Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Data Mining: Concepts and Techniques

Jiawei Han and **Micheline Kamber**

Simon Fraser University

Preface

Our capabilities of both generating and collecting data have been increasing rapidly in the last several decades. Contributing factors include the widespread use of bar codes for most commercial products, the computerization of many business, scientific and government transactions and managements, and advances in data collection tools ranging from scanned texture and image platforms, to on-line instrumentation in manufacturing and shopping, and to satellite remote sensing systems. In addition, popular use of the World Wide Web as a global information system has flooded us with a tremendous amount of data and information. This explosive growth in stored data has generated an urgent need for new techniques and automated tools that can intelligently assist us in transforming the vast amounts of data into useful information and knowledge.

This book explores the concepts and techniques of *data mining*, a promising and flourishing frontier in database systems and new database applications. Data mining, also popularly referred to as *knowledge discovery in databases (KDD)*, is the automated or convenient extraction of patterns representing knowledge implicitly stored in large databases, data warehouses, and other massive information repositories.

Data mining is a multidisciplinary field, drawing work from areas including database technology, artificial intelligence, machine learning, neural networks, statistics, pattern recognition, knowledge based systems, knowledge acquisition, information retrieval, high performance computing, and data visualization. We present the material in this book from a *database perspective*. That is, we focus on issues relating to the feasibility, usefulness, efficiency, and scalability of techniques for the discovery of patterns hidden *in large databases*. As a result, this book is not intended as an introduction to database systems, machine learning, or statistics, etc., although we do provide the background necessary in these areas in order to facilitate the reader's comprehension of their respective roles in data mining. Rather, the book is a comprehensive introduction to data mining, presented with database issues in focus. It should be useful for computing science students, application developers, and business professionals, as well as researchers involved in any of the disciplines listed above.

Data mining emerged during the late 1980's, has made great strides during the 1990's, and is expected to continue to flourish into the new millennium. This book presents an overall picture of the field from a database researcher's point of view, introducing interesting data mining techniques and systems, and discussing applications and research directions. An important motivation for writing this book was the need to build an organized framework for the study of data mining — a challenging task owing to the extensive multidisciplinary nature of this fast developing field. We hope that this book will encourage people with different backgrounds and experiences to exchange their views regarding data mining so as to contribute towards the further promotion and shaping of this exciting and dynamic field.

To the teacher

This book is designed to give a broad, yet in depth overview of the field of data mining. You will find it useful for teaching a course on data mining at an advanced undergraduate level, or the first-year graduate level. In addition, individual chapters may be included as material for courses on selected topics in database systems or in artificial intelligence. We have tried to make the chapters as self-contained as possible. For a course taught at the undergraduate level, you might use chapters 1 to 8 as the core course material. Remaining class material may be selected from among the more advanced topics described in chapters 9 and 10. For a graduate level course, you may choose to cover the entire book in one semester.

Each chapter ends with a set of exercises, suitable as assigned homework. The exercises are either short questions

that test basic mastery of the material covered, or longer questions which require analytical thinking.

To the student

We hope that this textbook will spark your interest in the fresh, yet evolving field of data mining. We have attempted to present the material in a clear manner, with careful explanation of the topics covered. Each chapter ends with a summary describing the main points. We have included many figures and illustrations throughout the text in order to make the book more enjoyable and "reader-friendly". Although this book was designed as a textbook, we have tried to organize it so that it will also be useful to you as a reference book or handbook, should you later decide to pursue a career in data mining.

What do you need to know in order to read this book?

- You should have some knowledge of the concepts and terminology associated with database systems. However, we do try to provide enough background of the basics in database technology, so that if your memory is a bit rusty, you will not have trouble following the discussions in the book. You should have some knowledge of database querying, although knowledge of any specific query language is not required.
- You should have some programming experience. In particular, you should be able to read pseudo-code, and understand simple data structures such as multidimensional arrays.
- It will be helpful to have some preliminary background in statistics, machine learning, or pattern recognition. However, we will familiarize you with the basic concepts of these areas that are relevant to data mining from a database perspective.

To the professional

This book was designed to cover a broad range of topics in the field of data mining. As a result, it is a good handbook on the subject. Because each chapter is designed to be as stand-alone as possible, you can focus on the topics that most interest you. Much of the book is suited to applications programmers or information service managers like yourself who wish to learn about the key ideas of data mining on their own.

The techniques and algorithms presented are of practical utility. Rather than selecting algorithms that perform well on small "toy" databases, the algorithms described in the book are geared for the discovery of data patterns hidden in large, real databases. In Chapter 10, we briefly discuss data mining systems in commercial use, as well as promising research prototypes. Each algorithm presented in the book is illustrated in pseudo-code. The pseudo-code is similar to the C programming language, yet is designed so that it should be easy to follow by programmers unfamiliar with C or C++. If you wish to implement any of the algorithms, you should find the translation of our pseudo-code into the programming language of your choice to be a fairly straightforward task.

Organization of the book

The book is organized as follows.

Chapter 1 provides an introduction to the multidisciplinary field of data mining. It discusses the evolutionary path of database technology which led up to the need for data mining, and the importance of its application potential. The basic architecture of data mining systems is described, and a brief introduction to the concepts of database systems and data warehouses is given. A detailed classification of data mining tasks is presented, based on the different kinds of knowledge to be mined. A classification of data mining systems is presented, and major challenges in the field are discussed.

Chapter 2 is an introduction to data warehouses and OLAP (On-Line Analytical Processing). Topics include the concept of data warehouses and multidimensional databases, the construction of data cubes, the implementation of on-line analytical processing, and the relationship between data warehousing and data mining.

Chapter 3 describes techniques for preprocessing the data prior to mining. Methods of data cleaning, data integration and transformation, and data reduction are discussed, including the use of concept hierarchies for dynamic and static discretization. The automatic generation of concept hierarchies is also described.

Chapter 4 introduces the primitives of data mining which define the specification of a data mining task. It describes a data mining query language (DMQL), and provides examples of data mining queries. Other topics include the construction of graphical user interfaces, and the specification and manipulation of concept hierarchies.

Chapter 5 describes techniques for concept description, including characterization and discrimination. An attribute-oriented generalization technique is introduced, as well as its different implementations including a generalized relation technique and a multidimensional data cube technique. Several forms of knowledge presentation and visualization are illustrated. Relevance analysis is discussed. Methods for class comparison at multiple abstraction levels, and methods for the extraction of characteristic rules and discriminant rules with interestingness measurements are presented. In addition, statistical measures for descriptive mining are discussed.

Chapter 6 presents methods for mining association rules in transaction databases as well as relational databases and data warehouses. It includes a classification of association rules, a presentation of the basic Apriori algorithm and its variations, and techniques for mining multiple-level association rules, multidimensional association rules, quantitative association rules, and correlation rules. Strategies for finding interesting rules by constraint-based mining and the use of interestingness measures to focus the rule search are also described.

Chapter 7 describes methods for data classification and predictive modeling. Major methods of classification and prediction are explained, including decision tree induction, Bayesian classification, the neural network technique of backpropagation, k-nearest neighbor classifiers, case-based reasoning, genetic algorithms, rough set theory, and fuzzy set approaches. Association-based classification, which applies association rule mining to the problem of classification, is presented. Methods of regression are introduced, and issues regarding classifier accuracy are discussed.

Chapter 8 describes methods of clustering analysis. It first introduces the concept of data clustering and then presents several major data clustering approaches, including partition-based clustering, hierarchical clustering, and model-based clustering. Methods for clustering continuous data, discrete data, and data in multidimensional data cubes are presented. The scalability of clustering algorithms is discussed in detail.

Chapter 9 discusses methods for data mining in advanced database systems. It includes data mining in object-oriented databases, spatial databases, text databases, multimedia databases, active databases, temporal databases, heterogeneous and legacy databases, and resource and knowledge discovery in the Internet information base.

Finally, in Chapter 10, we summarize the concepts presented in this book and discuss applications of data mining and some challenging research issues.

Errors

It is likely that this book may contain typos, errors, or omissions. If you notice any errors, have suggestions regarding additional exercises or have other constructive criticism, we would be very happy to hear from you. We welcome and appreciate your suggestions. You can send your comments to:

```
Data Mining: Concept and Techniques
Intelligent Database Systems Research Laboratory
Simon Fraser University,
Burnaby, British Columbia
Canada V5A 1S6
Fax: (604) 291-3045
```

Alternatively, you can use electronic mails to submit bug reports, request a list of known errors, or make constructive suggestions. To receive instructions, send email to dk@cs.sfu.ca with "Subject: help" in the message header. We regret that we cannot personally respond to all e-mails. The errata of the book and other updated information related to the book can be found by referencing the Web address: http://db.cs.sfu.ca/Book.

Acknowledgements

We would like to express our sincere thanks to all the members of the data mining research group who have been working with us at Simon Fraser University on data mining related research, and to all the members of the DBMiner system development team, who have been working on an exciting data mining project, DBMiner, and have made it a real success. The data mining research team currently consists of the following active members: Julia Gitline,

Kan Hu, Jean Hou, Pei Jian, Micheline Kamber, Eddie Kim, Jin Li, Xuebin Lu, Behzad Mortazav-Asl, Helen Pinto, Yiwen Yin, Zhaoxia Wang, and Hua Zhu. The DBMiner development team currently consists of the following active members: Kan Hu, Behzad Mortazav-Asl, and Hua Zhu, and some partime workers from the data mining research team. We are also grateful to Helen Pinto, Hua Zhu, and Lara Winstone for their help with some of the figures in this book.

More acknowledgements will be given at the final stage of the writing.

Contents

Chapter 1

Introduction

This book is an introduction to what has come to be known as *data mining* and *knowledge discovery in databases*. The material in this book is presented from a database perspective, where emphasis is placed on basic data mining concepts and techniques for uncovering interesting data patterns hidden in *large data sets*. The implementation methods discussed are particularly oriented towards the development of *scalable* and *efficient* data mining tools.

In this chapter, you will learn how data mining is part of the natural evolution of database technology, why data mining is important, and how it is defined. You will learn about the general architecture of data mining systems, as well as gain insight into the kinds of data on which mining can be performed, the types of patterns that can be found, and how to tell which patterns represent useful knowledge. In addition to studying a classification of data mining systems, you will read about challenging research issues for building data mining tools of the future.

1.1 What motivated data mining? Why is it important?

Necessity is the mother of invention.

— English proverb.

The major reason that data mining has attracted a great deal of attention in information industry in recent years is due to the wide availability of huge amounts of data and the imminent need for turning such data into useful information and knowledge. The information and knowledge gained can be used for applications ranging from business management, production control, and market analysis, to engineering design and science exploration.

Data mining can be viewed as a result of the natural evolution of information technology. An evolutionary path has been witnessed in the database industry in the development of the following functionalities (Figure 1.1): *data collection and database creation, data management* (including data storage and retrieval, and database transaction processing), and *data analysis and understanding* (involving data warehousing and data mining). For instance, the early development of data collection and database creation mechanisms served as a prerequisite for later development of effective mechanisms for data storage and retrieval, and query and transaction processing. With numerous database systems offering query and transaction processing as common practice, data analysis and understanding has naturally become the next target.

Since the 1960's, database and information technology has been evolving systematically from primitive file processing systems to sophisticated and powerful databases systems. The research and development in database systems since the 1970's has led to the development of relational database systems (where data are stored in relational table structures; see Section 1.3.1), data modeling tools, and indexing and data organization techniques. In addition, users gained convenient and flexible data access through query languages, query processing, and user interfaces. Efficient methods for **on-line transaction processing** (OLTP), where a query is viewed as a read-only transaction, have 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.

Database technology since the mid-1980s has been characterized by the popular adoption of relational technology and an upsurge of research and development activities on new and powerful database systems. These employ ad-

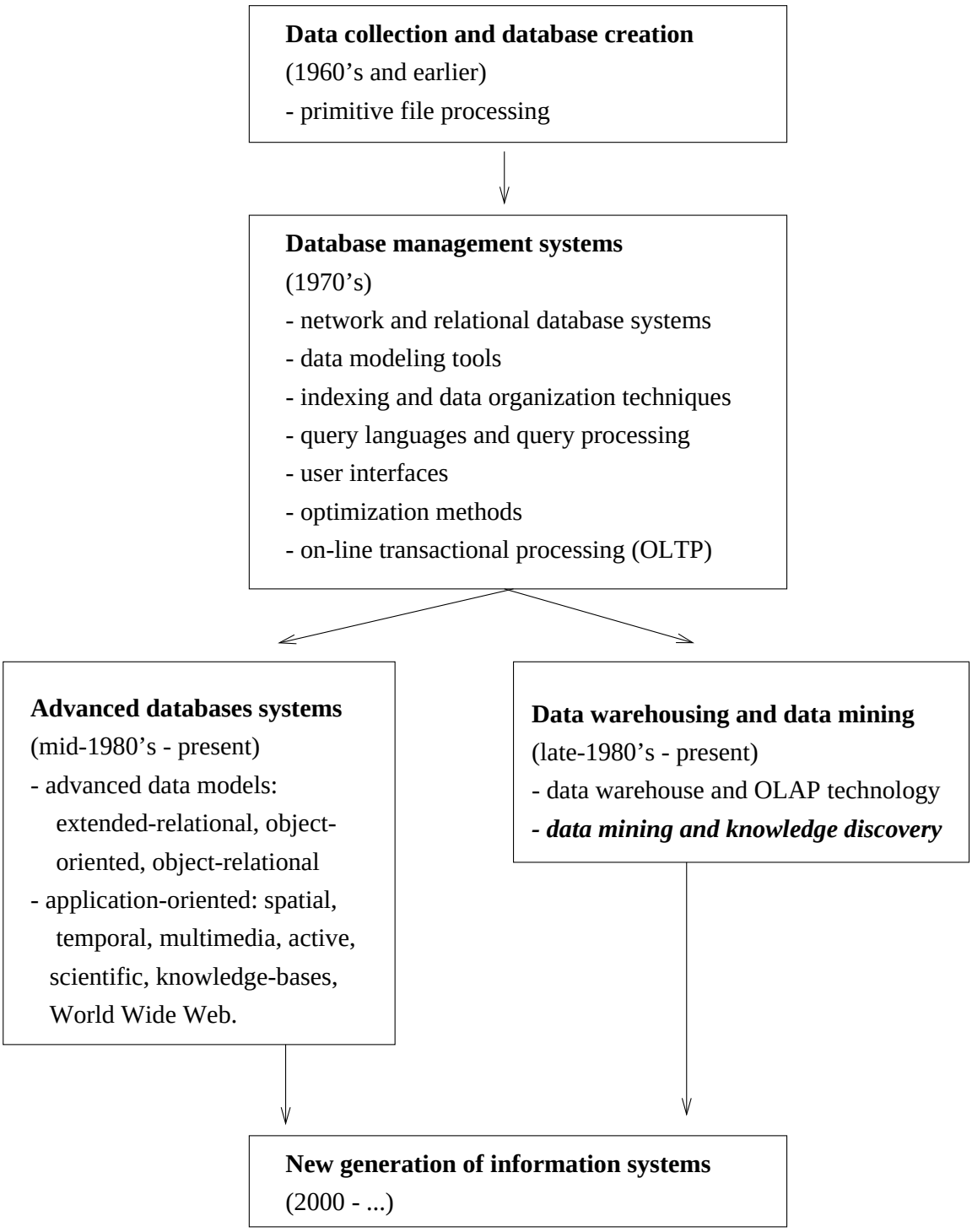


Figure 1.1: The evolution of database technology.

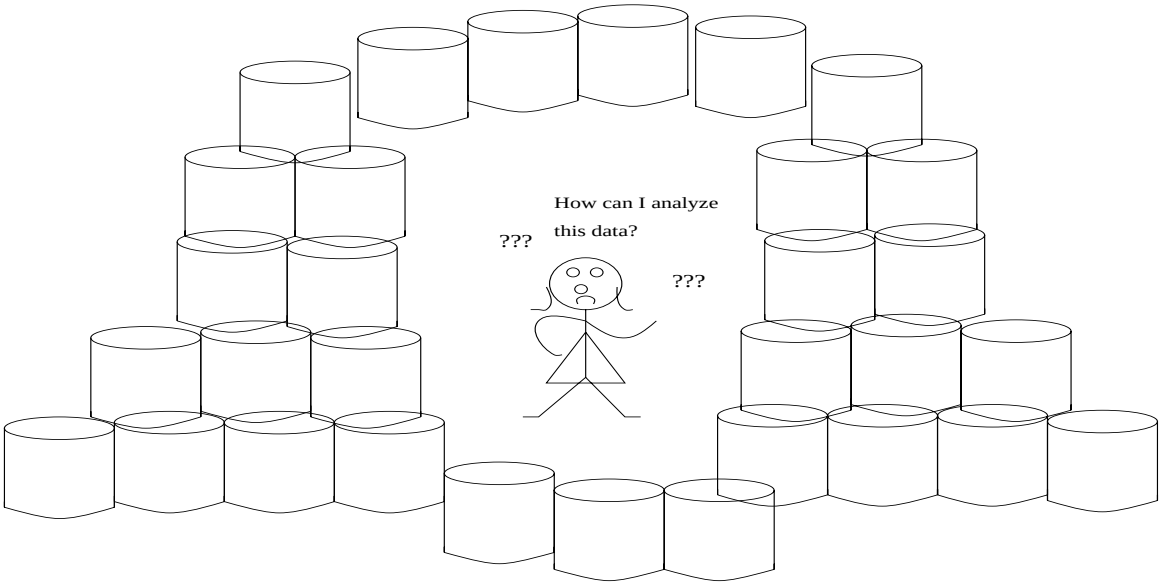


Figure 1.2: We are data rich, but information poor.

vanced data models such as extended-relational, object-oriented, object-relational, and deductive models Application-oriented database systems, including spatial, temporal, multimedia, active, and scientific databases, knowledge bases, and office information bases, have flourished. Issues related to the distribution, diversification, and sharing of data have been studied extensively. Heterogeneous database systems and Internet-based global information systems such as the World-Wide Web (WWW) also emerged and play a vital role in the information industry.

The steady and amazing progress of computer hardware technology in the past three decades has led to powerful, affordable, and large supplies of computers, data collection equipment, and storage media. This technology provides a great boost to the database and information industry, and makes a huge number of databases and information repositories available for transaction management, information retrieval, *and data analysis*.

Data can now be stored in many different types of databases. One database architecture that has recently emerged is the **data warehouse** (Section 1.3.2), a repository of multiple heterogeneous data sources, organized under a unified schema at a single site in order to facilitate management decision making. Data warehouse technology includes data cleansing, data integration, and **On-Line Analytical Processing (OLAP)**, that is, analysis techniques with functionalities such as summarization, consolidation and aggregation, as well as the ability to view information at different angles. Although OLAP tools support multidimensional analysis and decision making, additional data analysis tools are required for in-depth analysis, such as data classification, clustering, and the characterization of data changes over time.

The abundance of data, coupled with the need for powerful data analysis tools, has been described as a "*data rich but information poor*" situation. The fast-growing, tremendous amount of data, collected and stored in large and numerous databases, has far exceeded our human ability for comprehension *without powerful tools* (Figure 1.2). As a result, data collected in large databases become "data tombs" — data archives that are seldom revisited. Consequently, important decisions are often made based not on the information-rich data stored in databases 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. In addition, consider current expert system technologies, which typically rely on users or domain experts to *manually* input knowledge into knowledge bases. Unfortunately, this procedure is prone to biases and errors, and is extremely time-consuming and costly. Data mining tools which perform data analysis may uncover important data patterns, contributing greatly to business strategies, knowledge bases, and scientific and medical research. The widening gap between data and information calls for a systematic development of *data mining tools* which will turn data tombs into "golden nuggets" of knowledge.

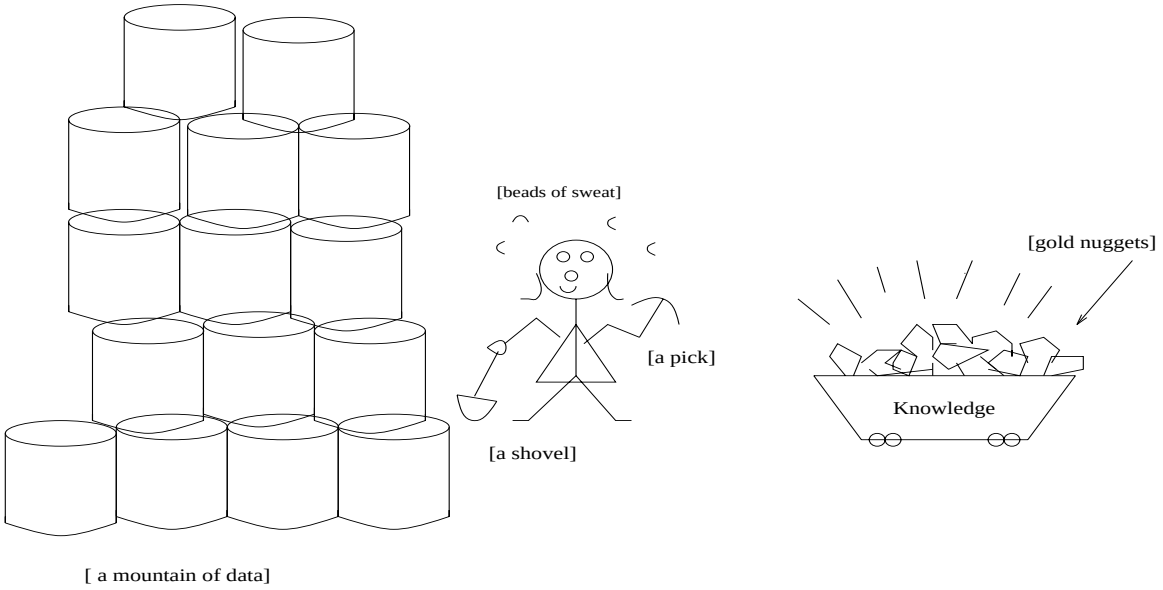


Figure 1.3: Data mining - searching for knowledge (interesting patterns) in your data.

1.2 So, what is data mining?

Simply stated, **data mining** refers to *extracting or "mining" knowledge from large amounts of data*. The term is actually a misnomer. Remember that the mining of gold from rocks or sand is referred to as *gold* mining rather than rock or sand mining. Thus, "data mining" should have been more appropriately named "knowledge mining from data", which is unfortunately somewhat long. "Knowledge mining", a shorter term, 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 which carries both "data" and "mining" became a popular choice. There are many other terms carrying a similar or slightly different meaning to data mining, such as **knowledge mining from databases, 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 in Databases**", or **KDD**. Alternatively, others view data mining as simply an essential step in the process of knowledge discovery in databases. Knowledge discovery as a process is depicted in Figure 1.4, and consists of an iterative sequence of the following steps:

- **data cleaning** (to remove noise or irrelevant data),
- **data integration** (where multiple data sources may be combined)[1],
- **data selection** (where data relevant to the analysis task are retrieved from the database),
- **data transformation** (where data are transformed or consolidated into forms appropriate for mining by performing summary or aggregation operations, for instance)[2],
- **data mining** (an essential process where intelligent methods are applied in order to extract data patterns),
- **pattern evaluation** (to identify the truly interesting patterns representing knowledge based on some **interestingness measures**; Section 1.5), and
- **knowledge presentation** (where visualization and knowledge representation techniques are used to present the mined knowledge to the user).

[1] 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.

[2] Sometimes data transformation and consolidation are performed before the data selection process, particularly in the case of data warehousing.

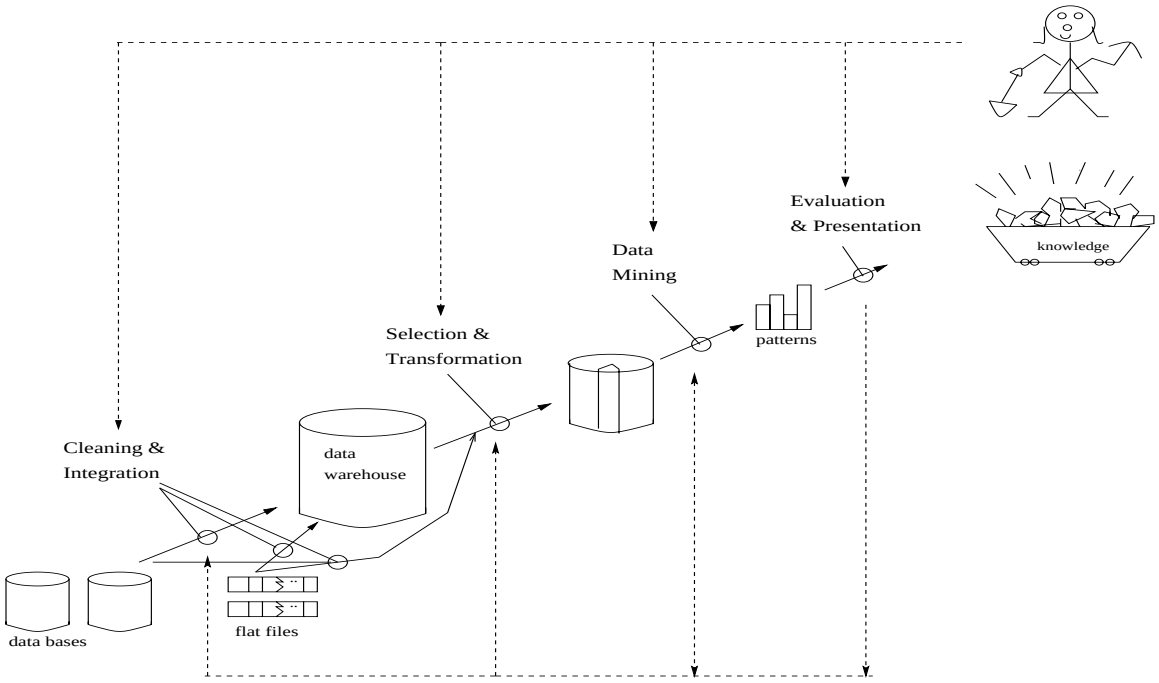


Figure 1.4: Data mining as a process of knowledge discovery.

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. Note that according to this view, data mining is only one step in the entire process, albeit an essential one since it uncovers hidden patterns for evaluation.

We agree that data mining is a knowledge discovery process. However, in industry, in media, and in the database research milieu, the term "data mining" is becoming more popular than the longer term of "knowledge discovery in databases". Therefore, in this book, we choose to use the term "data mining". We adopt a broad view of data mining functionality: **data mining** is the process of discovering interesting knowledge from *large* amounts of data stored either in databases, data warehouses, or other information repositories.

Based on this view, the architecture of a typical data mining system may have the following major components (Figure 1.5):

1. **Database, data warehouse, or other information repository.** This is one or a set of databases, data warehouses, spread sheets, or other kinds of information repositories. Data cleaning and data integration techniques may be performed on the data.

2. **Database or data warehouse server.** The database or data warehouse server is responsible for fetching the relevant data, based on the user's data mining request.

3. **Knowledge base.** This is the domain knowledge that is used to guide the search, or evaluate the interestingness of resulting patterns. Such knowledge can include **concept hierarchies**, used to organize attributes or attribute values into different levels of abstraction. Knowledge such as user beliefs, which can be used to assess a pattern's interestingness based on its unexpectedness, may also be included. Other examples of domain knowledge are additional interestingness constraints or thresholds, and metadata (e.g., describing data from multiple heterogeneous sources).

4. **Data mining engine.** This is essential to the data mining system and ideally consists of a set of functional modules for tasks such as characterization, association analysis, classification, evolution and deviation analysis.

5. **Pattern evaluation module.** This component typically employs interestingness measures (Section 1.5) and interacts with the data mining modules so as to *focus* the search towards interesting patterns. It may access interestingness thresholds stored in the knowledge base. Alternatively, the pattern evaluation module may be

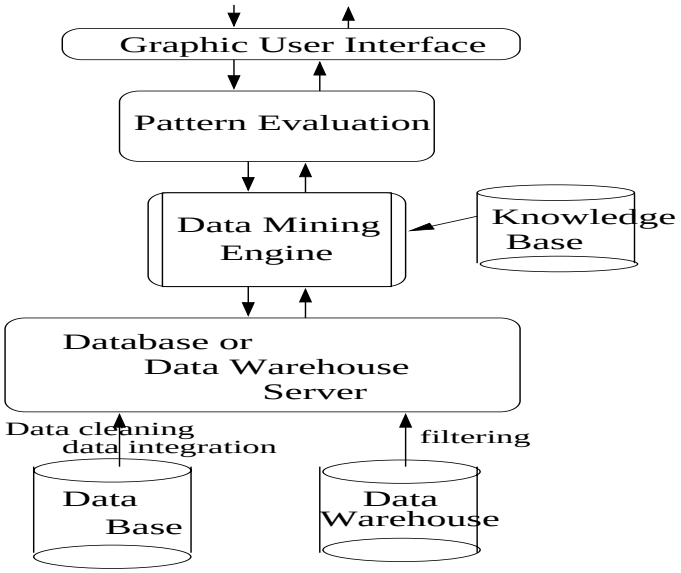


Figure 1.5: Architecture of a typical data mining system.

integrated with the mining module, depending on the implementation of the data mining method used. For efficient data mining, it is highly recommended to push the evaluation of pattern interestingness as deep as possible into the mining process so as to confine the search to only the interesting patterns.

6. **Graphical user interface.** This module communicates between users and the data mining system, allowing the user to interact with the system by specifying a data mining query or task, providing information to help focus the search, and performing exploratory data mining based on the intermediate data mining results. In addition, this component allows the user to browse database and data warehouse schemas or data structures, evaluate mined patterns, and visualize the patterns in different forms.

From a data warehouse perspective, data mining can be viewed as an advanced stage of on-line analytical processing (OLAP). However, data mining goes far beyond the narrow scope of summarization-style analytical processing of data warehouse systems by incorporating more advanced techniques for data understanding.

While there may be many "data mining systems" on the market, not all of them can perform true data mining. A data analysis system that does not handle large amounts of data can at most be categorized as a machine learning system, a statistical data analysis tool, or an experimental system prototype. A system that can only perform data or information retrieval, including finding aggregate values, or that performs deductive query answering in large databases should be more appropriately categorized as either a database system, an information retrieval system, or a deductive database system.

Data mining involves an integration of techniques from multiple disciplines such as database technology, statistics, machine learning, high performance computing, pattern recognition, neural networks, data visualization, information retrieval, image and signal processing, and spatial data analysis. We adopt a database perspective in our presentation of data mining in this book. That is, emphasis is placed on *efficient* and *scalable* data mining techniques for *large* databases. By performing data mining, interesting knowledge, regularities, or high-level information can be extracted from databases and viewed or browsed from different angles. The discovered knowledge can be applied to decision making, process control, information management, query processing, and so on. Therefore, data mining is considered as one of the most important frontiers in database systems and one of the most promising, new database applications in the information industry.

1.3 Data mining — on what kind of data?

In this section, we examine a number of different data stores on which mining can be performed. In principle, data mining should be applicable to any kind of information repository. This includes relational databases, data

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

Although this book assumes that readers have primitive knowledge of information systems, we provide a brief introduction to each of the major data repository systems listed above. In this section, we also introduce the fictitious *AllElectronics* store which will be used to illustrate concepts throughout the text.

1.3.1 Relational databases

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

A **relational database** is a collection of **tables**, each of which is assigned a unique name. Each table consists of a set of **attributes** (*columns* or *fields*) and usually stores a large number 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.

Consider the following example.

Example 1.1 The *AllElectronics* company is described by the following relation tables: *customer, item, employee*, and *branch*. Fragments of the tables described here are shown in Figure 1.6. The attribute which represents key or composite key component of each relation is underlined.

- The relation *customer* consists of a set of attributes, including a unique customer identity number (*cust_ID*), customer name, address, age, occupation, annual income, credit information, category, etc.

- Similarly, each of the relations *employee*, *branch*, and *items*, consists of a set of attributes, describing their properties.

- Tables can also be used to represent the relationships between or among multiple relation tables. For our example, these include *purchases* (customer purchases items, creating a sales transaction that is handled by an employee), *items_sold* (lists the items sold in a given transaction), and *works_at* (employee works at a branch of *AllElectronics*). □

Relational data can be accessed by **database queries** written in a relational query language, such as SQL, or with the assistance of graphical user interfaces. In the latter, the user may employ a menu, for example, to specify attributes to be included in the query, and the constraints on these attributes. 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 include aggregate functions such as `sum`, `avg` (average), `count`, `max` (maximum), and `min` (minimum). These allow you to find out things like "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 sales person had the highest amount of sales?".

When data mining is applied to relational databases, one can go further by *searching for trends or data patterns*. For example, data mining systems may 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, such as items whose sales are far from those expected in comparison with the previous year. Such deviations can then be further investigated, e.g., has there been a change in packaging of such items, or a significant increase in price?

Relational databases are one of the most popularly available and rich information repositories for data mining, and thus they are a major data form in our study of data mining.

customer

cust_ID	name	address	age	income	credit_info	...
C1	Smith, Sandy	5463 E. Hastings, Burnaby, BC, V5A 4S9, Canada	21	$27000	1	...
...	...	...	...	...	...	...

item

item_ID	name	brand	category	type	price	place_made	supplier	cost
I3	hi-res-TV	Toshiba	high resolution	TV	$988.00	Japan	NikoX	$600.00
I8	multidisc-CDplay	Sanyo	multidisc	CD player	$369.00	Japan	MusicFront	$120.00
...	...	...	...	...	...	...	...	...

employee

empl_ID	name	category	group	salary	commission
E55	Jones, Jane	home entertainment	manager	$18,000	2%
...	...	...	...	...	...

branch

branch_ID	name	address
B1	City Square	369 Cambie St., Vancouver, BC V5L 3A2, Canada
...	...	...

purchases

trans_ID	cust_ID	empl_ID	date	time	method_paid	amount
T100	C1	E55	09/21/98	15:45	Visa	$1357.00
...	...	...	...	...	...	...

items_sold

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

works_at

empl_ID	branch_ID
E55	B1
...	...

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

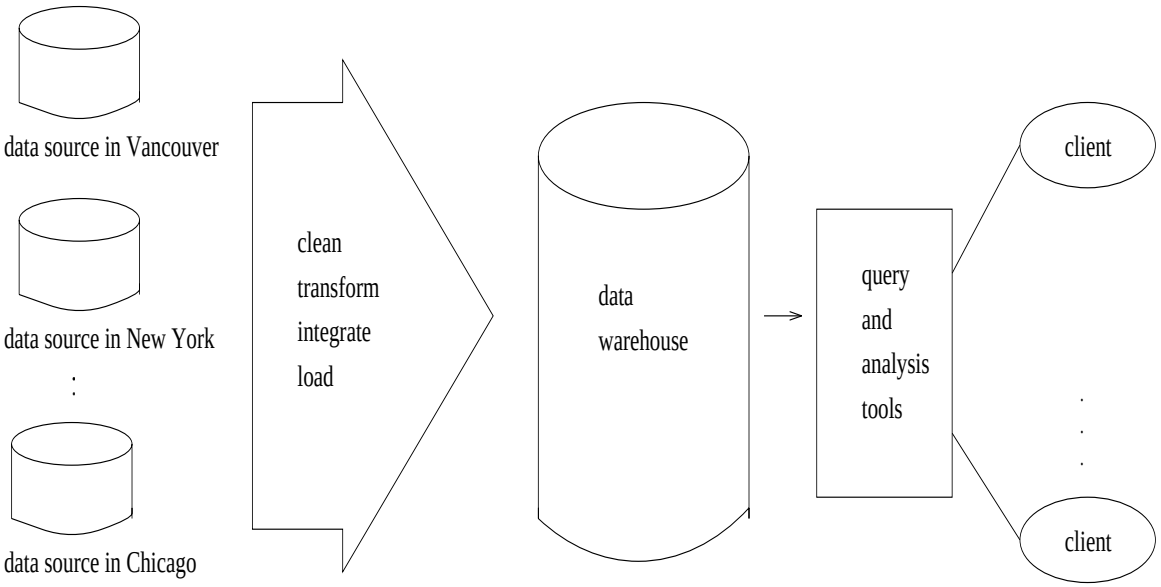


Figure 1.7: Architecture of a typical data warehouse.

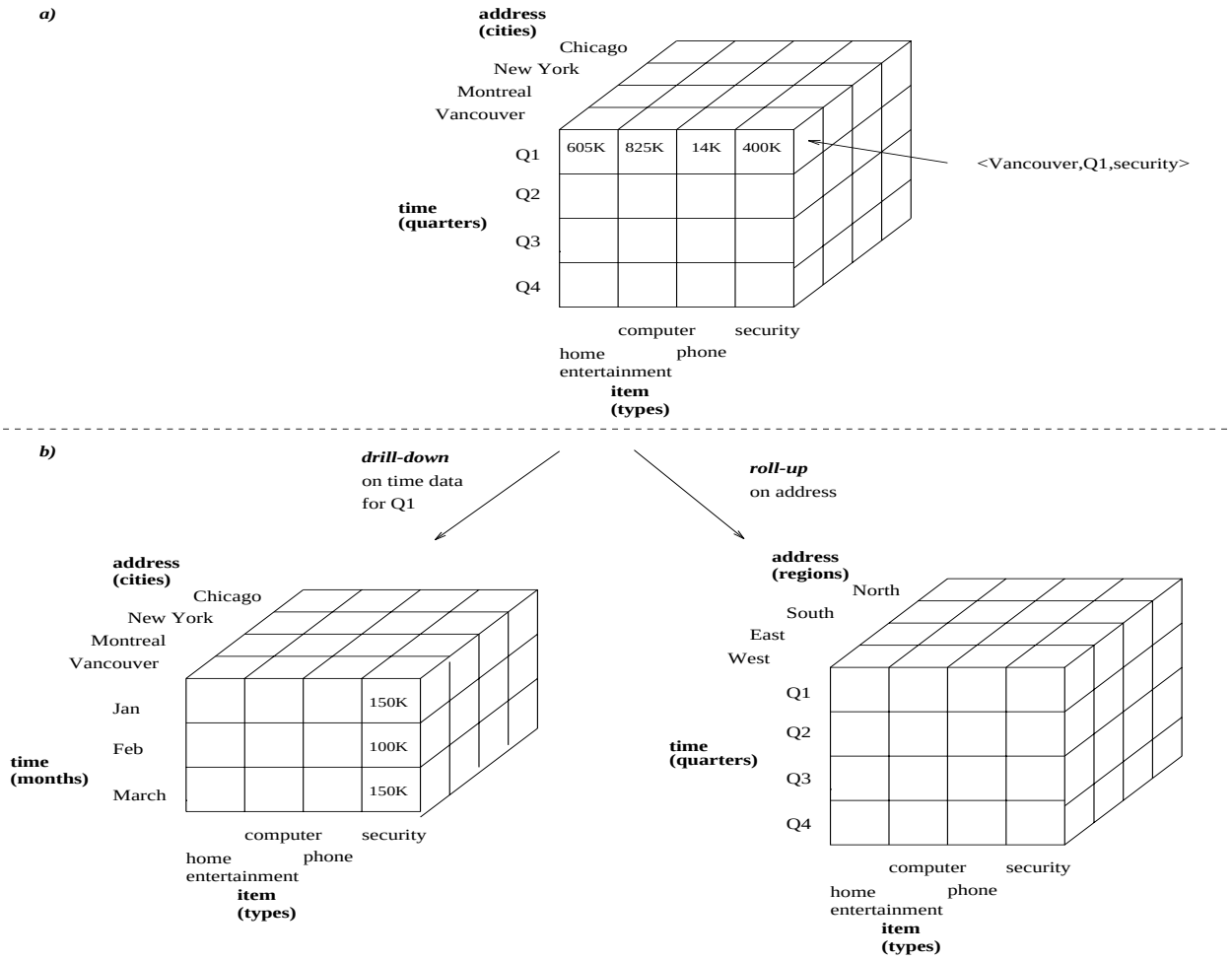


Figure 1.8: A multidimensional data cube, commonly used for data warehousing, *a)* showing summarized data for *AllElectronics* and *b)* showing summarized data resulting from drill-down and roll-up operations on the cube in *a)*.

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 which usually resides at a single site. Data warehouses are constructed via a process of data cleansing, data transformation, data integration, data loading, and periodic data refreshing. This process is studied in detail in Chapter 2. Figure 1.7 shows the basic architecture of a data warehouse for *AllElectronics*.

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

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

sales

trans_ID	list of item_ID's
T100	I1, I3, I8, I16
...	...

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

fast accessing of summarized data.

Example 1.2 A data cube for summarized sales data of *AllElectronics* is presented in Figure 1.8a). The cube has three **dimensions**: address (with *city* values *Chicago, New York, Montreal, 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*. For example, the total sales for $Q1$ of items relating to security systems in Vancouver is \$400K, 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. □

In research literature on data warehouses, the data cube structure that stores the primitive or lowest level of information is called a **base cuboid**. Its corresponding higher level multidimensional (cube) structures are called (non-base) **cuboids**. A base cuboid together with all of its corresponding higher level cuboids form a **data cube**.

By providing multidimensional data views and the precomputation of summarized data, data warehouse systems are well suited for **On-Line Analytical Processing**, or **OLAP**. OLAP operations make use of background knowledge regarding the domain of the data being studied in order 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 1.8b). For instance, one may drill down on sales data summarized by *quarter* to see the data summarized by *month*. Similarly, one may roll up on sales data summarized by *city* to view the data summarized by *region*.

Although data warehouse tools help support data analysis, additional tools for data mining are required to allow more in depth and automated analysis. Data warehouse technology is discussed in detail in Chapter 2.

1.3.3 Transactional databases

In general, a **transactional database** consists of a file where each record represents a transaction. A transaction typically includes a unique transaction identity number (*trans_ID*), and a list of the **items** making up the transaction (such as items purchased in a store). The transactional database may have additional tables associated with it, which contain other information regarding the sale, such as the date of the transaction, the customer ID number, the ID number of the sales person, and of the branch at which the sale occurred, and so on.

Example 1.3 Transactions can be stored in a table, with one record per transaction. A fragment of a transactional database for *AllElectronics* is shown in Figure 1.9. From the relational database point of view, the *sales* table in Figure 1.9 is a nested relation because the attribute "list of item_ID's" contains a set of *items*. Since 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 that of the table in Figure 1.9, or unfolded into a standard relation in a format similar to that of the *items_sold* table in Figure 1.6. □

As an analyst of the *AllElectronics* database, you may like to ask "Show me all the items purchased by Sandy Smith" or "How many transactions include item number I3?". Answering such queries may require a scan of the entire transactional database.

Suppose you would like to dig deeper into the data by asking "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 maximizing sales. For example, given the knowledge that printers are commonly purchased together with computers, you could offer

an expensive model of printers at a discount to customers buying selected computers, in the hopes of selling more of the expensive printers. A regular data retrieval system is not able to answer queries like the one above. However, data mining systems for transactional data can do so by identifying sets of items which are frequently sold together.

1.3.4 Advanced database systems and advanced database applications

Relational database systems have been widely used in business applications. With the advances of database technology, various kinds of advanced database systems have emerged and are undergoing development to address the requirements of new database applications.

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

In response to these needs, advanced database systems and specific application-oriented database systems have been developed. These include object-oriented and object-relational database systems, spatial database systems, temporal and time-series database systems, text and multimedia database systems, heterogeneous and legacy database systems, and the Web-based global information systems.

While such databases or information repositories require sophisticated facilities to efficiently store, retrieve, and update large amounts of complex data, they also provide fertile grounds and raise many challenging research and implementation issues for data mining.

1.4 Data mining functionalities — what kinds of patterns can be mined?

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

Data mining functionalities are used to specify the kind of patterns to be found in data mining tasks. In general, data mining tasks can be classified into two categories: **descriptive** and **predictive**. Descriptive mining tasks characterize the general properties of the data in the database. Predictive mining tasks perform inference on the current data in order to make predictions.

In some cases, users may have no idea of which kinds of patterns in their data may be interesting, and hence may like to search for several different kinds of patterns in parallel. Thus it is important to have a data mining system that can mine multiple kinds of patterns to accommodate different user expectations or applications. Furthermore, data mining systems should be able to discover patterns at various granularities (i.e., different levels of abstraction). To encourage interactive and exploratory mining, users should be able to easily "play" with the output patterns, such as by mouse clicking. Operations that can be specified by simple mouse clicks include adding or dropping a dimension (or an attribute), swapping rows and columns (**pivoting**, or axis rotation), changing dimension representations (e.g., from a 3-D cube to a sequence of 2-D cross tabulations, or **crosstabs**), or using OLAP roll-up or drill-down operations along dimensions. Such operations allow data patterns to be expressed from different angles of view and at multiple levels of abstraction.

Data mining systems should also allow users to specify hints to guide or focus the search for interesting patterns. Since some patterns may not hold for all of the data in the database, a measure of certainty or "trustworthiness" is usually associated with each discovered pattern.

Data mining functionalities, and the kinds of patterns they can discover, are described below.

1.4.1 Concept/class description: characterization and discrimination

Data 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 via (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 database query. For example, to study the characteristics of software products whose sales increased by 10% in the last year, one can collect the data related to such products by executing an SQL query.

There are several methods for effective data summarization and characterization. For instance, 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 Chapter 2 which discusses 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 described in Chapter 5.

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**). These different output forms and their transformations are discussed in Chapter 5.

Example 1.4 A data mining system should be able to produce a description summarizing the characteristics of customers who spend more than $1000 a year at *AllElectronics*. The result could be a general profile of the customers such as they are 40-50 years old, employed, and have excellent credit ratings. The system should allow users to drill-down on any dimension, such as on "employment" in order to view these customers according to their occupation. □

Data **discrimination** is a comparison of the general features of target class data objects with the general features of objets from one or a set of contrasting classes. The target and contrasting classes can be specified by the user, and the corresponding data objects retrieved through data base queries. For example, one may like to compare the general features of software products whose sales increased by 10% in the last year with those whose sales decreased by at least 30% during the same period.

The methods used for data discrimination are similar to those used for data characterization. The forms of output presentation are also similar, although discrimination descriptions should include comparative measures which help distinguish between the target and contrasting classes. Discrimination descriptions expressed in rule form are referred to as **discriminant rules**. The user should be able to manipulate the output for characteristic and discriminant descriptions.

Example 1.5 A data mining system should be able to compare two groups of *AllElectronics* customers, such as those who shop for computer products regularly (more than 4 times a month) vs. those who rarely shop for such products (i.e., less than three times a year). The resulting description could be a general, comparative profile of the customers such as 80% of the customers who frequently purchase computer products are between 20-40 years old and have a university education, whereas 60% of the customers who infrequently buy such products are either old or young, and have no university degree. Drilling-down on a dimension, such as *occupation*, or adding new dimensions, such as *income_level*, may help in finding even more discriminative features between the two classes. □

Concept description, including characterization and discrimination, is the topic of Chapter 5.

1.4.2 Association analysis

Association analysis is the discovery of *association rules* showing attribute-value conditions that occur frequently together in a given set of data. Association analysis is widely used for market basket or transaction data analysis.

More formally, **association rules** are of the form $X \Rightarrow Y$, i.e., "$A_1 \wedge \cdots \wedge A_m \rightarrow B_1 \wedge \cdots \wedge B_n$", where A_i (for $i \in \{1, \ldots, m\}$) and B_j (for $j \in \{1, \ldots, n\}$) are attribute-value pairs. The association rule $X \Rightarrow Y$ is interpreted as "database tuples that satisfy the conditions in X are also likely to satisfy the conditions in Y".

Example 1.6 Given the *AllElectronics* relational database, a data mining system may find association rules like

$$age(X, \text{“}20-29\text{”}) \wedge income(X, \text{“}20-30K\text{”}) \Rightarrow buys(X, \text{“}CD\ player\text{”}) \qquad [support = 2\%, confidence = 60\%]$$

meaning that of the *AllElectronics* customers under study, 2% (**support**) are 20-29 years of age with an income of 20-30K and have purchased a CD player at *AllElectronics*. There is a 60% probability (**confidence**, or certainty) that a customer in this age and income group will purchase a CD player.

Note that this is an association between 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**.

Suppose, as a marketing manager of *AllElectronics*, you would like to determine which items are frequently purchased together within the same transactions. An example of such a rule is

$$contains(T, \text{“}computer\text{”}) \Rightarrow contains(T, \text{“}software\text{”}) \qquad [support = 1\%, confidence = 50\%]$$

meaning that if a transaction T contains “computer”, there is a 50% chance that it contains “software” as well, and 1% of all of the transactions contain both. This association rule involves a single attribute or predicate (i.e., *contains*) which repeats. Association rules that contain a single predicate are referred to as **single-dimensional association rules**. Dropping the predicate notation, the above rule can be written simply as “$computer \Rightarrow software$ [1%, 50%]”. □

In recent years, many algorithms have been proposed for the efficient mining of association rules. Association rule mining is discussed in detail in Chapter 6.

1.4.3 Classification and prediction

Classification is the processing of finding a set of **models** (or functions) which describe and distinguish data classes or concepts, for the purposes of being able to use the model to predict the class of objects whose class label is unknown. The derived model is based on the analysis of a set of **training data** (i.e., data objects whose class label is known).

The derived model may be represented in various forms, such as *classification (IF-THEN) rules*, *decision trees*, *mathematical formulae*, or *neural networks*. A **decision tree** is a flow-chart-like tree structure, 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 be easily converted to classification rules. A **neural network** is a collection of linear threshold units that can be trained to distinguish objects of different classes.

Classification can be used for predicting the class label of data objects. However, in many applications, one may like to predict some missing or unavailable *data values* rather than class labels. This is usually the case when the predicted values are numerical data, and is often specifically referred to as **prediction**. Although prediction may refer to both data value prediction and class label prediction, it is usually confined to data value prediction and thus is distinct from classification. Prediction also encompasses the identification of distribution *trends* based on the available data.

Classification and prediction may need to be preceded by **relevance analysis** which attempts to identify attributes that do not contribute to the classification or prediction process. These attributes can then be excluded.

Example 1.7 Suppose, as sales manager of *AllElectronics*, you would like 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 would like 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 in the form of a decision tree. The decision tree, for instance, may identify *price* as being the single factor which best distinguishes the three classes. The tree may reveal that, after *price*, other features which help further distinguish objects of each class from 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 for the future. □

Chapter 7 discusses classification and prediction in further detail.

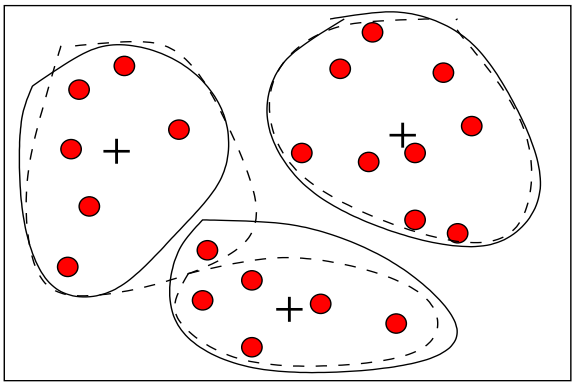


Figure 1.10: A 2-D plot of customer data with respect to customer locations in a city, showing three data clusters. Each cluster 'center' is marked with a '+'.

1.4.4 Clustering analysis

Unlike classification and predication, which analyze class-labeled data objects, **clustering** analyzes data objects without consulting a known class label. In general, the class labels are not present in the training data simply because they are not known to begin with. Clustering can be used to generate such labels. 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 very dissimilar to objects in other clusters. Each cluster that is 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 1.8 Clustering analysis can be performed on *AllElectronics* customer data in order 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. □

Clustering analysis forms the topic of Chapter 8.

1.4.5 Evolution and deviation analysis

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

Example 1.9 Suppose that you have the major stock market (time-series) data of the last several years available from the New York Stock Exchange and you would like to invest in shares of high-tech industrial companies. A data mining study of stock exchange data may identify stock evolution regularities for overall stocks and for the stocks of particular companies. Such regularities may help predict future trends in stock market prices, contributing to your decision making regarding stock investments. □

In the analysis of time-related data, it is often desirable not only to model the general evolutionary trend of the data, but also to identify data deviations which occur over time. **Deviations** are differences between measured values and corresponding references such as previous values or normative values. A data mining system performing deviation analysis, upon the detection of a set of deviations, may do the following: describe the characteristics of the deviations, try to explain the reason behind them, and suggest actions to bring the deviated values back to their expected values.

Example 1.10 A decrease in *total sales* at *AllElectronics* for the last month, in comparison to that of the same month of the last year, is a deviation pattern. Having detected a significant deviation, a data mining system may go further and attempt to explain the detected pattern (e.g., did the company have more sales personnel last year in comparison to the same period this year?). □

Data evolution and deviation analysis are discussed in Chapter 9.

1.5 Are all of the patterns interesting?

A data mining system has the potential to generate thousands or even millions of patterns, or rules. *Are all of the patterns interesting?* Typically not — only a small fraction of the patterns potentially generated would actually be of interest to any given user.

This raises some serious questions for data mining: *What makes a pattern interesting? Can a data mining system generate all of the interesting patterns? Can a data mining system generate only the interesting patterns?*

To answer the first question, a pattern is **interesting** if (1) it is *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 data samples that the given rule satisfies. Another objective measure for association rules is **confidence**, which assesses the degree of certainty of the detected association. It is defined as the conditional probability that a pattern Y is true given that X is true. More formally, support and confidence are defined as

$$support(X \Rightarrow Y) = Prob\{X \cup Y\}.$$

$$confidence(X \Rightarrow Y) = Prob\{Y|X\}.$$

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

Although objective measures help identify interesting patterns, they are insufficient unless combined with subjective measures that reflect the needs and interests of a particular user. For example, patterns describing the characteristics of customers who shop frequently at *AllElectronics* should interest the marketing manager, but may be of little interest to analysts studying the same database for patterns on employee performance. Furthermore, many patterns that are interesting by objective standards may represent common knowledge, and therefore, are actually uninteresting. **Subjective interestingness measures** are based on user beliefs in the data. These measures find patterns interesting if they are **unexpected** (contradicting a user belief) or offer strategic information on which the user can act. In the latter case, such patterns are referred to as **actionable**. Patterns that are expected can be interesting if they confirm a hypothesis that the user wished to validate, or resemble a user's hunch.

The second question, "*Can a data mining system generate* all *of the interesting patterns?*", refers to the **completeness** of a data mining algorithm. It is unrealistic and inefficient for data mining systems to generate all of the possible patterns. Instead, a focused search which makes use of interestingness measures should be used to control pattern generation. This is often sufficient to ensure the completeness of the algorithm. Association rule mining is an example where the use of 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 *the interesting patterns?"*, is an optimization problem in data mining. It is highly desirable for data mining systems to generate only the interesting patterns. This would be much more efficient for users and data mining systems, since neither would have to search through the patterns generated in order to identify the truely interesting ones. Such optimization remains a challenging issue in data mining.

Measures of pattern interestingness are essential for the efficient discovery of patterns of value to the given user. Such measures can be used after the data mining step in order to rank the discovered patterns according to their interestingness, filtering out the uninteresting ones. More importantly, such measures can be used to guide and

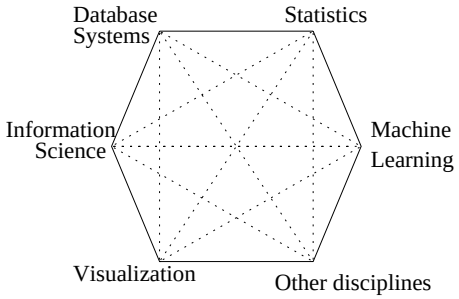


Figure 1.11: Data mining as a confluence of multiple disciplines.

constrain the discovery process, improving the search efficiency by pruning away subsets of the pattern space that do not satisfy pre-specified interestingness constraints.

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.

1.6 A classification of data mining systems

Data mining is an interdisciplinary field, the confluence of a set of disciplines (as shown in Figure 1.11), including database systems, statistics, machine learning, visualization, and information science. Moreover, depending on the data mining approach used, techniques from other disciplines may be applied, such as neural networks, fuzzy and/or rough set theory, knowledge representation, inductive logic programming, or high performance computing. Depending on the kinds of data to be mined or on the given data mining application, the data mining system may also integrate techniques from spatial data analysis, information retrieval, pattern recognition, image analysis, signal processing, computer graphics, Web technology, economics, or psychology.

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

- Classification according to the *kinds of databases* mined.

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

  For instance, if classifying according to data models, we may have a relational, transactional, object-oriented, object-relational, or data warehouse mining system. If classifying according to the special types of data handled, we may have a spatial, time-series, text, or multimedia data mining system, or a World-Wide Web mining system. Other system types include heterogeneous data mining systems, and legacy data mining systems.

- Classification according to the *kinds of knowledge* mined.

  Data mining systems can be categorized according to the kinds of knowledge they mine, i.e., based on data mining functionalities, such as characterization, discrimination, association, classification, clustering, trend and evolution analysis, deviation analysis, similarity analysis, etc. A comprehensive data mining system usually provides multiple and/or integrated data mining functionalities.

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

- Classification according to the *kinds of techniques* utilized.

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

Chapters 5 to 8 of this book are organized according to the various kinds of knowledge mined. In Chapter 9, we discuss the mining of different kinds of data on a variety of advanced and application-oriented database systems.

1.7 Major issues in data mining

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

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

   - *Mining different kinds of knowledge in databases.*
     Since different users can be interested in different kinds of knowledge, data mining should cover a wide spectrum of data analysis and knowledge discovery tasks, including data characterization, discrimination, association, classification, clustering, trend and deviation analysis, and similarity analysis. These tasks may use the same database in different ways and require the development of numerous data mining techniques.
   - *Interactive mining of knowledge at multiple levels of abstraction.*
     Since it is difficult to know exactly what can be discovered within a database, the data mining process should be *interactive*. For databases containing a huge amount of data, appropriate sampling technique can first be applied to facilitate interactive data exploration. Interactive mining allows users to focus the search for patterns, providing and refining data mining requests based on returned results. Specifically, knowledge should be mined by drilling-down, rolling-up, and pivoting through the data space and knowledge space interactively, similar to what OLAP can do on data cubes. In this way, the user can interact with the data mining system to view data and discovered patterns at multiple granularities and from different angles.
   - *Incorporation of background knowledge.*
     Background knowledge, or information regarding the domain under study, may be used to guide the discovery process and allow discovered patterns to be expressed in concise terms and at different levels of abstraction. Domain knowledge related to databases, such as integrity constraints and deduction rules, can help focus and speed up a data mining process, or judge the interestingness of discovered patterns.
   - *Data mining query languages and ad-hoc data mining.*
     Relational query languages (such as SQL) allow users to pose ad-hoc queries for data retrieval. In a similar vein, high-level **data mining query languages** need to be developed to allow users to describe ad-hoc data mining tasks by facilitating the specification of the relevant sets of data for analysis, the domain knowledge, the kinds of knowledge to be mined, and the conditions and interestingness constraints to be enforced on the discovered patterns. Such a language should be integrated with a database or data warehouse query language, and optimized for efficient and flexible data mining.
   - *Presentation and visualization of data mining results.*
     Discovered knowledge should be expressed in high-level languages, visual representations, or other expressive forms so that the knowledge can be easily understood and directly usable by humans. This is especially crucial if the data mining system is to be interactive. This requires the system to adopt expressive knowledge representation techniques, such as trees, tables, rules, graphs, charts, crosstabs, matrices, or curves.

- *Handling outlier or incomplete data.*

  The data stored in a database may reflect outliers — noise, exceptional cases, or incomplete data objects. These objects may confuse the analysis process, causing overfitting of the data to the knowledge model constructed. As a result, the accuracy of the discovered patterns can be poor. Data cleaning methods and data analysis methods which can handle outliers are required. While most methods discard outlier data, such data may be of interest in itself such as in fraud detection for finding unusual usage of telecommunication services or credit cards. This form of data analysis is known as **outlier mining**.

- *Pattern evaluation: the interestingness problem.*

  A data mining system can uncover thousands of patterns. Many of the patterns discovered may be uninteresting to the given user, representing common knowledge or lacking novelty. Several challenges remain regarding the development of techniques to assess the interestingness of discovered patterns, particularly with regard to subjective measures which estimate the value of patterns with respect to a given user class, based on user beliefs or expectations. The use of interestingness measures to guide the discovery process and reduce the search space is another active area of research.

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

   - *Efficiency and scalability of data mining algorithms.*

     To effectively extract information from a huge amount of data in databases, data mining algorithms must be **efficient** and **scalable**. That is, the running time of a data mining algorithm must be predictable and acceptable in large databases. Algorithms with exponential or even medium-order polynomial complexity will not be of practical use. From a database perspective on knowledge discovery, efficiency and scalability are key issues in the implementation of data mining systems. Many of the issues discussed above under *mining methodology and user-interaction* must also consider efficiency and scalability.

   - *Parallel, distributed, and incremental updating algorithms.*

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

3. **Issues relating to the diversity of database types.**

   - *Handling of relational and complex types of data.*

     There are many kinds of data stored in databases and data warehouses. Can we expect that a single data mining system can perform effective mining on all kinds of data? Since relational databases and data warehouses are widely used, the development of efficient and effective data mining systems for such data is important. However, other databases may contain complex data objects, hypertext and multimedia data, spatial data, temporal data, or transaction data. It is unrealistic to expect one system to mine all kinds of data due to the diversity of data types and different goals of data mining. Specific data mining systems should be constructed for mining specific kinds of data. Therefore, one may expect to have different data mining systems for different kinds of data.

   - *Mining information from heterogeneous databases and global information systems.*

     Local and wide-area computer networks (such as the Internet) connect many sources of data, forming huge, distributed, and heterogeneous databases. The discovery of knowledge from different sources of structured, semi-structured, or unstructured data with diverse data semantics poses great challenges to data mining. Data mining may help disclose high-level data regularities in multiple heterogeneous databases that are unlikely to be discovered by simple query systems and may improve information exchange and interoperability in heterogeneous databases.

The above issues are considered major requirements and challenges for the further evolution of data mining technology. Some of the challenges have been addressed in recent data mining research and development, *to a*

certain extent, and are now considered *requirements*, while others are still at the research stage. The issues, however, continue to stimulate further investigation and improvement. Additional issues relating to applications, privacy, and the social impact of data mining are discussed in Chapter 10, the final chapter of this book.

1.8 Summary

- **Database technology** has evolved from primitive file processing to the development of database management systems with query and transaction processing. Further progress has led to the increasing demand for efficient and effective data analysis and data understanding tools. This need is a result of the explosive growth in data collected from applications including business and management, government administration, scientific and engineering, and environmental control.
- **Data mining** is the task of discovering interesting patterns from large amounts of data where the data can be stored in databases, data warehouses, or other information repositories. It is a young interdisciplinary field, drawing from areas such as database systems, data warehousing, statistics, machine learning, data visualization, information retrieval, and high performance computing. Other contributing areas include neural networks, pattern recognition, spatial data analysis, image databases, signal processing, and inductive logic programming.
- A **knowledge discovery process** includes data cleaning, data integration, data selection, data transformation, data mining, pattern evaluation, and knowledge presentation.
- Data patterns can be mined from many different kinds of **databases**, such as relational databases, data warehouses, and transactional, object-relational, and object-oriented databases. Interesting data patterns can also be extracted from other kinds of **information repositories**, including spatial, time-related, text, multimedia, and legacy databases, and the World-Wide Web.
- 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 some data analysis capabilities, collectively referred to as **OLAP (On-Line Analytical Processing)**. OLAP operations include drill-down, roll-up, and pivot.
- **Data mining functionalities** include the discovery of concept/class descriptions (i.e., characterization and discrimination), association, classification, prediction, clustering, trend analysis, deviation analysis, and similarity analysis. Characterization and discrimination are forms of data summarization.
- A pattern represents **knowledge** if it is easily understood by humans, valid on test data with some degree of certainty, potentially useful, novel, or validates a hunch about which the user was curious. Measures of **pattern interestingness**, either *objective* or *subjective*, can be used to guide the discovery process.
- **Data mining systems** can be **classified** according to the kinds of databases mined, the kinds of knowledge mined, or the techniques used.
- Efficient and effective data mining in large databases poses numerous **requirements** and great **challenges** to researchers and developers. The issues involved include data mining methodology, user-interaction, performance and scalability, and the processing of a large variety of data types. Other issues include the exploration of data mining applications, and their social impacts.

Exercises

1. What is data mining? In your answer, address the following:
   (a) Is it another hype?
   (b) Is it a simple transformation of technology developed from databases, statistics, and machine learning?
   (c) Explain how the evolution of database technology led to data mining.
   (d) Describe the steps involved in data mining when viewed as a process of knowledge discovery.

2. Present an example where data mining is crucial to the success of a business. What data mining functions does this business need? Can they be performed alternatively by data query processing or simple statistical analysis?

3. How is a data warehouse different from a database? How are they similar to each other?

4. Define each of the following data mining functionalities: characterization, discrimination, association, classification, prediction, clustering, and evolution and deviation analysis. Give examples of each data mining functionality, using a real-life database that you are familiar with.

5. Suppose your task as a software engineer at *Big-University* is to design a data mining system to examine their university course database, which contains the following information: the name, address, and status (e.g., undergraduate or graduate) of each student, and their cumulative grade point average (GPA). Describe the architecture you would choose. What is the purpose of each component of this architecture?

6. Based on your observation, 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?

7. What is the difference between discrimination and classification? Between characterization and clustering? Between classification and prediction? For each of these pairs of tasks, how are they similar?

8. Describe three challenges to data mining regarding data mining methodology and user-interaction issues.

9. Describe two challenges to data mining regarding performance issues.

Bibliographic Notes

The book *Knowledge Discovery in Databases*, edited by Piatetsky-Shapiro and Frawley [26], is an early collection of research papers on knowledge discovery in databases. The book *Advances in Knowledge Discovery and Data Mining*, edited by Fayyad et al. [10], is a good collection of recent research results on knowledge discovery and data mining. Other books on data mining include *Predictive Data Mining* by Weiss and Indurkhya [37], and *Data Mining* by Adriaans and Zantinge [1]. There are also books containing collections of papers on particular aspects of knowledge discovery, such as *Machine Learning & Data Mining: Methods and Applications*, edited by Michalski, Bratko, and Kubat [20], *Rough Sets, Fuzzy Sets and Knowledge Discovery*, edited by Ziarko [39], as well as many tutorial notes on data mining, such as *Tutorial Notes of 1999 International Conference on Knowledge Disocvery and Data Mining* (*KDD99*) published by ACM Press.

KDD Nuggets is a regular, free electronic newsletter containing information relevant to knowledge discovery and data mining. Contributions can be e-mailed with a descriptive subject line (and a URL) to "gps@kdnuggets.com". Information regarding subscription can be found at "http://www.kdnuggets.com/subscribe.html". *KDD Nuggets* has been moderated by Piatetsky-Shapiro since 1991. The Internet site, *Knowledge Discovery Mine*, located at "http://www.kdnuggets.com/", contains a good collection of KDD-related information.

The research community of data mining set up a new academic organization called ACM-SIGKDD, a Special Interested Group on Knowledge Discovery in Databases under ACM in 1998. The community started its first international conference on knowledge discovery and data mining in 1995 [12]. The conference evolved from the four *international workshops on knowledge discovery in databases*, held from 1989 to 1994 [7, 8, 13, 11]. ACM-SIGKDD is organizing its first, but the fifth international conferences on knowledge discovery and data mining (KDD'99). A new journal, *Data Mining and Knowledge Discovery*, published by Kluwers Publishers, has been available since 1997.

Research in data mining has also been published in major textbooks, conferences and journals on databases, statistics, machine learning, and data visualization. References to such sources are listed below.

Popular textbooks on database systems include *Database System Concepts, 3rd ed.*, by Silberschatz, Korth, and Sudarshan [30], *Fundamentals of Database Systems, 2nd ed.*, by Elmasri and Navathe [9], and *Principles of Database and Knowledge-Base Systems, Vol. 1*, by Ullman [36]. For an edited collection of seminal articles on database systems, see *Readings in Database Systems* by Stonebraker [32]. Overviews and discussions on the achievements and research challenges in database systems can be found in Stonebraker et al. [33], and Silberschatz, Stonebraker, and Ullman [31].

Many books on data warehouse technology, systems and applications have been published in the last several years, such as *The Data Warehouse Toolkit* by Kimball [17], and *Building the Data Warehouse* by Inmon [14]. Chaudhuri and Dayal [3] present a 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 *ACM-SIGMOD International Conference on Management of Data (SIGMOD), International Conference on Very Large Data Bases (VLDB), ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS), International Conference on Data Engineering (ICDE), International Conference on Extending Database Technology (EDBT), International Conference on Database Theory (ICDT), International Conference on Information and Knowledge Management (CIKM)*, and *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), Journal of ACM (JACM), Information Systems, The VLDB Journal, Data and Knowledge Engineering*, and *International Journal of Intelligent Information Systems (JIIS)*.

There are many textbooks covering different topics in statistical analysis, such as *Probability and Statistics for Engineering and the Sciences, 4th ed.* by Devore [4], *Applied Linear Statistical Models, 4th ed.* by Neter et al. [25], *An Introduction to Generalized Linear Models* by Dobson [5], *Applied Statistical Time Series Analysis* by Shumway [29], and *Applied Multivariate Statistical Analysis, 3rd ed.* by Johnson and Wichern [15].

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 source of publication include the *Journal of the Royal Statistical Society*, *The Annals of Statistics*, *Journal of American Statistical Association*, *Technometrics*, and *Biometrika*.

Textbooks and reference books on machine learning include *Machine Learning* by Mitchell [24], *Machine Learning, An Artificial Intelligence Approach*, Vols. 1-4, edited by Michalski et al. [21, 22, 18, 23], *C4.5: Programs for Machine Learning* by Quinlan [27], and *Elements of Machine Learning* by Langley [19]. The book *Computer Systems that Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems*, by Weiss and Kulikowski [38], compares classification and prediction methods from several different fields, including statistics, machine learning, neural networks, and expert systems. For an edited collection of seminal articles on machine learning, see *Readings in Machine Learning* by Shavlik and Dietterich [28].

Machine learning research is published in the proceedings of several large machine learning and artificial intelligence conferences, including the *International Conference on Machine Learning (ML), ACM Conference on Computational Learning Theory (COLT), International Joint Conference on Artificial Intelligence (IJCAI)*, and *American Association of Artificial Intelligence Conference (AAAI)*. Other sources of publication include major machine learning, artificial intelligence, and knowledge system journals, some of which have been mentioned above. Others include *Machine Learning (ML), Artificial Intelligence Journal (AI)* and *Cognitive Science*. An overview of classification from a statistical pattern recognition perspective can be found in Duda and Hart [6].

Pioneering work on data visualization techniques is described in *The Visual Display of Quantitative Information* [34] and *Envisioning Information* [35], both by Tufte, and *Graphics and Graphic Information Processing* by Bertin [2]. *Visual Techniques for Exploring Databases* by Keim [16] presents a broad tutorial on visualization for data mining. Major conferences and symposiums on visualization include *ACM Human Factors in Computing Systems (CHI)*, *Visualization*, and *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*.

Bibliography

[1] P. Adriaans and D. Zantinge. *Data Mining.* Addison-Wesley: Harlow, England, 1996.

[2] J. Bertin. *Graphics and Graphic Information Processing.* Berlin, 1981.

[3] S. Chaudhuri and U. Dayal. An overview of data warehousing and OLAP technology. *ACM SIGMOD Record*, 26:65–74, 1997.

[4] J. L. Devore. *Probability and Statistics for Engineering and the Science, 4th ed.* Duxbury Press, 1995.

[5] A. J. Dobson. *An Introduction to Generalized Linear Models.* Chapman and Hall, 1990.

[6] R. Duda and P. Hart. *Pattern Classification and Scene Analysis.* Wiley: New York, 1973.

[7] G. Piatetsky-Shapiro (ed.). *Notes of IJCAI'89 Workshop Knowledge Discovery in Databases (KDD'89).* Detroit, Michigan, July 1989.

[8] G. Piatetsky-Shapiro (ed.). *Notes of AAAI'91 Workshop Knowledge Discovery in Databases (KDD'91).* Anaheim, CA, July 1991.

[9] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems, 2nd ed.* Bemjamin/Cummings, 1994.

[10] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy (eds.). *Advances in Knowledge Discovery and Data Mining.* AAAI/MIT Press, 1996.

[11] U.M. Fayyad and R. Uthurusamy (eds.). *Notes of AAAI'94 Workshop Knowledge Discovery in Databases (KDD'94).* Seattle, WA, July 1994.

[12] U.M. Fayyad and R. Uthurusamy (eds.). *Proc. 1st Int. Conf. Knowledge Discovery and Data Mining (KDD'95).* AAAI Press, Aug. 1995.

[13] U.M. Fayyad, R. Uthurusamy, and G. Piatetsky-Shapiro (eds.). *Notes of AAAI'93 Workshop Knowledge Discovery in Databases (KDD'93).* Washington, DC, July 1993.

[14] W. H. Inmon. *Building the Data Warehouse.* John Wiley, 1996.

[15] R. A. Johnson and D. W. Wickern. *Applied Multivariate Statistical Analysis, 3rd ed.* Prentice Hall, 1992.

[16] D. A. Keim. Visual techniques for exploring databases. In *Tutorial Notes, 3rd Int. Conf. on Knowledge Discovery and Data Mining (KDD'97)*, Newport Beach, CA, Aug. 1997.

[17] R. Kimball. *The Data Warehouse Toolkit.* John Wiley & Sons, New York, 1996.

[18] Y. Kodratoff and R. S. Michalski. *Machine Learning, An Artificial Intelligence Approach, Vol. 3.* Morgan Kaufmann, 1990.

[19] P. Langley. *Elements of Machine Learning.* Morgan Kaufmann, 1996.

[20] R. S. Michalski, I. Bratko, and M. Kubat. *Machine Learning and Data Mining: Methods and Applications.* John Wiley & Sons, 1998.

[21] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell. *Machine Learning, An Artificial Intelligence Approach, Vol. 1.* Morgan Kaufmann, 1983.

[22] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell. *Machine Learning, An Artificial Intelligence Approach, Vol. 2.* Morgan Kaufmann, 1986.

[23] R. S. Michalski and G. Tecuci. *Machine Learning, A Multistrategy Approach, Vol. 4.* Morgan Kaufmann, 1994.

[24] T. M. Mitchell. *Machine Learning.* McGraw Hill, 1997.

[25] J. Neter, M. H. Kutner, C. J. Nachtsheim, and L. Wasserman. *Applied Linear Statistical Models, 4th ed.* Irwin: Chicago, 1996.

[26] G. Piatetsky-Shapiro and W. J. Frawley. *Knowledge Discovery in Databases.* AAAI/MIT Press, 1991.

[27] J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.

[28] J.W. Shavlik and T.G. Dietterich. *Readings in Machine Learning.* Morgan Kaufmann, 1990.

[29] R. H. Shumway. *Applied Statistical Time Series Analysis.* Prentice Hall, 1988.

[30] A. Silberschatz, H. F. Korth, and S. Sudarshan. *Database System Concepts, 3ed.* McGraw-Hill, 1997.

[31] A. Silberschatz, M. Stonebraker, and J. D. Ullman. Database research: Achievements and opportunities into the 21st century. *ACM SIGMOD Record*, 25:52–63, March 1996.

[32] M. Stonebraker. *Readings in Database Systems, 2ed.* Morgan Kaufmann, 1993.

[33] M. Stonebraker, R. Agrawal, U. Dayal, E. Neuhold, and A. Reuter. DBMS research at a crossroads: The vienna update. In *Proc. 19th Int. Conf. Very Large Data Bases*, pages 688–692, Dublin, Ireland, Aug. 1993.

[34] E. R. Tufte. *The Visual Display of Quantitative Information.* Graphics Press, Cheshire, CT, 1983.

[35] E. R. Tufte. *Envisioning Information.* Graphics Press, Cheshire, CT, 1990.

[36] J. D. Ullman. *Principles of Database and Knowledge-Base Systems, Vol. 1.* Computer Science Press, 1988.

[37] S. M. Weiss and N. Indurkhya. *Predictive Data Mining.* Morgan Kaufmann, 1998.

[38] S. M. Weiss and C. A. Kulikowski. *Computer Systems that Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems.* Morgan Kaufman, 1991.

[39] W. Ziarko. *Rough Sets, Fuzzy Sets and Knowledge Discovery.* Springer-Verlag, 1994.

Contents

Chapter 2

Data Warehouse and OLAP Technology for Data Mining

The construction of data warehouses, which involves data cleaning and data integration, can be viewed as an important preprocessing step for data mining. Moreover, data warehouses provide *on-line analytical processing (OLAP)* tools for the interactive analysis of multidimensional data of varied granularities, which facilitates effective data mining. Furthermore, many other data mining functions such as classification, prediction, association, and clustering, can be integrated with OLAP operations to enhance interactive mining of knowledge at multiple levels of abstraction. Hence, data warehouse has become an increasingly important platform for data analysis and on-line analytical processing and will provide an effective platform for data mining. Therefore, prior to presenting a systematic coverage of data mining technology in the remainder of this book, we devote this chapter to an overview of data warehouse technology. Such an overview is essential for understanding data mining technology.

In this chapter, you will learn the basic concepts, general architectures, and major implementation techniques employed in data warehouse and OLAP technology, as well as their relationship with data mining.

2.1 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. A large number of organizations have found that 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 keep customers by learning more about their needs.

"So", you may ask, full of intrigue, *"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 database that is maintained separately from an organization's operational databases. Data warehouse systems allow for the integration of a variety of application systems. They support information processing by providing a solid platform of consolidated, historical data for analysis.

According to W. 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." (Inmon 1992). 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, vendor, 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 around 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 on-line 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 a historical perspective (e.g., the past 5-10 years). Every key structure in the data warehouse contains, either implicitly or explicitly, an element of time.
- **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 and stores the information on which 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.

"OK", you now ask, *"what, then, is data warehousing?"*

Based on the above, we view **data warehousing** as the *process of constructing and using data warehouses*. The construction of a data warehouse requires data integration, data cleaning, 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 are using 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 the 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*. Many organizations typically collect diverse kinds of data and maintain large databases from multiple, heterogeneous, autonomous, and distributed information sources. To integrate such data, and provide easy and efficient access to it is highly desirable, yet challenging. Much effort has been spent in the database industry and research community towards 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. A variety of data joiner and data blade products belong to this category. 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 for resources with processing at local sources. It is inefficient and potentially expensive for frequent queries, especially for queries requiring aggregations.

Data warehousing provides an interesting alternative to the traditional approach of heterogeneous database integration described above. 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 on-line 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 since 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 historical information and support complex multidimensional queries. As a result, data warehousing has become very popular in industry.

Differences between operational database systems and data warehouses

Since 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 on-line operational database systems is to perform on-line transaction and query processing. These systems are called **on-line 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 the different users. These systems are known as **on-line analytical processing (OLAP)** systems.

The major distinguishing features between OLTP and OLAP are summarized as follows.

1. **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.

2. **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 historical data, provides facilities for summarization and aggregation, and stores and manages information at different levels of granularity. These features make the data easier for use in informed decision making.

3. **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 *snowflake* model (to be discussed in Section 2.2.2), and a subject-oriented database design.

4. **View**: An OLTP system focuses mainly on the current data within an enterprise or department, without referring to historical 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.

5. **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 (since most data warehouses store historical rather than up-to-date information), although many could be complex queries.

Other features which distinguish between OLTP and OLAP systems include database size, frequency of operations, and performance metrics. These are summarized in Table 2.1.

But, why have a separate data warehouse?

"Since operational databases store huge amounts of data", you observe, *"why not perform on-line 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, such as 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 groups of data 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 several transactions. Concurrency control and recovery mechanisms, such as 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.

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	E-R based, application-oriented	star/snowflake, subject-oriented
Data	current; guaranteed up-to-date	historical; 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
# of records accessed	tens	millions
# of users	thousands	hundreds
DB size	100 MB to GB	100 GB to TB
Priority	high performance, high availability	high flexibility, end-user autonomy
Metric	transaction throughput	query throughput, response time

Table 2.1: Comparison between OLTP and OLAP systems.

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 support requires historical data, whereas operational databases do not typically maintain historical data. In this context, the data in operational databases, though abundant, is usually far from complete for decision making. Decision support requires consolidation (such as aggregation and summarization) of data from heterogeneous sources, resulting in high quality, cleansed and integrated data. In contrast, operational databases contain only detailed raw data, such as transactions, which need to be consolidated before analysis. Since the two systems provide quite different functionalities and require different kinds of data, it is necessary to maintain separate databases.

2.2 A multidimensional data model

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. You will also learn about concept hierarchies and how they can be used in basic OLAP operations to allow interactive mining at multiple levels of abstraction.

2.2.1 From tables to data cubes

"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, like *sales*, for instance. This theme

is represented by a fact table. **Facts** are numerical 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 later 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 which is, in fact, a table 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 2.2. 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*.

time (quarter)	Sales for all **locations** in Vancouver			
	item (type)			
	home entertainment	computer	phone	security
Q1	605K	825K	14K	400K
Q2	680K	952K	31K	512K
Q3	812K	1023K	30K	501K
Q4	927K	1038K	38K	580K

Table 2.2: A 2-D view of sales data for *AllElectronics* according to the dimensions *time* and *item*, where the sales are from branches located in the city of Vancouver. The measure displayed is *dollars_sold*.

	location = "Vancouver"				**location** = "Montreal"				**location** = "New York"				**location** = "Chicago"			
time	**item**				**item**				**item**				**item**			
	home ent.	comp.	phone	sec.	home ent.	comp.	phone	sec.	home ent.	comp.	phone	sec.	home ent.	comp.	phone	sec.
Q1	605K	825K	14K	400K	818K	746K	43K	591K	1087K	968K	38K	872K	854K	882K	89K	623K
Q2	680K	952K	31K	512K	894K	769K	52K	682K	1130K	1024K	41K	925K	943K	890K	64K	698K
Q3	812K	1023K	30K	501K	940K	795K	58K	728K	1034K	1048K	45K	1002K	1032K	924K	59K	789K
Q4	927K	1038K	38K	580K	978K	864K	59K	784K	1142K	1091K	54K	984K	1129K	992K	63K	870K

Table 2.3: A 3-D view of sales data for *AllElectronics*, according to the dimensions *time*, *item*, and *location*. The measure displayed is *dollars_sold*.

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*, *item*, as well as *location*. These 3-D data are shown in Table 2.3. The 3-D data of Table 2.3 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 2.1.

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 2.2. If we continue in this way, we may display any n-D data as a series of $(n-1)$-D "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.

The above tables show the data at different degrees of summarization. In the data warehousing research literature, a data cube such as each of the above is referred to as a **cuboid**. Given a set of dimensions, we can construct a *lattice* of cuboids, each showing the data at a different level of summarization, or `group by` (i.e., summarized by a different subset of the dimensions). The lattice of cuboids is then referred to as a **data cube**. Figure 2.8 shows a lattice of cuboids forming a data cube for the dimensions *time*, *item*, *location*, and *supplier*.

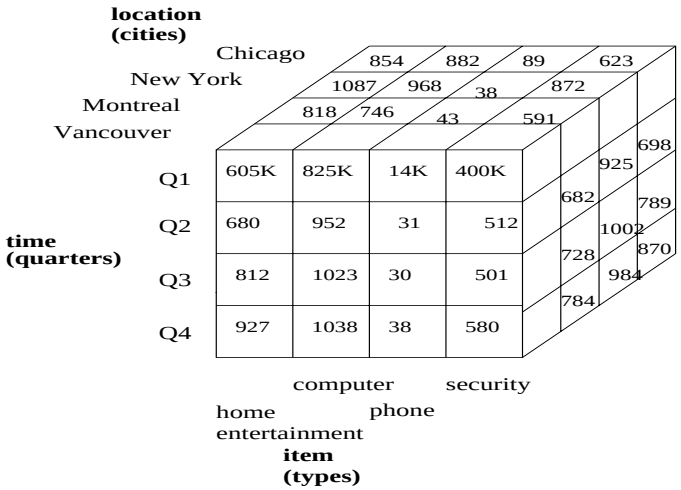


Figure 2.1: A 3-D data cube representation of the data in Table 2.3, according to the dimensions *time*, *item*, and *location*. The measure displayed is *dollars_sold*.

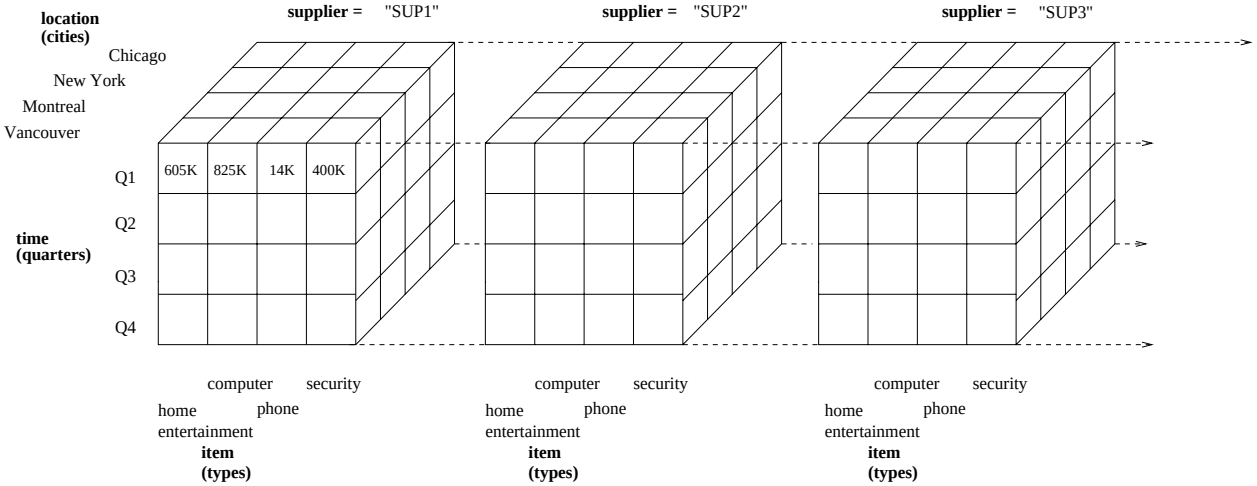


Figure 2.2: A 4-D data cube representation of sales data, according to the dimensions *time*, *item*, *location*, and *supplier*. The measure displayed is *dollars_sold*.

The cuboid which holds the lowest level of summarization is called the **base cuboid**. For example, the 4-D cuboid in Figure 2.2 is the base cuboid for the given *time*, *item*, *location*, and *supplier* dimensions. Figure 2.1 is a 3-D (non-base) cuboid for *time*, *item*, 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 for all four dimensions. The apex cuboid is typically denoted by all.

2.2.2 Stars, snowflakes, and fact constellations: schemas for multidimensional databases

The entity-relationship data model is commonly used in the design of relational databases, where a database schema consists of a set of entities or objects, and the relationships between them. Such a data model is appropriate for on-line transaction processing. Data warehouses, however, require a concise, subject-oriented schema which facilitates on-line data analysis.

The most popular data model for data warehouses is a **multidimensional model**. This model can exist in the form of a **star schema**, a **snowflake schema**, or a **fact constellation schema**. Let's have a look at each of these schema types.

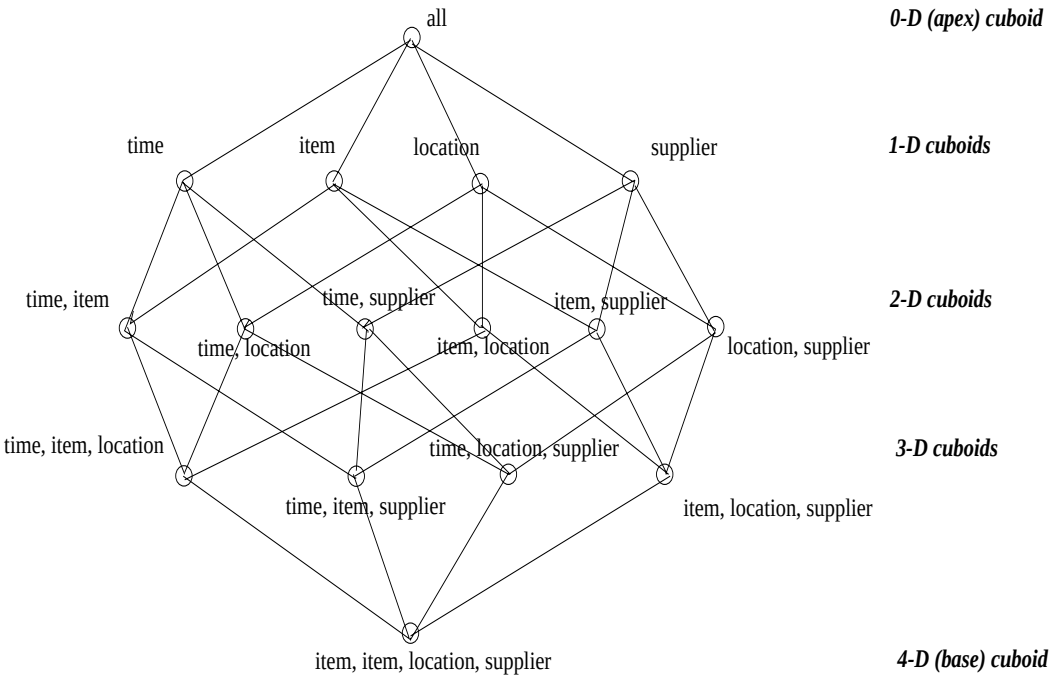


Figure 2.3: Lattice of cuboids, making up a 4-D data cube for the dimensions *time*, *item*, *location*, and *supplier*. Each cuboid represents a different degree of summarization.

- **Star schema**: The star schema is a modeling paradigm in which the data warehouse contains (1) a large central table (**fact table**), 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.

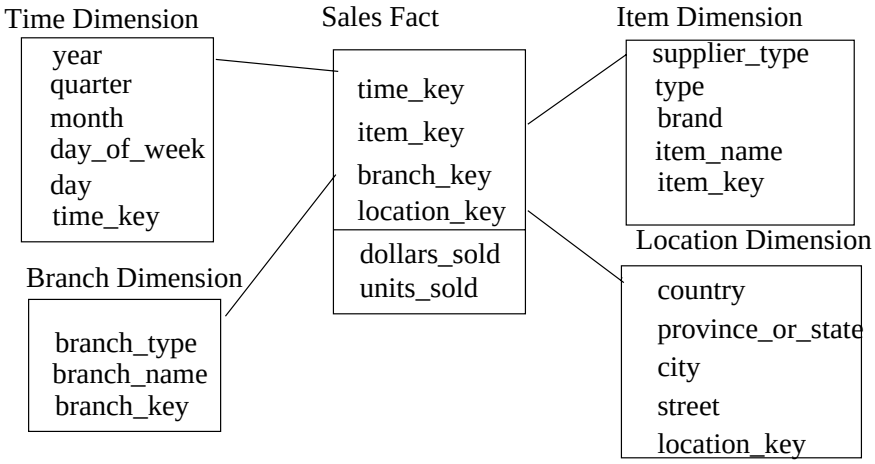


Figure 2.4: Star schema of a data warehouse for sales.

Example 2.1 An example of a star schema for *AllElectronics* sales is shown in Figure 2.4. Sales are considered along four dimensions, namely *time, item, branch*, and *location*. The schema contains a central fact table for *sales* which contains keys to each of the four dimensions, along with two measures: *dollars_sold* and *units_sold*. □

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, "Vancouver" and "Victoria" are both cities in the Canadian province of British Columbia. Entries for such cities in the location dimension table will create redundancy among the attributes *province_or_state* and *country*, i.e., (.., `Vancouver, British Columbia, Canada`) and (.., `Victoria, British Columbia, Canada`). 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. Such a table is easy to maintain and also saves storage space because a large dimension table can be extremely large when the dimensional structure is included as columns. Since much of this space is redundant data, creating a normalized structure will reduce the overall space requirement. However, 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. Performance benchmarking can be used to determine what is best for your design.

