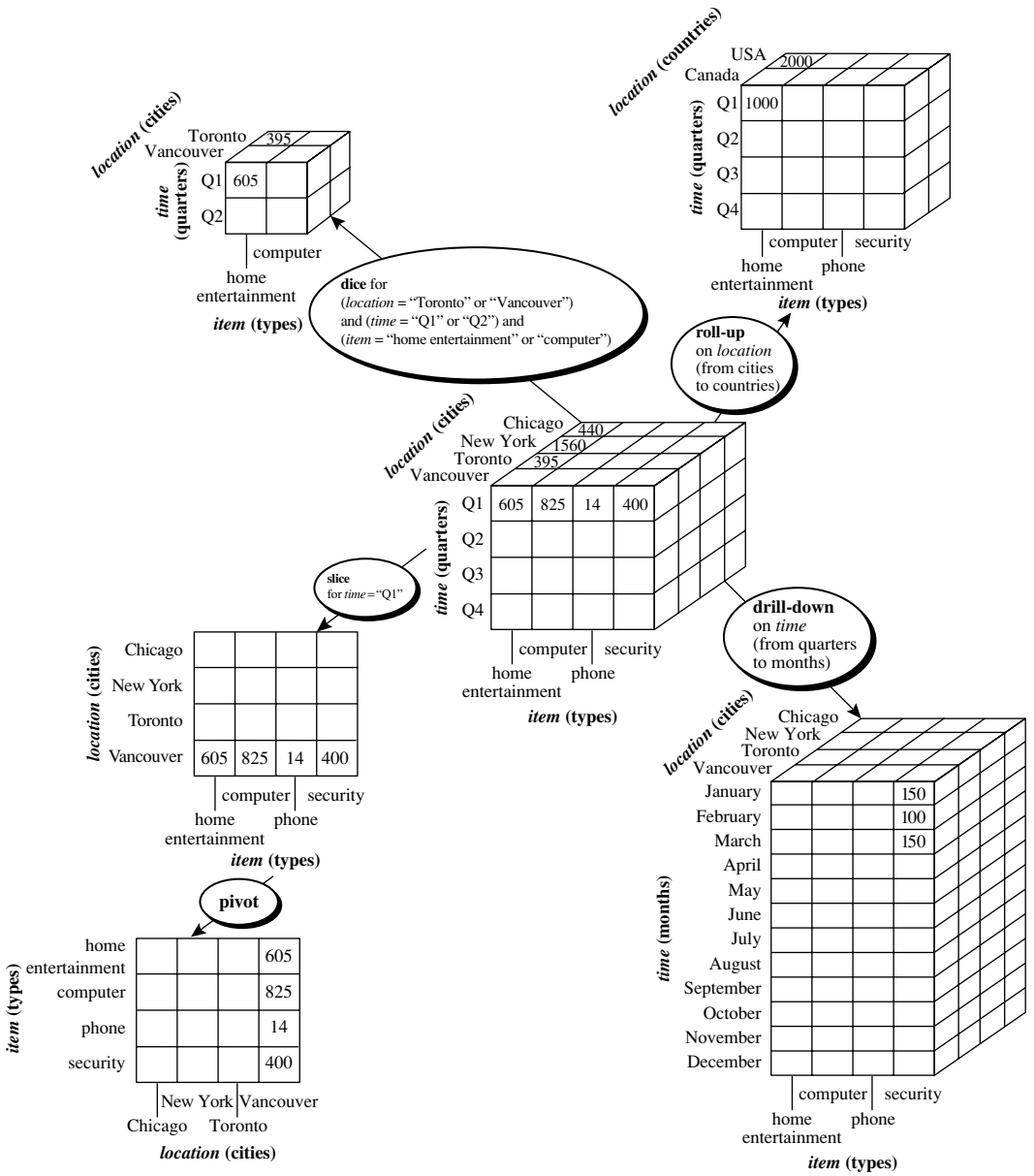


Figure 4.12 Examples of typical OLAP operations on multidimensional data.

concept hierarchy for *time* defined as "*day* < *month* < *quarter* < *year*." Drill-down occurs by descending the *time* hierarchy from the level of *quarter* to the more detailed level of *month*. The resulting data cube details the total sales per month rather than summarizing them by quarter.

Because a drill-down adds more detail to the given data, it can also be performed by adding new dimensions to a cube. For example, a drill-down on the central cube of Figure 4.12 can occur by introducing an additional dimension, such as *customer_group*.

Slice and dice: The *slice* operation performs a selection on one dimension of the given cube, resulting in a subcube. Figure 4.12 shows a slice operation where the sales data are selected from the central cube for the dimension *time* using the criterion *time* = "Q1." The *dice* operation defines a subcube by performing a selection on two or more dimensions. Figure 4.12 shows a dice operation on the central cube based on the following selection criteria that involve three dimensions: (*location* = "Toronto" or "Vancouver") and (*time* = "Q1" or "Q2") and (item = "home entertainment" or "computer").

Pivot (rotate): *Pivot* (also called *rotate*) is a visualization operation that rotates the data axes in view to provide an alternative data presentation. Figure 4.12 shows a pivot operation where the *item* and *location* axes in a 2-D slice are rotated. Other examples include rotating the axes in a 3-D cube, or transforming a 3-D cube into a series of 2-D planes.

Other OLAP operations: Some OLAP systems offer additional drilling operations. For example, **drill-across** executes queries involving (i.e., across) more than one fact table. The **drill-through** operation uses relational SQL facilities to drill through the bottom level of a data cube down to its back-end relational tables.

Other OLAP operations may include ranking the top N or bottom N items in lists, as well as computing moving averages, growth rates, interests, internal return rates, depreciation, currency conversions, and statistical functions. ■

OLAP offers analytical modeling capabilities, including a calculation engine for deriving ratios, variance, and so on, and for computing measures across multiple dimensions. It can generate summarizations, aggregations, and hierarchies at each granularity level and at every dimension intersection. OLAP also supports functional models for forecasting, trend analysis, and statistical analysis. In this context, an OLAP engine is a powerful data analysis tool.

OLAP Systems versus Statistical Databases

Many OLAP systems' characteristics (e.g., the use of a multidimensional data model and concept hierarchies, the association of measures with dimensions, and the notions of roll-up and drill-down) also exist in earlier work on statistical databases (SDBs). A **statistical database** is a database system that is designed to support statistical applications. Similarities between the two types of systems are rarely discussed, mainly due to differences in terminology and application domains.