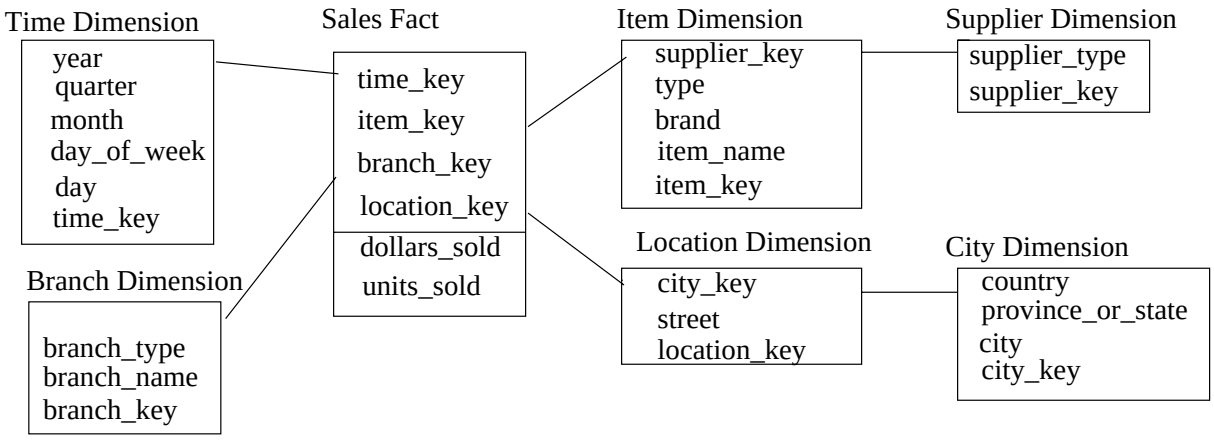


Figure 2.5: Snowflake schema of a data warehouse for sales.

Example 2.2 An example of a snowflake schema for *AllElectronics* sales is given in Figure 2.5. Here, the *sales* fact table is identical to that of the star schema in Figure 2.4. 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 *supplier_key*, *type*, *brand*, *item_name*, and *item_key*, the latter of which is linked to the *supplier* dimension table, containing *supplier_type* and *supplier_key* information. Similarly, the single dimension table for *location* in the star schema can be normalized into two tables: new *location* and *city*. The *location_key* of the new *location* table now links to the *city* dimension. Notice that further normalization can be performed on *province_or_state* and *country* in the snowflake schema shown in Figure 2.5, when desirable. □

A compromise between the star schema and the snowflake schema is to adopt a **mixed schema** where only the very large dimension tables are normalized. Normalizing large dimension tables saves storage space, while keeping small dimension tables unnormalized may reduce the cost and performance degradation due to joins on multiple dimension tables. Doing both may lead to an overall performance gain. However, careful performance tuning could be required to determine which dimension tables should be normalized and split into multiple tables.

- **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**.

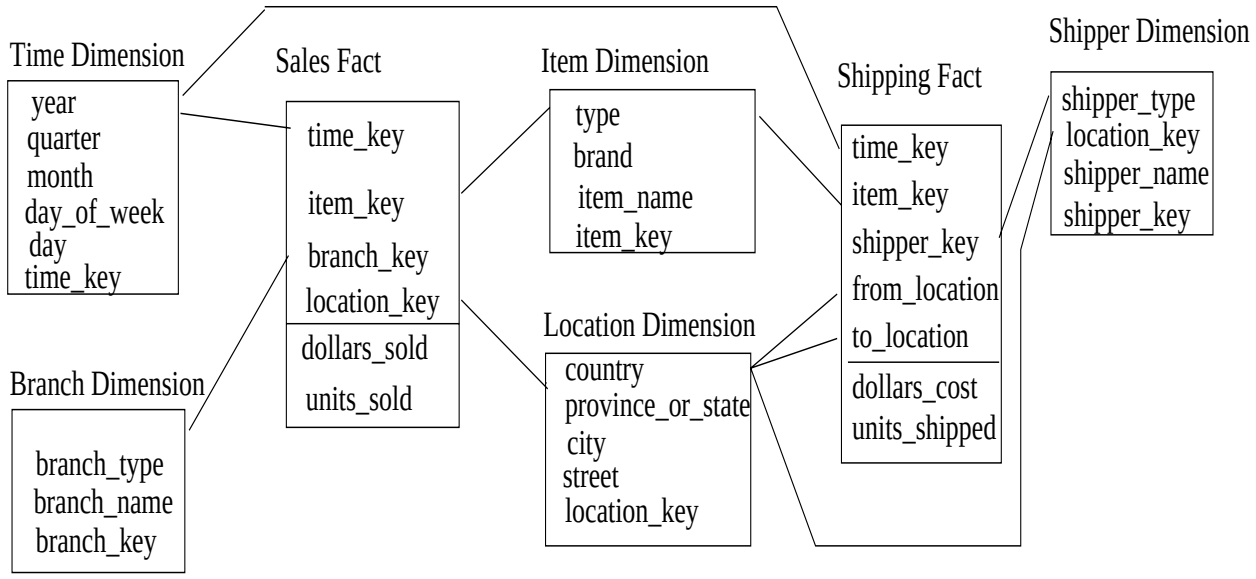


Figure 2.6: Fact constellation schema of a data warehouse for sales and shipping.

Example 2.3 An example of a fact constellation schema is shown in Figure 2.6. This schema specifies two fact tables, *sales* and *shipping*. The *sales* table definition is identical to that of the star schema (Figure 2.4). The *shipping* table has five dimensions, or keys: *time_key, item_key, shipper_key, from_location*, and *to_location*, and two measures: *dollars_cost* 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 both 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 are popular since each are geared towards modeling single subjects.

2.2.3 Examples for defining star, snowflake, and fact constellation schemas

"How can I define a multidimensional schema for my data?"

Just as relational query languages like SQL can be used to specify relational queries, a **data mining query language** can be used to specify data mining tasks. In particular, we examine an SQL-based data mining query language called **DMQL** which contains language primitives for defining data warehouses and data marts. Language primitives for specifying other data mining tasks, such as the mining of concept/class descriptions, associations, classifications, and so on, will be introduced in Chapter 4.

Data warehouses and data marts can be defined using two language primitives, one for *cube definition* and one for *dimension definition*. The *cube definition* statement has the following syntax.

define cube ⟨cube_name⟩ [⟨dimension_list⟩] : ⟨measure_list⟩

The *dimension definition* statement has the following syntax.

define dimension ⟨dimension_name⟩ as (⟨attribute_or_subdimension_list⟩)

Let's look at examples of how to define the star, snowflake and constellations schemas of Examples 2.1 to 2.3 using DMQL. DMQL keywords are displayed in sans serif font.

Example 2.4 The star schema of Example 2.1 and Figure 2.4 is defined in DMQL as follows.

```
define cube sales_star [time, item, branch, location]:
                        dollars_sold = sum(sales_in_dollars), units_sold = count(*)
define dimension time as (time_key, day, day_of_week, month, quarter, year)
define dimension item as (item_key, item_name, brand, type, supplier_type)
define dimension branch as (branch_key, branch_name, branch_type)
define dimension location as (location_key, street, city, province_or_state, country)
```

The define cube statement defines a data cube called *sales_star*, which corresponds to the central *sales* fact table of Example 2.1. This command specifies the keys to the dimension tables, and the two measures, *dollars_sold* and *units_sold*. The data cube has four dimensions, namely *time, item, branch*, and *location*. A define dimension statement is used to define each of the dimensions. □

Example 2.5 The snowflake schema of Example 2.2 and Figure 2.5 is defined in DMQL as follows.

```
define cube sales_snowflake [time, item, branch, location]:
                        dollars_sold = sum(sales_in_dollars), units_sold = count(*)
define dimension time as (time_key, day, day_of_week, month, quarter, year)
define dimension item as (item_key, item_name, brand, type, supplier (supplier_key, supplier_type))
define dimension branch as (branch_key, branch_name, branch_type)
define dimension location as (location_key, street, city (city_key, city, province_or_state, country))
```

This definition is similar to that of *sales_star* (Example 2.4), except that, here, the *item* and *location* dimensions tables are normalized. For instance, the *item* dimension of the *sales_star* data cube has been normalized in the *sales_snowflake* cube into two dimension tables, *item* and *supplier*. Note that the dimension definition for *supplier* is specified within the definition for *item*. Defining *supplier* in this way implicitly creates a *supplier_key* in the *item* dimension table definition. Similarly, the *location* dimension of the *sales_star* data cube has been normalized in the *sales_snowflake* cube into two dimension tables, *location* and *city*. The dimension definition for *city* is specified within the definition for *location*. In this way, a *city_key* is implicitly created in the *location* dimension table definition. □

Finally, a fact constellation schema can be defined as a set of interconnected cubes. Below is an example.

Example 2.6 The fact constellation schema of Example 2.3 and Figure 2.6 is defined in DMQL as follows.

```
define cube sales [time, item, branch, location]:
                        dollars_sold = sum(sales_in_dollars), units_sold = count(*)
define dimension time as (time_key, day, day_of_week, month, quarter, year)
define dimension item as (item_key, item_name, brand, type)
define dimension branch as (branch_key, branch_name, branch_type)
define dimension location as (location_key, street, city, province_or_state, country)
define cube shipping [time, item, shipper, from_location, to_location]:
                        dollars_cost = sum(cost_in_dollars), units_shipped = count(*)
define dimension time as time in cube sales
define dimension item as item in cube sales
define dimension shipper as (shipper_key, shipper_name, location as location in cube sales, shipper_type)
define dimension from_location as location in cube sales
define dimension to_location as location in cube sales
```

A define cube statement is used to define data cubes for *sales* and *shipping*, corresponding to the two fact tables of the schema of Example 2.3. Note that the *time, item*, and *location* dimensions of the *sales* cube are shared with the *shipping* cube. This is indicated for the *time* dimension, for example, as follows. Under the define cube statement for *shipping*, the statement "define dimension *time* as *time* in cube *sales*" is specified. □

Instead of having users or experts explicitly define data cube dimensions, dimensions can be automatically generated or adjusted based on the examination of data distributions. DMQL primitives for specifying such automatic generation or adjustments are discussed in the following chapter.

2.2.4 Measures: their categorization and computation

"How are measures computed?"

To answer this question, we will first look at how measures can be categorized. Note that multidimensional points in the data cube space are defined by dimension-value pairs. For example, the dimension-value pairs in ⟨time="Q1", location="Vancouver", item="computer"⟩ define a point in data cube space. A data cube **measure** is a numerical 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, 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 is partitioned into n sets. The computation of the function on each partition derives one aggregate value. If the result derived by applying the function to the n aggregate values is the same as that derived by applying the function on all the data without partitioning, the function can be computed in a distributed manner. For example, `count()` can be computed for a data cube by first partitioning the cube into a set of subcubes, computing `count()` for each subcube, and then summing up the counts obtained for each subcube. Hence `count()` is a distributive aggregate function. For the same reason, `sum()`, `min()`, and `max()` are distributive aggregate functions. A measure is *distributive* if it is obtained by applying a distributive aggregate function.

- **algebraic**: An aggregate function is *algebraic* if it can be computed by an algebraic function with M arguments (where M is a bounded 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()`, `max_N()`, 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()` (i.e., the most frequently occurring item(s)), 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 can be difficult to compute holistic measures efficiently. Efficient techniques to *approximate* the computation of some holistic measures, however, do exist. For example, instead of computing the exact `median()`, there are techniques which can estimate the approximate median value for a large data set with satisfactory results. In many cases, such techniques are sufficient to overcome the difficulties of efficient computation of holistic measures.

Example 2.7 Many measures of a data cube can be computed by relational aggregation operations. In Figure 2.4, we saw a star schema for *AllElectronics* sales which contains two measures, namely *dollars_sold* and *units_sold*. In Example 2.4, the *sales_star* data cube corresponding to the schema was defined using DMQL commands. *"But, how are these commands interpreted in order to generate the specified data cube?"*

Suppose that the relational database schema of *AllElectronics* is the following:

time(time_key, day, day_of_week, month, quarter, year)
item(item_key, item_name, brand, type)
branch(branch_key, branch_name, branch_type)
location(location_key, street, city, province_or_state, country)
sales(time_key, item_key, branch_key, location_key, number_of_units_sold, price)

The DMQL specification of Example 2.4 is translated into the following SQL query, which generates the required *sales_star* cube. Here, the `sum` aggregate function is used to compute both *dollars_sold* and *units_sold*.

```
select s.time_key, s.item_key, s.branch_key, s.location_key,
                        sum(s.number_of_units_sold * s.price), sum(s.number_of_units_sold)
from time t, item i, branch b, location l, sales s,
where s.time_key = t.time_key and s.item_key = i.item_key
          and s.branch_key = b.branch_key and s.location_key = l.location_key
group by s.time_key, s.item_key, s.branch_key, s.location_key
```

The cube created in the above query is the base cuboid of the *sales_star* data cube. It contains all of the dimensions specified in the data cube definition, where the granularity of each dimension is at the **join key** level. A join key is a key that links a fact table and a dimension table. The fact table associated with a base cuboid is sometimes referred to as the **base fact table**.

By changing the `group by` clauses, we may generate other cuboids for the *sales_star* data cube. For example, instead of grouping by *s.time_key*, we can group by *t.month*, which will sum up the measures of each group by month. Also, removing "`group by` *s.branch_key*" will generate a higher level cuboid (where sales are summed for all branches, rather than broken down per branch). Suppose we modify the above SQL query by removing *all* of the `group by` clauses. This will result in obtaining the total sum of *dollars_sold* and the total count of *units_sold* for the given data. This zero-dimensional cuboid is the apex cuboid of the *sales_star* data cube. In addition, other cuboids can be generated by applying selection and/or projection operations on the base cuboid, resulting in a lattice of cuboids as described in Section 2.2.1. Each cuboid corresponds to a different degree of summarization of the given data. □

Most of the current data cube technology confines the measures of multidimensional databases to *numerical data*. However, measures can also be applied to other kinds of data, such as spatial, multimedia, or text data. Techniques for this are discussed in Chapter 9.

2.2.5 Introducing concept hierarchies

"What is a concept hierarchy?"

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, Montreal, 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 to which they belong, such as Canada or the USA. 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). The concept hierarchy described above is illustrated in Figure 2.7.

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, zipcode*, 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 2.8a). Alternatively, the attributes of a dimension may 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 2.8b). 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 may be predefined in the data mining system, such as the the concept hierarchy for *time*. Data mining systems should provide users with the flexibility to tailor predefined hierarchies according to their particular needs. For example, one may like 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 2.9 for the dimension *price*.

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*.

[1] Since a *week* usually 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.

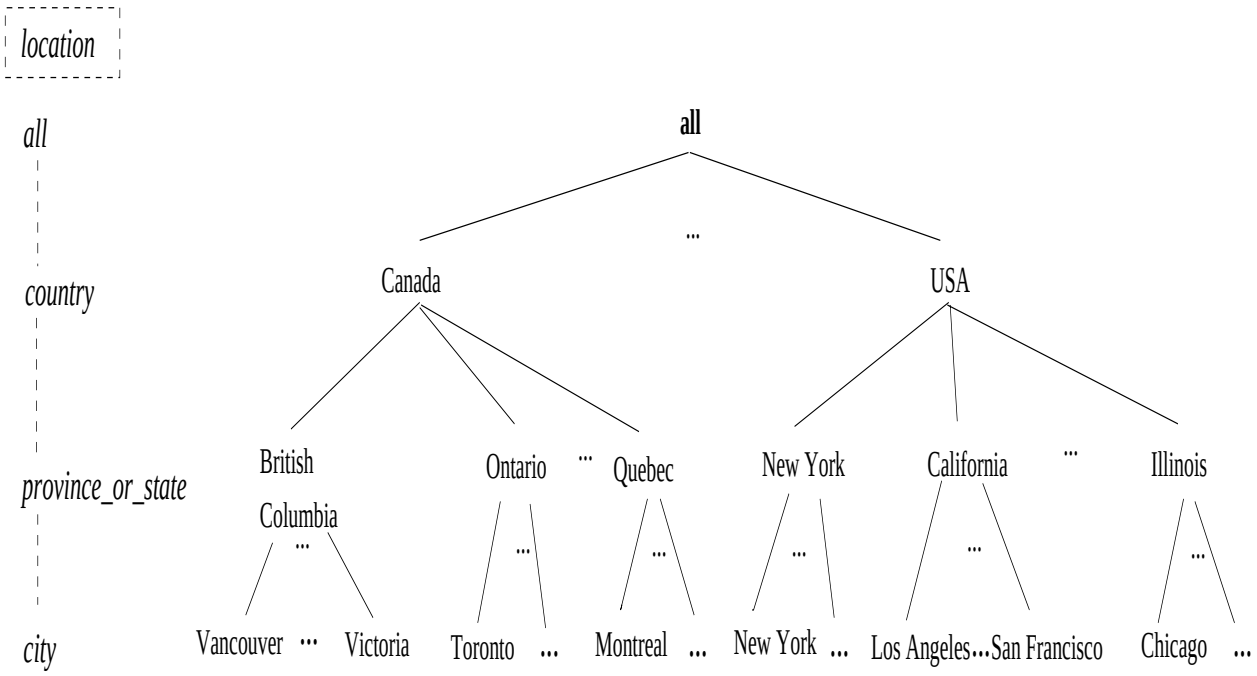


Figure 2.7: A concept hierarchy for the dimension *location*.

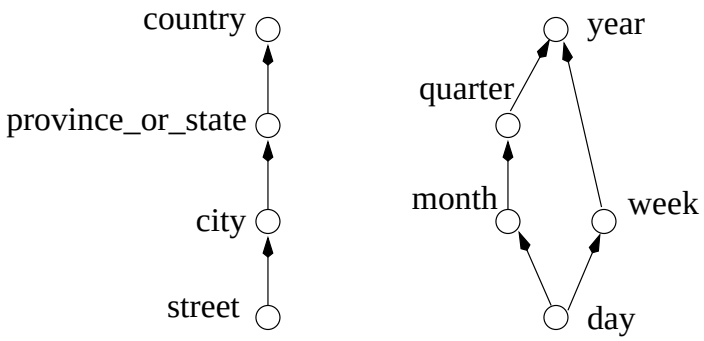
Concept hierarchies may be provided manually by system users, domain experts, knowledge engineers, or automatically generated based on statistical analysis of the data distribution. The automatic generation of concept hierarchies is discussed in Chapter 3. Concept hierarchies are further discussed in Chapter 4.

Concept hierarchies allow data to be handled at varying levels of abstraction, as we shall see in the following subsection.

2.2.6 OLAP operations in the multidimensional data model

"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



a) a hierarchy for *location* b) a lattice for *time*

Figure 2.8: Hierarchical and lattice structures of attributes in warehouse dimensions.

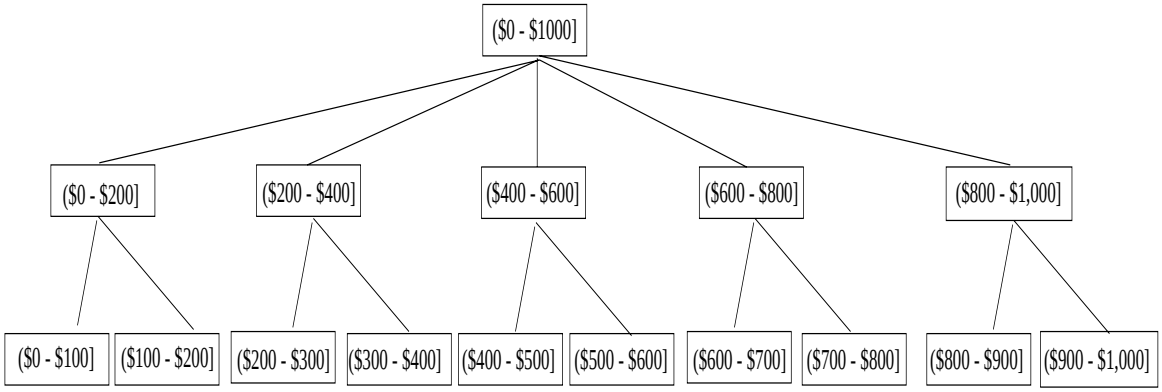


Figure 2.9: A concept hierarchy for the attribute *price*.

for interactive data analysis.

Example 2.8 Let's have a look at some typical OLAP operations for multidimensional data. Each of the operations described below is illustrated in Figure 2.10. 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 data examined are for the cities Vancouver, Montreal, New York, and Chicago.

1. **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 2.10 shows the result of a roll-up operation performed on the central cube by climbing up the concept hierarchy for *location* given in Figure 2.7. 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 two dimensions *location* and *time*. 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.

2. **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 2.10 shows the result of a drill-down operation performed on the central cube by stepping down a 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 summarized by quarter.

   Since 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 2.10 can occur by introducing an additional dimension, such as *customer_type*.

3. **slice and dice**: The *slice* operation performs a selection on one dimension of the given cube, resulting in a subcube. Figure 2.10 shows a slice operation where the sales data are selected from the central cube for the dimension *time* using the criteria *time="Q2"*. The *dice* operation defines a subcube by performing a selection on two or more dimensions. Figure 2.10 shows a dice operation on the central cube based on the following selection criteria which involves three dimensions: (*location="Montreal"* or *"Vancouver"*) and (*time="Q1"* or *"Q2"*) and (*item="home entertainment"* or *"computer"*).

4. **pivot (rotate)**: *Pivot* (also called "*rotate*") is a visualization operation which rotates the data axes in view in order to provide an alternative presentation of the data. Figure 2.10 shows a pivot operation where the

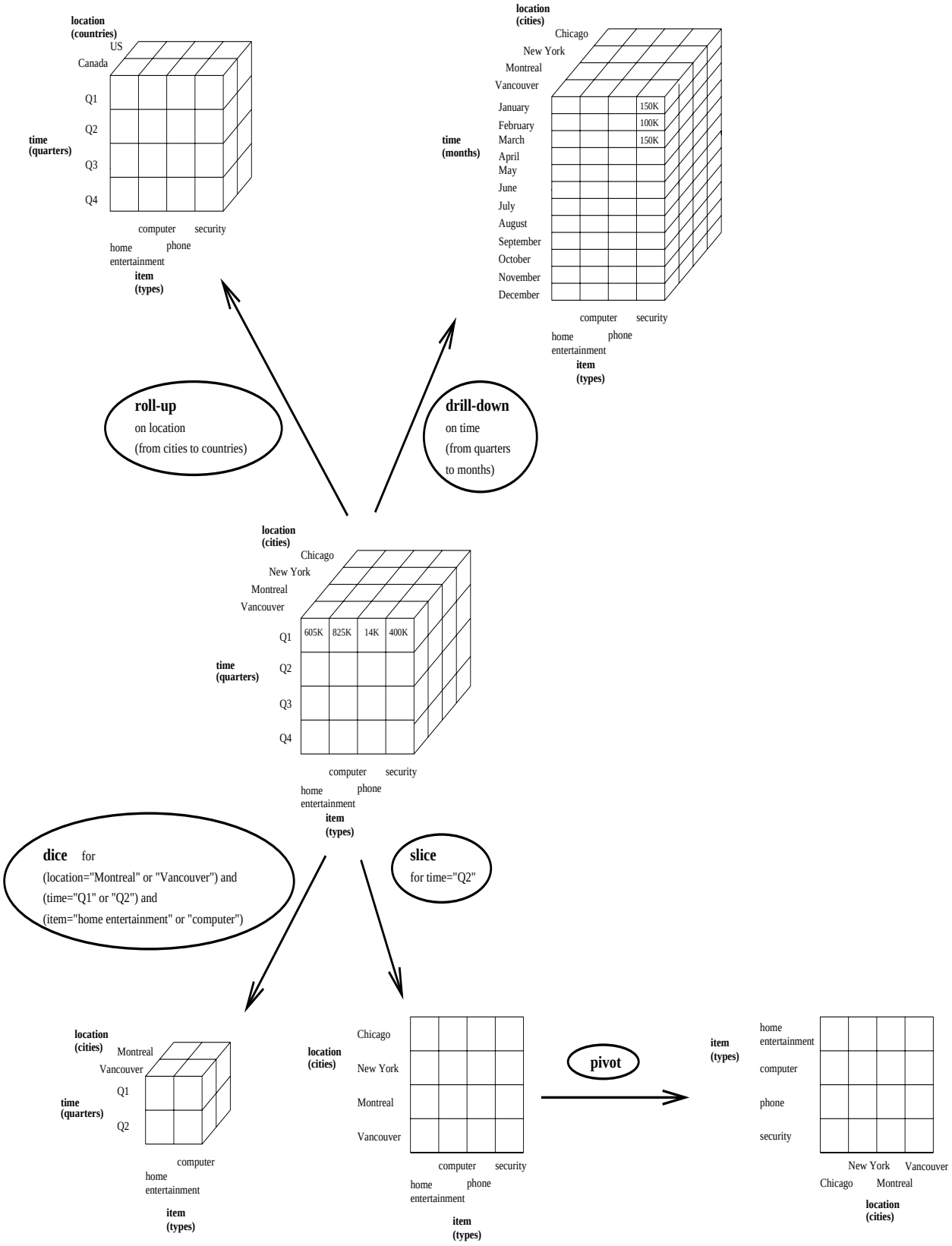


Figure 2.10: Examples of typical OLAP operations on multidimensional data.

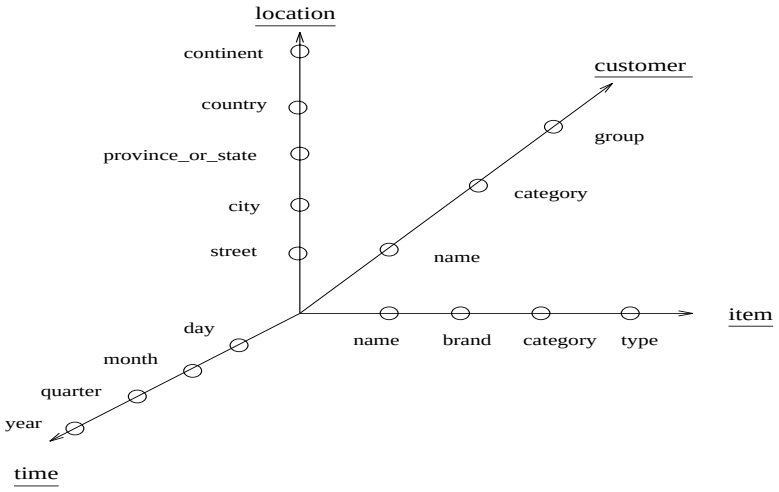


Figure 2.11: Modeling business queries: A starnet model.

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.

5. **other OLAP operations**: Some OLAP systems offer additional drilling operations. For example, **drill-across** executes queries involving (i.e., acrosss) more than one fact table. The **drill-through** operation makes use of 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 rates of return, depreciation, currency conversions, and statistical functions.

OLAP offers analytical modeling capabilities, including a calculation engine for deriving ratios, variance, etc., 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.

2.2.7 A starnet query model for querying multidimensional databases

The querying of multidimensional databases can be based on a **starnet model**. A starnet model consists of radial lines emanating from a central point, where each line represents a concept hierarchy for a dimension. Each abstraction level in the hierarchy is called a **footprint**. These represent the granularities available for use by OLAP operations such as drill-down and roll-up.

Example 2.9 A starnet query model for the *AllElectronics* data warehouse is shown in Figure 2.11. This starnet consists of four radial lines, representing concept hierarchies for the dimensions *location, customer, item*, and *time*, respectively. Each line consists of footprints representing abstraction levels of the dimension. For example, the *time* line has four footprints: "day", "month", "quarter" and "year". A concept hierarchy may involve a single attribute (like *date* for the *time* hierarchy), or several attributes (e.g., the concept hierarchy for *location* involves the attributes *street, city, province_or_state*, and *country*). In order to examine the item sales at *AllElectronics*, one can roll up along the *time* dimension from *month* to *quarter*, or, say, drill down along the *location* dimension from *country* to *city*. Concept hierarchies can be used to **generalize** data by replacing low-level values (such as "day" for the *time* dimension) by higher-level abstractions (such as "year"), or to **specialize** data by replacing higher-level abstractions with lower-level values. □

2.3 Data warehouse architecture

2.3.1 Steps for the design and construction of data warehouses

The design of a data warehouse: A business analysis framework

"What does the data warehouse provide for business analysts?"

First, having a data warehouse may provide a *competitive advantage* by presenting relevant information from which to measure performance and make critical adjustments in order to help win over competitors. Second, a data warehouse can enhance business *productivity* since it is able to quickly and efficiently gather information which accurately describes the organization. Third, a data warehouse facilitates *customer relationship marketing* since it provides a consistent view of customers and items across all lines of business, all departments, and all markets. Finally, a data warehouse may bring about *cost reduction* by tracking trends, patterns, and exceptions over long periods of time in a consistent and reliable manner.

To design an effective data warehouse one needs to understand and analyze business needs, and construct a *business analysis framework*. The construction of a large and complex information system can be viewed as the construction of a large and complex building, for which the owner, architect, and builder have different views. These views are combined to form a complex framework which represents the top-down, business-driven, or owner's perspective, as well as the bottom-up, builder-driven, or implementor's view of the information system.

Four different views regarding the design of a data warehouse must be considered: the *top-down view*, the *data source view*, the *data warehouse view*, and the *business query view*.

- The **top-down view** allows the selection of the relevant information necessary for the data warehouse. This information matches the current and coming business needs.
- The **data source view** exposes the information being captured, stored, and managed by operational systems. This information may be documented at various levels of detail and accuracy, from individual data source tables to integrated data source tables. Data sources are often modeled by traditional data modeling techniques, such as the entity-relationship model or CASE (Computer Aided Software Engineering) tools.
- The **data warehouse view** includes fact tables and dimension tables. It represents the information that is stored inside the data warehouse, including precalculated totals and counts, as well as information regarding the source, date, and time of origin, added to provide historical context.
- Finally, the **business query view** is the perspective of data in the data warehouse from the view point of the end-user.

Building and using a data warehouse is a complex task since it requires *business skills, technology skills*, and *program management skills*. Regarding *business skills*, building a data warehouse involves understanding how such systems store and manage their data, how to build **extractors** which transfer data from the operational system to the data warehouse, and how to build **warehouse refresh software** that keeps the data warehouse reasonably up to date with the operational system's data. Using a data warehouse involves understanding the significance of the data it contains, as well as understanding and translating the business requirements into queries that can be satisfied by the data warehouse. Regarding *technology skills*, data analysts are required to understand how to make assessments from quantitative information and derive facts based on conclusions from historical information in the data warehouse. These skills include the ability to discover patterns and trends, to extrapolate trends based on history and look for anomalies or paradigm shifts, and to present coherent managerial recommendations based on such analysis. Finally, *program management skills* involve the need to interface with many technologies, vendors and end-users in order to deliver results in a timely and cost-effective manner.

The process of data warehouse design

"How can I design a data warehouse?"

A data warehouse can be built using a *top-down approach*, a *bottom-up approach*, or a *combination of both*. The **top-down approach** starts with the overall design and planning. It is useful in cases where the technology is mature and well-known, and where the business problems that must be solved are clear and well-understood. The

bottom-up approach starts with experiments and prototypes. This is useful in the early stage of business modeling and technology development. It allows an organization to move forward at considerably less expense and to evaluate the benefits of the technology before making significant commitments. In the **combined approach**, an organization can exploit the planned and strategic nature of the top-down approach while retaining the rapid implementation and opportunistic application of the bottom-up approach.

From the software engineering point of view, the design and construction of a data warehouse may consist of the following steps: *planning, requirements study, problem analysis, warehouse design, data integration and testing*, and finally *deployment of the data warehouse*. Large software systems can be developed using two methodologies: the *waterfall method* or the *spiral method*. The **waterfall method** performs a structured and systematic analysis at each step before proceeding to the next, which is like a waterfall, falling from one step to the next. The **spiral method** involves the rapid generation of increasingly functional systems, with short intervals between successive releases. This is considered a good choice for data warehouse development, especially for data marts, because the turn-around time is short, modifications can be done quickly, and new designs and technologies can be adapted in a timely manner.

In general, the warehouse design process consists of the following steps.

1. Choose a *business process* to model, e.g., orders, invoices, shipments, inventory, account administration, sales, and the general ledger. If the business process is organizational and involves multiple, complex object collections, a data warehouse model should be followed. However, if the process is departmental and focuses on the analysis of one kind of business process, a data mart model should be chosen.

2. Choose the *grain* of the business process. The grain is the fundamental, atomic level of data to be represented in the fact table for this process, e.g., individual transactions, individual daily snapshots, etc.

3. Choose the *dimensions* that will apply to each fact table record. Typical dimensions are time, item, customer, supplier, warehouse, transaction type, and status.

4. Choose the *measures* that will populate each fact table record. Typical measures are numeric additive quantities like *dollars_sold* and *units_sold*.

Since data warehouse construction is a difficult and long term task, its implementation scope should be clearly defined. The goals of an initial data warehouse implementation should be *specific, achievable*, and *measurable*. This involves determining the time and budget allocations, the subset of the organization which is to be modeled, the number of data sources selected, and the number and types of departments to be served.

Once a data warehouse is designed and constructed, the initial deployment of the warehouse includes initial installation, rollout planning, training and orientation. Platform upgrades and maintenance must also be considered. Data warehouse administration will include data refreshment, data source synchronization, planning for disaster recovery, managing access control and security, managing data growth, managing database performance, and data warehouse enhancement and extension. Scope management will include controlling the number and range of queries, dimensions, and reports; limiting the size of the data warehouse; or limiting the schedule, budget, or resources.

Various kinds of data warehouse design tools are available. **Data warehouse development tools** provide functions to define and edit metadata repository contents such as schemas, scripts or rules, answer queries, output reports, and ship metadata to and from relational database system catalogues. **Planning and analysis tools** study the impact of schema changes and of refresh performance when changing refresh rates or time windows.

2.3.2 A three-tier data warehouse architecture

"What is data warehouse architecture like?"

Data warehouses often adopt a three-tier architecture, as presented in Figure 2.12. The bottom tier is a **warehouse database server** which is almost always a relational database system. The middle tier is an **OLAP server** which 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. The top tier is a **client**, which contains query and reporting tools, analysis tools, and/or data mining tools (e.g., trend analysis, prediction, and so on).

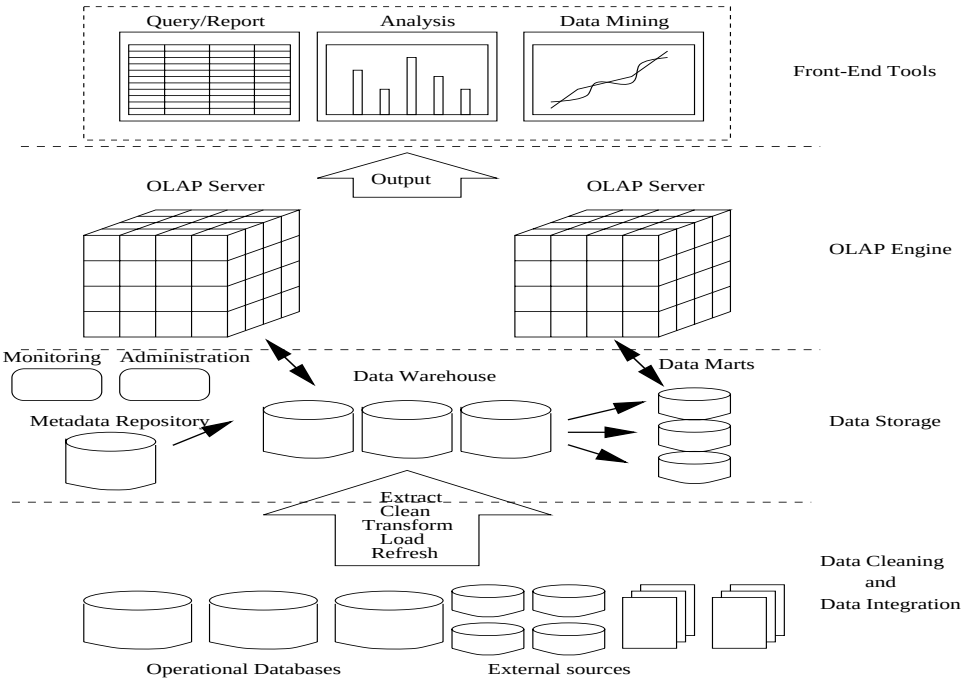


Figure 2.12: A three-tier data warehousing architecture.

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, UNIX 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-, Windows/NT-, or OS/2-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 into the following two classes:

  - *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.

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

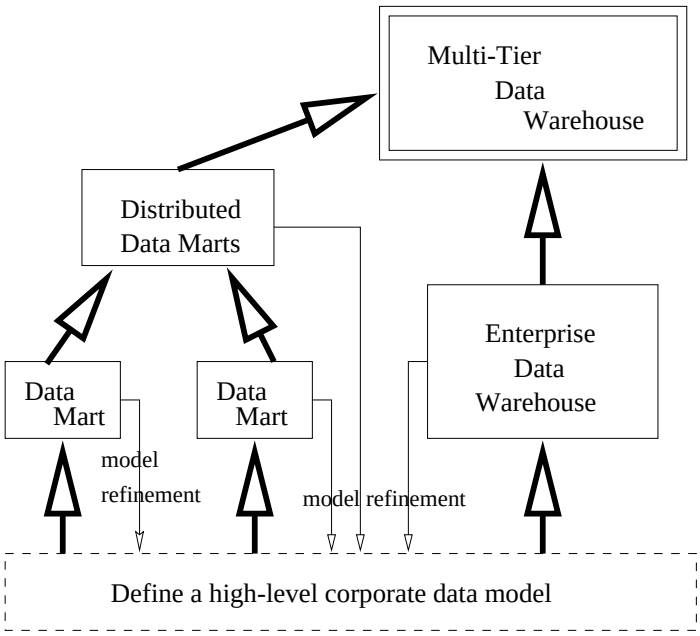


Figure 2.13: A recommended approach for data warehouse development.

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 2.13. First, a high-level corporate data model is defined within a reasonably short period of time (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 warehouse based on the same corporate data model set as above. Third, distributed data marts can be constructed to integrate different data marts via hub servers. Finally, a **multi-tier 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.

2.3.3 OLAP server architectures: ROLAP vs. MOLAP vs. HOLAP

"*What is OLAP server architecture like?*"

Logically, OLAP engines present business users with multidimensional data from data warehouses or data marts, without concerns regarding how or where the data are stored. However, the physical architecture and implementation of OLAP engines must consider data storage issues. Implementations of a warehouse server engine for OLAP processing include:

- **Relational OLAP (ROLAP) servers**: These are the intermediate servers that stand in between a relational back-end server and client front-end tools. They use a relational or extended-relational DBMS to store and manage warehouse data, and OLAP middleware to support missing pieces. ROLAP servers include optimization for each DBMS back-end, implementation of aggregation navigation logic, and additional tools and services. ROLAP technology tends to have greater scalability than MOLAP technology. The DSS server of Microstrategy and Metacube of Informix, for example, adopt the ROLAP approach[2].

- **Multidimensional OLAP (MOLAP) servers**: These servers support multidimensional views of data through array-based multidimensional storage engines. They map multidimensional views directly to data

[2] Information on these products can be found at www.informix.com and www.microstrategy.com, respectively.

cube array structures. For example, Essbase of Arbor is a MOLAP server. The advantage of using a data cube is that it allows fast indexing to precomputed summarized data. Notice that with multidimensional data stores, the storage utilization may be low if the data set is sparse. In such cases, sparse matrix compression techniques (see Section 2.4) should be explored.

Many OLAP servers adopt a two-level storage representation to handle sparse and dense data sets: the dense subcubes are identified and stored as array structures, while the sparse subcubes employ compression technology for efficient storage utilization.

- **Hybrid OLAP (HOLAP) servers:** The hybrid OLAP approach combines ROLAP and MOLAP technology, benefitting from the greater scalability of ROLAP and the faster computation of MOLAP. For example, a HOLAP server may allow large volumes of detail data to be stored in a relational database, while aggregations are kept in a separate MOLAP store. The Microsoft SQL Server 7.0 OLAP Services supports a hybrid OLAP server.

- **Specialized SQL servers**: To meet the growing demand of OLAP processing in relational databases, some relational and data warehousing firms (e.g., Redbrick) implement specialized SQL servers which provide advanced query language and query processing support for SQL queries over star and snowflake schemas in a read-only environment.

The OLAP functional architecture consists of three components: the *data store*, the *OLAP server*, and the *user presentation module*. The data store can be further classified as a *relational data store* or a *multidimensional data store*, depending on whether a ROLAP or MOLAP architecture is adopted.

"*So, how are data actually stored in ROLAP and MOLAP architectures?*"

As its name implies, ROLAP uses relational tables to store data for on-line analytical processing. Recall that the fact table associated with a base cuboid is referred to as a *base fact table*. The base fact table stores data at the abstraction level indicated by the join keys in the schema for the given data cube. Aggregated data can also be stored in fact tables, referred to as **summary fact tables**. Some summary fact tables store both base fact table data and aggregated data, as in Example 2.10. Alternatively, separate summary fact tables can be used for each level of abstraction, to store only aggregated data.

Example 2.10 Table 2.4 shows a summary fact table which contains both base fact data and aggregated data. The schema of the table is "⟨*record_identifier (RID), item, location, day, month, quarter, year, dollars_sold* (i.e., sales amount)⟩", where *day, month, quarter*, and *year* define the date of sales. Consider the tuple with an RID of 1001. The data of this tuple are at the base fact level. Here, the date of sales is October 15, 1997. Consider the tuple with an RID of 5001. This tuple is at a more general level of abstraction than the tuple having an RID of 1001. Here, the "*Main Street*" value for *location* has been generalized to "*Vancouver*". The *day* value has been generalized to all, so that the corresponding *time* value is October 1997. That is, the *dollars_sold* amount shown is an aggregation representing the entire month of October 1997, rather than just October 15, 1997. The special value all is used to represent subtotals in summarized data.

RID	item	location	day	month	quarter	year	dollars_sold
1001	TV	Main Street	15	10	Q4	1997	250.60
...	...	...	...	...	...	...	...
5001	TV	Vancouver	all	10	Q4	1997	45,786.08
...	...	...	...	...	...	...	...

Table 2.4: Single table for base and summary facts.

□

MOLAP uses multidimensional array structures to store data for on-line analytical processing. For example, the data cube structure described and referred to throughout this chapter is such an array structure.

Most data warehouse systems adopt a client-server architecture. A relational data store always resides at the data warehouse/data mart server site. A multidimensional data store can reside at either the database server site or the client site. There are several alternative physical configuration options.

If a multidimensional data store resides at the client side, it results in a "fat client". In this case, the system response time may be quick since OLAP operations will not consume network traffic, and the network bottleneck happens only at the warehouse loading stage. However, loading a large data warehouse can be slow and the processing at the client side can be heavy, which may degrade the system performance. Moreover, data security could be a problem because data are distributed to multiple clients. A variation of this option is to partition the multidimensional data store and distribute selected subsets of the data to different clients.

Alternatively, a multidimensional data store can reside at the server site. One option is to set both the multidimensional data store and the OLAP server at the data mart site. This configuration is typical for data marts that are created by refining or re-engineering the data from an enterprise data warehouse.

A variation is to separate the multidimensional data store and OLAP server. That is, an OLAP server layer is added between the client and data mart. This configuration is used when the multidimensional data store is large, data sharing is needed, and the client is "thin" (i.e., does not require many resources).

2.3.4 SQL extensions to support OLAP operations

"How can SQL be extended to support OLAP operations?"

An OLAP server should support several data types including text, calendar, and numeric data, as well as data at different granularities (such as regarding the estimated and actual sales per item). An OLAP server should contain a calculation engine which includes domain-specific computations (such as for calendars) and a rich library of aggregate functions. Moreover, an OLAP server should include data load and refresh facilities so that *write* operations can update precomputed aggregates, and *write/load* operations are accompanied by data cleaning.

A multidimensional view of data is the foundation of OLAP. SQL extensions to support OLAP operations have been proposed and implemented in extended-relational servers. Some of these are enumerated as follows.

1. Extending the family of aggregate functions.

   Relational database systems have provided several useful aggregate functions, including `sum()`, `avg()`, `count()`, `min()`, and `max()` as SQL standards. OLAP query answering requires the extension of these standards to include other aggregate functions such as `rank()`, `N_tile()`, `median()`, and `mode()`. For example, a user may like to list the *top five most profitable items* (using `rank()`), list the *firms whose performance is in the bottom 10% in comparison to all other firms* (using `N_tile()`), or print *the most frequently sold items in March* (using `mode()`).

2. Adding reporting features.

   Many report writer softwares allow aggregate features to be evaluated on a time window. Examples include running totals, cumulative totals, moving averages, break points, etc. OLAP systems, to be truly useful for decision support, should introduce such facilities as well.

3. Implementing multiple group-by's.

   Given the multidimensional view point of data warehouses, it is important to introduce *group-by*'s for grouping sets of attributes. For example, one may want to list the total sales from 1996 to 1997 grouped by item, by region, and by quarter. Although this can be simulated by a set of SQL statements, it requires multiple scans of databases, and is thus a very inefficient solution. New operations, including `cube` and `roll-up`, have been introduced in some relational system products which explore efficient implementation methods.

2.4 Data warehouse implementation

Data warehouses contain huge volumes of data. OLAP engines demand that decision support queries be answered in the order of seconds. Therefore, it is crucial for data warehouse systems to support highly efficient cube computation techniques, access methods, and query processing techniques. *"How can this be done?"*, you may wonder. In this section, we examine methods for the efficient implementation of data warehouse systems.

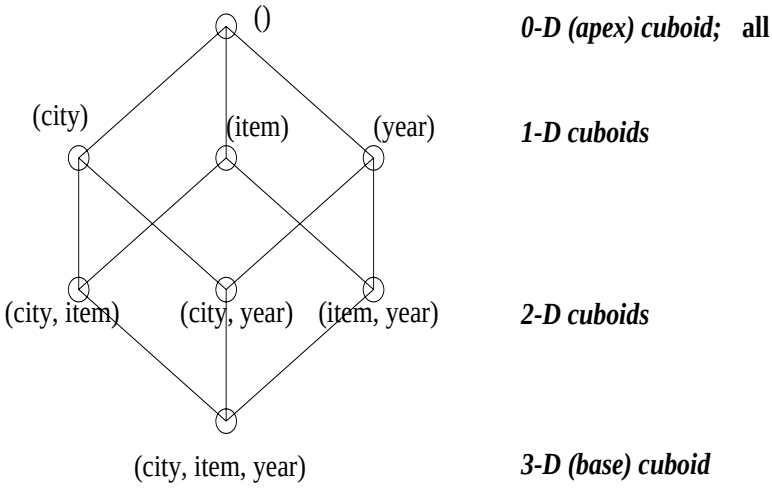


Figure 2.14: Lattice of cuboids, making up a 3-dimensional data cube. Each cuboid represents a different group-by. The base cuboid contains the three dimensions, *city, item*, and *year*.

2.4.1 Efficient computation of data cubes

At the core of multidimensional data analysis is the efficient computation of aggregations across many sets of dimensions. In SQL terms, these aggregations are referred to as **group-by's**.

The compute cube operator and its implementation

One approach to cube computation extends SQL so as to include a compute cube operator. The compute cube operator computes aggregates over all subsets of the dimensions specified in the operation.

Example 2.11 Suppose that you would like to create a data cube for *AllElectronics* sales which contains the following: *item, city, year*, and *sales_in_dollars*. You would like to be able to analyze the data, with queries such as the following:

1. "*Compute the sum of sales, grouping by item and city.*"
2. "*Compute the sum of sales, grouping by item.*"
3. "*Compute the sum of sales, grouping by city*".

What is the total number of cuboids, or group-by's, that can be computed for this data cube? Taking the three attributes, *city, item*, and *year*, as three dimensions and *sales_in_dollars* as the measure, the total number of cuboids, or group-by's, that can be computed for this data cube is $2^3 = 8$. The possible group-by's are the following: $\{(city, item, year), (city, item), (city, year), (item, year), (city), (item), (year), ()\}$, where () means that the group-by is empty (i.e., the dimensions are not grouped). These group-by's form a lattice of cuboids for the data cube, as shown in Figure 2.14. The base cuboid contains all three dimensions, *city, item*, and *year*. It can return the total sales for any combination of the three dimensions. The apex cuboid, or 0-D cuboid, refers to the case where the group-by is empty. It contains the total sum of all sales. Consequently, it is represented by the special value all. □

An SQL query containing no group-by, such as "compute the sum of total sales" is a *zero-dimensional operation*. An SQL query containing one group-by, such as "compute the sum of sales, group by city" is a *one-dimensional operation*. A cube operator on n dimensions is equivalent to a collection of group by statements, one for each subset of the n dimensions. Therefore, the cube operator is the n-dimensional generalization of the group by operator.

Based on the syntax of DMQL introduced in Section 2.2.3, the data cube in Example 2.11, can be defined as

```
define cube sales [item, city, year]: sum(sales_in_dollars)
```

For a cube with n dimensions, there are a total of 2^n cuboids, including the base cuboid. The statement

compute cube sales

explicitly instructs the system to compute the sales aggregate cuboids for all of the eight subsets of the set {*item, city, year*}, including the empty subset. A cube computation operator was first proposed and studied by Gray, et al. (1996).

On-line analytical processing may need to access different cuboids for different queries. Therefore, it does seem like a good idea to compute all or at least some of the cuboids in a data cube in advance. Precomputation leads to fast response time and avoids some redundant computation. Actually, most, if not all, OLAP products resort to some degree of precomputation of multidimensional aggregates.

A major challenge related to this precomputation, however, is that the required storage space may explode if all of the cuboids in a data cube are precomputed, especially when the cube has several dimensions associated with multiple level hierarchies.

"*How many cuboids are there in an n-dimensional data cube?*" If there were no hierarchies associated with each dimension, then the total number of cuboids for an n-dimensional data cube, as we have seen above, is 2^n. However, in practice, many dimensions do have hierarchies. For example, the dimension *time* is usually not just one level, such as *year*, but rather a hierarchy or a lattice, such as $day < week < month < quarter < year$. For an n-dimensional data cube, the total number of cuboids that can be generated (including the cuboids generated by climbing up the hierarchies along each dimension) is:

$$T = \prod_{i=1}^{n} (L_i + 1),$$

where L_i is the number of levels associated with dimension i (excluding the *virtual* top level all since generalizing to all is equivalent to the removal of a dimension). This formula is based on the fact that at most one abstraction level in each dimension will appear in a cuboid. For example, if the cube has 10 dimensions and each dimension has 4 levels, the total number of cuboids that can be generated will be $5^{10} \approx 9.8 \times 10^6$.

By now, you probably realize that it is unrealistic to precompute and materialize all of the cuboids that can possibly be generated for a data cube (or, from a base cuboid). If there are many cuboids, and these cuboids are large in size, a more reasonable option is *partial materialization*, that is, to materialize only *some* of the possible cuboids that can be generated.

Partial materialization: Selected computation of cuboids

There are three choices for data cube materialization: (1) precompute only the base cuboid and none of the remaining "non-base" cuboids (**no materialization**), (2) precompute all of the cuboids (**full materialization**), and (3) selectively compute a proper subset of the whole set of possible cuboids (**partial materialization**). The first choice leads to computing expensive multidimensional aggregates on the fly, which could be slow. The second choice may require huge amounts of memory space in order to store all of the precomputed cuboids. The third choice presents an interesting trade-off between storage space and response time.

The partial materialization of cuboids should consider three factors: (1) identify the subset of cuboids to materialize, (2) exploit the materialized cuboids during query processing, and (3) efficiently update the materialized cuboids during load and refresh.

The selection of the subset of cuboids to materialize should take into account the queries in the workload, their frequencies, and their accessing costs. In addition, it should consider workload characteristics, the cost for incremental updates, and the total storage requirements. The selection must also consider the broad context of physical database design, such as the generation and selection of indices. Several OLAP products have adopted heuristic approaches for cuboid selection. A popular approach is to materialize the set of cuboids having relatively simple structure. Even with this restriction, there are often still a large number of possible choices. Under a simplified assumption, a greedy algorithm has been proposed and has shown good performance.

Once the selected cuboids have been materialized, it is important to take advantage of them during query processing. This involves determining the relevant cuboid(s) from among the candidate materialized cuboids, how to use available index structures on the materialized cuboids, and how to transform the OLAP operations on to the selected cuboid(s). These issues are discussed in Section 2.4.3 on query processing.

Finally, during load and refresh, the materialized cuboids should be updated efficiently. Parallelism and incremental update techniques for this should be explored.

Multiway array aggregation in the computation of data cubes

In order to ensure fast on-line analytical processing, however, we may need to precompute all of the cuboids for a given data cube. Cuboids may be stored on secondary storage, and accessed when necessary. Hence, it is important to explore efficient methods for computing all of the cuboids making up a data cube, that is, for full materialization. These methods must take into consideration the limited amount of main memory available for cuboid computation, as well as the time required for such computation. To simplify matters, we may exclude the cuboids generated by climbing up existing hierarchies along each dimension.

Since Relational OLAP (ROLAP) uses tuples and relational tables as its basic data structures, while the basic data structure used in multidimensional OLAP (MOLAP) is the multidimensional array, one would expect that ROLAP and MOLAP each explore very different cube computation techniques.

ROLAP cube computation uses the following major optimization techniques.

1. Sorting, hashing, and grouping operations are applied to the dimension attributes in order to reorder and cluster related tuples.
2. Grouping is performed on some subaggregates as a "partial grouping step". These "partial groupings" may be used to speed up the computation of other subaggregates.
3. Aggregates may be computed from previously computed aggregates, rather than from the base fact tables.

"How do these optimization techniques apply to MOLAP?" ROLAP uses value-based addressing, where dimension values are accessed by key-based addressing search strategies. In contrast, MOLAP uses direct array addressing, where dimension values are accessed via the position or index of their corresponding array locations. Hence, MOLAP cannot perform the value-based reordering of the first optimization technique listed above for ROLAP. Therefore, a different approach should be developed for the array-based cube construction of MOLAP, such as the following.

1. Partition the array into chunks. A **chunk** is a subcube that is small enough to fit into the memory available for cube computation. **Chunking** is a method for dividing an n-dimensional array into small n-dimensional chunks, where each chunk is stored as an object on disk. The chunks are compressed so as to remove wasted space resulting from empty array cells (i.e., cells that do not contain any valid data). For instance, "*chunkID* + *offset*" can be used as a cell addressing mechanism to **compress a sparse array structure** and when searching for cells within a chunk. Such a compression technique is powerful enough to handle sparse cubes, both on disk and in memory.
2. Compute aggregates by visiting (i.e., accessing the values at) cube cells. The order in which cells are visited can be optimized so as to *minimize the number of times that each cell must be revisited*, thereby reducing memory access and storage costs. The trick is to exploit this ordering so that partial aggregates can be computed simultaneously, and any unnecessary revisiting of cells is avoided.

   Since this chunking technique involves "overlapping" some of the aggregation computations, it is referred to as **multiway array aggregation** in data cube computation.

We explain this approach to MOLAP cube construction by looking at a concrete example.

Example 2.12 Consider a 3-D data array containing the three dimensions, A, B, and C.

- The 3-D array is partitioned into small, memory-based chunks. In this example, the array is partitioned into 64 chunks as shown in Figure 2.15. Dimension A is organized into 4 partitions, a_0, a_1, a_2, and a_3. Dimensions B and C are similarly organized into 4 partitions each. Chunks 1, 2, ..., 64 correspond to the subcubes $a_0b_0c_0$, $a_1b_0c_0$, ..., $a_3b_3c_3$, respectively. Suppose the size of the array for each dimension, A, B, and C is 40, 400, 4000, respectively.
- Full materialization of the corresponding data cube involves the computation of all of the cuboids defining this cube. These cuboids consist of:

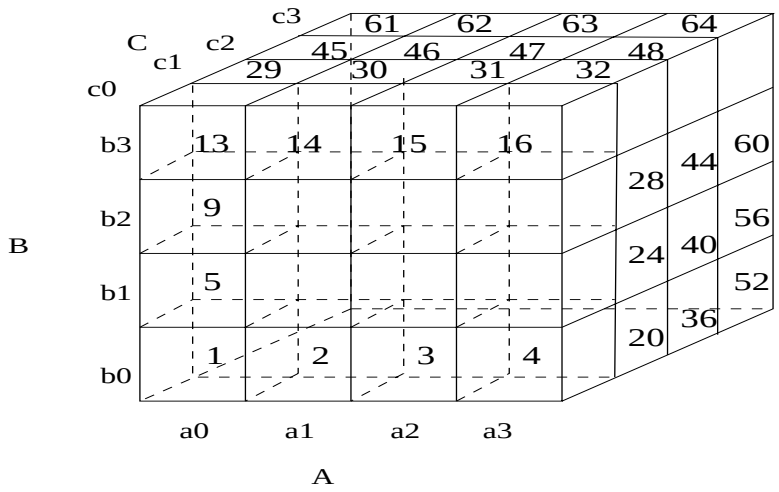


Figure 2.15: A 3-D array for the dimensions A, B, and C, organized into 64 *chunks*.

- The base cuboid, denoted by ABC (from which all of the other cuboids are directly or indirectly computed). This cube is already computed and corresponds to the given 3-D array.
- The 2-D cuboids, AB, AC, and BC, which respectively correspond to the group-by's AB, AC, and BC. These cuboids must be computed.
- The 1-D cuboids, A, B, and C, which respectively correspond to the group-by's A, B, and C. These cuboids must be computed.
- The 0-D (apex) cuboid, denoted by all, which corresponds to the group-by (), i.e., there is no group-by here. This cuboid must be computed.

Let's look at how the multiway array aggregation technique is used in this computation.

- There are many possible orderings with which chunks can be read into memory for use in cube computation. Consider the ordering labeled from 1 to 64, shown in Figure 2.15. Suppose we would like to compute the b_0c_0 chunk of the BC cuboid. We allocate space for this chunk in *"chunk memory"*. By scanning chunks 1 to 4 of ABC, the b_0c_0 chunk is computed. That is, the cells for b_0c_0 are aggregated over a_0 to a_3.

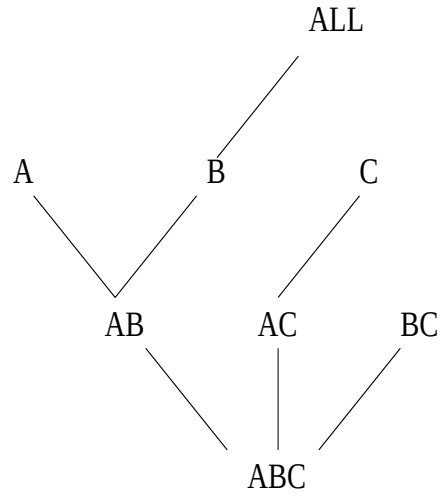
  The chunk memory can then be assigned to the next chunk, b_1c_0, which completes its aggregation after the scanning of the next 4 chunks of ABC: 5 to 8.

  Continuing in this way, the entire BC cuboid can be computed. Therefore, only *one* chunk of BC needs to be in memory, at a time, for the computation of all of the chunks of BC.

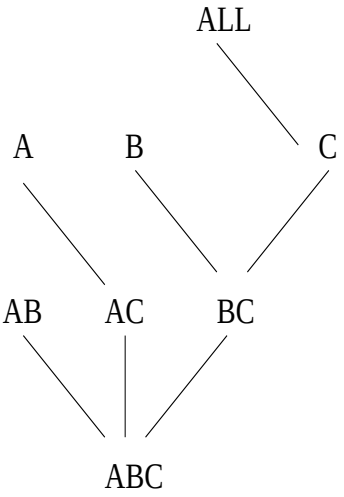
- In computing the BC cuboid, we will have scanned each of the 64 chunks. *"Is there a way to avoid having to rescan all of these chunks for the computation of other cuboids, such as AC and AB?"* The answer is, most definitely - *yes*. This is where the multiway computation idea comes in. For example, when chunk 1, i.e., $a_0b_0c_0$, is being scanned (say, for the computation of the 2-D chunk b_0c_0 of BC, as described above), all of the other 2-D chunks relating to $a_0b_0c_0$ can be simultaneously computed. That is, when $a_0b_0c_0$, is being scanned, each of the three chunks, b_0c_0, a_0c_0, and a_0b_0, on the three 2-D aggregation planes, BC, AC, and AB, should be computed then as well. In other words, multiway computation aggregates to each of the 3-D planes while a 3-D chunk is in memory.

Let's look at how different orderings of chunk scanning and of cuboid computation can affect the overall data cube computation efficiency. Recall that the size of the dimensions A, B, and C is 40, 400, and 4000, respectively. Therefore, the largest 2-D plane is BC (of size $400 \times 4,000 = 1,600,000$). The second largest 2-D plane is AC (of size $40 \times 4,000 = 160,000$). AB is the smallest 2-D plane (with a size of $40 \times 400 = 16,000$).

- Suppose that the chunks are scanned in the order shown, from chunk 1 to 64. By scanning in this order, one chunk of the largest 2-D plane, BC, is *fully* computed for each row scanned. That is, b_0c_0 is fully aggregated



a) Most efficient ordering of array aggregation (min. memory requirements = 156,000 memory units)



b) Least efficient ordering of array aggregation (min. memory requirements = 1,641,000 memory units)

Figure 2.16: Two orderings of multiway array aggregation for computation of the 3-D cube of Example 2.12.

after scanning the row containing chunks 1 to 4; b_1c_0 is fully aggregated after scanning chunks 5 to 8, and so on. In comparison, the complete computation of one chunk of the second largest 2-D plane, AC, requires scanning 13 chunks (given the ordering from 1 to 64). For example, a_0c_0 is fully aggregated after the scanning of chunks 1, 5, 9, and 13. Finally, the complete computation of one chunk of the smallest 2-D plane, AB, requires scanning 49 chunks. For example, a_0b_0 is fully aggregated after scanning chunks 1, 17, 33, and 49. Hence, AB requires the longest scan of chunks in order to complete its computation. To avoid bringing a 3-D chunk into memory more than once, the minimum memory requirement for holding all relevant 2-D planes in chunk memory, according to the chunk ordering of 1 to 64 is as follows: 40×400 (for the whole AB plane) + $40 \times 1,000$ (for one row of the AC plane) + $100 \times 1,000$ (for one chunk of the BC plane) = 16,000 + 40,000 + 100,000 = 156,000.

- Suppose, instead, that the chunks are scanned in the order 1, 17, 33, 49, 5, 21, 37, 53, etc. That is, suppose the scan is in the order of first aggregating towards the AB plane, and then towards the AC plane and lastly towards the BC plane. The minimum memory requirement for holding 2-D planes in chunk memory would be as follows: $400 \times 4,000$ (for the whole BC plane) + $40 \times 1,000$ (for one row of the AC plane) + 10×100 (for one chunk of the AB plane) = 1,600,000 + 40,000 + 1,000 = 1,641,000. Notice that this is *more than 10 times* the memory requirement of the scan ordering of 1 to 64.

- Similarly, one can work out the minimum memory requirements for the multiway computation of the 1-D and 0-D cuboids. Figure 2.16 shows a) the most efficient ordering and b) the least efficient ordering, based on the minimum memory requirements for the data cube computation. The most efficient ordering is the chunk ordering of 1 to 64.

- In conclusion, this example shows that the planes should be sorted and computed according to their size in ascending order. Since $|AB| < |AC| < |BC|$, the AB plane should be computed first, followed by the AC and BC planes. Similarly, for the 1-D planes, $|A| < |B| < |C|$ and therefore the A plane should be computed before the B plane, which should be computed before the C plane.

□

Example 2.12 assumes that there is enough memory space for *one-pass* cube computation (i.e., to compute all of the cuboids from one scan of all of the chunks). If there is insufficient memory space, the computation will require

more than one pass through the 3-D array. In such cases, however, the basic principle of ordered chunk computation remains the same.

"*Which is faster — ROLAP or MOLAP cube computation?*" With the use of appropriate sparse array compression techniques and careful ordering of the computation of cuboids, it has been shown that MOLAP cube computation is significantly faster than ROLAP (relational record-based) computation. Unlike ROLAP, the array structure of MOLAP does not require saving space to store search keys. Furthermore, MOLAP uses direct array addressing, which is faster than the key-based addressing search strategy of ROLAP. In fact, for ROLAP cube computation, instead of cubing a table directly, it is even faster to convert the table to an array, cube the array, and then convert the result back to a table.

2.4.2 Indexing OLAP data

To facilitate efficient data accessing, most data warehouse systems support index structures and materialized views (using cuboids). Methods to select cuboids for materialization were discussed in the previous section. In this section, we examine how to index OLAP data by *bitmap indexing* and *join indexing*.

The **bitmap indexing** method is popular in OLAP products because it allows quick searching in data cubes. The bitmap index is an alternative representation of the *record_ID (RID)* list. In the bitmap index for a given attribute, there is a distinct bit vector, Bv, for each value v in the domain of the attribute. If the domain of a given attribute consists of n values, then n bits are needed for each entry in the bitmap index (i.e., there are n bit vectors). If the attribute has the value v for a given row in the data table, then the bit representing that value is set to 1 in the corresponding row of the bitmap index. All other bits for that row are set to 0.

Bitmap indexing is especially advantageous for low cardinality domains because comparison, join, and aggregation operations are then reduced to bit-arithmetic, which substantially reduces the processing time. Bitmap indexing also leads to significant reductions in space and I/O since a string of characters can be represented by a single bit. For higher cardinality domains, the method can be adapted using compression techniques.

The **join indexing** method gained popularity from its use in relational database query processing. Traditional indexing maps the value in a given column to a list of rows having that value. In contrast, join indexing registers the joinable rows of two relations from a relational database. For example, if two relations $R(RID, A)$ and $S(B, SID)$ join on the attributes A and B, then the join index record contains the pair (RID, SID), where RID and SID are record identifiers from the R and S relations, respectively. Hence, the join index records can identify joinable tuples without performing costly join operations. Join indexing is especially useful for maintaining the relationship between a foreign key[3] and its matching primary keys, from the joinable relation.

The star schema model of data warehouses makes join indexing attractive for cross table search because the linkage between a fact table and its corresponding dimension tables are the foreign key of the fact table and the primary key of the dimension table. Join indexing maintains relationships between attribute values of a dimension (e.g., within a dimension table) and the corresponding rows in the fact table. Join indices may span multiple dimensions to form **composite join indices**. We can use join indexing to identify subcubes that are of interest.

To further speed up query processing, the join indexing and bitmap indexing methods can be integrated to form **bitmapped join indices**.

2.4.3 Efficient processing of OLAP queries

The purpose of materializing cuboids and constructing OLAP index structures is to speed up query processing in data cubes. Given materialized views, query processing should proceed as follows:

1. **Determine which operations should be performed on the available cuboids.** This involves transforming any selection, projection, roll-up (group-by) and drill-down operations specified in the query into corresponding SQL and/or OLAP operations. For example, slicing and dicing of a data cube may correspond to selection and/or projection operations on a materialized cuboid.

2. **Determine to which materialized cuboid(s) the relevant operations should be applied.** This involves identifying all of the materialized cuboids that may potentially be used to answer the query, pruning the

[3] A set of attributes in a relation schema that forms a primary key for another schema is called a **foreign key**.

above set using knowledge of "dominance" relationships among the cuboids, estimating the costs of using the remaining materialized cuboids, and selecting the cuboid with the least cost.

Example 2.13 Suppose that we define a data cube for *AllElectronics* of the form "*sales* [*time, item, location*]: `sum`*(sales_in_dollars)*". The dimension hierarchies used are "*day* < *month* < *quarter* < *year*" for *time*, "*item_name* < *brand* < *type*" for *item*, and "*street* < *city* < *province_or_state* < *country*" for *location*.

Suppose that the query to be processed is on {*brand, province_or_state*}, with the selection constant "*year* = *1997*". Also, suppose that there are four materialized cuboids available, as follows.

- cuboid 1: {*item_name, city, year*}
- cuboid 2: {*brand, country, year*}
- cuboid 3: {*brand, province_or_state, year*}
- cuboid 4: {*item_name, province_or_state*} where *year* = *1997*.

"Which of the above four cuboids should be selected to process the query?" Finer granularity data cannot be generated from coarser granularity data. Therefore, cuboid 2 cannot be used since *country* is a more general concept than *province_or_state*. Cuboids 1, 3 and 4 can be used to process the query since: (1) they have the same set or a superset of the dimensions in the query, and (2) the selection clause in the query can imply the selection in the cuboid, and (3) the abstraction levels for the *item* and *location* dimensions in these cuboids are at a finer level than *brand* and *province_or_state*, respectively.

"How would the costs of each cuboid compare if used to process the query?" It is likely that using cuboid 1 would cost the most since both *item_name* and *city* are at a lower level than the *brand* and *province_or_state* concepts specified in the query. If there are not many *year* values associated with *items* in the cube, but there are several *item_names* for each *brand*, then cuboid 3 will be smaller than cuboid 4, and thus cuboid 3 should be chosen to process the query. However, if efficient indices are available for cuboid 4, then cuboid 4 may be a better choice. Therefore, some cost-based estimation is required in order to decide which set of cuboids should be selected for query processing. □

Since the storage model of a MOLAP sever is an n-dimensional array, the front-end multidimensional queries are mapped directly to server storage structures, which provide direct addressing capabilities. The straightforward array representation of the data cube has good indexing properties, but has poor storage utilization when the data are sparse. For efficient storage and processing, sparse matrix and data compression techniques (Section 2.4.1) should therefore be applied.

The storage structures used by dense and sparse arrays may differ, making it advantageous to adopt a two-level approach to MOLAP query processing: use arrays structures for dense arrays, and sparse matrix structures for sparse arrays. The two-dimensional dense arrays can be indexed by B-trees.

To process a query in MOLAP, the dense one- and two- dimensional arrays must first be identified. Indices are then built to these arrays using traditional indexing structures. The two-level approach increases storage utilization without sacrificing direct addressing capabilities.

2.4.4 Metadata repository

"What are metadata?"

Metadata are data about data. When used in a data warehouse, metadata are the data that define warehouse objects. 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:

- a description of *the structure of the data warehouse*. This 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;
- *the 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; and
- *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 type. 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). Notice that the only type of summarization that is permanently stored in the data warehouse is that data which is frequently used.

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, as a guide to the mapping of data when the data are transformed from the operational environment to the data warehouse environment, and 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).

2.4.5 Data warehouse back-end tools and utilities

Data warehouse systems use back-end tools and utilities to populate and refresh their data. These tools and facilities include the following functions:

1. **data extraction**, which typically gathers data from multiple, heterogeneous, and external sources;
2. **data cleaning**, which detects errors in the data and rectifies them when possible;
3. **data transformation**, which converts data from legacy or host format to warehouse format;
4. **load**, which sorts, summarizes, consolidates, computes views, checks integrity, and builds indices and partitions; and
5. **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.

Since we are mostly interested in the aspects of data warehousing technology related to data mining, we will not get into the details of these tools and recommend interested readers to consult books dedicated to data warehousing technology.

2.5 Further development of data cube technology

In this section, you will study further developments in data cube technology. Section 2.5.1 describes data mining by *discovery-driven exploration of data cubes*, where anomalies in the data are automatically detected and marked for the user with visual cues. Section 2.5.2 describes *multifeature cubes* for complex data mining queries involving multiple dependent aggregates at multiple granularities.

2.5.1 Discovery-driven exploration of data cubes

As we have seen in this chapter, data can be summarized and stored in a multidimensional data cube of an OLAP system. A user or analyst can search for interesting patterns in the cube by specifying a number of OLAP operations, such as drill-down, roll-up, slice, and dice. While these tools are available to help the user explore the data, the discovery process is not automated. It is the user who, following her own intuition or hypotheses, tries to recognize exceptions or anomalies in the data. This **hypothesis-driven exploration** has a number of disadvantages. The search space can be very large, making manual inspection of the data a daunting and overwhelming task. High level aggregations may give no indication of anomalies at lower levels, making it easy to overlook interesting patterns. Even when looking at a subset of the cube, such as a slice, the user is typically faced with many data values to examine. The sheer volume of data values alone makes it easy for users to miss exceptions in the data if using hypothesis-driven exploration.

Discovery-driven exploration is an alternative approach in which precomputed measures indicating data exceptions are used to guide the user in the data analysis process, at all levels of aggregation. We hereafter refer to these measures as *exception indicators*. Intuitively, an **exception** is a data cube cell value that is significantly different from the value anticipated, based on a statistical model. The model considers variations and patterns in the measure value across *all of the dimensions* to which a cell belongs. For example, if the analysis of item-sales data reveals an increase in sales in December in comparison to all other months, this may seem like an exception in the time dimension. However, it is not an exception if the item dimension is considered, since there is a similar increase in sales for other items during December. The model considers exceptions hidden at all aggregated group-by's of a data cube. Visual cues such as background color are used to reflect the degree of exception of each cell, based on the precomputed exception indicators. Efficient algorithms have been proposed for cube construction, as discussed in Section 2.4.1. The computation of exception indicators can be overlapped with cube construction, so that the overall construction of data cubes for discovery-driven exploration is efficient.

Three measures are used as exception indicators to help identify data anomalies. These measures indicate the degree of surprise that the quantity in a cell holds, with respect to its expected value. The measures are computed and associated with every cell, for all levels of aggregation. They are:

1. **SelfExp**: This indicates the degree of surprise of the cell value, relative to other cells at the same level of aggregation.
2. **InExp**: This indicates the degree of surprise somewhere beneath the cell, if we were to drill down from it.
3. **PathExp**: This indicates the degree of surprise for each drill-down path from the cell.

The use of these measures for discovery-driven exploration of data cubes is illustrated in the following example.

Example 2.14 Suppose that you would like to analyze the monthly sales at *AllElectronics* as a percentage difference from the previous month. The dimensions involved are *item*, *time*, and *region*. You begin by studying the data aggregated over all items and sales regions for each month, as shown in Figure 2.17.

To view the exception indicators, you would click on a button marked highlight exceptions on the screen. This translates the SelfExp and InExp values into visual cues, displayed with each cell. The background color of each cell is based on its SelfExp value. In addition, a box is drawn around each cell, where the thickness and color of the box are a function of its InExp value. Thick boxes indicate high InExp values. In both cases, the darker the color is, the greater the degree of exception is. For example, the dark thick boxes for sales during July, August, and September signal the user to explore the lower level aggregations of these cells by drilling down.

Drill downs can be executed along the aggregated *item* or *region* dimensions. Which path has more exceptions? To find this out, you select a cell of interest and trigger a path exception module that colors each dimension based on the PathExp value of the cell. This value reflects the degree of surprise of that path. Consider the PathExp indicators for *item* and *region* in the upper left-hand corner of Figure 2.17. We see that the path along *item* contains more exceptions, as indicated by the darker color.

A drill-down along *item* results in the cube slice of Figure 2.18, showing the sales over time for each item. At this point, you are presented with many different sales values to analyze. By clicking on the highlight exceptions button, the visual cues are displayed, bringing focus towards the exceptions. Consider the sales difference of 41% for "Sony b/w printers" in September. This cell has a dark background, indicating a high SelfExp value, meaning that the cell is an exception. Consider now the the sales difference of -15% for "Sony b/w printers" in November, and of -11% in

item	all
region	all

Sum of sales	month											
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Total		1%	-1%	0%	1%	3%	-1	-9%	-1%	2%	-4%	3%

Figure 2.17: Change in sales over time.

Avg sales	month											
item	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Sony b/w printer		9%	-8%	2%	-5%	14%	-4%	0%	41%	-13%	-15%	-11%
Sony color printer		0%	0%	3%	2%	4%	-10%	-13%	0%	4%	-6%	4%
HP b/w printer		-2%	1%	2%	3%	8%	0%	-12%	-9%	3%	-3%	6%
HP color printer		0%	0%	-2%	1%	0%	-1%	-7%	-2%	1%	-5%	1%
IBM home computer		1%	-2%	-1%	-1%	3%	3%	-10%	4%	1%	-4%	-1%
IBM laptop computer		0%	0%	-1%	3%	4%	2%	-10%	-2%	0%	-9%	3%
Toshiba home computer		-2%	-5%	1%	1%	-1%	1%	5%	-3%	-5%	-1%	-1%
Toshiba laptop computer		1%	0%	3%	0%	-2%	-2%	-5%	3%	2%	-1%	0%
Logitech mouse		3%	-2%	-1%	0%	4%	6%	-11%	2%	1%	-4%	0%
Ergo-way mouse		0%	0%	2%	3%	1%	-2%	-2%	-5%	0%	-5%	8%

Figure 2.18: Change in sales for each item-time combination.

item	IBM home computer

Avg sales	month											
region	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
North		-1%	-3%	-1%	0%	3%	4%	-7%	1%	0%	-3%	-3%
South		-1%	1%	-9%	6%	-1%	-39%	9%	-34%	4%	1%	7%
East		-1%	-2%	2%	-3%	1%	18%	-2%	11%	-3%	-2%	-1%
West		4%	0%	-1%	-3%	5%	1%	-18%	8%	5%	-8%	1%

Figure 2.19: Change in sales for the item *"IBM home computer"* per region.

December. The -11% value for December is marked as an exception, while the -15% value is not, even though -15% is a bigger deviation than -11%. This is because the exception indicators consider all of the dimensions that a cell is in. Notice that the December sales of most of the other items have a large positive value, while the November sales do not. Therefore, by considering the position of the cell in the cube, the sales difference for "Sony b/w printers" in December is exceptional, while the November sales difference of this item is not.

The InExp values can be used to indicate exceptions at lower levels that are not visible at the current level. Consider the cells for "IBM home computers" in July and September. These both have a dark thick box around them, indicating high InExp values. You may decide to further explore the sales of "IBM home computers" by drilling down along *region*. The resulting sales difference by region is shown in Figure 2.19, where the highlight exceptions option has been invoked. The visual cues displayed make it easy to instantly notice an exception for the sales of "IBM home computers" in the southern region, where such sales have decreased by -39% and -34% in July and September, respectively. These detailed exceptions were far from obvious when we were viewing the data as an item-time group-by, aggregated over region in Figure 2.18. Thus, the InExp value is useful for searching for exceptions at lower level cells of the cube. Since there are no other cells in Figure 2.19 having a high InExp value, you may roll up back to the data of Figure 2.18, and choose another cell from which to drill down. In this way, the exception indicators can be used to guide the discovery of interesting anomalies in the data. □

"How are the exception values computed?" The SelfExp, InExp, and PathExp measures are based on a statistical method for table analysis. They take into account all of the group-by (aggregations) in which a given cell value participates. A cell value is considered an exception based on how much it differs from its expected value, where its expected value is determined with a statistical model described below. The difference between a given cell value and its expected value is called a **residual**. Intuitively, the larger the residual, the more the given cell value is an exception. The comparison of residual values requires us to scale the values based on the expected standard deviation associated with the residuals. A cell value is therefore considered an exception if its scaled residual value exceeds a prespecified threshold. The SelfExp, InExp, and PathExp measures are based on this scaled residual.

The expected value of a given cell is a function of the higher level group-by's of the given cell. For example, given a cube with the three dimensions A, B, and C, the expected value for a cell at the ith position in A, the jth position in B, and the kth position in C is a function of $\gamma, \gamma_i^A, \gamma_j^B, \gamma_k^C, \gamma_{ij}^{AB}, \gamma_{ik}^{AC}$, and γ_{jk}^{BC}, which are coefficients of the statistical model used. The coefficients reflect how different the values at more detailed levels are, based on generalized impressions formed by looking at higher level aggregations. In this way, the exception quality of a cell value is based on the exceptions of the values below it. Thus, when seeing an exception, it is natural for the user to further explore the exception by drilling down.

"How can the data cube be efficiently constructed for discovery-driven exploration?" This computation consists of three phases. The first step involves the computation of the aggregate values defining the cube, such as `sum` or `count`, over which exceptions will be found. There are several efficient techniques for cube computation, such as the multiway array aggregation technique discussed in Section 2.4.1. The second phase consists of model fitting, in

which the coefficients mentioned above are determined and used to compute the standardized residuals. This phase can be overlapped with the first phase since the computations involved are similar. The third phase computes the SelfExp, InExp, and PathExp values, based on the standardized residuals. This phase is computationally similar to phase 1. Therefore, the computation of data cubes for discovery-driven exploration can be done efficiently.

2.5.2 Complex aggregation at multiple granularities: Multifeature cubes

Data cubes facilitate the answering of data mining queries as they allow the computation of aggregate data at multiple levels of granularity. In this section, you will learn about *multifeature cubes* which compute complex queries involving multiple dependent aggregates at multiple granularities. These cubes are very useful in practice. Many complex data mining queries can be answered by multifeature cubes without any significant increase in computational cost, in comparison to cube computation for simple queries with standard data cubes.

All of the examples in this section are from the Purchases data of *AllElectronics*, where an *item* is purchased in a sales *region* on a business day (*year, month, day*). The shelf life in months of a given item is stored in *shelf*. The item price and sales (in dollars) at a given region are stored in *price* and *sales*, respectively. To aid in our study of multifeature cubes, let's first look at an example of a simple data cube.

Example 2.15 Query 1: A simple data cube query: Find the total sales in 1997, broken down by item, region, and month, with subtotals for each dimension.

To answer Query 1, a data cube is constructed which aggregates the total sales at the following 8 different levels of granularity: {(*item, region, month*), (*item, region*), (*item, month*), (*month, region*), (*item*), (*month*), (*region*), ()}, where () represents all. There are several techniques for computing such data cubes efficiently (Section 2.4.1). □

Query 1 uses a data cube like that studied so far in this chapter. We call such a data cube a simple data cube since it does not involve any dependent aggregates.

"*What is meant by "dependent aggregates"?*" We answer this by studying the following example of a complex query.

Example 2.16 Query 2: A complex query: Grouping by all subsets of {item, region, month}, find the maximum price in 1997 for each group, and the total sales among all maximum price tuples.

The specification of such a query using standard SQL can be long, repetitive, and difficult to optimize and maintain. Alternatively, Query 2 can be specified concisely using an extended SQL syntax as follows:

```
select      item, region, month, MAX(price), SUM(R.sales)
from        Purchases
where       year = 1997
cube by     item, region, month: R
such that   R.price = MAX(price)
```

The tuples representing purchases in 1997 are first selected. The cube by clause computes aggregates (or group-by's) for all possible combinations of the attributes item, region, and month. It is an n-dimensional generalization of the group by clause. The attributes specified in the cube by clause are the **grouping attributes**. Tuples with the same value on all grouping attributes form one group. Let the groups be $g_1, .., g_r$. For each group of tuples g_i, the maximum price max_{g_i} among the tuples forming the group is computed. The variable R is a **grouping variable**, ranging over all tuples in group g_i whose price is equal to max_{g_i} (as specified in the such that clause). The sum of sales of the tuples in g_i that R ranges over is computed, and returned with the values of the grouping attributes of g_i. The resulting cube is a **multifeature cube** in that it supports complex data mining queries for which multiple dependent aggregates are computed at a variety of granularities. For example, the sum of sales returned in Query 2 is dependent on the set of maximum price tuples for each group. □

Let's look at another example.

Example 2.17 Query 3: An even more complex query: Grouping by all subsets of {item, region, month}, find the maximum price in 1997 for each group. Among the maximum price tuples, find the minimum and maximum

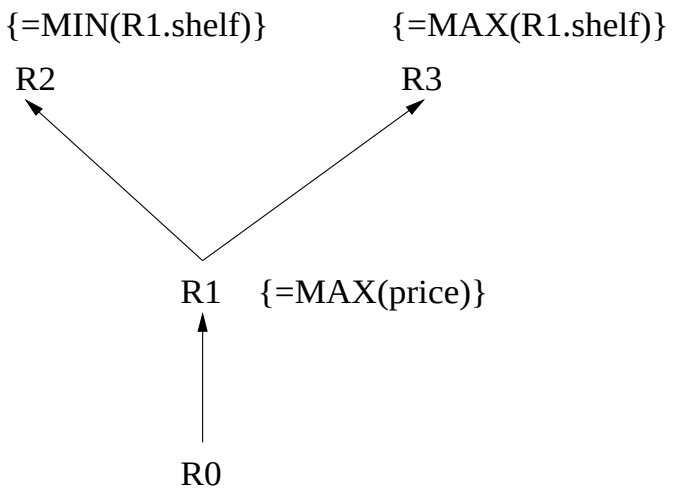


Figure 2.20: A multifeature cube graph for Query 3.

item shelf lives. Also find the fraction of the total sales due to tuples that have minimum shelf life within the set of all maximum price tuples, and the fraction of the total sales due to tuples that have maximum shelf life within the set of all maximum price tuples.

The **multifeature cube graph** of Figure 2.20 helps illustrate the aggregate dependencies in the query. There is one node for each grouping variable, plus an additional initial node, R0. Starting from node R0, the set of maximum price tuples in 1997 is first computed (node R1). The graph indicates that grouping variables R2 and R3 are "dependent" on R1, since a directed line is drawn from R1 to each of R2 and R3. In a multifeature cube graph, a directed line from grouping variable R_i to R_j means that R_j always ranges over a subset of the tuples that R_i ranges for. When expressing the query in extended SQL, we write "R_j in R_i" as shorthand to refer to this case. For example, the minimum shelf life tuples at R2 range over the maximum price tuples at R1, i.e., R2 in R1. Similarly, the maximum shelf life tuples at R3 range over the maximum price tuples at R1, i.e., R3 in R1.

From the graph, we can express Query 3 in extended SQL as follows:

select	item, region, month, MAX(price), MIN(R1.shelf), MAX(R1.shelf),
	SUM(R1.sales), SUM(R2.sales), SUM(R3.sales)
from	Purchases
where	year = 1997
cube by	item, region, month: R1, R2, R3
such that	R1.price = MAX(price) and
	R2 in R1 and R2.shelf = MIN(R1.shelf) and
	R3 in R1 and R3.shelf = MAX(R1.shelf)

□

"How can multifeature cubes be computed efficiently?" The computation of a multifeature cube depends on the types of aggregate functions used in the cube. Recall in Section 2.2.4, we saw that aggregate functions can be categorized as either *distributive* (such as `count()`, `sum()`, `min()`, and `max()`), *algebraic* (such as `avg()`, `min_N()`, `max_N()`), or *holistic* (such as `median()`, `mode()`, and `rank()`). Multifeature cubes can be organized into the same categories.

Intuitively, Query 2 is a distributive multifeature cube since we can distribute its computation by incrementally generating the output of the cube at a higher level granularity using only the output of the cube at a lower level granularity. Similarly, Query 3 is also distributive. Some multifeature cubes that are not distributive may be "converted" by adding aggregates to the select clause so that the resulting cube is distributive. For example, suppose that the select clause for a given multifeature cube has `AVG(sales)`, but neither `COUNT(sales)` nor `SUM(sales)`. By adding `SUM(sales)` to the select clause, the resulting data cube is distributive. The original cube is therefore algebraic. In the new distributive cube, the average sales at a higher level granularity can be computed from the average and total sales at lower level granularities. A cube that is neither distributive nor algebraic is holistic.

The type of multifeature cube determines the approach used in its computation. There are a number of methods for the efficient computation of data cubes (Section 2.4.1). The basic strategy of these algorithms is to exploit the lattice structure of the multiple granularities defining the cube, where higher level granularities are computed from lower level granularities. This approach suits distributive multifeature cubes, as described above. For example, in Query 2, the computation of `MAX(price)` at a higher granularity group can be done by taking the maximum of all of the `MAX(price)` values at the lower granularity groups. Similarly, `SUM(sales)` can be computed for a higher level group by summing all of the `SUM(sales)` values in its lower level groups. Some algorithms for efficient cube construction employ optimization techniques based on the estimated size of answers of groups within a data cube. Since the output size for each group in a multifeature cube is constant, the same estimation techniques can be used to estimate the size of intermediate results. Thus, the basic algorithms for efficient computation of simple data cubes can be used to compute distributive multifeature cubes for complex queries without any increase in I/O complexity. There may be a negligible increase in the CPU cost if the aggregate function of the multifeature cube is more complex than, say, a simple `SUM()`. Algebraic multifeature cubes must first be transformed into distributive multifeature cubes in order for these algorithms to apply. The computation of holistic multifeature cubes is sometimes significantly more expensive than the computation of distributive cubes, although the CPU cost involved is generally acceptable. Therefore, multifeature cubes can be used to answer complex queries with very little additional expense in comparison to simple data cube queries.

2.6 From data warehousing to data mining

2.6.1 Data warehouse usage

Data warehouses and data marts are used in a wide range of applications. Business executives in almost every industry use the data collected, integrated, preprocessed, and stored in data warehouses and data marts to perform data analysis and make strategic decisions. In many firms, data warehouses are used as an integral part of a *plan-execute-assess* "closed-loop" feedback system for enterprise management. Data warehouses are used extensively in banking and financial services, consumer goods and retail distribution sectors, and controlled manufacturing, such as demand-based production.

Typically, the longer a data warehouse has been in use, the more it will have evolved. This evolution takes place throughout a number of phases. Initially, the data warehouse is mainly used for generating reports and answering predefined queries. Progressively, it is used to analyze summarized and detailed data, where the results are presented in the form of reports and charts. Later, the data warehouse is used for strategic purposes, performing multidimensional analysis and sophisticated slice-and-dice operations. Finally, the data warehouse may be employed for knowledge discovery and strategic decision making using data mining tools. In this context, the tools for data warehousing can be categorized into *access and retrieval tools*, *database reporting tools*, *data analysis tools*, and *data mining tools*.

Business users need to have the means to know what exists in the data warehouse (through metadata), how to access the contents of the data warehouse, how to examine the contents using analysis tools, and how to present the results of such analysis.

There are three kinds of data warehouse applications: *information processing, analytical processing*, and *data mining*:

- **Information processing** supports querying, basic statistical analysis, and reporting using crosstabs, tables, charts or graphs. A current trend in data warehouse information processing is to construct low cost Web-based accessing tools which are then integrated with Web browsers.

- **Analytical processing** supports basic OLAP operations, including slice-and-dice, drill-down, roll-up, and pivoting. It generally operates on historical data in both summarized and detailed forms. The major strength of on-line analytical processing over information processing is the multidimensional data analysis of data warehouse data.

- **Data mining** supports knowledge discovery by finding hidden patterns and associations, constructing analytical models, performing classification and prediction, and presenting the mining results using visualization tools.

"*How does data mining relate to information processing and on-line analytical processing?*"

Information processing, based on queries, can find useful information. However, answers to such queries reflect the information directly stored in databases or computable by aggregate functions. They do not reflect sophisticated patterns or regularities buried in the database. Therefore, information processing is not data mining.

On-line analytical processing comes a step closer to data mining since it can derive information summarized at multiple granularities from user-specified subsets of a data warehouse. Such descriptions are equivalent to the class/concept descriptions discussed in Chapter 1. Since data mining systems can also mine generalized class/concept descriptions, this raises some interesting questions: Do OLAP systems perform data mining? Are OLAP systems actually data mining systems?

The functionalities of OLAP and data mining can be viewed as disjoint: OLAP is a data summarization/aggregation *tool* which helps simplify data analysis, while data mining allows the *automated discovery* of implicit patterns and interesting knowledge hidden in large amounts of data. OLAP tools are targeted toward simplifying and supporting interactive data analysis, but the goal of data mining tools is to automate as much of the process as possible, while still allowing users to guide the process. In this sense, data mining goes one step beyond traditional on-line analytical processing.

An alternative and broader view of data mining may be adopted in which data mining covers both data description and data modeling. Since OLAP systems can present general descriptions of data from data warehouses, OLAP functions are essentially for user-directed data summary and comparison (by drilling, pivoting, slicing, dicing, and other operations). These are, though limited, data mining functionalities. Yet according to this view, data mining covers a much broader spectrum than simple OLAP operations because it not only performs data summary and comparison, but also performs association, classification, prediction, clustering, time-series analysis, and other data analysis tasks.

Data mining is not confined to the analysis of data stored in data warehouses. It may analyze data existing at more detailed granularities than the summarized data provided in a data warehouse. It may also analyze transactional, textual, spatial, and multimedia data which are difficult to model with current multidimensional database technology. In this context, data mining covers a broader spectrum than OLAP with respect to data mining functionality and the complexity of the data handled.

Since data mining involves more automated and deeper analysis than OLAP, data mining is expected to have broader applications. Data mining can help business managers find and reach more suitable customers, as well as gain critical business insights that may help to drive market share and raise profits. In addition, data mining can help managers understand customer group characteristics and develop optimal pricing strategies accordingly, correct item bundling based not on intuition but on actual item groups derived from customer purchase patterns, reduce promotional spending and at the same time, increase net effectiveness of promotions overall.

2.6.2 From on-line analytical processing to on-line analytical mining

In the field of data mining, substantial research has been performed for data mining at various platforms, including transaction databases, relational databases, spatial databases, text databases, time-series databases, flat files, data warehouses, etc.

Among many different paradigms and architectures of data mining systems, **On-Line Analytical Mining (OLAM)** (also called **OLAP mining**), which integrates on-line analytical processing (OLAP) with data mining and mining knowledge in multidimensional databases, is particularly important for the following reasons.

1. **High quality of data in data warehouses.** Most data mining tools need to work on integrated, consistent, and cleaned data, which requires costly data cleaning, data transformation, and data integration as preprocessing steps. A data warehouse constructed by such preprocessing serves as a valuable source of high quality data for OLAP as well as for data mining. Notice that data mining may also serve as a valuable tool for data cleaning and data integration as well.

2. **Available information processing infrastructure surrounding data warehouses.** Comprehensive information processing and data analysis infrastructures have been or will be systematically constructed surrounding data warehouses, which include accessing, integration, consolidation, and transformation of multiple, heterogeneous databases, ODBC/OLEDB connections, Web-accessing and service facilities, reporting and OLAP

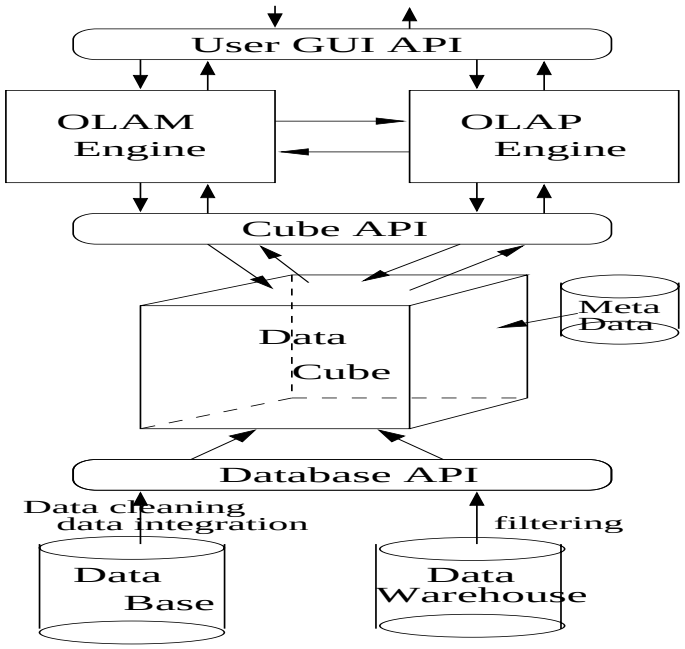


Figure 2.21: An integrated OLAM and OLAP architecture.

analysis tools. It is prudent to make the best use of the available infrastructures rather than constructing everything from scratch.

3. **OLAP-based exploratory data analysis.** Effective data mining needs exploratory data analysis. A user will often want to traverse through a database, select portions of relevant data, analyze them at different granularities, and present knowledge/results in different forms. On-line analytical mining provides facilities for data mining on different subsets of data and at different levels of abstraction, by drilling, pivoting, filtering, dicing and slicing on a data cube and on some intermediate data mining results. This, together with data/knowledge visualization tools, will greatly enhance the power and flexibility of exploratory data mining.

4. **On-line selection of data mining functions.** Often a user may not know what kinds of knowledge that she wants to mine. By integrating OLAP with multiple data mining functions, on-line analytical mining provides users with the flexibility to select desired data mining functions and swap data mining tasks dynamically.

Architecture for on-line analytical mining

An OLAM engine performs analytical mining in data cubes in a similar manner as an OLAP engine performs on-line analytical processing. An integrated OLAM and OLAP architecture is shown in Figure 2.21, where the OLAM and OLAP engines both accept users' on-line queries (or commands) via a User_GUI_API and work with the data cube in the data analysis via a Cube_API. A metadata directory is used to guide the access of the data cube. The data cube can be constructed by accessing and/or integrating multiple databases and/or by filtering a data warehouse via a Database_API which may support OLEDB or ODBC connections. Since an OLAM engine may perform multiple data mining tasks, such as concept description, association, classification, prediction, clustering, time-series analysis, etc., it usually consists of multiple, integrated data mining modules and is more sophisticated than an OLAP engine.

The following chapters of this book are devoted to the study of data mining techniques. As we have seen, the introduction to data warehousing and OLAP technology presented in this chapter is essential to our study of data mining. This is because data warehousing provides users with large amounts of clean, organized, and summarized data, which greatly facilitates data mining. For example, rather than storing the details of each sales transaction, a data warehouse may store a summary of the transactions per item type for each branch, or, summarized to a higher level, for each country. The capability of OLAP to provide multiple and dynamic views of summarized data in a data warehouse sets a solid foundation for successful data mining.

Moreover, we also believe that data mining should be a human-centered process. Rather than asking a data mining system to generate patterns and knowledge automatically, a user will often need to interact with the system to perform exploratory data analysis. OLAP sets a good example for interactive data analysis, and provides the necessary preparations for exploratory data mining. Consider the discovery of association patterns, for example. Instead of mining associations at a primitive (i.e., low) data level among transactions, users should be allowed to specify roll-up operations along any dimension. For example, a user may like to roll-up on the *item* dimension to go from viewing the data for particular TV sets that were purchased to viewing the brands of these TVs, such as SONY or Panasonic. Users may also navigate from the transaction level to the customer level or customer-type level in the search for interesting associations. Such an OLAP-style of data mining is characteristic of OLAP mining.

In our study of the principles of data mining in the following chapters, we place particular emphasis on OLAP mining, that is, on the *integration of data mining and OLAP technology.*

2.7 Summary

- A **data warehouse** is a *subject-oriented, integrated, time-variant*, and *nonvolatile* collection of data organized in support of management decision making. Several factors distinguish data warehouses from operational databases. Since the two systems provide quite different functionalities and require different kinds of data, it is necessary to maintain data warehouses separately from operational databases.
- A **multidimensional data model** is typically used for the design of corporate *data warehouses* and *departmental data marts*. Such a model can adopt either a *star schema*, *snowflake schema*, or *fact constellation schema*. The core of the *multidimensional model* is the **data cube**, which consists of a large set of *facts* (or *measures*) and a number of *dimensions*. Dimensions are the entities or perspectives with respect to which an organization wants to keep records, and are hierarchical in nature.
- **Concept hierarchies** organize the values of attributes or dimensions into gradual levels of abstraction. They are useful in mining at multiple levels of abstraction.
- **On-line analytical processing (OLAP)** can be performed in data warehouses/marts using the multidimensional data model. Typical OLAP operations include *roll-up*, *drill-*(*down, cross, through*), *slice-and-dice*, *pivot* (*rotate*), as well as statistical operations such as ranking, computing moving averages and growth rates, etc. OLAP operations can be implemented efficiently using the data cube structure.
- Data warehouses often adopt a **three-tier architecture**. The bottom tier is a *warehouse database server*, which is typically a relational database system. The middle tier is an *OLAP server*, and the top tier is a *client*, containing query and reporting tools.
- OLAP servers may use **Relational OLAP (ROLAP)**, or **Multidimensional OLAP (MOLAP)**, or **Hybrid OLAP (HOLAP)**. A ROLAP server uses an extended relational DBMS that maps OLAP operations on multidimensional data to standard relational operations. A MOLAP server maps multidimensional data views directly to array structures. A HOLAP server combines ROLAP and MOLAP. For example, it may use ROLAP for historical data while maintaining frequently accessed data in a separate MOLAP store.
- A data cube consists of a **lattice of cuboids**, each corresponding to a different degree of summarization of the given multidimensional data. **Partial materialization** refers to the selective computation of a subset of the cuboids in the lattice. **Full materialization** refers to the computation of all of the cuboids in the lattice. If the cubes are implemented using MOLAP, then **multiway array aggregation** can be used. This technique "overlaps" some of the aggregation computation so that full materialization can be computed efficiently.
- OLAP query processing can be made more efficient with the use of indexing techniques. In **bitmap indexing**, each attribute has its own bimap vector. Bitmap indexing reduces join, aggregation, and comparison operations to bit arithmetic. **Join indexing** registers the joinable rows of two or more relations from a relational database, reducing the overall cost of OLAP join operations. **Bitmapped join indexing**, which combines the bitmap and join methods, can be used to further speed up OLAP query processing.
- Data warehouse **metadata** are data defining the warehouse objects. A metadata repository provides details regarding the warehouse structure, data history, the algorithms used for summarization, mappings from the source data to warehouse form, system performance, and business terms and issues.

- A data warehouse contains **back-end tools and utilities** for populating and refreshing the warehouse. These cover data extraction, data cleaning, data transformation, loading, refreshing, and warehouse management.
- **Discovery-driven exploration** of data cubes uses precomputed measures and visual cues to indicate data exceptions, guiding the user in the data analysis process, at all levels of aggregation. **Multifeature cubes** compute complex queries involving multiple dependent aggregates at multiple granularities. The computation of cubes for discovery-driven exploration and of multifeature cubes can be achieved efficiently by taking advantage of efficient algorithms for standard data cube computation.
- Data warehouses are used for *information processing* (querying and reporting), *analytical processing* (which allows users to navigate through summarized and detailed data by OLAP operations), and *data mining* (which supports knowledge discovery). OLAP-based data mining is referred to as **OLAP mining**, or on-line analytical mining (**OLAM**), which emphasizes the interactive and exploratory nature of OLAP mining.

Exercises

1. State why, for the integration of multiple, heterogeneous information sources, many companies in industry prefer the update-driven approach (which constructs and uses data warehouses), rather than the query-driven approach (which applies wrappers and integrators). Describe situations where the query-driven approach is preferable over the update-driven approach.
2. Design a data warehouse for a regional weather bureau. The weather bureau has about 1,000 probes which are scattered throughout various land and ocean locations in the region to collect basic weather data, including air pressure, temperature, and precipitation at each hour. All data are sent to the central station, which has collected such data for over 10 years. Your design should facilitate efficient querying and on-line analytical processing, and derive general weather patterns in multidimensional space.
3. What are the differences between the three typical methods for modeling multidimensional data: the *star* model, the *snowflake* model, and the *fact constellation* model? What is the difference between the *star* warehouse model and the *starnet* query model? Use an example to explain your point(s).
4. A popular data warehouse implementation is to construct a multidimensional database, known as a data cube. Unfortunately, this may often generate a huge, yet very sparse multidimensional matrix.
   (a) Present an example, illustrating such a huge and sparse data cube.
   (b) Design an implementation method which can elegantly overcome this sparse matrix problem. Note that you need to explain your data structures in detail and discuss the space needed, as well as how to retrieve data from your structures, and how to handle incremental data updates.
5. Data warehouse design:
   (a) Enumerate three classes of schemas that are popularly used for modeling data warehouses.
   (b) Draw a schema diagram for a data warehouse which consists of three dimensions: *time, doctor*, and *patient*, and two measures: *count*, and *charge*, where *charge* is the fee that a doctor charges a patient for a visit.
   (c) Starting with the base cuboid $(day, doctor, patient)$, what specific OLAP operations should be performed in order to list the total fee collected by each doctor in VGH (Vancouver General Hospital) in 1997?
   (d) To obtain the same list, write an SQL query assuming the data is stored in a relational database with the schema.
   $$fee(day, month, year, doctor, hospital, patient, count, charge).$$
6. Computing measures in a data cube:
   (a) Enumerate three categories of measures, based on the kind of aggregate functions used in computing a data cube.
   (b) For a data cube with three dimensions: *time, location*, and *product*, which category does the function *variance* belong to? Describe how to compute it if the cube is partitioned into many chunks.
   Hint: The formula for computing *variance* is: $\frac{1}{n}\sum_{i=1}^{n}(x_i)^2 - \bar{x_i}^2$, where $\bar{x_i}$ is the average of x_i's.

(c) Suppose the function is "*top 10 sales*". Discuss how to efficiently compute this measure in a data cube.

7. Suppose that one needs to record three measures in a data cube: `min`, `average`, and `median`. Design an efficient computation and storage method for each measure given that the cube allows data to be *deleted* incrementally (i.e., in small portions at a time) from the cube.

8. In data warehouse technology, a multiple dimensional view can be implemented by a multidimensional database technique (MOLAP), or by a relational database technique (ROLAP), or a hybrid database technique (HOLAP).

   (a) Briefly describe each implementation technique.

   (b) For each technique, explain how each of the following functions may be implemented:

      i. The generation of a data warehouse (including aggregation).
      ii. Roll-up.
      iii. Drill-down.
      iv. Incremental updating.

   Which implementation techniques do you prefer, and why?

9. Suppose that a data warehouse contains 20 dimensions each with about 5 levels of granularity.

   (a) Users are mainly interested in four particular dimensions, each having three frequently accessed levels for rolling up and drilling down. How would you design a data cube structure to support this preference efficiently?

   (b) At times, a user may want to *drill-through* the cube, down to the raw data for one or two particular dimensions. How would you support this feature?

10. Data cube computation: Suppose a base cuboid has 3 dimensions, (A, B, C), with the number of cells shown below: $|A| = 1,000,000$, $|B| = 100$, and $|C| = 1,000$. Suppose each dimension is partitioned evenly into 10 portions for chunking.

    (a) Assuming each dimension has only one level, draw the complete lattice of the cube.

    (b) If each cube cell stores one measure with 4 bytes, what is the total size of the computed cube if the cube is *dense*?

    (c) If the cube is very *sparse*, describe an effective multidimensional array structure to store the sparse cube.

    (d) State the order for computing the chunks in the cube which requires the least amount of space, and compute the total amount of main memory space required for computing the 2-D planes.

11. In both data warehousing and data mining, it is important to have some hierarchy information associated with each dimension. If such a hierarchy is not given, discuss how to generate such a hierarchy automatically for the following cases:

    (a) a dimension containing only numerical data.

    (b) a dimension containing only categorical data.

12. Suppose that a data cube has 2 dimensions, (A, B), and each dimension can be generalized through 3 levels (with the top-most level being all). That is, starting with level A_0, A can be generalized to A_1, then to A_2, and then to all. How many *different* cuboids (i.e., views) can be generated for this cube? Sketch a lattice of these cuboids to show how you derive your answer. Also, give a general formula for a data cube with D dimensions, each starting at a base level and going up through L levels, with the top-most level being all.

13. Consider the following *multifeature cube* query: Grouping by all subsets of {item, region, month}, find the minimum shelf life in 1997 for each group, and the fraction of the total sales due to tuples whose price is less than \$100, and whose shelf life is within 25% of the minimum shelf life, and within 50% of the minimum shelf life.

    (a) Draw the multifeature cube graph for the query.

(b) Express the query in extended SQL.

(c) Is this a *distributive* multifeature cube? Why or why not?

14. What are the differences between the three main types of data warehouse usage: *information processing*, *analytical processing*, and *data mining*? Discuss the motivation behind *OLAP mining (OLAM)*.

Bibliographic Notes

There are a good number of introductory level textbooks on data warehousing and OLAP technology, including Inmon [15], Kimball [16], Berson and Smith [4], and Thomsen [24]. Chaudhuri and Dayal [6] provide a general overview of data warehousing and OLAP technology.

The history of decision support systems can be traced back to the 1960s. However, the proposal of the construction of large data warehouses for multidimensional data analysis is credited to Codd [7] who coined the term *OLAP* for *on-line analytical processing*. The OLAP council was established in 1995. Widom [26] identified several research problems in data warehousing. Kimball [16] provides an overview of the deficiencies of SQL regarding the ability to support comparisons that are common in the business world.

The DMQL data mining query language was proposed by Han et al. [11] Data mining query languages are further discussed in Chapter 4. Other SQL-based languages for data mining are proposed in Imielinski, Virmani, and Abdulghani [14], Meo, Psaila, and Ceri [17], and Baralis and Psaila [3].

Gray et al. [9, 10] proposed the data cube as a relational aggregation operator generalizing group-by, crosstabs, and sub-totals. Harinarayan, Rajaraman, and Ullman [13] proposed a greedy algorithm for the partial materialization of cuboids in the computation of a data cube. Agarwal et al. [1] proposed several methods for the efficient computation of multidimensional aggregates for ROLAP servers. The chunk-based multiway array aggregation method described in Section 2.4.1 for data cube computation in MOLAP was proposed in Zhao, Deshpande, and Naughton [27]. Additional methods for the fast computation of data cubes can be found in Beyer and Ramakrishnan [5], and Ross and Srivastava [19]. Sarawagi and Stonebraker [22] developed a chunk-based computation technique for the efficient organization of large multidimensional arrays.

For work on the selection of materialized cuboids for efficient OLAP query processing, see Harinarayan, Rajaraman, and Ullman [13], and Sristava et al. [23]. Methods for cube size estimation can be found in Beyer and Ramakrishnan [5], Ross and Srivastava [19], and Deshpande et al. [8]. Agrawal, Gupta, and Sarawagi [2] proposed operations for modeling multidimensional databases.

The use of join indices to speed up relational query processing was proposed by Valduriez [25]. O'Neil and Graefe [18] proposed a bitmapped join index method to speed-up OLAP-based query processing.

There are some recent studies on the implementation of discovery-oriented data cubes for data mining. This includes the discovery-driven exploration of OLAP data cubes by Sarawagi, Agrawal, and Megiddo [21], and the construction of multifeature data cubes by Ross, Srivastava, and Chatziantoniou [20]. For a discussion of methodologies for OLAM (On-Line Analytical Mining), see Han et al. [12].

Bibliography

[1] S. Agarwal, R. Agrawal, P. M. Deshpande, A. Gupta, J. F. Naughton, R. Ramakrishnan, and S. Sarawagi. On the computation of multidimensional aggregates. In *Proc. 1996 Int. Conf. Very Large Data Bases*, pages 506–521, Bombay, India, Sept. 1996.

[2] R. Agrawal, A. Gupta, and S. Sarawagi. Modeling multidimensional databases. In *Proc. 1997 Int. Conf. Data Engineering*, pages 232–243, Birmingham, England, April 1997.

[3] E. Baralis and G. Psaila. Designing templates for mining association rules. *Journal of Intelligent Information Systems*, 9:7–32, 1997.

[4] A. Berson and S. J. Smith. *Data Warehousing, Data Mining, and OLAP*. New York: McGraw-Hill, 1997.

[5] K. Beyer and R. Ramakrishnan. Bottom-up computation of sparse and iceberg cubes. In *Proc. 1999 ACM-SIGMOD Int. Conf. Management of Data*, pages 359–370, Philadelphia, PA, June 1999.

[6] S. Chaudhuri and U. Dayal. An overview of data warehousing and OLAP technology. *ACM SIGMOD Record*, 26:65–74, 1997.

[7] E. F Codd, S. B. Codd, and C. T. Salley. Providing OLAP (on-line analytical processing) to user-analysts: An IT mandate. In *E. F. Codd & Associates available at http://www.arborsoft.com/OLAP.html*, 1993.

[8] P. Deshpande, J. Naughton, K. Ramasamy, A. Shukla, K. Tufte, and Y. Zhao. Cubing algorithms, storage estimation, and storage and processing alternatives for olap. *Data Engineering Bulletin*, 20:3–11, 1997.

[9] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data cube: A relational operator generalizing group-by, cross-tab and sub-totals. In *Proc. 1996 Int. Conf. Data Engineering*, pages 152–159, New Orleans, Louisiana, Feb. 1996.

[10] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab and sub-totals. *Data Mining and Knowledge Discovery*, 1:29–54, 1997.

[11] J. Han, Y. Fu, W. Wang, J. Chiang, W. Gong, K. Koperski, D. Li, Y. Lu, A. Rajan, N. Stefanovic, B. Xia, and O. R. Zaïane. DBMiner: A system for mining knowledge in large relational databases. In *Proc. 1996 Int. Conf. Data Mining and Knowledge Discovery (KDD'96)*, pages 250–255, Portland, Oregon, August 1996.

[12] J. Han, Y. J. Tam, E. Kim, H. Zhu, and S. H. S. Chee. Methodologies for integration of data mining and on-line analytical processing in data warehouses. In *submitted to DAMI*, 1999.

[13] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data*, pages 205–216, Montreal, Canada, June 1996.

[14] T. Imielinski, A. Virmani, and A. Abdulghani. DataMine – application programming interface and query language for KDD applications. In *Proc. 1996 Int. Conf. Data Mining and Knowledge Discovery (KDD'96)*, pages 256–261, Portland, Oregon, August 1996.

[15] W. H. Inmon. *Building the Data Warehouse*. QED Technical Publishing Group, Wellesley, Massachusetts, 1992.

[16] R. Kimball. *The Data Warehouse Toolkit*. John Wiley & Sons, New York, 1996.

[17] R. Meo, G. Psaila, and S. Ceri. A new SQL-like operator for mining association rules. In *Proc. 1996 Int. Conf. Very Large Data Bases*, pages 122–133, Bombay, India, Sept. 1996.

[18] P. O'Neil and G. Graefe. Multi-table joins through bitmapped join indices. *SIGMOD Record*, 24:8–11, September 1995.

[19] K. Ross and D. Srivastava. Fast computation of sparse datacubes. In *Proc. 1997 Int. Conf. Very Large Data Bases*, pages 116–125, Athens, Greece, Aug. 1997.

[20] K. A. Ross, D. Srivastava, and D. Chatziantoniou. Complex aggregation at multiple granularities. In *Proc. Int. Conf. of Extending Database Technology (EDBT'98)*, pages 263–277, Valencia, Spain, March 1998.

[21] S. Sarawagi, R. Agrawal, and N. Megiddo. Discovery-driven exploration of OLAP data cubes. In *Proc. Int. Conf. of Extending Database Technology (EDBT'98)*, pages 168–182, Valencia, Spain, March 1998.

[22] S. Sarawagi and M. Stonebraker. Efficient organization of large multidimensional arrays. In *Proc. 1994 Int. Conf. Data Engineering*, pages 328–336, Feb. 1994.

[23] D. Sristava, S. Dar, H. V. Jagadish, and A. V. Levy. Answering queries with aggregation using views. In *Proc. 1996 Int. Conf. Very Large Data Bases*, pages 318–329, Bombay, India, September 1996.

[24] E. Thomsen. *OLAP Solutions: Building Multidimensional Information Systems.* John Wiley & Sons, 1997.

[25] P. Valduriez. Join indices. In *ACM Trans. Database System*, volume 12, pages 218–246, 1987.

[26] J. Widom. Research problems in data warehousing. In *Proc. 4th Int. Conf. Information and Knowledge Management*, pages 25–30, Baltimore, Maryland, Nov. 1995.

[27] Y. Zhao, P. M. Deshpande, and J. F. Naughton. An array-based algorithm for simultaneous multidimensional aggregates. In *Proc. 1997 ACM-SIGMOD Int. Conf. Management of Data*, pages 159–170, Tucson, Arizona, May 1997.

Contents

Chapter 3

Data Preprocessing

Today's real-world databases are highly susceptible to noise, missing, and inconsistent data due to their typically huge size, often several gigabytes or more. 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 a number of data preprocessing techniques. *Data cleaning* can be applied to remove noise and correct inconsistencies in the data. *Data integration* merges data from multiple sources into a coherent data store, such as a data warehouse or a data cube. *Data transformations*, such as normalization, may be applied. For example, normalization may improve the accuracy and efficiency of mining algorithms involving distance measurements. *Data reduction* can reduce the data size by aggregating, eliminating redundant features, or clustering, for instance. These data processing techniques, when applied prior to mining, can substantially improve the overall data mining results.

In this chapter, you will learn methods for data preprocessing. These methods are organized into the following categories: data cleaning, data integration and transformation, and data reduction. The use of concept hierarchies for data discretization, an alternative form of data reduction, is also discussed. Concept hierarchies can be further used to promote mining at multiple levels of abstraction. You will study how concept hierarchies can be generated automatically from the given data.

3.1 Why preprocess the data?

Imagine that you are a manager at *AllElectronics* and have been charged with analyzing the company's data with respect to the sales at your branch. You immediately set out to perform this task. You carefully study/inspect the company's database or data warehouse, identifying and selecting the attributes or dimensions to be included in your analysis, such as *item*, *price*, and *units_sold*. Alas! You note 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), **noisy** (containing errors, or *outlier* values which deviate from the expected), and **inconsistent** (e.g., containing discrepancies in the department codes used to categorize items). Welcome to the real world!

Incomplete, noisy, and inconsistent data are commonplace properties of large, real-world databases and data warehouses. 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 it was 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 history or modifications to the data may have been overlooked. Missing data, particularly for tuples with missing values for some attributes, may need to be inferred.

Data can be noisy, having incorrect attribute values, owing to the following. The data collection instruments used

may be faulty. There may have been human or computer errors occurring at data entry. 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 used. Duplicate tuples also require data cleaning.

Data cleaning routines work to "clean" the data by filling in missing values, smoothing noisy data, identifying or removing outliers, and resolving inconsistencies. Dirty data can cause confusion for the mining procedure. 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 "cleaning" up your data.

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* is 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, but "William" in another, and "B." in the third. Furthermore, you suspect that some attributes may be "derived" or 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 the data for a data warehouse. Additional data cleaning may be performed to detect and remove redundancies that may have resulted from data integration.

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. Such methods provide better results if the data to be analyzed have been *normalized*, that is, scaled to a specific range such as [0, 1.0]. Your customer data, for example, contains the attributes *age*, and *annual salary*. The *annual salary* attribute can take many more values than *age*. Therefore, if the attributes are left un-normalized, then distance measurements taken on *annual salary* will generally outweigh distance measurements taken on *age*. Furthermore, it would be useful for your analysis to obtain aggregate information as to the sales per customer region — something which is not part of any precomputed data cube in your data warehouse. You soon realize that **data transformation** operations, such as normalization and aggregation, are additional data preprocessing procedures that would contribute towards the success of the mining process. Data integration and data transformation are discussed in Section 3.3.

"Hmmm", you wonder, as you consider your data even further. *"The data set I have selected for analysis is huge — it is sure to slow or wear down the mining process. Is there any 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. There are a number of strategies for data reduction. These include *data aggregation* (e.g., building a data cube), *dimension reduction* (e.g., removing irrelevant attributes through correlation analysis), *data compression* (e.g., using encoding schemes such as minimum length encoding or wavelets), and *numerosity reduction* (e.g., "replacing" the data by alternative, smaller representations such as clusters, or parametric models). Data can also be "reduced" by *generalization*, where low level concepts such as *city* for customer location, are replaced with higher level concepts, such as *region* or *province_or_state*. A concept hierarchy is used to organize the concepts into varying levels of abstraction. Data reduction is the topic of Section 3.4. Since concept hierarchies are so useful in mining at multiple levels of abstraction, we devote a separate section to the automatic generation of this important data structure. Section 3.5 discusses concept hierarchy generation, a form of data reduction by data discretization.

Figure 3.1 summarizes the data preprocessing steps described here. Note that the above 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 the quality of the data, thereby helping to improve the accuracy and efficiency of the subsequent mining process. Data preprocessing is therefore an important step in the knowledge discovery process, since 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 pay-offs for decision making.

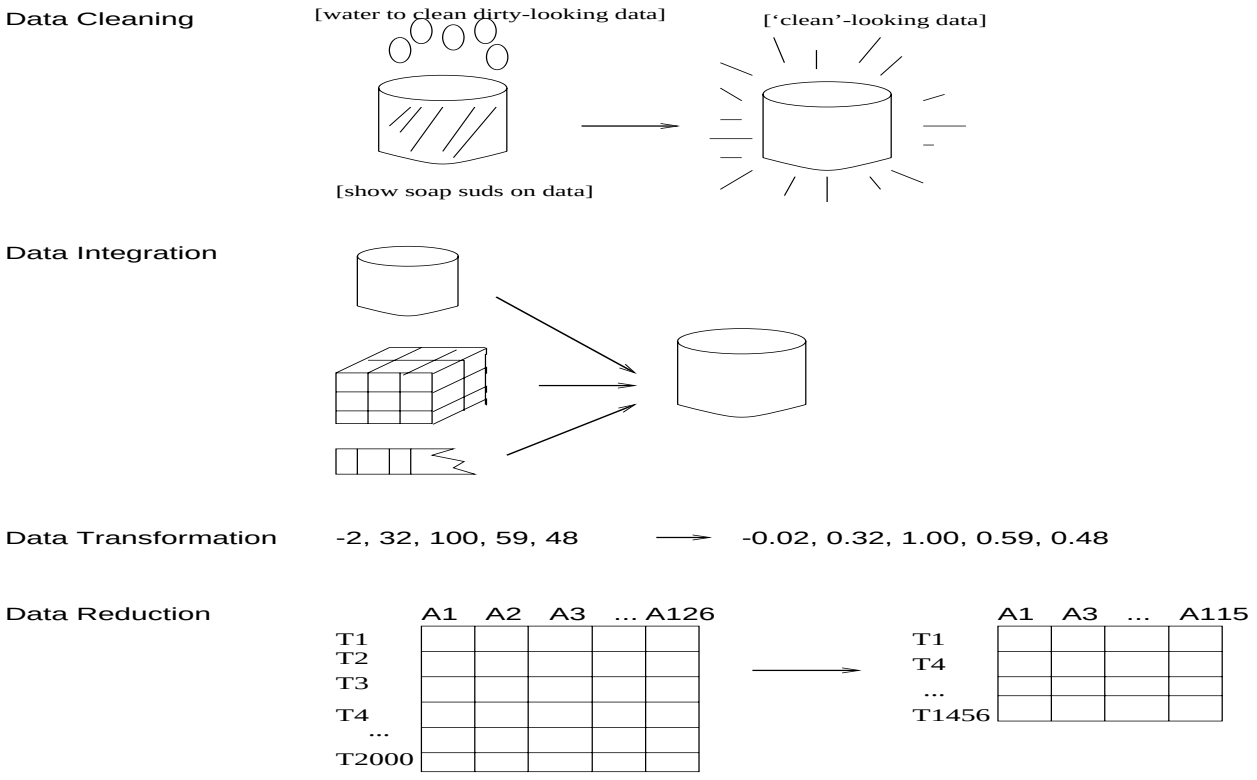


Figure 3.1: Forms of data preprocessing.

3.2 Data cleaning

Real-world data tend to be incomplete, noisy, and inconsistent. *Data cleaning* 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.

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 or description). 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.

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 recommended.

4. **Use the attribute mean to fill in the missing value**: For example, suppose that the average income of *AllElectronics* customers is \$28,000. Use this value to replace the missing value for *income*.

5. **Use the attribute mean for all samples belonging to the same class as the given tuple**: For example, if classifying customers according to *credit_risk*, replace the missing value with the average *income* value for

- **Sorted data for *price* (in dollars)**: 4, 8, 15, 21, 21, 24, 25, 28, 34
- **Partition into (equi-width) 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.

customers in the same credit risk category as that of the given tuple.

6. **Use the most probable value to fill in the missing value**: This may be determined with 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 are described in detail in Chapter 7.

Methods 3 to 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.

3.2.2 Noisy data

"What is noise?" **Noise** is a random error or variance in a measured variable. 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.

1. **Binning methods:** Binning methods smooth a sorted data value by consulting the "neighborhood", or 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 partitioned into *equi-depth* bins (of depth 3). 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 greater the effect of the smoothing. Alternatively, bins may be equi-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, and in Chapter 6 on association rule mining.

2. **Clustering:** Outliers may be detected by clustering, where similar values are organized into groups or "clusters". Intuitively, values which fall outside of the set of clusters may be considered outliers (Figure 3.3). Chapter 9 is dedicated to the topic of clustering.

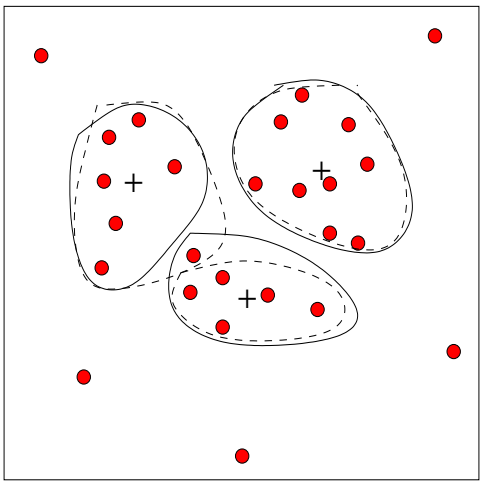


Figure 3.3: Outliers may be detected by clustering analysis.

3. **Combined computer and human inspection:** Outliers may be identified through a combination of computer and human inspection. In one application, for example, an information-theoretic measure was used to help identify outlier patterns in a handwritten character database for classification. The measure's value reflected the "surprise" content of the predicted character label with respect to the known label. Outlier patterns may be informative (e.g., identifying useful data exceptions, such as different versions of the characters "0" or "7"), or "garbage" (e.g., mislabeled characters). Patterns whose surprise content is above a threshold are output to a list. A human can then sort through the patterns in the list to identify the actual garbage ones. This is much faster than having to manually search through the entire database. The garbage patterns can then be removed from the (training) database.

4. **Regression:** Data can be smoothed by fitting the data to a function, such as with regression. *Linear regression* involves finding the "best" line to fit two variables, so that one variable can be used to predict the other. *Multiple linear regression* is an extension of linear regression, where more than two variables are involved and the data are fit to a multidimensional surface. Using regression to find a mathematical equation to fit the data helps smooth out the noise. Regression is further described in Section 3.4.4, as well as in Chapter 7.

Many methods for data smoothing are also methods of data reduction involving discretization. For example, the binning techniques described above 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 make value comparisons on sorted data. Concept hierarchies are a form of data discretization which can also be used for data smoothing. A concept hierarchy for *price*, for example, may map *price* real values into *"inexpensive"*, *"moderately_priced"*, and *"expensive"*, thereby reducing the number of data values to be handled by the mining process. Data discretization is discussed in Section 3.5. Some methods of classification, such as neural networks, have built-in data smoothing mechanisms. Classification is the topic of Chapter 7.

3.2.3 Inconsistent data

There may be inconsistencies in the data recorded for some transactions. 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. This may be coupled with routines designed to help correct the inconsistent use of codes. Knowledge engineering tools may also be used to detect the violation of known data constraints. For example, known functional dependencies between attributes can be used to find values contradicting the functional constraints.

There may also be inconsistencies due to data integration, where a given attribute can have different names in different databases. Redundancies may also result. Data integration and the removal of redundant data are described in Section 3.3.1.

3.3 Data integration and transformation

3.3.1 Data integration

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* can be tricky. How can like 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 entity? Databases and data warehouses typically have metadata - that is, data about the data. Such metadata can be used to help avoid errors in schema integration.

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

Some redundancies can be detected by **correlation analysis**. For example, given two attributes, such analysis can measure how strongly one attribute implies the other, based on the available data. The correlation between attributes A and B can be measured by

$$\frac{P(A \wedge B)}{P(A)P(B)}. \tag{3.1}$$

If the resulting value of Equation (3.1) is greater than 1, then A and B are positively correlated. The higher the value, the more each attribute implies the other. Hence, a high value may indicate that A (or B) may be removed as a redundancy. If the resulting value is equal to 1, then A and B are independent and there is no correlation between them. If the resulting value is less than 1, then A and B are negatively correlated. This means that each attribute discourages the other. Equation (3.1) may detect a correlation between the *customer_id* and *cust_number* attributes described above. Correlation analysis is further described in Chapter 6 (Section 6.5.2 on mining correlation rules).

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.

A third important issue in data integration is 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. The *price* of different hotels may involve not only different currencies but also different services (such as free breakfast) and taxes. Such semantic heterogeneity of data poses great challenges in data integration.

Careful integration of the data from multiple sources can help reduce and avoid redundancies and inconsistencies in the resulting data set. This can help improve the accuracy and speed of the subsequent mining process.

3.3.2 Data transformation

In *data transformation*, the data are transformed or consolidated into forms appropriate for mining. Data transformation can involve the following:

1. **Normalization**, where the attribute data are scaled so as to fall within a small specified range, such as -1.0 to 1.0, or 0 to 1.0.

2. **Smoothing**, which works to remove the noise from data. Such techniques include binning, clustering, and regression.

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 analysis of the data at multiple granularities.

4. **Generalization** of the data, where low level or 'primitive' (raw) data are replaced by higher level concepts through the use of concept hierarchies. For example, categorical attributes, like *street*, can be generalized to higher level concepts, like *city* or *county*. Similarly, values for numeric attributes, like *age*, may be mapped to higher level concepts, like *young, middle-aged*, and *senior*.

In this section, we discuss normalization. Smoothing is a form of data cleaning, and was discussed in Section 3.2.2. Aggregation and generalization also serve as forms of data reduction, and are discussed in Sections 3.4 and 3.5, respectively.

An attribute is normalized by scaling its values so that they fall within a small specified range, such as 0 to 1.0. 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 7), normalizing the input values for each attribute measured in the training samples 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). There are many methods for data normalization. We study three: min-max normalization, z-score normalization, and normalization by decimal scaling.

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 of A to v' in the range $[new\_min_A, new\_max_A]$ by computing

$$v' = \frac{v - min_A}{max_A - min_A}(new\_max_A - new\_min_A) + new\_min_A. \tag{3.2}$$

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 3.1 Suppose that the maximum and minimum values for the attribute *income* are \$98,000 and \$12,000, respectively. We would like to map *income* to the range $[0, 1]$. By min-max normalization, a value of \$73,600 for *income* is transformed to $\frac{73{,}600-12{,}000}{98{,}000-12{,}000}(1-0)+0 = 0.716$. □

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

$$v' = \frac{v - mean_A}{stand\_dev_A} \tag{3.3}$$

where $mean_A$ and $stand\_dev_A$ are the mean and standard deviation, respectively, of attribute A. This method of normalization is useful when the actual minimum and maximum of attribute A are unknown, or when there are outliers which dominate the min-max normalization.

Example 3.2 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 $\frac{73{,}600-54{,}000}{16{,}000} = 1.225$. □

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 of A is normalized to v' by computing

$$v' = \frac{v}{10^j}, \tag{3.4}$$

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

Example 3.3 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 1,000 (i.e., $j = 3$) so that -986 normalizes to -0.986. □

Note that normalization can change the original data quite a bit, especially the latter two of the methods shown above. It is also necessary to save the normalization parameters (such as the mean and standard deviation if using z-score normalization) so that future data can be normalized in a uniform manner.

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 may take a very long time, making such analysis impractical or infeasible. Is there any way to "reduce" the size of the data set without jeopardizing the data mining results?

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.

Strategies for data reduction include the following.

1. **Data cube aggregation**, where aggregation operations are applied to the data in the construction of a data cube.

2. **Dimension reduction**, where irrelevant, weakly relevant, or redundant attributes or dimensions may be detected and removed.

3. **Data compression**, where encoding mechanisms are used to reduce the data set size.

4. **Numerosity reduction**, where the data are replaced or estimated by alternative, smaller data representations such as parametric models (which need store only the model parameters instead of the actual data), or non-parametric methods such as clustering, sampling, and the use of histograms.

5. **Discretization and concept hierarchy generation**, where raw data values for attributes are replaced by ranges or higher conceptual levels. Concept hierarchies allow the mining of data at multiple levels of abstraction, and are a powerful tool for data mining. We therefore defer the discussion of automatic concept hierarchy generation to Section 3.5 which is devoted entirely to this topic.

Strategies 1 to 4 above are discussed in the remainder of this section. The computational time spent on data reduction should not outweight or "erase" the time saved by mining on a reduced data set size.

3.4.1 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 1997 to 1999. 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 3.4. The resulting data set is smaller in volume, without loss of information necessary for the analysis task.

Data cubes were discussed in Chapter 2. For completeness, we briefely review some of that material here. Data cubes store multidimensional aggregated information. For example, Figure 3.5 shows a data cube for multidimensional analysis of sales data with respect to annual sales per item type for each *AllElectronics* branch. Each cells holds an aggregate data value, corresponding to the data point in multidimensional space. Concept hierarchies may exist for each attribute, allowing the analysis of data at multiple levels of abstraction. 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 on-line analytical processing as well as data mining.

The cube created at the lowest level of abstraction is referred to as the base cuboid. A cube for the highest level of abstraction is the apex cuboid. For the sales data of Figure 3.5, 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 sometimes referred to as *cuboids*, so that a "data cube" may instead refer to a *lattice of cuboids*. Each higher level of abstraction further reduces the resulting data size.

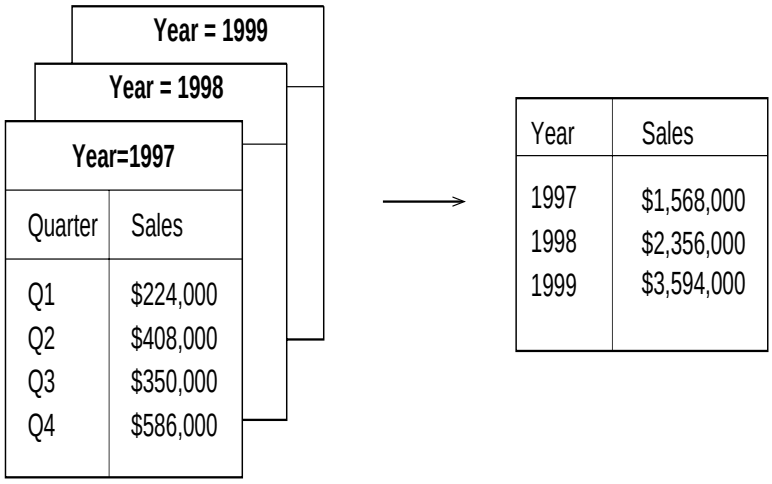


Figure 3.4: Sales data for a given branch of *AllElectronics* for the years 1997 to 1999. In the data on the left, the sales are shown per quarter. In the data on the right, the data are aggregated to provide the *annual_sales*.

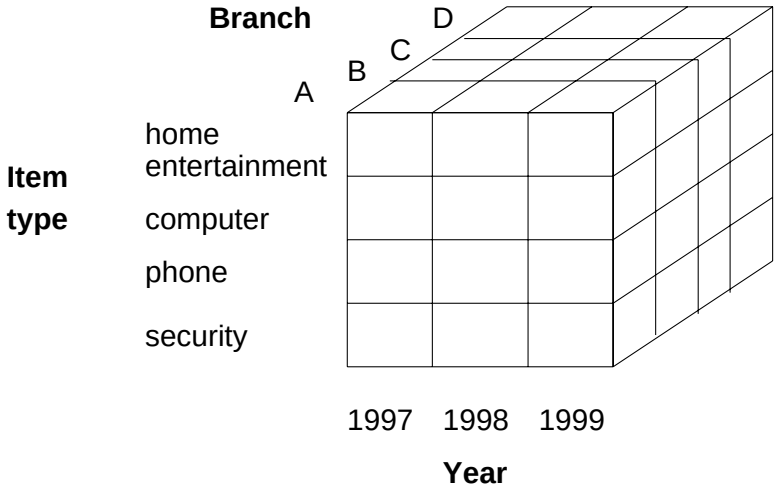


Figure 3.5: A data cube for sales at *AllElectronics*.

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. Since data cubes provide fast accessing to precomputed, summarized data, they should be used when possible to reply to queries regarding aggregated information. When replying to such OLAP queries or data mining requests, the *smallest* available cuboid relevant to the given task should be used. This issue is also addressed in Chapter 2.

3.4.2 Dimensionality reduction

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 as to 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 behavior of the data 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.

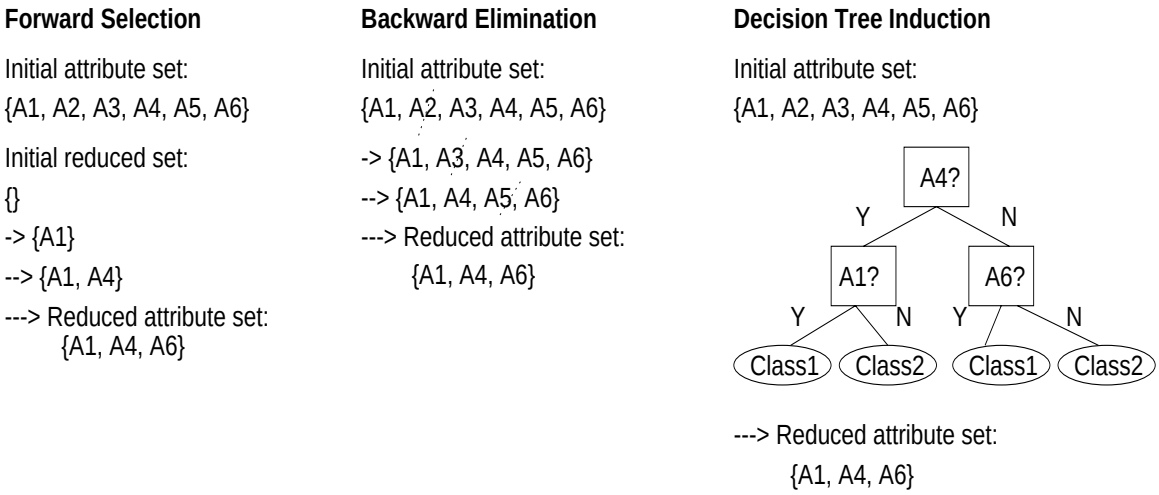


Figure 3.6: Greedy (heuristic) methods for attribute subset selection.

Dimensionality reduction reduces the data set size by removing such attributes (or dimensions) from it. Typically, methods of attribute subset selection are applied. 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?" There are 2^d possible subsets of d attributes. An exhaustive search for the optimal subset of attributes can be prohibitively expensive, especially as d and the number of data classes increase. Therefore, heuristic methods which 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 selected 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[1].

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

1. **Step-wise forward selection**: The procedure starts with an empty set of attributes. The best of the original attributes is determined and added to the set. At each subsequent iteration or step, the best of the remaining original attributes is added to the set.

2. **Step-wise 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 forward selection and backward elimination**: The step-wise forward selection and backward elimination methods can be combined, where at each step one selects the best attribute and removes the worst from among the remaining attributes.

The stopping criteria for methods 1 to 3 may vary. The procedure may employ a threshold on the measure used to determine when to stop the attribute selection process.

4. **Decision tree induction**: Decision tree algorithms, such as ID3 and C4.5, were originally intended for classification. Decision tree induction constructs a flow-chart-like structure where each internal (non-leaf) node denotes a test on an attribute, each branch corresponds to an outcome of the test, and each external (leaf)

[1] The information gain measure is described in Chapters 5 and 7.

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. This method of attribute selection is visited again in greater detail in Chapter 5 on concept description.

3.4.3 Data compression

In *data compression*, data encoding or 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 loss of information, the data compression technique used is called *lossless*. If, instead, we can reconstruct only an approximation of the original data, then the data compression technique is called *lossy*. There are several well-tuned algorithms for string compression. Although they are typically lossless, they allow only limited manipulation of the data. In this section, we instead focus on two popular and effective methods of lossy data compression: *wavelet transforms*, and *principal components analysis*.

Wavelet transforms

The **discrete wavelet transform (DWT)** is a linear signal processing technique that, when applied to a data vector D, transforms it to a numerically different vector, D', of **wavelet coefficients**. The two vectors are of the same length.

"Hmmm", you wonder. *"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. The remaining 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 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 are 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. Unlike DFT, wavelets are quite localized in space, contributing to the conservation of local detail.

There is only one DFT, yet there are several DWTs. The general algorithm for a discrete wavelet transform is as follows.

1. The length, L, of the input data vector must be an integer power of two. This condition can be met by padding the data vector with zeros, as necessary.

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.

3. The two functions are applied to pairs of the input data, resulting in two sets of data of length $L/2$. In general, these respectively represent a smoothed version of the input data, and the high-frequency content of it.

4. The two functions are recursively applied to the sets of data obtained in the previous loop, until the resulting data sets obtained are of desired length.

5. A selection of values from the data sets obtained in the above iterations are designated the wavelet coefficients of the transformed data.

Equivalently, a matrix multiplication can be applied to the input data in order to obtain the wavelet coefficients. For example, given an input vector of length 4 (represented as the column vector $[x_0, x_1, x_2, x_3]$), the 4-point **Haar**

transform of the vector can be obtained by the following matrix multiplication:

$$\begin{bmatrix} 1/2 & 1/2 & 1/2 & 1/2 \\ 1/2 & 1/2 & -1/2 & -1/2 \\ 1/\sqrt{2} & -1/\sqrt{2} & 0 & 0 \\ 0 & 0 & 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix} \times \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{3.5}$$

The matrix on the left is *orthonormal*, meaning that the columns are unit vectors (multiplied by a constant) 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. Other popular wavelet transforms include the Daubechies-4 and the Daubechies-6 transforms.

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 data with ordered attributes.

Principal components analysis

Herein, we provide an intuitive introduction to principal components analysis as a method of data compression. A detailed theoretical explanation is beyond the scope of this book.

Suppose that the data to be compressed consists of N tuples or data vectors, from k-dimensions. **Principal components analysis (PCA)** searches for c k-dimensional orthogonal vectors that can best be used to represent the data, where $c << N$. The original data is thus projected onto a much smaller space, resulting in data compression. PCA can be used as a form of dimensionality reduction. However, unlike attribute subset selection, 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.

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 N orthonormal vectors which 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, 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. This information helps identify groups or patterns within the data.
4. Since the components are sorted according to decreasing order of "significance", the size of the data can be reduced by eliminating the weaker components, i.e., 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. For example, a 3-D data cube for sales with the dimensions *item_type, branch*, and *year* must first be reduced to a 2-D cube, such as with the dimensions *item_type*, and *branch* $\times$ *year*.

3.4.4 Numerosity reduction

"Can we reduce the data volume by choosing alternative, 'smaller' forms of data representation?" Techniques of *numerosity reduction* can indeed be applied for this purpose. These techniques may be parametric or non-parametric. For *parametric methods*, a model is used to estimate the data, so that typically only the data parameters need be stored, instead of the actual data. (Outliers may also be stored). Log-linear models, which estimate discrete multidimensional probability distributions, are an example. *Non-parametric methods* for storing reduced representations of the data include histograms, clustering, and sampling.

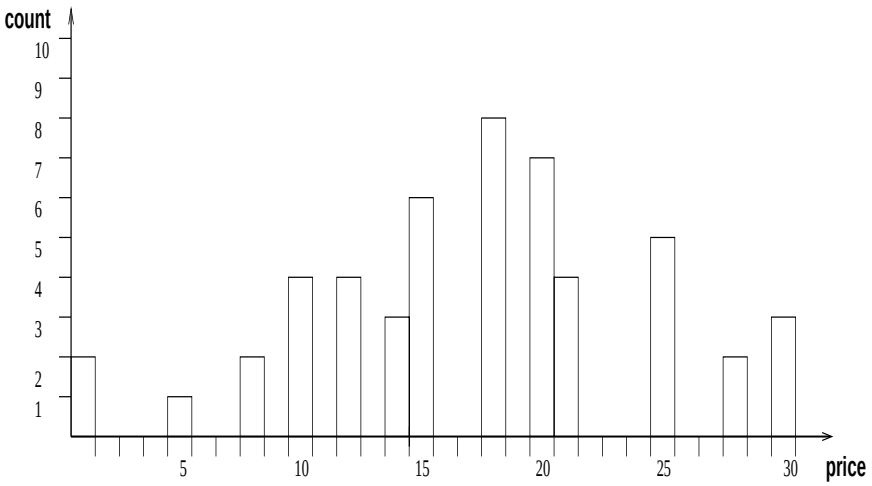


Figure 3.7: A histogram for *price* using singleton buckets - each bucket represents one price-value/frequency pair.

Let's have a look at each of the numerosity reduction techniques mentioned above.

Regression and log-linear models

Regression and log-linear models can be used to approximate the given data. In **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

$$Y = \alpha + \beta X, \tag{3.6}$$

where the variance of Y is assumed to be constant. The coefficients α and β (called *regression coefficients*) specify the Y-intercept and slope of the line, 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 regression** is an extension of linear regression allowing a response variable Y to be modeled as a linear function of a multidimensional feature vector.

Log-linear models approximate discrete multidimensional probability distributions. The method can be used to estimate the probability of each cell in a base cuboid for a set of discretized attributes, based on the smaller cuboids making up the data cube lattice. This allows higher order data cubes to be constructed from lower order ones. Log-linear models are therefore also useful for data compression (since the smaller order cuboids together typically occupy less space than the base cuboid) and data smoothing (since cell estimates in the smaller order cuboids are less subject to sampling variations than cell estimates in the base cuboid). Regression and log-linear models are further discussed in Chapter 7 (Section 7.8 on Prediction).

Histograms

Histograms use binning to approximate data distributions and are a popular form of data reduction. A **histogram** for an attribute A partitions the data distribution of A into disjoint subsets, or *buckets*. The buckets are displayed on a horizontal axis, while the height (and area) of a bucket typically reflects the average frequency of the values represented by the bucket. 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 3.4 The following data are a list of prices of commonly sold items at *AllElectronics* (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.

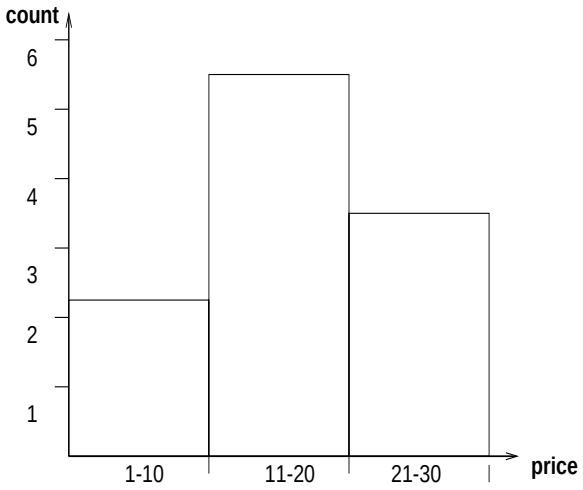


Figure 3.8: A histogram for *price* where values are aggregated so that each bucket has a uniform width of $10.

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 range of values for the given attribute. In Figure 3.8, each bucket represents a different $10 range for *price*. □

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

1. **Equi-width**: In an equi-width histogram, the width of each bucket range is constant (such as the width of $10 for the buckets in Figure 3.8).

2. **Equi-depth** (or equi-height): In an equi-depth histogram, the buckets are created so that, roughly, the frequency of each bucket is constant (that is, each bucket contains roughly the same number of contiguous data samples).

3. **V-Optimal**: If we consider all of the possible histograms for a given number of buckets, the V-optimal histogram is the one with the least variance. Histogram variance is a weighted sum of the original values that each bucket represents, where bucket weight is equal to the number of values in the bucket.

4. **MaxDiff**: In a MaxDiff histogram, we consider the difference between each pair of adjacent values. A bucket boundary is established between each pair for pairs having the $\beta-1$ largest differences, where β is user-specified.

V-Optimal and MaxDiff histograms tend to be the most accurate and practical. Histograms are highly effective at approximating both sparse and dense data, as well as highly skewed, and uniform data. The histograms described above for single attributes can be extended for multiple attributes. *Multidimensional histograms* can capture dependencies between attributes. Such histograms have been found effective in approximating data with up to five attributes. More studies are needed regarding the effectiveness of multidimensional histograms for very high dimensions. Singleton buckets are useful for storing outliers with high frequency. Histograms are further described in Chapter 5 (Section 5.6 on mining descriptive statistical measures in large databases).

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 3.9 shows

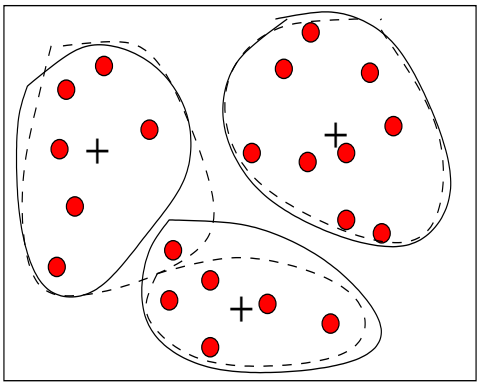


Figure 3.9: A 2-D plot of customer data with respect to customer locations in a city, showing three data clusters. Each cluster centroid is marked with a "+".

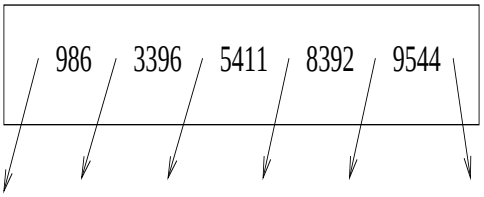


Figure 3.10: The root of a B+-tree for a given set of data.

a 2-D plot of customer data with respect to customer locations in a city, where the centroid of each cluster is shown with a "+". 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 nature of the data. It is much more effective for data that can be organized into distinct clusters, than for smeared data.

In database systems, **multidimensional index trees** are primarily used for providing fast data access. They can also be used for hierarchical data reduction, providing a multiresolution clustering of the data. This can be used to provide approximate answers to queries. An index tree recursively partitions the multidimensional space for a given set of data objects, with the root node representing the entire space. Such trees are typically balanced, consisting of internal and leaf nodes. Each parent node contains keys and pointers to child nodes that, collectively, represent the space represented by the parent node. Each leaf node contains pointers to the data tuples they represent (or to the actual tuples).

An index tree can therefore store aggregate and detail data at varying levels of resolution or abstraction. It provides a hierarchy of clusterings of the data set, where each cluster has a label that holds for the data contained in the cluster. If we consider each child of a parent node as a bucket, then an index tree can be considered as a *hierarchical histogram*. For example, consider the root of a B+-tree as shown in Figure 3.10, with pointers to the data keys 986, 3396, 5411, 8392, and 9544. Suppose that the tree contains 10,000 tuples with keys ranging from 1 to 9,999. The data in the tree can be approximated by an equi-depth histogram of 6 buckets for the key ranges 1 to 985, 986 to 3395, 3396 to 5410, 5411 to 8392, 8392 to 9543, and 9544 to 9999. Each bucket contains roughly 10,000/6 items. Similarly, each bucket is subdivided into smaller buckets, allowing for aggregate data at a finer-detailed level. The use of multidimensional index trees as a form of data resolution relies on an ordering of the attribute values in each dimension. Multidimensional index trees include R-trees, quad-trees, and their variations. They are well-suited for handling both sparse and skewed data.

There are many measures for defining clusters and cluster quality. Clustering methods are further described in Chapter 8.

Sampling

Sampling can be used as a data reduction technique since it allows a large data set to be represented by a much smaller random sample (or subset) of the data. Suppose that a large data set, D, contains N tuples. Let's have a look at some possible samples for D.

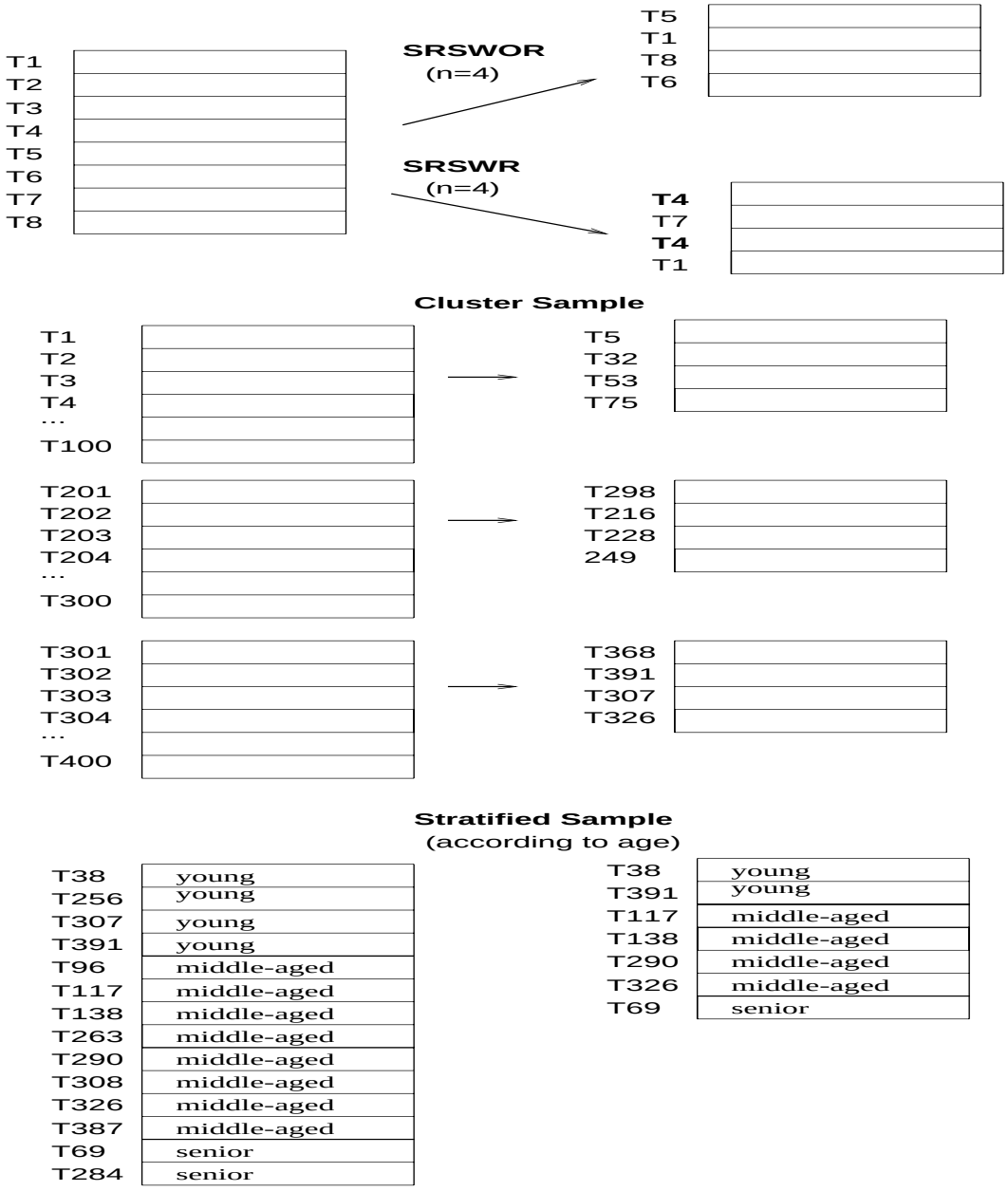


Figure 3.11: Sampling can be used for data reduction.

1. **Simple random sample without replacement (SRSWOR)** of size n: This is created by drawing n of the N tuples from D ($n < N$), where the probably of drawing any tuple in D is $1/N$, i.e., all tuples are equally likely.

2. **Simple random sample with replacement (SRSWR) of size** n: 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.

3. **Cluster sample**: If the tuples in D are grouped into M mutually disjoint "clusters", then a SRS of m clusters can be obtained, where $m < M$. For example, tuples in a database are usually retrieved a page at a time, so that each page can be considered a cluster. A reduced data representation can be obtained by applying, say, SRSWOR to the pages, resulting in a cluster sample of the tuples.

4. **Stratified sample**: If D is divided into mutually disjoint parts called "strata", a stratified sample of D is generated by obtaining a SRS at each stratum. This helps to ensure a representative sample, especially when the data are skewed. For example, a stratified sample may be obtained from customer data, where 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.

These samples are illustrated in Figure 3.11. They represent the most commonly used forms of sampling for data reduction.

An advantage of sampling for data reduction is that the cost of obtaining a sample *is proportional to the size of the sample*, n, as opposed to N, the data set size. Hence, sampling complexity is potentially *sub-linear* 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, d, increases, while techniques using histograms, for example, increase exponentially in d.

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, n, 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.

3.5 Discretization and concept hierarchy generation

Discretization techniques can be used to reduce the number of values for a given continuous attribute, by dividing the range of the attribute into intervals. Interval labels can then be used to replace actual data values. Reducing the number of values for an attribute is especially beneficial if decision tree-based methods of classification mining are to be applied to the preprocessed data. These methods are typically recursive, where a large amount of time is spent on sorting the data at each step. Hence, the smaller the number of distinct values to sort, the faster these methods should be. Many discretization techniques can be applied recursively in order to provide a hierarchical, or multiresolution partitioning of the attribute values, known as a concept hierarchy. Concept hierarchies were introduced in Chapter 2. They are useful for mining at multiple levels of abstraction.

A concept hierarchy for a given numeric attribute defines a discretization of the attribute. Concept hierarchies can be used to reduce the data by collecting and replacing low level concepts (such as numeric values for the attribute *age*) by higher level concepts (such as *young, middle-aged*, or *senior*). Although detail is lost by such data generalization, the generalized data may be more meaningful and easier to interpret, and will require less space than the original data. Mining on a reduced data set will require fewer input/output operations and be more efficient than mining on a larger, ungeneralized data set. An example of a concept hierarchy for the attribute *price* is given in Figure 3.12. More than one concept hierarchy can be defined for the same attribute in order to accommodate the needs of the various users.

Manual definition of concept hierarchies can be a tedious and time-consuming task for the user or domain expert. Fortunately, many hierarchies are implicit within the database schema, and can be defined at the schema definition level. Concept hierarchies often can be automatically generated or dynamically refined based on statistical analysis of the data distribution.

Let's look at the generation of concept hierarchies for numeric and categorical data.

3.5.1 Discretization and concept hierarchy generation for numeric data

It is difficult and tedious to specify concept hierarchies for numeric attributes due to the wide diversity of possible data ranges and the frequent updates of data values.

Concept hierarchies for numeric attributes can be constructed automatically based on data distribution analysis. We examine five methods for numeric concept hierarchy generation. These include *binning*, *histogram analysis*,

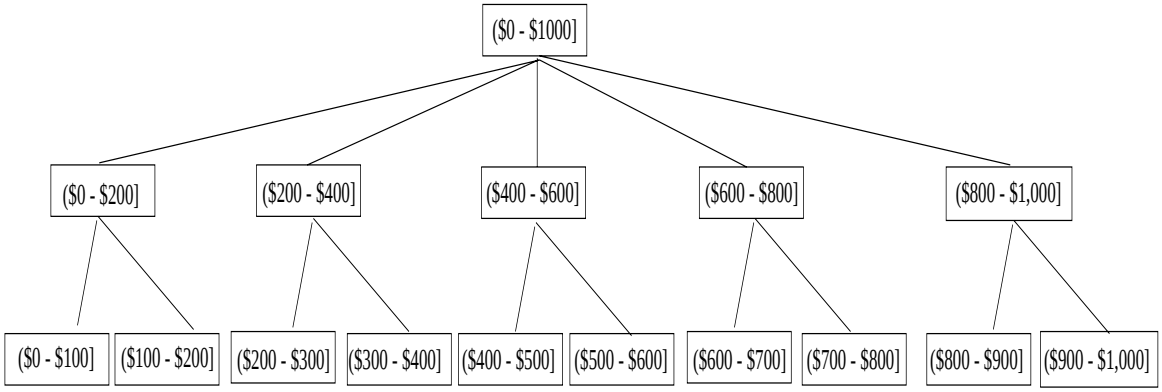


Figure 3.12: A concept hierarchy for the attribute *price*.

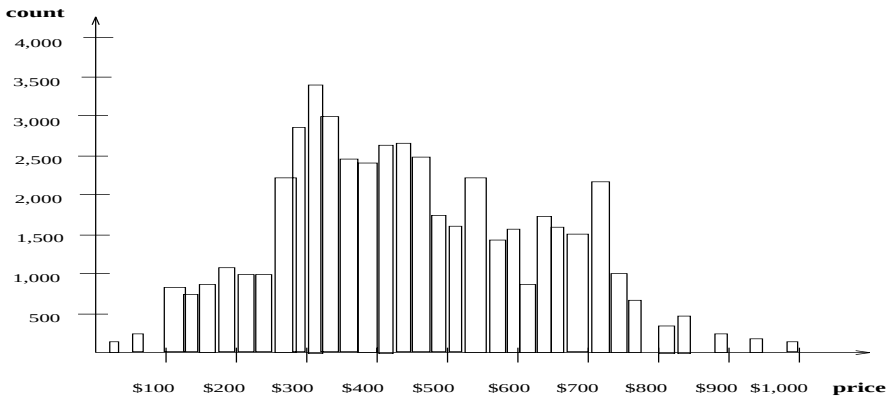


Figure 3.13: Histogram showing the distribution of values for the attribute *price*.

clustering analysis, *entropy-based discretization*, and *data segmentation by "natural partitioning"*.

1. **Binning.**

   Section 3.2.2 discussed binning methods for data smoothing. These methods are also forms of discretization. For example, attribute values can be discretized by 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 in order to generate concept hierarchies.

2. **Histogram analysis.**

   Histograms, as discussed in Section 3.4.4, can also be used for discretization. Figure 3.13 presents a **histogram** showing the data distribution of the attribute *price* for a given data set. For example, the most frequent price range is roughly \$300-\$325. Partitioning rules can be used to define the ranges of values. For instance, in an *equi-width* histogram, the values are partitioned into equal sized partions or ranges (e.g., (\$0-\$100], (\$100-\$200], ..., (\$900-\$1,000]). With an *equi-depth* histogram, the values are partitioned so that, ideally, each partition contains the same number of data samples. 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 pre-specified 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. A concept hierarchy for *price*, generated from the data of Figure 3.13 is shown in Figure 3.12.

3. **Clustering analysis.**

   A clustering algorithm can be applied to partition data into clusters or groups. Each cluster forms a node of a concept hierarchy, where all nodes are at the same conceptual level. Each cluster may be further decomposed

into several subclusters, forming a lower level of the hierarchy. Clusters may also be grouped together in order to form a higher conceptual level of the hierarchy. Clustering methods for data mining are studied in Chapter 8.

4. **Entropy-based discretization.**

An information-based measure called "entropy" can be used to recursively partition the values of a numeric attribute A, resulting in a hierarchical discretization. Such a discretization forms a numerical concept hierarchy for the attribute. Given a set of data tuples, S, the basic method for entropy-based discretization of A is as follows.

- Each value of A can be considered a potential interval boundary or threshold T. For example, a value v of A can partition the samples in S into two subsets satisfying the conditions $A < v$ and $A \geq v$, respectively, thereby creating a binary discretization.
- Given S, the threshold value selected is the one that maximizes the information gain resulting from the subsequent partitioning. The information gain is:

$$I(S,T) = \frac{|S_1|}{|S|} Ent(S_1) + \frac{|S_2|}{|S|} Ent(S_2), \tag{3.7}$$

where S_1 and S_2 correspond to the samples in S satisfying the conditions $A < T$ and $A \geq T$, respectively. The entropy function *Ent* for a given set is calculated based on the class distribution of the samples in the set. For example, given m classes, the entropy of S_1 is:

$$Ent(S_1) = -\sum_{i=1}^{m} p_i log_2(p_i), \tag{3.8}$$

where p_i is the probability of class i in S_1, determined by dividing the number of samples of class i in S_1 by the total number of samples in S_1. The value of *Ent(*S_2*)* can be computed similarly.

- The process of determining a threshold value is recursively applied to each partition obtained, until some stopping criterion is met, such as

$$Ent(S) - I(S,T) > \delta \tag{3.9}$$

Experiments show that entropy-based discretization can reduce data size and may improve classification accuracy. The information gain and entropy measures described here are also used for decision tree induction. These measures are revisited in greater detail in Chapter 5 (Section 5.4 on analytical characterization) and Chapter 7 (Section 7.3 on decision tree induction).

5. **Segmentation by natural partitioning**.

Although binning, histogram analysis, clustering and entropy-based discretization are useful in the generation of numerical hierarchies, many users would like to see numerical ranges partitioned into relatively uniform, easy-to-read intervals that appear intuitive or "natural". For example, annual salaries broken into ranges like [$50,000, $60,000) are often more desirable than ranges like [$51263.98, $60872.34), obtained by some sophisticated clustering analysis.

The **3-4-5 rule** can be used to segment numeric data into relatively uniform, "natural" intervals. In general, the rule partitions a given range of data into either 3, 4, or 5 relatively equi-length intervals, recursively and level by level, based on the value range at the most significant digit. The rule is as follows.

(a) If an interval covers 3, 6, 7 or 9 distinct values at the most significant digit, then partition the range into 3 intervals (3 equi-width intervals for 3, 6, 9, and three intervals in the grouping of 2-3-2 for 7);

(b) if it covers 2, 4, or 8 distinct values at the most significant digit, then partition the range into 4 equi-width intervals; and

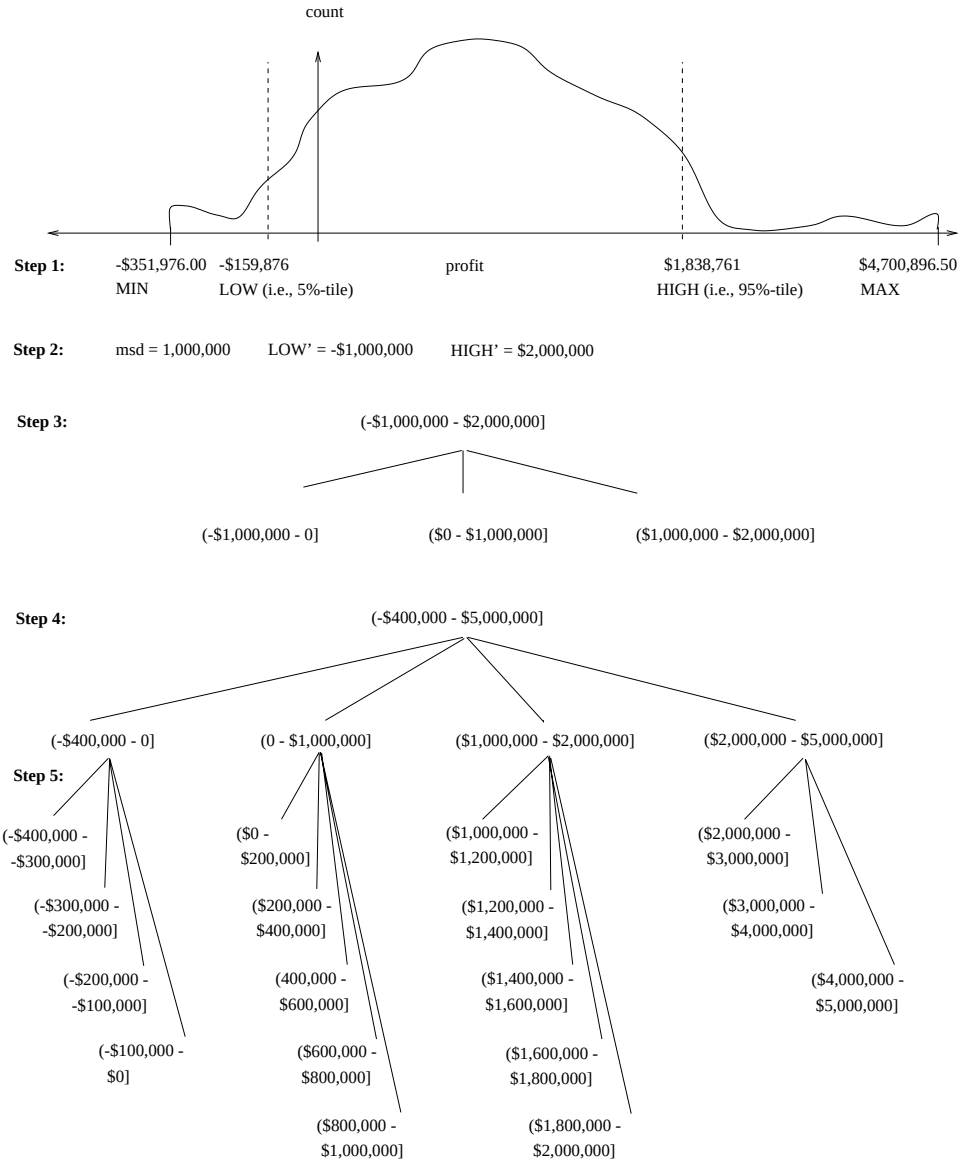


Figure 3.14: Automatic generation of a concept hierarchy for *profit* based on the 3-4-5 rule.

(c) if it covers 1, 5, or 10 distinct values at the most significant digit, then partition the range into 5 equi-width intervals.

The rule can be recursively applied to each interval, creating a concept hierarchy for the given numeric attribute. Since there could be some dramatically large positive or negative values in a data set, the top level segmentation, based merely on the minimum and maximum values, may derive distorted results. For example, the assets of a few people could be several orders of magnitude higher than those of others in a data set. Segmentation based on the maximal asset values may lead to a highly biased hierarchy. Thus the top level segmentation can be performed based on the range of data values representing the majority (e.g., 5%-tile to 95%-tile) of the given data. The extremely high or low values beyond the top level segmentation will form distinct interval(s) which can be handled separately, but in a similar manner.

The following example illustrates the use of the 3-4-5 rule for the automatic construction of a numeric hierarchy.

Example 3.5 Suppose that profits at different branches of *AllElectronics* for the year 1997 cover a wide range, from −$351,976.00 to $4,700,896.50. A user wishes to have a concept hierarchy for *profit* automatically

generated. For improved readability, we use the notation $(l \text{ — } r]$ to represent the interval $(l, r]$. For example, (−$1,000,000 — $0] denotes the range from −$1,000,000 (exclusive) to $0 (inclusive).

Suppose that the data within the 5%-tile and 95%-tile are between −$159,876 and $1,838,761. The results of applying the 3-4-5 rule are shown in Figure 3.14.

- *Step 1:* Based on the above information, the minimum and maximum values are: $MIN = -\$351,976.00$, and $MAX = \$4,700,896.50$. The low (5%-tile) and high (95%-tile) values to be considered for the top or first level of segmentation are: $LOW = -\$159,876$, and $HIGH = \$1,838,761$.
- *Step 2:* Given LOW and HIGH, the most significant digit is at the million dollar digit position (i.e., $msd =$ 1,000,000). Rounding LOW down to the million dollar digit, we get $LOW' = -\$1,000,000$; and rounding $HIGH$ up to the million dollar digit, we get $HIGH' = +\$2,000,000$.
- *Step 3:* Since this interval ranges over 3 distinct values at the most significant digit, i.e., $(2,000,000 - (-1,000,000))/1,000,000 = 3$, the segment is partitioned into 3 equi-width subsegments according to the 3-4-5 rule: (−$1,000,000 — $0], ($0 — $1,000,000], and ($1,000,000 — $2,000,000]. This represents the top tier of the hierarchy.
- *Step 4:* We now examine the MIN and MAX values to see how they "fit" into the first level partitions.

  Since the first interval, $(-\$1,000,000 \text{ — } \$0]$ covers the MIN value, i.e., $LOW' < MIN$, we can adjust the left boundary of this interval to make the interval smaller. The most significant digit of MIN is the hundred thousand digit position. Rounding MIN down to this position, we get $MIN' = -\$400,000$. Therefore, the first interval is redefined as $(-\$400,000 \text{ — } 0]$.

  Since the last interval, ($1,000,000 — $2,000,000] does not cover the MAX value, i.e., $MAX > HIGH'$, we need to create a new interval to cover it. Rounding up MAX at its most significant digit position, the new interval is ($2,000,000 — $5,000,000]. Hence, the top most level of the hierarchy contains four partitions, (−$400,000 — $0], ($0 — $1,000,000], ($1,000,000 — $2,000,000], and ($2,000,000 — $5,000,000].
- *Step 5:* Recursively, each interval can be further partitioned according to the 3-4-5 rule to form the next lower level of the hierarchy:
  - The first interval (−$400,000 — $0] is partitioned into 4 sub-intervals: (−$400,000 — −$300,000], (−$300,000 — −$200,000], (−$200,000 — −$100,000], and (−$100,000 — $0].
  - The second interval, ($0 — $1,000,000], is partitioned into 5 sub-intervals: ($0 — $200,000], ($200,000 — $400,000], ($400,000 — $600,000], ($600,000 — $800,000], and ($800,000 — $1,000,000].
  - The third interval, ($1,000,000 — $2,000,000], is partitioned into 5 sub-intervals: ($1,000,000 — $1,200,000], ($1,200,000 — $1,400,000], ($1,400,000 — $1,600,000], ($1,600,000 — $1,800,000], and ($1,800,000 — $2,000,000].
  - The last interval, ($2,000,000 — $5,000,000], is partitioned into 3 sub-intervals: ($2,000,000 — $3,000,000], ($3,000,000 — $4,000,000], and ($4,000,000 — $5,000,000].

  Similarly, the 3-4-5 rule can be carried on iteratively at deeper levels, as necessary. □

3.5.2 Concept hierarchy generation for categorical data

Categorical data are discrete data. Categorical 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*. There are several methods for the generation of concept hierarchies for categorical data.

1. **Specification of a partial ordering of attributes explicitly at the schema level by users or experts.**

   Concept hierarchies for categorical attributes or dimensions typically involve a group of attributes. A user or an expert can easily define a concept hierarchy by specifying a partial or total ordering of the attributes at the schema level. For example, a relational database or a dimension *location* of a data warehouse may contain the following group of attributes: *street, city, province_or_state*, and *country*. 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*.

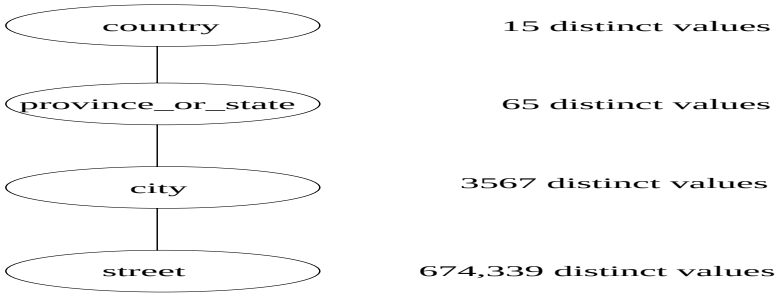


Figure 3.15: Automatic generation of a schema concept hierarchy based on the number of distinct attribute values.

2. **Specification of a portion of a hierarchy by explicit data grouping.**

   This is essentially the manual definition of a portion of concept hierarchy. In a large database, it is unrealistic to define an entire concept hierarchy by explicit value enumeration. However, it is realistic to 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, one may like to add some intermediate levels manually, such as defining explicitly, "{*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 simply group a set of attributes as a preferred dimension or hierarchy, but may omit stating their partial order explicitly. This may require the system to automatically generate the attribute ordering so as to construct a meaningful concept hierarchy. Without knowledge of data semantics, it is difficult to provide an ideal hierarchical ordering for an arbitrary set of attributes. However, an important observation is that since higher level concepts generally cover several subordinate lower level concepts, an attribute defining a high concept level will usually contain a smaller number of distinct values than an attribute defining a lower concept level. 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 level of the hierarchy. The lesser the number of distinct values an attribute has, the higher it is in the generated concept hierarchy. This heuristic rule works fine in many cases. Some local level swapping or adjustments may be performed by users or experts, when necessary, after examination of the generated hierarchy.

   Let's examine an example of this method.

   Example 3.6 Suppose a user selects a set of attributes, *street, country, province_or_state*, and *city*, for a dimension *location* from the database *AllElectronics*, but does not specify the hierarchical ordering among the attributes.

   The concept hierarchy for *location* can be generated automatically as follows. 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* (*65*), *city* (*3567*), and *street* (*674,339*). Second, generate the hierarchy from top down according to the sorted order, with the first attribute at the top-level and the last attribute at the bottom-level. The resulting hierarchy is shown in Figure 3.15. Finally, the user examines the generated hierarchy, and when necessary, modifies it to reflect desired semantic relationship among the attributes. In this example, it is obvious that there is no need to modify the generated hierarchy. □

   Note that this heristic rule cannot be pushed to the extreme since there are obvious cases which do not follow such a heuristic. 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 sloppy when defining a hierarchy, or may 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 relevant attributes in a hierarchy specification. For example, instead of including all the hierarchically relevant attributes for *location*, one may specify 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 over-ride this feature, as necessary.

Example 3.7 Suppose that a database system 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 may automatically drag all of the above five semantically-related attributes to form a hierarchy. The user may choose to drop any of these attributes, such as *number* and *street*, from the hierarchy, keeping *city* as the lowest conceptual level in the hierarchy. □

3.6 Summary

- **Data preparation** is an important issue for both data warehousing and data mining, as real-world data tends to be incomplete, noisy, and inconsistent. Data preparation includes data cleaning, data integration, data transformation, and data reduction.
- **Data cleaning** routines can be used to fill in missing values, smooth noisy data, identify outliers, and correct data inconsistencies.
- **Data integration** combines data from multiples sources to form a coherent data store. Metadata, correlation analysis, data conflict detection, and the resolution of semantic heterogeneity contribute towards smooth data integration.
- **Data transformation** routines conform the data into appropriate forms for mining. For example, attribute data may be **normalized** so as to fall between a small range, such as 0 to 1.0.
- **Data reduction** techniques such as data cube aggregation, dimension reduction, data compression, numerosity reduction, and discretization can be used to obtain a reduced representation of the data, while minimizing the loss of information content.
- **Concept hierarchies** organize the values of attributes or dimensions into gradual levels of abstraction. They are a form a discretization that is particularly useful in multilevel mining.
- **Automatic generation of concept hierarchies** for categoric data may be based on the number of distinct values of the attributes defining the hierarchy. For numeric data, techniques such as data segmentation by partition rules, histogram analysis, and clustering analysis can be used.
- Although several methods of data preparation have been developed, data preparation remains an active area of research.

Exercises

1. Data quality can be assessed in terms of accuracy, completeness, and consistency. Propose two other dimensions of data quality.
2. In real-world data, tuples with missing values for some attributes are a common occurrence. Describe various methods for handling this problem.
3. 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) Use smoothing by bin means to smooth the above 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?

4. Discuss issues to consider during data integration.

5. Using the data for *age* given in Question 3, answer the following:

(a) Use min-max normalization to transform the value 35 for *age* onto the range $[0, 1]$.

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

(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.

6. Use a flow-chart to illustrate the following procedures for attribute subset selection:

(a) step-wise forward selection.

(b) step-wise backward elimination

(c) a combination of forward selection and backward elimination.

7. Using the data for *age* given in Question 3:

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

(b) Sketch examples of each of the following sample techniques: SRSWOR, SRSWR, cluster sampling, stratified sampling.

8. Propose a concept hierarchy for the attribute *age* using the 3-4-5 partition rule.

9. Propose an algorithm, in pseudo-code or in your favorite programming language, for

(a) the automatic generation of a concept hierarchy for categorical 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 *equi-width* partitioning rule, and

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

Bibliographic Notes

Data preprocessing is discussed in a number of textbooks, including Pyle [28], Kennedy et al. [21], and Weiss and Indurkhya [37]. More specific references to individual preprocessing techniques are given below.

For discussion regarding data quality, see Ballou and Tayi [3], Redman [31], Wand and Wang [35], and Wang, Storey and Firth [36]. The handling of missing attribute values is discussed in Quinlan [29], Breiman et al. [5], and Friedman [11]. A method for the detection of outlier or "garbage" patterns in a handwritten character database is given in Guyon, Matic, and Vapnik [14]. Binning and data normalization are treated in several texts, including [28, 21, 37].

A good survey of data reduction techniques can be found in Barbará et al. [4]. For algorithms on data cubes and their precomputation, see [33, 16, 1, 38, 32]. Greedy methods for attribute subset selection (or *feature subset selection*) are described in several texts, such as Neter et al. [24], and John [18]. A combination forward selection and backward elimination method was proposed in Siedlecki and Sklansky [34]. For a description of wavelets for data compression, see Press et al. [27]. Daubechies transforms are described in Daubechies [6]. The book by Press et al. [27] also contains an introduction to singular value decomposition for principal components analysis.

An introduction to regression and log-linear models can be found in several textbooks, such as [17, 9, 20, 8, 24]. For log-linear models (known as *multiplicative models* in the computer science literature), see Pearl [25]. For a

general introduction to histograms, see [7, 4]. For extensions of single attribute histograms to multiple attributes, see Muralikrishna and DeWitt [23], and Poosala and Ioannidis [26]. Several references to clustering algorithms are given in Chapter 7 of this book, which is devoted to this topic. A survey of multidimensional indexing structures is in Gaede and Günther [12]. The use of multidimensional index trees for data aggregation is discussed in Aoki [2]. Index trees include R-trees (Guttman [13]), quad-trees (Finkel and Bentley [10]), and their variations. For discussion on sampling and data mining, see John and Langley [19], and Kivinen and Mannila [22].

Entropy and information gain are described in Quinlan [30]. Concept hierarchies, and their automatic generation from categorical data are described in Han and Fu [15].

Bibliography

[1] S. Agarwal, R. Agrawal, P. M. Deshpande, A. Gupta, J. F. Naughton, R. Ramakrishnan, and S. Sarawagi. On the computation of multidimensional aggregates. In *Proc. 1996 Int. Conf. Very Large Data Bases*, pages 506–521, Bombay, India, Sept. 1996.

[2] P. M. Aoki. Generalizing "search" in generalized search trees. In *Proc. 1998 Int. Conf. Data Engineering (ICDE'98)*, April 1998.

[3] D. P. Ballou and G. K. Tayi. Enhancing data quality in data warehouse environments. *Communications of ACM*, 42:73–78, 1999.

[4] D. Barbará et al. The new jersey data reduction report. *Bulletin of the Technical Committee on Data Engineering*, 20:3–45, December 1997.

[5] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth International Group, 1984.

[6] I. Daubechies. *Ten Lectures on Wavelets*. Capital City Press, Montpelier, Vermont, 1992.

[7] J. Devore and R. Peck. *Statistics: The Exploration and Analysis of Data*. New York: Duxbury Press, 1997.

[8] J. L. Devore. *Probability and Statistics for Engineering and the Science, 4th ed.* Duxbury Press, 1995.

[9] A. J. Dobson. *An Introduction to Generalized Linear Models*. Chapman and Hall, 1990.

[10] R. A. Finkel and J. L. Bentley. Quad-trees: A data structure for retrieval on composite keys. *ACTA Informatica*, 4:1–9, 1974.

[11] J. H. Friedman. A recursive partitioning decision rule for nonparametric classifiers. *IEEE Trans. on Comp.*, 26:404–408, 1977.

[12] V. Gaede and O. Günther. Multdimensional access methods. *ACM Comput. Surv.*, 30:170–231, 1998.

[13] A. Guttman. R-tree: A dynamic index structure for spatial searching. In *Proc. 1984 ACM-SIGMOD Int. Conf. Management of Data*, June 1984.

[14] I. Guyon, N. Matic, and V. Vapnik. Discoverying informative patterns and data cleaning. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 181–203. AAAI/MIT Press, 1996.

[15] J. Han and Y. Fu. Dynamic generation and refinement of concept hierarchies for knowledge discovery in databases. In *Proc. AAAI'94 Workshop Knowledge Discovery in Databases (KDD'94)*, pages 157–168, Seattle, WA, July 1994.

[16] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data*, pages 205–216, Montreal, Canada, June 1996.

[17] M. James. *Classification Algorithms*. John Wiley, 1985.

[18] G. H. John. *Enhancements to the Data Mining Process*. Ph.D. Thesis, Computer Science Dept., Stanford Univeristy, 1997.

[19] G. H. John and P. Langley. Static versus dynamic sampling for data mining. In *Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining (KDD'96)*, pages 367–370, Portland, OR, Aug. 1996.

[20] R. A. Johnson and D. W. Wickern. *Applied Multivariate Statistical Analysis, 3rd ed.* Prentice Hall, 1992.

[21] R. L Kennedy, Y. Lee, B. Van Roy, C. D. Reed, and R. P. Lippman. *Solving Data Mining Problems Through Pattern Recognition.* Upper Saddle River, NJ: Prentice Hall, 1998.

[22] J. Kivinen and H. Mannila. The power of sampling in knowledge discovery. In *Proc. 13th ACM Symp. Principles of Database Systems*, pages 77–85, Minneapolis, MN, May 1994.

[23] M. Muralikrishna and D. J. DeWitt. Equi-depth histograms for extimating selectivity factors for multi-dimensional queries. In *Proc. 1988 ACM-SIGMOD Int. Conf. Management of Data*, pages 28–36, Chicago, IL, June 1988.

[24] J. Neter, M. H. Kutner, C. J. Nachtsheim, and L. Wasserman. *Applied Linear Statistical Models, 4th ed.* Irwin: Chicago, 1996.

[25] J. Pearl. *Probabilistic Reasoning in Intelligent Systems.* Palo Alto, CA: Morgan Kauffman, 1988.

[26] V. Poosala and Y. Ioannidis. Selectivity estimationwithout the attribute value independence assumption. In *Proc. 23rd Int. Conf. on Very Large Data Bases*, pages 486–495, Athens, Greece, Aug. 1997.

[27] W. H. Press, S. A. Teukolosky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, Cambridge, MA, 1996.

[28] D. Pyle. *Data Preparation for Data Mining.* Morgan Kaufmann, 1999.

[29] J. R. Quinlan. Unknown attribute values in induction. In *Proc. 6th Int. Workshop on Machine Learning*, pages 164–168, Ithaca, NY, June 1989.

[30] J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.

[31] T. Redman. *Data Quality: Management and Technology.* Bantam Books, New York, 1992.

[32] K. Ross and D. Srivastava. Fast computation of sparse datacubes. In *Proc. 1997 Int. Conf. Very Large Data Bases*, pages 116–125, Athens, Greece, Aug. 1997.

[33] S. Sarawagi and M. Stonebraker. Efficient organization of large multidimensional arrays. In *Proc. 1994 Int. Conf. Data Engineering*, pages 328–336, Feb. 1994.

[34] W. Siedlecki and J. Sklansky. On automatic feature selection. *Int. J. of Pattern Recognition and Artificial Intelligence*, 2:197–220, 1988.

[35] Y. Wand and R. Wang. Anchoring data quality dimensions ontological foundations. *Communications of ACM*, 39:86–95, 1996.

[36] R. Wang, V. Storey, and C. Firth. A framework for analysis of data quality research. *IEEE Trans. Knowledge and Data Engineering*, 7:623–640, 1995.

[37] S. M. Weiss and N. Indurkhya. *Predictive Data Mining.* Morgan Kaufmann, 1998.

[38] Y. Zhao, P. M. Deshpande, and J. F. Naughton. An array-based algorithm for simultaneous multidimensional aggregates. In *Proc. 1997 ACM-SIGMOD Int. Conf. Management of Data*, pages 159–170, Tucson, Arizona, May 1997.

Contents

Chapter 4

Primitives for Data Mining

A popular misconception about data mining is to expect that data mining systems can *autonomously* dig out *all* of the valuable knowledge that is embedded in a given large database, without human intervention or guidance. Although it may at first sound appealing to have an autonomous data mining system, in practice, such systems will uncover an overwhelmingly large set of patterns. The entire set of generated patterns may easily surpass the size of the given database! To let a data mining system "run loose" in its discovery of patterns, without providing it with any indication regarding the portions of the database that the user wants to probe or the kinds of patterns the user would find interesting, is to let loose a data mining "monster". Most of the patterns discovered would be irrelevant to the analysis task of the user. Furthermore, many of the patterns found, though related to the analysis task, may be difficult to understand, or lack of validity, novelty, or utility — making them uninteresting. Thus, it is neither realistic nor desirable to generate, store, or present all of the patterns that could be discovered from a given database.

A more realistic scenario is to expect that users can communicate with the data mining system using a set of *data mining primitives* designed in order to facilitate efficient and fruitful knowledge discovery. Such primitives include the specification of the portions of the database or the set of data in which the user is interested (including the database attributes or data warehouse dimensions of interest), the kinds of knowledge to be mined, background knowledge useful in guiding the discovery process, interestingness measures for pattern evaluation, and how the discovered knowledge should be visualized. These primitives allow the user to *interactively* communicate with the data mining system during discovery in order to examine the findings from different angles or depths, and direct the mining process.

A data mining query language can be designed to incorporate these primitives, allowing users to flexibly interact with data mining systems. Having a data mining query language also provides a foundation on which friendly graphical user interfaces can be built. In this chapter, you will learn about the data mining primitives in detail, as well as study the design of a data mining query language based on these principles.

4.1 Data mining primitives: what defines a data mining task?

Each user will have a **data mining task** in mind, i.e., some form of data analysis that she would like to have performed. A data mining task can be specified in the form of a **data mining query**, which is input to the data mining system. A data mining query is defined in terms of the following primitives, as illustrated in Figure 4.1.

1. **task-relevant data**: This is the database portion to be investigated. For example, suppose that you are a manager of *AllElectronics* in charge of sales in the United States and Canada. In particular, you would like to study the buying trends of customers in Canada. Rather than mining on the entire database, you can specify that only the data relating to customer purchases in Canada need be retrieved, along with the related customer profile information. You can also specify attributes of interest to be considered in the mining process. These are referred to as **relevant attributes**[1]. For example, if you are interested only in studying possible

[1] If mining is to be performed on data from a multidimensional data cube, the user can specify relevant **dimensions**.

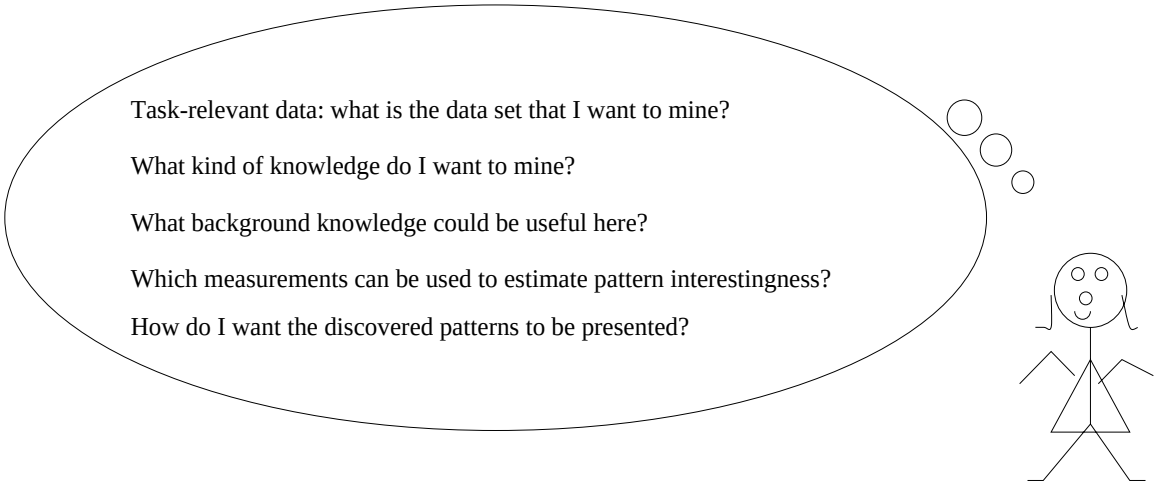


Figure 4.1: Defining a data mining task or query.

relationships between, say, the items purchased, and customer annual income and age, then the attributes *name* of the relation *item*, and *income* and *age* of the relation *customer* can be specified as the relevant attributes for mining. The portion of the database to be mined is called the **minable view**. A minable view can also be sorted and/or grouped according to one or a set of attributes or dimensions.

2. **the kinds of knowledge to be mined**: This specifies the *data mining functions* to be performed, such as characterization, discrimination, association, classification, clustering, or evolution analysis. For instance, if studying the buying habits of customers in Canada, you may choose to mine associations between customer profiles and the items that these customers like to buy.

3. **background knowledge**: Users can specify *background knowledge*, or knowledge about the domain to be mined. This knowledge is useful for guiding the knowledge discovery process, and for evaluating the patterns found. There are several kinds of background knowledge. In this chapter, we focus our discussion on a popular form of background knowledge known as *concept hierarchies*. Concept hierarchies are useful in that they allow data to be mined at multiple levels of abstraction. Other examples include user beliefs regarding relationships in the data. These can be used to evaluate the discovered patterns according to their degree of unexpectedness, where unexpected patterns are deemed interesting.

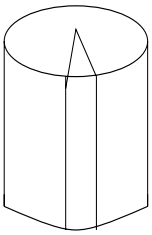
4. **interestingness measures**: These functions are used to separate uninteresting patterns from knowledge. They may be used to guide the mining process, or after discovery, to evaluate the discovered patterns. Different kinds of knowledge may have different interestingness measures. For example, interestingness measures for association rules include **support** (the percentage of task-relevant data tuples for which the rule pattern appears), and **confidence** (the strength of the implication of the rule). Rules whose support and confidence values are below user-specified thresholds are considered uninteresting.

5. **presentation and visualization of discovered patterns**: This refers to the form in which discovered patterns are to be displayed. Users can choose from different forms for knowledge presentation, such as rules, tables, charts, graphs, decision trees, and cubes.

Below, we examine each of these primitives in greater detail. The specification of these primitives is summarized in Figure 4.2.

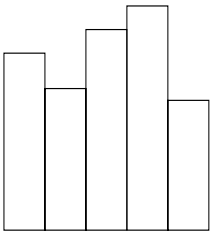
4.1.1 Task-relevant data

The first primitive is the specification of the data on which mining is to be performed. Typically, a user is interested in only a subset of the database. It is impractical to indiscriminately mine the entire database, particularly since the number of patterns generated could be exponential with respect to the database size. Furthermore, many of these patterns found would be irrelevant to the interests of the user.



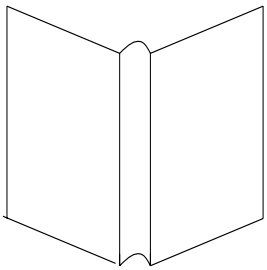
Task-relevant data

- database or data warehouse name
- database tables or data warehouse cubes
- conditions for data selection
- relevant attributes or dimensions
- data grouping criteria



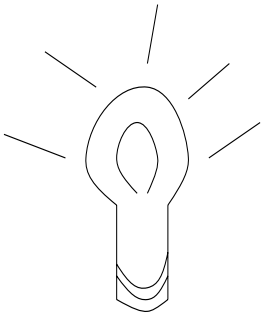
Knowledge type to be mined

- characterization
- discrimination
- association
- classification/prediction
- clustering



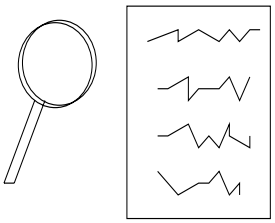
Background knowledge

- concept hierarchies
- user beliefs about relationships in the data



Pattern interestingness measurements

- simplicity
- certainty (e.g., confidence)
- utility (e.g., support)
- novelty



Visualization of discovered patterns

- rules, tables, reports, charts, graphs, decisison trees, and cubes
- drill-down and roll-up

Figure 4.2: Primitives for specifying a data mining task.

In a relational database, the set of task-relevant data can be collected via a relational query involving operations like selection, projection, join, and aggregation. This retrieval of data can be thought of as a "subtask" of the data mining task. The data collection process results in a new data relation, called the **initial data relation**. The initial data relation can be ordered or grouped according to the conditions specified in the query. The data may be cleaned or transformed (e.g., aggregated on certain attributes) prior to applying data mining analysis. The initial relation may or may not correspond to a physical relation in the database. Since virtual relations are called **views** in the field of databases, the set of task-relevant data for data mining is called a **minable view**.

Example 4.1 If the data mining task is to study associations between items frequently purchased at *AllElectronics* by customers in Canada, the task-relevant data can be specified by providing the following information:

- the name of the *database* or *data warehouse* to be used (e.g., *AllElectronics_db*),
- the names of the *tables* or *data cubes* containing the relevant data (e.g., *item*, *customer*, *purchases*, and *items_sold*),
- *conditions* for selecting the relevant data (e.g., retrieve data pertaining to purchases made in Canada for the current year),
- the *relevant attributes or dimensions* (e.g., *name* and *price* from the *item* table, and *income* and *age* from the *customer* table).

In addition, the user may specify that the data retrieved be grouped by certain attributes, such as "**group by** *date*". Given this information, an SQL query can be used to retrieve the task-relevant data. □

In a data warehouse, data are typically stored in a multidimensional database, known as a **data cube**, which can be implemented using a multidimensional array structure, a relational structure, or a combination of both, as discussed in Chapter 2. The set of task-relevant data can be specified by condition-based data filtering, *slicing* (extracting data for a given attribute value, or "slice"), or *dicing* (extracting the intersection of several slices) of the data cube.

Notice that in a data mining query, the conditions provided for data selection can be at a level that is conceptually higher than the data in the database or data warehouse. For example, a user may specify a selection on items at *AllElectronics* using the concept "*type = home entertainment*", even though individual items in the database may not be stored according to type, but rather, at a lower conceptual, such as "*TV*", "*CD player*", or "*VCR*". A concept hierarchy on item which specifies that "*home entertainment*" is at a higher concept level, composed of the lower level concepts {*"TV", "CD player", "VCR"*} can be used in the collection of the task-relevant data.

The set of relevant attributes specified may involve other attributes which were not explicitly mentioned, but which should be included because they are implied by the concept hierarchy or dimensions involved in the set of relevant attributes specified. For example, a query-relevant set of attributes may contain *city*. This attribute, however, may be part of other concept hierarchies such as the concept hierarchy *street* < *city* < *province_or_state* < *country* for the dimension *location*. In this case, the attributes *street*, *province_or_state*, and *country* should also be included in the set of relevant attributes since they represent lower or higher level abstractions of *city*. This facilitates the mining of knowledge at multiple levels of abstraction by specialization (drill-down) and generalization (roll-up).

Specification of the relevant attributes or dimensions can be a difficult task for users. A user may have only a rough idea of what the interesting attributes for exploration might be. Furthermore, when specifying the data to be mined, the user may overlook additional relevant data having strong semantic links to them. For example, the sales of certain items may be closely linked to particular events such as Christmas or Halloween, or to particular groups of people, yet these factors may not be included in the general data analysis request. For such cases, mechanisms can be used which help give a more precise specification of the task-relevant data. These include functions to evaluate and rank attributes according to their relevancy with respect to the operation specified. In addition, techniques that search for attributes with strong semantic ties can be used to enhance the initial dataset specified by the user.

4.1.2 The kind of knowledge to be mined

It is important to specify the kind of knowledge to be mined, as this determines the data mining function to be performed. The kinds of knowledge include concept description (characterization and discrimination), association, classification, prediction, clustering, and evolution analysis.

In addition to specifying the kind of knowledge to be mined for a given data mining task, the user can be more specific and provide pattern templates that all discovered patterns must match. These templates, or **metapatterns** (also called **metarules** or **metaqueries**), can be used to guide the discovery process. The use of metapatterns is illustrated in the following example.

Example 4.2 A user studying the buying habits of *AllElectronics* customers may choose to mine *association rules* of the form

$$P(X : customer, W) \wedge Q(X, Y) \Rightarrow buys(X, Z)$$

where X is a key of the *customer* relation, P and Q are **predicate variables** which can be instantiated to the relevant attributes or dimensions specified as part of the task-relevant data, and W, Y, and Z are **object variables** which can take on the values of their respective predicates for customers X.

The search for association rules is confined to those matching the given metarule, such as

$$age(X, \text{“30\_39”}) \wedge income(X, \text{“40\_50}K\text{”}) \Rightarrow buys(X, \text{“}VCR\text{”}) \quad [2.2\%, 60\%] \quad (4.1)$$

and

$$occupation(X, \text{“}student\text{”}) \wedge age(X, \text{“20\_29”}) \Rightarrow buys(X, \text{“}computer\text{”}) \quad [1.4\%, 70\%]. \quad (4.2)$$

The former rule states that customers in their thirties, with an annual income of between 40K and 50K, are likely (with 60% confidence) to purchase a VCR, and such cases represent about 2.2% of the total number of transactions. The latter rule states that customers who are students and in their twenties are likely (with 70% confidence) to purchase a computer, and such cases represent about 1.4% of the total number of transactions. □

4.1.3 Background knowledge: concept hierarchies

Background knowledge is information about the domain to be mined that can be useful in the discovery process. In this section, we focus our attention on a simple yet powerful form of background knowledge known as *concept hierarchies*. Concept hierarchies allow the discovery of knowledge at multiple levels of abstraction.

As described in Chapter 2, a **concept hierarchy** defines a sequence of mappings from a set of low level concepts to higher level, more general concepts. A concept hierarchy for the dimension *location* is shown in Figure 4.3, mapping low level concepts (i.e., cities) to more general concepts (i.e., countries).

Notice that this concept hierarchy is represented as a set of **nodes** organized in a tree, where each node, in itself, represents a concept. A special node, all, is reserved for the root of the tree. It denotes the most generalized value of the given dimension. If not explicitly shown, it is implied. This concept hierarchy consists of four **levels**. By convention, levels within a concept hierarchy are numbered from top to bottom, starting with level 0 for the all node. In our example, level 1 represents the concept *country*, while levels 2 and 3 respectively represent the concepts *province_or_state* and *city*. The leaves of the hierarchy correspond to the dimension's raw data values (**primitive level data**). These are the most specific values, or concepts, of the given attribute or dimension. Although a concept hierarchy often defines a taxonomy represented in the shape of a tree, it may also be in the form of a general lattice or partial order.

Concept hierarchies are a useful form of background knowledge in that they allow raw data to be handled at higher, generalized levels of abstraction. Generalization of the data, or **rolling up** is achieved by replacing primitive level data (such as city names for *location*, or numerical values for *age*) by higher level concepts (such as continents for *location*, or ranges like “20-39”, “40-59”, “60+” for *age*). This allows the user to view the data at *more meaningful* and explicit abstractions, and makes the discovered patterns easier to understand. Generalization has an added advantage of compressing the data. Mining on a compressed data set will require fewer input/output operations and be more efficient than mining on a larger, uncompressed data set.

If the resulting data appear overgeneralized, concept hierarchies also allow specialization, or **drilling down**, whereby concept values are replaced by lower level concepts. By rolling up and drilling down, users can view the data from different perspectives, gaining further insight into hidden data relationships.

Concept hierarchies can be provided by system users, domain experts, or knowledge engineers. The mappings are typically data- or application-specific. Concept hierarchies can often be automatically discovered or dynamically refined based on statistical analysis of the data distribution. The automatic generation of concept hierarchies is discussed in detail in Chapter 3.

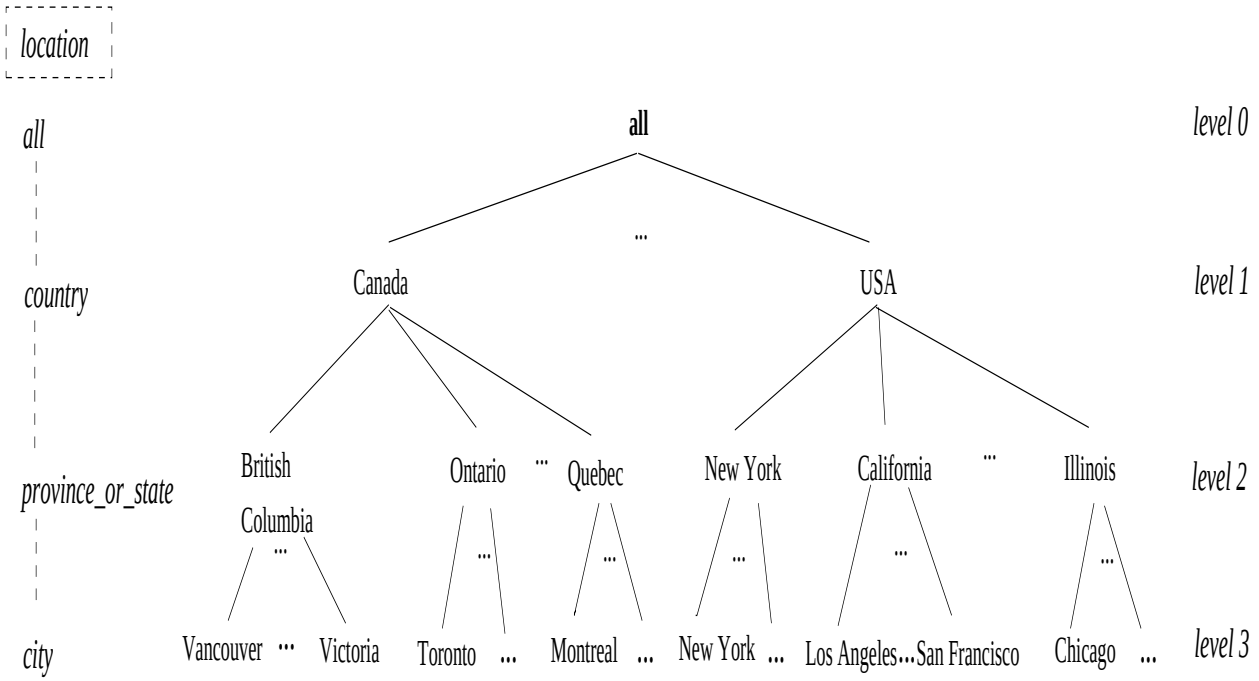


Figure 4.3: A concept hierarchy for the dimension *location*.

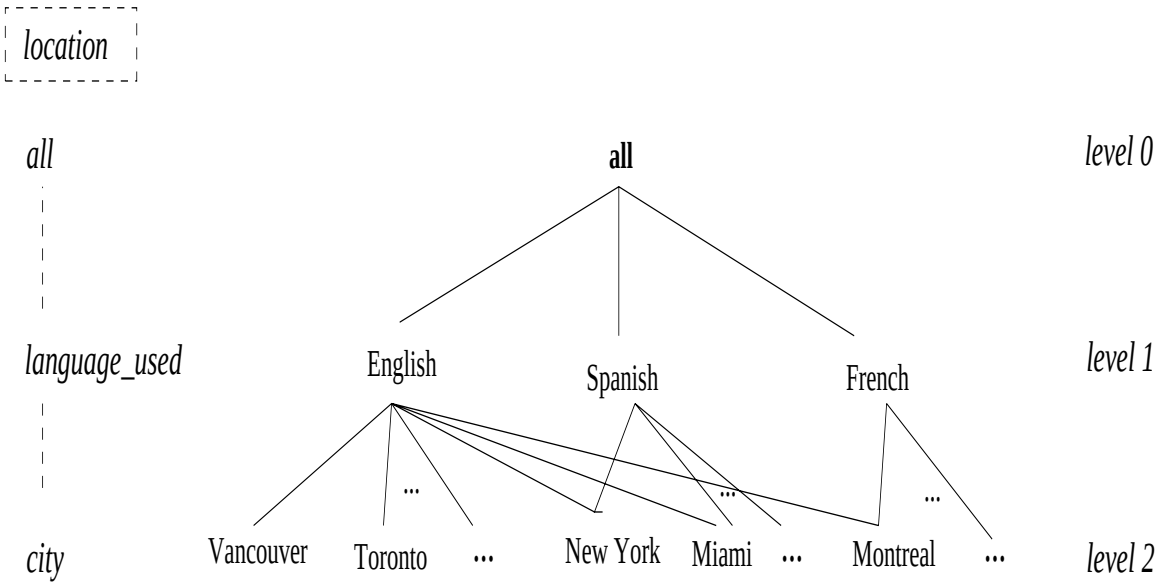


Figure 4.4: Another concept hierarchy for the dimension *location*, based on language.

There may be more than one concept hierarchy for a given attribute or dimension, based on different user viewpoints. Suppose, for instance, that a regional sales manager of *AllElectronics* is interested in studying the buying habits of customers at different locations. The concept hierarchy for *location* of Figure 4.3 should be useful for such a mining task. Suppose that a marketing manager must devise advertising campaigns for *AllElectronics*. This user may prefer to see *location* organized with respect to linguistic lines (e.g., including English for Vancouver, Montreal and New York; French for Montreal; Spanish for New York and Miami; and so on) in order to facilitate the distribution of commercial ads. This alternative hierarchy for *location* is illustrated in Figure 4.4. Note that this concept hierarchy forms a lattice, where the node "New York" has two parent nodes, namely "English" and "Spanish".

There are four major types of concept hierarchies. Chapter 2 introduced the most common types — *schema hierarchies* and *set-grouping hierarchies*, which we review here. In addition, we also study *operation-derived hierarchies* and *rule-based hierarchies*.

1. A **schema hierarchy** (or more rigorously, a *schema-defined hierarchy*) is a total or partial order among attributes in the database schema. Schema hierarchies may formally express existing semantic relationships between attributes. Typically, a schema hierarchy specifies a data warehouse dimension.

   Example 4.3 Given the schema of a relation for *address* containing the attributes *street, city, province_or_state*, and *country*, we can define a *location* schema hierarchy by the following total order:

   $$street < city < province\_or\_state < country$$

   This means that *street* is at a conceptually lower level than *city*, which is lower than *province_or_state*, which is conceptually lower than *country*. A schema hierarchy provides metadata information, i.e., data about the data. Its specification in terms of a total or partial order among attributes is more concise than an equivalent definition that lists all instances of streets, provinces or states, and countries.

   Recall that when specifying the task-relevant data, the user specifies relevant attributes for exploration. If a user had specified only one attribute pertaining to location, say, *city*, other attributes pertaining to any schema hierarchy containing *city* may automatically be considered relevant attributes as well. For instance, the attributes *street, province_or_state*, and *country* may also be automatically included for exploration. □

2. A **set-grouping hierarchy** organizes values for a given attribute or dimension into groups of constants or range values. A total or partial order can be defined among groups. Set-grouping hierarchies can be used to refine or enrich schema-defined hierarchies, when the two types of hierarchies are combined. They are typically used for defining small sets of object relationships.

   Example 4.4 A set-grouping hierarchy for the attribute *age* can be specified in terms of ranges, as in the following.

   $$\{20 - 39\} \subset young$$
   $$\{40 - 59\} \subset middle\_aged$$
   $$\{60 - 89\} \subset senior$$
   $$\{young, middle\_aged, senior\} \subset \mathsf{all}(age)$$

   Notice that similar range specifications can also be generated automatically, as detailed in Chapter 3. □

   Example 4.5 A set-grouping hierarchy may form a portion of a schema hierarchy, and vice versa. For example, consider the concept hierarchy for *location* in Figure 4.3, defined as $city < province\_or\_state < country$. Suppose that possible constant values for *country* include "*Canada*", "*USA*", "*Germany*", "*England*", and "*Brazil*". Set-grouping may be used to refine this hierarchy by adding an additional level above *country*, such as *continent*, which groups the country values accordingly. □

3. **Operation-derived hierarchies** are based on operations specified by users, experts, or the data mining system. Operations can include the decoding of information-encoded strings, information extraction from complex data objects, and data clustering.

Example 4.6 An e-mail address or a URL of the WWW may contain hierarchy information relating departments, universities (or companies), and countries. Decoding operations can be defined to extract such information in order to form concept hierarchies.

For example, the e-mail address "dmbook@cs.sfu.ca" gives the partial order, "*login-name* < *department* < *university* < *country*", forming a concept hierarchy for e-mail addresses. Similarly, the URL address "http://www.cs.sfu.ca/research/DB/DBMiner" can be decoded so as to provide a partial order which forms the base of a concept hierarchy for URLs. □

Example 4.7 Operations can be defined to extract information from complex data objects. For example, the string "Ph.D. in Computer Science, UCLA, 1995" is a complex object representing a university degree. This string contains rich information about the type of academic degree, major, university, and the year that the degree was awarded. Operations can be defined to extract such information, forming concept hierarchies. □

Alternatively, mathematical and statistical operations, such as data clustering and data distribution analysis algorithms, can be used to form concept hierarchies, as discussed in Section 3.5

4. **A rule-based hierarchy** occurs when either a whole concept hierarchy or a portion of it is defined by a set of rules, and is evaluated dynamically based on the current database data and the rule definition.

Example 4.8 The following rules may be used to categorize *AllElectronics* items as *low_profit_margin* items, *medium_profit_margin* items, and *high_profit_margin* items, where the profit margin of an item X is defined as the difference between the retail price and actual cost of X. Items having a profit margin of less than \$50 may be defined as *low_profit_margin* items, items earning a profit between \$50 and \$250 may be defined as *medium_profit_margin* items, and items earning a profit of more than \$250 may be defined as *high_profit_margin* items.

$$\begin{aligned} low\_profit\_margin(X) &\Leftarrow price(X, P1) \wedge cost(X, P2) \wedge ((P1 - P2) < \$50) \\ medium\_profit\_margin(X) &\Leftarrow price(X, P1) \wedge cost(X, P2) \wedge ((P1 - P2) > \$50) \wedge ((P1 - P2) \leq \$250) \\ high\_profit\_margin(X) &\Leftarrow price(X, P1) \wedge cost(X, P2) \wedge ((P1 - P2) > \$250) \end{aligned}$$

□

The use of concept hierarchies for data mining is described in the remaining chapters of this book.

4.1.4 Interestingness measures

Although specification of the task-relevant data and of the kind of knowledge to be mined (e.g, characterization, association, etc.) may substantially reduce the number of patterns generated, a data mining process may still generate a large number of patterns. Typically, only a small fraction of these patterns will actually be of interest to the given user. Thus, users need to further confine the number of uninteresting patterns returned by the process. This can be achieved by specifying interestingness measures which estimate the simplicity, certainty, utility, and novelty of patterns.

In this section, we study some objective measures of pattern interestingness. Such objective measures are based on the structure of patterns and the statistics underlying them. In general, each measure is associated with a *threshold* that can be controlled by the user. Rules that do not meet the threshold are considered uninteresting, and hence are not presented to the user as knowledge.

- **Simplicity.** A factor contributing to the interestingness of a pattern is the pattern's overall simplicity for human comprehension. Objective measures of pattern simplicity can be viewed as functions of the pattern structure, defined in terms of the pattern size in bits, or the number of attributes or operators appearing in the pattern. For example, the more complex the structure of a rule is, the more difficult it is to interpret, and hence, the less interesting it is likely to be.

  Rule length, for instance, is a simplicity measure. For rules expressed in conjunctive normal form (i.e., as a set of conjunctive predicates), rule length is typically defined as the number of conjuncts in the rule.

Association, discrimination, or classification rules whose lengths exceed a user-defined threshold are considered uninteresting. For patterns expressed as decision trees, simplicity may be a function of the number of tree leaves or tree nodes.

- **Certainty.** Each discovered pattern should have a measure of certainty associated with it which assesses the validity or "trustworthiness" of the pattern. A certainty measure for association rules of the form "$A \Rightarrow B$," is **confidence**. Given a set of task-relevant data tuples (or transactions in a transaction database) the confidence of "$A \Rightarrow B$" is defined as:

$$\text{Confidence}(A \Rightarrow B) = P(B|A) = \frac{\#\_tuples\_containing\_both\_A\_and\_B}{\#\_tuples\_containing\_A}. \tag{4.3}$$

Example 4.9 Suppose that the set of task-relevant data consists of transactions from the computer department of *AllElectronics*. A confidence of 85% for the association rule

$$buys(X, \text{"}computer\text{"}) \Rightarrow buys(X, \text{"}software\text{"}) \tag{4.4}$$

means that 85% of all customers who purchased a computer also bought software. □

A confidence value of 100%, or 1, indicates that the rule is always correct on the data analyzed. Such rules are called **exact**.

For classification rules, confidence is referred to as **reliability** or **accuracy**. Classification rules propose a model for distinguishing objects, or tuples, of a target class (say, *bigSpenders*) from objects of contrasting classes (say, *budgetSpenders*). A low reliability value indicates that the rule in question incorrectly classifies a large number of contrasting class objects as target class objects. Rule reliability is also known as **rule strength, rule quality, certainty factor**, and **discriminating weight**.

- **Utility.** The potential usefulness of a pattern is a factor defining its interestingness. It can be estimated by a utility function, such as support. The **support** of an association pattern refers to the percentage of task-relevant data tuples (or transactions) for which the pattern is true. For association rules of the form "$A \Rightarrow B$", it is defined as

$$\text{Support}(A \Rightarrow B) = P(A \cup B) = \frac{\#\_tuples\_containing\_both\_A\_and\_B}{total\_\#\_of\_tuples}. \tag{4.5}$$

Example 4.10 Suppose that the set of task-relevant data consists of transactions from the computer department of *AllElectronics*. A support of 30% for the association rule (4.4) means that 30% of all customers in the computer department purchased both a computer and software. □

Association rules that satisfy both a user-specified *minimum confidence threshold* and user-specified *minimum support threshold* are referred to as **strong association rules**, and are considered interesting. Rules with low support likely represent noise, or rare or exceptional cases.

The numerator of the support equation is also known as the rule **count**. Quite often, this number is displayed instead of support. Support can easily be derived from it.

Characteristic and discriminant descriptions are, in essence, generalized tuples. Any generalized tuple representing less than Y% of the total number of task-relevant tuples is considered noise. Such tuples are not displayed to the user. The value of Y is referred to as the **noise threshold**.

- **Novelty.** Novel patterns are those that contribute new information or increased performance to the given pattern set. For example, a data exception may be considered novel in that it differs from that expected based on a statistical model or user beliefs. Another strategy for detecting novelty is to remove redundant patterns. If a discovered rule can be implied by another rule that is already in the knowledge base or in the derived rule set, then either rule should be re-examined in order to remove the potential redundancy.

Mining with concept hierarchies can result in a large number of redundant rules. For example, suppose that the following association rules were mined from the *AllElectronics* database, using the concept hierarchy in Figure 4.3 for *location*:

$$location(X, \text{“}Canada\text{”}) \Rightarrow buys(X, \text{“}SONY\_TV\text{”}) \quad [8\%, 70\%] \quad (4.6)$$

$$location(X, \text{“}Montreal\text{”}) \Rightarrow buys(X, \text{“}SONY\_TV\text{”}) \quad [2\%, 71\%] \quad (4.7)$$

Suppose that Rule (4.6) has 8% support and 70% confidence. One may expect Rule (4.7) to have a confidence of around 70% as well, since all the tuples representing data objects for Montreal are also data objects for Canada. Rule (4.6) is more general than Rule (4.7), and therefore, we would expect the former rule to occur more frequently than the latter. Consequently, the two rules should not have the same support. Suppose that about one quarter of all sales in Canada comes from Montreal. We would then expect the support of the rule involving Montreal to be one quarter of the support of the rule involving Canada. In other words, we expect the support of Rule (4.7) to be $8\% \times \frac{1}{4} = 2\%$. If the actual confidence and support of Rule (4.7) are as expected, then the rule is considered redundant since it does not offer any additional information and is less general than Rule (4.6). These ideas are further discussed in Chapter 6 on association rule mining.

The above example also illustrates that when mining knowledge at multiple levels, it is reasonable to have different support and confidence thresholds, depending on the degree of granularity of the knowledge in the discovered pattern. For instance, since patterns are likely to be more scattered at lower levels than at higher ones, we may set the minimum support threshold for rules containing low level concepts to be lower than that for rules containing higher level concepts.

Data mining systems should allow users to flexibly and interactively specify, test, and modify interestingness measures and their respective thresholds. There are many other objective measures, apart from the basic ones studied above. Subjective measures exist as well, which consider user beliefs regarding relationships in the data, in addition to objective statistical measures. Interestingness measures are discussed in greater detail throughout the book, with respect to the mining of characteristic, association, and classification rules, and deviation patterns.

4.1.5 Presentation and visualization of discovered patterns

For data mining to be effective, data mining systems should be able to display the discovered patterns in multiple forms, such as rules, tables, crosstabs, pie or bar charts, decision trees, cubes, or other visual representations (Figure 4.5). Allowing the visualization of discovered patterns in various forms can help users with different backgrounds to identify patterns of interest and to interact or guide the system in further discovery. A user should be able to specify the kinds of presentation to be used for displaying the discovered patterns.

The use of concept hierarchies plays an important role in aiding the user to visualize the discovered patterns. Mining with concept hierarchies allows the representation of discovered knowledge in high level concepts, which may be more understandable to users than rules expressed in terms of primitive (i.e., raw) data, such as functional or multivalued dependency rules, or integrity constraints. Furthermore, data mining systems should employ concept hierarchies to implement drill-down and roll-up operations, so that users may inspect discovered patterns at multiple levels of abstraction. In addition, pivoting (or rotating), slicing, and dicing operations aid the user in viewing generalized data and knowledge from different perspectives. These operations were discussed in detail in Chapter 2. A data mining system should provide such interactive operations for any dimension, as well as for individual values of each dimension.

Some representation forms may be better suited than others for particular kinds of knowledge. For example, generalized relations and their corresponding crosstabs (cross-tabulations) or pie/bar charts are good for presenting characteristic descriptions, whereas decision trees are a common choice for classification. Interestingness measures should be displayed for each discovered pattern, in order to help users identify those patterns representing useful knowledge. These include confidence, support, and count, as described in Section 4.1.4.

4.2 A data mining query language

Why is it important to have a data mining query language? Well, recall that a desired feature of data mining systems is the ability to *support ad-hoc and interactive data mining* in order to facilitate flexible and effective knowledge discovery. Data mining query languages can be designed to support such a feature.

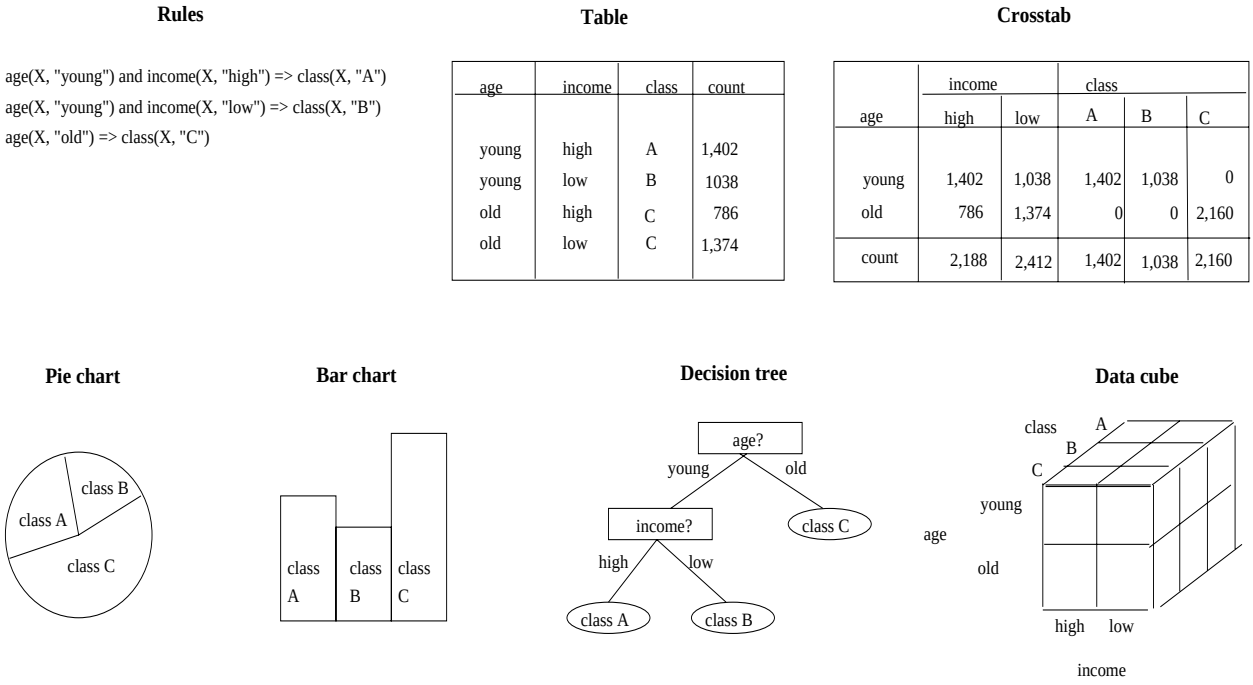


Figure 4.5: Various forms of presenting and visualizing the discovered patterns.

The importance of the design of a good data mining query language can also be seen from observing the history of relational database systems. Relational database systems have dominated the database market for decades. The standardization of relational query languages, which occurred at the early stages of relational database development, is widely credited for the success of the relational database field. Although each commercial relational database system has its own graphical user interace, the underlying core of each interface is a standardized relational query language. The standardization of relational query languages provided a foundation on which relational systems were developed, and evolved. It facilitated information exchange and technology transfer, and promoted commercialization and wide acceptance of relational database technology. The recent standardization activities in database systems, such as work relating to SQL-3, OMG, and ODMG, further illustrate the importance of having a standard database language for success in the development and commercialization of database systems. Hence, having a good query language for data mining may help standardize the development of platforms for data mining systems.

Designing a comprehensive data mining language is challenging because data mining covers a wide spectrum of tasks, from data characterization to mining association rules, data classification, and evolution analysis. Each task has different requirements. The design of an effective data mining query language requires a deep understanding of the power, limitation, and underlying mechanisms of the various kinds of data mining tasks.

How would you design a data mining query language? Earlier in this chapter, we looked at primitives for defining a data mining task in the form of a data mining query. The primitives specify:

- the set of task-relevant data to be mined,
- the kind of knowledge to be mined,
- the background knowledge to be used in the discovery process,
- the interestingness measures and thresholds for pattern evaluation, and
- the expected representation for visualizing the discovered patterns.

Based on these primitives, we design a query language for data mining called DMQL which stands for **D**ata **M**ining **Q**uery **L**anguage. DMQL allows the ad-hoc mining of several kinds of knowledge from relational databases and data warehouses at multiple levels of abstraction [2].

[2] DMQL syntax for defining data warehouses and data marts is given in Chapter 2.

```
⟨DMQL⟩ ::= ⟨DMQL_Statement⟩; {⟨DMQL_Statement⟩}
⟨DMQL_Statement⟩ ::= ⟨Data_Mining_Statment⟩
    | ⟨Concept_Hierarchy_Definition_Statement⟩
    | ⟨Visualization_and_Presentation⟩
⟨Data_Mining_Statement⟩ ::=
    use database ⟨database_name⟩ | use data warehouse ⟨data_warehouse_name⟩
    {use hierarchy ⟨hierarchy_name⟩ for ⟨attribute_or_dimension⟩}
    ⟨Mine_Knowledge_Specification⟩
    in relevance to ⟨attribute_or_dimension_list⟩
    from ⟨relation(s)/cube⟩
    [where ⟨condition⟩]
    [order by ⟨order_list⟩]
    [group by ⟨grouping_list⟩]
    [having ⟨condition⟩]
    {with [⟨interest_measure_name⟩] threshold = ⟨threshold_value⟩ [for ⟨attribute(s)⟩]}
⟨Mine_Knowledge_Specification⟩::= ⟨Mine_Char⟩ | ⟨Mine_Discr⟩ | ⟨Mine_Assoc⟩ | ⟨Mine_Class⟩ | ⟨Mine_Pred⟩
⟨Mine_Char⟩ ::= mine characteristics [as ⟨pattern_name⟩]
    analyze ⟨measure(s)⟩
⟨Mine_Discr⟩ ::= mine comparison [as ⟨pattern_name⟩]
    for ⟨target_class⟩ where ⟨target_condition⟩
    {versus ⟨contrast_class_i⟩ where ⟨contrast_condition_i⟩}
    analyze ⟨measure(s)⟩
⟨Mine_Assoc⟩ ::= mine associations [as ⟨pattern_name⟩]
    [matching ⟨metapattern⟩]
⟨Mine_Class⟩ ::= mine classification [as ⟨pattern_name⟩]
    analyze ⟨classifying_attribute_or_dimension⟩
⟨Mine_Pred⟩ ::= mine prediction [as ⟨pattern_name⟩]
    analyze ⟨prediction_attribute_or_dimension⟩
    {set {⟨attribute_or_dimension_i⟩= ⟨value_i⟩}}
⟨Concept_Hierarchy_Defintion_Statement⟩ ::=
    define hierarchy ⟨hierarchy_name⟩
    [for ⟨attribute_or_dimension⟩]
    on ⟨relation_or_cube_or_hierarchy⟩
    as ⟨hierarchy_description⟩
    [where ⟨condition⟩]
⟨Visualization_and_Presentation⟩ ::=
    display as ⟨result_form⟩
    |  roll up on ⟨attribute_or_dimension ⟩
    |  drill down on ⟨attribute_or_dimension⟩
    |  add ⟨attribute_or_dimension⟩
    |  drop ⟨attribute_or_dimension⟩
```

Figure 4.6: Top-level syntax of a data mining query language, DMQL.

The language adopts an SQL-like syntax, so that it can easily be integrated with the relational query language, SQL. The syntax of DMQL is defined in an extended BNF grammar, where "[ ]" represents 0 or one occurrence, "{ }" represents 0 or more occurrences, and words in sans serif font represent keywords.

In Sections 4.2.1 to 4.2.5, we develop DMQL syntax for each of the data mining primitives. In Section 4.2.6, we show an example data mining query, specified in the proposed syntax. A top-level summary of the language is shown in Figure 4.6.

4.2.1 Syntax for task-relevant data specification

The first step in defining a data mining task is the specification of the task-relevant data, i.e., the data on which mining is to be performed. This involves specifying the database and tables or data warehouse containing the relevant data, conditions for selecting the relevant data, the relevant attributes or dimensions for exploration, and instructions regarding the ordering or grouping of the data retrieved. DMQL provides clauses for the specification of such information, as follows.

- use database ⟨database_name⟩, or use data warehouse ⟨data_warehouse_name⟩: The use clause directs the mining task to the database or data warehouse specified.
- from ⟨relation(s)/cube(s)⟩ [where ⟨condition⟩]: The from and where clauses respectively specify the database tables or data cubes involved, and the conditions defining the data to be retrieved.
- in relevance to ⟨att_or_dim_list⟩: This clause lists the attributes or dimensions for exploration.
- order by ⟨order_list⟩: The order by clause specifies the sorting order of the task-relevant data.
- group by ⟨grouping_list⟩: The group by clause specifies criteria for grouping the data.
- having ⟨condition⟩: The having clause specifies the condition by which groups of data are considered relevant.

These clauses form an SQL query to collect the task-relevant data.

Example 4.11 This example shows how to use DMQL to specify the task-relevant data described in Example 4.1 for the mining of associations between items frequently purchased at *AllElectronics* by Canadian customers, with respect to customer *income* and *age*. In addition, the user specifies that she would like the data to be grouped by date. The data are retrieved from a relational database.

```
use database AllElectronics_db
in relevance to I.name, I.price, C.income, C.age
from customer C, item I, purchases P, items_sold S
where I.item_ID = S.item_ID and S.trans_ID = P.trans_ID and P.cust_ID = C.cust_ID
        and C.address = "Canada"
group by P.date
```

□

4.2.2 Syntax for specifying the kind of knowledge to be mined

The ⟨Mine_Knowledge_Specification⟩ statement is used to specify the kind of knowledge to be mined. In other words, it indicates the data mining functionality to be performed. Its syntax is defined below for characterization, discrimination, association, classification, and prediction.

1. **Characterization**.

```
⟨Mine_Knowledge_Specification⟩ ::=
            mine characteristics [as ⟨pattern_name⟩]
            analyze ⟨measure(s)⟩
```

This specifies that characteristic descriptions are to be mined. The analyze clause, when used for characterization, specifies aggregate measures, such as `count`, `sum`, or `count%` (percentage count, i.e., the percentage of tuples in the relevant data set with the specified characteristics). These measures are to be computed for each data characteristic found.

Example 4.12 The following specifies that the kind of knowledge to be mined is a characteristic description describing customer purchasing habits. For each characteristic, the percentage of task-relevant tuples satisfying that characteristic is to be displayed.

```
mine characteristics as customerPurchasing
analyze count%
```

□

2. **Discrimination**.

⟨Mine_Knowledge_Specification⟩ ::=
mine comparison [as ⟨pattern_name⟩]
for ⟨target_class⟩ where ⟨target_condition⟩
{versus ⟨contrast_class_*i*⟩ where ⟨contrast_condition_*i*⟩}
analyze ⟨measure(s)⟩

This specifies that discriminant descriptions are to be mined. These descriptions compare a given target class of objects with one or more other contrasting classes. Hence, this kind of knowledge is referred to as a comparison. As for characterization, the analyze clause specifies aggregate measures, such as `count`, `sum`, or `count%`, to be computed and displayed for each description.

Example 4.13 The user may define categories of customers, and then mine descriptions of each category. For instance, a user may define *bigSpenders* as customers who purchase items that cost \$100 or more on average, and *budgetSpenders* as customers who purchase items at less than \$100 on average. The mining of discriminant descriptions for customers from each of these categories can be specified in DMQL as shown below, where I refers to the *item* relation. The count of task-relevant tuples satisfying each description is to be displayed.

```
mine comparison as purchaseGroups
for bigSpenders where avg(I.price) ≥ $100
versus budgetSpenders where avg(I.price) < $100
analyze count
```

□

3. **Association.**

⟨Mine_Knowledge_Specification⟩ ::=
mine associations [as ⟨pattern_name⟩]
[matching ⟨metapattern⟩]

This specifies the mining of patterns of association. When specifying association mining, the user has the option of providing templates (also known as *metapatterns* or *metarules*) with the matching clause. The metapatterns can be used to focus the discovery towards the patterns that match the given metapatterns, thereby enforcing additional syntactic constraints for the mining task. In addition to providing syntactic constraints, the metapatterns represent data hunches or hypotheses that the user finds interesting for investigation. Mining with the use of metapatterns, or **metarule-guided mining**, allows additional flexibility for ad-hoc rule mining. While metapatterns may be used in the mining of other forms of knowledge, they are most useful for association mining due to the vast number of potentially generated associations.

Example 4.14 The metapattern of Example 4.2 can be specified as follows to guide the mining of association rules describing customer buying habits.

mine associations as buyingHabits
matching $P(X : customer, W) \wedge Q(X, Y) \Rightarrow buys(X, Z)$

□

4. **Classification.**

⟨Mine_Knowledge_Specification⟩ ::=
mine classification [as ⟨pattern_name⟩]
analyze ⟨classifying_attribute_or_dimension⟩

This specifies that patterns for data classification are to be mined. The analyze clause specifies that the classification is performed according to the values of ⟨classifying_attribute_or_dimension⟩. For categorical attributes or dimensions, typically each value represents a class (such as "Vancouver", "New York", "Chicago", and so on for the dimension *location*). For numeric attributes or dimensions, each class may be defined by a range of values (such as "20-39", "40-59", "60-89" for *age*). Classification provides a concise framework which best describes the objects in each class and distinguishes them from other classes.

Example 4.15 To mine patterns classifying customer credit rating where credit rating is determined by the attribute *credit_info*, the following DMQL specification is used:

mine classification as classifyCustomerCreditRating
analyze credit_info

□

5. **Prediction.**

⟨Mine_Knowledge_Specification⟩ ::=
mine prediction [as ⟨pattern_name⟩]
analyze ⟨prediction_attribute_or_dimension⟩
{set {⟨attribute_or_dimension_*i*⟩= ⟨value_*i*⟩}}

This DMQL syntax is for prediction. It specifies the mining of missing or unknown continuous data values, or of the data distribution, for the attribute or dimension specified in the analyze clause. A predictive model is constructed based on the analysis of the values of the other attributes or dimensions describing the data objects (tuples). The set clause can be used to fix the values of these other attributes.

Example 4.16 To predict the retail price of a new item at *AllElectronics*, the following DMQL specification is used:

mine prediction as predictItemPrice
analyze price
set category = "TV" and brand = "SONY"

The set clause specifies that the resulting predictive patterns regarding price are for the subset of task-relevant data relating to SONY TV's. If no set clause is specified, then the prediction returned would be a data distribution for all categories and brands of *AllElectronics* items in the task-relevant data. □

The data mining language should also allow the specification of other kinds of knowledge to be mined, in addition to those shown above. These include the mining of data clusters, evolution rules or sequential patterns, and deviations.

4.2.3 Syntax for concept hierarchy specification

Concept hierarchies allow the mining of knowledge at multiple levels of abstraction. In order to accommodate the different viewpoints of users with regards to the data, there may be more than one concept hierarchy per attribute or dimension. For instance, some users may prefer to organize branch locations by provinces and states, while others may prefer to organize them according to languages used. In such cases, a user can indicate which concept hierarchy is to be used with the statement

use hierarchy ⟨hierarchy⟩ for ⟨attribute_or_dimension⟩.

Otherwise, a default hierarchy per attribute or dimension is used.

How can we define concept hierarchies, using DMQL? In Section 4.1.3, we studied four types of concept hierarchies, namely schema, set-grouping, operation-derived, and rule-based hierarchies. Let's look at the following syntax for defining each of these hierarchy types.

1. **Definition of schema hierarchies.**

   Example 4.17 Earlier, we defined a schema hierarchy for a relation *address* as the total order $street < city < province\_or\_state < country$. This can be defined in the data mining query language as:

   define hierarchy location_hierarchy on address as [street, city, province_or_state, country]

   The ordering of the listed attributes is important. In fact, a total order is defined which specifies that *street* is conceptually one level lower than *city*, which is in turn conceptually one level lower than *province_or_state*, and so on. □

   Example 4.18 A data mining system will typically have a predefined concept hierarchy for the schema *date* (*day*, *month*, *quarter*, *year*), such as:

   define hierarchy time_hierarchy on date as [day, month, quarter, year]

   □

   Example 4.19 Concept hierarchy definitions can involve several relations. For example, an *item_hierarchy* may involve two relations, *item* and *supplier*, defined by the following schema.

   $item(item\_ID, brand, type, place\_made, supplier)$
   $supplier(name, type, headquarter\_location, owner, size, assets, revenue)$

   The hierarchy *item_hierarchy* can be defined as follows:

   define hierarchy item_hierarchy on item, supplier as
   [item_ID, brand, item.supplier, item.type, supplier.type]
   where item.supplier = supplier.name

   If the concept hierarchy definition contains an attribute name that is shared by two relations, then the attribute is prefixed by its relation name, using the same dot (".") notation as in SQL (e.g., item.supplier). The join condition of the two relations is specified by a where clause. □

2. **Definition of set-grouping hierarchies.**

   Example 4.20 The set-grouping hierarchy for *age* of Example 4.4 can be defined in terms of ranges as follows:

   define hierarchy age_hierarchy for age on customer as
   level1: {*young, middle_aged, senior*} < level0: all
   level2: {20, ..., 39} < level1: *young*
   level2: {40, ..., 59} < level1: *middle_aged*
   level2: {60, ..., 89} < level1: *senior*

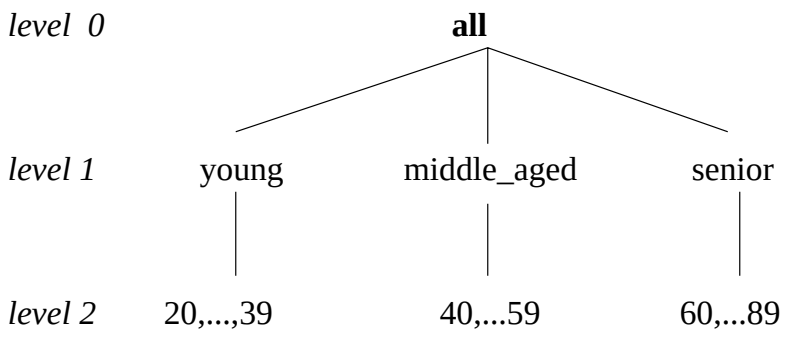


Figure 4.7: A concept hierarchy for the attribute *age*.

The notation "..." implicitly specifies all the possible values within the given range. For example, "{20, ..., 39}" includes all integers within the range of the endpoints, 20 and 39. Ranges may also be specified with real numbers as endpoints. The corresponding concept hierarchy is shown in Figure 4.7. The most general concept for *age* is all, and is placed at the root of the hierarchy. By convention, the all value is always at level 0 of any hierarchy. The all node in Figure 4.7 has three child nodes, representing more specific abstractions of *age*, namely *young*, *middle_aged*, and *senior*. These are at level 1 of the hierarchy. The age ranges for each of these level 1 concepts are defined at level 2 of the hierarchy. □

Example 4.21 The schema hierarchy in Example 4.17 for *location* can be refined by adding an additional concept level, *continent*.

define hierarchy on location_hierarchy as
country: {*Canada, USA, Mexico*} < continent: *NorthAmerica*
country: {*England, France, Germany, Italy*} < continent: *Europe*
...
continent: {*NorthAmerica, Europe, Asia*} < all

By listing the countries (for which *AllElectronics* sells merchandise) belonging to each continent, we build an additional concept layer on top of the schema hierarchy of Example 4.17. □

3. **Definition of operation-derived hierarchies**

Example 4.22 As an alternative to the set-grouping hierarchy for *age* in Example 4.20, a user may wish to define an operation-derived hierarchy for *age* based on data clustering routines. This is especially useful when the values of a given attribute are not uniformly distributed. A hierarchy for *age* based on clustering can be defined with the following statement:

define hierarchy age_hierarchy for age on customer as
{age_category(1), ..., age_category(5)} := cluster(default, age, 5) < all(age)

This statement indicates that a default clustering algorithm is to be performed on all of the *age* values in the relation *customer* in order to form five clusters. The clusters are ranges with names explicitly defined as "age_category(1), ..., age_category(5)", organized in ascending order. □

4. **Definition of rule-based hierarchies**

Example 4.23 A concept hierarchy can be defined based on a set of rules. Consider the concept hierarchy of Example 4.8 for *items* at *AllElectronics*. This hierarchy is based on item profit margins, where the profit margin of an item is defined as the difference between the retail price of the item, and the cost incurred by *AllElectronics* to purchase the item for sale. The hierarchy organizes items into *low_profit_margin* items, *medium-profit_margin* items, and *high_profit_margin* items, and is defined in DMQL by the following set of rules.

define hierarchy profit_margin_hierarchy on item as
level_1: *low_profit_margin* < level_0: all
if $(price - cost) < \$50$
level_1: *medium-profit_margin* < level_0: all
if $((price - cost) > \$50)$ and $((price - cost) \leq \$250))$
level_1: *high_profit_margin* < level_0: all
if $(price - cost) > \$250$

□

4.2.4 Syntax for interestingness measure specification

The user can control the number of uninteresting patterns returned by the data mining system by specifying measures of pattern interestingness and their corresponding thresholds. Interestingness measures include the confidence, support, noise, and novelty measures described in Section 4.1.4. Interestingness measures and thresholds can be specified by the user with the statement:

with [⟨interest_measure_name⟩] threshold = ⟨threshold_value⟩

Example 4.24 In mining association rules, a user can confine the rules to be found by specifying a minimum support and minimum confidence threshold of 0.05 and 0.7, respectively, with the statements:

with support threshold = 0.05
with confidence threshold = 0.7

□

The interestingness measures and threshold values can be set and modified interactively.

4.2.5 Syntax for pattern presentation and visualization specification

How can users specify the forms of presentation and visualization to be used in displaying the discovered patterns? Our data mining query language needs syntax which allows users to specify the display of discovered patterns in one or more forms, including rules, tables, crosstabs, pie or bar charts, decision trees, cubes, curves, or surfaces. We define the DMQL display statement for this purpose:

display as ⟨result_form⟩

where the ⟨result_form⟩ could be any of the knowledge presentation or visualization forms listed above.

Interactive mining should allow the discovered patterns to be viewed at different concept levels or from different angles. This can be accomplished with roll-up and drill-down operations, as described in Chapter 2. Patterns can be rolled-up, or viewed at a more general level, by *climbing up* the concept hierarchy of an attribute or dimension (replacing lower level concept values by higher level values). Generalization can also be performed by dropping attributes or dimensions. For example, suppose that a pattern contains the attribute *city*. Given the *location* hierarchy *city* < *province_or_state* < *country* < *continent*, then dropping the attribute *city* from the patterns will generalize the data to the next lowest level attribute, *province_or_state*. Patterns can be drilled-down on, or viewed at a less general level, by *stepping down* the concept hierarchy of an attribute or dimension. Patterns can also be made less general by adding attributes or dimensions to their description. The attribute added must be one of the attributes listed in the in relevance to clause for task-relevant specification. The user can alternately view the patterns at different levels of abstractions with the use of the following DMQL syntax:

⟨Multilevel_Manipulation⟩ ::= roll up on ⟨attribute_or_dimension⟩
| drill down on ⟨attribute_or_dimension⟩
| add ⟨attribute_or_dimension⟩
| drop ⟨attribute_or_dimension⟩

Example 4.25 Suppose descriptions are mined based on the dimensions *location*, *age*, and *income*. One may "roll up on location" or "drop age" to generalize the discovered patterns. □

age	type	place_made	count%
30-39	home security system	USA	19
40-49	home security system	USA	15
20-29	CD player	Japan	26
30-39	CD player	USA	13
40-49	large screen TV	Japan	8
...	...	...	...
			100%

Figure 4.8: Characteristic descriptions in the form of a table, or generalized relation.

4.2.6 Putting it all together — an example of a DMQL query

In the above discussion, we presented DMQL syntax for specifying data mining queries in terms of the five data mining primitives. For a given query, these primitives define the task-relevant data, the kind of knowledge to be mined, the concept hierarchies and interestingness measures to be used, and the representation forms for pattern visualization. Here we put these components together. Let's look at an example for the full specification of a DMQL query.

Example 4.26 Mining characteristic descriptions. Suppose, as a marketing manager of *AllElectronics*, you would like to characterize the buying habits of customers who purchase items priced at no less than \$100, with respect to the customer's age, the type of item purchased, and the place in which the item was made. For each characteristic discovered, you would like to know the percentage of customers having that characteristic. In particular, you are only interested in purchases made in Canada, and paid for with an American Express ("AmEx") credit card. You would like to view the resulting descriptions in the form of a table. This data mining query is expressed in DMQL as follows.

```
use database AllElectronics_db
use hierarchy location_hierarchy for B.address
mine characteristics as customerPurchasing
analyze count%
in relevance to C.age, I.type, I.place_made
from customer C, item I, purchases P, items_sold S, works_at W, branch B
where I.item_ID = S.item_ID and S.trans_ID = P.trans_ID and P.cust_ID = C.cust_ID
        and P.method_paid = "AmEx" and P.empl_ID = W.empl_ID and W.branch_ID = B.branch_ID
        and B.address = "Canada" and I.price ≥ 100
with noise threshold = 0.05
display as table
```

The data mining query is parsed to form an SQL query which retrieves the set of task-relevant data from the *AllElectronics* database. The concept hierarchy *location_hierarchy*, corresponding to the concept hierarchy of Figure 4.3 is used to generalize branch locations to high level concept levels such as "Canada". An algorithm for mining characteristic rules, which uses the generalized data, can then be executed. Algorithms for mining characteristic rules are introduced in Chapter 5. The mined characteristic descriptions, derived from the attributes *age, type* and *place_made*, are displayed as a table, or generalized relation (Figure 4.8). The percentage of task-relevant tuples satisfying each generalized tuple is shown as `count%`. If no visualization form is specified, a default form is used. The noise threshold of 0.05 means any generalized tuple found that represents less than 5% of the total count is omitted from display. □

Similarly, the complete DMQL specification of data mining queries for discrimination, association, classification, and prediction can be given. Example queries are presented in the following chapters which respectively study the mining of these kinds of knowledge.

4.3 Designing graphical user interfaces based on a data mining query language

A data mining query language provides necessary primitives which allow users to communicate with data mining systems. However, inexperienced users may find data mining query languages awkward to use, and the syntax difficult to remember. Instead, users may prefer to communicate with data mining systems through a Graphical User Interface (GUI). In relational database technology, SQL serves as a standard "core" language for relational systems, on top of which GUIs can easily be designed. Similarly, a data mining query language may serve as a "core language" for data mining system implementations, providing a basis for the development of GUI's for effective data mining.

A data mining GUI may consist of the following functional components.

1. **Data collection and data mining query composition**: This component allows the user to specify task-relevant data sets, and to compose data mining queries. It is similar to GUIs used for the specification of relational queries.

2. **Presentation of discovered patterns**: This component allows the display of the discovered patterns in various forms, including tables, graphs, charts, curves, or other visualization techniques.

3. **Hierarchy specification and manipulation:** This component allows for concept hierarchy specification, either manually by the user, or automatically (based on analysis of the data at hand). In addition, this component should allow concept hierarchies to be modified by the user, or adjusted automatically based on a given data set distribution.

4. **Manipulation of data mining primitives**: This component may allow the dynamic adjustment of data mining thresholds, as well as the selection, display, and modification of concept hierarchies. It may also allow the modification of previous data mining queries or conditions.

5. **Interactive multilevel mining**: This component should allow roll-up or drill-down operations on discovered patterns.

6. **Other miscellaneous information**: This component may include on-line help manuals, indexed search, debugging, and other interactive graphical facilities.

Do you think that data mining query languages may evolve to form a standard for designing data mining GUIs? If such an evolution is possible, the standard would facilitate data mining software development and system communication. Some GUI primitives, such as pointing to a particular point in a curve or graph, however, are difficult to specify using a text-based data mining query language like DMQL. Alternatively, a standardized GUI-based language may evolve and replace SQL-like data mining languages. Only time will tell.

4.4 Summary

- We have studied five **primitives** for specifying a data mining task in the form of a **data mining query**. These primitives are the specification of task-relevant data (i.e., the data set to be mined), the kind of knowledge to be mined (e.g., characterization, discrimination, association, classification, or prediction), background knowledge (typically in the form of concept hierarchies), interestingness measures, and knowledge presentation and visualization techniques to be used for displaying the discovered patterns.

- In defining the **task-relevant data**, the user specifies the database and tables (or data warehouse and data cubes) containing the data to be mined, conditions for selecting and grouping such data, and the attributes (or dimensions) to be considered during mining.

- **Concept hierarchies** provide useful background knowledge for expressing discovered patterns in concise, high level terms, and facilitate the mining of knowledge at multiple levels of abstraction.

- Measures of **pattern interestingness** assess the simplicity, certainty, utility, or novelty of discovered patterns. Such measures can be used to help reduce the number of uninteresting patterns returned to the user.

- Users should be able to specify the desired form for **visualizing** the discovered patterns, such as rules, tables, charts, decision trees, cubes, graphs, or reports. Roll-up and drill-down operations should also be available for the inspection of patterns at multiple levels of abstraction.
- **Data mining query languages** can be designed to support ad-hoc and interactive data mining. A data mining query language, such as DMQL, should provide commands for specifying each of the data mining primitives, as well as for concept hierarchy generation and manipulation. Such query languages are SQL-based, and may eventually form a standard on which graphical user interfaces for data mining can be based.

Exercises

1. List and describe the five primitives for specifying a data mining task.
2. Suppose that the university course database for *Big-University* contains the following attributes: the name, address, status (e.g., undergraduate or graduate), and major of each student, and their cumulative grade point average (GPA).
   (a) Propose a concept hierarchy for the attributes *status*, *major*, *GPA*, and *address*.
   (b) For each concept hierarchy that you have proposed above, what type of concept hierarchy have you proposed?
   (c) Define each hierarchy using DMQL syntax.
   (d) Write a DMQL query to find the characteristics of students who have an excellent GPA.
   (e) Write a DMQL query to compare students majoring in science with students majoring in arts.
   (f) Write a DMQL query to find associations involving course instructors, student grades, and some other attribute of your choice. Use a metarule to specify the format of associations you would like to find. Specify minimum thresholds for the confidence and support of the association rules reported.
   (g) Write a DMQL query to predict student grades in "Computing Science 101" based on student GPA to date and course instructor
3. Consider association rule 4.8 below, which was mined from the student database at *Big-University*.

$$major(X, \text{“}science\text{”}) \Rightarrow status(X, \text{“}undergrad\text{”}). \tag{4.8}$$

   Suppose that the number of students at the university (that is, the number of task-relevant data tuples) is 5000, that 56% of undergraduates at the university major in science, that 64% of the students are registered in programs leading to undergraduate degrees, and that 70% of the students are majoring in science.
   (a) Compute the confidence and support of Rule (4.8).
   (b) Consider Rule (4.9) below.

$$major(X, \text{“}biology\text{”}) \Rightarrow status(X, \text{“}undergrad\text{”}) \quad [17\%, 80\%] \tag{4.9}$$

   Suppose that 30% of science students are majoring in biology. Would you consider Rule (4.9) to be novel with respect to Rule (4.8)? Explain.
4. The ⟨Mine_Knowledge_Specification⟩ statement can be used to specify the mining of characteristic, discriminant, association, classification, and prediction rules. Propose a syntax for the mining of clusters.
5. Rather than requiring users to manually specify concept hierarchy definitions, some data mining systems can generate or modify concept hierarchies automatically *based on the analysis of data distributions*.
   (a) Propose concise DMQL syntax for the automatic generation of concept hierarchies.
   (b) A concept hierarchy may be automatically adjusted to reflect changes in the data. Propose concise DMQL syntax for the automatic adjustment of concept hierarchies.

(c) Give examples of your proposed syntax.

6. In addition to concept hierarchy creation, DMQL should also provide syntax which allows users to modify previously defined hierarchies. This syntax should allow the *insertion* of new nodes, the *deletion* of nodes, and the *moving* of nodes within the hierarchy.

   - To insert a new node N into level L of a hierarchy, one should specify its parent node P in the hierarchy, unless N is at the topmost layer.
   - To delete node N from a hierarchy, all of its descendent nodes should be removed from the hierarchy as well.
   - To move a node N to a different location within the hierarchy, the parent of N will change, and all of the descendents of N should be moved accordingly.

   (a) Propose DMQL syntax for each of the above operations.

   (b) Show examples of your proposed syntax.

   (c) For each operation, illustrate the operation by drawing the corresponding concept hierarchies ("before" and "after").

Bibliographic Notes

A number of objective interestingness measures have been proposed in the literature. Simplicity measures are given in Michalski [23]. The confidence and support measures for association rule interestingness described in this chapter were proposed in Agrawal, Imielinski, and Swami [1]. The strategy we described for identifying redundant multilevel association rules was proposed in Srikant and Agrawal [31, 32]. Other objective interestingness measures have been presented in [1, 6, 12, 17, 27, 19, 30]. Subjective measures of interestingness, which consider user beliefs regarding relationships in the data, are discussed in [18, 21, 20, 26, 29].

The DMQL data mining query language was proposed by Han et al. [11] for the DBMiner data mining system. *Discovery Board* (formerly *Data Mine*) was proposed by Imielinski, Virmani, and Abdulghani [13] as an application development interface prototype involving an SQL-based operator for data mining query specification and rule retrieval. An SQL-like operator for mining single-dimensional association rules was proposed by Meo, Psaila, and Ceri [22], and extended by Baralis and Psaila [4]. Mining with metarules is described in Klemettinen et al. [16], Fu and Han [9], Shen et al. [28], and Kamber et al. [14]. Other ideas involving the use of templates or predicate constraints in mining have been discussed in [3, 7, 18, 29, 33, 25].

For a comprehensive survey of visualization techniques, see *Visual Techniques for Exploring Databases* by Keim [15].

Bibliography

[1] R. Agrawal, T. Imielinski, and A. Swami. Database mining: A performance perspective. *IEEE Trans. Knowledge and Data Engineering*, 5:914–925, 1993.

[2] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proc. 1995 Int. Conf. Data Engineering*, pages 3–14, Taipei, Taiwan, March 1995.

[3] T. Anand and G. Kahn. Opportunity explorer: Navigating large databases using knowledge discovery templates. In *Proc. AAAI-93 Workshop Knowledge Discovery in Databases*, pages 45–51, Washington DC, July 1993.

[4] E. Baralis and G. Psaila. Designing templates for mining association rules. *Journal of Intelligent Information Systems*, 9:7–32, 1997.

[5] R.G.G. Cattell. *Object Data Management: Object-Oriented and Extended Relational Databases, Rev. Ed.* Addison-Wesley, 1994.

[6] M. S. Chen, J. Han, and P. S. Yu. Data mining: An overview from a database perspective. *IEEE Trans. Knowledge and Data Engineering*, 8:866–883, 1996.

[7] V. Dhar and A. Tuzhilin. Abstract-driven pattern discovery in databases. *IEEE Trans. Knowledge and Data Engineering*, 5:926–938, 1993.

[8] M. Ester, H.-P. Kriegel, and X. Xu. Knowledge discovery in large spatial databases: Focusing techniques for efficient class identification. In *Proc. 4th Int. Symp. Large Spatial Databases (SSD'95)*, pages 67–82, Portland, Maine, August 1995.

[9] Y. Fu and J. Han. Meta-rule-guided mining of association rules in relational databases. In *Proc. 1st Int. Workshop Integration of Knowledge Discovery with Deductive and Object-Oriented Databases (KDOOD'95)*, pages 39–46, Singapore, Dec. 1995.

[10] J. Han, Y. Cai, and N. Cercone. Data-driven discovery of quantitative rules in relational databases. *IEEE Trans. Knowledge and Data Engineering*, 5:29–40, 1993.

[11] J. Han, Y. Fu, W. Wang, J. Chiang, W. Gong, K. Koperski, D. Li, Y. Lu, A. Rajan, N. Stefanovic, B. Xia, and O. R. Zaïane. DBMiner: A system for mining knowledge in large relational databases. In *Proc. 1996 Int. Conf. Data Mining and Knowledge Discovery (KDD'96)*, pages 250–255, Portland, Oregon, August 1996.

[12] J. Hong and C. Mao. Incremental discovery of rules and structure by hierarchical and parallel clustering. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 177–193. AAAI/MIT Press, 1991.

[13] T. Imielinski, A. Virmani, and A. Abdulghani. DataMine – application programming interface and query language for KDD applications. In *Proc. 1996 Int. Conf. Data Mining and Knowledge Discovery (KDD'96)*, pages 256–261, Portland, Oregon, August 1996.

[14] M. Kamber, J. Han, and J. Y. Chiang. Metarule-guided mining of multi-dimensional association rules using data cubes. In *Proc. 3rd Int. Conf. Knowledge Discovery and Data Mining (KDD'97)*, pages 207–210, Newport Beach, California, August 1997.

[15] D. A. Keim. Visual techniques for exploring databases. In *Tutorial Notes, 3rd Int. Conf. on Knowledge Discovery and Data Mining (KDD'97)*, Newport Beach, CA, Aug. 1997.

[16] M. Klemettinen, H. Mannila, P. Ronkainen, H. Toivonen, and A.I. Verkamo. Finding interesting rules from large sets of discovered association rules. In *Proc. 3rd Int. Conf. Information and Knowledge Management*, pages 401–408, Gaithersburg, Maryland, Nov. 1994.

[17] A. J. Knobbe and P. W. Adriaans. Analysing binary associations. In *Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining (KDD'96)*, pages 311–314, Portland, OR, Aug. 1996.

[18] B. Liu, W. Hsu, and S. Chen. Using general impressions to analyze discovered classification rules. In *Proc. 3rd Int.. Conf. on Knowledge Discovery and Data Mining (KDD'97)*, pages 31–36, Newport Beach, CA, August 1997.

[19] J. Major and J. Mangano. Selecting among rules induced from a hurricane database. *Journal of Intelligent Information Systems*, 4:39–52, 1995.

[20] C. J. Matheus and G. Piatesky-Shapiro. An application of KEFIR to the analysis of healthcare information. In *Proc. AAAI'94 Workshop Knowledge Discovery in Databases (KDD'94)*, pages 441–452, Seattle, WA, July 1994.

[21] C.J. Matheus, G. Piatetsky-Shapiro, and D. McNeil. Selecting and reporting what is interesting: The KEFIR application to healthcare data. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 495–516. AAAI/MIT Press, 1996.

[22] R. Meo, G. Psaila, and S. Ceri. A new SQL-like operator for mining association rules. In *Proc. 1996 Int. Conf. Very Large Data Bases*, pages 122–133, Bombay, India, Sept. 1996.

[23] R. S. Michalski. A theory and methodology of inductive learning. In Michalski et al., editor, *Machine Learning: An Artificial Intelligence Approach, Vol. 1*, pages 83–134. Morgan Kaufmann, 1983.

[24] R. Ng and J. Han. Efficient and effective clustering method for spatial data mining. In *Proc. 1994 Int. Conf. Very Large Data Bases*, pages 144–155, Santiago, Chile, September 1994.

[25] R. Ng, L. V. S. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained associations rules. In *Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data*, pages 13–24, Seattle, Washington, June 1998.

[26] G. Piatesky-Shapiro and C. J. Matheus. The interestingness of deviations. In *Proc. AAAI'94 Workshop Knowledge Discovery in Databases (KDD'94)*, pages 25–36, Seattle, WA, July 1994.

[27] G. Piatetsky-Shapiro. Discovery, analysis, and presentation of strong rules. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 229–238. AAAI/MIT Press, 1991.

[28] W. Shen, K. Ong, B. Mitbander, and C. Zaniolo. Metaqueries for data mining. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 375–398. AAAI/MIT Press, 1996.

[29] A. Silberschatz and A. Tuzhilin. What makes patterns interesting in knowledge discovery systems. *IEEE Trans. on Knowledge and Data Engineering*, 8:970–974, Dec. 1996.

[30] P. Smyth and R.M. Goodman. An information theoretic approch to rule induction. *IEEE Trans. Knowledge and Data Engineering*, 4:301–316, 1992.

[31] R. Srikant and R. Agrawal. Mining generalized association rules. In *Proc. 1995 Int. Conf. Very Large Data Bases*, pages 407–419, Zurich, Switzerland, Sept. 1995.

[32] R. Srikant and R. Agrawal. Mining quantitative association rules in large relational tables. In *Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data*, pages 1–12, Montreal, Canada, June 1996.

[33] R. Srikant, Q. Vu, and R. Agrawal. Mining association rules with item constraints. In *Proc. 3rd Int. Conf. Knowledge Discovery and Data Mining (KDD'97)*, pages 67–73, Newport Beach, California, August 1997.

[34] M. Stonebraker. *Readings in Database Systems, 2ed.* Morgan Kaufmann, 1993.

[35] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: an efficient data clustering method for very large databases. In *Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data*, pages 103–114, Montreal, Canada, June 1996.

Contents

List of Figures

Chapter 5

Concept Description: Characterization and Comparison

From a data analysis point of view, data mining can be classified into two categories: *descriptive data mining* and *predictive data mining*. The former describes the data set in a concise and summarative manner and presents interesting general properties of the data; whereas the latter constructs one or a set of models, by performing certain analysis on the available set of data, and attempts to predict the behavior of new data sets.

Databases usually store large amounts of data in great detail. However, users often like to view sets of *summarized* data in concise, descriptive terms. Such data descriptions may provide an overall picture of a class of data or distinguish it from a set of comparative classes. Moreover, users like the ease and flexibility of having data sets described at different levels of granularity and from different angles. Such descriptive data mining is called *concept description*, and forms an important component of data mining.

In this chapter, you will learn how concept description can be performed efficiently and effectively.

5.1 What is concept description?

A database management system usually provides convenient tools for users to extract various kinds of data stored in large databases. Such data extraction tools often use database query languages, such as SQL, or report writers. These tools, for example, may be used to locate a person's telephone number from an on-line telephone directory, or print a list of records for all of the transactions performed in a given computer store in 1997. The retrieval of data from databases, and the application of aggregate functions (such as summation, counting, etc.) to the data represent an important functionality of database systems: that of **query processing**. Various kinds of query processing techniques have been developed. However, query processing is not data mining. While query processing retrieves sets of data from databases and can compute aggregate functions on the retrieved data, data mining analyzes the data and discovers interesting patterns *hidden* in the database.

The simplest kind of descriptive data mining is *concept description*. **Concept description** is sometimes called **class description** when the concept to be described refers to a class of objects. A concept usually refers to a collection of data such as *stereos, frequent_buyers, graduate_students*, and so on. As a data mining task, concept description is not a simple enumeration of the data. Instead, it generates descriptions for *characterization* and *comparison* of the data. **Characterization** provides a concise and succinct summarization of the given collection of data, while concept or class **comparison** (also known as **discrimination**) provides descriptions comparing two or more collections of data. Since concept description involves both characterization and comparison, we will study techniques for accomplishing each of these tasks.

There are often many ways to describe a collection of data, and different people may like to view the same concept or class of objects from different angles or abstraction levels. Therefore, the description of a concept or a class is usually not unique. Some descriptions may be more preferred than others, based on objective interestingness measures regarding the conciseness or coverage of the description, or on subjective measures which consider the users' background knowledge or beliefs. Therefore, it is important to be able to generate different concept descriptions

both efficiently and conveniently.

Concept description has close ties with *data generalization*. Given the large amount of data stored in databases, it is useful to be able to describe concepts in concise and succinct terms at generalized (rather than low) levels of abstraction. Allowing data sets to be generalized at multiple levels of abstraction facilitates users in examining the general behavior of the data. Given the *AllElectronics* database, for example, instead of examining individual customer transactions, sales managers may prefer to view the data generalized to higher levels, such as summarized by customer groups according to geographic regions, frequency of purchases per group, and customer income. Such multiple dimensional, multilevel data generalization is similar to multidimensional data analysis in data warehouses. In this context, concept description resembles *on-line analytical processing* (OLAP) in data warehouses, discussed in Chapter 2.

"What are the differences between concept description in large databases and on-line analytical processing?" The fundamental differences between the two include the following.

- Data warehouses and OLAP tools are based on a multidimensional data model which views data in the form of a data cube, consisting of dimensions (or attributes) and measures (aggregate functions). However, the possible data types of the dimensions and measures for most commercial versions of these systems are restricted. Many current OLAP systems confine dimensions to nonnumeric data[1]. Similarly, measures (such as `count()`, `sum()`, `average()`) in current OLAP systems apply only to numeric data. In contrast, for concept formation, the database attributes can be of various data types, including numeric, nonnumeric, spatial, text or image. Furthermore, the aggregation of attributes in a database may include sophisticated data types, such as the collection of nonnumeric data, the merge of spatial regions, the composition of images, the integration of texts, and the group of object pointers. Therefore, OLAP, with its restrictions on the possible dimension and measure types, represents a simplified model for data analysis. Concept description in databases can handle complex data types of the attributes and their aggregations, as necessary.

- On-line analytical processing in data warehouses is a purely user-controlled process. The selection of dimensions and the application of OLAP operations, such as drill-down, roll-up, dicing, and slicing, are directed and controlled by the users. Although the control in most OLAP systems is quite user-friendly, users do require a good understanding of the role of each dimension. Furthermore, in order to find a satisfactory description of the data, users may need to specify a long sequence of OLAP operations. In contrast, concept description in data mining strives for a more automated process which helps users determine which dimensions (or attributes) should be included in the analysis, and the degree to which the given data set should be generalized in order to produce an interesting summarization of the data.

In this chapter, you will learn methods for concept description, including multilevel generalization, summarization, characterization and discrimination. Such methods set the foundation for the implementation of two major functional modules in data mining: multiple-level *characterization* and *discrimination*. In addition, you will also examine techniques for the presentation of concept descriptions in multiple forms, including tables, charts, graphs, and rules.

5.2 Data generalization and summarization-based characterization

Data and objects in databases often contain detailed information at primitive concept levels. For example, the *item* relation in a *sales* database may contain attributes describing low level item information such as *item_ID*, *name*, *brand*, *category*, *supplier*, *place_made*, and *price*. It is useful to be able to summarize a large set of data and present it at a high conceptual level. For example, summarizing a large set of items relating to Christmas season sales provides a general description of such data, which can be very helpful for sales and marketing managers. This requires an important functionality in data mining: *data generalization*.

Data generalization is a process which abstracts a large set of task-relevant data in a database from a relatively low conceptual level to higher conceptual levels. Methods for the efficient and flexible generalization of large data sets can be categorized according to two approaches: (1) the data cube approach, and (2) the attribute-oriented induction approach.

[1] Note that in Chapter 3, we showed how concept hierarchies may be automatically generated from numeric data to form numeric dimensions. This feature, however, is a result of recent research in data mining and is not available in most commercial systems.

5.2.1 Data cube approach for data generalization

In the **data cube approach** (or **OLAP approach**) to data generalization, the data for analysis are stored in a multidimensional database, or *data cube*. Data cubes and their use in OLAP for data generalization were described in detail in Chapter 2. In general, the data cube approach "materializes data cubes" by first identifying expensive computations required for frequently-processed queries. These operations typically involve aggregate functions, such as `count()`, `sum()`, `average()`, and `max()`. The computations are performed, and their results are stored in data cubes. Such computations may be performed for various levels of data abstraction. These materialized views can then be used for decision support, knowledge discovery, and many other applications.

A set of attributes may form a hierarchy or a lattice structure, defining a data cube dimension. For example, *date* may consist of the attributes *day, week, month, quarter*, and *year* which form a lattice structure, and a data cube dimension for *time*. A data cube can store pre-computed aggregate functions for all or some of its dimensions. The precomputed aggregates correspond to specified `group-by`'s of different sets or subsets of attributes.

Generalization and specialization can be performed on a multidimensional data cube by *roll-up* or *drill-down* operations. A roll-up operation reduces the number of dimensions in a data cube, or generalizes attribute values to higher level concepts. A drill-down operation does the reverse. Since many aggregate functions need to be computed repeatedly in data analysis, the storage of precomputed results in a multidimensional data cube may ensure fast response time and offer flexible views of data from different angles and at different levels of abstraction.

The data cube approach provides an efficient implementation of data generalization, which in turn forms an important function in descriptive data mining. However, as we pointed out in Section 5.1, most commercial data cube implementations confine the data types of dimensions to *simple, nonnumeric data* and of measures to *simple, aggregated numeric values*, whereas many applications may require the analysis of more complex data types. Moreover, the data cube approach cannot answer some important questions which concept description can, such as which dimensions should be used in the description, and at what levels should the generalization process reach. Instead, it leaves the responsibility of these decisions to the users.

In the next subsection, we introduce an alternative approach to data generalization called *attribute-oriented induction*, and examine how it can be applied to concept description. Moreover, we discuss how to integrate the two approaches, data cube and attribute-oriented induction, for concept description.

5.2.2 Attribute-oriented induction

The attribute-oriented induction approach to data generalization and summarization-based characterization was first proposed in 1989, a few years prior to the introduction of the data cube approach. The data cube approach can be considered as a data warehouse-based, precomputation-oriented, materialized view approach. It performs *off-line* aggregation before an OLAP or data mining query is submitted for processing. On the other hand, the *attribute-oriented approach*, at least in its initial proposal, is a relational database query-oriented, generalization-based, *on-line* data analysis technique. However, there is no inherent barrier distinguishing the two approaches based on on-line aggregation versus off-line precomputation. Some aggregations in the data cube can be computed on-line, while off-line precomputation of multidimensional space can speed up attribute-oriented induction as well. In fact, data mining systems based on attribute-oriented induction, such as DBMiner, have been optimized to include such off-line precomputation.

Let's first introduce the attribute-oriented induction approach. We will then perform a detailed analysis of the approach and its variations and extensions.

The general idea of attribute-oriented induction is to first collect the task-relevant data using a relational database query and then perform generalization based on the examination of the number of distinct values of each attribute in the relevant set of data. The generalization is performed by either *attribute removal* or *attribute generalization* (also known as *concept hierarchy ascension*). Aggregation is performed by merging identical, generalized tuples, and accumulating their respective counts. This reduces the size of the generalized data set. The resulting generalized relation can be mapped into different forms for presentation to the user, such as charts or rules.

The following series of examples illustrates the process of attribute-oriented induction.

Example 5.1 Specifying a data mining query for characterization with DMQL. Suppose that a user would like to describe the general characteristics of graduate students in the *Big-University* database, given the attributes *name, gender, major, birth_place, birth_date, residence, phone# (telephone number)*, and *gpa (grade_point_average)*.

A data mining query for this characterization can be expressed in the data mining query language DMQL as follows.

```
use Big_University_DB
mine characteristics as "Science_Students"
in relevance to name, gender, major, birth_place, birth_date, residence, phone#, gpa
from student
where status in "graduate"
```

We will see how this example of a typical data mining query can apply attribute-oriented induction for mining characteristic descriptions. □

"*What is the first step of attribute-oriented induction?*"

First, **data focusing** should be performed *prior* to attribute-oriented induction. This step corresponds to the specification of the task-relevant data (or, data for analysis) as described in Chapter 4. The data are collected based on the information provided in the data mining query. Since a data mining query is usually relevant to only a portion of the database, selecting the relevant set of data not only makes mining more efficient, but also derives more meaningful results than mining on the entire database.

Specifying the set of relevant attributes (i.e., attributes for mining, as indicated in DMQL with the in relevance to clause) may be difficult for the user. Sometimes a user may select only a few attributes which she feels may be important, while missing others that would also play a role in the description. For example, suppose that the dimension *birth_place* is defined by the attributes *city, province_or_state*, and *country*. Of these attributes, the user has only thought to specify *city*. In order to allow generalization on the *birth_place* dimension, the other attributes defining this dimension should also be included. In other words, having the system automatically include *province_or_state* and *country* as relevant attributes allows *city* to be generalized to these higher conceptual levels during the induction process.

At the other extreme, a user may introduce too many attributes by specifying all of the possible attributes with the clause "in relevance to ∗". In this case, all of the attributes in the relation specified by the from clause would be included in the analysis. Many of these attributes are unlikely to contribute to an interesting description. Section 5.4 describes a method to handle such cases by filtering out statistically irrelevant or weakly relevant attributes from the descriptive mining process.

"*What does the '*where *status* in *"graduate"' clause mean?*"

The above where clause implies that a concept hierarchy exists for the attribute *status*. Such a concept hierarchy organizes primitive level data values for *status*, such as *"M.Sc."*, *"M.A."*, *"M.B.A."*, *"Ph.D."*, *"B.Sc."*, *"B.A."*, into higher conceptual levels, such as *"graduate"* and *"undergraduate"*. This use of concept hierarchies does not appear in traditional relational query languages, yet is a common feature in data mining query languages.

Example 5.2 Transforming a data mining query to a relational query. The data mining query presented in Example 5.1 is transformed into the following relational query for the collection of the task-relevant set of data.

```
use Big_University_DB
select name, gender, major, birth_place, birth_date, residence, phone#, gpa
from student
where status in {"M.Sc.", "M.A.", "M.B.A.", "Ph.D."}
```

The transformed query is executed against the relational database, *Big_University_DB*, and returns the data shown in Table 5.1. This table is called the (task-relevant) **initial working relation**. It is the data on which induction will be performed. Note that each tuple is, in fact, a conjunction of attribute-value pairs. Hence, we can think of a tuple within a relation as a rule of conjuncts, and of induction on the relation as the generalization of these rules.

□

"*Now that the data are ready for attribute-oriented induction, how is attribute-oriented induction performed?*"

The essential operation of attribute-oriented induction is *data generalization*, which can be performed in one of two ways on the initial working relation: (1) *attribute removal*, or (2) *attribute generalization*.

name	gender	major	birth_place	birth_date	residence	phone#	gpa
Jim Woodman	M	CS	Vancouver, BC, Canada	8-12-76	3511 Main St., Richmond	687-4598	3.67
Scott Lachance	M	CS	Montreal, Que, Canada	28-7-75	345 1st Ave., Vancouver	253-9106	3.70
Laura Lee	F	physics	Seattle, WA, USA	25-8-70	125 Austin Ave., Burnaby	420-5232	3.83
...	...	...	...	...	...	...	...

Table 5.1: Initial working relation: A collection of task-relevant data.

1. **Attribute removal** is based on the following rule: *If there is a large set of distinct values for an attribute of the initial working relation, but either (1) there is no generalization operator on the attribute (e.g., there is no concept hierarchy defined for the attribute), or (2) its higher level concepts are expressed in terms of other attributes, then the attribute should be removed from the working relation.*

   What is the reasoning behind this rule? An attribute-value pair represents a conjunct in a generalized tuple, or rule. The removal of a conjunct eliminates a constraint and thus generalizes the rule. If, as in case 1, there is a large set of distinct values for an attribute but there is no generalization operator for it, the attribute should be removed because it cannot be generalized, and preserving it would imply keeping a large number of disjuncts which contradicts the goal of generating concise rules. On the other hand, consider case 2, where the higher level concepts of the attribute are expressed in terms of other attributes. For example, suppose that the attribute in question is *street*, whose higher level concepts are represented by the attributes $\langle$*city, province_or_state, country*$\rangle$. The removal of *street* is equivalent to the application of a generalization operator. This rule corresponds to the generalization rule known as *dropping conditions* in the machine learning literature on *learning-from-examples*.

2. **Attribute generalization** is based on the following rule: *If there is a large set of distinct values for an attribute in the initial working relation, and there exists a set of generalization operators on the attribute, then a generalization operator should be selected and applied to the attribute.*

   This rule is based on the following reasoning. Use of a generalization operator to generalize an attribute value within a tuple, or rule, in the working relation will make the rule cover more of the original data tuples, thus generalizing the concept it represents. This corresponds to the generalization rule known as *climbing generalization trees* in *learning-from-examples*.

Both rules, *attribute removal* and *attribute generalization*, claim that if there is a *large* set of distinct values for an attribute, further generalization should be applied. This raises the question: how large is "*a large set of distinct values for an attribute*" considered to be?

Depending on the attributes or application involved, a user may prefer some attributes to remain at a rather low abstraction level while others to be generalized to higher levels. The control of how high an attribute should be generalized is typically quite subjective. The control of this process is called **attribute generalization control**. If the attribute is generalized "too high", it may lead to over-generalization, and the resulting rules may not be very informative. On the other hand, if the attribute is not generalized to a "sufficiently high level", then under-generalization may result, where the rules obtained may not be informative either. Thus, a balance should be attained in attribute-oriented generalization.

There are many possible ways to control a generalization process. Two common approaches are described below.

- The first technique, called **attribute generalization threshold control**, either sets one generalization threshold for all of the attributes, or sets one threshold for each attribute. If the number of distinct values in an attribute is greater than the attribute threshold, further attribute removal or attribute generalization should be performed. Data mining systems typically have a default attribute threshold value (typically ranging from 2 to 8), and should allow experts and users to modify the threshold values as well. If a user feels that the generalization reaches too high a level for a particular attribute, she can increase the threshold. This corresponds to drilling down along the attribute. Also, to further generalize a relation, she can reduce the threshold of a particular attribute, which corresponds to rolling up along the attribute.

- The second technique, called **generalized relation threshold control**, sets a threshold for the generalized relation. If the number of (distinct) tuples in the generalized relation is greater than the threshold, further

generalization should be performed. Otherwise, no further generalization should be performed. Such a threshold may also be preset in the data mining system (usually within a range of 10 to 30), or set by an expert or user, and should be adjustable. For example, if a user feels that the generalized relation is too small, she can increase the threshold, which implies drilling down. Otherwise, to further generalize a relation, she can reduce the threshold, which implies rolling up.

These two techniques can be applied in sequence: first apply the attribute threshold control technique to generalize each attribute, and then apply relation threshold control to further reduce the size of the generalized relation.

Notice that no matter which generalization control technique is applied, the user should be allowed to adjust the generalization thresholds in order to obtain interesting concept descriptions. This adjustment, as we saw above, is similar to drilling down and rolling up, as discussed under OLAP operations in Chapter 2. However, there is a methodological distinction between these OLAP operations and attribute-oriented induction. In OLAP, each step of drilling down or rolling up is directed and controlled by the user; whereas in attribute-oriented induction, most of the work is performed automatically by the induction process and controlled by generalization thresholds, and only minor adjustments are made by the user after the automated induction.

In many database-oriented induction processes, users are interested in obtaining quantitative or statistical information about the data at different levels of abstraction. Thus, it is important to accumulate count and other aggregate values in the induction process. Conceptually, this is performed as follows. A special measure, or numerical attribute, that is associated with each database tuple is the aggregate function, `count`. Its value for each tuple in the initial working relation is initialized to 1. Through attribute removal and attribute generalization, tuples within the initial working relation may be generalized, resulting in groups of *identical tuples*. In this case, all of the identical tuples forming a group should be merged into one tuple. The count of this new, generalized tuple is set to the total number of tuples from the initial working relation that are represented by (i.e., were merged into) the new generalized tuple. For example, suppose that by attribute-oriented induction, 52 data tuples from the initial working relation are all generalized to the same tuple, T. That is, the generalization of these 52 tuples resulted in 52 identical instances of tuple T. These 52 identical tuples are merged to form one instance of T, whose count is set to 52. Other popular aggregate functions include `sum` and `avg`. For a given generalized tuple, `sum` contains the sum of the values of a given numeric attribute for the initial working relation tuples making up the generalized tuple. Suppose that tuple T contained `sum(`*units_sold*`)` as an aggregate function. The sum value for tuple T would then be set to the total number of units sold for each of the 52 tuples. The aggregate `avg` (average) is computed according to the formula, `avg = sum/count`.

Example 5.3 Attribute-oriented induction. Here we show how attributed-oriented induction is performed on the initial working relation of Table 5.1, obtained in Example 5.2. For each attribute of the relation, the generalization proceeds as follows:

1. *name*: Since there are a large number of distinct values for *name* and there is no generalization operation defined on it, this attribute is removed.

2. *gender*: Since there are only two distinct values for *gender*, this attribute is retained and no generalization is performed on it.

3. *major*: Suppose that a concept hierarchy has been defined which allows the attribute *major* to be generalized to the values {*letters&science, engineering, business*}. Suppose also that the attribute generalization threshold is set to 5, and that there are over 20 distinct values for *major* in the initial working relation. By attribute generalization and attribute generalization control, *major* is therefore generalized by climbing the given concept hierarchy.

4. *birth_place*: This attribute has a large number of distinct values, therefore, we would like to generalize it. Suppose that a concept hierarchy exists for *birth_place*, defined as *city* $<$ *province_or_state* $<$ *country*. Suppose also that the number of distinct values for *country* in the initial working relation is greater than the attribute generalization threshold. In this case, *birth_place* would be removed, since even though a generalization operator exists for it, the generalization threshold would not be satisfied. Suppose instead that for our example, the number of distinct values for *country* is less than the attribute generalization threshold. In this case, *birth_place* is generalized to *birth_country*.

5. *birth_date*: Suppose that a hierarchy exists which can generalize *birth_date* to *age*, and *age* to *age_range*, and that the number of age ranges (or intervals) is small with respect to the attribute generalization threshold. Generalization of *birth_date* should therefore take place.

6. *residence*: Suppose that *residence* is defined by the attributes *number, street, residence_city, residence_province_or_state* and *residence_country*. The number of distinct values for *number* and *street* will likely be very high, since these concepts are quite low level. The attributes *number* and *street* should therefore be removed, so that *residence* is then generalized to *residence_city*, which contains fewer distinct values.

7. *phone#*: As with the attribute *name* above, this attribute contains too many distinct values and should therefore be removed in generalization.

8. *gpa*: Suppose that a concept hierarchy exists for *gpa* which groups grade point values into numerical intervals like {3.75-4.0, 3.5-3.75, ...}, which in turn are grouped into descriptive values, such as {*excellent, very good, ...*}. The attribute can therefore be generalized.

The generalization process will result in groups of identical tuples. For example, the first two tuples of Table 5.1 both generalize to the same identical tuple (namely, the first tuple shown in Table 5.2). Such identical tuples are then merged into one, with their `count`s accumulated. This process leads to the generalized relation shown in Table 5.2.

gender	major	birth_country	age_range	residence_city	gpa	count
M	Science	Canada	20-25	Richmond	very_good	16
F	Science	Foreign	25-30	Burnaby	excellent	22
...	...	...	...	...	...	...

Table 5.2: A generalized relation obtained by attribute-oriented induction on the data of Table 4.1.

Based on the vocabulary used in OLAP, we may view `count` as a *measure*, and the remaining attributes as *dimensions*. Note that aggregate functions, such as `sum`, may be applied to numerical attributes, like *salary* and *sales*. These attributes are referred to as *measure attributes*.

The generalized relation can also be presented in other forms, as discussed in the following subsection. □

5.2.3 Presentation of the derived generalization

"Attribute-oriented induction generates one or a set of generalized descriptions. How can these descriptions be visualized?" The descriptions can be presented to the user in a number of different ways.

Generalized descriptions resulting from attribute-oriented induction are most commonly displayed in the form of a **generalized relation**, such as the generalized relation presented in Table 5.2 of Example 5.3.

Example 5.4 Suppose that attribute-oriented induction was performed on a *sales* relation of the *AllElectronics* database, resulting in the generalized description of Table 5.3 for sales in 1997. The description is shown in the form of a generalized relation.

location	item	sales (in million dollars)	count (in thousands)
Asia	TV	15	300
Europe	TV	12	250
North_America	TV	28	450
Asia	computer	120	1000
Europe	computer	150	1200
North_America	computer	200	1800

Table 5.3: A generalized relation for the sales in 1997.

□

Descriptions can also be visualized in the form of **cross-tabulations**, or **crosstabs**. In a two-dimensional crosstab, each row represents a value from an attribute, and each column represents a value from another attribute. In an n-dimensional crosstab (for $n > 2$), the columns may represent the values of more than one attribute, with subtotals shown for attribute-value groupings. This representation is similar to *spreadsheets*. It is easy to map directly from a data cube structure to a crosstab.

Example 5.5 The generalized relation shown in Table 5.3 can be transformed into the 3-dimensional cross-tabulation shown in Table 5.4.

location \ item	TV		computer		*both_items*	
	sales	count	sales	count	sales	count
Asia	15	300	120	1000	135	1300
Europe	12	250	150	1200	162	1450
North_America	28	450	200	1800	228	2250
all_regions	45	1000	470	4000	525	5000

Table 5.4: A crosstab for the sales in 1997.

□

Generalized data may be presented in graph forms, such as bar charts, pie charts, and curves. Visualization with graphs is popular in data analysis. Such graphs and curves can represent 2-D or 3-D data.

Example 5.6 The sales data of the crosstab shown in Table 5.4 can be transformed into the bar chart representation of Figure 5.1, and the pie chart representation of Figure 5.2. □

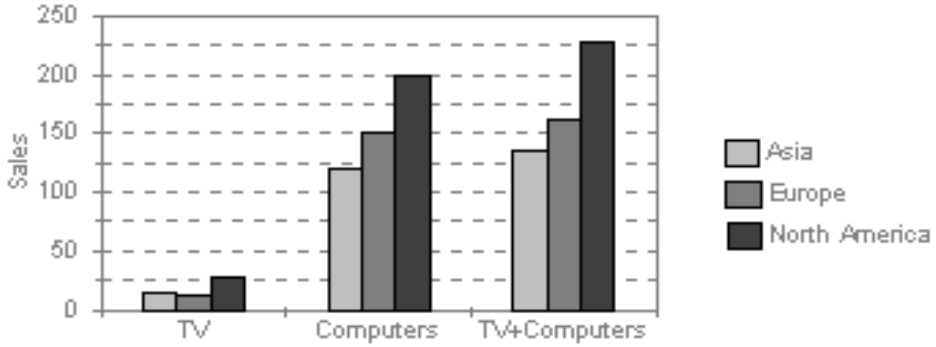


Figure 5.1: Bar chart representation of the sales in 1997.

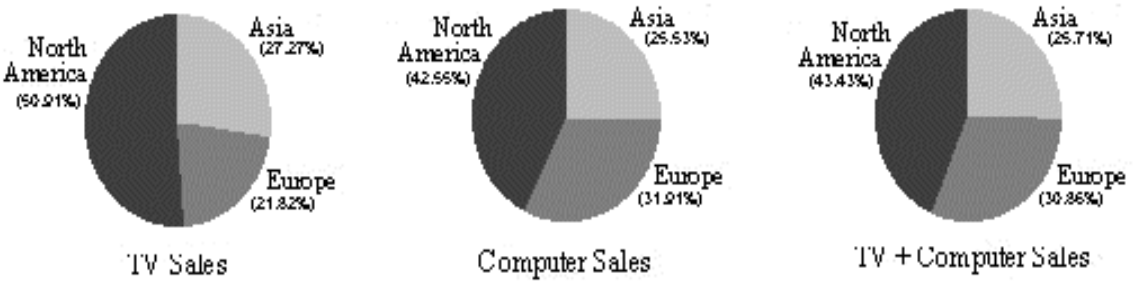


Figure 5.2: Pie chart representation of the sales in 1997.

Finally, a three-dimensional generalized relation or crosstab can be represented by a 3-D data cube. Such a 3-D cube view is an attractive tool for cube browsing.

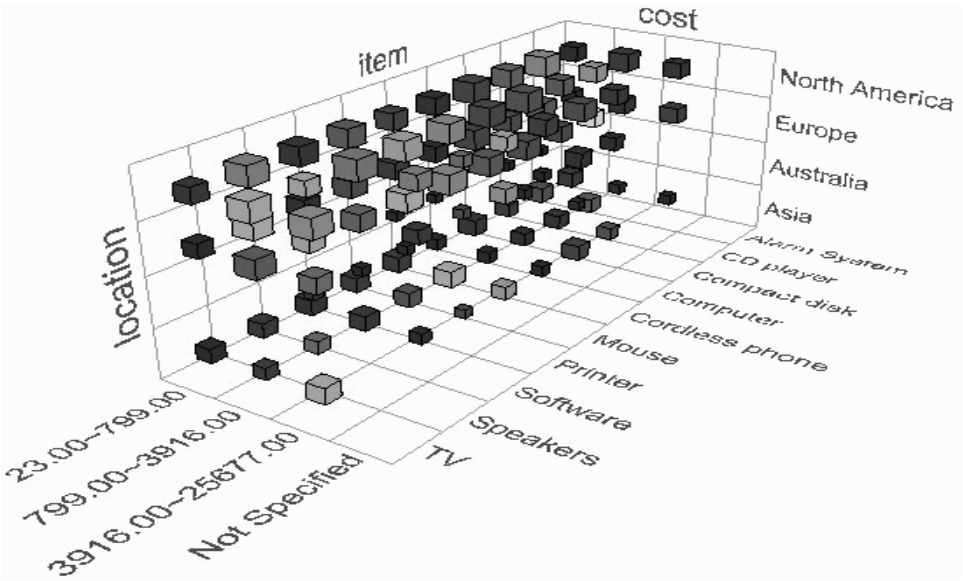


Figure 5.3: A 3-D Cube view representation of the sales in 1997.

Example 5.7 Consider the data cube shown in Figure 5.3 for the dimensions *item, location*, and *cost*. The *size* of a cell (displayed as a tiny cube) represents the `count` of the corresponding cell, while the *brightness* of the cell can be used to represent another measure of the cell, such as `sum`(*sales*). Pivoting, drilling, and slicing-and-dicing operations can be performed on the data cube browser with mouse clicking. □

A generalized relation may also be represented in the form of logic rules. Typically, each generalized tuple represents a rule disjunct. Since data in a large database usually span a diverse range of distributions, a single generalized tuple is unlikely to *cover*, or represent, 100% of the initial working relation tuples, or *cases*. Thus quantitative information, such as the percentage of data tuples which satisfies the left-hand side of the rule that also satisfies the right-hand side the rule, should be associated with each rule. A logic rule that is associated with quantitative information is called a **quantitative rule**.

To define a quantitative characteristic rule, we introduce the **t-weight** as an interestingness measure which describes the *typicality* of each *disjunct* in the rule, or of each *tuple* in the corresponding generalized relation. The measure is defined as follows. Let the class of objects that is to be characterized (or described by the rule) be called the *target class*. Let q_a be a generalized tuple describing the target class. The **t-weight** for q_a is the percentage of tuples of the target class from the initial working relation that are covered by q_a. Formally, we have

$$t\_weight = count(q_a)/\Sigma_{i=1}^{N} count(q_i), \tag{5.1}$$

where N is the number of tuples for the target class in the generalized relation, $q_1, \ldots, q_N$ are tuples for the target class in the generalized relation, and q_a is in $q_1, \ldots, q_N$. Obviously, the range for the t-weight is [0, 1] (or [0%, 100%]).

A quantitative characteristic rule can then be represented either (i) in logic form by associating the corresponding t-weight value with each disjunct covering the target class, or (ii) in the relational table or crosstab form by changing the `count` values in these tables for tuples of the target class to the corresponding t-weight values.

Each disjunct of a quantitative characteristic rule represents a condition. In general, the disjunction of these conditions forms a *necessary* condition of the target class, since the condition is derived based on all of the cases of the target class, that is, all tuples of the target class must satisfy this condition. However, the rule may not be a *sufficient* condition of the target class, since a tuple satisfying the same condition could belong to another class. Therefore, the rule should be expressed in the form

$$\forall X,\ target\_class(X) \ \Rightarrow\ condition_1(X)[t : w_1] \vee \cdots \vee condition_n(X)[t : w_n]. \tag{5.2}$$

The rule indicates that if X is in the $target\_class$, there is a possibility of w_i that X satisfies $condition_i$, where w_i is the t-weight value for condition or disjunct i, and i is in $\{1, \ldots, n\}$,

Example 5.8 The crosstab shown in Table 5.4 can be transformed into logic rule form. Let the target class be the set of computer items. The corresponding characteristic rule, in logic form, is

$$\begin{aligned}\forall X,\ item(X) = \text{“}computer\text{”} \ \Rightarrow \\ (location(X) = \text{“}Asia\text{”})\ [t:25.00\%] \vee (location(X) = \text{“}Europe\text{”})\ [t:30.00\%] \vee \\ (location(X) = \text{“}North\_America\text{”})\ [t:45.00\%]\end{aligned} \tag{5.3}$$

Notice that the first t-weight value of 25.00% is obtained by 1000, the value corresponding to the count slot for $(computer, Asia)$, divided by 4000, the value corresponding to the count slot for $(computer, all\_regions)$. (That is, 4000 represents the total number of computer items sold). The t-weights of the other two disjuncts were similarly derived. Quantitative characteristic rules for other target classes can be computed in a similar fashion. □

5.3 Efficient implementation of attribute-oriented induction

5.3.1 Basic attribute-oriented induction algorithm

Based on the above discussion, we summarize the attribute-oriented induction technique with the following algorithm which mines generalized characteristic rules in a relational database based on a user's data mining request.

Algorithm 5.3.1 (Basic attribute-oriented induction for mining data characteristics) *Mining generalized characteristics in a relational database based on a user's data mining request.*

Input. (i) A relational database DB, (ii) a data mining query, $DMQuery$, (iii) $Gen(a_i)$, a set of concept hierarchies or generalization operators on attributes a_i, and (iv) T_i, a set of *attribute generalization thresholds* for attributes a_i, and T, a *relation generalization threshold*.

Output. A characteristic description based on $DMQuery$.

Method.

1. **InitRel**: Derivation of the *initial working relation*, $\mathcal{W}_0$. This is performed by deriving a relational database query based on the data mining query, $DMQuery$. The relational query is executed against the database, DB, and the query result forms the set of task-relevant data, $\mathcal{W}_0$.

2. **PreGen**: Preparation of the generalization process. This is performed by (1) scanning the initial working relation $\mathcal{W}_0$ once and collecting the distinct values for each attribute a_i and the number of occurrences of each distinct value in $\mathcal{W}_0$, (2) computing the minimum desired level L_i for each attribute a_i based on its given or default attribute threshold T_i, as explained further in the following paragraph, and (3) determining the mapping-pairs (v, v') for each attribute a_i in $\mathcal{W}_0$, where v is a distinct value of a_i in $\mathcal{W}_0$, and v' is its corresponding generalized value at level L_i.

   Notice that the minimum desirable level L_i of a_i is determined based on a sequence of Gen operators and/or the available concept hierarchy so that all of the distinct values for attribute a_i in $\mathcal{W}_0$ can be generalized to a small number τ of distinct generalized concepts, where τ is the largest possible number of distinct generalized values of a_i in $\mathcal{W}_0$ at a level of concept hierarchy which is no greater than the attribute threshold of a_i. Notice that a concept hierarchy, if given, can be adjusted or refined dynamically, or, if not given, may be generated dynamically based on data distribution statistics, as discussed in Chapter 3.

3. **PrimeGen**: Derivation of the **prime generalized relation**, $\mathcal{R}_p$. This is done by (1) replacing each value v in a_i of $\mathcal{W}_0$ with its corresponding ancestor concept v' determined at the $PreGen$ stage; and (2) merging identical tuples in the working relation. This involves accumulating the `count` information and computing any other aggregate values for the resulting tuples. The resulting relation is $\mathcal{R}_p$.

   This step can be efficiently implemented in two variations: (1) For each generalized tuple, insert the tuple into a sorted prime relation $\mathcal{R}_p$ by a binary search: if the tuple is already in $\mathcal{R}_p$, simply increase its count and other aggregate values accordingly; otherwise, insert it into $\mathcal{R}_p$. (2) Since in most cases the number of distinct values at the prime relation level is small, the prime relation can be coded as an m-dimensional array where m is the number of attributes in $\mathcal{R}_p$, and each dimension contains the corresponding generalized attribute values. Each array element holds the corresponding count and other aggregation values, if any. The insertion of a generalized tuple is performed by measure aggregation in the corresponding array element.

4. **Presentation**: Presentation of the derived generalization.

   - Determine whether the generalization is to be presented at the abstraction level of the prime relation, or if further enforcement of the relation generalization threshold is desired. In the latter case, further generalization is performed on $\mathcal{R}_p$ by selecting attributes for further generalization. (This can be performed by either interactive drilling or presetting some preference standard for such a selection). This generalization process continues until the number of distinct generalized tuples is no greater than T. This derives the final generalized relation $\mathcal{R}_f$.
   - Multiple forms can be selected for visualization of the output relation. These include a (1) generalized relation, (2) crosstab, (3) bar chart, pie chart, or curve, and (4) quantitative characteristic rule. □

"How efficient is this algorithm?"

Let's examine its computational complexity. Step 1 of the algorithm is essentially a relational query whose processing efficiency depends on the query processing methods used. With the successful implementation and commercialization of numerous database systems, this step is expected to have good performance.

For Steps 2 & 3, the collection of the statistics of the initial working relation $\mathcal{W}_0$ scans the relation only once. The cost for computing the minimum desired level and determining the mapping pairs (v, v') for each attribute is dependent on the number of distinct values for each attribute and is smaller than n, the number of tuples in the initial relation. The derivation of the prime relation $\mathcal{R}_p$ is performed by inserting generalized tuples into the prime relation. There are a total of n tuples in $\mathcal{W}_0$ and p tuples in $\mathcal{R}_p$. For each tuple t in $\mathcal{W}_0$, substitute its attribute values based on the derived mapping-pairs. This results in a generalized tuple t'. If variation (1) is adopted, each t' takes $O(\log p)$ to find the location for count incrementation or tuple insertion. Thus the total time complexity is $O(n \times \log p)$ for all of the generalized tuples. If variation (2) is adopted, each t' takes $O(1)$ to find the tuple for count incrementation. Thus the overall time complexity is $O(n)$ for all of the generalized tuples. (Note that the total array size could be quite large if the array is sparse). Therefore, the worst case time complexity should be $O(n \times \log p)$ if the prime relation is structured as a sorted relation, or $O(n)$ if the prime relation is structured as a m-dimensional array, and the array size is reasonably small.

Finally, since Step 4 for visualization works on a much smaller generalized relation, Algorithm 5.3.1 is efficient based on this complexity analysis.

5.3.2 Data cube implementation of attribute-oriented induction

Section 5.3.1 presented a database implementation of attribute-oriented induction based on a descriptive data mining query. This implementation, though efficient, has some limitations.

First, the power of drill-down analysis is limited. Algorithm 5.3.1 generalizes its task-relevant data from the database primitive concept level to the prime relation level *in a single step*. This is efficient. However, it facilitates only the roll up operation from the prime relation level, and the drill down operation from some higher abstraction level to the prime relation level. It cannot drill from the prime relation level down to any lower level because the system saves only the prime relation and the initial task-relevant data relation, but nothing in between. Further drilling-down from the prime relation level has to be performed by proper generalization from the initial task-relevant data relation.

Second, the generalization in Algorithm 5.3.1 is initiated by a data mining query. That is, no precomputation is performed before a query is submitted. The performance of such query-triggered processing is acceptable for a query whose relevant set of data is not very large, e.g., in the order of a few mega-bytes. If the relevant set of data is large, as in the order of many giga-bytes, the on-line computation could be costly and time-consuming. In such cases, it is recommended to perform precomputation using data cube or relational OLAP structures, as described in Chapter 2.

Moreover, many data analysis tasks need to examine a good number of dimensions or attributes. For example, an interactive data mining system may *dynamically* introduce and test additional attributes rather than just those specified in the mining query. Advanced descriptive data mining tasks, such as *analytical characterization* (to be discussed in Section 5.4), require attribute relevance analysis for a large set of attributes. Furthermore, a user with little knowledge of the *truly* relevant set of data may simply specify "in relevance to $*$" in the mining query. In these cases, the precomputation of aggregation values will speed up the analysis of a large number of dimensions or attributes.

The data cube implementation of attribute-oriented induction can be performed in two ways.

- **Construct a data cube on-the-fly for the given data mining query**: The first method constructs a data cube dynamically based on the task-relevant set of data. This is desirable if either the task-relevant data set is too specific to match any predefined data cube, or it is not very large. Since such a data cube is computed only after the query is submitted, the major motivation for constructing such a data cube is to facilitate efficient drill-down analysis. With such a data cube, drilling-down below the level of the prime relation will simply require retrieving data from the cube, or performing minor generalization from some intermediate level data stored in the cube instead of generalization from the primitive level data. This will speed up the drill-down process. However, since the attribute-oriented data generalization involves the computation of a query-related data cube, it may involve more processing than simple computation of the prime relation and thus increase the response time. A balance between the two may be struck by computing a cube-structured "subprime" relation in which each dimension of the generalized relation is a few levels deeper than the level of the prime relation. This will facilitate drilling-down to these levels with a reasonable storage and processing cost, although further drilling-down beyond these levels will still require generalization from the primitive level data. Notice that such further drilling-down is more likely to be localized, rather than spread out over the full spectrum of the cube.

- **Use a predefined data cube**: The second alternative is to construct a data cube before a data mining query is posed to the system, and use this predefined cube for subsequent data mining. This is desirable if the granularity of the task-relevant data can match that of the predefined data cube and the set of task-relevant data is quite large. Since such a data cube is precomputed, it facilitates attribute relevance analysis, attribute-oriented induction, dicing and slicing, roll-up, and drill-down. The cost one must pay is the cost of cube computation and the nontrivial storage overhead. A balance between the computation/storage overheads and the accessing speed may be attained by precomputing a selected set of all of the possible materializable cuboids, as explored in Chapter 2.

5.4 Analytical characterization: Analysis of attribute relevance

5.4.1 Why perform attribute relevance analysis?

The first limitation of class characterization for multidimensional data analysis in data warehouses and OLAP tools is the handling of complex objects. This was discussed in Section 5.2. The second limitation is the *lack of an automated generalization process*: the user must explicitly tell the system which dimensions should be included in the class characterization and to how high a level each dimension should be generalized. Actually, each step of generalization or specialization on any dimension must be specified by the user.

Usually, it is not difficult for a user to instruct a data mining system regarding how high a level each dimension should be generalized. For example, users can set attribute generalization thresholds for this, or specify which level a given dimension should reach, such as with the command "generalize dimension *location* to the *country* level". Even without explicit user instruction, a default value such as 2 to 8 can be set by the data mining system, which would allow each dimension to be generalized to a level that contains only 2 to 8 distinct values. If the user is not satisfied with the current level of generalization, she can specify dimensions on which drill-down or roll-up operations should be applied.

However, it is nontrivial for users to determine which dimensions should be included in the analysis of class characteristics. Data relations often contain 50 to 100 attributes, and a user may have little knowledge regarding which attributes or dimensions should be selected for effective data mining. A user may include too few attributes in the analysis, causing the resulting mined descriptions to be incomplete or incomprehensive. On the other hand, a user may introduce too many attributes for analysis (e.g., by indicating "in relevance to $*$", which includes all the attributes in the specified relations).

Methods should be introduced to perform attribute (or dimension) relevance analysis in order to filter out statistically irrelevant or weakly relevant attributes, and retain or even rank the most relevant attributes for the descriptive mining task at hand. Class characterization which includes the analysis of attribute/dimension relevance is called **analytical characterization**. Class comparison which includes such analysis is called **analytical comparison**.

Intuitively, an attribute or dimension is considered *highly relevant* with respect to a given class if it is likely that the values of the attribute or dimension may be used to distinguish the class from others. For example, it is unlikely that the color of an automobile can be used to distinguish expensive from cheap cars, but the model, make, style, and number of cylinders are likely to be more relevant attributes. Moreover, even within the same dimension, different

levels of concepts may have dramatically different powers for distinguishing a class from others. For example, in the *birth_date* dimension, *birth_day* and *birth_month* are unlikely relevant to the *salary* of employees. However, the *birth_decade* (i.e., age interval) may be highly relevant to the *salary* of employees. This implies that the analysis of dimension relevance should be performed at *multilevels of abstraction*, and only the most relevant levels of a dimension should be included in the analysis.

Above we said that attribute/dimension relevance is evaluated based on the ability of the attribute/dimension to distinguish objects of a class from others. When mining a class comparison (or discrimination), the target class and the contrasting classes are explicitly given in the mining query. The relevance analysis should be performed by comparison of these classes, as we shall see below. However, when mining class characteristics, there is only one class to be characterized. That is, no contrasting class is specified. It is therefore not obvious what the contrasting class to be used in the relevance analysis should be. In this case, typically, the contrasting class is taken to be the *set of comparable data in the database which excludes the set of the data to be characterized*. For example, to characterize graduate students, the contrasting class is composed of the set of students who are registered but are not graduate students.

5.4.2 Methods of attribute relevance analysis

There have been many studies in machine learning, statistics, fuzzy and rough set theories, etc. on attribute relevance analysis. The general idea behind attribute relevance analysis is to compute some measure which is used to quantify the relevance of an attribute with respect to a given class. Such measures include the information gain, Gini index, uncertainty, and correlation coefficients.

Here we introduce a method which integrates an information gain analysis technique (such as that presented in the ID3 and C4.5 algorithms for learning decision trees[2]) with a dimension-based data analysis method. The resulting method removes the less informative attributes, collecting the more informative ones for use in class description analysis.

We first examine the **information-theoretic approach** applied to the analysis of attribute relevance. Let's take ID3 as an example. ID3 constructs a decision tree based on a given set of data tuples, or *training objects*, where the class label of each tuple is known. The decision tree can then be used to classify objects for which the class label is not known. To build the tree, ID3 uses a measure known as **information gain** to rank each attribute. The attribute with the highest information gain is considered the most discriminating attribute of the given set. A tree node is constructed to represent a test on the attribute. Branches are grown from the test node according to each of the possible values of the attribute, and the given training objects are partitioned accordingly. In general, a node containing objects which all belong to the same class becomes a *leaf node* and is labeled with the class. The procedure is repeated recursively on each non-leaf partition of objects, until no more leaves can be created. This attribute selection process minimizes the expected number of tests to classify an object. When performing descriptive mining, we can use the information gain measure to perform relevance analysis, as we shall show below.

"How does the information gain calculation work?" Let S be a set of training objects where the class label of each object is known. (Each object is in fact a tuple. One attribute is used to determine the class of the objects). Suppose that there are m classes. Let S contain s_i objects of class C_i, for $i = 1, \ldots, m$. An arbitrary object belongs to class C_i with probability s_i/s, where s is the total number of objects in set S. When a decision tree is used to classify an object, it returns a class. A decision tree can thus be regarded as a source of messages for C_i's with the expected information needed to generate this message given by

$$I(s_1, s_2, \ldots, s_m) \quad = \quad \Leftrightarrow \sum_{i=1}^{m} \frac{s_i}{s} log_2 \frac{s_i}{s}. \tag{5.4}$$

If an attribute A with values $\{a_1, a_2, \cdots, a_v\}$ is used as the test at the root of the decision tree, it will partition S into the subsets $\{S_1, S_2, \cdots, S_v\}$, where S_j contains those objects in S that have value a_j of A. Let S_j contain s_{ij} objects of class C_i. The expected information based on this partitioning by A is known as the *entropy* of A. It is the

[2] A **decision tree** is a flow-chart-like tree structure, where each node denotes a test on an attribute, each branch represents an outcome of the test, and tree leaves represent classes or class distributions. Decision trees are useful for classification, and can easily be converted to logic rules. Decision tree induction is described in Chapter 7.

weighted average:

$$E(A) = \sum_{j=1}^{v} \frac{s_{1j} + \cdots + s_{mj}}{s} I(s_{1j}, \ldots, s_{mj}). \tag{5.5}$$

The information gained by branching on A is defined by:

$$Gain(A) \quad = \quad I(s_1, s_2, \ldots, s_m) \Leftrightarrow E(A). \tag{5.6}$$

ID3 computes the information gain for each of the attributes defining the objects in S. The attribute which maximizes $Gain(A)$ is selected, a tree root node to test this attribute is created, and the objects in S are distributed accordingly into the subsets $S_1, S_2, \cdots, S_m$. ID3 uses this process recursively on each subset in order to form a decision tree.

Notice that class characterization is different from the decision tree-based classification analysis. The former identifies a set of informative attributes for class characterization, summarization and comparison, whereas the latter constructs a model in the form of a decision tree for classification of unknown data (i.e., data whose class label is not known) in the future. Therefore, for the purpose of class description, only the attribute relevance analysis step of the decision tree construction process is performed. That is, rather than constructing a decision tree, we will use the information gain measure to rank and select the attributes to be used in class description.

Attribute relevance analysis for class description is performed as follows.

1. Collect data for both the target class and the contrasting class by query processing.

   Notice that for class comparison, both the *target class* and the *contrasting class* are provided by the user in the data mining query. For class characterization, the *target class* is the class to be characterized, whereas the *contrasting class* is the set of comparable data which are not in the target class.

2. Identify a set of dimensions and attributes on which the relevance analysis is to be performed.

   Since different levels of a dimension may have dramatically different relevance with respect to a given class, each attribute defining the conceptual levels of the dimension should be included in the relevance analysis in principle. However, although attributes having a very large number of distinct values (such as *name* and *phone#*) may return nontrivial relevance measure values, they are unlikely to be meaningful for concept description. Thus, such attributes should first be removed or generalized before attribute relevance analysis is performed. Therefore, only the dimensions and attributes remaining after attribute removal and attribute generalization should be included in the relevance analysis. The thresholds used for attributes in this step are called the **attribute analytical thresholds**. To be conservative in this step, note that the attribute analytical threshold should be set reasonably large so as to allow more attributes to be considered in the relevance analysis. The relation obtained by such an attribute removal and attribute generalization process is called the **candidate relation** of the mining task.

3. Perform relevance analysis for each attribute in the candidation relation.

   The relevance measure used in this step may be built into the data mining system, or provided by the user (depending on whether the system is flexible enough to allow users to define their own relevance measurements). For example, the information gain measure described above may be used. The attributes are then sorted (i.e., ranked) according to their computed relevance to the data mining task.

4. Remove from the candidate relation the attributes which are not relevant or are weakly relevant to the class description task.

   A threshold may be set to define "weakly relevant". This step results in an **initial target class working relation** and an **initial contrasting class working relation**.

   If the class description task is class characterization, only the initial target class working relation will be included in further analysis. If the class description task is class comparison, both the initial target class working relation and the initial contrasting class working relation will be included in further analysis.

The above discussion is summarized in the following algorithm for analytical characterization in relational databases.

Algorithm 5.4.1 (Analytical characterization) *Mining class characteristic descriptions by performing both attribute relevance analysis and class characterization.*

Input. 1. A mining task for characterization of a specified set of data from a relational database,
2. $Gen(a_i)$, a set of concept hierarchies or generalization operators on attributes a_i,
3. U_i, a set of *attribute analytical thresholds* for attributes a_i,
4. T_i, a set of *attribute generalization thresholds* for attributes a_i, and
5. R, an *attribute relevance threshold*.

Output. Class characterization presented in user-specified visualization formats.

Method. 1. **Data collection**: Collect data for both the target class and the contrasting class by query processing, where the *target class* is the class to be characterized, and the *contrasting class* is the set of comparable data which are in the database but are not in the target class.
2. **Analytical generalization**: Perform attribute removal and attribute generalization based on the set of provided *attribute analytical thresholds*, U_i. That is, if the attribute contains many distinct values, it should be either removed or generalized to satisfy the thresholds. This process identifies the set of attributes on which the relevance analysis is to be performed. The resulting relation is the *candidate relation*.
3. **Relevance analysis**: Perform relevance analysis for each attribute of the *candidate relation* using the specified relevance measurement. The attributes are ranked according to their computed relevance to the data mining task.
4. **Initial working relation derivation**: Remove from the *candidate relation* the attributes which are not relevant or are weakly relevant to the class description task, based on the *attribute relevance threshold*, R. Then remove the contrasting class. The result is called the *initial (target class) working relation*.
5. **Induction on the initial working relation**: Perform attribute-oriented induction according to Algorithm 5.3.1, using the *attribute generalization thresholds*, T_i. □

Since the algorithm is derived following the reasoning provided before the algorithm, its correctness can be proved accordingly. The complexity of the algorithm is similar to the attribute-oriented induction algorithm since the induction process is performed twice in both analytical generalization (Step 2) and induction on the initial working relation (Step 5). Relevance analysis (Step 3) is performed by scanning through the database once to derive the probability distribution for each attribute.

5.4.3 Analytical characterization: An example

If the mined class descriptions involve many attributes, analytical characterization should be performed. This procedure first removes irrelevant or weakly relevant attributes prior to performing generalization. Let's examine an example of such an analytical mining process.

Example 5.9 Suppose that we would like to mine the general characteristics describing graduate students at *Big-University* using analytical characterization. Given are the attributes *name, gender, major, birth_place, birth_date, phone#*, and *gpa*.

"How is the analytical characterization performed?"

1. In Step 1, the target class data are collected, consisting of the set of graduate students. Data for a contrasting class are also required in order to perform relevance analysis. This is taken to be the set of undergraduate students.

2. In Step 2, analytical generalization is performed in the form of attribute removal and attribute generalization. Similar to Example 5.3, the attributes *name* and *phone#* are removed because their number of distinct values exceeds their respective attribute analytical thresholds. Also as in Example 5.3, concept hierarchies are used to generalize *birth_place* to *birth_country*, and *birth_date* to *age_range*. The attributes *major* and *gpa* are also generalized to higher abstraction levels using the concept hierarchies described in Example 5.3. Hence, the attributes remaining for the candidate relation are *gender, major, birth_country, age_range*, and *gpa*. The resulting relation is shown in Table 5.5.

gender	major	birth_country	age_range	gpa	count
M	Science	Canada	20-25	very_good	16
F	Science	Foreign	25-30	excellent	22
M	Engineering	Foreign	25-30	excellent	18
F	Science	Foreign	25-30	excellent	25
M	Science	Canada	20-25	excellent	21
F	Engineering	Canada	20-25	excellent	18

Target class: Graduate students

gender	major	birth_country	age_range	gpa	count
M	Science	Foreign	<20	very_good	18
F	Business	Canada	<20	fair	20
M	Business	Canada	<20	fair	22
F	Science	Canada	20-25	fair	24
M	Engineering	Foreign	20-25	very_good	22
F	Engineering	Canada	< 20	excellent	24

Contrasting class: Undergraduate students

Table 5.5: Candidate relation obtained for analytical characterization: the target class and the contrasting class.

3. In Step 3, relevance analysis is performed on the attributes in the candidate relation. Let C_1 correspond to the class *graduate* and class C_2 correspond to *undergraduate*. There are 120 samples of class *graduate* and 130 samples of class *undergraduate*. To compute the information gain of each attribute, we first use Equation (5.4) to compute the expected information needed to classify a given sample. This is:

$$I(s_1, s_2) = I(120, 130) = \Leftrightarrow\frac{120}{250}\log_2\frac{120}{250} \Leftrightarrow\frac{130}{250}\log_2\frac{130}{250} = 0.9988$$

Next, we need to compute the entropy of each attribute. Let's try the attribute *major*. We need to look at the distribution of *graduate* and *undergraduate* students for each value of *major*. We compute the expected information for each of these distributions.

for *major* = *"Science"*:	$s_{11} = 84$	$s_{21} = 42$	$I(s_{11}, s_{21}) = 0.9183$
for *major* = *"Engineering"*:	$s_{12} = 36$	$s_{22} = 46$	$I(s_{12}, s_{22}) = 0.9892$
for *major* = *"Business"*:	$s_{13} = 0$	$s_{23} = 42$	$I(s_{13}, s_{23}) = 0$

Using Equation (5.5), the expected information needed to classify a given sample if the samples are partitioned according to *major*, is:

$$E(major) = \frac{126}{250}I(s_{11}, s_{21}) + \frac{82}{250}I(s_{12}, s_{22}) + \frac{42}{250}I(s_{13}, s_{23}) = 0.7873$$

Hence, the gain in information from such a partitioning would be:

$$Gain(age) = I(s_1, s_2) \Leftrightarrow E(major) = 0.2115$$

Similarly, we can compute the information gain for each of the remaining attributes. The information gain for each attribute, sorted in increasing order, is : 0.0003 for *gender*, 0.0407 for *birth_country*, 0.2115 for *major*, 0.4490 for *gpa*, and 0.5971 for *age_range*.

4. In Step 4, suppose that we use an attribute relevance threshold of 0.1 to identify weakly relevant attributes. The information gain of the attributes *gender* and *birth_country* are below the threshold, and therefore considered weakly relevant. Thus, they are removed. The contrasting class is also removed, resulting in the initial target class working relation.

5. In Step 5, attribute-oriented induction is applied to the initial target class working relation, following Algorithm 5.3.1.

□

5.5 Mining class comparisons: Discriminating between different classes

In many applications, one may not be interested in having a single class (or concept) described or characterized, but rather would prefer to mine a description which compares or distinguishes one class (or concept) from other comparable classes (or concepts). Class discrimination or comparison (hereafter referred to as **class comparison**) mines descriptions which distinguish a target class from its contrasting classes. Notice that the target and contrasting classes must be *comparable* in the sense that they share similar dimensions and attributes. For example, the three classes *person, address*, and *item* are not comparable. However, the sales in the last three years are comparable classes, and so are computer science students versus physics students.

Our discussions on class characterization in the previous several sections handle multilevel data summarization and characterization in a single class. The techniques developed should be able to be extended to handle class comparison across several comparable classes. For example, attribute generalization is an interesting method used in class characterization. When handling multiple classes, attribute generalization is still a valuable technique. However, for effective comparison, the generalization should be performed *synchronously* among all the classes compared so that the attributes in all of the classes can be generalized to the same levels of abstraction. For example, suppose we are given the *AllElectronics* data for sales in 1999 and sales in 1998, and would like to compare these two classes. Consider the dimension *location* with abstractions at the *city*, *province_or_state*, and *country* levels. Each class of data should be generalized to the same *location* level. That is, they are synchronously all generalized to either the *city* level, or the *province_or_state* level, or the *country* level. Ideally, this is more useful than comparing, say, the sales in Vancouver in 1998 with the sales in U.S.A. in 1999 (i.e., where each set of sales data are generalized to different levels). The users, however, should have the option to over-write such an automated, synchronous comparison with their own choices, when preferred.

5.5.1 Class comparison methods and implementations

"How is class comparison performed?"

In general, the procedure is as follows.

1. **Data collection**: The set of relevant data in the database is collected by query processing and is partitioned respectively into a *target class* and one or a set of *contrasting class(es)*.
2. **Dimension relevance analysis**: If there are many dimensions and *analytical class comparison* is desired, then dimension relevance analysis should be performed on these classes as described in Section 5.4, and only the highly relevant dimensions are included in the further analysis.
3. **Synchronous generalization**: Generalization is performed on the target class to the level controlled by a user- or expert-specified dimension threshold, which results in a **prime target class relation/cuboid**. The concepts in the contrasting class(es) are generalized to the same level as those in the prime target class relation/cuboid, forming the **prime contrasting class(es) relation/cuboid**.
4. **Drilling down, rolling up, and other OLAP adjustment**: Synchronous or asynchronous (when such an option is allowed) drill-down, roll-up, and other OLAP operations, such as dicing, slicing, and pivoting, can be performed on the target and contrasting classes based on the user's instructions.
5. **Presentation of the derived comparison**: The resulting class comparison description can be visualized in the form of tables, graphs, and rules. This presentation usually includes a "contrasting" measure (such as `count%`) which reflects the comparison between the target and contrasting classes.

The above discussion outlines a general algorithm for mining analytical class comparisons in databases. In comparison with Algorithm 5.4.1 which mines analytical class characterization, the above algorithm involves synchronous generalization of the target class with the contrasting classes so that classes are simultaneously compared at the *same* levels of abstraction.

"Can class comparison mining be implemented efficiently using data cube techniques?" Yes — the procedure is similar to the implementation for mining data characterizations discussed in Section 5.3.2. A flag can be used to indicate whether or not a tuple represents a target or contrasting class, where this flag is viewed as an additional dimension in the data cube. Since all of the other dimensions of the target and contrasting classes share the same

portion of the cube, the synchronous generalization and specialization are realized automatically by rolling up and drilling down in the cube.

Let's study an example of mining a class comparison describing the graduate students and the undergraduate students at *Big-University*.

Example 5.10 Mining a class comparison. Suppose that you would like to compare the general properties between the graduate students and the undergraduate students at *Big-University*, given the attributes *name, gender, major, birth_place, birth_date, residence, phone#*, and *gpa (grade_point_average)*.

This data mining task can be expressed in DMQL as follows.

```
use Big_University_DB
mine comparison as "grad_vs_undergrad_students"
in relevance to name, gender, major, birth_place, birth_date, residence, phone#, gpa
for "graduate_students"
where status in "graduate"
versus "undergraduate_students"
where status in "undergraduate"
analyze count%
from student
```

Let's see how this typical example of a data mining query for mining comparison descriptions can be processed.

name	gender	major	birth_place	birth_date	residence	phone#	gpa
Jim Woodman	M	CS	Vancouver, BC, Canada	8-12-76	3511 Main St., Richmond	687-4598	3.67
Scott Lachance	M	CS	Montreal, Que, Canada	28-7-75	345 1st Ave., Vancouver	253-9106	3.70
Laura Lee	F	Physics	Seattle, WA, USA	25-8-70	125 Austin Ave., Burnaby	420-5232	3.83
...	...	...	...	...	...	...	...

Target class: Graduate students

name	gender	major	birth_place	birth_date	residence	phone#	gpa
Bob Schumann	M	Chemistry	Calgary, Alt, Canada	10-1-78	2642 Halifax St., Burnaby	294-4291	2.96
Amy Eau	F	Biology	Golden, BC, Canada	30-3-76	463 Sunset Cres., Vancouver	681-5417	3.52
...	...	...	...	...	...	...	...

Contrasting class: Undergraduate students

Table 5.6: Initial working relations: the target class vs. the contrasting class.

1. First, the query is transformed into two relational queries which collect two sets of task-relevant data: one for the initial target class working relation, and the other for the initial contrasting class working relation, as shown in Table 5.6. This can also be viewed as the construction of a data cube, where the status {*graduate, undergraduate*} serves as one dimension, and the other attributes form the remaining dimensions.

2. Second, dimension relevance analysis is performed on the two classes of data. After this analysis, irrelevant or weakly relevant dimensions, such as *name, gender, major*, and *phone#* are removed from the resulting classes. Only the highly relevant attributes are included in the subsequent analysis.

3. Third, synchronous generalization is performed: Generalization is performed on the target class to the levels controlled by user- or expert-specified dimension thresholds, forming the *prime target class relation/cuboid*. The contrasting class is generalized to the same levels as those in the prime target class relation/cuboid, forming the *prime contrasting class(es) relation/cuboid*, as presented in Table 5.7. The table shows that in comparison with undergraduate students, graduate students tend to be older and have a higher GPA, in general.

4. Fourth, drilling and other OLAP adjustment are performed on the target and contrasting classes, based on the user's instructions to adjust the levels of abstractions of the resulting description, as necessary.

birth_country	age_range	gpa	count%
Canada	20-25	good	5.53%
Canada	25-30	good	2.32%
Canada	over_30	very_good	5.86%
...	...	...	...
other	over_30	excellent	4.68%

Prime generalized relation for the target class: Graduate students

birth_country	age_range	gpa	count%
Canada	15-20	fair	5.53%
Canada	15-20	good	4.53%
...	...	...	...
Canada	25-30	good	5.02%
...	...	...	...
other	over_30	excellent	0.68%

Prime generalized relation for the contrasting class: Undergraduate students

Table 5.7: Two generalized relations: the prime target class relation and the prime contrasting class relation.

5. Finally, the resulting class comparison is presented in the form of tables, graphs, and/or rules. This visualization includes a contrasting measure (such as `count%`) which compares between the target class and the contrasting class. For example, only 2.32% of the graduate students were born in Canada, are between 25-30 years of age, and have a "good" GPA, while 5.02% of undergraduates have these same characteristics.

□

5.5.2 Presentation of class comparison descriptions

"How can class comparison descriptions be visualized?"

As with class characterizations, class comparisons can be presented to the user in various kinds of forms, including generalized relations, crosstabs, bar charts, pie charts, curves, and rules. With the exception of logic rules, these forms are used in the same way for characterization as for comparison. In this section, we discuss the visualization of class comparisons in the form of discriminant rules.

As is similar with characterization descriptions, the discriminative features of the target and contrasting classes of a comparison description can be described quantitatively by a *quantitative discriminant rule*, which associates a statistical interestingness measure, *d-weight*, with each generalized tuple in the description.

Let q_a be a generalized tuple, and C_j be the target class, where q_a covers some tuples of the target class. Note that it is possible that q_a also covers some tuples of the contrasting classes, particularly since we are dealing with a comparison description. The **d-weight** for q_a is the ratio of the number of tuples from the initial target class working relation that are covered by q_a to the total number of tuples in both the initial target class and contrasting class working relations that are covered by q_a. Formally, the d-weight of q_a for the class C_j is defined as

$$d\_weight = count(q_a \in C_j)/\Sigma_{i=1}^{m} count(q_a \in C_i), \tag{5.7}$$

where m is the total number of the target and contrasting classes, C_j is in $\{C_1, \ldots, C_m\}$, and $count(q_a \in C_i)$ is the number of tuples of class C_i that are covered by q_a. The range for the d-weight is [0, 1] (or [0%, 100%]).

A high d-weight in the target class indicates that the concept represented by the generalized tuple is primarily derived from the target class, whereas a low d-weight implies that the concept is primarily derived from the contrasting classes.

Example 5.11 In Example 5.10, suppose that the count distribution for the generalized tuple, "*birth_country = "Canada" and age_range = "25-30" and gpa = "good"*" from Table 5.7 is as shown in Table 5.8.

The d-weight for the given generalized tuple is 90/(90 + 210) = 30% with respect to the target class, and 210/(90 + 210) = 70% with respect to the contrasting class. That is, *if a student was born in Canada, is in the age range*

status	birth_country	age_range	gpa	count
graduate	Canada	25-30	good	90
undergraduate	Canada	25-30	good	210

Table 5.8: Count distribution between graduate and undergraduate students for a generalized tuple.

of [25, 30), and has a "good" gpa, then based on the data, there is a 30% probability that she is a graduate student, versus a 70% probability that she is an undergraduate student. Similarly, the d-weights for the other generalized tuples in Table 5.7 can be derived. □

A **quantitative discriminant rule** for the target class of a given comparison description is written in the form

$$\forall X,\ target\_class(X) \Leftarrow condition(X) \quad [d : d\_weight], \tag{5.8}$$

where the condition is formed by a generalized tuple of the description. This is different from rules obtained in class characterization where the arrow of implication is from left to right.

Example 5.12 Based on the generalized tuple and count distribution in Example 5.11, a quantitative discriminant rule for the target class *graduate_student* can be written as follows:

$$\forall X,\ graduate\_student(X) \Leftarrow birth\_country(X) = \text{“}Canada\text{”} \wedge age\_range = \text{“}25\_30\text{”} \wedge gpa = \text{“}good\text{”}[d : 30\%]. \tag{5.9}$$

□

Notice that a discriminant rule provides a *sufficient* condition, but not a *necessary* one, for an object (or tuple) to be in the target class. For example, Rule (5.9) implies that if X satisfies the condition, then the probability that X is a graduate student is 30%. However, it does not imply the probability that X meets the condition, given that X is a graduate student. This is because although the tuples which meet the condition are in the target class, other tuples that do not necessarily satisfy this condition may also be in the target class, since the rule may not cover *all* of the examples of the target class in the database. Therefore, the condition is sufficient, but not necessary.

5.5.3 Class description: Presentation of both characterization and comparison

"Since class characterization and class comparison are two aspects forming a class description, can we present both in the same table or in the same rule?"

Actually, as long as we have a clear understanding of the meaning of the t-weight and d-weight measures and can interpret them correctly, there is no additional difficulty in presenting both aspects in the same table. Let's examine an example of expressing both class characterization and class discrimination in the same crosstab.

Example 5.13 Let Table 5.9 be a crosstab showing the total number (in thousands) of TVs and computers sold at *AllElectronics* in 1998.

location \ item	TV	computer	*both_items*
Europe	80	240	320
North_America	120	560	680
both_regions	200	800	1000

Table 5.9: A crosstab for the total number (*count*) of TVs and computers sold in thousands in 1998.

Let *Europe* be the target class and *North_America* be the contrasting class. The t-weights and d-weights of the sales distribution between the two classes are presented in Table 5.10. According to the table, the t-weight of a generalized tuple or object (e.g., the tuple '*item* = *"TV"*') for a given class (e.g. the target class *Europe*) shows how typical the tuple is of the given class (e.g., what proportion of these sales in Europe are for TVs?). The d-weight of

location \ item	TV			computer			*both_items*		
	count	t-weight	d-weight	count	t-weight	d-weight	count	t-weight	d-weight
Europe	80	25%	40%	240	75%	30%	320	100%	32%
North_America	120	17.65%	60%	560	82.35%	70%	680	100%	68%
both_regions	200	20%	100%	800	80%	100%	1000	100%	100%

Table 5.10: The same crosstab as in Table 4.8, but here the t-weight and d-weight values associated with each class are shown.

a tuple shows how distinctive the tuple is in the given (target or contrasting) class in comparison with its rival class (e.g., how do the TV sales in Europe compare with those in North America?).

For example, the t-weight for *(Europe, TV)* is 25% because the number of TVs sold in Europe (80 thousand) represents only 25% of the European sales for both items (320 thousand). The d-weight for *(Europe, TV)* is 40% because the number of TVs sold in Europe (80 thousand) represents 40% of the number of TVs sold in both the target and the contrasting classes of Europe and North America, respectively (which is 200 thousand). □

Notice that the count measure in the crosstab of Table 5.10 obeys the general property of a crosstab (i.e., the count values per row and per column, when totaled, match the corresponding totals in the *both_items* and *both_regions* slots, respectively, for *count*. However, this property is not observed by the t-weight and d-weight measures. This is because the semantic meaning of each of these measures is different from that of *count*, as we explained in Example 5.13.

"Can a quantitative characteristic rule and a quantitative discriminant rule be expressed together in the form of one rule?" The answer is yes – a quantitative characteristic rule and a quantitative discriminant rule for the same class can be combined to form a *quantitative description rule* for the class, which displays the t-weights *and* d-weights associated with the corresponding characteristic and discriminant rules. To see how this is done, let's quickly review how quantitative characteristic and discriminant rules are expressed.

- As discussed in Section 5.2.3, a quantitative characteristic rule provides a necessary condition for the given target class since it presents a probability measurement for each property which can occur in the target class. Such a rule is of the form

$$\forall X,\ target\_class(X) \ \Rightarrow\ condition_1(X)[t : w_1] \vee \cdots \vee condition_n(X)[t : w_n], \tag{5.10}$$

  where each condition represents a property of the target class. The rule indicates that if X is in the $target\_class$, the possibility that X satisfies $condition_i$ is the value of the t-weight, w_i, where i is in $\{1, \ldots, n\}$.

- As previously discussed in Section 5.5.1, a quantitative discriminant rule provides a sufficient condition for the target class since it presents a quantitative measurement of the properties which occur in the target class versus those that occur in the contrasting classes. Such a rule is of the form

$$\forall X,\ target\_class(X) \ \Leftarrow\ condition_1(X)[d : w_1] \vee \cdots \vee condition_n(X)[d : w_n].$$

  The rule indicates that if X satisfies $condition_i$, there is a possibility of w_i (the d-weight value) that x is in the $target\_class$, where i is in $\{1, \ldots, n\}$.

A quantitative characteristic rule and a quantitative discriminant rule for a given class can be combined as follows to form a **quantitative description rule**: (1) For each condition, show both the associated t-weight and d-weight; and (2) A bi-directional arrow should be used between the given class and the conditions. That is, a quantitative description rule is of the form

$$\forall X,\ target\_class(X) \Leftrightarrow condition_1(X)[t : w_1, d : w'_1] \vee \cdots \vee condition_n(X)[t : w_n, d : w'_n]. \tag{5.11}$$

This form indicates that for i from 1 to n, if X is in the $target\_class$, there is a possibility of w_i that X satisfies $condition_i$; and if X satisfies $condition_i$, there is a possibility of w'_i that X is in the $target\_class$.

Example 5.14 It is straightfoward to transform the crosstab of Table 5.10 in Example 5.13 into a class description in the form of quantitative description rules. For example, the quantitative description rule for the target class, *Europe*, is

$$\forall X,\ Europe(X) \Leftrightarrow (item(X) = \text{“}TV\text{”})\ [t : 25\%, d : 40\%] \vee (item(X) = \text{“}computer\text{”})\ [t : 75\%, d : 30\%] \quad (5.12)$$

The rule states that for the sales of TV's and computers at *AllElectronics* in 1998, if the sale of one of these items occurred in Europe, then the probability of the item being a TV is 25%, while that of being a computer is 75%. On the other hand, if we compare the sales of these items in Europe and North America, then 40% of the TV's were sold in Europe (and therefore we can deduce that 60% of the TV's were sold in North America). Furthermore, regarding computer sales, 30% of these sales took place in Europe. □

5.6 Mining descriptive statistical measures in large databases

Earlier in this chapter, we discussed class description in terms of popular measures, such as *count, sum*, and *average*. Relational database systems provide five built-in aggregate functions: `count()`, `sum()`, `avg()`, `max()`, and `min()`. These functions can also be computed efficiently (in incremental and distributed manners) in data cubes. Thus, there is no problem in including these aggregate functions as basic measures in the descriptive mining of multidimensional data.

However, for many data mining tasks, users would like to learn more data characteristics regarding both central tendency and data dispersion. Measures of central tendency include *mean, median, mode*, and *midrange*, while measures of data dispersion include *quartiles, outliers, variance*, and other statistical measures. These descriptive statistics are of great help in understanding the distribution of the data. Such measures have been studied extensively in the statistical literature. However, from the data mining point of view, we need to examine how they can be computed efficiently in large, multidimensional databases.

5.6.1 Measuring the central tendency

- The most common and most effective numerical 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. The **mean** of this set of values is

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i. \quad (5.13)$$

  This corresponds to the built-in aggregate function, *average* (`avg()` in SQL), provided in relational database systems. In most data cubes, *sum* and *count* are saved in precomputation. Thus, the derivation of *average* is straightforward, using the formula $average = sum/count$.

- 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

$$\bar{x} = \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i}. \quad (5.14)$$

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

  In Chapter 2, a measure was defined as *algebraic* if it can be computed from distributive aggregate measures. Since `avg()` can be computed by `sum()/count()`, where both `sum()` and `count()` are distributive aggregate measures in the sense that they can be computed in a distributive manner, then `avg()` is an algebraic measure. One can verify that the weighted average is also an algebraic measure.

- Although the mean is the single most useful quantity that we use to describe a set of data, it is not the only, or even always the best, way of measuring the center of a set of data. For skewed data, a better measure of center of data is the *median*, M. Suppose that the values forming a given set of data are in numerical order.

The **median** is *the middle value* of the ordered set if the number of values n is an odd number; otherwise (i.e., if n is even), it is *the average of the middle two values.*

Based on the categorization of measures in Chapter 2, the median is neither a distributive measure nor an algebraic measure — it is a holistic measure in the sense that it cannot be computed by partitioning a set of values arbitrarily into smaller subsets, computing their medians independently, and merging the median values of each subset. On the contrary, `count()`, `sum()`, `max()`, and `min()` can be computed in this manner (being distributive measures), and are therefore easier to compute than the median.

Although it is not easy to compute the exact median value in a large database, an approximate median can be computed efficiently. For example, for grouped data, the median, obtained by interpolation, is given by

$$median = L_1 + (\frac{n/2 + (\sum f)_l}{f_{median}})c. \tag{5.15}$$

where L_1 is the lower class boundary of (i.e., lowest value for) the class containing the median, n is the number of values in the data, $(\sum f)_l$ is the sum of the frequencies of all of the classes that are lower than the median class, and f_{median} is the frequency of the median class, and c is the size of the median class interval.

- Another measure of central tendency is the *mode*. The **mode** for a set of data is the value that occurs most frequently in the set. 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**. If a data set has more than three modes, it is **multimodal**. At the other extreme, if each data value occurs only once, then there is no mode.

  For unimodal frequency curves that are moderately skewed (asymmetrical), we have the following empirical relation

$$mean \Leftrightarrow mode = 3 \times (mean \Leftrightarrow median). \tag{5.16}$$

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

- The **midrange**, that is, *the average of the largest and smallest values in a data set*, can be used to measure the central tendency of the set of data. It is trivial to compute the midrange using the SQL aggregate functions, `max()` and `min()`.

5.6.2 Measuring the dispersion of data

The degree to which numeric data tend to spread is called the **dispersion**, or **variance** of the data. The most common measures of data dispersion are the *five-number summary* (based on *quartiles*), the *interquartile range*, and *standard deviation*. The plotting of *boxplots* (which show outlier values) also serves as a useful graphical method.

Quartiles, outliers and boxplots

- The **kth percentile** of a set of data in numerical order is the value x having the property that k percent of the data entries lies at or below x. Values at or below the *median* M (discussed in the previous subsection) correspond to the 50-th percentile.

- The most commonly used percentiles other than the median are **quartiles**. The **first quartile**, denoted by Q_1, is the 25-th percentile; and the **third quartile**, denoted by Q_3, is the 75-th percentile.

- The quartiles together with the median give some indication of the center, spread, and shape of a 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

$$IQR = Q_3 \Leftrightarrow Q_1. \tag{5.17}$$

  We should be aware that no single numerical measure of spread, such as IQR, is very useful for describing skewed distributions. The spreads of two sides of a skewed distribution are unequal. Therefore, it is more informative to also provide the two quartiles Q_1 and Q_3, along with the median, M.

unit price ($)	number of items sold
40	275
43	300
47	250
..	..
74	360
75	515
78	540
..	..
115	320
117	270
120	350

Table 5.11: A set of data.

- One 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, M, and Q_3 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 highest and lowest data values as well. This is known as the *five-number summary*. The **five-number summary** of a distribution consists of the median M, the quartiles Q_1 and Q_3, and the smallest and largest individual observations, written in the order

$$Minimum,\ Q_1,\ M,\ Q_3,\ Maximum.$$

- A popularly used visual representation of a distribution is the **boxplot**. In a boxplot:
  1. The ends of the box are at the quartiles, so that the box length is the interquartile range, IQR.
  2. The median is marked by a line within the box.
  3. Two lines (called *whiskers*) outside the box extend to the smallest (*Minimum*) and largest (*Maximum*) observations.

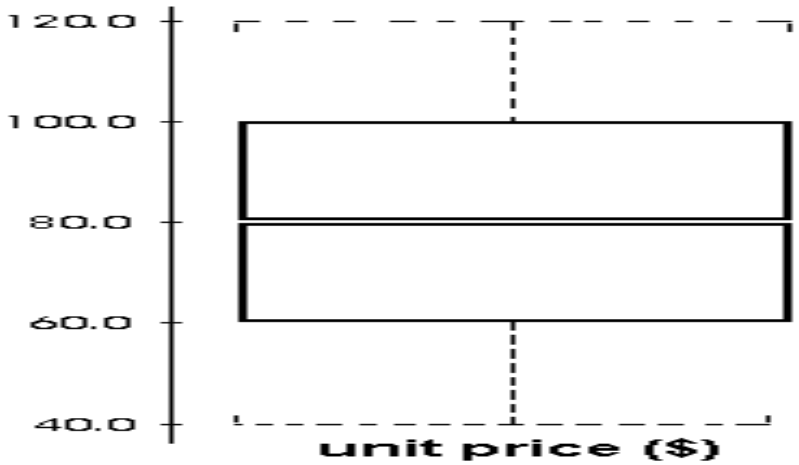


Figure 5.4: A boxplot for the data set of Table 5.11.

When dealing with a moderate numbers of observations, it is worthwhile to plot potential outliers individually. To do this in a boxplot, the whiskers are extended to the extreme high and low 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. Figure 5.4 shows a boxplot for the set of price data in Table 5.11, where we see that Q1 is \$60, Q3 is \$100, and the median is \$80.

Based on similar reasoning as in our analysis of the median in Section 5.6.1, we can conclude that Q_1 and Q_3 are holistic measures, as is IQR. The efficient computation of boxplots or even *approximate boxplots* is interesting regarding the mining of large data sets.

Variance and standard deviation

The **variance** of n observations $x_1, x_2, \ldots, x_n$ is

$$s^2 = \frac{1}{n \Leftrightarrow 1} \sum_{i=1}^{n} (x_i \Leftrightarrow \bar{x})^2 = \frac{1}{n \Leftrightarrow 1} [\sum x_i^2 \Leftrightarrow \frac{1}{n} (\sum x_i)^2] \tag{5.18}$$

The **standard deviation** s is the square root of the variance s^2.

The basic properties of the standard deviation s as a measure of spread are:

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

Notice that variance and standard deviation are algebraic measures because n (which is `count()` in SQL), $\sum x_i$ (which is the `sum()` of x_i), and $\sum x_i^2$ (which is the `sum()` of x_i^2) can be computed in any partition and then merged to feed into the algebraic equation (5.18). Thus the computation of the two measures is scalable in large databases.

5.6.3 Graph displays of basic statistical class descriptions

Aside from the bar charts, pie charts, and line graphs discussed earlier in this chapter, there are also a few additional popularly used graphs for the display of data summaries and distributions. These include *histograms, quantile plots, Q-Q plots, scatter plots*, and *loess curves*.

- A **histogram**, or **frequency histogram**, is a univariate graphical method. It denotes the frequencies of the classes present in a given set of data. A histogram consists of a set of rectangles where the *area* of each rectangle is proportional to the relative frequency of the class it represents. The base of each rectangle is on the horizontal axis, centered at a "class" mark, and the base length is equal to the class width. Typically, the class width is uniform, with classes being defined as the values of a categoric attribute, or equi-width ranges of a discretized continuous attribute. In these cases, the height of each rectangle is the relative frequency (or frequency) of the class it represents, and the histogram is generally referred to as a **bar chart**. Alternatively, classes for a continuous attribute may be defined by ranges of non-uniform width. In this case, for a given class, the class width is equal to the range width, and the height of the rectangle is the class density (that is, the relative frequency of the class, divided by the class width). Partitioning rules for constructing histograms were discussed in Chapter 3.

  Figure 5.5 shows a histogram for the data set of Table 5.11, where classes are defined by equi-width ranges representing \$10 increments. Histograms are at least a century old, and are a widely used univariate graphical method. However, they may not be as effective as the quantile plot, Q-Q plot and boxplot methods for comparing groups of univariate observations.

- A **quantile plot** is a simple and effective way to have a first look at data distribution. First, it displays all of the data (allowing the user to assess both the overall behavior and unusual occurrences). Second, it plots quantile information. The mechanism used in this step is slightly different from the percentile computation. Let $x_{(i)}$, for $i = 1$ to n, be the data ordered from the smallest to the largest; thus $x_{(1)}$ is the smallest observation and $x_{(n)}$ is the largest. Each observation $x_{(i)}$ is paired with a percentage, f_i, which indicates that $100 f_i\%$ of the data are below or equal to the value $x_{(i)}$. Let

$$f_i = \frac{i \Leftrightarrow 0.5}{n}.$$

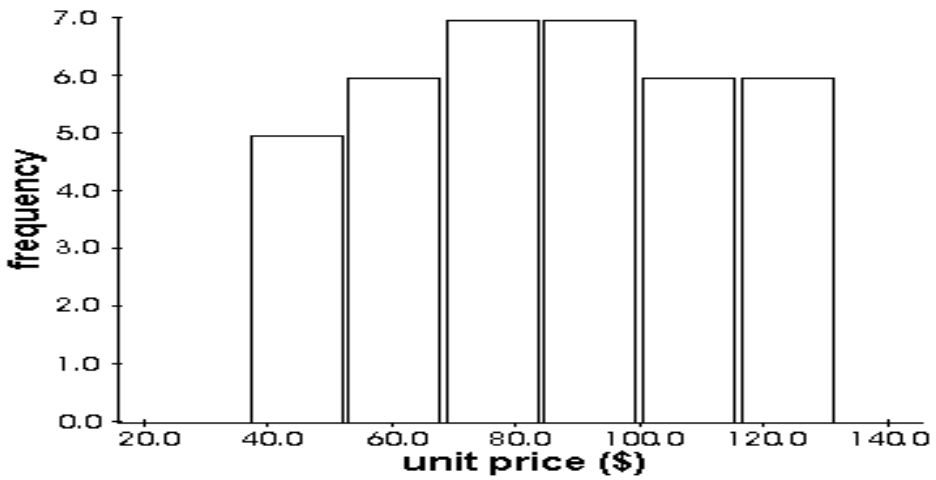


Figure 5.5: A histogram for the data set of Table 5.11.

These numbers increase in equal steps of $1/n$ beginning with $1/2n$, which is slightly above zero, and ending with $1 \Leftrightarrow 1/2n$, which is slightly below one. On a quantile plot, $x_{(i)}$ is graphed against f_i. This allows visualization of the f_i quantiles. Figure 5.6 shows a quantile plot for the set of data in Table 5.11.

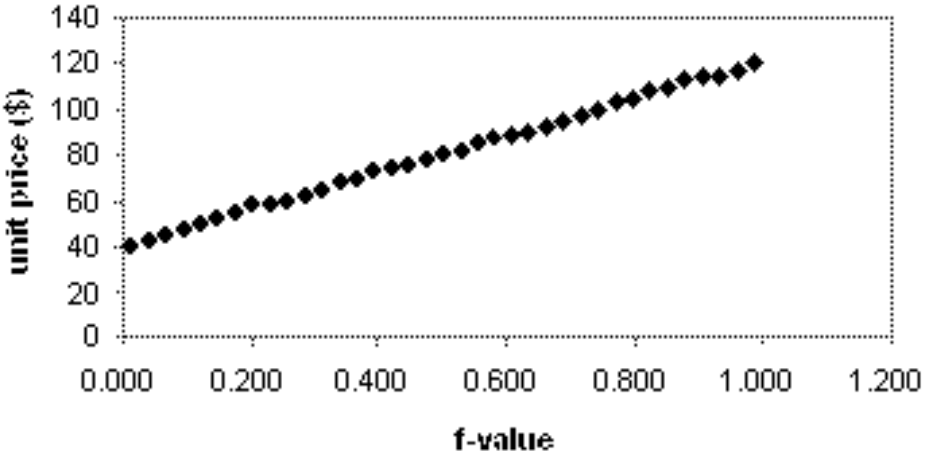


Figure 5.6: A quantile plot for the data set of Table 5.11.

- A **Q-Q plot**, or **quantile-quantile plot**, is a powerful visualization method for comparing the distributions of two or more sets of univariate observations. When distributions are compared, the goal is to understand how the distributions differ from one data set to the next. The most effective way to investigate the shifts of distributions is to compare corresponding quantiles.

  Suppose there are just two sets of univariate observations to be compared. Let $x_{(1)}, \ldots, x_{(n)}$ be the first data set, ordered from smallest to largest. Let $y_{(1)}, \ldots, y_{(m)}$ be the second, also ordered. Suppose $m \leq n$. If $m = n$, then $y_{(i)}$ and $x_{(i)}$ are both $(i \Leftrightarrow 0.5)/n$ quantiles of their respective data sets, so on the Q-Q plot, $y_{(i)}$ is graphed against $x_{(i)}$; that is, the ordered values for one set of data are graphed against the ordered values of the other set. If $m < n$, the $y_{(i)}$ is the $(i \Leftrightarrow 0.5)/m$ quantile of the y data, and $y_{(i)}$ is graphed against the $(i \Leftrightarrow 0.5)/m$ quantile of the x data, which typically must be computed by interpolation. With this method, there are always m points on the graph, where m is the number of values in the smaller of the two data sets. Figure 5.7 shows a quantile-quantile plot for the data set of Table 5.11.

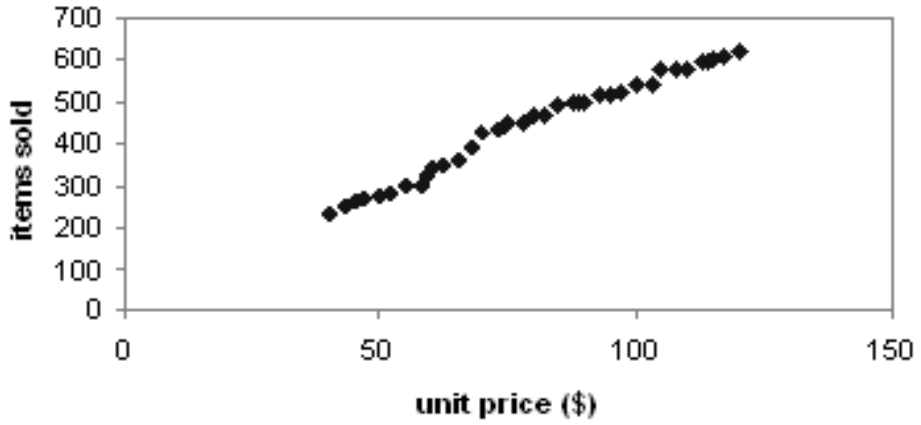


Figure 5.7: A quantile-quantile plot for the data set of Table 5.11.

- 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 quantitative variables. 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. The scatter plot is a useful exploratory method for providing a first look at bivariate data to see how they are distributed throughout the plane, for example, and to see clusters of points, outliers, and so forth. Figure 5.8 shows a scatter plot for the set of data in Table 5.11.

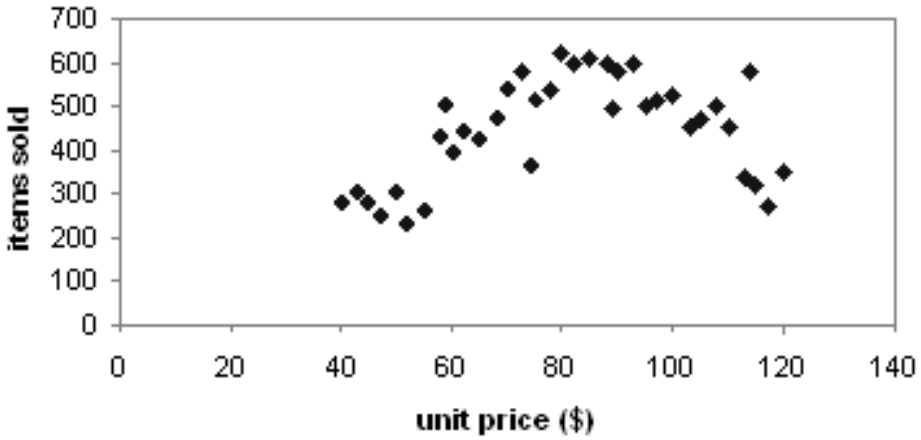


Figure 5.8: A scatter plot for the data set of Table 5.11.

- A **loess curve** is another important exploratory graphic aid which adds a smooth curve to a scatter plot in order to provide better perception of the pattern of dependence. The word *loess* is short for local regression. Figure 5.9 shows a loess curve for the set of data in Table 5.11.

  Two parameters need to be chosen to fit a loess curve. The first parameter, α, is a smoothing parameter. It can be any positive number, but typical values are between 1/4 to 1. The goal in choosing α is to produce a fit that is as smooth as possible without unduly distorting the underlying pattern in the data. As α increases, the curve becomes smoother. If α becomes large, the fitted function could be very smooth. There may be some lack of fit, however, indicating possible "missing" data patterns. If α is very small, the underlying pattern is tracked, yet overfitting of the data may occur, where local "wiggles" in the curve may not be supported by the data. The second parameter, λ, is the degree of polynomials that are fitted by the method; λ can be 1 or 2. If the underlying pattern of the data has a "gentle" curvature with no local maxima and minima, then

locally linear fitting is usually sufficient ($\lambda = 1$). However, if there are local maxima or minima, then locally quadratic fitting ($\lambda = 2$) typically does a better job of following the pattern of the data and maintaining local smoothness.

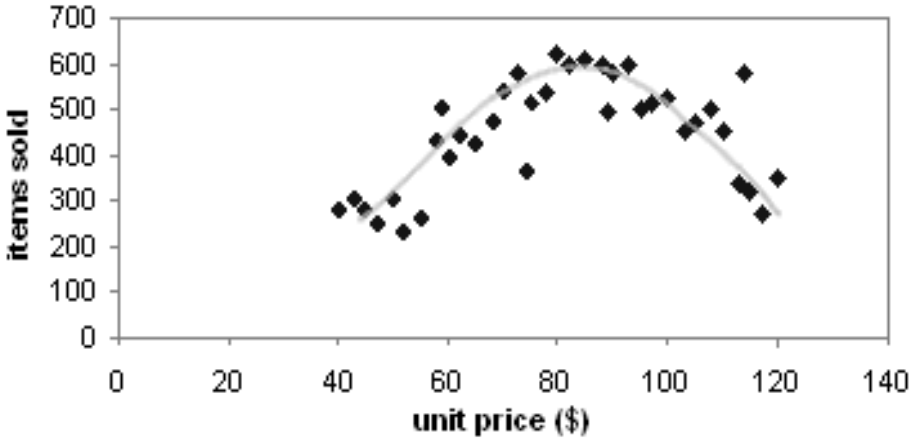


Figure 5.9: A loess curve for the data set of Table 5.11.

5.7 Discussion

We have presented a set of scalable methods for mining concept or class descriptions in large databases. In this section, we discuss related issues regarding such descriptions. These include a comparison of the cube-based and attribute-oriented induction approaches to data generalization with typical machine learning methods, the implementation of incremental and parallel mining of concept descriptions, and interestingness measures for concept description.

5.7.1 Concept description: A comparison with typical machine learning methods

In this chapter, we studied a set of database-oriented methods for mining concept descriptions in large databases. These methods included a data cube-based and an attribute-oriented induction approach to data generalization for concept description. Other influential concept description methods have been proposed and studied in the machine learning literature since the 1980s. Typical machine learning methods for concept description follow a *learning-from-examples* paradigm. In general, such methods work on sets of concept or class-labeled training examples which are examined in order to derive or learn a hypothesis describing the class under study.

"What are the major differences between methods of learning-from-examples and the data mining methods presented here?"

- First, there are differences in the philosophies of the machine learning and data mining approaches, and their basic assumptions regarding the concept description problem.

  In most of the learning-from-examples algorithms developed in machine learning, the set of examples to be analyzed is partitioned into two sets: *positive* examples and *negative* ones, respectively representing target and contrasting classes. The learning process selects one positive example at random, and uses it to form a hypothesis describing objects of that class. The learning process then performs *generalization* on the hypothesis using the remaining positive examples, and *specialization* using the negative examples. In general, the resulting hypothesis covers all the positive examples, but none of the negative examples.

  A database usually does not store the negative data explicitly. Thus no explicitly specified negative examples can be used for specialization. This is why, for analytical characterization mining and for comparison mining in general, data mining methods must collect a set of comparable data which are not in the target (positive) class, for use as negative data (Sections 5.4 and 5.5). Most database-oriented methods also therefore tend to

be generalization-based. Even though most provide the drill-down (specialization) operation, this operation is essentially implemented by backtracking the generalization process to a previous state.

Another major difference between machine learning and database-oriented techniques for concept description concerns the size of the set of training examples. For traditional machine learning methods, the training set is typically relatively small in comparison with the data analyzed by database-oriented techniques. Hence, for machine learning methods, it is easier to find descriptions which cover all of the positive examples without covering any negative examples. However, considering the diversity and huge amount of data stored in real-world databases, it is unlikely for analysis of such data to derive a rule or pattern which covers all of the positive examples but none of the negative ones. Instead, what one may expect to find is a set of features or rules which cover a *majority* of the data in the positive class, *maximally* distinguishing the positive from the negative examples. (This can also be described as a probability distribution).

- Second, distinctions between the machine learning and database-oriented approaches also exist regarding the methods of generalization used.

  Both approaches do employ attribute removal and attribute generalization (also known as concept tree ascension) as their main generalization techniques. Consider the set of training examples as a set of tuples. The machine learning approach thus performs generalization *tuple by tuple*, whereas the database-oriented approach performs generalization on an *attribute by attribute* (or entire dimension) basis.

  In the tuple by tuple strategy of the machine learning approach, the training examples are examined one at a time in order to induce generalized concepts. In order to form the most specific hypothesis (or concept description) that is consistent with all of the positive examples and none of the negative ones, the algorithm must search every node in the search space representing all of the possible concepts derived from generalization on each training example. Since different attributes of a tuple may be generalized to various levels of abstraction, the number of nodes searched for a given training example may involve a huge number of possible combinations.

  On the other hand, a database approach employing an attribute-oriented strategy performs generalization on each attribute or dimension uniformly for all of the tuples in the data relation at the *early* stages of generalization. Such an approach essentially focuses its attention on individual attributes, rather than on combinations of attributes. This is referred to as *factoring the version space*, where **version space** is defined as the subset of hypotheses consistent with the training examples. Factoring the version space can substantially improve the computational efficiency. Suppose there are k concept hierarchies used in the generalization and there are p nodes in each concept hierarchy. The total size of k factored version spaces is $p \times k$. In contrast, the size of the unfactored version space searched by the machine learning approach is p^k for the same concept tree.

  Notice that algorithms which, during the early generalization stages, explore many possible combinations of different attribute-value conditions given a large number of tuples cannot be productive since such combinations will eventually be merged during further generalizations. Different possible combinations should be explored only when the relation has first been generalized to a relatively smaller relation, as is done in the database-oriented approaches described in this chapter.

- Another obvious advantage of the attribute-oriented approach over many other machine learning algorithms is the integration of the data mining process with set-oriented database operations. In contrast to most existing learning algorithms which do not take full advantages of database facilities, the attribute-oriented induction approach primarily adopts relational operations, such as selection, join, projection (extracting task-relevant data and removing attributes), tuple substitution (ascending concept trees), and sorting (discovering common tuples among classes). Since relational operations are set-oriented whose implementation has been optimized in many existing database systems, the attribute-oriented approach is not only efficient but also can easily be exported to other relational systems. This comment applies to data cube-based generalization algorithms as well. The data cube-based approach explores more optimization techniques than traditional database query processing techniques by incorporating sparse cube techniques, various methods of cube computation, as well as indexing and accessing techniques. Therefore, a high performance gain of database-oriented algorithms over machine learning techniques, is expected when handling large data sets.

5.7.2 Incremental and parallel mining of concept description

Given the huge amounts of data in a database, it is highly preferable to update data mining results *incrementally* rather than mining from scratch on each database update. Thus incremental data mining is an attractive goal for many kinds of mining in large databases or data warehouses.

Fortunately, it is straightforward to extend the database-oriented concept description mining algorithms for incremental data mining.

Let's first examine extending the attribute-oriented induction approach for use in incremental data mining. Suppose a generalized relation R is stored in the database. When a set of new tuples, ΔDB, is inserted into the database, attribute-oriented induction can be performed on ΔDB in order to generalize the attributes to the same conceptual levels as the respective corresponding attributes in the generalized relation, R. The associated aggregation information, such as count, sum, etc., can be calculated by applying the generalization algorithm to ΔDB rather than to R. The generalized relation so derived, ΔR, on ΔDB, can then easily be merged into the generalized relation R, since R and ΔR share the same dimensions and exist at the same abstraction levels for each dimension. The union, $R \cup \Delta R$, becomes a new generalized relation, R'. Minor adjustments, such as dimension generalization or specialization, can be performed on R' as specified by the user, if desired. Similarly, a set of deletions can be viewed as the deletion of a small database, ΔDB, from DB. The incremental update should be the difference $R \Leftrightarrow \Delta R$, where R is the existing generalized relation and ΔR is the one generated from ΔDB. Similar algorithms can be worked out for data cube-based concept description. (This is left as an exercise).

Data sampling methods, parallel algorithms, and distributed algorithms can be explored for concept description mining, based on the same philosophy. For example, attribute-oriented induction can be performed by *sampling* a subset of data from a huge set of task-relevant data or by first performing induction *in parallel* on several partitions of the task-relevant data set, and then merging the generalized results.

5.7.3 Interestingness measures for concept description

"When examining concept descriptions, how can the data mining system objectively evaluate the interestingness of each description?"

Different users may have different preferences regarding what makes a given description interesting or useful. Let's examine a few interestingness measures for mining concept descriptions.

1. **Significance threshold**:

   Users may like to examine what kind of objects contribute "*significantly*" to the summary of the data. That is, given a concept description in the form of a generalized relation, say, they may like to examine the generalized tuples (acting as "object descriptions") which contribute a nontrivial *weight* or portion to the summary, while ignoring those which contribute only a negligible weight to the summary. In this context, one may introduce a **significance threshold** to be used in the following manner: if the weight of a generalized tuple/object is lower than the threshold, it is considered to represent only a negligible portion of the database and can therefore be ignored as uninteresting. Notice that ignoring such negligible tuples does not mean that they should be removed from the intermediate results (i.e., the prime generalized relation, or the data cube, depending on the implementation) since they may contribute to subsequent further exploration of the data by the user via interactive rolling up or drilling down of other dimensions and levels of abstraction. Such a threshold may also be called the **support threshold**, adopting the term popularly used in association rule mining.

   For example, if the significance threshold is set to 1%, a generalized tuple (or data cube cell) which represents less than 1% in count of the number of tuples (objects) in the database is omitted in the result presentation.

   Moreover, although the significance threshold, by default, is calculated based on count, other measures can be used. For example, one may use the sum of an amount (such as total sales) as the significance measure to observe the major objects contributing to the overall sales. Alternatively, the t-weight and d-weight measures studied earlier (Sections 5.2.3 and 5.5.2), which respectively indicate the typicality and discriminability of generalized tuples (or objects), may also be used.

2. **Deviation threshold.** Some users may already know the general behavior of the data and would like to instead explore the objects which *deviate* from this general behavior. Thus, it is interesting to examine how to identify the kind of data values that are considered outliers, or deviations.

Suppose the data to be examined are numeric. As discussed in Section 5.6, a common rule of thumb identifies suspected outliers as those values which fall at least $1.5 \times IQR$ above the third quartile or below the first quartile. Depending on the application at hand, however, such a rule of thumb may not always work well. It may therefore be desirable to provide a **deviation threshold** as an adjustable threshold to enlarge or shrink the set of possible outliers. This facilitates interactive analysis of the general behavior of outliers. We leave the identification of outliers in time-series data to Chapter 9, where time-series analysis will be discussed.

5.8 Summary

- Data mining can be classified into *descriptive data mining* and *predictive data mining*. **Concept description** is the most basic form of descriptive data mining. It describes a given set of task-relevant data in a concise and summarative manner, presenting interesting general properties of the data.

- Concept (or class) description consists of **characterization** and **comparison** (or **discrimination**). The former summarizes and describes a collection of data, called the **target class**; whereas the latter summarizes and distinguishes one collection of data, called the **target class**, from other collection(s) of data, collectively called the **contrasting class(es)**.

- There are two general approaches to concept characterization: the **data cube OLAP-based approach** and the **attribute-oriented induction approach**. Both are attribute- or dimension-based generalization approaches. The attribute-oriented induction approach can be implemented using either relational or data cube structures.

- The **attribute-oriented induction approach** consists of the following techniques: *data focusing*, *generalization by attribute removal or attribute generalization*, *count and aggregate value accumulation*, *attribute generalization control*, and *generalization data visualization*.

- Generalized data can be **visualized** in multiple forms, including generalized relations, crosstabs, bar charts, pie charts, cube views, curves, and rules. Drill-down and roll-up operations can be performed on the generalized data interactively.

- **Analytical data characterization/comparison** performs attribute and dimension relevance analysis in order to filter out irrelevant or weakly relevant attributes prior to the induction process.

- **Concept comparison** can be performed by the attribute-oriented induction or data cube approach in a manner similar to concept characterization. Generalized tuples from the target and contrasting classes can be quantitatively compared and contrasted.

- Characterization and comparison descriptions (which form a concept description) can both be visualized in the *same* generalized relation, crosstab, or quantitative rule form, although they are displayed with different interestingness measures. These measures include the **t-weight** (for tuple typicality) and **d-weight** (for tuple discriminability).

- From the descriptive statistics point of view, additional **statistical measures** should be introduced in describing central tendency and data dispersion. Quantiles, variations, and outliers are useful additional information which can be mined in databases. Boxplots, quantile plots, scattered plots, and quantile-quantile plots are useful visualization tools in descriptive data mining.

- In comparison with machine learning algorithms, database-oriented concept description leads to efficiency and scalability in large databases and data warehouses.

- Concept description mining can be performed *incrementally*, in *parallel*, or in a *distributed manner*, by making minor extensions to the basic methods involved.

- Additional interestingness measures, such as the **significance threshold** or **deviation threshold**, can be included and dynamically adjusted by users for mining interesting class descriptions.

Exercises

1. Suppose that the *employee* relation in a store database has the data set presented in Table 5.12.

name	gender	department	age	years_worked	residence	salary	#_of_children
Jamie Wise	M	Clothing	21	3	3511 Main St., Richmond	$20K	0
Sandy Jones	F	Shoe	39	20	125 Austin Ave., Burnaby	$25K	2
...	...	...	...	...	...	...	...

Table 5.12: The *employee* relation for data mining.

   (a) Propose a concept hierarchy for each of the attributes *department, age, years_worked, residence, salary* and *#_of_children*.

   (b) Mine the prime generalized relation for characterization of all of the employees.

   (c) Drill down along the dimension *years_worked*.

   (d) Present the above description as a crosstab, bar chart, pie chart, and as logic rules.

   (e) Characterize only the employees is the Shoe Department.

   (f) Compare the set of employees who have children vs. those who have no children.

2. Outline the major steps of the data cube-based implementation of class characterization. What are the major differences between this method and a relational implementation such as attribute-oriented induction? Discuss which method is most efficient and under what conditions this is so.

3. Discuss why analytical data characterization is needed and how it can be performed. Compare the result of two induction methods: (1) with relevance analysis, and (2) without relevance analysis.

4. Give three additional commonly used statistical measures (i.e., not illustrated in this chapter) for the characterization of data dispersion, and discuss how they can be computed efficiently in large databases.

5. Outline a data cube-based *incremental* algorithm for mining analytical class comparisons.

6. Outline a method for (1) parallel and (2) distributed mining of statistical measures.

Bibliographic Notes

Generalization and summarization methods have been studied in the statistics literature long before the onset of computers. Good summaries of statistical descriptive data mining methods include Cleveland [7], and Devore [10]. Generalization-based induction techniques, such as learning-from-examples, were proposed and studied in the machine learning literature before data mining became active. A theory and methodology of inductive learning was proposed in Michalski [23]. Version space was proposed by Mitchell [25]. The method of factoring the version space described in Section 5.7 was presented by Subramanian and Feigenbaum [30]. Overviews of machine learning techniques can be found in Dietterich and Michalski [11], Michalski, Carbonell, and Mitchell [24], and Mitchell [27].

The data cube-based generalization technique was initially proposed by Codd, Codd, and Salley [8] and has been implemented in many OLAP-based data warehouse systems, such as Kimball [20]. Gray et al. [13] proposed a cube operator for computing aggregations in data cubes. Recently, there have been many studies on the efficient computation of data cubes, which contribute to the efficient computation of data generalization. A comprehensive survey on the topic can be found in Chaudhuri and Dayal [6].

Database-oriented methods for concept description explore scalable and efficient techniques for describing large sets of data in databases and data warehouses. The attribute-oriented induction method described in this chapter was first proposed by Cai, Cercone, and Han [5] and further extended by Han, Cai, and Cercone [15], and Han and Fu [16].

There are many methods for assessing attribute relevance. Each has its own bias. The information gain measure is biased towards attributes with many values. Many alternatives have been proposed, such as gain ratio (Quinlan [29])

which considers the probability of each attribute value. Other relevance measures include the gini index (Breiman et al. [2]), the χ^2 contingency table statistic, and the uncertainty coefficient (Johnson and Wickern [19]). For a comparison of attribute selection measures for decision tree induction, see Buntine and Niblett [3]. For additional methods, see Liu and Motoda [22], Dash and Liu [9], Almuallim and Dietterich [1], and John [18].

For statistics-based visualization of data using boxplots, quantile plots, quantile-quantile plots, scattered plots, and loess curves, see Cleveland [7], and Devore [10]. Ng and Knorr [21] studied a unified approach for defining and computing outliers.

Bibliography

[1] H. Almuallim and T. G. Dietterich. Learning with many irrelevant features. In *Proc. 9th National Conf. on Artificial Intelligence (AAAI'91)*, pages 547–552, July 1991.

[2] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth International Group, 1984.

[3] W. L. Buntine and Tim Niblett. A further comparison of splitting rules for decision-tree induction. *Machine Learning*, 8:75–85, 1992.

[4] Y. Cai, N. Cercone, and J. Han. Attribute-oriented induction in relational databases. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 213–228. AAAI/MIT Press, 1991.

[5] Y. Cai, N. Cercone, and J. Han. Attribute-oriented induction in relational databases. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 213–228. AAAI/MIT Press Also in Proc. IJCAI-89 Workshop Knowledge Discovery in Databases, Detroit, MI, August 1989, 26-36., 1991.

[6] S. Chaudhuri and U. Dayal. An overview of data warehousing and OLAP technology. *ACM SIGMOD Record*, 26:65–74, 1997.

[7] W. Cleveland. Visualizing data. In *AT&T Bell Laboratories*, Hobart Press, Summit NJ, 1993.

[8] E. F Codd, S. B. Codd, and C. T. Salley. Providing OLAP (on-line analytical processing) to user-analysts: An IT mandate. In *E. F. Codd & Associates available at http://www.arborsoft.com/OLAP.html*, 1993.

[9] M. Dash and H. Liu. Feature selecion for classificaion. In *Intelligent Data Analysis*, volume 1 of *3*, 1997.

[10] J. L. Devore. *Probability and Statistics for Engineering and the Science, 4th ed.* Duxbury Press, 1995.

[11] T. G. Dietterich and R. S. Michalski. A comparative review of selected methods for learning from examples. In Michalski et al., editor, *Machine Learning: An Artificial Intelligence Approach, Vol. 1*, pages 41–82. Morgan Kaufmann, 1983.

[12] M. Genesereth and N. Nilsson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann, 1987.

[13] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data cube: A relational operator generalizing group-by, cross-tab and sub-totals. In *Proc. 1996 Int. Conf. Data Engineering*, pages 152–159, New Orleans, Louisiana, Feb. 1996.

[14] A. Gupta, V. Harinarayan, and D. Quass. Aggregate-query processing in data warehousing environment. In *Proc. 21st Int. Conf. Very Large Data Bases*, pages 358–369, Zurich, Switzerland, Sept. 1995.

[15] J. Han, Y. Cai, and N. Cercone. Data-driven discovery of quantitative rules in relational databases. *IEEE Trans. Knowledge and Data Engineering*, 5:29–40, 1993.

[16] J. Han and Y. Fu. Exploration of the power of attribute-oriented induction in data mining. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 399–421. AAAI/MIT Press, 1996.

[17] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data*, pages 205–216, Montreal, Canada, June 1996.

[18] G. H. John. *Enhancements to the Data Mining Process.* Ph.D. Thesis, Computer Science Dept., Stanford Univeristy, 1997.

[19] R. A. Johnson and D. W. Wickern. *Applied Multivariate Statistical Analysis, 3rd ed.* Prentice Hall, 1992.

[20] R. Kimball. *The Data Warehouse Toolkit.* John Wiley & Sons, New York, 1996.

[21] E. Knorr and R. Ng. Algorithms for mining distance-based outliers in large datasets. In *Proc. 1998 Int. Conf. Very Large Data Bases*, pages 392–403, New York, NY, August 1998.

[22] H. Liu and H. Motoda. *Feature Selection for Knowledge Discovery and Data Mining.* Kluwer Academic Publishers, 1998.

[23] R. S. Michalski. A theory and methodology of inductive learning. In Michalski et al., editor, *Machine Learning: An Artificial Intelligence Approach, Vol. 1*, pages 83–134. Morgan Kaufmann, 1983.

[24] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell. *Machine Learning, An Artificial Intelligence Approach, Vol. 2.* Morgan Kaufmann, 1986.

[25] T. M. Mitchell. Version spaces: A candidate elimination approach to rule learning. In *Proc. 5th Int. Joint Conf. Artificial Intelligence*, pages 305–310, Cambridge, MA, 1977.

[26] T. M. Mitchell. Generalization as search. *Artificial Intelligence*, 18:203–226, 1982.

[27] T. M. Mitchell. *Machine Learning.* McGraw Hill, 1997.

[28] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[29] J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.

[30] D. Subramanian and J. Feigenbaum. Factorization in experiment generation. In *Proc. 1986 AAAI Conf.*, pages 518–522, Philadelphia, PA, August 1986.

[31] J. Widom. Research problems in data warehousing. In *Proc. 4th Int. Conf. Information and Knowledge Management*, pages 25–30, Baltimore, Maryland, Nov. 1995.

[32] W. P. Yan and P. Larson. Eager aggregation and lazy aggregation. In *Proc. 21st Int. Conf. Very Large Data Bases*, pages 345–357, Zurich, Switzerland, Sept. 1995.

[33] W. Ziarko. *Rough Sets, Fuzzy Sets and Knowledge Discovery.* Springer-Verlag, 1994.

Contents

Chapter 6

Mining Association Rules in Large Databases

Association rule mining finds interesting association or correlation relationships among a large set of data items. With massive amounts of data continuously being collected and stored in databases, many industries are becoming interested in mining association rules from their databases. For example, the discovery of interesting association relationships among huge amounts of business transaction records can help catalog design, cross-marketing, loss-leader analysis, and other business decision making processes.

A typical example of association rule mining is **market basket analysis**. This process analyzes customer buying habits by finding associations between the different items that customers place in their "shopping baskets" (Figure 6.1). The discovery of such associations can help retailers develop marketing strategies by gaining insight into which items are frequently purchased together by customers. For instance, if customers are buying milk, how likely are they to also buy bread (and what kind of bread) on the same trip to the supermarket? Such information can lead to increased sales by helping retailers to do selective marketing and plan their shelf space. For instance, placing milk and bread within close proximity may further encourage the sale of these items together within single visits to the store.

How can we find association rules from large amounts of data, where the data are either transactional or relational? Which association rules are the most interesting? How can we help or guide the mining procedure to discover interesting associations? What language constructs are useful in defining a data mining query language for association rule mining? In this chapter, we will delve into each of these questions.

6.1 Association rule mining

Association rule mining searches for interesting relationships among items in a given data set. This section provides an introduction to association rule mining. We begin in Section 6.1.1 by presenting an example of market basket analysis, the earliest form of association rule mining. The basic concepts of mining associations are given in Section 6.1.2. Section 6.1.3 presents a road map to the different kinds of association rules that can be mined.

6.1.1 Market basket analysis: A motivating example for association rule mining

Suppose, as manager of an *AllElectronics* branch, you would like to learn more about the buying habits of your customers. Specifically, you wonder *"Which groups or sets of items are customers likely to purchase on a given trip to the store?"*. To answer your question, market basket analysis may be performed on the retail data of customer transactions at your store. The results may be used to plan marketing or advertising strategies, as well as catalog design. For instance, market basket analysis may help managers design different store layouts. In one strategy, items that are frequently purchased together can be placed in close proximity in order to further encourage the sale of such items together. If customers who purchase computers also tend to buy financial management software at the same time, then placing the hardware display close to the software display may help to increase the sales of both of these

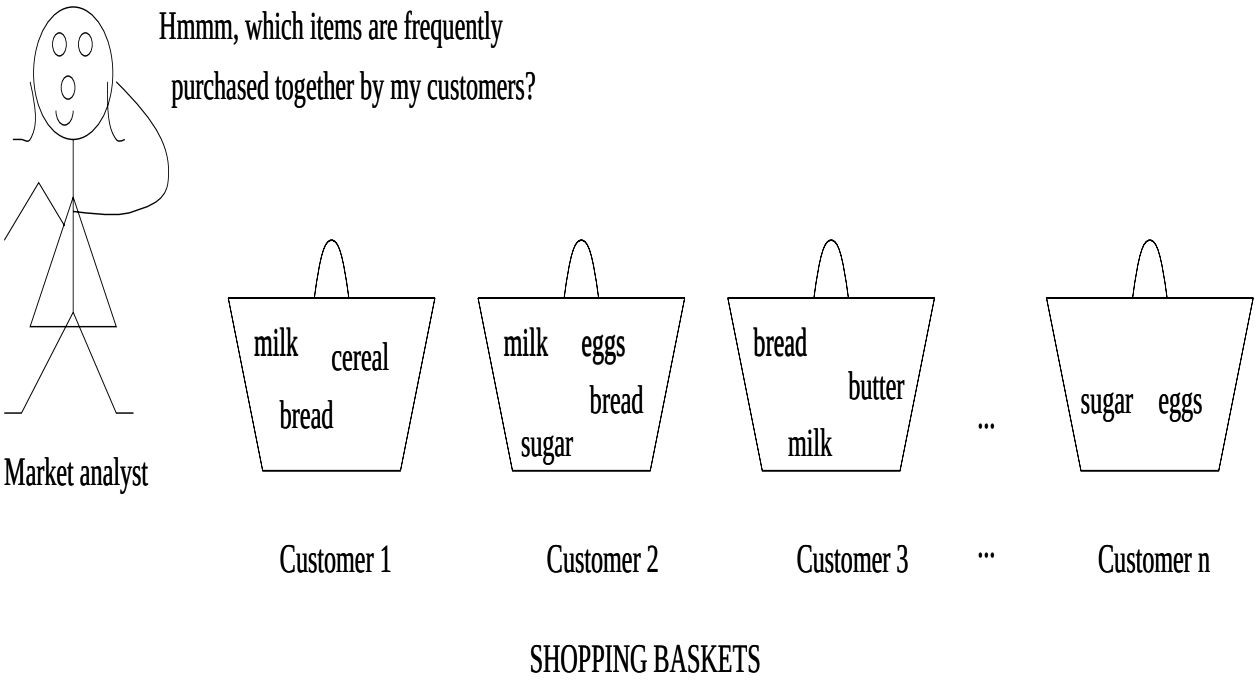


Figure 6.1: Market basket analysis.

items. In an alternative strategy, placing hardware and software at opposite ends of the store may entice customers who purchase such items to pick up other items along the way. For instance, after deciding on an expensive computer, a customer may observe security systems for sale while heading towards the software display to purchase financial management software, and may decide to purchase a home security system as well. Market basket analysis can also help retailers to plan which items to put on sale at reduced prices. If customers tend to purchase computers and printers together, then having a sale on printers may encourage the sale of printers *as well as* computers.

If we think of the universe as the set of items available at the store, then each item has a Boolean variable representing the presence or absence of that item. Each basket can then be represented by a Boolean vector of values assigned to these variable. The Boolean vectors can be analyzed for buying patterns which reflect items that are frequent *associated* or purchased together. These patterns can be represented in the form of **association rules**. For example, the information that customers who purchase computers also tend to buy financial management software at the same time is represented in association Rule (6.1) below.

$$computer \Rightarrow financial\_management\_software \qquad [support = 2\%, confidence = 60\%] \tag{6.1}$$

Rule **support** and **confidence** are two measures of rule interestingness that were described earlier in Section 1.5. They respectively reflect the usefulness and certainty of discovered rules. A support of 2% for association Rule (6.1) means that 2% of all the transactions under analysis show that computer and financial management software are purchased together. A confidence of 60% means that 60% of the customers who purchased a computer also bought the software. Typically, association rules are considered interesting if they satisfy both a **minimum support threshold** and a **minimum confidence threshold**. Such thresholds can be set by users or domain experts.

6.1.2 Basic concepts

Let $\mathcal{I}=\{i_1, i_2, ..., i_m\}$ be a set of items. Let D, the task relevant data, be a set of database transactions where each transaction T is a set of items such that $T \subseteq \mathcal{I}$. Each transaction is associated with an identifier, called TID. Let A be a set of items. A transaction T is said to contain A if and only if $A \subseteq T$. An **association rule** is an implication of the form $A \Rightarrow B$, where $A \subset \mathcal{I}$, $B \subset \mathcal{I}$ and $A \cap B = \phi$. The rule $A \Rightarrow B$ holds in the transaction set D with **support** s, where s is the percentage of transactions in D that contain $A \cup B$. The rule $A \Rightarrow B$ has **confidence** c

in the transaction set D if c is the percentage of transactions in D containing A which also contain B. That is,

$$support(A \Rightarrow B) = Prob\{A \cup B\} \quad (6.2)$$
$$confidence(A \Rightarrow B) = Prob\{B|A\}. \quad (6.3)$$

Rules that satisfy both a minimum support threshold (*min_sup*) and a minimum confidence threshold (*min_conf*) are called **strong**.

A set of items is referred to as an **itemset**. An itemset that contains k items is a **k-itemset**. The set {*computer, financial_management_software*} is a 2-itemset. The **occurrence frequency of an itemset** is the number of transactions that contain the itemset. This is also known, simply, as the **frequency** or **support count** of the itemset. An itemset satisfies minimum support if the occurrence frequency of the itemset is greater than or equal to the product of *min_sup* and the total number of transactions in D. If an itemset satisfies minimum support, then it is a **frequent** itemset[1]. The set of frequent k-itemsets is commonly denoted by L_k [2].

"How are association rules mined from large databases?" Association rule mining is a two-step process:

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

Additional interestingness measures can be applied, if desired. The second step is the easiest of the two. The overall performance of mining association rules is determined by the first step.

6.1.3 Association rule mining: A road map

Market basket analysis is just one form of association rule mining. In fact, there are many kinds of association rules. Association rules can be classified in various ways, based on the following criteria:

1. Based on the *types of values* handled in the rule:

   If a rule concerns associations between the presence or absence of items, it is a **Boolean association rule**. For example, Rule (6.1) above is a Boolean association rule obtained from market basket analysis.

   If a rule describes associations between quantitative items or attributes, then it is a **quantitative association rule**. In these rules, quantitative values for items or attributes are partitioned into intervals. Rule (6.4) below is an example of a quantitative association rule.

   $$age(X, \text{“}30 - 34\text{”}) \wedge income(X, \text{“}42K - 48K\text{”}) \Rightarrow buys(X, \text{“}high\ resolution\ TV\text{”}) \quad (6.4)$$

   Note that the quantitative attributes, *age* and *income*, have been discretized.

2. Based on the *dimensions* of data involved in the rule:

   If the items or attributes in an association rule each reference only one dimension, then it is a **single-dimensional association rule**. Note that Rule (6.1) could be rewritten as

   $$buys(X, \text{“}computer\text{”}) \Rightarrow buys(X, \text{“}financial\_management\_software\text{”}) \quad (6.5)$$

   Rule (6.1) is therefore a single-dimensional association rule since it refers to only one dimension, i.e., *buys*.

   If a rule references two or more dimensions, such as the dimensions *buys, time_of_transaction*, and *customer_category*, then it is a **multidimensional association rule**. Rule (6.4) above is considered a multi-dimensional association rule since it involves three dimensions, *age, income*, and *buys*.

[1] In early work, itemsets satisfying minimum support were referred to as **large**. This term, however, is somewhat confusing as it has connotations to the number of items in an itemset rather than the frequency of occurrence of the set. Hence, we use the more recent term of **frequent**.

[2] Although the term **frequent** is preferred over **large**, for historical reasons frequent k-itemsets are still denoted as L_k.

3. Based on the *levels of abstractions* involved in the rule set:

   Some methods for association rule mining can find rules at differing levels of abstraction. For example, suppose that a set of association rules mined included Rule (6.6) and (6.7) below.

$$age(X, \text{“}30 - 34\text{”}) \Rightarrow buys(X, \text{“}laptop\ computer\text{”}) \tag{6.6}$$

$$age(X, \text{“}30 - 34\text{”}) \Rightarrow buys(X, \text{“}computer\text{”}) \tag{6.7}$$

   In Rules (6.6) and (6.7), the items bought are referenced at different levels of abstraction. (That is, "*computer*" is a higher level abstraction of "*laptop computer*"). We refer to the rule set mined as consisting of **multilevel association rules**. If, instead, the rules within a given set do not reference items or attributes at different levels of abstraction, then the set contains **single-level association rules**.

4. Based on the *nature of the association* involved in the rule: Association mining can be extended to correlation analysis, where the absence or presence of correlated items can be identified.

Throughout the rest of this chapter, you will study methods for mining each of the association rule types described.

6.2 Mining single-dimensional Boolean association rules from transactional databases

In this section, you will learn methods for mining the simplest form of association rules - *single-dimensional, single-level, Boolean association rules*, such as those discussed for market basket analysis in Section 6.1.1. We begin by presenting **Apriori**, a basic algorithm for finding frequent itemsets (Section 6.2.1). A procedure for generating strong association rules from frequent itemsets is discussed in Section 6.2.2. Section 6.2.3 describes several variations to the Apriori algorithm for improved efficiency and scalability.

6.2.1 The Apriori algorithm: Finding frequent itemsets

Apriori is an influential algorithm for mining frequent itemsets for Boolean association rules. The name of the algorithm is based on the fact that the algorithm uses *prior knowledge* of frequent itemset properties, as we shall see below. Apriori employs an iterative approach known as a *level-wise* search, where k-itemsets are used to explore *(k+1)*-itemsets. First, the set of frequent 1-itemsets is found. This set is denoted L_1. L_1 is used to find L_2, the frequent 2-itemsets, which is used to find L_3, and so on, until no more frequent k-itemsets can be found. The finding of each L_k requires one full scan of the database.

To improve the efficiency of the level-wise generation of frequent itemsets, an important property called *the Apriori property*, presented below, is used to reduce the search space.

The Apriori property. All non-empty subsets of a frequent itemset must also be frequent.

This property is based on the following observation. By definition, if an itemset I does not satisfy the minimum support threshold, s, then I is not frequent, i.e., $Prob\{I\} < s$. If an item A is added to the itemset I, then the resulting itemset (i.e., $I \cup A$) cannot occur more frequently than I. Therefore, $I \cup A$ is not frequent either, i.e., $Prob\{I \cup A\} < s$.

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

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

1. **The join step**: To find L_k, a set of **candidate** k-itemsets is generated by joining L_{k-1} with itself. This set of candidates is denoted C_k. The join, $L_{k-1} \bowtie L_{k-1}$, is performed, where members of L_{k-1} are joinable if they have $(k-2)$ items in common, that is, $L_{k-1} \bowtie L_{k-1} = \{A \bowtie B | A, B \in L_{k-1}, |A \cap B| = k-2\}$.

2. **The prune step**: C_k is a superset of L_k, that is, its members may or may not be frequent, but all of the frequent k-itemsets are included in C_k. A scan of the database to determine the count of each candidate in C_k would result in the determination of L_k (i.e., all candidates having a count no less than the minimum support count are frequent by definition, and therefore belong to L_k). C_k, however, can be huge, and so this could involve heavy computation. To reduce the size of C_k, the Apriori property is used as follows. Any *(k-1)*-itemset that is not frequent cannot be a subset of a frequent k-itemset. Hence, if any *(k-1)*-subset of a candidate k-itemset is not in L_{k-1}, then the candidate cannot be frequent either and so can be removed from C_k. This subset testing can be done quickly by maintaining a hash tree of all frequent itemsets.

AllElectronics database

TID	List of item_ID's
T100	I1, I2, I5
T200	I2, I3, I4
T300	I3, I4
T400	I1, I2, I3, I4

Figure 6.2: Transactional data for an *AllElectronics* branch.

Example 6.1 Let's look at a concrete example of Apriori, based on the *AllElectronics* transaction database, D, of Figure 6.2. There are four transactions in this database, i.e., $|D| = 4$. Apriori assumes that items within a transaction are sorted in lexicographic order. We use Figure 6.3 to illustrate the APriori algorithm for finding frequent itemsets in D.

- In the first iteration of the algorithm, each item is a member of the set of candidate 1-itemsets, C_1. The algorithm simply scans all of the transactions in order to count the number of occurrences of each item.
- Suppose that the minimum transaction support count required is 2 (i.e., $min\_sup = 50\%$). The set of frequent 1-itemsets, L_1, can then be determined. It consists of the candidate 1-itemsets having minimum support.
- To discover the set of frequent 2-itemsets, L_2, the algorithm uses $L_1 \bowtie L_1$ to generate a candidate set of 2-itemsets, C_2 [3]. C_2 consists of $\binom{|L_1|}{2}$ 2-itemsets.
- Next, the transactions in D are scanned and the support count of each candidate itemset in C_2 is accumulated, as shown in the middle table of the second row in Figure 6.3.
- The set of frequent 2-itemsets, L_2, is then determined, consisting of those candidate 2-itemsets in C_2 having minimum support.
- The generation of the set of candidate 3-itemsets, C_3, is detailed in Figure 6.4. First, let $C_3 = L_2 \bowtie L_2 = \{\{I1, I2, I3\}, \{I1, I2, I4\}, \{I2, I3, I4\}\}$. Based on the Apriori property that all subsets of a frequent itemset must also be frequent, we can determine that the candidates {I1,I2,I3} and {I1,I2,I4} cannot possibly be frequent. We therefore remove them from C_3, thereby saving the effort of unnecessarily obtaining their counts during the subsequent scan of D to determine L_3. Note that since the Apriori algorithm uses a level-wise search strategy, then given a k-itemset, we only need to check if its *(k-1)*-subsets are frequent.
- The transactions in D are scanned in order to determine L_3, consisting of those candidate 3-itemsets in C_3 having minimum support (Figure 6.3).
- No more frequent itemsets can be found (since here, $C_4 = \phi$), and so the algorithm terminates, having found all of the frequent itemsets.

□

[3] $L_1 \bowtie L_1$ is equivalent to $L_1 \times L_1$ since the definition of $L_k \bowtie L_k$ requires the two joining itemsets to share $k - 1 = 0$ items.

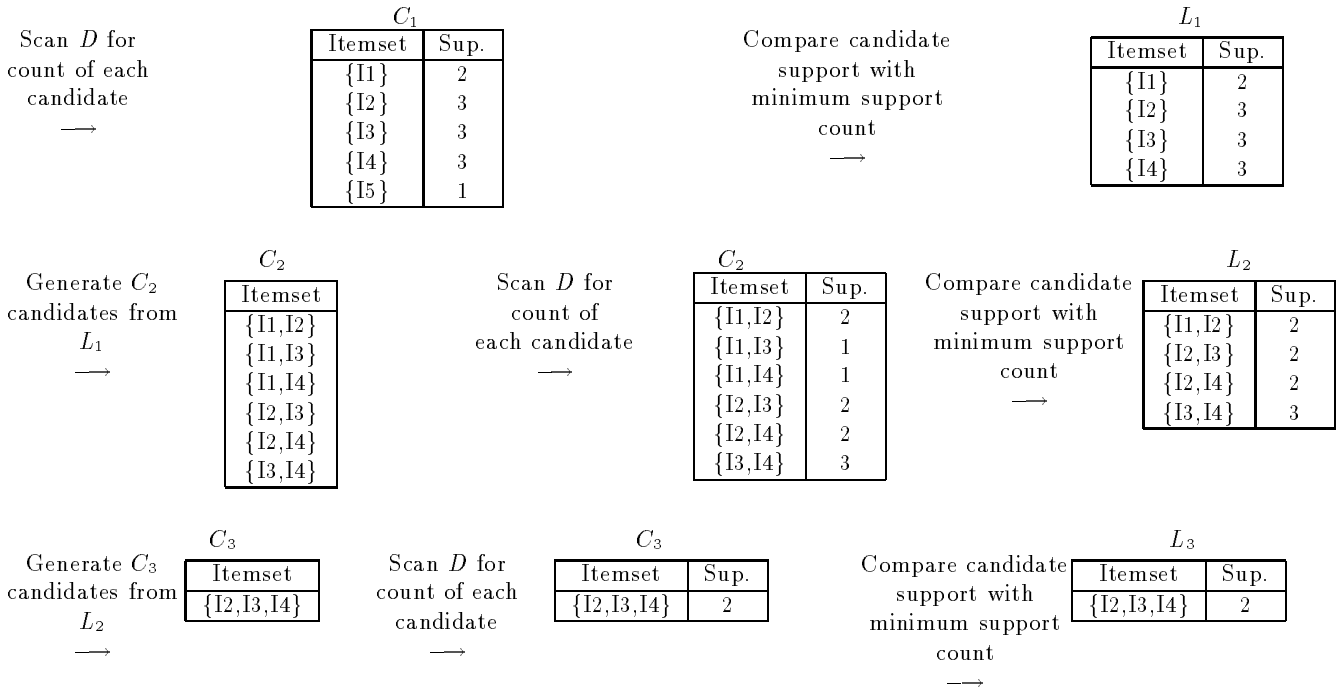


Figure 6.3: Generation of candidate itemsets and frequent itemsets, where the minimum support count is 2.

1. $C_3 = L_2 \bowtie L_2 = \{\{\text{I1,I2}\}, \{\text{I2,I3}\}, \{\text{I2,I4}\}, \{\text{I3,I4}\}\} \bowtie \{\{\text{I1,I2}\}, \{\text{I2,I3}\}, \{\text{I2,I4}\}, \{\text{I3,I4}\}\} = \{\{I1, I2, I3\}, \{I1, I2, I4\}, \{I2, I3, I4\}\}$.
2. Apriori property: All subsets of a frequent itemset must also be frequent. Do any of the candidates have a subset that is not frequent?
   - The 2-item subsets of {I1,I2,I3} are {I1,I2}, {I1,I3}, and {I2,I3}. {I1,I3} is not a member of L_2, and so it is not frequent. Therefore, remove {I1,I2,I3} from C_3.
   - The 2-item subsets of {I1,I2,I4} are {I1,I2}, {I1,I4}, and {I2,I4}. {I1,I4} is not a member of L_2, and so it is not frequent. Therefore, remove {I1,I2,I4} from C_3.
   - The 2-item subsets of {I2,I3,I4} are {I2,I3}, {I2,I4}, and {I3,I4}. All 2-item subsets of {I2,I3,I4} are members of L_2. Therefore, keep {I2,I3,I4} in C_3.
3. Therefore, $C_3 = \{\{I2, I3, I4\}\}$.

Figure 6.4: Generation of candidate 3-itemsets, C_3, from L_2 using the Apriori property.

Algorithm 6.2.1 (Apriori) Find frequent itemsets using an iterative level-wise approach.

Input: Database, D, of transactions; minimum support threshold, *min_sup*.

Output: L, frequent itemsets in D.

Method:

```
1)  L_1 = find_frequent_1-itemsets(D);
2)  for (k = 2; L_{k-1} ≠ φ; k++) {
3)      C_k = apriori_gen(L_{k-1}, min_sup);
4)      for each transaction t ∈ D { // scan D for counts
5)          C_t = subset(C_k, t); // get the subsets of t that are candidates
6)          for each candidate c ∈ C_t
7)              c.count++;
8)      }
9)      L_k = {c ∈ C_k | c.count ≥ min_sup}
10) }
11) return L = ∪_k L_k;

procedure apriori_gen(L_{k-1}:frequent (k-1)-itemsets; min_sup: minimum support)
1)  for each itemset l_1 ∈ L_{k-1}
2)      for each itemset l_2 ∈ L_{k-1}
3)          if (l_1[1] = l_2[1]) ∧ (l_1[2] = l_2[2]) ∧ ... ∧ (l_1[k-2] = l_2[k-2]) ∧ (l_1[k-1] < l_2[k-1]) then {
4)              c = l_1 ⋈ l_2; // join step: generate candidates
5)              if has_infrequent_subset(c, L_{k-1}) then
6)                  delete c; // prune step: remove unfruitful candidate
7)              else add c to C_k;
8)          }
9)  return C_k;

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

Figure 6.5: The Apriori algorithm for discovering frequent itemsets for mining Boolean association rules.

Figure 6.5 shows pseudo-code for the Apriori algorithm and its related procedures. Step 1 of Apriori finds the frequent 1-itemsets, L_1. In steps 2-10, L_{k-1} is used to generate candidates C_k in order to find L_k. The **apriori_gen** procedure generates the candidates and then uses the Apriori property to eliminate those having a subset that is not frequent (step 3). This procedure is described below. Once all the candidates have been generated, the database is scanned (step 4). For each transaction, a `subset` function is used to find all subsets of the transaction that are candidates (step 5), and the count for each of these candidates is accumulated (steps 6-7). Finally, all those candidates satisfying minimum support form the set of frequent itemsets, L. A procedure can then be called to generate association rules from the frequent itemsets. Such as procedure is described in Section 6.2.2.

The apriori_gen procedure performs two kinds of actions, namely **join** and **prune**, as described above. In the join component, L_{k-1} is joined with L_{k-1} to generate potential candidates (steps 1-4). The condition $l_1[k-1] < l_2[k-1]$ simply ensures that no duplicates are generated (step 3). The prune component (steps 5-7) employs the Apriori property to remove candidates that have a subset that is not frequent. The test for infrequent subsets is shown in procedure **has_infrequent_subset**.

6.2.2 Generating association rules from frequent itemsets

Once the frequent itemsets from transactions in a database D have been found, it is straightforward to generate strong association rules from them (where *strong* association rules satisfy both minimum support and minimum

confidence). This can be done using Equation (6.8) for confidence, where the conditional probability is expressed in terms of itemset support:

$$confidence(A \Rightarrow B) = Prob(B|A) = \frac{support(A \cup B)}{support(A)}, \tag{6.8}$$

where $support(A \cup B)$ is the number of transactions containing the itemsets $A \cup B$, and $support(A)$ is the number of transactions containing the itemset A.

Based on this equation, association rules can be generated as follows.

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

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

Example 6.2 Let's try an example based on the transactional data for *AllElectronics* shown in Figure 6.2. Suppose the data contains the frequent itemset l = {I2,I3,I4}. What are the association rules that can be generated from l? The non-empty subsets of l are {I2,I3}, {I2,I4}, {I3,I4}, {I2}, {I3}, and {I4}. The resulting association rules are as shown below, each listed with its confidence.

$$\begin{aligned}
&I2 \wedge I3 \Rightarrow I4, && confidence = 2/2 = 100\% \\
&I2 \wedge I4 \Rightarrow I3, && confidence = 2/2 = 100\% \\
&I3 \wedge I4 \Rightarrow I2, && confidence = 2/3 = 67\% \\
&I2 \Rightarrow I3 \wedge I4, && confidence = 2/3 = 67\% \\
&I3 \Rightarrow I2 \wedge I4, && confidence = 2/3 = 67\% \\
&I4 \Rightarrow I2 \wedge I3, && confidence = 2/3 = 67\%
\end{aligned}$$

If the minimum confidence threshold is, say, 70%, then only the first and second rules above are output, since these are the only ones generated that are strong. □

6.2.3 Variations of the Apriori algorithm

"How might the efficiency of Apriori be improved?"

Many variations of the Apriori algorithm have been proposed. A number of these variations are enumerated below. Methods 1 to 6 focus on improving the efficiency of the original algorithm, while methods 7 and 8 consider transactions over time.

1. A **hash-based technique**: Hashing itemset counts.

   A hash-based technique can be used to reduce the size of the candidate k-itemsets, C_k, for $k > 1$. For example, when scanning each transaction in the database to generate the frequent 1-itemsets, L_1, from the candidate 1-itemsets in C_1, we can generate all of the 2-itemsets for each transaction, hash (i.e., map) them into the different *buckets* of a *hash table* structure, and increase the corresponding bucket counts (Figure 6.6). A 2-itemset whose corresponding bucket count in the hash table is below the support threshold cannot be frequent and thus should be removed from the candidate set. Such a hash-based technique may substantially reduce the number of the candidate k-itemsets examined (especially when $k = 2$).

2. **Scan reduction**: Reducing the number of database scans.

   Recall that in the Apriori algorithm, one scan is required to determine L_k *for each* C_k. A **scan reduction** technique reduces the total number of scans required by doing extra work in some scans. For example, in the Apriori algorithm, C_3 is generated based on $L_2 \bowtie L_2$. However, C_2 can also be used to generate the candidate 3-itemsets. Let C'_3 be the candidate 3-itemsets generated from $C_2 \bowtie C_2$, instead of from $L_2 \bowtie L_2$. Clearly, $|C'_3|$ will be greater than $|C_3|$. However, if $|C'_3|$ is not much larger than $|C_3|$, and both C_2 and C'_3 can be stored in

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

H_2

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

Figure 6.6: Hash table, H_2, for candidate 2-itemsets: This hash table was generated by scanning the transactions of Figure 6.2 while determining L_1 from C_1. If the minimum support count is 2, for example, then the itemsets in buckets 0, 1, and 4 cannot be frequent and so they should not be included in C_2.

main memory, we can find L_2 and L_3 together when the next scan of the database is performed, thereby saving one database scan. Using this strategy, we can determine all L_k's by as few as two scans of the database (i.e., one initial scan to determine L_1 and a final scan to determine all other large itemsets), assuming that C'_k for $k \geq 3$ is generated from C'_{k-1} and all C'_ks for $k > 2$ can be kept in the memory.

3. **Transaction reduction**: Reducing the number of transactions scanned in future iterations.

   A transaction which does not contain any frequent k-itemsets cannot contain any frequent $(k + 1)$-itemsets. Therefore, such a transaction can be marked or removed from further consideration since subsequent scans of the database for j-itemsets, where $j > k$, will not require it.

4. **Partitioning**: Partitioning the data to find candidate itemsets.

   A partitioning technique can be used which requires just two database scans to mine the frequent itemsets (Figure 6.7). It consists of two phases. In Phase I, the algorithm subdivides the transactions of D into n non-overlapping partitions. If the minimum support threshold for transactions in D is min_sup, then the minimum itemset support count for a partition is $min\_sup \times$ *the number of transactions in that partition.* For each partition, all frequent itemsets within the partition are found. These are referred to as **local frequent itemsets**. The procedure employs a special data structure which, for each itemset, records the TID's of the transactions containing the items in the itemset. This allows it to find all of the local frequent k-itemsets, for $k = 1, 2, \ldots$, in just one scan of the database.

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

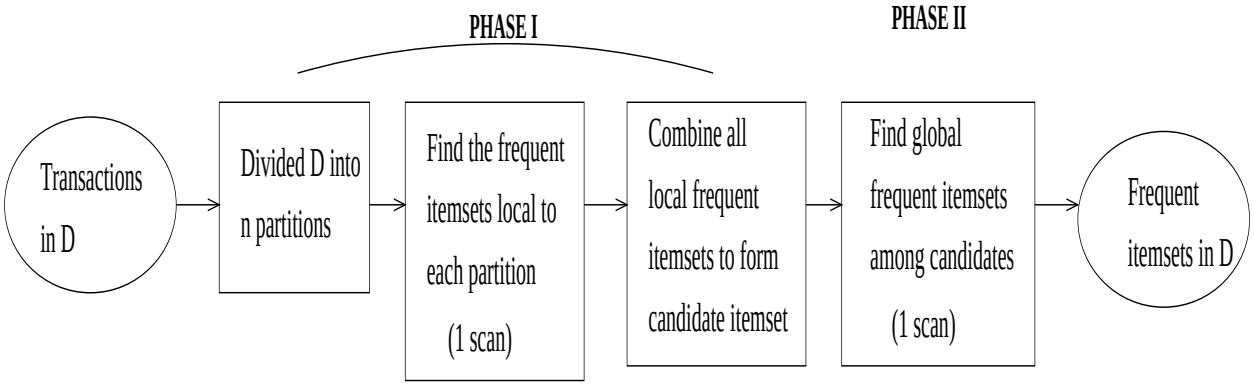


Figure 6.7: Mining by partitioning the data.

5. **Sampling**: Mining on a subset of the given data.

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

6. **Dynamic itemset counting**: Adding candidate itemsets at different points during a scan.

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

7. **Calendric market basket analysis**: Finding itemsets that are frequent in a set of user-defined time intervals.

   Calendric market basket analysis uses transaction time stamps to define subsets of the given database. An itemset that does not satisfy minimum support may be considered frequent with respect to a subset of the database which satisfies user-specified time constraints.

8. **Sequential patterns**: Finding sequences of transactions associated over time.

   The goal of sequential pattern analysis is to find sequences of itemsets that many customers have purchased in roughly the same order. A **transaction sequence** is said to contain an **itemset sequence** if each itemset is contained in one transaction, and the following condition is satisfied: If the ith itemset in the itemset sequence is contained in transaction j in the transaction sequence, then the $(i+1)$th itemset in the itemset sequence is contained in a transaction numbered greater than j. The support of an itemset sequence is the percentage of transaction sequences that contain it.

Other variations involving the mining of multilevel and multidimensional association rules are discussed in the rest of this chapter. The mining of time sequences is further discussed in Chapter 9.

6.3 Mining multilevel association rules from transaction databases

6.3.1 Multilevel association rules

For many applications, it is difficult to find strong associations among data items at low or primitive levels of abstraction due to the sparsity of data in multidimensional space. Strong associations discovered at very high concept levels may represent common sense knowledge. However, what may represent common sense to one user, may seem novel to another. Therefore, data mining systems should provide capabilities to mine association rules at multiple levels of abstraction and traverse easily among different abstraction spaces.

Let's examine the following example.

Example 6.3 Suppose we are given the task-relevant set of transactional data in Table 6.1 for sales at the computer department of an *AllElectronics* branch, showing the items purchased for each transaction TID. The concept hierarchy for the items is shown in Figure 6.8. A concept hierarchy defines a sequence of mappings from a set of low level concepts to higher level, more general concepts. Data can be generalized by replacing low level concepts within the data by their higher level concepts, or *ancestors*, from a concept hierarchy [4]. The concept hierarchy of Figure 6.8 has

[4] Concept hierarchies were described in detail in Chapters 2 and 4. In order to make the chapters of this book as self-contained as possible, we offer their definition again here. Generalization was described in Chapter 5.

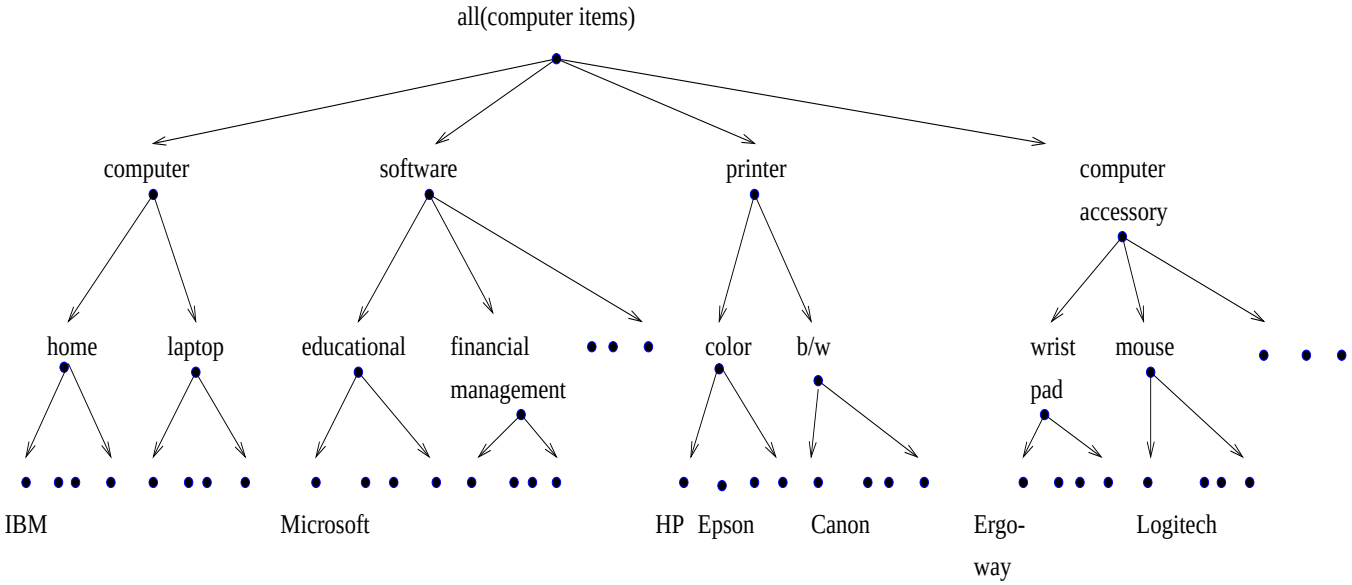


Figure 6.8: A concept hierarchy for *AllElectronics* computer items.

four levels, referred to as levels 0, 1, 2, and 3. By convention, levels within a concept hierarchy are numbered from top to bottom, starting with level 0 at the root node for all (the most general abstraction level). Here, level 1 includes *computer, software, printer* and *computer accessory*, level 2 includes *home computer, laptop computer, education software, financial management software, ..*, and level 3 includes *IBM home computer, .., Microsoft educational software*, and so on. Level 3 represents the most specific abstraction level of this hierarchy. Concept hierarchies may be specified by users familiar with the data, or may exist implicitly in the data.

TID	Items Purchased
1	IBM home computer, Sony b/w printer
2	Microsoft educational software, Microsoft financial management software
3	Logitech mouse computer-accessory, Ergo-way wrist pad computer-accessory
4	IBM home computer, Microsoft financial management software
5	IBM home computer
...	...

Table 6.1: Task-relevant data, D.

The items in Table 6.1 are at the lowest level of the concept hierarchy of Figure 6.8. It is difficult to find interesting purchase patterns at such raw or primitive level data. For instance, if "IBM home computer" or "Sony b/w (black and white) printer" each occurs in a very small fraction of the transactions, then it may be difficult to find strong associations involving such items. Few people may buy such items together, making it is unlikely that the itemset "{IBM home computer, Sony b/w printer}" will satisfy minimum support. However, consider the generalization of "Sony b/w printer" to "b/w printer". One would expect that it is easier to find strong associations between "IBM home computer" and "b/w printer" rather than between "IBM home computer" and "Sony b/w printer". Similarly, many people may purchase "computer" and "printer" together, rather than specifically purchasing "IBM home computer" and "Sony b/w printer" together. In other words, itemsets containing generalized items, such as "{IBM home computers, b/w printer}" and "{computer, printer}" are more likely to have minimum support than itemsets containing only primitive level data, such as "{IBM home computers, Sony b/w printer}". Hence, it is easier to find interesting associations among items at *multiple* concept levels, rather than only among low level data. □

Rules generated from association rule mining with concept hierarchies are called **multiple-level** or **multilevel**

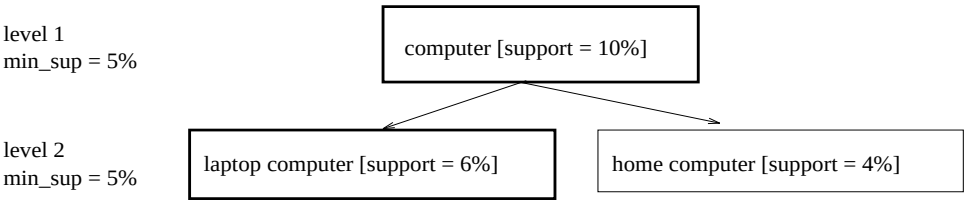


Figure 6.9: Multilevel mining with uniform support.

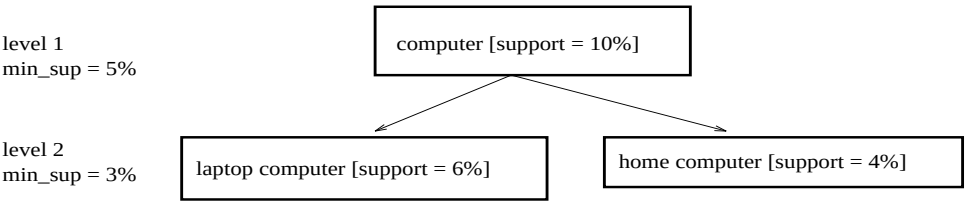


Figure 6.10: Multilevel mining with reduced support.

association rules, since they consider more than one concept level.

6.3.2 Approaches to mining multilevel association rules

"How can we mine multilevel association rules efficiently using concept hierarchies?"

Let's look at some approaches based on a support-confidence framework. In general, a top-down strategy is employed, where counts are accumulated for the calculation of frequent itemsets at each concept level, starting at the concept level 1 and working towards the lower, more specific concept levels, until no more frequent itemsets can be found. That is, once all frequent itemsets at concept level 1 are found, then the frequent itemsets at level 2 are found, and so on. For each level, any algorithm for discovering frequent itemsets may be used, such as Apriori or its variations. A number of variations to this approach are described below, and illustrated in Figures 6.9 to 6.13, where rectangles indicate an item or itemset that has been examined, and rectangles with thick borders indicate that an examined item or itemset is frequent.

1. **Using uniform minimum support for all levels** (referred to as **uniform support**): The same minimum support threshold is used when mining at each level of abstraction. For example, in Figure 6.9, a minimum support threshold of 5% is used throughout (e.g., for mining from "computer" down to "laptop computer"). Both "computer" and "laptop computer" are found to be frequent, while "home computer" is not.

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

   The uniform support approach, however, has some difficulties. It is unlikely that items at lower levels of abstraction will occur as frequently as those at higher levels of abstraction. If the minimum support threshold is set too high, it could miss several meaningful associations occurring at low abstraction levels. If the threshold is

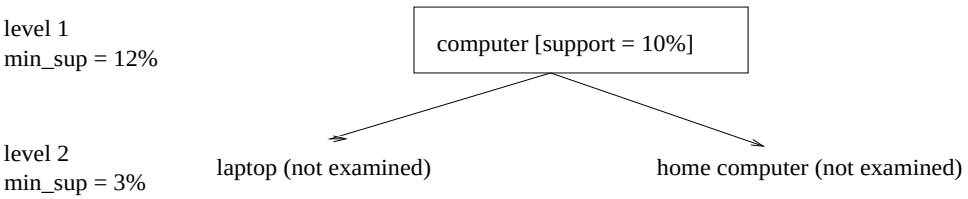


Figure 6.11: Multilevel mining with reduced support, using level-cross filtering by a single item.

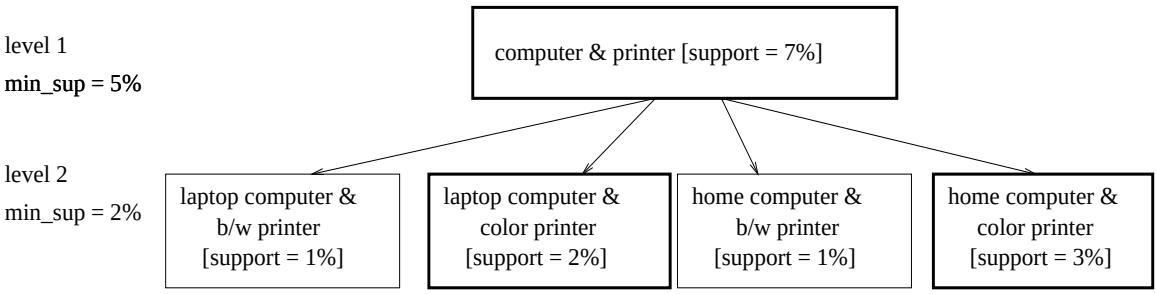


Figure 6.12: Multilevel mining with reduced support, using level-cross filtering by a k-itemset. Here, $k = 2$.

set too low, it may generate many uninteresting associations occurring at high abstraction levels. This provides the motivation for the following approach.

2. **Using reduced minimum support at lower levels** (referred to as **reduced support**): Each level of abstraction has its own minimum support threshold. The lower the abstraction level is, the smaller the corresponding threshold is. For example, in Figure 6.10, the minimum support thresholds for levels 1 and 2 are 5% and 3%, respectively. In this way, "computer", "laptop computer", and "home computer" are all considered frequent.

For mining multiple-level associations with *reduced support*, there are a number of alternative search strategies. These include:

1. **level-by-level independent**: This is a full breadth search, where no background knowledge of frequent itemsets is used for pruning. Each node is examined, regardless of whether or not its parent node is found to be frequent.

2. **level-cross filtering by single item**: An item at the i-th level is examined if and only if its parent node at the $(i-1)$-th level is frequent. In other words, we investigate a more specific association from a more general one. If a node is frequent, its children will be examined; otherwise, its descendents are pruned from the search. For example, in Figure 6.11, the descendent nodes of "computer" (i.e., "laptop computer" and "home computer") are not examined, since "computer" is not frequent.

3. **level-cross filtering by k-itemset**: A k-itemset at the i-th level is examined if and only if its corresponding parent k-itemset at the $(i-1)$-th level is frequent. For example, in Figure 6.12, the 2-itemset "{computer, printer}" is frequent, therefore the nodes "{laptop computer, b/w printer}", "{laptop computer, color printer}", "{home computer, b/w printer}", and "{home computer, color printer}" are examined.

"How do these methods compare?"

The *level-by-level independent* strategy is very relaxed in that it may lead to examining numerous infrequent items at low levels, finding associations between items of little importance. For example, if "computer furniture" is rarely purchased, it may not be beneficial to examine whether the more specific "computer chair" is associated with "laptop". However, if "computer accessories" are sold frequently, it may be beneficial to see whether there is an associated purchase pattern between "laptop" and "mouse".

The *level-cross filtering by k-itemset* strategy allows the mining system to examine only the children of frequent k-itemsets. This restriction is very strong in that there usually are not many k-itemsets (especially when $k > 2$) which, when combined, are also frequent. Hence, many valuable patterns may be filtered out using this approach.

The *level-cross filtering by single item* strategy represents a compromise between the two extremes. However, this method may miss associations between low level items that are frequent based on a reduced minimum support, but whose ancestors do not satisfy minimum support (since the support thresholds at each level can be different). For example, if "color monitor" occurring at concept level i is frequent based on the minimum support threshold of level i, but its parent "monitor" at level $(i-1)$ is not frequent according to the minimum support threshold of level $(i-1)$, then frequent associations such as "*home computer* $\Rightarrow$ *color monitor*" will be missed.

A modified version of the *level-cross filtering by single item* strategy, known as the **controlled level-cross filtering by single item** strategy, addresses the above concern as follows. A threshold, called the **level passage**

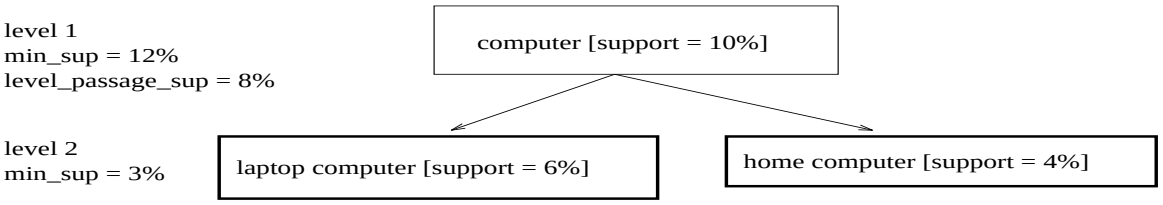


Figure 6.13: Multilevel mining with controlled level-cross filtering by single item

.

threshold, can be set up for "passing down" relatively frequent items (called **subfrequent items**) to lower levels. In other words, this method allows the children of items that do not satisfy the minimum support threshold to be examined if these items satisfy the level passage threshold. Each concept level can have its own level passage threshold. The level passage threshold for a given level is typically set to a value between the minimum support threshold of the next lower level and the minimum support threshold of the given level. Users may choose to "slide down" or lower the level passage threshold at high concept levels to allow the descendents of the subfrequent items at lower levels to be examined. Sliding the level passage threshold down to the minimum support threshold of the lowest level would allow the descendents of all of the items to be examined. For example, in Figure 6.13, setting the level passage threshold (*level_passage_sup*) of level 1 to 8% allows the nodes "laptop computer" and "home computer" at level 2 to be examined and found frequent, even though their parent node, "computer", is not frequent. By adding this mechanism, users have the flexibility to further control the mining process at multiple abstraction levels, as well as reduce the number of meaningless associations that would otherwise be examined and generated.

So far, our discussion has focussed on finding frequent itemsets where all items within the itemset must belong to the same concept level. This may result in rules such as "*computer* $\Rightarrow$ *printer*" (where "computer" and "printer" are both at concept level 1) and "*home computer* $\Rightarrow$ *b/w printer*" (where "home computer" and "b/w printer" are both at level 2 of the given concept hierarchy). Suppose, instead, that we would like to find rules that *cross concept level boundaries*, such as "*computer* $\Rightarrow$ *b/w printer*", where items within the rule are not required to belong to the same concept level. These rules are called **cross-level association rules**.

"How can cross-level associations be mined?" If mining associations from concept levels i and j, where level j is more specific (i.e., at a lower abstraction level) than i, then the reduced minimum support threshold of level j should be used overall so that items from level j can be included in the analysis.

6.3.3 Checking for redundant multilevel association rules

Concept hierarchies are useful in data mining since they permit the discovery of knowledge at different levels of abstraction, such as multilevel association rules. However, when multilevel association rules are mined, some of the rules found will be redundant due to "ancestor" relationships between items. For example, consider Rules (6.9) and (6.10) below, where "home computer" is an ancestor of "IBM home computer" based on the concept hierarchy of Figure 6.8.

$$home\ computer \Rightarrow b/w\ printer, \qquad [support = 8\%, confidence = 70\%] \tag{6.9}$$

$$IBM\ home\ computer \Rightarrow b/w\ printer, \qquad [support = 2\%, confidence = 72\%] \tag{6.10}$$

"If Rules (6.9) and (6.10) are both mined, then how useful is the latter rule?", you may wonder. *"Does it really provide any novel information?"*

If the latter, less general rule does not provide new information, it should be removed. Let's have a look at how this may be determined. A rule $R1$ is an **ancestor** of a rule $R2$ if $R1$ can be obtained by replacing the items in $R2$ by their ancestors in a concept hierarchy. For example, Rule (6.9) is an ancestor of Rule (6.10) since "home computer" is an ancestor of "IBM home computer". Based on this definition, a rule can be considered redundant if its support and confidence are close to their "expected" values, based on an ancestor of the rule. As an illustration, suppose that Rule (6.9) has a 70% confidence and 8% support, and that about one quarter of all "home computer" sales are for "IBM home computers", and a quarter of all "printers" sales are "black/white printers" sales. One may expect

Rule (6.10) to have a confidence of around 70% (since all data samples of "IBM home computer" are also samples of "home computer") and a support of 2% (i.e., $8\% \times \frac{1}{4}$). If this is indeed the case, then Rule (6.10) is not interesting since it does not offer any additional information and is less general than Rule (6.9).

6.4 Mining multidimensional association rules from relational databases and data warehouses

6.4.1 Multidimensional association rules

Up to this point in this chapter, we have studied association rules which imply a single predicate, that is, the predicate *buys*. For instance, in mining our *AllElectronics* database, we may discover the Boolean association rule "*IBM home computer* $\Rightarrow$ *Sony b/w printer*", which can also be written as

$$buys(X, \text{“}IBM\ home\ computer\text{”}) \Rightarrow buys(X, \text{“}Sony\ b/w\ printer\text{”}), \tag{6.11}$$

where X is a variable representing customers who purchased items in *AllElectronics* transactions. Similarly, if "printer" is a generalization of "Sony b/w printer", then a multilevel association rule like "*IBM home computers* $\Rightarrow$ *printer*" can be expressed as

$$buys(X, \text{“}IBM\ home\ computer\text{”}) \Rightarrow buys(X, \text{“}printer\text{”}). \tag{6.12}$$

Following the terminology used in multidimensional databases, we refer to each distinct predicate in a rule as a dimension. Hence, we can refer to Rules (6.11) and (6.12) as **single-dimensional** or **intra-dimension association rules** since they each contain a single distinct predicate (e.g., *buys*) with multiple occurrences (i.e., the predicate occurs more than once within the rule). As we have seen in the previous sections of this chapter, such rules are commonly mined from transactional data.

Suppose, however, that rather than using a transactional database, sales and related information are stored in a relational database or data warehouse. Such data stores are multidimensional, by definition. For instance, in addition to keeping track of the items purchased in sales transactions, a relational database may record other attributes associated with the items, such as the quantity purchased or the price, or the branch location of the sale. Addition relational information regarding the customers who purchased the items, such as customer age, occupation, credit rating, income, and address, may also be stored. Considering each database attribute or warehouse dimension as a predicate, it can therefore be interesting to mine association rules containing *multiple* predicates, such as

$$age(X, \text{“}19-24\text{”}) \wedge occupation(X, \text{“}student\text{”}) \Rightarrow buys(X, \text{“}laptop\text{”}). \tag{6.13}$$

Association rules that involve two or more dimensions or predicates can be referred to as **multidimensional association rules**. Rule (6.13) contains three predicates (*age, occupation*, and *buys*), each of which occurs *only once* in the rule. Hence, we say that it has **no repeated predicates**. Multidimensional association rules with no repeated predicates are called **inter-dimension association rules**. We may also be interested in mining multidimensional association rules with repeated predicates, which contain multiple occurrences of some predicate. These rules are called **hybrid-dimension association rules**. An example of such a rule is Rule (6.14), where the predicate *buys* is repeated.

$$age(X, \text{“}19-24\text{”}) \wedge buys(X, \text{“}laptop\text{”}) \Rightarrow buys(X, \text{“}b/w\ printer\text{”}). \tag{6.14}$$

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

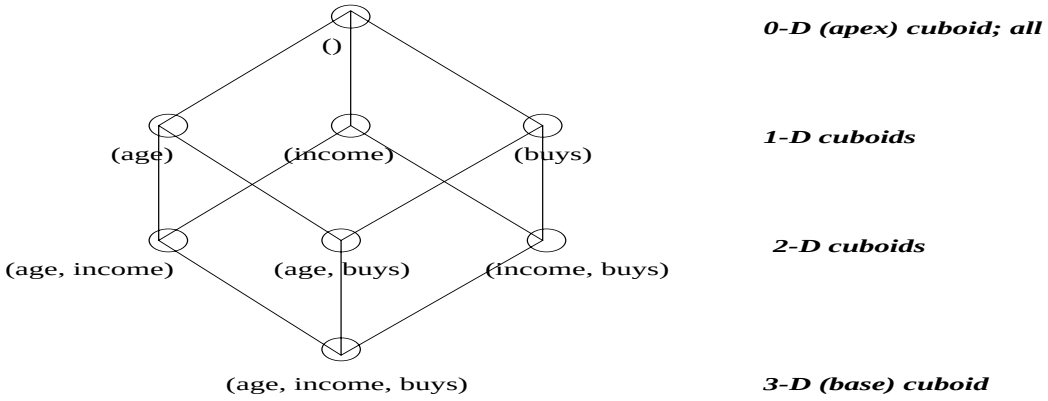


Figure 6.14: Lattice of cuboids, making up a 3-dimensional data cube. Each cuboid represents a different group-by. The base cuboid contains the three predicates, *age, income*, and *buys*.

1. In the first approach, *quantitative attributes are discretized using predefined concept hierarchies.* This discretization occurs prior to mining. For instance, a concept hierarchy for *income* may be used to replace the original numeric values of this attribute by ranges, such as *"0-20K", "21-30K", "31-40K"*, and so on. Here, discretization is *static* and predetermined. The discretized numeric attributes, with their range values, can then be treated as categorical attributes (where each range is considered a category). We refer to this as **mining multidimensional association rules using static discretization of quantitative attributes**.

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

3. In the third approach, *quantitative attributes are discretized so as to capture the semantic meaning of such interval data.* This dynamic discretization procedure considers the distance between data points. Hence, such quantitative association rules are also referred to as **distance-based association rules**.

Let's study each of these approaches for mining multidimensional association rules. For simplicity, we confine our discussion to inter-dimension association rules. Note that rather than searching for frequent itemsets (as is done for single-dimensional association rule mining), in multidimensional association rule mining we search for frequent *predicatesets*. A **k-predicateset** is a set containing k conjunctive predicates. For instance, the set of predicates {*age, occupation, buys*} from Rule (6.13) is a 3-predicateset. Similar to the notation used for itemsets, we use the notation L_k to refer to the set of frequent k-predicatesets.

6.4.2 Mining multidimensional association rules using static discretization of quantitative attributes

Quantitative attributes, in this case, are discretized prior to mining using predefined concept hierarchies, where numeric values are replaced by ranges. Categorical attributes may also be generalized to higher conceptual levels if desired.

If the resulting task-relevant data are stored in a relational table, then the Apriori algorithm requires just a slight modification so as to find all frequent predicatesets rather than frequent itemsets (i.e., by searching through all of the relevant attributes, instead of searching only one attribute, like *buys*). Finding all frequent k-predicatesets will require k or $k + 1$ scans of the table. Other strategies, such as hashing, partitioning, and sampling may be employed to improve the performance.

Alternatively, the transformed task-relevant data may be stored in a *data cube*. Data cubes are well-suited for the mining of multidimensional association rules, since they are multidimensional by definition. Data cubes, and their computation, were discussed in detail in Chapter 2. To review, a data cube consists of a lattice of cuboids which

are multidimensional data structures. These structures can hold the given task-relevant data, as well as aggregate, group-by information. Figure 6.14 shows the lattice of cuboids defining a data cube for the dimensions *age, income*, and *buys*. The cells of an n-dimensional cuboid are used to store the counts, or support, of the corresponding n-predicatesets. The base cuboid aggregates the task-relevant data by *age, income*, and *buys*; the 2-D cuboid, *(age, income)*, aggregates by *age* and *income*; the 0-D (apex) cuboid contains the total number of transactions in the task relevant data, and so on.

Due to the ever-increasing use of data warehousing and OLAP technology, it is possible that a data cube containing the dimensions of interest to the user may already exist, fully materialized. *"If this is the case, how can we go about finding the frequent predicatesets?"* A strategy similar to that employed in Apriori can be used, based on prior knowledge that *every subset of a frequent predicateset must also be frequent*. This property can be used to reduce the number of candidate predicatesets generated.

In cases where no relevant data cube exists for the mining task, one must be created. Chapter 2 describes algorithms for fast, efficient computation of data cubes. These can be modified to search for frequent itemsets during cube construction. Studies have shown that even when a cube must be constructed on the fly, mining from data cubes can be faster than mining directly from a relational table.

6.4.3 Mining quantitative association rules

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

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

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

$$age(X, \text{“}30 - 34\text{”}) \wedge income(X, \text{“}42K - 48K\text{”}) \Rightarrow buys(X, \text{“}high\ resolution\ TV\text{”}) \quad (6.15)$$

"How can we find such rules?" Let's look at an approach used in a system called ARCS (Association Rule Clustering System) which borrows ideas from image-processing. Essentially, this approach maps pairs of quantitative attributes onto a 2-D grid for tuples satisfying a given categorical attribute condition. The grid is then searched for clusters of points, from which the association rules are generated. The following steps are involved in ARCS:

Binning. Quantitative attributes can have a very wide range of values defining their domain. Just think about how big a 2-D grid would be if we plotted *age* and *income* as axes, where each possible value of *age* was assigned a unique position on one axis, and similarly, each possible value of *income* was assigned a unique position on the other axis! To keep grids down to a manageable size, we instead partition the ranges of quantitative attributes into intervals. These intervals are dynamic in that they may later be further combined during the mining process. The partitioning process is referred to as **binning**, i.e., where the intervals are considered "bins". Three common binning strategies are:

1. **equi-width binning**, where the interval size of each bin is the same,
2. **equi-depth binning**, where each bin has approximately the same number of tuples assigned to it, and
3. **homogeneity-based binning**, where bin size is determined so that the tuples in each bin are uniformly distributed.

In ARCS, equi-width binning is used, where the bin size for each quantitative attribute is input by the user. A 2-D array for each possible bin combination involving both quantitative attributes is created. Each array cell holds

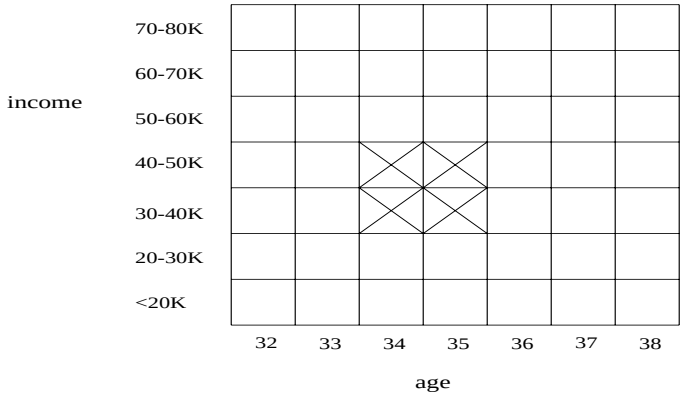


Figure 6.15: A 2-D grid for tuples representing customers who purchase high resolution TVs

.

the corresponding count distribution for each possible class of the categorical attribute of the rule right-hand side. By creating this data structure, the task-relevant data need only be scanned once. The same 2-D array can be used to generate rules for any value of the categorical attribute, based on the same two quantitative attributes. Binning is also discussed in Chapter 3.

Finding frequent predicatesets. Once the 2-D array containing the count distribution for each category is set up, this can be scanned in order to find the frequent predicatesets (those satisfying minimum support) that also satisfy minimum confidence. Strong association rules can then be generated from these predicatesets, using a rule generation algorithm like that described in Section 6.2.2.

Clustering the association rules. The strong association rules obtained in the previous step are then mapped to a 2-D grid. Figure 6.15 shows a 2-D grid for 2-D quantitative association rules predicting the condition *buys(X, "high resolution TV")* on the rule right-hand side, given the quantitative attributes *age* and *income*. The four "X"'s correspond to the rules

$$age(X, 34) \wedge income(X, \text{“}30 - 40K\text{”}) \Rightarrow buys(X, \text{“}high\ resolution\ TV\text{”}) \quad (6.16)$$

$$age(X, 35) \wedge income(X, \text{“}30 - 40K\text{”}) \Rightarrow buys(X, \text{“}high\ resolution\ TV\text{”}) \quad (6.17)$$

$$age(X, 34) \wedge income(X, \text{“}40 - 50K\text{”}) \Rightarrow buys(X, \text{“}high\ resolution\ TV\text{”}) \quad (6.18)$$

$$age(X, 35) \wedge income(X, \text{“}40 - 50K\text{”}) \Rightarrow buys(X, \text{“}high\ resolution\ TV\text{”}) \quad (6.19)$$

"Can we find a simpler rule to replace the above four rules?" Notice that these rules are quite "close" to one another, forming a rule cluster on the grid. Indeed, the four rules can be combined or "clustered" together to form Rule (6.20) below, a simpler rule which subsumes and replaces the above four rules.

$$age(X, \text{“}34 - 35\text{”}) \wedge income(X, \text{“}30 - 50K\text{”}) \Rightarrow buys(X, \text{“}high\ resolution\ TV\text{”}) \quad (6.20)$$

ARCS employs a clustering algorithm for this purpose. The algorithm scans the grid, searching for rectangular clusters of rules. In this way, bins of the quantitative attributes occurring within a rule cluster may be further combined, and hence, further dynamic discretization of the quantitative attributes occurs.

The grid-based technique described here assumes that the initial association rules can be clustered into rectangular regions. Prior to performing the clustering, smoothing techniques can be used to help remove noise and outliers from the data. Rectangular clusters may oversimplify the data. Alternative approaches have been proposed, based on other shapes of regions which tend to better fit the data, yet require greater computation effort.

A non-grid-based technique has been proposed to find more general quantitative association rules where any number of quantitative and categorical attributes can appear on either side of the rules. In this technique, quantitative attributes are dynamically partitioned using equi-depth binning, and the partitions are combined based on a measure of *partial completeness* which quantifies the information lost due to partitioning. For references on these alternatives to ARCS, see the bibliographic notes.

Price ($)	Equi-width (width $10)	Equi-depth (depth $2)	Distance-based
7	[0, 10]	[7, 20]	[7, 7]
20	[11, 20]	[22, 50]	[20, 22]
22	[21, 30]	[51, 53]	[50, 53]
50	[31, 40]		
51	[41, 50]		
53	[51, 60]		

Figure 6.16: Binning methods like equi-width and equi-depth do not always capture the semantics of interval data.

6.4.4 Mining distance-based association rules

The previous section described quantitative association rules where quantitative attributes are discretized initially by binning methods, and the resulting intervals are then combined. Such an approach, however, may not capture the semantics of interval data since they do not consider the relative distance between data points or between intervals.

Consider, for example, Figure 6.16 which shows data for the attribute *price*, partitioned according to equi-width and equi-depth binning versus a distance-based partitioning. The distance-based partitioning seems the most intuitive, since it groups values that are close together within the same interval (e.g., [20, 22]). In contrast, equi-depth partitioning groups distant values together (e.g., [22, 50]). Equi-width may split values that are close together and create intervals for which there are no data. Clearly, a distance-based partitioning which considers the density or number of points in an interval, as well as the "closeness" of points in an interval helps produce a more meaningful discretization. Intervals for each quantitative attribute can be established by *clustering* the values for the attribute.

A disadvantage of association rules is that they do not allow for approximations of attribute values. Consider association rule (6.21):

$$item\_type(X, \text{“}electronic\text{”}) \wedge manufacturer(X, \text{“}foreign\text{”}) \Rightarrow price(X, \$200). \quad (6.21)$$

In reality, it is more likely that the prices of foreign electronic items are *close to or approximately* $200, rather than exactly $200. It would be useful to have association rules that can express such a notion of closeness. Note that the support and confidence measures do not consider the closeness of values for a given attribute. This motivates the mining of **distance-based association rules** which capture the semantics of interval data while allowing for approximation in data values. Distance-based based association rules can be mined by first employing clustering techniques to find the intervals or clusters, and then searching for groups of clusters that occur frequently together.

Clusters and distance measurements

"What kind of distance-based measurements can be used for identifying the clusters?", you wonder. *"What defines a cluster?"*

Let $S[X]$ be a set of N tuples $t_1, t_2, .., t_N$ projected on the attribute set X. The **diameter**, d, of $S[X]$ is the average pairwise distance between the tuples projected on X. That is,

$$d(S[X]) = \frac{\sum_{i=1}^{N} \sum_{j=1}^{N} dist_X(t_i[X], t_j[X])}{N(N-1)}, \quad (6.22)$$

where $dist_X$ is a distance metric on the values for the attribute set X, such as the Euclidean distance or the Manhattan. For example, suppose that X contains m attributes. The **Euclidean distance** between two tuples $t_1 = (x_{11}, x_{12}, .., x_{1m})$ and $t_2 = (x_{21}, x_{22}, .., x_{2m})$ is

$$Euclidean\_d(t_1, t_2) = \sqrt{\sum_{i=1}^{m} (x_{1i} - x_{2i})^2}. \quad (6.23)$$

The **Manhattan (city block) distance** between t_1 and t_2 is

$$Manhattan\_d(t_1, t_2) = \sum_{i=1}^{m} |x_{1i} - x_{2i}|. \tag{6.24}$$

The diameter metric assesses the closeness of tuples. The smaller the diameter of $S[X]$ is, the "closer" its tuples are when projected on X. Hence, the diameter metric assesses the *density* of a cluster. A **cluster** C_X is a set of tuples defined on an attribute set X, where the tuples satisfy a **density threshold**, d_0^X, and a **frequency threshold**, s_0, such that:

$$d(C_X) \leq d_0^X \tag{6.25}$$

$$|C_X| \geq s_0. \tag{6.26}$$

Clusters can be combined to form distance-based association rules. Consider a simple distance-based association rule of the form $C_X \Rightarrow C_Y$. Suppose that X is the attribute set $\{age\}$ and Y is the attribute set $\{income\}$. We want to ensure that the implication between the cluster C_X for *age* and C_Y for *income* is strong. This means that when the age-clustered tuples C_X are projected onto the attribute *income*, their corresponding *income* values lie within the income-cluster C_Y, or close to it. A cluster C_X projected onto the attribute set Y is denoted $C_X[Y]$. Therefore, the distance between $C_X[Y]$ and $C_Y[Y]$ must be small. This distance measures the *degree of association* between C_X and C_Y. The smaller the distance between $C_X[Y]$ and $C_Y[Y]$ is, the stronger the degree of association between C_X and C_Y is. The degree of association measure can be defined using standard statistical measures, such as the average inter-cluster distance, or the centroid Manhattan distance, where the centroid of a cluster represents the "average" tuple of the cluster.

Finding clusters and distance-based rules

An adaptive two-phase algorithm can be used to find distance-based association rules, where clusters are identified in the first phase, and combined in the second phase to form the rules.

A modified version of the BIRCH[5] clustering algorithm is used in the first phase, which requires just one pass through the data. To compute the distance between clusters, the algorithm maintains a data structure called an *association clustering feature* for each cluster which maintains information about the cluster and its projection onto other attribute sets. The clustering algorithm adapts to the amount of available memory.

In the second phase, clusters are combined to find distance-based association rules of the form

$C_{X_1}C_{X_2}..C_{X_x} \Rightarrow C_{Y_1}C_{Y_2}..C_{Y_y}$,

where X_i and Y_j are pairwise disjoint sets of attributes, D is measure of the degree of association between clusters as described above, and the following conditions are met:

1. The clusters in the rule antecedent each are strongly associated with each cluster in the consequent. That is, $D(C_{Y_j}[Y_j], C_{X_i}[Y_j]) \leq D_0$, $1 \leq i \leq x, 1 \leq j \leq y$, where D_0 is the **degree of association threshold**.

2. The clusters in the antecedent collectively occur together. That is, $D(C_{X_i}[X_i], C_{X_j}[X_i]) \leq d_0^{X_i}$ $\forall i \neq j$.

3. The clusters in the consequent collectively occur together. That is, $D(C_{Y_i}[Y_i], C_{Y_j}[Y_i]) \leq d_0^{Y_i}$ $\forall i \neq j$, where $d_0^{Y_i}$ is the density threshold on attribute set Y_i.

The degree of association replaces the confidence framework in non-distance-based association rules, while the density threshold replaces the notion of support.

Rules are found with the help of a clustering graph, where each node in the graph represents a cluster. An edge is drawn from one cluster node, n_{C_X}, to another, n_{C_Y}, if $D(C_X[X], C_Y[X]) \leq d_0^X$ and $D(C_X[Y], C_Y[Y]) \leq d_0^Y$. A **clique** in such a graph is a subset of nodes, each pair of which is connected by an edge. The algorithm searches for all maximal cliques. These correspond to frequent itemsets from which the distance-based association rules can be generated.

[5] The BIRCH clustering algorithm is described in detail in Chapter 8 on clustering.

6.5 From association mining to correlation analysis

"When mining association rules, how can the data mining system tell which rules are likely to be interesting to the user?"

Most association rule mining algorithms employ a support-confidence framework. In spite of using minimum support and confidence thresholds to help weed out or exclude the exploration of uninteresting rules, many rules that are not interesting to the user may still be produced. In this section, we first look at how even strong association rules can be uninteresting and misleading, and then discuss additional measures based on statistical independence and correlation analysis.

6.5.1 Strong rules are not necessarily interesting: An example

"In data mining, are all of the strong *association rules discovered (i.e., those rules satisfying the minimum support and minimum confidence thresholds) interesting enough to present to the user?"* Not necessarily. Whether a rule is interesting or not can be judged either subjectively or objectively. Ultimately, only the user can judge if a given rule is interesting or not, and this judgement, being subjective, may differ from one user to another. "behind" the data, can be used as one step towards the goal of weeding out uninteresting rules from presentation to the user.

"So, how can we tell which strong association rules are really interesting?" Let's examine the following example.

Example 6.4 Suppose we are interested in analyzing transactions at *AllElectronics* with respect to the purchase of computer games and videos. The event *game* refers to the transactions containing computer games, while *video* refers to those containing videos. Of the $10,000$ transactions analyzed, the data show that $6,000$ of the customer transactions included computer games, while $7,500$ included videos, and $4,000$ included both computer games and videos. Suppose that a data mining program for discovering association rules is run on the data, using a minimum support of, say, 30% and a minimum confidence of 60%. The following association rule is discovered.

$$buys(X, \text{“}computer\ games\text{”}) \Rightarrow buys(X, \text{“}videos\text{”}), \qquad [support = 40\%, confidence = 66\%] \tag{6.27}$$

Rule (6.27) is a strong association rule and would therefore be reported, since its support value of $\frac{4,000}{10,000} = 40\%$ and confidence value of $\frac{4,000}{6,000} = 66\%$ satisfy the minimum support and minimum confidence thresholds, respectively. However, Rule (6.27) is misleading since the probability of purchasing videos is 75%, which is even larger than 66%. In fact, computer games and videos are negatively associated because the purchase of one of these items actually decreases the likelihood of purchasing the other. Without fully understanding this phenomenon, one could make unwise business decisions based on the rule derived. □

The above example also illustrates that the confidence of a rule $A \Rightarrow B$ can be deceiving in that it is only an *estimate* of the conditional probability of B given A. It does not measure the real strength (or lack of strength) of the implication between A and B. Hence, alternatives to the support-confidence framework can be useful in mining interesting data relationships.

6.5.2 From association analysis to correlation analysis

Association rules mined using a support-confidence framework are useful for many applications. However, the support-confidence framework can be misleading in that it may identify a rule $A \Rightarrow B$ as interesting, when in fact, A does not imply B. In this section, we consider an alternative framework for finding interesting relationships between data items based on correlation.

Two events A and B are **independent** if $P(A) \wedge P(B) = 1$, otherwise A and B are **dependent** and **correlated**. This definition can easily be extended to more than two variables. The **correlation** between A and B can be measured by computing

$$\frac{P(A \wedge B)}{P(A)P(B)}. \tag{6.28}$$

	game	$\overline{game}$	Σ_{row}
video	4,000	3,500	7,500
$\overline{video}$	2,000	500	2,500
Σ_{col}	6,000	4,000	10,000

Table 6.2: A contingency table summarizing the transactions with respect to computer game and video purchases.

If the resulting value of Equation (6.28) is less than 1, then A and B are negatively correlated, meaning that each event discourages the occurrence of the other. If the resulting value is greater than 1, then A and B are positively correlated, meaning that each event implies the other. If the resulting value is equal to 1, then A and B are independent and there is no correlation between them.

Let's go back to the computer game and video data of Example 6.4.

Example 6.5 To help filter out misleading "strong" associations of the form $A \Rightarrow B$, we need to study how the two events, A and B, are correlated. Let $\overline{game}$ refer to the transactions of Example 6.4 which do not contain computer games, and $\overline{video}$ refer to those that do not contain videos. The transactions can be summarized in a **contingency table**. A contingency table for the data of Example 6.4 is shown in Table 6.2. From the table, one can see that the probability of purchasing a computer game is $P(game) = 0.60$, the probability of purchasing a video is $P(video) = 0.75$, and the probability of purchasing both is $P(game \wedge video) = 0.40$. By Equation (6.28), $P(game \wedge video)/(P(game) \times P(video)) = 0.40/(0.75 \times 0.60) = 0.89$. Since this value is significantly less than 1, there is a negative correlation between computer games and videos. The nominator is the likelihood of a customer purchasing both, while the denominator is what the likelihood would have been if the two purchases were completely independent. Such a negative correlation cannot be identified by a support-confidence framework. □

This motivates the mining of rules that identify correlations, or *correlation rules*. A **correlation rule** is of the form $\{e_1, e_2, .., e_m\}$ where the occurrences of the events (or items) $\{e_1, e_2, .., e_m\}$ are correlated. Given a correlation value determined by Equation (6.28), the χ^2 statistic can be used to determine if the correlation is statistically significant. The χ^2 statistic can also determine negative implication.

An advantage of correlation is that it is *upward closed*. This means that if a set S of items is correlated (i.e., the items in S are correlated), then every superset of S is also correlated. In other words, adding items to a set of correlated items does not remove the existing correlation. The χ^2 statistic is also upward closed within each significance level.

When searching for sets of correlations to form correlation rules, the upward closure property of correlation and χ^2 can be used. Starting with the empty set, we may explore the itemset space (or *itemset lattice*), adding one item at a time, looking for **minimal correlated itemsets** - itemsets that are correlated although no subset of them is correlated. These itemsets form a **border** within the lattice. Because of closure, no itemset below this border will be correlated. Since all supersets of a minimal correlated itemset are correlated, we can stop searching upwards. An algorithm that perform a series of such "walks" through itemset space is called a **random walk algorithm**. Such an algorithm can be combined with tests of support in order to perform additional pruning. Random walk algorithms can easily be implemented using data cubes. It is an open problem to adapt the procedure described here to very lary databases. Another limitation is that the χ^2 statistic is less accurate when the contingency table data are sparse. More research is needed in handling such cases.

6.6 Constraint-based association mining

For a given set of task-relevant data, the data mining process may uncover thousands of rules, many of which are uninteresting to the user. In **constraint-based mining**, mining is performed under the guidance of various kinds of constraints provided by the user. These constraints include the following.

1. **Knowledge type constraints**: These specify the type of knowledge to be mined, such as association.

2. **Data constraints**: These specify the set of task-relevant data.

3. **Dimension/level constraints**: These specify the dimension of the data, or levels of the concept hierarchies, to be used.

4. **Interestingness constraints**: These specify thresholds on statistical measures of rule interestingness, such as support and confidence.

5. **Rule constraints**. These specify the form of rules to be mined. Such constraints may be expressed as metarules (rule templates), or by specifying the maximum or minimum number of predicates in the rule antecedent or consequent, or the satisfaction of particular predicates on attribute values, or their aggregates.

The above constraints can be specified using a high-level declarative data mining query language, such as that described in Chapter 4.

The first four of the above types of constraints have already been addressed in earlier parts of this book and chapter. In this section, we discuss the use of rule constraints to focus the mining task. This form of constraint-based mining enriches the relevance of the rules mined by the system to the users' intentions, thereby making the data mining process more *effective*. In addition, a sophisticated mining query optimizer can be used to exploit the constraints specified by the user, thereby making the mining process more *efficient*.

Constraint-based mining encourages interactive exploratory mining and analysis. In Section 6.6.1, you will study metarule-guided mining, where syntactic rule constraints are specified in the form of rule templates. Section 6.6.2 discusses the use of additional rule constraints, specifying set/subset relationships, constant initiation of variables, and aggregate functions. The examples in these sections illustrate various data mining query language primitives for association mining.

6.6.1 Metarule-guided mining of association rules

"How are metarules useful?"

Metarules allow users to specify the syntactic form of rules that they are interested in mining. The rule forms can be used as constraints to help improve the efficiency of the mining process. Metarules may be based on the analyst's experience, expectations, or intuition regarding the data, or automatically generated based on the database schema.

Example 6.6 Suppose that as a market analyst for *AllElectronics*, you have access to the data describing customers (such as customer age, address, and credit rating) as well as the list of customer transactions. You are interested in finding associations between customer traits and the items that customers buy. However, rather than finding *all* of the association rules reflecting these relationships, you are particularly interested only in determining which pairs of customer traits promote the sale of educational software. A metarule can be used to specify this information describing the form of rules you are interested in finding. An example of such a metarule is

$$P_1(X, Y) \wedge P_2(X, W) \Rightarrow buys(X, \text{“}educational\ software\text{”}), \tag{6.29}$$

where P_1 and P_2 are **predicate variables** that are instantiated to attributes from the given database during the mining process, X is a variable representing a customer, and Y and W take on values of the attributes assigned to P_1 and P_2, respectively. Typically, a user will specify a list of attributes to be considered for instantiation with P_1 and P_2. Otherwise, a default set may be used.

In general, a metarule forms a hypothesis regarding the relationships that the user is interested in probing or confirming. The data mining system can then search for rules that match the given metarule. For instance, Rule (6.30) matches or **complies with** Metarule (6.29).

$$age(X, \text{“}35 - 45\text{”}) \wedge income(X, \text{“}40 - 60K\text{”}) \Rightarrow buys(X, \text{“}educational\ software\text{”}) \tag{6.30}$$

□

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

$$P_1 \wedge P_2 \wedge \ldots \wedge P_l \Rightarrow Q_1 \wedge Q_2 \wedge \ldots \wedge Q_r \tag{6.31}$$

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

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

This is a typical case of mining multidimensional association rules, which was described in Section 6.4. As shown there, data cubes are well-suited to the mining of multidimensional association rules owing to their ability to store aggregate dimension values. Owing to the popularity of OLAP and data warehousing, it is possible that a fully materialized n-D data cube suitable for the given mining task already exists, where n is the number of attributes to be considered for instantiation with the predicate variables plus the number of predicates already instantiated in the given metarule, and $n \geq p$. Such an n-D cube is typically represented by a lattice of cuboids, similar to that shown in Figure 6.14. In this case, we need only scan the p-D cuboids, comparing each cell count with the minimum support threshold, in order to find L_p. Since the l-D cuboids have already been computed and contain the counts of the l-D predicate subsets of L_p, a rule generation procedure can then be called to return strong rules that comply with the given metarule. We call this approach an **abridged n-D cube search**, since rather than searching the entire n-D data cube, only the p-D and l-D cuboids are ever examined.

If a relevant n-D data cube does not exist for the metarule-guided mining task, then one must be constructed and searched. Rather than constructing the entire cube, only the p-D and l-D cuboids need be computed. Methods for cube construction are discussed in Chapter 2.

6.6.2 Mining guided by additional rule constraints

Rule constraints specifying set/subset relationships, constant initiation of variables, and aggregate functions can be specified by the user. These may be used together with, or as an alternative to, metarule-guided mining. In this section, we examine rule constraints as to how they can be used to make the mining process more efficient. Let us study an example where rule constraints are used to mine hybrid-dimension association rules.

Example 6.7 Suppose that *AllElectronics* has a sales multidimensional database with the following inter-related relations:

- *sales(customer_name, item_name, transaction_id)*,
- *lives(customer_name, region, city)*,
- *item(item_name, category, price)*, and
- *transaction(transaction_id, day, month, year)*,

where *lives*, *item*, and *transaction* are three dimension tables, linked to the fact table *sales* via three keys, *customer_name*, *item_name*, and *transaction_id*, respectively.

Our association mining query is to "*find the sales of what cheap items (where the sum of the prices is less than $100) that may promote the sales of what expensive items (where the minimum price is $500) in the same category for Vancouver customers in 1998*". This can be expressed in the DMQL data mining query language as follows, where each line of the query has been enumerated to aid in our discussion.

1) mine associations as
2) $\quad lives(C, \_, \text{“}Vancouver\text{”}) \wedge sales^{+}(C, ?\{I\}, \{S\}) \Rightarrow sales^{+}(C, ?\{J\}, \{T\})$
3) from sales
4) where S.year = 1998 and T.year = 1998 and I.category = J.category
5) group by C, I.category

6) having sum(I.price) < 100 and min(J.price) ≥ 500
7) with support threshold $= 0.01$
8) with confidence threshold $= 0.5$

Before we discuss the rule constraints, let us have a closer look at the above query. Line 1 is a knowledge type constraint, where association patterns are to be discovered. Line 2 specified a metarule. This is an abbreviated form for the following metarule for hybrid-dimension association rules (multidimensional association rules where the repeated predicate here is *sales*):

$$\begin{aligned} &lives(C, \_, \text{“}Vancouver\text{”}) \\ &\quad \wedge sales(C, ?I_1, S_1) \wedge \ldots \wedge sales(C, ?I_k, S_k) \wedge I = \{I_1, \ldots, I_k\} \wedge S = \{S_1, \ldots, S_k\} \\ &\quad \Rightarrow sales(C, ?J_1, T_1) \wedge \ldots \wedge sales(C, ?J_m, T_m) \wedge J = \{J_1, \ldots, J_m\} \wedge T = \{T_1, \ldots, T_m\} \end{aligned}$$

which means that one or more *sales* records in the form of "$sales(C, ?I_1, S_1) \wedge \ldots sales(C, ?I_k, S_k)$" will reside at the rule antecedent (left-hand side), and the question mark "?" means that only *item_name*, I_1, ..., I_k need be printed out. "$I = \{I_1, \ldots, I_k\}$" means that all the I's at the antecedent are taken from a set I, obtained from the SQL-like where-clause of line 4. Similar notational conventions are used at the consequent (right-hand side).

The metarule may allow the generation of association rules like the following.

$$\begin{aligned} &lives(C, \_, \text{“}Vancouver\text{”}) \wedge sales(C, \text{“}Census\_CD\text{”}, \_) \wedge \\ &\quad sales(C, \text{“}MS/Office97\text{”}, \_) \Rightarrow sales(C, \text{“}MS/SQLServer\text{”}, \_), \quad [1.5\%, 68\%] \end{aligned} \tag{6.32}$$

which means that if a customer in Vancouver bought "Census_CD" and "MS/Office97", it is likely (with a probability of 68%) that she will buy "MS/SQLServer", and 1.5% of all of the customers bought all three.

Data constraints are specified in the "$lives(\_, \_, \text{“}Vancouver\text{”})$" portion of the metarule (i.e., all the customers whose city is Vancouver), and in line 3, which specifies that only the fact table, *sales*, need be explicitly referenced. In such a multidimensional database, variable reference is simplified. For example, "*S.year = 1998*" is equivalent to the SQL statement "**from** *sales S, transaction R* **where** *S.transaction_id = R.transaction_id* **and** *R.year = 1998*".

All three dimensions (*lives, item*, and *transaction*) are used. Level constraints are as follows: for *lives*, we consider just *customer_name* since only *city = "Vancouver"* is used in the selection; for *item*, we consider the levels *item_name* and *category* since they are used in the query; and for *transaction*, we are only concerned with *transaction_id* since *day* and *month* are not referenced and *year* is used only in the selection.

Rule constraints include most portions of the **where** (line 4) and **having** (line 6) clauses, such as *"S.year = 1998"*, *"T.year = 1998"*, *"I.category = J.category"*, *"sum(I.price)* $\leq$ *100" and "min(J.price)* $\geq$ *500"*. Finally, lines 7 and 8 specify two interestingness constraints (i.e., thresholds), namely, a minimum support of 1% and a minimum confidence of 50%. □

Knowledge type and data constraints are applied before mining. The remaining constraint types could be used after mining, to filter out discovered rules. This, however, may make the mining process very inefficient and expensive. Dimension/level constraints were discussed in Section 6.3.2, and interestingness constraints have been discussed throughout this chapter. Let's focus now on rule constraints.

"What kind of constraints can be used during the mining process to prune the rule search space?", you ask. *"More specifically, what kind of rule constraints can be "pushed" deep into the mining process and still ensure the completeness of the answers to a mining query?"*

Consider the rule constraint "*sum(I.price)* $\leq$ *100*" of Example 6.7. Suppose we are using an Apriori-like (level-wise) framework, which for each iteration k, explores itemsets of size k. Any itemset whose price summation is not less than 100 can be pruned from the search space, since further addition of more items to this itemset will make it more expensive and thus will never satisfy the constraint. In other words, if an itemset does not satisfy this rule constraint, then none of its supersets can satisfy the constraint either. If a rule constraint obeys this property, it is called **anti-monotone**, or **downward closed**. Pruning by anti-monotone rule constraints can be applied at each iteration of Apriori-style algorithms to help improve the efficiency of the overall mining process, while guaranteeing completeness of the data mining query response.

Note that the Apriori property, which states that all non-empty subsets of a frequent itemset must also be frequent, is also anti-monotone. If a given itemset does not satisfy minimum support, then none of its supersets can

1-var Constraint	Anti-Monotone	Succinct
$S\theta v$, $\theta \in \{=, \leq, \geq\}$	yes	yes
$v \in S$	no	yes
$S \supseteq V$	no	yes
$S \subseteq V$	yes	yes
$S = V$	partly	yes
$min(S) \leq v$	no	yes
$min(S) \geq v$	yes	yes
$min(S) = v$	partly	yes
$max(S) \leq v$	yes	yes
$max(S) \geq v$	no	yes
$max(S) = v$	partly	yes
$count(S) \leq v$	yes	weakly
$count(S) \geq v$	no	weakly
$count(S) = v$	partly	weakly
$sum(S) \leq v$	yes	no
$sum(S) \geq v$	no	no
$sum(S) = v$	partly	no
$avg(S)\theta v$, $\theta \in \{=, \leq, \geq\}$	no	no
(frequency constraint)	(yes)	(no)

Table 6.3: Characterization of 1-variable constraints: anti-monotonicity and succinctness.

either. This property is used at each iteration of the Apriori algorithm to reduce the number of candidate itemsets examined, thereby reducing the search space for association rules.

Other examples of anti-monotone constraints include "*min(J.price)* $\geq$ *500*" and "*S.year = 1998*". Any itemset which violates either of these constraints can be discarded since adding more items to such itemsets can never satisfy the constraints. A constraint such as "*avg(I.price)* $\leq$ *100*" is not anti-monotone. For a given set that does not satisfy this constraint, a superset created by adding some (cheap) items may result in satisfying the constraint. Hence, pushing this constraint inside the mining process will not guarantee completeness of the data mining query response. A list of 1-variable constraints, characterized on the notion of anti-monotonicity, is given in the second column of Table 6.3.

"What other kinds of constraints can we use for pruning the search space?" Apriori-like algorithms deal with other constraints by first generating candidate sets and then testing them for constraint satisfaction, thereby following a *generate-and-test* paradigm. Instead, is there a kind of constraint for which we can somehow *enumerate all and only those sets that are guaranteed to satisfy the constraint*? This property of constraints is called **succintness**. If a rule constraint is succinct, then we can directly generate precisely those sets that satisfy it, even before support counting begins. This avoids the substantial overhead of the generate-and-test paradigm. In other words, such constraints are *pre-counting prunable*. Let's study an example of how succinct constraints can be used in mining association rules.

Example 6.8 Based on Table 6.3, the constraint "$min(J.price) \leq 500$" is succinct. This is because we can explicitly and precisely generate all the sets of items satisfying the constraint. Specifically, such a set must contain at least one item whose price is less than \$500. It is of the form: $S_1 \cup S_2$, where $S_1 \neq \emptyset$ is a subset of the set of all those items with prices less than \$500, and S_2, possibly empty, is a subset of the set of all those items with prices $>$ \$500. Because there is a precise "formula" to generate all the sets satisfying a succinct constraint, there is no need to iteratively check the rule constraint during the mining process.

What about the constraint "$min(J.price) \geq 500$", which occurs in Example 6.7? This is also succinct, since we can generate all sets of items satisfying the constraint. In this case, we simply do not include items whose price is less than \$500, since they cannot be in any set that would satisfy the given constraint. □

Note that a constraint such as "$avg(I.price) \leq 100$" could not be pushed into the mining process, since it is neither anti-monotone nor succinct according to Table 6.3.

Although optimizations associated with succinctness (or anti-monotonicity) cannot be applied to constraints like "$avg(I.price) \leq 100$", heuristic optimization strategies are applicable and can often lead to significant pruning.

6.7 Summary

- The discovery of association relationships among huge amounts of data is useful in selective marketing, decision analysis, and business management. A popular area of application is market basket analysis, which studies the buying habits of customers by searching for sets of items that are frequently purchased together (or in sequence). **Association rule mining** consists of first finding **frequent** itemsets (set of items, such as A and B, satisfying a *minimum support threshold*, or percentage of the task-relevant tuples), from which **strong** association rules in the form of $A \Rightarrow B$ are generated. These rules also satisfy a *minimum confidence threshold* (a prespecified probability of satisfying B under the condition that A is satisfied).

- Association rules can be classified into several categories based on different criteria, such as:

  1. Based on the *types of values* handled in the rule, associations can be classified into **Boolean** vs. **quantitative**.

     A *Boolean* association shows relationships between discrete (categorical) objects. A *quantitative* association is a multidimensional association that involves numeric attributes which are discretized dynamically. It may involve categorical attributes as well.

  2. Based on the *dimensions* of data involved in the rules, associations can be classified into **single-dimensional** vs. **multidimensional**.

     Single-dimensional association involves a single predicate or dimension, such as *buys*; whereas multidimensional association involves multiple (distinct) predicates or dimensions. Single-dimensional association shows **intra-attribute** relationships (i.e., associations within one attribute or dimension); whereas multidimensional association shows **inter-attribute** relationships (i.e., between or among attributes/dimensions).

  3. Based on the *levels of abstractions* involved in the rule, associations can be classified into **single-level** vs. **multilevel**.

     In a *single-level* association, the items or predicates mined are not considered at different levels of abstraction, whereas a *multilevel* association does consider multiple levels of abstraction.

- The **Apriori algorithm** is an efficient association rule mining algorithm which explores the level-wise mining property: *all the subsets of a frequent itemset must also be frequent.* At the k-th iteration (for $k > 1$), it forms frequent $(k+1)$-itemset candidates based on the frequent k-itemsets, and scans the database once to find the *complete* set of frequent $(k+1)$-itemsets, L_{k+1}.

  Variations involving hashing and data scan reduction can be used to make the procedure more efficient. Other variations include partitioning the data (mining on each partition and them combining the results), and sampling the data (mining on a subset of the data). These variations can reduce the number of data scans required to as little as two or one.

- **Multilevel association rules** can be mined using several strategies, based on how minimum support thresholds are defined at each level of abstraction. When using **reduced minimum support** at lower levels, pruning approaches include *level-cross-filtering by single item* and *level-cross filtering by k-itemset.* Redundant multilevel (descendent) association rules can be eliminated from presentation to the user if their support and confidence are close to their expected values, based on their corresponding ancestor rules.

- Techniques for mining **multidimensional association rules** can be categorized according to their treatment of quantitative attributes. First, quantitative attributes may be *discretized statically*, based on predefined concept hierarchies. Data cubes are well-suited to this approach, since both the data cube and quantitative attributes can make use of concept hierarchies. Second, **quantitative association rules** can be mined where quantitative attributes are discretized dynamically based on binning, where "adjacent" association rules may be combined by clustering. Third, **distance-based association rules** can be mined to capture the semantics of interval data, where intervals are defined by clustering.

- Not all strong association rules are interesting. **Correlation rules** can be mined for items that are statistically correlated.

- **Constraint-based mining** allow users to focus the search for rules by providing metarules, (i.e., pattern templates) and additional mining constraints. Such mining is facilitated with the use of a declarative data mining query language and user interface, and poses great challenges for mining query optimization. In particular, the rule constraint properties of **anti-monotonicity** and **succinctness** can be used during mining to guide the process, leading to more efficient and effective mining.

Exercises

1. The Apriori algorithm makes use of prior knowledge of subset support properties.

   (a) Prove that all non-empty subsets of a frequent itemset must also be frequent.

   (b) Prove that the support of any non-empty subset s' of itemset s must be as great as the support of s.

   (c) Given frequent itemset l and subset s of l, prove that the confidence of the rule "$s' \Rightarrow (l - s')$" cannot be more than the confidence of "$s \Rightarrow (l - s)$", where s' is a subset of s.

2. Section 6.2.2 describes a method for generating association rules from frequent itemsets. Propose a more efficient method. Explain why it is more efficient than the one proposed in Section 6.2.2. (Hint: Consider incorporating the properties of Question 1b and 1c into your design).

3. Suppose we have the following transactional data.

   (INSERT TRANSACTIONAL DATA HERE).

   Assume that the minimum support and minimum confidence thresholds are 3% and 60%, respectively.

   (a) Find the set of frequent itemsets using the Apriori algorithm. Show the derivation of C_k and L_k for each iteration, k.

   (b) Generate strong association rules from the frequent itemsets found above.

4. In Section 6.2.3, we studied two methods of scan reduction. Can you think of another approach, which trims transactions by *removing* items that do not contribute to frequent itemsets? Show the details of this approach in pseudo-code and with an example.

5. Suppose that a large store has a transaction database that is distributed among four locations. Transactions in each component database have the same format, namely $T_j : \{i_1, \ldots, i_m\}$, where T_j is a transaction identifier, and i_k $(1 \leq k \leq m)$ is the identifier of an item purchased in the transaction. Propose an efficient algorithm to mine global association rules (without considering multilevel associations). You may present your algorithm in the form of an outline. Your algorithm should not require shipping all of the data to one site and should not cause excessive network communication overhead.

6. Suppose that a data relation describing students at *Big-University* has been generalized to the following generalized relation R.

major	status	age	nationality	gpa	count
French	M.A	30_	Canada	2.8_3.2	3
cs	junior	15_20	Europe	3.2_3.6	29
physics	M.S	25_30	Latin_America	3.2_3.6	18
engineering	Ph.D	25_30	Asia	3.6_4.0	78
philosophy	Ph.D	25_30	Europe	3.2_3.6	5
French	senior	15_20	Canada	3.2_3.6	40
chemistry	junior	20_25	USA	3.6_4.0	25
cs	senior	15_20	Canada	3.2_3.6	70
philosophy	M.S	30_	Canada	3.6_4.0	15
French	junior	15_20	USA	2.8_3.2	8
philosophy	junior	25_30	Canada	2.8_3.2	9
philosophy	M.S	25_30	Asia	3.2_3.6	9
French	junior	15_20	Canada	3.2_3.6	52
math	senior	15_20	USA	3.6_4.0	32
cs	junior	15_20	Canada	3.2_3.6	76
philosophy	Ph.D	25_30	Canada	3.6_4.0	14
philosophy	senior	25_30	Canada	2.8_3.2	19
French	Ph.D	30_	Canada	2.8_3.2	1
engineering	junior	20_25	Europe	3.2_3.6	71
math	Ph.D	25_30	Latin_America	3.2_3.6	7
chemistry	junior	15_20	USA	3.6_4.0	46
engineering	junior	20_25	Canada	3.2_3.6	96
French	M.S	30_	Latin_America	3.2_3.6	4
philosophy	junior	20_25	USA	2.8_3.2	8
math	junior	15_20	Canada	3.6_4.0	59

Let the concept hierarchies be as follows.

$status$: $\{freshman, sophomore, junior, senior\} \in undergraduate$.
$\{M.Sc., M.A., Ph.D.\} \in graduate$.
$major$: $\{physics, chemistry, math\} \in science$.
$\{cs, engineering\} \in appl.\_sciences$.
$\{French, philosophy\} \in arts$.
age: $\{15\_20, 21\_25\} \in young$.
$\{26\_30, 30\_\} \in old$.
$nationality$: $\{Asia, Europe, U.S.A., Latin\_America\} \in foreign$.

Let the minimum support threshold be 2% and the minimum confidence threshold be 50% (at each of the levels).

(a) Draw the concept hierarchies for *status, major, age*, and *nationality*.

(b) Find the set of strong multilevel association rules in R using uniform support for all levels.

(c) Find the set of strong multilevel association rules in R using level-cross filtering by single items, where a reduced support of 1% is used for the lowest abstraction level.

7. Show that the support of an itemset H that contains both an item h and its ancestor $\hat{h}$ will be the same as the support for the itemset $H - \hat{h}$. Explain how this can be used in cross-level association rule mining.

8. Propose and outline a **level-shared mining** approach to mining multilevel association rules in which each item is encoded by its level position, and an initial scan of the database collects the count for each item *at each concept level*, identifying frequent and subfrequent items. Comment on the processing cost of mining multilevel associations with this method in comparison to mining single-level associations.

9. When mining cross-level association rules, suppose it is found that the itemset "{*IBM home computer, printer*}" does not satisfy minimum support. Can this information be used to prune the mining of a "descendent" itemset such as "{*IBM home computer, b/w printer*}"? Give a general rule explaining how this information may be used for pruning the search space.

10. Propose a method for mining hybrid-dimension association rules (multidimensional association rules with repeating predicates).

11. (INSERT QUESTIONS FOR mining quantitative association rules and distance-based association rules).

12. The following contingency table summarizes supermarket transaction data, where *hot dogs* refer to the transactions containing hot dogs, $\overline{hotdogs}$ refer to the transactions which do not contain hot dogs, *hamburgers* refer to the transactions containing hamburgers, and $\overline{hamburgers}$ refer to the transactions which do not contain hamburgers.

	hot dogs	$\overline{hotdogs}$	Σ_{row}
hamburgers	2,000	500	2,500
$\overline{hamburgers}$	1,000	1,500	2,500
Σ_{col}	3,000	2,000	5,000

    (a) Suppose that the association rule "*hot dogs* $\Rightarrow$ *hamburgers*" is mined. Given a minimum support threshold of 25% and a minimum confidence threshold of 50%, is this association rule strong?

    (b) Based on the given data, is the purchase of *hot dogs* independent of the purchase of *hamburgers*? If not, what kind of correlation relationship exists between the two?

13. Sequential patterns can be mined in methods similar to the mining of association rules. Design an efficient algorithm to mine **multilevel sequential patterns** from a transaction database. An example of such a pattern is the following: "*a customer who buys a PC will buy Microsoft software within three months*", on which one may drill-down to find a more refined version of the pattern, such as "*a customer who buys a Pentium Pro will buy Microsoft Office'97 within three months*".

14. Prove the characterization of the following 1-variable rule constraints with respect to anti-monotonicity and succinctness.

	1-var Constraint	Anti-Monotone	Succinct
a)	$v \in S$	no	yes
b)	$min(S) \leq v$	no	yes
c)	$min(S) \geq v$	yes	yes
d)	$max(S) \leq v$	yes	yes

Bibliographic Notes

Association rule mining was first proposed by Agrawal, Imielinski, and Swami [1]. The Apriori algorithm discussed in Section 6.2.1 was presented by Agrawal and Srikant [4], and a similar level-wise association mining algorithm was developed by Klemettinen et al. [20]. A method for generating association rules is described in Agrawal and Srikant [3]. References for the variations of Apriori described in Section 6.2.3 include the following. The use of hash tables to improve association mining efficiency was studied by Park, Chen, and Yu [29]. Scan and transaction reduction techniques are described in Agrawal and Srikant [4], Han and Fu [16], and Park, Chen and Yu [29]. The partitioning technique was proposed by Savasere, Omiecinski and Navathe [33]. The sampling approach is discussed in Toivonen [41]. A dynamic itemset counting approach is given in Brin et al. [9]. Calendric market basket analysis is discussed in Ramaswamy, Mahajan, and Silberschatz [32]. Mining of sequential patterns is described in Agrawal and Srikant [5], and Mannila, Toivonen, and Verkamo [24].

Multilevel association mining was studied in Han and Fu [16], and Srikant and Agrawal [38]. In Srikant and Agrawal [38], such mining is studied in the context of *generalized association rules*, and an R-interest measure is proposed for removing redundant rules.

Mining multidimensional association rules using static discretization of quantitative attributes and data cubes was studied by Kamber, Han, and Chiang [19]. Zhao, Deshpande, and Naughton [44] found that even when a cube is constructed on the fly, mining from data cubes can be faster than mining directly from a relational table. The ARCS system described in Section 6.4.3 for mining quantitative association rules based on rule clustering was proposed by Lent, Swami, and Widom [22]. Techniques for mining quantitative rules based on x-monotone and rectilinear regions

were presented by Fukuda et al. [15], and Yoda et al. [42]. A non-grid-based technique for mining quantitative association rules, which uses a measure of partial completeness, was proposed by Srikant and Agrawal [39]. The approach described in Section 6.4.4 for mining (distance-based) association rules over interval data was proposed by Miller and Yang [26].

The statistical independence of rules in data mining was studied by Piatetski-Shapiro [31]. The interestingness problem of strong association rules is discussed by Chen, Han, and Yu [10], and Brin, Motwani, and Silverstein [8]. An efficient method for generalizing associations to correlations is given in Brin, Motwani, and Silverstein [8], and briefly summarized in Section 6.5.2.

The use of metarules as syntactic or semantic filters defining the form of interesting single-dimensional association rules was proposed in Klemettinen et al. [20]. Metarule-guided mining, where the metarule consequent specifies an action (such as Bayesian clustering or plotting) to be applied to the data satisfying the metarule antecedent, was proposed in Shen et al. [35]. A relation-based approach to metarule-guided mining of association rules is studied in Fu and Han [14]. A data cube-based approach is studied in Kamber et al. [19]. The constraint-based association rule mining of Section 6.6.2 was studied in Ng et al. [27] and Lakshmanan et al. [21]. Other ideas involving the use of templates or predicate constraints in mining have been discussed in [6, 13, 18, 23, 36, 40].

An SQL-like operator for mining single-dimensional association rules was proposed by Meo, Psaila, and Ceri [25], and further extended in Baralis and Psaila [7]. The data mining query language, DMQL, was proposed in Han et al. [17].

An efficient incremental updating of mined association rules was proposed by Cheung et al. [12]. Parallel and distributed association data mining under the Apriori framework was studied by Park, Chen, and Yu [30], Agrawal and Shafer [2], and Cheung et al. [11]. Additional work in the mining of association rules includes mining sequential association patterns by Agrawal and Srikant [5], mining negative association rules by Savasere, Omiecinski and Navathe [34], and mining cyclic association rules by Ozden, Ramaswamy, and Silberschatz [28].

Bibliography

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proc. 1993 ACM-SIGMOD Int. Conf. Management of Data*, pages 207–216, Washington, D.C., May 1993.

[2] R. Agrawal and J. C. Shafer. Parallel mining of association rules: Design, implementation, and experience. *IEEE Trans. Knowledge and Data Engineering*, 8:962–969, 1996.

[3] R. Agrawal and R. Srikant. Fast algorithm for mining association rules in large databases. In *Research Report RJ 9839*, IBM Almaden Research Center, San Jose, CA, June 1994.

[4] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. 1994 Int. Conf. Very Large Data Bases*, pages 487–499, Santiago, Chile, September 1994.

[5] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proc. 1995 Int. Conf. Data Engineering*, pages 3–14, Taipei, Taiwan, March 1995.

[6] T. Anand and G. Kahn. Opportunity explorer: Navigating large databases using knowledge discovery templates. In *Proc. AAAI-93 Workshop Knowledge Discovery in Databases*, pages 45–51, Washington DC, July 1993.

[7] E. Baralis and G. Psaila. Designing templates for mining association rules. *Journal of Intelligent Information Systems*, 9:7–32, 1997.

[8] S. Brin, R. Motwani, and C. Silverstein. Beyond market basket: Generalizing association rules to correlations. In *Proc. 1997 ACM-SIGMOD Int. Conf. Management of Data*, pages 265–276, Tucson, Arizona, May 1997.

[9] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket analysis. In *Proc. 1997 ACM-SIGMOD Int. Conf. Management of Data*, pages 255–264, Tucson, Arizona, May 1997.

[10] M. S. Chen, J. Han, and P. S. Yu. Data mining: An overview from a database perspective. *IEEE Trans. Knowledge and Data Engineering*, 8:866–883, 1996.

[11] D.W. Cheung, J. Han, V. Ng, A. Fu, and Y. Fu. A fast distributed algorithm for mining association rules. In *Proc. 1996 Int. Conf. Parallel and Distributed Information Systems*, pages 31–44, Miami Beach, Florida, Dec. 1996.

[12] D.W. Cheung, J. Han, V. Ng, and C.Y. Wong. Maintenance of discovered association rules in large databases: An incremental updating technique. In *Proc. 1996 Int. Conf. Data Engineering*, pages 106–114, New Orleans, Louisiana, Feb. 1996.

[13] V. Dhar and A. Tuzhilin. Abstract-driven pattern discovery in databases. *IEEE Trans. Knowledge and Data Engineering*, 5:926–938, 1993.

[14] Y. Fu and J. Han. Meta-rule-guided mining of association rules in relational databases. In *Proc. 1st Int. Workshop Integration of Knowledge Discovery with Deductive and Object-Oriented Databases (KDOOD'95)*, pages 39–46, Singapore, Dec. 1995.

[15] T. Fukuda, Y. Morimoto, S. Morishita, and T. Tokuyama. Data mining using two-dimensional optimized association rules: Scheme, algorithms, and visualization. In *Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data*, pages 13–23, Montreal, Canada, June 1996.

[16] J. Han and Y. Fu. Discovery of multiple-level association rules from large databases. In *Proc. 1995 Int. Conf. Very Large Data Bases*, pages 420–431, Zurich, Switzerland, Sept. 1995.

[17] J. Han, Y. Fu, W. Wang, K. Koperski, and O. R. Zaïane. DMQL: A data mining query language for relational databases. In *Proc. 1996 SIGMOD'96 Workshop Research Issues on Data Mining and Knowledge Discovery (DMKD'96)*, pages 27–34, Montreal, Canada, June 1996.

[18] P. Hoschka and W. Klösgen. A support system for interpreting statistical data. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 325–346. AAAI/MIT Press, 1991.

[19] M. Kamber, J. Han, and J. Y. Chiang. Metarule-guided mining of multi-dimensional association rules using data cubes. In *Proc. 3rd Int. Conf. Knowledge Discovery and Data Mining (KDD'97)*, pages 207–210, Newport Beach, California, August 1997.

[20] M. Klemettinen, H. Mannila, P. Ronkainen, H. Toivonen, and A.I. Verkamo. Finding interesting rules from large sets of discovered association rules. In *Proc. 3rd Int. Conf. Information and Knowledge Management*, pages 401–408, Gaithersburg, Maryland, Nov. 1994.

[21] L. V. S. Lakshmanan, R. Ng, J. Han, and A. Pang. Optimization of constrained frequent set queries with 2-variable constraints. In *Proc. 1999 ACM-SIGMOD Int. Conf. Management of Data*, pages 157–168, Philadelphia, PA, June 1999.

[22] B. Lent, A. Swami, and J. Widom. Clustering association rules. In *Proc. 1997 Int. Conf. Data Engineering (ICDE'97)*, pages 220–231, Birmingham, England, April 1997.

[23] B. Liu, W. Hsu, and S. Chen. Using general impressions to analyze discovered classification rules. In *Proc. 3rd Int.. Conf. on Knowledge Discovery and Data Mining (KDD'97)*, pages 31–36, Newport Beach, CA, August 1997.

[24] H. Mannila, H Toivonen, and A. I. Verkamo. Discovering frequent episodes in sequences. In *Proc. 1st Int. Conf. Knowledge Discovery and Data Mining*, pages 210–215, Montreal, Canada, Aug. 1995.

[25] R. Meo, G. Psaila, and S. Ceri. A new SQL-like operator for mining association rules. In *Proc. 1996 Int. Conf. Very Large Data Bases*, pages 122–133, Bombay, India, Sept. 1996.

[26] R.J. Miller and Y. Yang. Association rules over interval data. In *Proc. 1997 ACM-SIGMOD Int. Conf. Management of Data*, pages 452–461, Tucson, Arizona, May 1997.

[27] R. Ng, L. V. S. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained associations rules. In *Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data*, pages 13–24, Seattle, Washington, June 1998.

[28] B. Özden, S. Ramaswamy, and A. Silberschatz. Cyclic association rules. In *Proc. 1998 Int. Conf. Data Engineering (ICDE'98)*, pages 412–421, Orlando, FL, Feb. 1998.

[29] J.S. Park, M.S. Chen, and P.S. Yu. An effective hash-based algorithm for mining association rules. In *Proc. 1995 ACM-SIGMOD Int. Conf. Management of Data*, pages 175–186, San Jose, CA, May 1995.

[30] J.S. Park, M.S. Chen, and P.S. Yu. Efficient parallel mining for association rules. In *Proc. 4th Int. Conf. Information and Knowledge Management*, pages 31–36, Baltimore, Maryland, Nov. 1995.

[31] G. Piatetsky-Shapiro. Discovery, analysis, and presentation of strong rules. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 229–238. AAAI/MIT Press, 1991.

[32] S. Ramaswamy, S. Mahajan, and A. Silberschatz. On the discovery of interesting patterns in association rules. In *Proc. 1998 Int. Conf. Very Large Data Bases*, pages 368–379, New York, NY, August 1998.

[33] A. Savasere, E. Omiecinski, and S. Navathe. An efficient algorithm for mining association rules in large databases. In *Proc. 1995 Int. Conf. Very Large Data Bases*, pages 432–443, Zurich, Switzerland, Sept. 1995.

[34] A. Savasere, E. Omiecinski, and S. Navathe. Mining for strong negative associations in a large database of customer transactions. In *Proc. 1998 Int. Conf. Data Engineering (ICDE'98)*, pages 494–502, Orlando, FL, Feb. 1998.

[35] W. Shen, K. Ong, B. Mitbander, and C. Zaniolo. Metaqueries for data mining. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 375–398. AAAI/MIT Press, 1996.

[36] A. Silberschatz and A. Tuzhilin. What makes patterns interesting in knowledge discovery systems. *IEEE Trans. on Knowledge and Data Engineering*, 8:970–974, Dec. 1996.

[37] E. Simoudis, J. Han, and U. Fayyad (eds.). *Proc. 2nd Int. Conf. Knowledge Discovery and Data Mining (KDD'96)*. AAAI Press, August 1996.

[38] R. Srikant and R. Agrawal. Mining generalized association rules. In *Proc. 1995 Int. Conf. Very Large Data Bases*, pages 407–419, Zurich, Switzerland, Sept. 1995.

[39] R. Srikant and R. Agrawal. Mining quantitative association rules in large relational tables. In *Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data*, pages 1–12, Montreal, Canada, June 1996.

[40] R. Srikant, Q. Vu, and R. Agrawal. Mining association rules with item constraints. In *Proc. 3rd Int. Conf. Knowledge Discovery and Data Mining (KDD'97)*, pages 67–73, Newport Beach, California, August 1997.

[41] H. Toivonen. Sampling large databases for association rules. In *Proc. 1996 Int. Conf. Very Large Data Bases*, pages 134–145, Bombay, India, Sept. 1996.

[42] K. Yoda, T. Fukuda, Y. Morimoto, S. Morishita, and T. Tokuyama. Computing optimized rectilinear regions for association rules. In *Proc. 3rd Int. Conf. Knowledge Discovery and Data Mining (KDD'97)*, pages 96–103, Newport Beach, California, August 1997.

[43] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: an efficient data clustering method for very large databases. In *Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data*, pages 103–114, Montreal, Canada, June 1996.

[44] Y. Zhao, P. M. Deshpande, and J. F. Naughton. An array-based algorithm for simultaneous multidimensional aggregates. In *Proc. 1997 ACM-SIGMOD Int. Conf. Management of Data*, pages 159–170, Tucson, Arizona, May 1997.

Contents

Chapter 7

Classification and Prediction

Databases are rich with hidden information that can be used for making intelligent business decisions. Classification and prediction are two forms of data analysis which can be used to extract models describing important data classes or to predict future data trends. Whereas *classification* predicts categorical labels (or discrete values), *prediction* models continuous-valued functions. For example, a classification model may be built to categorize bank loan applications as either safe or risky, while a prediction model may be built to predict the expenditures of potential customers on computer equipment given their income and occupation. Many classification and prediction methods have been proposed by researchers in machine learning, expert systems, statistics, and neurobiology. Most algorithms are memory resident, typically assuming a small data size. Recent database mining research has built on such work, developing scalable classification and prediction techniques capable of handling large, disk resident data. These techniques often consider parallel and distributed processing.

In this chapter, you will learn basic techniques for data classification such as decision tree induction, Bayesian classification and Bayesian belief networks, and neural networks. The integration of data warehousing technology with classification is also discussed, as well as association-based classification. Other approaches to classification, such as k-nearest neighbor classifiers, case-based reasoning, genetic algorithms, rough sets, and fuzzy logic techniques are introduced. Methods for prediction, including linear, nonlinear, and generalized linear regression models are briefly discussed. Where applicable, you will learn of modifications, extensions and optimizations to these techniques for their application to data classification and prediction for large databases.

7.1 What is classification? What is prediction?

Data classification is a two step process (Figure 7.1). In the first step, a model is built describing a predetermined set of data classes or concepts. The model is constructed by analyzing database tuples described by attributes. Each tuple is assumed to belong to a predefined class, as determined by one of the attributes, called the **class label attribute**. In the context of classification, data tuples are also referred to as *samples, examples*, or *objects*. The data tuples analyzed to build the model collectively form the **training data set**. The individual tuples making up the training set are referred to as **training samples** and are randomly selected from the sample population. Since the class label of each training sample *is provided*, this step is also known as **supervised learning** (i.e., the learning of the model is 'supervised' in that it is told to which class each training sample belongs). It contrasts with **unsupervised learning** (or **clustering**), in which the class labels of the training samples are not known, and the number or set of classes to be learned may not be known in advance. Clustering is the topic of Chapter 8.

Typically, the learned model is represented in the form of classification rules, decision trees, or mathematical formulae. For example, given a database of customer credit information, classification rules can be learned to identify customers as having either excellent or fair credit ratings (Figure 7.1a). The rules can be used to categorize future data samples, as well as provide a better understanding of the database contents.

In the second step (Figure 7.1b), the model is used for classification. First, the predictive accuracy of the model (or classifier) is estimated. Section 7.9 of this chapter describes several methods for estimating classifier accuracy. The **holdout method** is a simple technique which uses a **test set** of class-labeled samples. These samples are

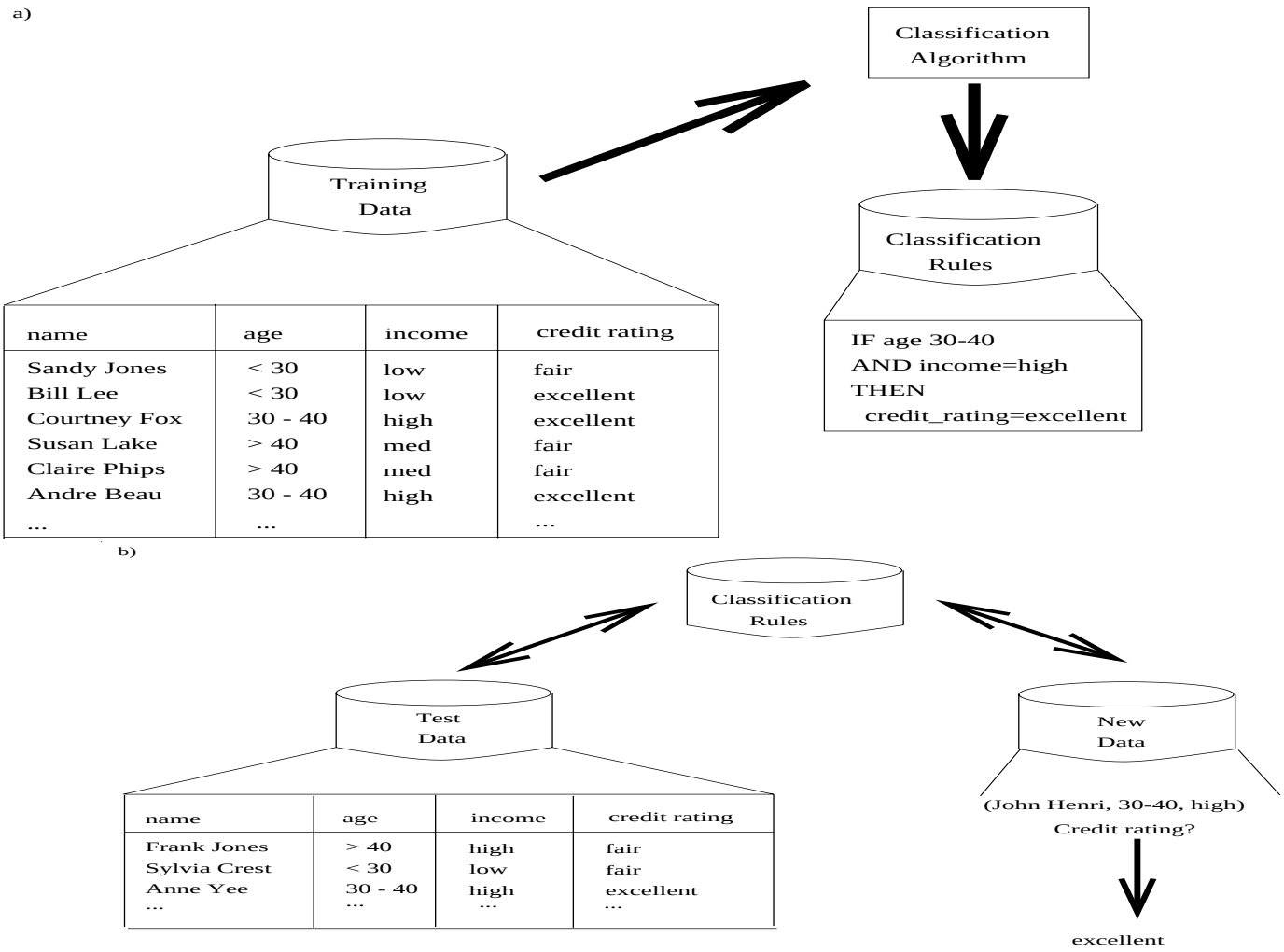


Figure 7.1: The data classification process: a) *Learning:* Training data are analyzed by a classification algorithm. Here, the class label attribute is *credit_rating*, and the learned model or classifier is represented in the form of classification rules. b) *Classification:* Test data are used to estimate the accuracy of the classification rules. If the accuracy is considered acceptable, the rules can be applied to the classification of new data tuples.

randomly selected and are independent of the training samples. The **accuracy** of a model on a given test set is the percentage of test set samples that are correctly classified by the model. For each test sample, the known class label is compared with the learned model's class prediction for that sample. Note that if the accuracy of the model were estimated based on the training data set, this estimate could be optimistic since the learned model tends to **overfit** the data (that is, it may have incorporated some particular anomalies of the training data which are not present in the overall sample population). Therefore, a test set is used.

If the accuracy of the model is considered acceptable, the model can be used to classify future data tuples or objects for which the class label is not known. (Such data are also referred to in the machine learning literature as "unknown" or "previously unseen" data). For example, the classification rules learned in Figure 7.1a from the analysis of data from existing customers can be used to predict the credit rating of new or future (i.e., previously unseen) customers.

"How is prediction different from classification?" **Prediction** can be viewed as the construction and use of a model to assess the class of an unlabeled object, or to assess the value or value ranges of an attribute that a given object is likely to have. In this view, classification and regression are the two major types of prediction problems where classification is used to predict discrete or nominal values, while regression is used to predict continuous or

ordered values. In our view, however, we refer to the use of predication to predict class labels as *classification* and the use of predication to predict continuous values (e.g., using regression techniques) as *prediction*. This view is commonly accepted in data mining.

Classification and prediction have numerous applications including credit approval, medical diagnosis, performance prediction, and selective marketing.

Example 7.1 Suppose that we have a database of customers on the *AllElectronics* mailing list. The mailing list is used to send out promotional literature describing new products and upcoming price discounts. The database describes attributes of the customers, such as their name, age, income, occupation, and credit rating. The customers can be classified as to whether or not they have purchased a computer at *AllElectronics*. Suppose that new customers are added to the database and that you would like to notify these customers of an uncoming computer sale. To send out promotional literature to every new customer in the database can be quite costly. A more cost efficient method would be to only target those new customers who are likely to purchase a new computer. A classification model can be constructed and used for this purpose.

Suppose instead that you would like to predict the number of major purchases that a customer will make at *AllElectronics* during a fiscal year. Since the predicted value here is ordered, a prediction model can be constructed for this purpose. □

7.2 Issues regarding classification and prediction

Preparing the data for classification and prediction. The following preprocessing steps may be applied to the data in order to help improve the accuracy, efficiency, and scalability of the classification or prediction process.

- **Data cleaning.** This refers to the preprocessing of data in order to remove or reduce *noise* (by applying smoothing techniques, for example), and the treatment of *missing values* (e.g., by replacing a missing value with the most commonly occurring value for that attribute, or with the most probable value based on statistics). Although most classification algorithms have some mechanisms for handling noisy or missing data, this step can help reduce confusion during learning.

- **Relevance analysis.** Many of the attributes in the data may be *irrelevant* to the classification or prediction task. For example, data recording the day of the week on which a bank loan application was filed is unlikely to be relevant to the success of the application. Furthermore, other attributes may be *redundant*. Hence, relevance analysis may be performed on the data with the aim of removing any irrelevant or redundant attributes from the learning process. In machine learning, this step is known as *feature selection*. Including such attributes may otherwise slow down, and possibly mislead, the learning step.

  Ideally, the time spent on relevance analysis, when added to the time spent on learning from the resulting "reduced" feature subset, should be less than the time that would have been spent on learning from the original set of features. Hence, such analysis can help improve classification efficiency and scalability.

- **Data transformation.** The data can be *generalized* to higher-level concepts. Concept hierarchies may be used for this purpose. This is particularly useful for continuous-valued attributes. For example, numeric values for the attribute *income* may be generalized to discrete ranges such as *low, medium*, and *high*. Similarly, nominal-valued attributes, like *street*, can be generalized to higher-level concepts, like *city*. Since generalization compresses the original training data, fewer input/output operations may be involved during learning.

  The data may also be normalized, particularly when neural networks or methods involving distance measurements are used in the learning step. **Normalization** involves scaling all values for a given attribute so that they fall within a small specified range, such as -1.0 to 1.0, or 0 to 1.0. In methods which use distance measurements, for example, this would prevent attributes with initially large ranges (like, say *income*) from outweighing attributes with initially smaller ranges (such as binary attributes).

Data cleaning, relevance analysis, and data transformation are described in greater detail in Chapter 3 of this book.

Comparing classification methods. Classification and prediction methods can be compared and evaluated according to the following criteria:

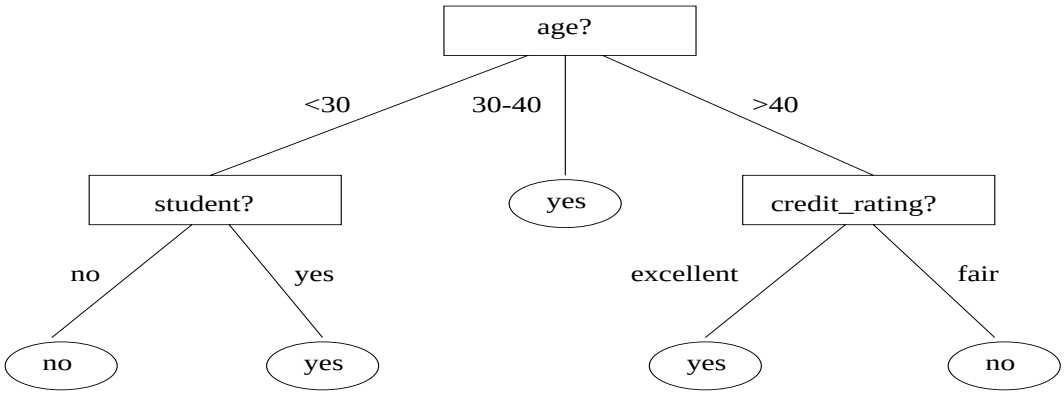


Figure 7.2: A decision tree for the concept *buys_computer*, indicating whether or not a customer at *AllElectronics* is likely to purchase a computer. Each internal (non-leaf) node represents a test on an attribute. Each leaf node represents a class (either *buys_computer = yes* or *buys_computer = no*).

1. **Predictive accuracy.** This refers to the ability of the model to correctly predict the class label of new or previously unseen data.
2. **Speed.** This refers to the computation costs involved in generating and using the model.
3. **Robustness.** This is the ability of the model to make correct predictions given noisy data or data with missing values.
4. **Scalability.** This refers to the ability of the learned model to perform efficiently on large amounts of data.
5. **Interpretability.** This refers is the level of understanding and insight that is provided by the learned model.

These issues are discussed throughout the chapter. The database research community's contributions to classification and prediction for data mining have strongly emphasized the scalability aspect, particularly with respect to decision tree induction.

7.3 Classification by decision tree induction

"What is a decision tree?"

A **decision tree** is a flow-chart-like tree structure, where each *internal node* denotes a test on an attribute, each *branch* represents an outcome of the test, and *leaf nodes* represent classes or class distributions. The topmost node in a tree is the *root* node. A typical decision tree is shown in Figure 7.2. It represents the concept *buys_computer*, that is, it predicts whether or not a customer at *AllElectronics* is likely to purchase a computer. Internal nodes are denoted by rectangles, and leaf nodes are denoted by ovals.

In order to classify an unknown sample, the attribute values of the sample are tested against the decision tree. A path is traced from the root to a leaf node which holds the class prediction for that sample. Decision trees can easily be converted to classification rules.

In Section 7.3.1, we describe a basic algorithm for learning decision trees. When decision trees are built, many of the branches may reflect noise or outliers in the training data. *Tree pruning* attempts to identify and remove such branches, with the goal of improving classification accuracy on unseen data. Tree pruning is described in Section 7.3.2. The extraction of classification rules from decision trees is discussed in Section 7.3.3. Enhancements of the basic decision tree algorithm are given in Section 7.3.4. Scalability issues for the induction of decision trees from large databases are discussed in Section 7.3.5. Section 7.3.6 describes the integration of decision tree induction with data warehousing facilities, such as data cubes, allowing the mining of decision trees at multiple levels of granularity. Decision trees have been used in many application areas ranging from medicine to game theory and business. Decision trees are the basis of several commercial rule induction systems.

Algorithm 7.3.1 (Generate_decision_tree) Generate a decision tree from the given training data.

Input: The training samples, *samples*, represented by discrete-valued attributes; the set of candidate attributes, *attribute-list*.

Output: A decision tree.

Method:

1) create a node N;
2) **if** *samples* are all of the same class, C **then**
3) return N as a leaf node labeled with the class C;
4) **if** *attribute-list* is empty **then**
5) return N as a leaf node labeled with the most common class in *samples*; // majority voting
6) select *test-attribute*, the attribute among *attribute-list* with the highest information gain;
7) label node N with *test-attribute*;
8) **for each** known value a_i **of** *test-attribute* // partition the samples
9) grow a branch from node N for the condition *test-attribute*=a_i;
10) let s_i be the set of samples in *samples* for which *test-attribute*=a_i; // a partition
11) **if** s_i is empty **then**
12) attach a leaf labeled with the most common class in *samples*;
13) **else** attach the node returned by Generate_decision_tree(s_i, *attribute-list* - *test-attribute*);

□

Figure 7.3: Basic algorithm for inducing a decision tree from training samples.

7.3.1 Decision tree induction

The basic algorithm for decision tree induction is a greedy algorithm which constructs decision trees in a top-down recursive divide-and-conquer manner. The algorithm, summarized in Figure 7.3, is a version of ID3, a well-known decision tree induction algorithm. Extensions to the algorithm are discussed in Sections 7.3.2 to 7.3.6.

The basic strategy is as follows:

- The tree starts as a single node representing the training samples (step 1).
- If the samples are all of the same class, then the node becomes a leaf and is labeled with that class (steps 2 and 3).
- Otherwise, the algorithm uses an entropy-based measure known as *information gain* as a heuristic for selecting the attribute that will best separate the samples into individual classes (step 6). This attribute becomes the "test" or "decision" attribute at the node (step 7). In this version of the algorithm, all attributes are categorical, i.e., discrete-valued. Continuous-valued attributes must be discretized.
- A branch is created for each known value of the test attribute, and the samples are partitioned accordingly (steps 8-10).
- The algorithm uses the same process recursively to form a decision tree for the samples at each partition. Once an attribute has occurred at a node, it need not be considered in any of the node's descendents (step 13).
- The recursive partitioning stops only when any one of the following conditions is true:
  1. All samples for a given node belong to the same class (step 2 and 3), or
  2. There are no remaining attributes on which the samples may be further partitioned (step 4). In this case, **majority voting** is employed (step 5). This involves converting the given node into a leaf and labeling it with the class in majority among *samples*. Alternatively, the class distribution of the node samples may be stored; or
  3. There are no samples for the branch *test-attribute*=a_i (step 11). In this case, a leaf is created with the majority class in *samples* (step 12).

Attribute selection measure. The **information gain** measure is used to select the test attribute at each node in the tree. Such a measure is referred to as an *attribute selection measure* or a *measure of the goodness of split*. The attribute with the highest information gain (or greatest *entropy* reduction) is chosen as the test attribute for the current node. This attribute minimizes the information needed to classify the samples in the resulting partitions and reflects the least randomness or "impurity" in these partitions. Such an information-theoretic approach minimizes the expected number of tests needed to classify an object and guarantees that a simple (but not necessarily the simplest) tree is found.

Let S be a set consisting of s data samples. Suppose the class label attribute has m distinct values defining m distinct classes, C_i (for $i = 1, \ldots, m$). Let s_i be the number of samples of S in class C_i. The expected information needed to classify a given sample is given by:

$$I(s_1, s_2, \ldots, s_m) \quad = \quad -\sum_{i=1}^{m} p_i log_2(p_i) \tag{7.1}$$

where p_i is the probability than an arbitrary sample belongs to class C_i and is estimated by s_i/s. Note that a *log* function to the base 2 is used since the information is encoded in bits.

Let attribute A have v distinct values, $\{a_1, a_2, \cdots, a_v\}$. Attribute A can be used to partition S into v subsets, $\{S_1, S_2, \cdots, S_v\}$, where S_j contains those samples in S that have value a_j of A. If A were selected as the test attribute (i.e., best attribute for splitting), then these subsets would correspond to the branches grown from the node containing the set S. Let s_{ij} be the number of samples of class C_i in a subset S_j. The **entropy**, or expected information based on the partitioning into subsets by A is given by:

$$E(A) = \sum_{j=1}^{v} \frac{s_{1j} + \cdots + s_{mj}}{s} I(s_{1j}, \ldots, s_{mj}). \tag{7.2}$$

The term $\sum_{j=1}^{v} \frac{s_{1j}+\cdots+s_{mj}}{s}$ acts as the weight of the j^{th} subset and is the number of samples in the subset (i.e., having value a_j of A) divided by the total number of samples in S. The smaller the entropy value is, the greater the purity of the subset partitions. The encoding information that would be gained by branching on A is

$$Gain(A) \quad = \quad I(s_1, s_2, \ldots, s_m) - E(A). \tag{7.3}$$

In other words, *Gain(A)* is the expected reduction in entropy caused by knowing the value of attribute A.

The algorithm computes the information gain of each attribute. The attribute with the highest information gain is chosen as the test attribute for the given set S. A node is created and labeled with the attribute, branches are created for each value of the attribute, and the samples are partitioned accordingly.

Example 7.2 Induction of a decision tree. Table 7.1 presents a training set of data tuples taken from the *AllElectronics* customer database. (The data are adapted from [Quinlan 1986b]). The class label attribute, *buys_computer*, has two distinct values (namely {*yes, no*}), therefore, there are two distinct classes ($m = 2$). Let C_1 correspond to the class *yes* and class C_2 correspond to *no*. There are 9 samples of class *yes* and 5 samples of class *no*. To compute the information gain of each attribute, we first use Equation (7.1) to compute the expected information needed to classify a given sample. This is:

$$I(s_1, s_2) = I(9, 5) = -\frac{9}{14}\log_2\frac{9}{14} - \frac{5}{14}\log_2\frac{5}{14} = 0.940$$

Next, we need to compute the entropy of each attribute. Let's start with the attribute *age*. We need to look at the distribution of *yes* and *no* samples for each value of *age*. We compute the expected information for each of these distributions.

for *age* = *"<30"*:	$s_{11} = 2$	$s_{21} = 3$	$I(s_{11}, s_{21}) = 0.971$
for *age* = *"30-40"*:	$s_{12} = 4$	$s_{22} = 0$	$I(s_{12}, s_{22}) = 0$
for *age* = *">40"*:	$s_{13} = 3$	$s_{23} = 2$	$I(s_{13}, s_{23}) = 0.971$

rid	age	income	student	credit_rating	Class: buys_computer
1	<30	high	no	fair	no
2	<30	high	no	excellent	no
3	30-40	high	no	fair	yes
4	>40	medium	no	fair	yes
5	>40	low	yes	fair	yes
6	>40	low	yes	excellent	no
7	30-40	low	yes	excellent	yes
8	<30	medium	no	fair	no
9	<30	low	yes	fair	yes
10	>40	medium	yes	fair	yes
11	<30	medium	yes	excellent	yes
12	30-40	medium	no	excellent	yes
13	30-40	high	yes	fair	yes
14	>40	medium	no	excellent	no

Table 7.1: Training data tuples from the *AllElectronics* customer database.

Using Equation (7.2), the expected information needed to classify a given sample if the samples are partitioned according to *age*, is:

$$E(age) = \frac{5}{14}I(s_{11}, s_{21}) + \frac{4}{14}I(s_{12}, s_{22}) + \frac{5}{14}I(s_{13}, s_{23}) = 0.694.$$

Hence, the gain in information from such a partitioning would be:

$$Gain(age) = I(s_1, s_2) - E(age) = 0.246$$

Similarly, we can compute $Gain(income) = 0.029$, $Gain(student) = 0.151$, and $Gain(credit\_rating) = 0.048$. Since *age* has the highest information gain among the attributes, it is selected as the test attribute. A node is created and labeled with *age*, and branches are grown for each of the attribute's values. The samples are then partitioned accordingly, as shown in Figure 7.4. Notice that the samples falling into the partition for *age = 30-40* all belong to the same class. Since they all belong to class *yes*, a leaf should therefore be created at the end of this branch and labeled with *yes*. The final decision tree returned by the algorithm is shown in Figure 7.2. □

In summary, decision tree induction algorithms have been used for classification in a wide range of application domains. Such systems do not use domain knowledge. The learning and classification steps of decision tree induction are generally fast. Classification accuracy is typically high for data where the mapping of classes consists of long and thin regions in concept space.

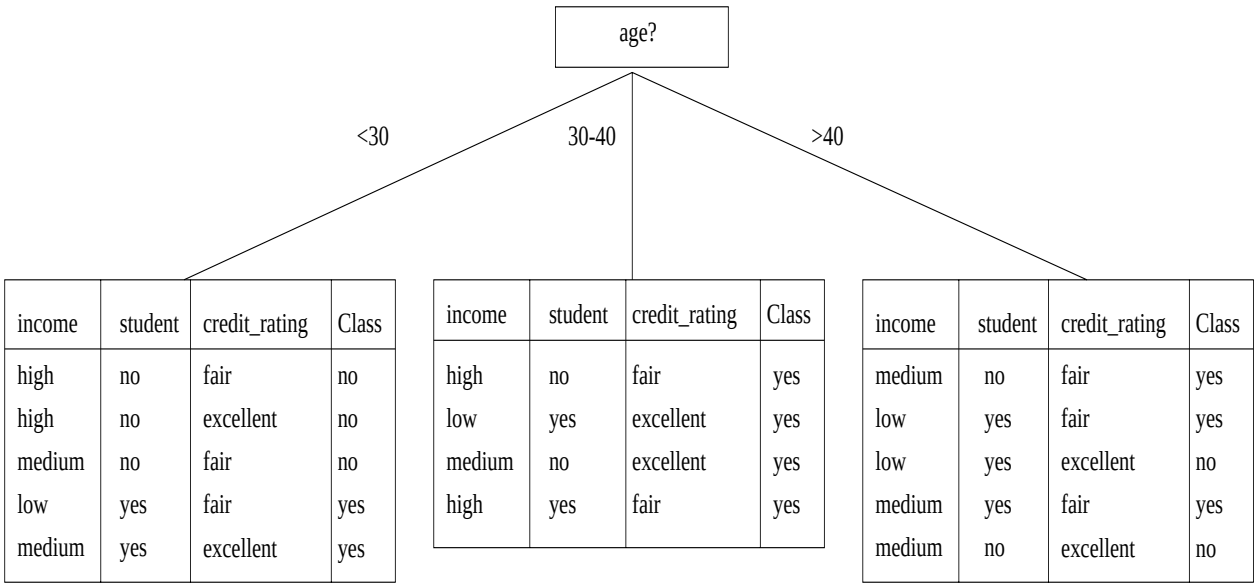
7.3.2 Tree pruning

When a decision tree is built, many of the branches will reflect anomalies in the training data due to noise or outliers. Tree pruning methods address this problem of *overfitting* the data. Such methods typically use statistical measures to remove the least reliable branches, generally resulting in faster classification and an improvement in the ability of the tree to correctly classify independent test data.

"How does tree pruning work?" There are two common approaches to tree pruning.

- In the **prepruning** approach, a tree is "pruned" by halting its construction early (e.g., by deciding not to further split or partition the subset of training samples at a given node). Upon halting, the node becomes a leaf. The leaf may hold the most frequent class among the subset samples, or the probability distribution of those samples.

  When constructing a tree, measures such as statistical significance, χ^2, information gain, etc., can be used to assess the goodness of a split. If partitioning the samples at a node would result in a split that falls below a prespecified threshold, then further partitioning of the given subset is halted. There are difficulties, however,



income	student	credit_rating	Class
high	no	fair	no
high	no	excellent	no
medium	no	fair	no
low	yes	fair	yes
medium	yes	excellent	yes

income	student	credit_rating	Class
high	no	fair	yes
low	yes	excellent	yes
medium	no	excellent	yes
high	yes	fair	yes

income	student	credit_rating	Class
medium	no	fair	yes
low	yes	fair	yes
low	yes	excellent	no
medium	yes	fair	yes
medium	no	excellent	no

Figure 7.4: The attribute *age* has the highest information gain and therefore becomes a test attribute at the root node of the decision tree. Branches are grown for each value of *age*. The samples are shown partitioned according to each branch.

in choosing an appropriate threshold. High thresholds could result in oversimplified trees, while low thresholds could result in very little simplification.

- The **postpruning** approach removes branches from a "fully grown" tree. A tree node is pruned by removing its branches.

  The *cost complexity* pruning algorithm is an example of the postpruning approach. The pruned node becomes a leaf and is labeled by the most frequent class among its former branches. For each non-leaf node in the tree, the algorithm calculates the expected error rate that would occur if the subtree at that node were pruned. Next, the expected error rate occurring if the node were not pruned is calculated using the error rates for each branch, combined by weighting according to the proportion of observations along each branch. If pruning the node leads to a greater expected error rate, then the subtree is kept. Otherwise, it is pruned. After generating a set of progressively pruned trees, an independent test set is used to estimate the accuracy of each tree. The decision tree that minimizes the expected error rate is preferred.

  Rather than pruning trees based on expected error rates, we can prune trees based on the number of bits required to encode them. The "best pruned tree" is the one that minimizes the number of encoding bits. This method adopts the Minimum Description Length (MDL) principle which follows the notion that the simplest solution is preferred. Unlike cost complexity pruning, it does not require an independent set of samples.

Alternatively, prepruning and postpruning may be interleaved for a combined approach. Postpruning requires more computation than prepruning, yet generally leads to a more reliable tree.

7.3.3 Extracting classification rules from decision trees

"Can I get classification rules out of my decision tree? If so, how?"

The knowledge represented in decision trees can be extracted and represented in the form of classification IF-THEN rules. One rule is created for each path from the root to a leaf node. Each attribute-value pair along a given path forms a conjunction in the rule antecedent ("IF" part). The leaf node holds the class prediction, forming the rule consequent ("THEN" part). The IF-THEN rules may be easier for humans to understand, particularly if the given tree is very large.

Example 7.3 Generating classification rules from a decision tree. The decision tree of Figure 7.2 can be converted to classification IF-THEN rules by tracing the path from the root node to each leaf node in the tree. The rules extracted from Figure 7.2 are:

IF *age* = "<30" AND *student* = *no* THEN *buys_computer* = *no*
IF *age* = "<30" AND *student* = *yes* THEN *buys_computer* = *yes*
IF *age* = "30-40" THEN *buys_computer* = *yes*
IF *age* = ">40" AND *credit_rating* = *excellent* THEN *buys_computer* = *yes*
IF *age* = ">40" AND *credit_rating* = *fair* THEN *buys_computer* = *no*

□

C4.5, a later version of the ID3 algorithm, uses the training samples to estimate the accuracy of each rule. Since this would result in an optimistic estimate of rule accuracy, C4.5 employs a pessimistic estimate to compensate for the bias. Alternatively, a set of test samples independent from the training set can be used to estimate rule accuracy.

A rule can be "pruned" by removing any condition in its antecedent that does not improve the estimated accuracy of the rule. For each class, rules within a class may then be ranked according to their estimated accuracy. Since it is possible that a given test sample will not satisfy any rule antecedent, a default rule assigning the majority class is typically added to the resulting rule set.

7.3.4 Enhancements to basic decision tree induction

"What are some enhancements to basic decision tree induction?"

Many enhancements to the basic decision tree induction algorithm of Section 7.3.1 have been proposed. In this section, we discuss several major enhancements, many of which are incorporated into C4.5, a successor algorithm to ID3.

The basic decision tree induction algorithm of Section 7.3.1 requires all attributes to be categorical or discretized. The algorithm can be modified to allow for continuous-valued attributes. A test on a continuous-valued attribute A results in two branches, corresponding to the conditions $A \leq V$ and $A > V$ for some numeric value, V, of A. Given v values of A, then $v-1$ possible splits are considered in determining V. Typically, the midpoints between each pair of adjacent values are considered. If the values are sorted in advance, then this requires only one pass through the values.

The basic algorithm for decision tree induction creates one branch for each value of a test attribute, and then distributes the samples accordingly. This partitioning can result in numerous small subsets. As the subsets become smaller and smaller, the partitioning process may end up using sample sizes that are statistically insufficient. The detection of useful patterns in the subsets may become impossible due to insufficiency of the data. One alternative is to allow for the grouping of categorical attribute values. A tree node may test whether the value of an attribute belongs to a given set of values, such as $A_i \in \{a_1, a_2, \ldots, a_n\}$. Another alternative is to create binary decision trees, where each branch holds a boolean test on an attribute. Binary trees result in less fragmentation of the data. Some empirical studies have found that binary decision trees tend to be more accurate that traditional decision trees.

The information gain measure is biased in that it tends to prefer attributes with many values. Many alternatives have been proposed, such as gain ratio, which considers the probability of each attribute value. Various other selection measures exist, including the gini index, the χ^2 contingency table statistic, and the G-statistic.

Many methods have been proposed for handling missing attribute values. A missing or unknown value for an attribute A may be replaced by the most common value for A, for example. Alternatively, the apparent information gain of attribute A can be reduced by the proportion of samples with unknown values of A. In this way, "fractions" of a sample having a missing value can be partitioned into more than one branch at a test node. Other methods may look for the most probable value of A, or make use of known relationships between A and other attributes.

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

Additional enhancements to basic decision tree induction which address scalability, and the integration of data warehousing techniques, are discussed in Sections 7.3.5 and 7.3.6, respectively.

7.3.5 Scalability and decision tree induction

"How scalable is decision tree induction?"

The efficiency of existing decision tree algorithms, such as ID3 and C4.5, has been well established for relatively small data sets. Efficiency and scalability become issues of concern when these algorithms are applied to the mining of very large, real-world databases. Most decision tree algorithms have the restriction that the training samples should reside in main memory. In data mining applications, very large training sets of millions of samples are common. Hence, this restriction limits the scalability of such algorithms, where the decision tree construction can become inefficient due to swapping of the training samples in and out of main and cache memories.

Early strategies for inducing decision trees from large databases include discretizing continuous attributes, and sampling data at each node. These, however, still assume that the training set can fit in memory. An alternative method first partitions the data into subsets which individually can fit into memory, and then builds a decision tree from each subset. The final output classifier combines each classifier obtained from the subsets. Although this method allows for the classification of large data sets, its classification accuracy is not as high as the single classifier that would have been built using all of the data at once.

rid	credit_rating	age	buys_computer
1	excellent	38	yes
2	excellent	26	yes
3	fair	35	no
4	excellent	49	no

Table 7.2: Sample data for the class *buys_computer*.

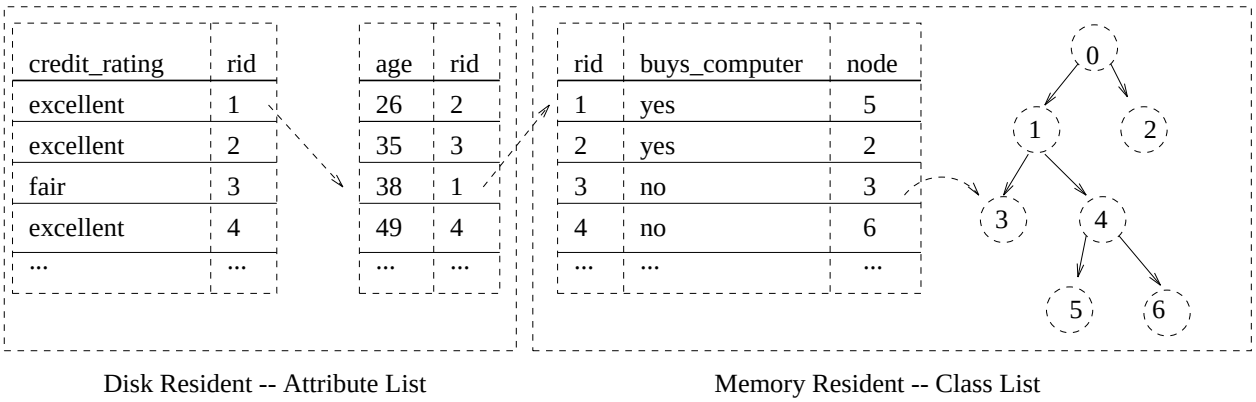


Figure 7.5: Attribute list and class list data structures used in SLIQ for the sample data of Table 7.2.

More recent decision tree algorithms which address the scalability issue have been proposed. Algorithms for the induction of decision trees from very large training sets include SLIQ and SPRINT, both of which can handle categorical and continuous-valued attributes. Both algorithms propose pre-sorting techniques on disk-resident data sets that are too large to fit in memory. Both define the use of new data structures to facilitate the tree construction. SLIQ employs disk resident *attribute lists* and a single memory resident *class list*. The attribute lists and class lists generated by SLIQ for the sample data of Table 7.2 are shown in Figure 7.5. Each attribute has an associated attribute list, indexed by *rid* (a record identifier). Each tuple is represented by a linkage of one entry from each attribute list to an entry in the class list (holding the class label of the given tuple), which in turn is linked to its corresponding leaf node in the decision tree. The class list remains in memory since it is often accessed and modified in the building and pruning phases. The size of the class list grows proportionally with the number of tuples in the training set. When a class list cannot fit into memory, the performance of SLIQ decreases.

SPRINT uses a different *attribute list* data structure which holds the class and *rid* information, as shown in Figure 7.6. When a node is split, the attribute lists are partitioned and distributed among the resulting child nodes

credit_rating	buys_computer	rid
excellent	yes	1
excellent	yes	2
fair	no	3
excellent	no	4
...	...	...

age	buys_computer	rid
26	y	2
35	n	3
38	y	1
49	n	4
...	...	...

Figure 7.6: Attribute list data structure used in SPRINT for the sample data of Table 7.2.

accordingly. When a list is partitioned, the order of the records in the list is maintained. Hence, partitioning lists does not require resorting. SPRINT was designed to be easily parallelized, further contributing to its scalability.

While both SLIQ and SPRINT handle disk-resident data sets that are too large to fit into memory, the scalability of SLIQ is limited by the use of its memory-resident data structure. SPRINT removes all memory restrictions, yet requires the use of a hash tree proportional in size to the training set. This may become expensive as the training set size grows.

RainForest is a framework for the scalable induction of decision trees. The method adapts to the amount of main memory available, and apply to any decision tree induction algorithm. It maintains an AVC-set (Attribute-Value, Class label) indicating the class distribution for each attribute. RainForest reports a speed-up over SPRINT.

7.3.6 Integrating data warehousing techniques and decision tree induction

Decision tree induction can be integrated with data warehousing techniques for data mining. In this section we discuss the method of attribute-oriented induction to generalize the given data, and the use of multidimensional data cubes to store the generalized data at multiple levels of granularity. We then discuss how these approaches can be integrated with decision tree induction in order to facilitate interactive multilevel mining. The use of a data mining query language to specify classification tasks is also discussed. In general, the techniques described here are applicable to other forms of learning as well.

Attribute-oriented induction (AOI) uses concept hierarchies to generalize the training data by replacing lower level data with higher level concepts (Chapter 5). For example, numerical values for the attribute *income* may be generalized to the ranges *"<30K", "30K-40K", ">40K"*, or the categories *low, medium*, or *high*. This allows the user to view the data at more meaningful levels. In addition, the generalized data are more compact than the original training set, which may result in fewer input/output operations. Hence, AOI also addresses the scalability issue by compressing the training data.

The generalized training data can be stored in a **multidimensional data cube**, such as the structure typically used in data warehousing (Chapter 2). The data cube is a multidimensional data structure, where each dimension represents an attribute or a set of attributes in the data schema, and each cell stores the value of some aggregate measure (such as *count*). Figure 7.7 shows a data cube for customer information data, with the dimensions *income*, *age*, and *occupation*. The original numeric values of *income and age* have been generalized to ranges. Similarly, original values for *occupation*, such as *accountant* and *banker*, or *nurse* and *X-ray technician*, have been generalized to *finance* and *medical*, respectively. The advantage of the multidimensional structure is that it allows fast indexing to cells (or *slices*) of the cube. For instance, one may easily and quickly access the total count of customers in occupations relating to finance who have an income greater than $40K, or the number of customers who work in the area of medicine and are less than 40 years old.

Data warehousing systems provide a number of operations that allow mining on the data cube at multiple levels of granularity. To review, the **roll-up** operation performs aggregation on the cube, either by climbing up a concept hierarchy (e.g., replacing the value *banker* for *occupation* by the more general, *finance*), or by removing a dimension in the cube. **Drill-down** performs the reverse of roll-up, by either stepping down a concept hierarchy or adding a dimension (e.g., time). A **slice** performs a selection on one dimension of the cube. For example, we may obtain a data slice for the generalized value *accountant* of *occupation*, showing the corresponding *income* and *age* data. A **dice** performs a selection on two or more dimensions. The **pivot** or *rotate* operation rotates the data axes in view

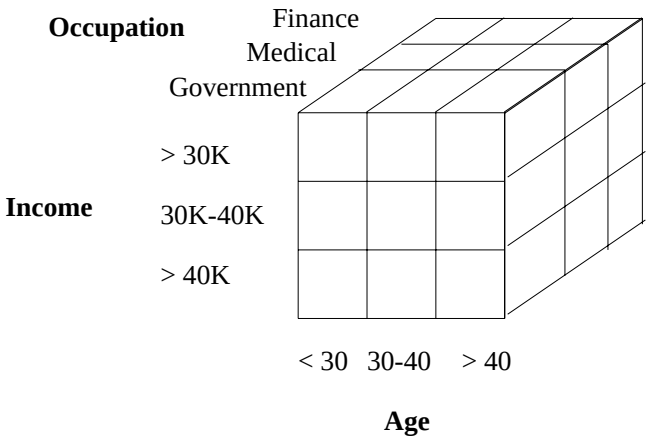


Figure 7.7: A multidimensional data cube.

in order to provide an alternative presentation of the data. For example, pivot may be used to transform a 3-D cube into a series of 2-D planes.

The above approaches can be integrated with decision tree induction to provide interactive multilevel mining of decision trees. The data cube and knowledge stored in the concept hierarchies can be used to induce decision trees at different levels of abstraction. Furthermore, once a decision tree has been derived, the concept hierarchies can be used to generalize or specialize individual nodes in the tree, allowing attribute roll-up or drill-down, and reclassification of the data for the newly specified abstraction level. This interactive feature will allow users to focus their attention on areas of the tree or data which they find interesting.

When integrating AOI with decision tree induction, generalization to a very low (specific) concept level can result in quite large and bushy trees. Generalization to a very high concept level can result in decision trees of little use, where interesting and important subconcepts are lost due to overgeneralization. Instead, generalization should be to some intermediate concept level, set by a domain expert or controlled by a user-specified threshold. Hence, the use of AOI may result in classification trees that are more understandable, smaller, and therefore easier to interpret than trees obtained from methods operating on ungeneralized (larger) sets of low-level data (such as SLIQ or SPRINT).

A criticism of typical decision tree generation is that, because of the recursive partitioning, some resulting data subsets may become so small that partitioning them further would have no statistically significant basis. The maximum size of such "insignificant" data subsets can be statistically determined. To deal with this problem, an **exception threshold** may be introduced. If the portion of samples in a given subset is less than the threshold, further partitioning of the subset is halted. Instead, a leaf node is created which stores the subset and class distribution of the subset samples.

Owing to the large amount and wide diversity of data in large databases, it may not be reasonable to assume that each leaf node will contain samples belonging to a common class. This problem may be addressed by employing a **precision** or **classification threshold**. Further partitioning of the data subset at a given node is terminated if the percentage of samples belonging to any given class at that node exceeds this threshold.

A data mining query language may be used to specify and facilitate the enhanced decision tree induction method. Suppose that the data mining task is to predict the credit risk of customers aged 30-40, based on their income and occupation. This may be specified as the following data mining query:

```
mine classification
analyze credit_risk
in relevance to income, occupation
from Customer_db
where (age >= 30) and (age < 40)
display as rules
```

The above query, expressed in DMQL[1], executes a relational query on *Customer_db* to retrieve the task-relevant data. Tuples not satisfying the where clause are ignored, and only the data concerning the attributes specified in the in relevance to clause, and the class label attribute (*credit_risk*) are collected. AOI is then performed on this data. Since the query has not specified which concept hierarchies to employ, default hierarchies are used. A graphical user interface may be designed to facilitate user specification of data mining tasks via such a data mining query language. In this way, the user can help guide the automated data mining process.

Hence, many ideas from data warehousing can be integrated with classification algorithms, such as decision tree induction, in order to facilitate data mining. Attribute-oriented induction employs concept hierarchies to generalize data to multiple abstraction levels, and can be integrated with classification methods in order to perform multilevel mining. Data can be stored in multidimensional data cubes to allow quick accessing to aggregate data values. Finally, a data mining query language can be used to assist users in interactive data mining.

7.4 Bayesian classification

"What are Bayesian classifiers"?

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

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

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

Section 7.4.1 reviews basic probability notation and Bayes theorem. You will then learn naive Bayesian classification in Section 7.4.2. Bayesian belief networks are described in Section 7.4.3.

7.4.1 Bayes theorem

Let X be a data sample whose class label is unknown. Let H be some hypothesis, such as that the data sample X belongs to a specified class C. For classification problems, we want to determine $P(H|X)$, the probability that the hypothesis H holds given the observed data sample X.

$P(H|X)$ is the **posterior probability**, or *a posteriori probability*, of H conditioned on X. For example, suppose the world of data samples consists of fruits, described by their color and shape. Suppose that X is red and round, and that H is the hypothesis that X is an apple. Then $P(H|X)$ reflects our confidence that X is an apple given that we have seen that X is red and round. In contrast, $P(H)$ is the **prior probability**, or *a priori probability* of H. For our example, this is the probability that any given data sample is an apple, regardless of how the data sample looks. The posterior probability, $P(H|X)$ is based on more information (such as background knowledge) than the prior probability, $P(H)$, which is independent of X.

Similarly, $P(X|H)$ is the posterior probability of X conditioned on H. That is, it is the probability that X is red and round given that we know that it is true that X is an apple. $P(X)$ is the prior probability of X. Using our example, it is the probability that a data sample from our set of fruits is red and round.

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

$$P(H|X) = \frac{P(X|H)P(H)}{P(X)} \tag{7.4}$$

In the next section, you will learn how Bayes theorem is used in the naive Bayesian classifier.

[1] The use of a data mining query language to specify data mining queries is discussed in Chapter 4, using the SQL-based DMQL language.

7.4.2 Naive Bayesian classification

The **naive Bayesian** classifier, or **simple Bayesian** classifier, works as follows:

1. Each data sample is represented by an n-dimensional feature vector, $X = (x_1, x_2, \ldots, x_n)$, depicting n measurements made on the sample from n attributes, respectively $A_1, A_2, .., A_n$.

2. Suppose that there are m classes, $C_1, C_2, \ldots, C_m$. Given an unknown data sample, X (i.e., having no class label), the classifier will predict that X belongs to the class having the highest posterior probability, conditioned on X. That is, the naive Bayesian classifier assigns an unknown sample X to the class C_i if and only if :

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

   Thus we maximize $P(C_i|X)$. The class C_i for which $P(C_i|X)$ is maximized is called the *maximum posteriori hypothesis*. By Bayes theorem (Equation (7.4)),

$$P(C_i|X) = \frac{P(X|C_i)P(C_i)}{P(X)}. \tag{7.5}$$

3. As $P(X)$ is constant for all classes, only $P(X|C_i)P(C_i)$ need be maximized. If the class prior probabilities are not known, then it is commonly assumed that the classes are equally likely, i.e. $P(C_1) = P(C_2) = \ldots = P(C_m)$, and we would therefore maximize $P(X|C_i)$. Otherwise, we maximize $P(X|C_i)P(C_i)$. Note that the class prior probabilities may be estimated by $P(C_i) = \frac{s_i}{s}$, where s_i is the number of training samples of class C_i, and s is the total number of training samples.

4. Given data sets with many attributes, it would be extremely computationally expensive to compute $P(X|C_i)$. In order to reduce computation in evaluating $P(X|C_i)$, the naive assumption of **class conditional independence** is made. This presumes that the values of the attributes are conditionally independent of one another, given the class label of the sample, i.e., that there are no dependence relationships among the attributes. Thus,

$$P(X|C_i) = \prod_{k=1}^{n} P(x_k|C_i). \tag{7.6}$$

   The probabilities $P(x_1|C_i), P(x_2|C_i), \ldots, P(x_n|C_i)$ can be estimated from the training samples, where:

   (a) If A_k is categorical, then $P(x_k|C_i) = \frac{s_{ik}}{s_i}$, where s_{ik} is the number of training samples of class C_i having the value x_k for A_k, and s_i is the number of training samples belonging to C_i.

   (b) If A_k is continuous-valued, then the attribute is assumed to have a Gaussian distribution. Therefore,

$$P(x_k|C_i) = g(x_k, \mu_{C_i}, \sigma_{C_i}) = \frac{1}{\sqrt{2\pi}\sigma_{C_i}} e^{-\frac{(x-\mu_{C_i})^2}{2\sigma_{C_i}^2}}, \tag{7.7}$$

   where $g(x_k, \mu_{C_i}, \sigma_{C_i})$ is the **Gaussian (normal) density function** for attribute A_k, while μ_{C_i} and σ_{C_i} are the mean and variance respectively given the values for attribute A_k for training samples of class C_i.

5. In order to classify an unknown sample X, $P(X|C_i)P(C_i)$ is evaluated for each class C_i. Sample X is then assigned to the class C_i if and only if :

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

   In other words, it is assigned to the class, C_i, for which $P(X|C_i)P(C_i)$ is the maximum.

"How effective are Bayesian classifiers?"

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

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

Example 7.4 Predicting a class label using naive Bayesian classification. We wish to predict the class label of an unknown sample using naive Bayesian classification, given the same training data as in Example 7.2 for decision tree induction. The training data are in Table 7.1. The data samples are described by the attributes *age*, *income*, *student*, and *credit_rating*. The class label attribute, *buys_computer*, has two distinct values (namely {*yes*, *no*}). Let C_1 correspond to the class *buys_computer = yes* and C_2 correspond to *buys_computer = no*. The unknown sample we wish to classify is

$$X = (age = \text{"<30"},\ income = medium,\ student = yes,\ credit\_rating = fair).$$

We need to maximize $P(X|C_i)P(C_i)$, for $i = 1, 2$. $P(C_i)$, the prior probability of each class, can be computed based on the training samples:

$$P(buys\_computer = yes) = 9/14 = 0.643$$
$$P(buys\_computer = no) = 5/14 = 0.357$$

To compute $P(X|C_i)$, for $i = 1, 2$, we compute the following conditional probabilities:

$$\begin{aligned}
&P(age = \text{"<30"} \mid buys_computer = yes) &&= 2/9 = 0.222\\
&P(age = \text{"<30"} \mid buys_computer = no) &&= 3/5 = 0.600\\
&P(income = medium \mid buys_computer = yes) &&= 4/9 = 0.444\\
&P(income = medium \mid buys_computer = no) &&= 2/5 = 0.400\\
&P(student = yes \mid buys_computer = yes) &&= 6/9 = 0.667\\
&P(student = yes \mid buys_computer = no) &&= 1/5 = 0.200\\
&P(credit_rating = fair \mid buys_computer = yes) &&= 6/9 = 0.667\\
&P(credit_rating = fair \mid buys_computer = no) &&= 2/5 = 0.400
\end{aligned}$$

Using the above probabilities, we obtain

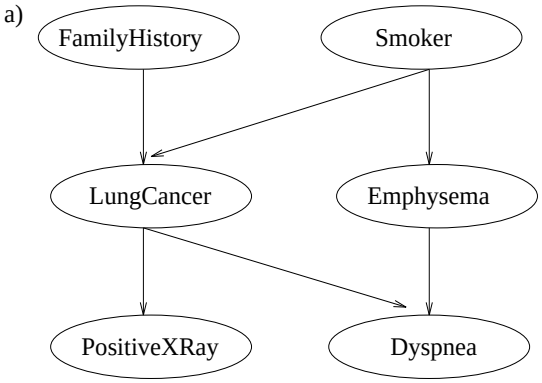
$$P(X|buys\_computer = yes) = 0.222 \times 0.444 \times 0.667 \times 0.667 = 0.044$$
$$P(X|buys\_computer = no) = 0.600 \times 0.400 \times 0.200 \times 0.400 = 0.019$$
$$P(X|buys\_computer = yes)P(buys\_computer = yes) = 0.044 \times 0.643 = 0.028$$
$$P(X|buys\_computer = no)P(buys\_computer = no) = 0.019 \times 0.357 = 0.007$$

Therefore, the naive Bayesian classifier predicts *"buys_computer = yes"* for sample X. □

7.4.3 Bayesian belief networks

The naive Bayesian classifier makes the assumption of class conditional independence, i.e., that given the class label of a sample, the values of the attributes are conditionally independent of one another. This assumption simplifies computation. When the assumption holds true, then the naive Bayesian classifier is the most accurate in comparison with all other classifiers. In practice, however, dependencies can exist between variables. **Bayesian belief networks** specify joint conditional probability distributions. They allow class conditional independencies to be defined between subsets of variables. They provide a graphical model of causal relationships, on which learning can be performed. These networks are also known as **belief networks**, **Bayesian networks**, and **probabilistic networks**. For brevity, we will refer to them as belief networks.

a)



b)

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

Figure 7.8: a) A simple Bayesian belief network; b) The conditional probability table for the values of the variable *LungCancer (LC)* showing each possible combination of the values of its parent nodes, *Family History (FH)* and *Smoker (S)*.

A belief network is defined by two components. The first is a *directed acyclic graph*, where each node represents a random variable, and each arc represents a probabilistic dependence. If an arc is drawn from a node Y to a node Z, then Y is a **parent** or **immediate predecessor** of Z, and Z is a **descendent** of Y. Each variable is conditionally independent of its nondescendents in the graph, given its parents. The variables may be discrete or continuous-valued. They may correspond to actual attributes given in the data, or to "hidden variables" believed to form a relationship (such as medical syndromes in the case of medical data).

Figure 7.8a) shows a simple belief network, adapted from [Russell et al. 1995a] for six Boolean variables. The arcs allow a representation of causal knowledge. For example, having lung cancer is influenced by a person's family history of lung cancer, as well as whether or not the person is a smoker. Furthermore, the arcs also show that the variable *LungCancer* is conditionally independent of *Emphysema*, given its parents, *FamilyHistory* and *Smoker*. This means that once the values of *FamilyHistory* and *Smoker* are known, then the variable *Emphysema* does not provide any additional information regarding *LungCancer*.

The second component defining a belief network consists of one *conditional probability table (CPT)* for each variable. The CPT for a variable Z specifies the conditional distribution $P(Z|Parents(Z))$, where $Parents(Z)$ are the parents of Z. Figure 7.8b) showns a CPT for *LungCancer*. The conditional probability for each value of *LungCancer* is given for each possible combination of values of its parents. For instance, from the upper leftmost and bottom rightmost entries, respectively, we see that

$$P(LungCancer = Yes \mid FamilyHistory = Yes,\ Smoker = Yes) = 0.8, \text{ and}$$
$$P(LungCancer = No \mid FamilyHistory = No,\ Smoker = No) = 0.9.$$

The joint probability of any tuple $(z_1, ..., z_n)$ corresponding to the variables or attributes $Z_1, ..., Z_n$ is computed by

$$P(z_1, ..., z_n) = \prod_{i=1}^{n} P(z_i|Parents(Z_i)), \tag{7.8}$$

where the values for $P(z_i|Parents(Z_i))$ correspond to the entries in the CPT for Z_i.

A node within the network can be selected as an "output" node, representing a class label attribute. There may be more than one output node. Inference algorithms for learning can be applied on the network. The classification process, rather than returning a single class label, can return a probability distribution for the class label attribute, i.e., predicting the probability of each class.

7.4.4 Training Bayesian belief networks

"How does a Bayesian belief network learn?"

In learning or training a belief network, a number of scenarios are possible. The network structure may be given in advance, or inferred from the data. The network variables may be *observable* or *hidden* in all or some of the training samples. The case of hidden data is also referred to as *missing values* or *incomplete data*.

If the network structure is known and the variables are observable, then learning the network is straightforward. It consists of computing the CPT entries, as is similarly done when computing the probabilities involved in naive Bayesian classification.

When the network structure is given and some of the variables are hidden, then a method of gradient descent can be used to train the belief network. The object is to learn the values for the CPT entries. Let S be a set of s training samples, $X_1, X_2, .., X_s$. Let w_{ijk} be a CPT entry for the variable $Y_i = y_{ij}$ having the parents $U_i = u_{ik}$. For example, if w_{ijk} is the upper leftmost CPT entry of Figure 7.8b), then Y_i is *LungCancer*; y_{ij} is its value, *Yes*; U_i lists the parent nodes of Y_i, namely $\{$*FamilyHistory, Smoker*$\}$; and u_{ik} lists the values of the parent nodes, namely $\{$*Yes, Yes*$\}$. The w_{ijk} are viewed as weights, analogous to the weights in hidden units of neural networks (Section 7.5). The weights, w_{ijk}, are initialized to random probability values. The gradient descent strategy performs greedy hill-climbing. At each iteration, the weights are updated, and will eventually converge to a local optimum solution.

The method aims to maximize $P(S|H)$. This is done by following the gradient of $lnP(S|H)$, which makes the problem simpler. Given the network structure and initialized w_{ijk}, the algorithm proceeds as follows.

1. **Compute the gradients:** For each i, j, k, compute

$$\frac{\partial lnP(S|H)}{\partial w_{ijk}} = \sum_{d=1}^{s} \frac{P(Y_i = y_{ij}, U_i = u_{ik}|X_d)}{w_{ijk}} \tag{7.9}$$

   The probability in the right-hand side of Equation (7.9) is to be calculated for each training sample X_d in S. For brevity, let's refer to this probability simply as p. When the variables represented by Y_i and U_i are hidden for some X_d, then the corresponding probability p can be computed from the observed variables of the sample using standard algorithms for Bayesian network inference (such as those available by the commercial software package, Hugin).

2. **Take a small step in the direction of the gradient:** The weights are updated by

$$w_{ijk} \leftarrow w_{ijk} + (l)\frac{\partial lnP(S|H)}{\partial w_{ijk}}, \tag{7.10}$$

   where l is the **learning rate** representing the step size, and $\frac{\partial lnP(S|H)}{\partial w_{ijk}}$ is computed from Equation (7.9). The learning rate is set to a small constant.

3. **Renormlize the weights:** Because the weights w_{ijk} are probability values, they must be between 0 and 1.0, and $\sum_j w_{ijk}$ must equal 1 for all i, k. These criteria are achieved by renormalizing the weights after they have been updated by Equation (7.10).

Several algorithms exist for learning the network structure from the training data given observable variables. The problem is one of discrete optimization. For solutions, please see the bibliographic notes at the end of this chapter.

7.5 Classification by backpropagation

"What is backpropagation?"

Backpropagation is a neural network learning algorithm. The field of neural networks was originally kindled by psychologists and neurobiologists who sought to develop and test computational analogues of neurons. Roughly

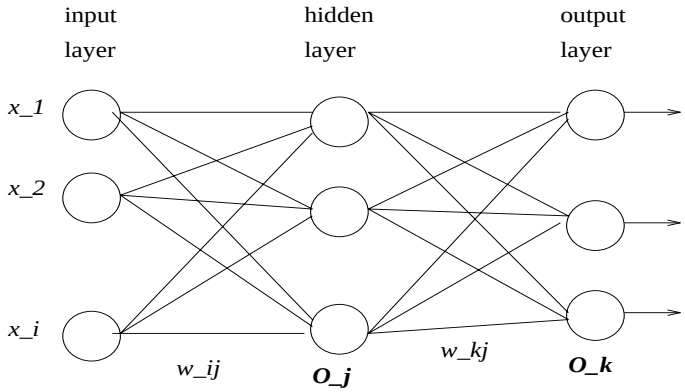


Figure 7.9: A multilayer feed-forward neural network: A training sample, $X = (x_1, x_2, .., x_i)$, is fed to the input layer. Weighted connections exist between each layer, where w_{ij} denotes the weight from a unit j in one layer to a unit i in the previous layer.

speaking, a **neural network** is a set of connected input/output units where each connection has a weight associated with it. During the learning phase, the network learns by adjusting the weights so as to be able to predict the correct class label of the input samples. Neural network learning is also referred to as *connectionist learning* due to the connections between units.

Neural networks involve long training times, and are therefore more suitable for applications where this is feasible. They require a number of parameters which are typically best determined empirically, such as the network topology or "structure". Neural networks have been criticized for their poor interpretability, since it is difficult for humans to interpret the symbolic meaning behind the learned weights. These features initially made neural networks less desirable for data mining.

Advantages of neural networks, however, include their high tolerance to noisy data as well as their ability to classify patterns on which they have not been trained. In addition, several algorithms have recently been developed for the extraction of rules from trained neural networks. These factors contribute towards the usefulness of neural networks for classification in data mining.

The most popular neural network algorithm is the backpropagation algorithm, proposed in the 1980's. In Section 7.5.1 you will learn about multilayer feed-forward networks, the type of neural network on which the backpropagation algorithm performs. Section 7.5.2 discusses defining a network topology. The backpropagation algorithm is described in Section 7.5.3. Rule extraction from trained neural networks is discussed in Section 7.5.4.

7.5.1 A multilayer feed-forward neural network

The backpropagation algorithm performs learning on a **multilayer feed-forward** neural network. An example of such a network is shown in Figure 7.9. The inputs correspond to the attributes measured for each training sample. The inputs are fed simultaneously into a layer of units making up the **input layer**. The weighted outputs of these units are, in turn, fed simultaneously to a second layer of "neuron-like" units, known as a **hidden layer**. The hidden layer's weighted outputs can be input to another hidden layer, and so on. The number of hidden layers is arbitrary, although in practice, usually only one is used. The weighted outputs of the last hidden layer are input to units making up the **output layer**, which emits the network's prediction for given samples.

The units in the hidden layers and output layer are sometimes referred to as **neurodes**, due to their symbolic biological basis, or as **output units**. The multilayer neural network shown in Figure 7.9 has two layers of output units. Therefore, we say that it is a **two-layer** neural network. Similarly, a network containing two hidden layers is called a *three-layer* neural network, and so on. The network is **feed-forward** in that none of the weights cycle back to an input unit or to an output unit of a previous layer. It is **fully connected** in that each unit provides input to each unit in the next forward layer.

Multilayer feed-forward networks of linear threshold functions, given enough hidden units, can closely approximate any function.

7.5.2 Defining a network topology

"How can I design the topology of the neural network?"

Before training can begin, the user must decide on the network topology by specifying the number of units in the input layer, the number of hidden layers (if more than one), the number of units in each hidden layer, and the number of units in the output layer.

Normalizing the input values for each attribute measured in the training samples will help speed up the learning phase. Typically, input values are normalized so as to fall between 0 and 1.0. Discrete-valued attributes may be encoded such that there is one input unit per domain value. For example, if the domain of an attribute A is $\{a_0, a_1, a_2\}$, then we may assign three input units to represent A. That is, we may have, say, I_0, I_1, I_2, as input units. Each unit is initialized to 0. If $A = a_0$, then I_0 is set to 1. If $A = a_1$, I_1 is set to 1, and so on. One output unit may be used to represent two classes (where the value 1 represents one class, and the value 0 represents the other). If there are more than two classes, then 1 output unit per class is used.

There are no clear rules as to the "best" number of hidden layer units. Network design is a trial by error process and may affect the accuracy of the resulting trained network. The initial values of the weights may also affect the resulting accuracy. Once a network has been trained and its accuracy is not considered acceptable, then it is common to repeat the training process with a different network topology or a different set of initial weights.

7.5.3 Backpropagation

"How does backpropagation work?"

Backpropagation learns by iteratively processing a set of training samples, comparing the network's prediction for each sample with the actual known class label. For each training sample, the weights are modified so as to minimize the mean squared error between the network's prediction and the actual class. These modifications are made in the "backwards" direction, i.e., from the output layer, through each hidden layer down to the first hidden layer (hence the name *backpropagation*). Although it is not guaranteed, in general the weights will eventually converge, and the learning process stops. The algorithm is summarized in Figure 7.10. Each step is described below.

Initialize the weights. The weights in the network are initialized to small random numbers (e.g., ranging from -1.0 to 1.0, or -0.5 to 0.5). Each unit has a *bias* associated with it, as explained below. The biases are similarly initialized to small random numbers.

Each training sample, X, is processed by the following steps.

Propagate the inputs forward. In this step, the net input and output of each unit in the hidden and output layers are computed. First, the training sample is fed to the input layer of the network. The net input to each unit in the hidden and output layers is then computed as a linear combination of its inputs. To help illustrate this, a hidden layer or output layer unit is shown in Figure 7.11. The inputs to the unit are, in fact, the outputs of the units connected to it in the previous layer. To compute the net input to the unit, each input connected to the unit is multiplied by its corresponding weight, and this is summed. Given a unit j in a hidden or output layer, the net input, I_j, to unit j is:

$$I_j = \sum_i w_{ij} O_i + \theta_j \tag{7.11}$$

where w_{ij} is the weight of the connection from unit i in the previous layer to unit j; O_i is the output of unit i from the previous layer; and θ_j is the **bias** of the unit. The bias acts as a threshold in that it serves to vary the activity of the unit.

Each unit in the hidden and output layers takes its net input, and then applies an **activation** function to it, as illustrated in Figure 7.11. The function symbolizes the activation of the neuron represented by the unit. The **logistic**, or **simoid** function is used. Given the net input I_j to unit j, then O_j, the output of unit j, is computed as:

$$O_j = \frac{1}{1 + e^{-I_j}} \tag{7.12}$$

This function is also referred to as a *squashing function*, since it maps a large input domain onto the smaller range

Algorithm 7.5.1 (Backpropagation) Neural network learning for classification, using the backpropagation algorithm.

Input: The training samples, *samples*; the learning rate, *l*; a multilayer feed-forward network, *network*.

Output: A neural network trained to classify the samples.

Method:

1) Initialize all weights and biases in *network*;
2) **while** terminating condition is not satisfied {
3) **for** each training sample X in *samples* {
4) // Propagate the inputs forward:
5) **for** each hidden or output layer unit j
6) $I_j = \sum_i w_{ij} O_i + \theta_j$; //compute the net input of unit j
7) **for** each hidden or output layer unit j
8) $O_j = \frac{1}{1+e^{-I_j}}$; // compute the output of each unit j
9) // Backpropagate the errors:
10) **for** each unit j in the output layer
11) $Err_j = O_j(1-O_j)(T_j - O_j)$; // compute the error
12) **for** each unit j in the hidden layers
13) $Err_j = O_j(1-O_j)\sum_k Err_k w_{jk}$; // compute the error
14) **for** each weight w_{ij} in *network* {
15) $\Delta w_{ij} = (l) Err_j O_i$; // weight increment
16) $w_{ij} = w_{ij} + \Delta w_{ij}$; } // weight update
17) **for** each bias θ_j in *network* {
18) $\Delta \theta_j = (l) Err_j$; // bias increment
19) $\theta_j = \theta_j + \Delta \theta_j$; } // bias update
20) } }

□

Figure 7.10: Backpropagation algorithm.

of 0 to 1. The logistic function is nonlinear and differentiable, allowing the backpropagation algorithm to model classification problems that are linearly inseparable.

Backpropagate the error. The error is propagated backwards by updating the weights and biases to reflect the error of the network's prediction. For a unit j in the output layer, the error Err_j is computed by:

$$Err_j = O_j(1-O_j)(T_j - O_j) \tag{7.13}$$

where O_j is the actual output of unit j, and T_j is the *true* output, based on the known class label of the given training sample. Note that $O_j(1-O_j)$ is the derivative of the logistic function.

To compute the error of a hidden layer unit j, the weighted sum of the errors of the units connected to unit j in the next layer are considered. The error of a hidden layer unit j is:

$$Err_j = O_j(1-O_j)\sum_k Err_k w_{jk} \tag{7.14}$$

where w_{jk} is the weight of the connection from unit j to a unit k in the next higher layer, and Err_k is the error of unit k.

The weights and biases are updated to reflect the propagated errors. Weights are updated by Equations (7.15) and (7.16) below, where Δw_{ij} is the change in weight w_{ij}.

$$\Delta w_{ij} = (l) Err_j O_i \tag{7.15}$$

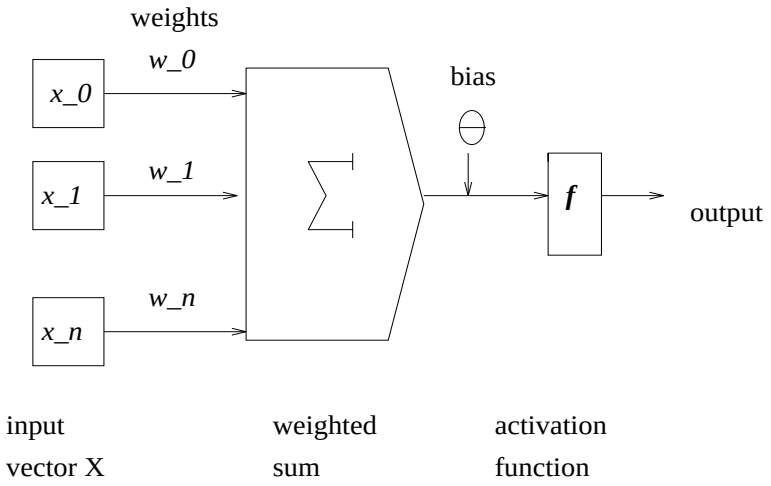


Figure 7.11: A hidden or output layer unit: The inputs are multiplied by their corresponding weights in order to form a weighted sum, which is added to the bias associated with the unit. A nonlinear activation function is applied to the net input.

$$w_{ij} = w_{ij} + \Delta w_{ij} \tag{7.16}$$

"What is the 'l' in Equation 7.15?" The variable l is the **learning rate**, a constant typically having a value between 0 and 1.0. Backpropagation learns using a method of gradient descent to search for a set of weights which can model the given classification problem so as to minimize the mean squared distance between the network's class predictions and the actual class label of the samples. The learning rate helps to avoid getting stuck at a local minimum in decision space (i.e., where the weights appear to converge, but are not the optimum solution), and encourages finding the global minimum. If the learning rate is too small, then learning will occur at a very slow pace. If the learning rate is too large, then oscillation between inadequate solutions may occur. A rule of thumb is to set the learning rate to $1/t$, where t is the number of iterations through the training set so far.

Biases are updated by Equations (7.17) and (7.18) below, where $\Delta\theta_j$ is the change in bias θ_j.

$$\Delta\theta_j = (l)Err_j \tag{7.17}$$

$$\theta_j = \theta_j + \Delta\theta_j \tag{7.18}$$

Note that here we are updating the weights and biases after the presentation of each sample. This is referred to as **case updating**. Alternatively, the weight and bias increments could be accumulated in variables, so that the weights and biases are updated after all of the samples in the training set have been presented. This latter strategy is called **epoch updating**, where one iteration through the training set is an **epoch**. In theory, the mathematical derivation of backpropagation employs epoch updating, yet in practice, case updating is more common since it tends to yield more accurate results.

Terminating condition. Training stops when either

1. all Δw_{ij} in the previous epoch were so small as to be below some specified threshold, or
2. the percentage of samples misclassified in the previous epoch is below some threshold, or
3. a prespecified number of epochs has expired.

In practice, several hundreds of thousands of epochs may be required before the weights will converge.

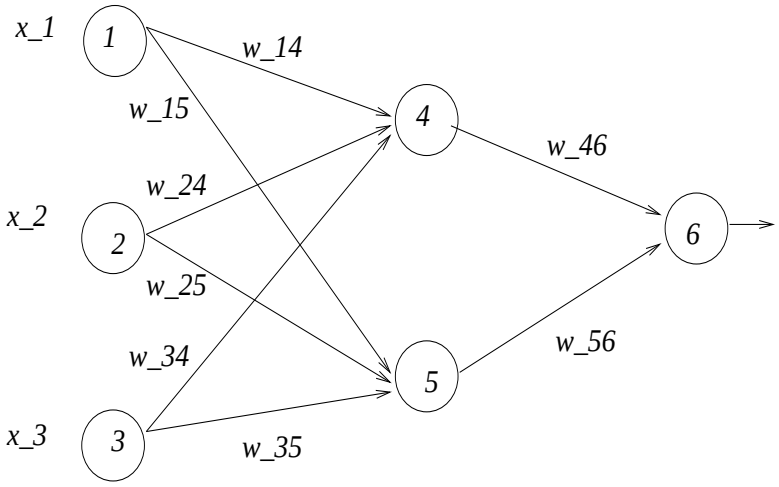


Figure 7.12: An example of a multilayer feed-forward neural network.

Example 7.5 Sample calculations for learning by the backpropagation algorithm. Figure 7.12 shows a multilayer feed-forward neural network. The initial weight and bias values of the network are given in Table 7.3, along with the first training sample, $X = (1, 0, 1)$.

x_1	x_2	x_3	w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{56}	θ_4	θ_5	θ_6
1	0	1	0.2	-0.3	0.4	0.1	-0.5	0.2	-0.3	-0.2	-0.4	0.2	0.1

Table 7.3: Initial input, weight, and bias values.

This example shows the calculations for backpropagation, given the first training sample, X. The sample is fed into the network, and the net input and output of each unit are computed. These values are shown in Table 7.4.

Unit j	Net Input, I_j	Output, O_j
4	$0.2 + 0 - 0.5 - 0.4 = -0.7$	$1/(1 + e^{0.7}) = 0.33$
5	$-0.3 + 0 + 0.2 + 0.2 = 0.1$	$1/(1 + e^{-0.1}) = 0.52$
6	$(0.3)(0.33) - (0.2)(0.52) + 0.1 - 0.19$	$1/(1 + e^{-0.19}) = 0.55$

Table 7.4: The net input and output calculations.

The error of each unit is computed and propagated backwards. The error values are shown in Table 7.5. The weight and bias updates are shown in Table 7.6.

□

Several variations and alternatives to the backpropagation algorithm have been proposed for classification in neural networks. These may involve the dynamic adjustment of the network topology, and of the learning rate or other parameters, or the use of different error functions.

7.5.4 Backpropagation and interpretability

"How can I 'understand' what the backpropgation network has learned?"

A major disadvantage of neural networks lies in their knowledge representation. Acquired knowledge in the form of a network of units connected by weighted links is difficult for humans to interpret. This factor has motivated research

Unit j	Err_j
6	$(0.55)(1-0.55)(1-0.55) = 0.495$
5	$(0.52)(1-0.52)(0.495)(-0.3) = 0.037$
4	$(0.33)(1-0.33)(0.495)(-0.2) = -0.022$

Table 7.5: Calculation of the error at each node.

Weight or Bias	New Value
w_{46}	$-0.3 = (0.9)(0.495)(0.33) = -0.153$
w_{56}	$-0.2 = (0.9)(0.495)(0.52) = -0.032$
w_{14}	$0.2 = (0.9)(-0.022)(1) = 0.180$
w_{15}	$-0.3 = (0.9)(0.037)(1) = -0.267$
w_{24}	$0.4 = (0.9)(-0.022)(0) = 0.4$
w_{25}	$0.1 = (0.9)(0.037)(0) = 0.1$
w_{34}	$-0.5 = (0.9)(-0.022)(1) = -0.520$
w_{35}	$0.2 = (0.9)(0.037)(1) = 0.233$
θ_6	$0.1 + (0.9)(0.495) = 0.546$
θ_5	$0.2 + (0.9)(0.037) = 0.233$
θ_4	$-0.4 + (0.9)(-0.022) = -0.420$

Table 7.6: Calculations for weight and bias updating.

in extracting the knowledge embedded in trained neural networks and in representing that knowledge symbolically. Methods include extracting rules from networks and sensitivity analysis.

Various algorithms for the extraction of rules have been proposed. The methods typically impose restrictions regarding procedures used in training the given neural network, the network topology, and the discretization of input values.

Fully connected networks are difficult to articulate. Hence, often, the first step towards extracting rules from neural networks is **network pruning**. This consists of removing weighted links that do not result in a decrease in the classification accuracy of the given network.

Once the trained network has been pruned, some approaches will then perform link, unit, or activation value clustering. In one method, for example, clustering is used to find the set of common activation values for each hidden unit in a given trained two-layer neural network (Figure 7.13). The combinations of these activation values for each hidden unit are analyzed. Rules are derived relating combinations of activation values with corresponding output unit values. Similarly, the sets of input values and activation values are studied to derive rules describing the relationship between the input and hidden unit layers. Finally, the two sets of rules may be combined to form IF-THEN rules. Other algorithms may derive rules of other forms, including M-of-N rules (where M out of a given N conditions in the rule antecedent must be true in order for the rule consequent to be applied), decision trees with M-of-N tests, fuzzy rules, and finite automata.

Sensitivity analysis is used to assess the impact that a given input variable has on a network output. The input to the variable is varied while the remaining input variables are fixed at some value. Meanwhile, changes in the network output are monitored. The knowledge gained from this form of analysis can be represented in rules such as "*IF X decreases* 5% *THEN Y increases* 8%".

7.6 Association-based classification

"Can association rule mining be used for classification?"

Association rule mining is an important and highly active area of data mining research. Chapter 6 of this book described many algorithms for association rule mining. Recently, data mining techniques have been developed which apply association rule mining to the problem of classification. In this section, we study such association-based

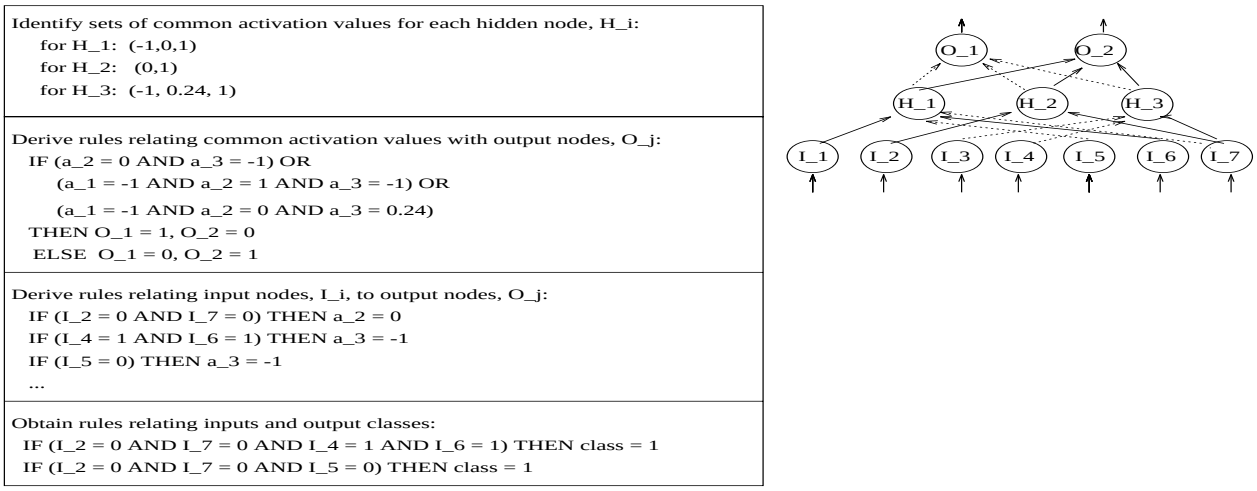


Figure 7.13: Rules can be extracted from training neural networks.

classification.

One method of association-based classification, called **associative classification**, consists of two steps. In the first step, association rules are generated using a modified version of the standard association rule mining algorithm known as Apriori. The second step constructs a classifier based on the association rules discovered.

Let D be the training data, and Y be the set of all classes in D. The algorithm maps categorical attributes to consecutive positive integers. Continuous attributes are discretized and mapped accordingly. Each data sample d in D then is represented by a set of (attribute, integer-value) pairs called **items**, and a class label y. Let I be the set of all items in D. A **class association rule (CAR)** is of the form $condset \Rightarrow y$, where $condset$ is a set of items ($condset \subseteq I$) and $y \in Y$. Such rules can be represented by **ruleitems** of the form <**condset, y**>.

A CAR has confidence c if c% of the samples in D that contain $condset$ belong to class y. A CAR has support s if s% of the samples in D contain $condset$ and belong to class y. The support count of a condset (**condsupCount**) is the number of samples in D that contain the condset. The rule count of a ruleitem (**rulesupCount**) is the number of samples in D that contain the condset and are labeled with class y. Ruleitems that satisfy minimum support are **frequent ruleitems**. If a set of ruleitems has the same condset, then the rule with the highest confidence is selected as the **possible rule (PR)** to represent the set. A rule satisfying minimum confidence is called **accurate**.

"How does associative classification work?"

The first step of the associative classification method finds the set of all PRs that are both frequent and accurate. These are the class association rules (CARs). A ruleitem whose condset contains k items is a **k-ruleitem**. The algorithm employs an iterative approach, similar to that described for Apriori in Section 5.2.1, where ruleitems are processed rather than itemsets. The algorithm scans the database, searching for the frequent k-ruleitems, for $k = 1, 2, ..$, until all frequent k-ruleitems have been found. One scan is made for each value of k. The k-ruleitems are used to explore (k+1)-ruleitems. In the first scan of the database, the count support of 1-ruleitems is determined, and the frequent 1-ruleitems are retained. The frequent 1-ruleitems, referred to as the set F_1, are used to generate candidate 2-ruleitems, C_2. Knowledge of frequent ruleitem properties is used to prune candidate ruleitems that cannot be frequent. This knowledge states that *all non-empty subsets of a frequent ruleitem must also be frequent.* The database is scanned a second time to compute the support counts of each candidate, so that the frequent 2-ruleitems (F_2) can be determined. This process repeats, where F_k is used to generate C_{k+1}, until no more frequent ruleitems are found. The frequent ruleitems that satisfy minimum confidence form the set of CARs. Pruning may be applied to this rule set.

The second step of the associative classification method processes the generated CARs in order to construct the classifier. Since the total number of rule subsets that would be examined in order to determine the most accurate set of rules can be huge, a heuristic method is employed. A precedence ordering among rules is defined where a rule r_i has greater precedence over a rule r_j (i.e., $r_i \succ r_j$) if (1) the confidence of r_i is greater than that of r_j, or (2) the confidences are the same, but r_i has greater support, or (3) the confidences and supports of r_i and r_j are the same, but r_i is generated earlier than r_j. In general, the algorithm selects a set of high precedence CARs to cover the samples in D. The algorithm requires slightly more than one pass over D in order to determine the final classifier. The classifier maintains the selected rules from high to low precedence order. When classifying a new sample, the first rule satisfying the sample is used to classify it. The classifier also contains a default rule, having lowest precedence, which specifies a default class for any new sample that is not satisfied by any other rule in the classifier.

In general, the above associative classification method was empirically found to be more accurate than C4.5 on several data sets. Each of the above two steps was shown to have linear scale-up.

Association rule mining based on clustering has also been applied to classification. The ARCS, or Association Rule Clustering System (Section 6.4.3) mines association rules of the form $A_{quan1} \wedge A_{quan2} \Rightarrow A_{cat}$, where A_{quan1} and A_{quan2} are tests on quantitative attribute ranges (where the ranges are dynamically determined), and A_{cat} assigns a class label for a categorical attribute from the given training data. Association rules are plotted on a 2-D grid. The algorithm scans the grid, searching for rectangular clusters of rules. In this way, adjacent ranges of the quantitative attributes occurring within a rule cluster may be combined. The clustered association rules generated by ARCS were applied to classification, and their accuracy was compared to C4.5. In general, ARCS is slightly more accurate when there are outliers in the data. The accuracy of ARCS is related to the degree of discretization used. In terms of scalability, ARCS requires a constant amount of memory, regardless of the database size. C4.5 has exponentially higher execution times than ARCS, requiring the entire database, multiplied by some factor, to fit entirely in main memory. Hence, association rule mining is an important strategy for generating accurate and scalable classifiers.

7.7 Other classification methods

In this section, we give a brief description of a number of other classification methods. These methods include k-nearest neighbor classification, case-based reasoning, genetic algorithms, rough set and fuzzy set approaches. In general, these methods are less commonly used for classification in commercial data mining systems than the methods described earlier in this chapter. Nearest-neighbor classification, for example, stores all training samples, which may present difficulties when learning from very large data sets. Furthermore, many applications of case-based reasoning, genetic algorithms, and rough sets for classification are still in the prototype phase. These methods, however, are enjoying increasing popularity, and hence we include them here.

7.7.1 k-nearest neighbor classifiers

Nearest neighbor classifiers are based on learning by analogy. The training samples are described by n-dimensional numeric attributes. Each sample represents a point in an n-dimensional space. In this way, all of the training samples are stored in an n-dimensional pattern space. When given an unknown sample, a **k-nearest neighbor classifier** searches the pattern space for the k training samples that are closest to the unknown sample. These k training samples are the k "nearest neighbors" of the unknown sample. "Closeness" is defined in terms of Euclidean distance, where the Euclidean distance between two points, $X = (x_1, x_2, ..., x_n)$ and $Y = (y_1, y_2, ..., y_n)$ is:

$$d(X, Y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}. \tag{7.19}$$

The unknown sample is assigned the most common class among its k nearest neighbors. When $k = 1$, the unknown sample is assigned the class of the training sample that is closest to it in pattern space.

Nearest neighbor classifiers are **instance-based** since they store all of the training samples. They can incur expensive computational costs when the number of potential neighbors (i.e., stored training samples) with which to compare a given unlabeled sample is great. Therefore, efficient indexing techniques are required. Unlike decision tree

induction and backpropagation, nearest neighbor classifiers assign equal weight to each attribute. This may cause confusion when there are many irrelevant attributes in the data.

Nearest neighbor classifiers can also be used for prediction, i.e., to return a real-valued prediction for a given unknown sample. In this case, the classifier returns the average value of the real-valued labels associated with the k nearest neighbors of the unknown sample.

7.7.2 Case-based reasoning

Case-based reasoning (CBR) classifiers are instanced-based. Unlike nearest neighbor classifiers, which store training samples as points in Euclidean space, the samples or "cases" stored by CBR are complex symbolic descriptions. Business applications of CBR include problem resolution for customer service help desks, for example, where cases describe product-related diagnostic problems. CBR has also been applied to areas such as engineering and law, where cases are either technical designs or legal rulings, respectively.

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

Challenges in case-based reasoning include finding a good similarity metric (e.g., for matching subgraphs), developing efficient techniques for indexing training cases, and methods for combining solutions.

7.7.3 Genetic algorithms

Genetic algorithms attempt to incorporate ideas of natural evolution. In general, genetic learning starts as follows. An initial **population** is created consisting of randomly generated rules. Each rule can be represented by a string of bits. As a simple example, suppose that samples in a given training set are described by two Boolean attributes, A_1 and A_2, and that there are two classes, C_1 and C_2. The rule "*IF* A_1 *and not* A_2 *THEN* C_2" can be encoded as the bit string "100", where the two leftmost bits represent attributes A_1 and A_2, respectively, and the rightmost bit represents the class. Similarly, the rule "*if not* A_1 *and not* A_2 *then* C_1" can be encoded as "001". If an attribute has k values where $k > 2$, then k bits may be used to encode the attribute's values. Classes can be encoded in a similar fashion.

Based on the notion of survival of the fittest, a new population is formed to consist of the *fittest* rules in the current population, as well as *offspring* of these rules. Typically, the **fitness** of a rule is assessed by its classification accuracy on a set of training samples.

Offspring are created by applying genetic operators such as crossover and mutation. In **crossover**, substrings from pairs of rules are swapped to form new pairs of rules. In **mutation**, randomly selected bits in a rule's string are inverted.

The process of generating new populations based on prior populations of rules continues until a population P "evolves" where each rule in P satisfies a prespecified fitness threshold.

Genetic algorithms are easily parallelizable and have been used for classification as well as other optimization problems. In data mining, they may be used to evaluate the fitness of other algorithms.

7.7.4 Rough set theory

Rough set theory can be used for classification to discover structural relationships within imprecise or noisy data. It applies to discrete-valued attributes. Continuous-valued attributes must therefore be discretized prior to its use.

Rough set theory is based on the establishment of **equivalence classes** within the given training data. All of the data samples forming an equivalence class are indiscernible, that is, the samples are identical with respect to the attributes describing the data. Given real-world data, it is common that some classes cannot be distinguished

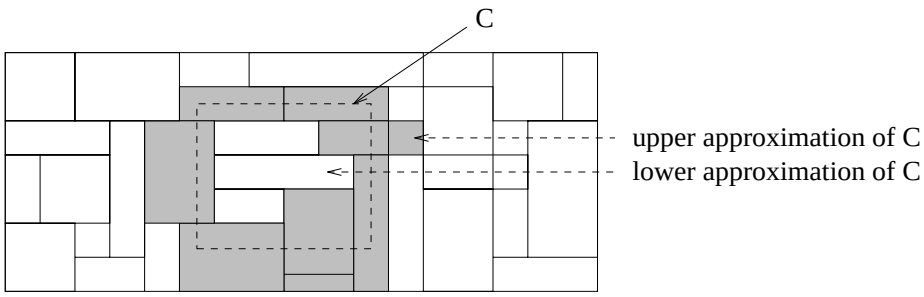


Figure 7.14: A rough set approximation of the set of samples of the class C using lower and upper approximation sets of C. The rectangular regions represent equivalence classes.

in terms of the available attributes. Rough sets can be used to approximately or "roughly" define such classes. A rough set definition for a given class C is approximated by two sets - a **lower approximation** of C and an **upper approximation** of C. The lower approximation of C consists of all of the data samples which, based on the knowledge of the attributes, are certain to belong to C without ambiguity. The upper approximation of C consists of all of the samples which, based on the knowledge of the attributes, cannot be described as not belonging to C. The lower and upper approximations for a class C are shown in Figure 7.14, where each rectangular region represents an equivalence class. Decision rules can be generated for each class. Typically, a decision table is used to represent the rules.

Rough sets can also be used for feature reduction (where attributes that do not contribute towards the classification of the given training data can be identified and removed), and relevance analysis (where the contribution or significance of each attribute is assessed with respect to the classification task). The problem of finding the minimal subsets (**reducts**) of attributes that can describe all of the concepts in the given data set is NP-hard. However, algorithms to reduce the computation intensity have been proposed. In one method, for example, a **discernibility matrix** is used which stores the differences between attribute values for each pair of data samples. Rather than searching on the entire training set, the matrix is instead searched to detect redundant attributes.

7.7.5 Fuzzy set approaches

Rule-based systems for classification have the disadvantage that they involve sharp cut-offs for continuous attributes. For example, consider Rule (7.20) below for customer credit application approval. The rule essentially says that applications for customers who have had a job for two or more years, and who have a high income (i.e., of more than \$50K) are approved.

$$IF\ (years\_employed >= 2) \wedge (income \geq 50K)\ THEN\ credit = approved. \tag{7.20}$$

By Rule (7.20), a customer who has had a job for at least 2 years will receive credit if her income is, say, \$51K, but not if it is \$50K. Such harsh thresholding may seem unfair. Instead, fuzzy logic can be introduced into the system to allow "fuzzy" thresholds or boundaries to be defined. Rather than having a precise cutoff between categories or sets, fuzzy logic uses truth values between 0.0 and 1.0 to represent the degree of membership that a certain value has in a given category. Hence, with fuzzy logic, we can capture the notion that an income of \$50K is, to some degree, high, although not as high as an income of \$51K.

Fuzzy logic is useful for data mining systems performing classification. It provides the advantage of working at a high level of abstraction. In general, the use of fuzzy logic in rule-based systems involves the following:

- Attribute values are converted to fuzzy values. Figure 7.15 shows how values for the continuous attribute *income* are mapped into the discrete categories {*low, medium, high*}, as well as how the fuzzy membership or truth values are calculated. Fuzzy logic systems typically provide graphical tools to assist users in this step.
- For a given new sample, more than one fuzzy rule may apply. Each applicable rule contributes a vote for membership in the categories. Typically, the truth values for each predicted category are summed.

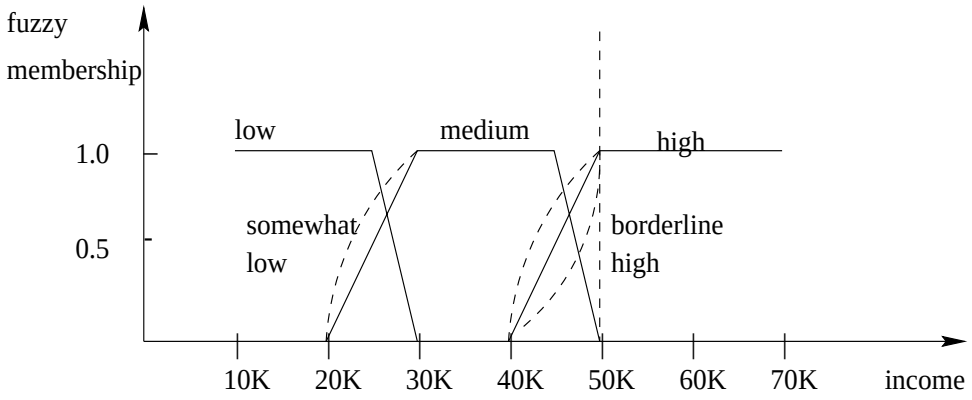


Figure 7.15: Fuzzy values for *income*.

- The sums obtained above are combined into a value that is returned by the system. This process may be done by weighting each category by its truth sum and multiplying by the mean truth value of each category. The calculations involved may be more complex, depending on the complexity of the fuzzy membership graphs.

Fuzzy logic systems have been used in numerous areas for classification, including health care and finance.

7.8 Prediction

"What if we would like to predict a continuous value, rather than a categorical label?"

The prediction of continuous values can be modeled by statistical techniques of *regression*. For example, we may like to develop a model to predict the salary of college graduates with 10 years of work experience, or the potential sales of a new product given its price. Many problems can be solved by *linear regression*, and even more can be tackled by applying transformations to the variables so that a nonlinear problem can be converted to a linear one. For reasons of space, we cannot give a fully detailed treatment of regression. Instead, this section provides an intuitive introduction to the topic. By the end of this section, you will be familiar with the ideas of linear, multiple, and nonlinear regression, as well as generalized linear models.

Several software packages exist to solve regression problems. Examples include SAS (*http://www.sas.com*), SPSS (*http://www.spss.com*), and S-Plus (*http://www.mathsoft.com*).

7.8.1 Linear and multiple regression

"What is linear regression?"

In **linear regression**, data are modeled using a straight line. Linear regression is the simplest form of regression. Bivariate linear regression models a random variable, Y (called a **response variable**), as a linear function of another random variable, X (called a **predictor variable**), i.e.,

$$Y = \alpha + \beta X, \tag{7.21}$$

where the variance of Y is assumed to be constant, and α and β are **regression coefficients** specifying the Y-intercept and slope of the line, 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. Given s samples or data points of the form (x_1, y_1), (x_2, y_2), .., (x_s, y_s), then the regression coefficients can be estimated using this method with Equations (7.22) and (7.23),

$$\beta = \frac{\sum_{i=1}^{s}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{s}(x_i - \bar{x})^2}, \tag{7.22}$$

$$\alpha = \bar{y} - \beta\bar{x}, \tag{7.23}$$

where $\bar{x}$ is the average of $x_1, x_2, .., x_s$, and $\bar{y}$ is the average of $y_1, y_2, .., y_s$. The coefficients α and β often provide good approximations to otherwise complicated regression equations.

X *years experience*	Y *salary (in $1000)*
3	30
8	57
9	64
13	72
3	36
6	43
11	59
21	90
1	20
16	83

Table 7.7: Salary data.

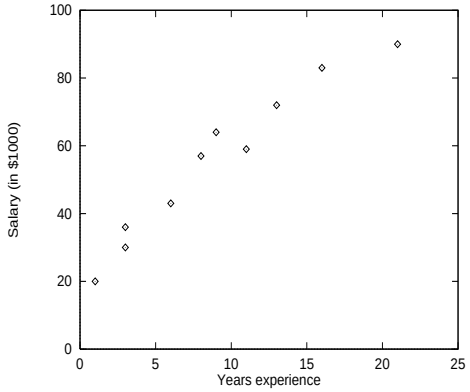


Figure 7.16: Plot of the data in Table 7.7 for Example 7.6. Although the points do not fall on a straight line, the overall pattern suggests a linear relationship between X (*years experience*) and Y (*salary*).

Example 7.6 Linear regression using the method of least squares. Table 7.7 shows a set of paired data where X is the number of years of work experience of a college graduate and Y is the corresponding salary of the graduate. A plot of the data is shown in Figure 7.16, suggesting a linear relationship between the two variables, X and Y. We model the relationship that salary may be related to the number of years of work experience with the equation $Y = \alpha + \beta X$.

Given the above data, we compute $\bar{x} = 9.1$ and $\bar{y} = 55.4$. Substituting these values into Equation (7.22), we get

$\beta = \frac{(3-9.1)(30-55.4)+(8-9.1)(57-55.4)+...+(16-9.1)(83-55.4)}{(3-9.1)^2+(8-9.1)^2+...+(16-9.1)^2} = 3.7$

$\alpha = 55.4 - (3.7)(9.1) = 21.7$

Thus, the equation of the least squares line is estimated by $Y = 21.7 + 3.7X$. Using this equation, we can predict that the salary of a college graduate with, say, 10 years of experience is $58.7K. □

Multiple regression is an extension of linear regression involving more than one predictor variable. It allows response variable Y to be modeled as a linear function of a multidimensional feature vector. An example of a multiple regression model based on two predictor attributes or variables, X_1 and X_2, is shown in Equation (7.24).

$$Y = \alpha + \beta_1 X_1 + \beta_2 X_2 \tag{7.24}$$

The method of least squares can also be applied here to solve for α, β_1, and β_2.

7.8.2 Nonlinear regression

"How can we model data that does not show a linear dependence? For example, what if a given response variable and predictor variables have a relationship that may be modeled by a polynomial function?"

Polynomial regression can be modeled by adding polynomial terms to the basic linear model. By applying transformations to the variables, we can convert the nonlinear model into a linear one that can then be solved by the method of least squares.

Example 7.7 Transformation of a polynomial regression model to a linear regression model. Consider a cubic polynomial relationship given by Equation (7.25).

$$Y = \alpha + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 \tag{7.25}$$

To convert this equation to linear form, we define new variables as shown in Equation (7.26).

$$X_1 = X \qquad X_2 = X^2 \qquad X_3 = X^3 \tag{7.26}$$

Equation (7.25) can then be converted to linear form by applying the above assignments, resulting in the equation $Y = \alpha + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3$, which is solvable by the method of least squares.

□

In Exercise 7, you are asked to find the transformations required to convert a nonlinear model involving a power function into a linear regression model.

Some models are intractably nonlinear (such as the sum of exponential terms, for example) and cannot be converted to a linear model. For such cases, it may be possible to obtain least-square estimates through extensive calculations on more complex formulae.

7.8.3 Other regression models

Linear regression is used to model continuous-valued functions. It is widely used, owing largely to its simplicity. *"Can it also be used to predict categorical labels?"* **Generalized linear models** represent the theoretical foundation on which linear regression can be applied to the modeling of categorical response variables. In generalized linear models, the variance of the response variable Y is a function of the mean value of Y, unlike in linear regression, where the variance of Y is constant. Common types of generalized linear models include **logistic regression** and **Poisson regression**. Logistic regression models the probability of some event occurring as a linear function of a set of predictor variables. Count data frequently exhibit a Poisson distribution and are commonly modeled using Poisson regression.

Log-linear models approximate *discrete* multidimensional probability distributions. They may be used to estimate the probability value associated with data cube cells. For example, suppose we are given data for the attributes *city, item, year*, and *sales*. In the log-linear method, all attributes must be categorical, hence continuous-valued attributes (like *sales*) must first be discretized. The method can then be used to estimate the probability of each cell in the 4-D base cuboid for the given attributes, based on the 2-D cuboids for *city* and *item*, *city* and *year*, *city* and *sales*, and the 3-D cuboid for *item, year*, and *sales*. In this way, an iterative technique can be used to build higher order data cubes from lower order ones. The technique scales up well to allow for many dimensions. Aside from prediction, the log-linear model is useful for data compression (since the smaller order cuboids together typically occupy less space than the base cuboid) and data smoothing (since cell estimates in the smaller order cuboids are less subject to sampling variations than cell estimates in the base cuboid).

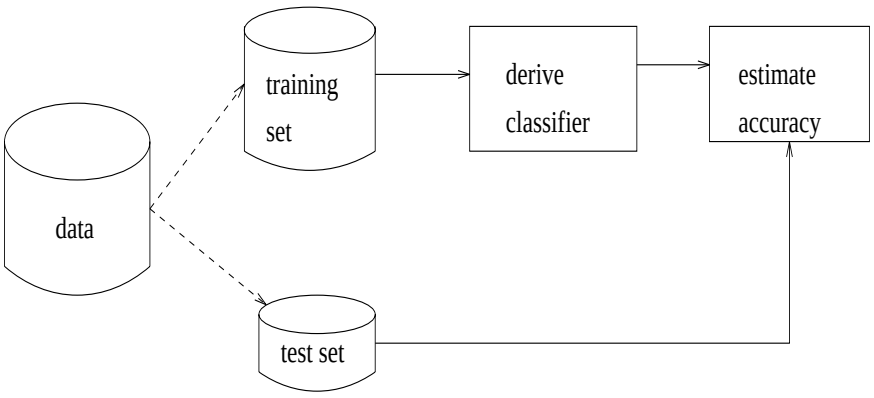


Figure 7.17: Estimating classifier accuracy with the holdout method.

7.9 Classifier accuracy

Estimating classifier accuracy is important in that it allows one to evaluate how accurately a given classifier will correctly label future data, i.e., data on which the classifier has not been trained. For example, if data from previous sales are used to train a classifier to predict customer purchasing behavior, we would like some estimate of how accurately the classifier can predict the purchasing behavior of future customers. Accuracy estimates also help in the comparison of different classifiers. In Section 7.9.1, we discuss techniques for estimating classifier accuracy, such as the *holdout* and *k-fold cross-validation* methods. Section 7.9.2 describes *bagging* and *boosting*, two strategies for increasing classifier accuracy. Section 7.9.3 discusses additional issues relating to classifier selection.

7.9.1 Estimating classifier accuracy

Using training data to derive a classifier and then to estimate the accuracy of the classifier can result in misleading over-optimistic estimates due to overspecialization of the learning algorithm (or model) to the data. Holdout and cross-validation are two common techniques for assessing classifier accuracy, based on randomly-sampled partitions of the given data.

In the **holdout** method, the given data are randomly partitioned into two independent sets, a *training set* and a *test set*. Typically, two thirds of the data are allocated to the training set, and the remaining one third is allocated to the test set. The training set is used to derive the classifier, whose accuracy is estimated with the test set (Figure 7.17). The estimate is pessimistic since only a portion of the initial data is used to derive the classifier. **Random subsampling** is a variation of the holdout method in which the holdout method is repeated k times. The overall accuracy estimate is taken as the average of the accuracies obtained from each iteration.

In **k-fold cross validation**, the initial data are randomly partitioned into k mutually exclusive subsets or "folds", $S_1, S_2, ..., S_k$, each of approximately equal size. Training and testing is performed k times. In iteration i, the subset S_i is reserved as the test set, and the remaining subsets are collectively used to train the classifier. That is, the classifier of the first iteration is trained on subsets $S_2, .., S_k$, and tested on S_1; the classifier of the section iteration is trained on subsets $S_1, S_3, .., S_k$, and tested on S_2; and so on. The accuracy estimate is the overall number of correct classifications from the k iterations, divided by the total number of samples in the initial data. In **stratified cross-validation**, the folds are stratified so that the class distribution of the samples in each fold is approximately the same as that in the initial data.

Other methods of estimating classifier accuracy include **bootstrapping**, which samples the given training instances uniformly *with replacement*, and **leave-one-out**, which is k-fold cross validation with k set to s, the number of initial samples. In general, stratified 10-fold cross-validation is recommended for estimating classifier accuracy (even if computation power allows using more folds) due to its relatively low bias and variance.

The use of such techniques to estimate classifier accuracy increases the overall computation time, yet is useful for selecting among several classifiers.

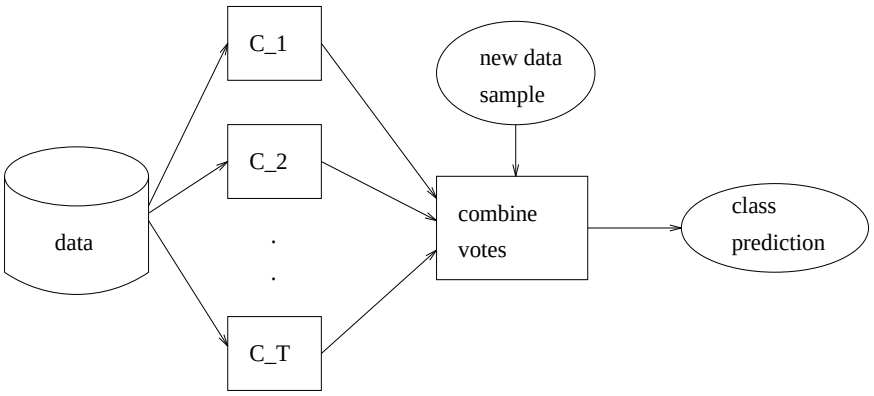


Figure 7.18: Increasing classifier accuracy: Bagging and boosting each generate a set of classifiers, $C_1, C_2, .., C_T$. Voting strategies are used to combine the class predictions for a given unknown sample.

7.9.2 Increasing classifier accuracy

In the previous section, we studied methods of estimating classifier accuracy. In Section 7.3.2, we saw how pruning can be applied to decision tree induction to help improve the accuracy of the resulting decision trees. Are there *general* techniques for improving classifier accuracy?

The answer is yes. **Bagging** (or *boostrap aggregation*) and **boosting** are two such techniques (Figure 7.18). Each combines a series of T learned classifiers, $C_1, C_2, .., C_T$, with the aim of creating an improved composite classifier, $C*$.

"How do these methods work?" Suppose that you are a patient and would like to have a diagnosis made based on your symptoms. Instead of asking one doctor, you may choose to ask several. If a certain diagnosis occurs more than the others, you may choose this as the final or best diagnosis. Now replace each doctor by a classifier, and you have the intuition behind bagging. Suppose instead, that you assign weights to the "value" or worth of each doctor's diagnosis, based on the accuracies of previous diagnoses they have made. The final diagnosis is then a combination of the weighted diagnoses. This is the essence behind boosting. Let us have a closer look at these two techniques.

Given a set S of s samples, bagging works as follows. For iteration t ($t = 1, 2, .., T$), a training set S_t is sampled with replacement from the original set of samples, S. Since sampling with replacement is used, some of the original samples of S may not be included in S_t, while others may occur more than once. A classifier C_t is learned for each training set, S_t. To classify an unknown sample, X, each classifier C_t returns its class prediction, which counts as one vote. The bagged classifier, $C*$, counts the votes and assigns the class with the most votes to X. Bagging can be applied to the prediction of continuous values by taking the average value of each vote, rather than the majority.

In boosting, weights are assigned to each training sample. A series of classifiers is learned. After a classifier C_t is learned, the weights are updated to allow the subsequent classifier, C_{t+1}, to "pay more attention" to the misclassification errors made by C_t. The final boosted classifier, $C*$, combines the votes of each individual classifier, where the weight of each classifier's vote is a function of its accuracy. The boosting algorithm can be extended for the prediction of continuous values.

7.9.3 Is accuracy enough to judge a classifier?

In addition to accuracy, classifiers can be compared with respect to their speed, robustness (e.g., accuracy on noisy data), scalability, and interpretability. Scalability can be evaluated by assessing the number of I/O operations involved for a given classification algorithm on data sets of increasingly large size. Interpretability is subjective, although we may use objective measurements such as the complexity of the resulting classifier (e.g., number of tree nodes for decision trees, or number of hidden units for neural networks, etc.) in assessing it.

"Is it always possible to assess accuracy?" In classification problems, it is commonly assumed that all objects are uniquely classifiable, i.e., that each training sample can belong to only one class. As we have discussed above, classification algorithms can then be compared according to their accuracy. However, owing to the wide diversity

of data in large databases, it is not always reasonable to assume that all objects are uniquely classifiable. Rather, it is more probable to assume that each object may belong to more than one class. How then, can the accuracy of classifiers on large databases be measured? The accuracy measure is not appropriate, since it does not take into account the possibility of samples belonging to more than one class.

Rather than returning a class label, it is useful to return a probability class distribution. Accuracy measures may then use a **second guess** heuristic whereby a class prediction is judged as correct if it agrees with the first or second most probable class. Although this does take into consideration, in some degree, the non-unique classification of objects, it is not a complete solution.

7.10 Summary

- Classification and prediction are two forms of data analysis which can be used to extract models describing important data classes or to predict future data trends. While **classification** predicts categorical labels (classes), **prediction** models continuous-valued functions.
- Preprocessing of the data in preparation for classification and prediction can involve **data cleaning** to reduce noise or handle missing values, **relevance analysis** to remove irrelevant or redundant attributes, and **data transformation**, such as generalizing the data to higher level concepts, or normalizing the data.
- Predictive accuracy, computational speed, robustness, scalability, and interpretability are five **criteria** for the evaluation of classification and prediction methods.
- **ID3** and **C4.5** are greedy algorithms for the induction of decision trees. Each algorithm uses an information theoretic measure to select the attribute tested for each non-leaf node in the tree. **Pruning** algorithms attempt to improve accuracy by removing tree branches reflecting noise in the data. Early decision tree algorithms typically assume that the data are memory resident - a limitation to data mining on large databases. Since then, several scalable algorithms have been proposed to address this issue, such as SLIQ, SPRINT, and RainForest. Decision trees can easily be converted to classification IF-THEN rules.
- **Naive Bayesian classification** and **Bayesian belief networks** are based on Bayes theorem of posterior probability. Unlike naive Bayesian classification (which assumes class conditional independence), Bayesian belief networks allow class conditional independencies to be defined between subsets of variables.
- **Backpropagation** is a neural network algorithm for classification which employs a method of gradient descent. It searches for a set of weights which can model the data so as to minimize the mean squared distance between the network's class prediction and the actual class label of data samples. Rules may be extracted from trained neural networks in order to help improve the interpretability of the learned network.
- **Association mining** techniques, which search for frequently occurring patterns in large databases, can be applied to and used for classification.
- Nearest neighbor classifiers and cased-based reasoning classifiers are **instance-based** methods of classification in that they store all of the training samples in pattern space. Hence, both require efficient indexing techniques. In **genetic algorithms**, populations of rules "evolve" via operations of crossover and mutation until all rules within a population satisfy a specified threshold. **Rough set theory** can be used to approximately define classes that are not distinguishable based on the available attributes. **Fuzzy set** approaches replace "brittle" threshold cutoffs for continuous-valued attributes with degree of membership functions.
- Linear, nonlinear, and generalized linear models of **regression** can be used for prediction. Many nonlinear problems can be converted to linear problems by performing transformations on the predictor variables.
- Data warehousing techniques, such as attribute-oriented induction and the use of multidimensional data cubes, can be integrated with classification methods in order to allow fast **multilevel mining**. Classification tasks may be specified using a data mining query language, promoting interactive data mining.
- **Stratified k-fold cross validation** is a recommended method for estimating classifier accuracy. **Bagging** and **boosting** methods can be used to increase overall classification accuracy by learning and combining a series of individual classifiers.

Exercises

1. Table 7.8 consists of training data from an employee database. The data have been generalized. For a given row entry, *count* represents the number of data tuples having the values for *department, status, age*, and *salary* given in that row.

department	*status*	*age*	*salary*	*count*
sales	senior	31-35	45-50K	30
sales	junior	26-30	25-30K	40
sales	junior	31-35	30-35K	40
systems	junior	21-25	45-50K	20
systems	senior	31-35	65-70K	5
systems	junior	26-30	45-50K	3
systems	senior	41-45	65-70K	3
marketing	senior	36-40	45-50K	10
marketing	junior	31-35	40-45K	4
secretary	senior	46-50	35-40K	4
secretary	junior	26-30	25-30K	6

Table 7.8: Generalized relation from an employee database.

Let *salary* be the class label attribute.

   (a) How would you modify the ID3 algorithm to take into consideration the *count* of each data tuple (i.e., of each row entry)?

   (b) Use your modified version of ID3 to construct a decision tree from the given data.

   (c) Given a data sample with the values *"systems"*, *"junior"*, and *"20-24"* for the attributes *department, status*, and *age*, respectively, what would a naive Bayesian classification of the *salary* for the sample be?

   (d) Design a multilayer feed-forward neural network for the given data. Label the nodes in the input and output layers.

   (e) Using the multilayer feed-forward neural network obtained above, show the weight values after one iteration of the backpropagation algorithm given the training instance *"(sales, senior, 31-35, 45-50K)"*. Indicate your initial weight values and the learning rate used.

2. Write an algorithm for k-nearest neighbor classification given k, and n, the number of attributes describing each sample.

3. What is a drawback of using a separate set of samples to evaluate pruning?

4. Given a decision tree, you have the option of (a) converting the decision tree to rules and then pruning the resulting rules, or (b) pruning the decision tree and then converting the pruned tree to rules? What advantage does (a) have over (b)?

5. ADD QUESTIONS ON OTHER CLASSIFICATION METHODS.

6. Table 7.9 shows the mid-term and final exam grades obtained for students in a database course.

   (a) Plot the data. Do X and Y seem to have a linear relationship?

   (b) Use the method of least squares to find an equation for the prediction of a student's final exam grade based on the student's mid-term grade in the course.

   (c) Predict the final exam grade of a student who received an 86 on the mid-term exam.

7. Some nonlinear regression models can be converted to linear models by applying transformations to the predictor variables. Show how the nonlinear regression equation $Y = \alpha X^{\beta}$ can be converted to a linear regression equation solvable by the method of least squares.

X *mid-term exam*	Y *final exam*
72	84
50	63
81	77
74	78
94	90
86	75
59	49
83	79
65	77
33	52
88	74
81	90

Table 7.9: Mid-term and final exam grades.

8. It is difficult to assess classification accuracy when individual data objects may belong to more than one class at a time. In such cases, comment on what criteria you would use to compare different classifiers modeled after the same data.

Bibliographic Notes

Classification from a machine learning perspective is described in several books, such as Weiss and Kulikowski [136], Michie, Spiegelhalter, and Taylor [88], Langley [67], and Mitchell [91]. Weiss and Kulikowski [136] compare classification and prediction methods from many different fields, in addition to describing practical techniques for the evaluation of classifier performance. Many of these books describe each of the basic methods of classification discussed in this chapter. Edited collections containing seminal articles on machine learning can be found in Michalksi, Carbonell, and Mitchell [85, 86], Kodratoff and Michalski [63], Shavlik and Dietterich [123], and Michalski and Tecuci [87]. For a presentation of machine learning with respect to data mining applications, see Michalski, Bratko, and Kubat [84].

The C4.5 algorithm is described in a book by J. R. Quinlan [108]. The book gives an excellent presentation of many of the issues regarding decision tree induction, as does a comprehensive survey on decision tree induction by Murthy [94]. Other algorithms for decision tree induction include the predecessor of C4.5, ID3 (Quinlan [104]), CART (Breiman et al. [11]), FACT (Loh and Vanichsetakul [76]), QUEST (Loh and Shih [75]), and PUBLIC (Rastogi and Shim [111]). Incremental versions of ID3 include ID4 (Schlimmer and Fisher [120]) and ID5 (Utgoff [132]). In addition, INFERULE (Uthurusamy, Fayyad, and Spangler [133]) learns decision trees from inconclusive data. KATE (Manago and Kodratoff [80]) learns decision trees from complex structured data. Decision tree algorithms that address the scalability issue in data mining include SLIQ (Mehta, Agrawal, and Rissanen [81]), SPRINT (Shafer, Agrawal, and Mehta [121]), RainForest (Gehrke, Ramakrishnan, and Ganti [43]), and Kamber et al. [61]. Earlier approaches described include [16, 17, 18]. For a comparison of attribute selection measures for decision tree induction, see Buntine and Niblett [15], and Murthy [94]. For a detailed discussion on such measures, see Kononenko and Hong [65].

There are numerous algorithms for decision tree pruning, including cost complexity pruning (Breiman et al. [11]), reduced error pruning (Quinlan [105]), and pessimistic pruning (Quinlan [104]). PUBLIC (Rastogi and Shim [111]) integrates decision tree construction with tree pruning. MDL-based pruning methods can be found in Quinlan and Rivest [110], Mehta, Agrawal, and Rissanen [82], and Rastogi and Shim [111]. Others methods include Niblett and Bratko [96], and Hosking, Pednault, and Sudan [55]. For an empirical comparison of pruning methods, see Mingers [89], and Malerba, Floriana, and Semeraro [79].

For the extraction of rules from decision trees, see Quinlan [105, 108]. Rather than generating rules by extracting them from decision trees, it is also possible to induce rules directly from the training data. Rule induction algorithms

include CN2 (Clark and Niblett [21]), AQ15 (Hong, Mozetic, and Michalski [54]), ITRULE (Smyth and Goodman [126]), FOIL (Quinlan [107]), and Swap-1 (Weiss and Indurkhya [134]). Decision trees, however, tend to be superior in terms of computation time and predictive accuracy. Rule refinement strategies which identify the most interesting rules among a given rule set can be found in Major and Mangano [78].

For descriptions of data warehousing and multidimensional data cubes, see Harinarayan, Rajaraman, and Ullman [48], and Berson and Smith [8], as well as Chapter 2 of this book. Attribution-oriented induction (AOI) is presented in Han and Fu [45], and summarized in Chapter 5. The integration of AOI with decision tree induction is proposed in Kamber et al. [61]. The precision or classification threshold described in Section 7.3.6 is used in Agrawal et al. [2] and Kamber et al. [61].

Thorough presentations of Bayesian classification can be found in Duda and Hart [32], a classic textbook on pattern recognition, as well as machine learning textbooks such as Weiss and Kulikowski [136] and Mitchell [91]. For an analysis of the predictive power of naive Bayesian classifiers when the class conditional independence assumption is violated, see Domingos and Pazzani [31]. Experiments with kernel density estimation for continuous-valued attributes, rather than Gaussian estimation have been reported for naive Bayesian classifiers in John [59]. Algorithms for inference on belief networks can be found in Russell and Norvig [118] and Jensen [58]. The method of gradient descent, described in Section 7.4.4 for learning Bayesian belief networks, is given in Russell et al. [117]. The example given in Figure 7.8 is adapted from Russell et al. [117]. Alternative strategies for learning belief networks with hidden variables include the EM algorithm (Lauritzen [68]), and Gibbs sampling (York and Madigan [139]). Solutions for learning the belief network structure from training data given observable variables are proposed in [22, 14, 50].

The backpropagation algorithm was presented in Rumelhart, Hinton, and Williams [115]. Since then, many variations have been proposed involving, for example, alternative error functions (Hanson and Burr [47]), dynamic adjustment of the network topology (Fahlman and Lebiere [35], Le Cun, Denker, and Solla [70]), and dynamic adjustment of the learning rate and momentum parameters (Jacobs [56]). Other variations are discussed in Chauvin and Rumelhart [19]. Books on neural networks include [116, 49, 51, 40, 19, 9, 113]. Many books on machine learning, such as [136, 91], also contain good explanations of the backpropagation algorithm. There are several techniques for extracting rules from neural networks, such as [119, 42, 131, 40, 7, 77, 25, 69]. The method of rule extraction described in Section 7.5.4 is based on Lu, Setiono, and Liu [77]. Critiques of techniques for rule extraction from neural networks can be found in Andrews, Diederich, and Tickle [5], and Craven and Shavlik [26]. An extensive survey of applications of neural networks in industry, business, and science is provided in Widrow, Rumelhart, and Lehr [137].

The method of associative classification described in Section 7.6 was proposed in Liu, Hsu, and Ma [74]. ARCS was proposed in Lent, Swami, and Widom [73], and is also described in Chapter 6.

Nearest neighbor methods are discussed in many statistical texts on classification, such as Duda and Hart [32], and James [57]. Additional information can be found in Cover and Hart [24] and Fukunaga and Hummels [41]. References on case-based reasoning (CBR) include the texts [112, 64, 71], as well as [1]. For a survey of business applications of CBR, see Allen [4]. Examples of other applications include [6, 129, 138]. For texts on genetic algorithms, see [44, 83, 90]. Rough sets were introduced in Pawlak [97, 99]. Concise summaries of rough set theory in data mining include [141, 20]. Rough sets have been used for feature reduction and expert system design in many applications, including [98, 72, 128]. Algorithms to reduce the computation intensity in finding reducts have been proposed in [114, 125]. General descriptions of fuzzy logic can be found in [140, 8, 20].

There are many good textbooks which cover the techniques of regression. Example include [57, 30, 60, 28, 52, 95, 3]. The book by Press et al. [101] and accompanying source code contain many statistical procedures, such as the method of least squares for both linear and multiple regression. Recent nonlinear regression models include projection pursuit and MARS (Friedman [39]). Log-linear models are also known in the computer science literature as *multiplicative models*. For log-linear models from a computer science perspective, see Pearl [100]. Regression trees (Breiman et al. [11]) are often comparable in performance with other regression methods, particularly when there exist many higher order dependencies among the predictor variables.

Methods for data cleaning and data transformation are discussed in Pyle [102], Kennedy et al. [62], Weiss and Indurkhya [134], and Chapter 3 of this book. Issues involved in estimating classifier accuracy are described in Weiss and Kulikowski [136]. The use of stratified 10-fold cross-validation for estimating classifier accuracy is recommended over the holdout, cross-validation, leave-one-out (Stone [127]), and bootstrapping (Efron and Tibshirani [33]) methods, based on a theoretical and empirical study by Kohavi [66]. Bagging is proposed in Breiman [10]. The boosting technique of Freund and Schapire [38] has been applied to several different classifiers, including decision tree induction (Quinlan [109]), and naive Bayesian classification (Elkan [34]).

The University of California at Irvine (UCI) maintains a Machine Learning Repository of data sets for the development and testing of classification algorithms. For information on this repository, see *http://www.ics.uci.edu/~mlearn/MLRepository.html*.

No classification method is superior over all others for all data types and domains. Empirical comparisons on classification methods include [106, 37, 135, 122, 130, 12, 23, 27, 92, 29].

Bibliography

[1] A. Aamodt and E. Plazas. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Comm.*, 7:39–52, 1994.

[2] R. Agrawal, S. Ghosh, T. Imielinski, B. Iyer, and A. Swami. An interval classifier for database mining applications. In *Proc. 18th Int. Conf. Very Large Data Bases*, pages 560–573, Vancouver, Canada, August 1992.

[3] A. Agresti. *An Introduction to Categorical Data Analysis.* John Wiley & Sons, 1996.

[4] B. P. Allen. Case-based reasoning: Business applications. *Comm. ACM*, 37:40–42, 1994.

[5] R. Andrews, J. Diederich, and A. B. Tickle. A survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-Based Systems*, 8, 1995.

[6] K. D. Ashley. *Modeling Legal Argument: Reasoning with Cases and Hypotheticals.* Cambridge, MA: MIT Press, 1990.

[7] S. Avner. Discovery of comprehensible symbolic rules in a neural network. In *Intl. Symposium on Intelligence in Neural and Bilogical Systems*, pages 64–67, 1995.

[8] A. Berson and S. J. Smith. *Data Warehousing, Data Mining, and OLAP.* New York: McGraw-Hill, 1997.

[9] C. M. Bishop. *Neural Networks for Pattern Recognition.* Oxford, UK: Oxford University Press, 1995.

[10] L. Breiman. Bagging predictors. *Machine Learning*, 24:123–140, 1996.

[11] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees.* Wadsworth International Group, 1984.

[12] C. E. Brodley and P. E. Utgoff. Multivariate versus univariate decision trees. In *Technical Report 8*, Department of Computer Science, Univ. of Massachusetts, 1992.

[13] W. Buntine. Graphical models for discovering knowledge. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 59–82. AAAI/MIT Press, 1996.

[14] W. L. Buntine. Operations for learning with graphical models. *Journal of Artificial Intelligence Research*, 2:159–225, 1994.

[15] W. L. Buntine and Tim Niblett. A further comparison of splitting rules for decision-tree induction. *Machine Learning*, 8:75–85, 1992.

[16] J. Catlett. *Megainduction: Machine Learning on Very large Databases.* PHD Thesis, University of Sydney, 1991.

[17] P. K. Chan and S. J. Stolfo. Experiments on multistrategy learning by metalearning. In *Proc. 2nd. Int. Conf. Information and Knowledge Management*, pages 314–323, 1993.

[18] P. K. Chan and S. J. Stolfo. Metalearning for multistrategy and parallel learning. In *Proc. 2nd. Int. Workshop on Multistrategy Learning*, pages 150–165, 1993.

[19] Y. Chauvin and D. Rumelhart. *Backpropagation: Theory, Architectures, and Applications.* Hillsdale, NJ: Lawrence Erlbaum Assoc., 1995.

[20] K. Cios, W. Pedrycz, and R. Swiniarski. *Data Mining Methods for Knowledge Discovery.* Boston: Kluwer Academic Publishers, 1998.

[21] P. Clark and T. Niblett. The CN2 induction algorithm. *Machine Learning*, 3:261–283, 1989.

[22] G. Cooper and E. Herskovits. A bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9:309–347, 1992.

[23] V. Corruble, D. E. Brown, and C. L. Pittard. A comparison of decision classifiers with backpropagation neural networks for multimodal classification problems. *Patern Recognition*, 26:953–961, 1993.

[24] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Trans. Information Theory*, 13:21–27, 1967.

[25] M. W. Craven and J. W. Shavlik. Extracting tree-structured representations of trained networks. In D. Touretzky and M. Mozer M. Hasselmo, editors, *Advances in Neural Information Processing Systems.* Cambridge, MA: MIT Press, 1996.

[26] M. W. Craven and J. W. Shavlik. Using neural networks in data mining. *Future Generation Computer Systems*, 13:211–229, 1997.

[27] S. P. Curram and J. Mingers. Neural networks, decision tree induction and discriminant analysis: An empirical comparison. *J. Operational Research Society*, 45:440–450, 1994.

[28] J. L. Devore. *Probability and Statistics for Engineering and the Science, 4th ed.* Duxbury Press, 1995.

[29] T. G. Dietterich, H. Hild, and G. Bakiri. A comparison of ID3 and backpropagation for english text-to-speech mapping. *Machine Learning*, 18:51–80, 1995.

[30] A. J. Dobson. *An Introduction to Generalized Linear Models.* Chapman and Hall, 1990.

[31] P. Domingos and M. Pazzani. Beyond independence: Conditions for the optimality of the simple bayesian classifier. In *Proc. 13th Intl. Conf. Machine Learning*, pages 105–112, 1996.

[32] R. Duda and P. Hart. *Pattern Classification and Scene Analysis.* Wiley: New York, 1973.

[33] B. Efron and R. Tibshirani. *An Introduction to the Bootstrap.* New York: Chapman & Hall, 1993.

[34] C. Elkan. Boosting and naive bayesian learning. In *Technical Report CS97-557*, Dept. of Computer Science and Engineering, Univ. Calif. at San Diego, Sept. 1997.

[35] S. Fahlman and C. Lebiere. The cascade-correlation learning algorithm. In *Technical Report CMU-CS-90-100*, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, 1990.

[36] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy (eds.). *Advances in Knowledge Discovery and Data Mining.* AAAI/MIT Press, 1996.

[37] D. H. Fisher and K. B. McKusick. An empirical comparison of ID3 and back-propagation. In *Proc. 11th Intl. Joint Conf. AI*, pages 788–793, San Mateo, CA: Morgan Kaufmann, 1989.

[38] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55:119–139, 1997.

[39] J. Friedman. Multivariate adaptive regression. *Annalsof Statistics*, 19:1–141, 1991.

[40] L. Fu. *Neural Networks in Computer Intelligence.* McGraw-Hill, 1994.

[41] K. Fukunaga and D. Hummels. Bayes error estimation using parzen and k-nn procedure. In *IEEE Trans. Pattern Analysis and Machine Learning*, pages 634–643, 1987.

[42] S. I. Gallant. *Neural Network Learning and Expert Systems.* Cambridge, MA: MIT Press, 1993.

[43] J. Gehrke, R. Ramakrishnan, and V. Ganti. Rainforest: A framework for fast decision tree construction of large datasets. In *Proc. 1998 Int. Conf. Very Large Data Bases*, pages 416–427, New York, NY, August 1998.

[44] D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley: Reading, MA, 1989.

[45] J. Han and Y. Fu. Exploration of the power of attribute-oriented induction in data mining. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 399–421. AAAI/MIT Press, 1996.

[46] J. Han, Y. Fu, W. Wang, J. Chiang, W. Gong, K. Koperski, D. Li, Y. Lu, A. Rajan, N. Stefanovic, B. Xia, and O. R. Zaïane. DBMiner: A system for mining knowledge in large relational databases. In *Proc. 1996 Int. Conf. Data Mining and Knowledge Discovery (KDD'96)*, pages 250–255, Portland, Oregon, August 1996.

[47] S. J. Hanson and D. J. Burr. Minkowski back-propagation: Learning in connectionist models with non-euclidean error signals. In *Neural Information Processing Systems*, American Institute of Physics, 1988.

[48] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data*, pages 205–216, Montreal, Canada, June 1996.

[49] R. Hecht-Nielsen. *Neurocomputing.* Reading, MA: Addison Wesley, 1990.

[50] D. Heckerman, D. Geiger, and D. Chickering. Learning bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20:197, 1995.

[51] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the Theory of Neural Computation.* Addison Wesley: Reading, MA., 1991.

[52] R. V. Hogg and A. T. Craig. *Introduction to Mathematical Statistics, 5th ed.* Prentice Hall, 1995.

[53] L. B. Holder. Intermediate decision trees. In *Proc. 14th Int. Joint Conf. Artificial Intelligence (IJCAI95)*, pages 1056–1062, Montreal, Canada, Aug. 1995.

[54] J. Hong, I. Mozetic, and R. S. Michalski. AQ15: Incremental learning of attribute-based descriptions from examples, the method and user's guide. In *Report ISG 85-5, UIUCDCS-F-86-949,*, Department of Computer Science, University of Illinois at Urbana-Champagin, 1986.

[55] J. Hosking, E. Pednault, and M. Sudan. A statistical perspective on data mining. In *Future Generation Computer Systems*, pages 117–134, ???, 1997.

[56] R. Jacobs. Increased rates of convergence through learning rate adaptation. *Neural Networks*, 1:295–307, 1988.

[57] M. James. *Classification Algorithms.* John Wiley, 1985.

[58] F. V. Jensen. *An Introduction to Bayesian Networks.* Springer Verlag, 1996.

[59] G. H. John. *Enhancements to the Data Mining Process.* Ph.D. Thesis, Computer Science Dept., Stanford Univeristy, 1997.

[60] R. A. Johnson and D. W. Wickern. *Applied Multivariate Statistical Analysis, 3rd ed.* Prentice Hall, 1992.

[61] M. Kamber, L. Winstone, W. Gong, S. Cheng, and J. Han. Generalization and decision tree induction: Efficient classification in data mining. In *Proc. 1997 Int. Workshop Research Issues on Data Engineering (RIDE'97)*, pages 111–120, Birmingham, England, April 1997.

[62] R. L Kennedy, Y. Lee, B. Van Roy, C. D. Reed, and R. P. Lippman. *Solving Data Mining Problems Through Pattern Recognition.* Upper Saddle River, NJ: Prentice Hall, 1998.

[63] Y. Kodratoff and R. S. Michalski. *Machine Learning, An Artificial Intelligence Approach, Vol. 3.* Morgan Kaufmann, 1990.

[64] J. L. Kolodner. *Case-Based Reasoning*. Morgan Kaufmann, 1993.

[65] I. Kononenko and S. J. Hong. Attribute selection for modeling. In *Future Generation Computer Systems*, pages 181–195, ???, 1997.

[66] K. Koperski and J. Han. Discovery of spatial association rules in geographic information databases. In *Proc. 4th Int. Symp. Large Spatial Databases (SSD'95)*, pages 47–66, Portland, Maine, Aug. 1995.

[67] P. Langley. *Elements of Machine Learning*. Morgan Kaufmann, 1996.

[68] S. L. Lauritzen. The EM algorithm for graphical association models with missing data. *Computational Statistics and Data Analysis*, 19:191–201, 1995.

[69] S. Lawrence, C. L Giles, and A. C. Tsoi. Symbolic conversion, grammatical inference and rule extraction for foreign exchange rate prediction. In Y. Abu-Mostafa and A. S. Weigend P. N Refenes, editors, *Neural Networks in the Captial Markets*. Singapore: World Scientific, 1997.

[70] Y. Le Cun, J. S. Denker, and S. A. Solla. Optimal brain damage. In D. Touretzky, editor, *Advances in neural Information Processing Systems, 2*, pages San Mateo, CA: Morgan Kaufmann. Cambridge, MA: MIT Press, 1990.

[71] D. B. Leake. CBR in context: The present and future. In D. B. Leake, editor, *Cased-Based Reasoning: Experience, Lessons, and Future Directions*, pages 3–30. Menlo Park, CA: AAAI Press, 1996.

[72] A. Lenarcik and Z. Piasta. Probabilistic rough classifiers with mixture of discrete and continuous variables. In T. Y. Lin N. Cercone, editor, *Rough Sets and Data Mining: Analysis for Imprecise Data*, pages 373–383. Boston: Kluwer, 1997.

[73] B. Lent, A. Swami, and J. Widom. Clustering association rules. In *Proc. 1997 Int. Conf. Data Engineering (ICDE'97)*, pages 220–231, Birmingham, England, April 1997.

[74] B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. In *Proc. 1998 Int. Conf. Knowledge Discovery and Data Mining (KDD'98)*, pages 80–86, New York, NY, August 1998.

[75] W. Y. Loh and Y.S. Shih. Split selection methods for classification trees. *Statistica Sinica*, 7:815–840, 1997.

[76] W. Y. Loh and N. Vanichsetakul. Tree-structured classificaiton via generalized discriminant analysis. *Journal of the American Statistical Association*, 83:715–728, 1988.

[77] H. Lu, R. Setiono, and H. Liu. Neurorule: A connectionist approach to data mining. In *Proc. 21st Int. Conf. Very Large Data Bases*, pages 478–489, Zurich, Switzerland, Sept. 1995.

[78] J. Major and J. Mangano. Selecting among rules induced from a hurricane database. *Journal of Intelligent Information Systems*, 4:39–52, 1995.

[79] D. Malerba, E. Floriana, and G. Semeraro. A further comparison of simplification methods for decision tree induction. In D.Fisher H. Lenz, editor, *Learning from Data: AI and Statistics*. Springer-Verlag, 1995.

[80] M. Manago and Y. Kodratoff. Induction of decision trees from complex structured data. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 289–306. AAAI/MIT Press, 1991.

[81] M. Mehta, R. Agrawal, and J. Rissanen. SLIQ: A fast scalable classifier for data mining. In *Proc. 1996 Int. Conf. Extending Database Technology (EDBT'96)*, Avignon, France, March 1996.

[82] M. Mehta, J. Rissanen, and R. Agrawal. MDL-based decision tree pruning. In *Proc. 1st Intl. Conf. Knowledge Discovery and Data Mining (KDD95)*, Montreal, Canada, Aug. 1995.

[83] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer Verlag, 1992.

[84] R. S. Michalski, I. Bratko, and M. Kubat. *Machine Learning and Data Mining: Methods and Applications*. John Wiley & Sons, 1998.

[85] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell. *Machine Learning, An Artificial Intelligence Approach, Vol. 1.* Morgan Kaufmann, 1983.

[86] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell. *Machine Learning, An Artificial Intelligence Approach, Vol. 2.* Morgan Kaufmann, 1986.

[87] R. S. Michalski and G. Tecuci. *Machine Learning, A Multistrategy Approach, Vol. 4.* Morgan Kaufmann, 1994.

[88] D. Michie, D. J. Spiegelhalter, and C. C. Taylor. *Machine Learning, Neural and Statistical Classification.* Ellis Horwood, 1994.

[89] J. Mingers. An empirical comparison of pruning methods for decision-tree induction. *Machine Learning*, 4:227–243, 1989.

[90] M. Mitchell. *An Introduction to Genetic Algorithms.* MIT Press, 1996.

[91] T. M. Mitchell. *Machine Learning.* McGraw Hill, 1997.

[92] D. Mitchie, D. J. Spiegelhalter, and C. C. Taylor. *Machine Learning, Neural and Statistical Classification.* New York: Ellis Horwood, 1994.

[93] R. Mooney, J. Shavlik, G. Towell, and A. Grove. An experimental comparison of symbolic and connectionist learning algorithms. In *Proc. 11th Int. Joint Conf. on Artificial Intelligence (IJCAI'89)*, pages 775–787, Detroit, MI, Aug. 1989.

[94] S. K. Murthy. Automatic construction of decision trees from data: A multi-disciplinary survey. *Data Mining and Knowledge Discovery*, 2:345–389, 1998.

[95] J. Neter, M. H. Kutner, C. J. Nachtsheim, and L. Wasserman. *Applied Linear Statistical Models, 4th ed.* Irwin: Chicago, 1996.

[96] T. Niblett and I. Bratko. Learning decision rules in noisy domains. In M. A. Bramer, editor, *Expert Systems '86: Research and Development in Expert Systems III*, pages 25–34. British Computer Society Specialist Group on Expert Systems, Dec. 1986.

[97] Z. Pawlak. Rough sets. *Intl. J. Computer and Information Sciences*, 11:341–356, 1982.

[98] Z. Pawlak. On learning - rough set approach. In *Lecture Notes 208*, pages 197–227, New York: Springer-Verlag, 1986.

[99] Z. Pawlak. *Rough Sets, Theoretical Aspects of Reasonign about Data.* Boston: Kluwer, 1991.

[100] J. Pearl. *Probabilistic Reasoning in Intelligent Systems.* Palo Alto, CA: Morgan Kauffman, 1988.

[101] W. H. Press, S. A. Teukolsky, V. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C, The Art of Scientific Computing.* Cambridge, MA: Cambridge University Press, 1996.

[102] D. Pyle. *Data Preparation for Data Mining.* Morgan Kaufmann, 1999.

[103] J. R. Quinlan. The effect of noise on concept learning. In Michalski et al., editor, *Machine Learning: An Artificial Intelligence Approach, Vol. 2*, pages 149–166. Morgan Kaufmann, 1986.

[104] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[105] J. R. Quinlan. Simplifying decision trees. *Internation Journal of Man-Machine Studies*, 27:221–234, 1987.

[106] J. R. Quinlan. An empirical comparison of genetic and decision-tree classifiers. In *Proc. 5th Intl. Conf. Machine Learning*, pages 135–141, San Mateo, CA: Morgan Kaufmann, 1988.

[107] J. R. Quinlan. Learning logic definitions from relations. *Machine Learning*, 5:139–166, 1990.

[108] J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.

[109] J. R. Quinlan. Bagging, boosting, and C4.5. In *Proc. 13th Natl. Conf. on Artificial Intelligence (AAAI'96)*, volume 1, pages 725–730, Portland, OR, Aug. 1996.

[110] J. R. Quinlan and R. L. Rivest. Inferring decision trees using the minimum description length principle. *Information and Computation*, 80:227–248, March 1989.

[111] R. Rastogi and K. Shim. Public: A decision tree classifer that integrates building and pruning. In *Proc. 1998 Int. Conf. Very Large Data Bases*, pages 404–415, New York, NY, August 1998.

[112] C. Riesbeck and R. Schank. *Inside Case-Based Reasoning.* Hillsdale, NJ: Lawrence Erlbaum, 1989.

[113] B. D. Ripley. *Pattern Recognition and Neural Networks.* Cambridge: Cambridge University Press, 1996.

[114] S. Romanski. Operations on families of sets for exhaustive search, given a monotonic function. In *Proc. 3rd Intl. Conf on Data and Knowledge Bases, C. Beeri et. al. (eds.),*, pages 310–322, Jerusalem, Israel, 1988.

[115] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart J. L. McClelland, editor, *Parallel Distributed Processing.* Cambridge, MA: MIT Press, 1986.

[116] D. E. Rumelhart and J. L. McClelland. *Parallel Distributed Processing.* Cambridge, MA: MIT Press, 1986.

[117] S. Russell, J. Binder, D. Koller, and K. Kanazawa. Local learning in probabilistic networks with hidden variables. In *Proc. 14th Joint Int. Conf. on Artificial Intelligence (IJCAI'95)*, volume 2, pages 1146–1152, Montreal, Canada, Aug. 1995.

[118] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice-Hall, 1995.

[119] K. Saito and R. Nakano. Medical diagnostic expert system based on PDP model. In *Proc. IEEE International Conf. on Neural Networks*, volume 1, pages 225–262, San Mateo, CA, 1988.

[120] J. C. Schlimmer and D. Fisher. A case study of incremental concept induction. In *Proc. 5th Natl. Conf. Artificial Intelligence*, pages 496–501, Phildelphia, PA: Morgan Kaufmann, 1986.

[121] J. Shafer, R. Agrawal, and M. Mehta. SPRINT: A scalable parallel classifier for data mining. In *Proc. 1996 Int. Conf. Very Large Data Bases*, pages 544–555, Bombay, India, Sept. 1996.

[122] J. W. Shavlik, R. J. Mooney, and G. G. Towell. Symbolic and neural learning algorithms: An experimental comparison. *Machine Learning*, 6:111–144, 1991.

[123] J.W. Shavlik and T.G. Dietterich. *Readings in Machine Learning.* Morgan Kaufmann, 1990.

[124] A. Skowron, L. Polowski, and J. Komorowski. Learning tolerance relations by boolean descriptors: Automatic feature extraction from data tables. In *Proc. 4th Intl. Workshop on Rough Sets, Fuzzy Sets, and Machine Discovery, S. Tsumoto et. al. (eds.)*, pages 11–17, University of Tokyo, 1996.

[125] A. Skowron and C. Rauszer. The discernibility matrices and functions in information systems. In R. Slowinski, editor, *Intelligent Decision Support, Handbook of Applications and Advances of the Rough Set Theory*, pages 331–362. Boston: Kluwer, 1992.

[126] P. Smyth and R.M. Goodman. Rule induction using information theory. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 159–176. AAAI/MIT Press, 1991.

[127] M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society*, 36:111–147, 1974.

[128] R. Swiniarski. Rough sets and principal component analysis and their applications in feature extraction and seletion, data model building and classification. In S. Pal A. Skowron, editor, *Fuzzy Sets, Rough Sets and Decision Making Processes.* New York: Springer-Verlag, 1998.

[129] K. Sycara, R. Guttal, J. Koning, S. Narasimhan, and D. Navinchandra. CADET: A case-based synthesis tool for engineering design. *Int. Journal of Expert Systems*, 4:157–188, 1992.

[130] S. B. Thrun et al. The monk's problems: A performance comparison of different learning algorithms. In *Technical Report CMU-CS-91-197*, Department of Computer Science, Carnegie Mellon Univ., Pittsburgh, PA, 1991.

[131] G. G. Towell and J. W. Shavlik. Extracting refined rules from knowledge-based neural networks. *Machine Learning*, 13:71–101, Oct. 1993.

[132] P. E. Utgoff. An incremental ID3. In *Proc. Fifth Int. Conf. Machine Learning*, pages 107–120, San Mateo, California, 1988.

[133] R. Uthurusamy, U. M. Fayyad, and S. Spangler. Learning useful rules from inconclusive data. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 141–157. AAAI/MIT Press, 1991.

[134] S. M. Weiss and N. Indurkhya. *Predictive Data Mining.* Morgan Kaufmann, 1998.

[135] S. M. Weiss and I. Kapouleas. An empirical comparison of pattern recognition, neural nets, and machine learning classification methods. In *Proc. 11th Int. Joint Conf. Artificial Intelligence*, pages 781–787, Detroit, MI, Aug. 1989.

[136] S. M. Weiss and C. A. Kulikowski. *Computer Systems that Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems.* Morgan Kaufman, 1991.

[137] B. Widrow, D. E. Rumelhart, and M. A. Lehr. Neural networks: Applications in industry, business and science. *Comm. ACM*, 37:93–105, 1994.

[138] K. D. Wilson. Chemreg: Using case-based reasoning to support health and safety compliance in the chemical industry. *AI Magazine*, 19:47–57, 1998.

[139] J. York and D. Madigan. Markov chaine monte carlo methods for hierarchical bayesian expert systems. In *Cheesman and Oldford*, pages 445–452, 1994.

[140] L. A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

[141] W. Ziarko. The discovery, analysis, and representation of data dependencies in databases. In G. Piatetsky-Shapiro W. J. Frawley, editor, *Knowledge Discovery in Databases*, pages 195–209. AAAI Press, 1991.

Contents

Chapter 8

Cluster Analysis

8.1 What is cluster analysis?

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

Cluster analysis is an important human activity. Early in childhood, one learns how to distinguish between cats and dogs, or between animals and plants, by continuously improving subconscious classification schemes. Cluster analysis has been widely used in numerous applications, including pattern recognition, data analysis, image processing, market research, etc. By clustering, one can identify crowded and sparse regions, and therefore, discover overall distribution patterns and interesting correlations among data attributes. In business, clustering may help marketers discover distinct groups in their customer bases and characterize customer groups based on purchasing patterns. In biology, it can be used to derive plant and animal taxonomies, categorize genes with similar functionality, and gain insight into structures inherent in populations. Clustering may also help in the identification of areas of similar land use in an earth observation database, and in the identification of groups of motor insurance policy holders with a high average claim cost, as well as the identification of groups of houses in a city according to house type, value, and geographical location. It may also help classify documents on the WWW for information discovery. As a data mining function, cluster analysis can be used as a stand-alone tool to gain insight into the distribution of data, to observe the characteristics of each cluster, and to focus on a particular set of clusters for further analysis. Alternatively, it may serve as a preprocessing step for other algorithms, such as classification and characterization, operating on the detected clusters.

Data clustering is a young scientific discipline under vigorous development. There are a large number of research papers scattered in many conference proceedings and periodicals, mostly in the fields of data mining, statistics, machine learning, spatial database, biology, marketing, and so on, with different emphases and different techniques. Owing to the huge amounts of data collected in databases, cluster analysis has recently become a highly active topic in data mining research.

As a branch of statistics, cluster analysis has been studied extensively for many years, focussing mainly on distance-based cluster analysis. Cluster analysis tools based on k-means, k-medoids, and several other methods have also been built in many statistical analysis software packages or systems, such as S-Plus, SPSS, and SAS.

In machine learning, cluster analysis often refers to **unsupervised learning**. Unlike classification, clustering does not rely on predefined classes and class-labeled training examples. For this reason, it is a form of **learning by observation**, rather than *learning by examples*. In **conceptual clustering**, a group of objects forms a class only if it is describable by a concept. This differs from conventional clustering which measures similarity based on geometric distance. Conceptual clustering consists of two components: (1) it discovers the appropriate classes, and (2) it forms descriptions for each class, as in classification. The guideline of striving for high intraclass similarity and low interclass similarity still applies.

In data mining, people have been studying methods for efficient and effective cluster analysis in *large databases*.

Active themes of research focus on the *scalability* of clustering methods, the effectiveness of methods for clustering *complex shapes and types of data*, *high-dimensional* clustering techniques, and methods for clustering *mixed numerical and categorical data* in large databases.

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

1. **Scalability**: Many clustering algorithms work well in small data sets containing less than 200 data objects; however, a large database may contain millions of objects. Clustering on a *sample* of a given large data set may lead to biased results. How can we develop clustering algorithms that are highly scalable in large databases?

2. **Ability to deal with different types of attributes**: Many algorithms are designed to cluster interval-based (numerical) data. However, many applications may require clustering other types of data, such as binary, categorical (nominal), and ordinal data, or mixtures of these data types.

3. **Discovery of clusters with arbitrary shape**: Many clustering algorithms determine clusters based on Euclidean or Manhattan distance measures. Algorithms based on such distance measures tend to find spherical clusters with similar size and density. However, a cluster could be of any shape. It is important to develop algorithms which can detect clusters of arbitrary shape.

4. **Minimal requirements for domain knowledge to determine input parameters**: Many clustering algorithms require users to input certain parameters in cluster analysis (such as the number of desired clusters). The clustering results are often quite sensitive to input parameters. Many parameters are hard to determine, especially for data sets containing high-dimensional objects. This not only burdens users, but also makes the quality of clustering difficult to control.

5. **Ability to deal with noisy data**: Most real-world databases contain outliers or missing, unknown, or erroneous data. Some clustering algorithms are sensitive to such data and may lead to clusters of poor quality.

6. **Insensitivity to the order of input records**: Some clustering algorithms are sensitive to the order of input data, e.g., the same set of data, when presented with different orderings to such an algorithm, may generate dramatically different clusters. It is important to develop algorithms which are insensitive to the order of input.

7. **High dimensionality**: A database or a data warehouse may contain several dimensions or attributes. Many clustering algorithms are good at handling low dimensional data, involving only two to three dimensions. Human eyes are good at judging the quality of clustering for up to three dimensions. It is challenging to cluster data objects in high dimensional space, especially considering that data in high dimensional space can be very sparse and highly skewed.

8. **Constraint-based clustering**: Real-world applications may need to perform clustering under various kinds of constraints. Suppose that your job is to choose the locations for a given number of new automatic cash stations (ATMs) in a city. To decide upon this, you may cluster households while considering the city's rivers and highway networks, and customer requirements per region as constraints. A challenging task is to find groups of data with good clustering behavior and satisfying various constraints.

9. **Interpretability and usability**: Users may expect clustering results to be interpretable, comprehensible, and usable. That is, clustering may need to be tied up with specific semantic interpretations and applications. It is important to study how an application goal may influence the selection of clustering methods.

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

8.2 Types of data in clustering analysis

In this section, we study the types of data which ofter occur in clustering analysis and how to preprocess them for such an analysis. Suppose that a data set to be clustered contains n objects which may represent persons, houses,

documents, countries, etc. Main memory-based clustering algorithms typically operate on either of the following two data structures.

1. **Data matrix** (or *object-by-variable structure*): This represents n objects, such as persons, with p **variables** (also called *measurements* or *attributes*), such as age, height, weight, gender, race, and so on. The structure is in the form of relational table, or n-by-p matrix (n objects $\times$ p variables), as shown in (8.1).

$$\begin{bmatrix} x_{11} & \cdots & x_{1f} & \cdots & x_{1p} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{i1} & \cdots & x_{if} & \cdots & x_{ip} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{n1} & \cdots & x_{nf} & \cdots & x_{np} \end{bmatrix} \tag{8.1}$$

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

$$\begin{bmatrix} 0 & & & & \\ d(2,1) & 0 & & & \\ d(3,1) & d(3,2) & 0 & & \\ \vdots & \vdots & \vdots & & \\ d(n,1) & d(n,2) & \cdots & \cdots & 0 \end{bmatrix} \tag{8.2}$$

where $d(i,j)$ is the measured **difference** or **dissimilarity** between objects i and j. Since $d(i,j) = d(j,i)$, and $d(i,i) = 0$, we have the matrix in (8.2). Measures of dissimilarity are discussed throughout this section.

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

Clusters of objects are computed based on their similarities or dissimilarities. In this section, we first discuss how the quality of clustering can be assessed based on *correlation coefficients*, which can be converted to *dissimilarity coefficients* or *similarity coefficients*. We then discuss how the dissimilarity of objects can be computed for objects described by *interval-scaled* variables, by *binary* variables, by *nominal, ordinal*, and *ratio-scaled* variables, or combinations of these variable types.

8.2.1 Dissimilarities and similarities: Measuring the quality of clustering

Measures of *dissimilarity* or *dissimilarity coefficients* can be used to measure the quality of clustering. In general, **dissimilarity** $d(i,j)$ is a nonnegative number that is close to 0 when i and j are near each other, and becomes larger when they are more different.

Dissimilarities can be obtained by simple subjective ratings made by a set of observers or experts on how much certain objects differ from each other. For example, in social science, one may rate how close one subject, such as mathematics, is to another, such as biology, computer science, or physics. Alternatively, dissimilarities can be computed from correlation coefficients. Given n objects for clustering, the parametric **Pearson product-moment correlation** between two variables f and g is defined in (8.3), where f and g are variables describing the objects, m_f and m_g are the mean values of f and g, respectively, and x_{if} is the value of f for the ith object, and x_{ig} is the value of g for the ith object.

$$R(f,g) = \frac{\Sigma_{i=1}^{n}(x_{if} - m_f)(x_{ig} - m_g)}{\sqrt{\Sigma_{i=1}^{n}(x_{if} - m_f)^2}\sqrt{\Sigma_{i=1}^{n}(x_{ig} - m_g)^2}}. \tag{8.3}$$

The nonparametric *Spearman correlation* can also be used in some applications. Notice that R is used in statistics for multiple correlation or a matrix of correlations, while r is used for a correlation. Both coefficients are provided by most statistical packages, such as SPSS, SAS, and SPlus.

Conversion formula (8.4) can be used to compute dissimilarity coefficients, $d(f, g)$, from either parametric or nonparametric correlation coefficients, $R(f, g)$:

$$d(f, g) = (1 - R(f, g))/2. \tag{8.4}$$

Given variables with a high positive correlation, this assigns a dissimilarity coefficient that is close to zero. Variables with a strong negative correlation are assigned a dissimilarity coefficient that is close to 1 (meaning that the variables are very dissimilar).

In some applications, users may prefer to use conversion formula (8.5), where variables with a high positive or negative correlation are assigned the same high similarity value.

$$d(f, g) = 1 - |R(f, g)| \tag{8.5}$$

Alternatively, users may like to use a *similarity coefficient* $s(i, j)$ instead of a dissimilarity coefficient. Formula (8.6) can be used to transform between the two coefficients.

$$s(i, j) = 1 - d(i, j) \tag{8.6}$$

Notice that not all of the variables should be included in the clustering analysis. A variable that is meaningless to a given clustering is worse than useless, since it hides the useful information provided by other variables. For example, the telephone number of a person is often useless in the clustering of people according to their characteristics, such as age, height, weight, and so on. Such kind of "trash" variable should be given zero weight, excluding it from the clustering process.

8.2.2 Interval-scaled variables

This section discusses *interval-scaled variables* and their standardization. It then describes common distance measures used for computing the dissimilarity of objects described by interval-scaled variables. These measures include the *Euclidean, Manhattan*, and *Minkowski distances*.

Interval-scaled variables are continuous measurements of a roughly linear scale. Typical examples include weight and height, latitude and longitude coordinates (e.g., when clustering houses), and weather temperature.

The measurement unit used can affect the clustering analysis. For example, changing measurement units, such as changing from meters to inches for height, or from kilograms to pounds for weight, may lead to a very different clustering structure. In general, expressing a variable in smaller units will lead to a larger range for that variable, and thus a larger effect on the resulting clustering structure. To help avoid dependence on the choice of measurement units, the data should be standardized. Standardizing measurements attempts to give all variables an equal weight. This is particularly useful when given no prior knowledge of the data. However, in some applications, people may intentionally want to give more weight to a certain set of variables than others. For example, when clustering basketball player candidates, one may like to give more weight to the variable height.

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

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

$$s_f = \frac{1}{n}(|x_{1f} - m_f| + |x_{2f} - m_f| + \cdots + |x_{nf} - m_f|) \tag{8.7}$$

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

2. Calculate the **standardized measurement**, called **z-score**, as follows,

$$z_{if} = \frac{x_{if} - m_f}{s_f}. \tag{8.8}$$

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

Standardization may or may not be useful in a particular application. Thus the choice of whether and how to perform standardization should be left to users.

After standardization, or without standardization in certain applications, we compute the dissimilarity (or similarity) between the objects. Given interval-scaled variables, this is typically based on the distance between each pair of objects. There are a few approaches to defining the distance between objects. The most popular distance measure is **Euclidean distance**, which is defined as

$$d(i,j) = \sqrt{(|x_{i1} - x_{j1}|^2 + |x_{i2} - x_{j2}|^2 + \cdots + |x_{ip} - x_{jp}|^2}. \tag{8.9}$$

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

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

$$d(i,j) = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + \cdots + |x_{ip} - x_{jp}|. \tag{8.10}$$

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

1. $d(i,j) \geq 0$: This states that distance is a nonnegative number;
2. $d(i,i) = 0$: This states that the distance of an object to itself is 0;
3. $d(i,j) = d(j,i)$: This states that distance is a symmetric function; and
4. $d(i,j) \leq d(i,h) + d(h,j)$: This is a *triangular inequality* which states that going directly from point i to point j is no more than making a detour over any other point h.

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

$$d(i,j) = (|x_{i1} - x_{j1}|^q + |x_{i2} - x_{j2}|^q + \cdots + |x_{ip} - x_{jp}|^q)^{1/q}, \tag{8.11}$$

where q is a positive integer. It represents the Manhattan distance when $q = 1$, and Euclidean distance when $q = 2$.

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

$$d(i,j) = \sqrt{w_1|x_{i1} - x_{j1}|^2 + w_2|x_{i2} - x_{j2}|^2 + \cdots + w_p|x_{ip} - x_{jp}|^2}. \tag{8.12}$$

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

8.2.3 Binary variables

This section describes how to compute the dissimilarity between objects described by either *symmetric* or *asymmetric binary variables*.

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

One approach is to compute a dissimilarity matrix from the given binary data. If all binary variables are thought of as having the same weight, we have a 2-by-2 contingency table, Table 8.1, where a is the number of variables that equal 1 for both objects i and j, b is the number of variables that equal 1 for object i but that are 0 for object j, c is the number of variables that equal 0 for object i but equal 1 for object j, and d is the number of variables that equal 0 for both objects i and j. The total number of variables is p, where $p = a + b + c + d$.

A binary variable is **symmetric** if both of its states are equally valuable and carry the same weight, that is, there is no preference on which outcome should be coded as 0 or 1. One such example could be *male* and *female* in *gender*. Similarity based on symmetric binary variables is called **invariant similarity** in that the result does not

object i \ object j	1	0	sum
1	a	b	$a+b$
0	c	d	$c+d$
sum	$a+c$	$b+d$	p

Table 8.1: A contingency table for binary variables.

change when some or all of the binary variables are coded differently. For invariant similarities, the most well-known coefficient is the **simple matching coefficient**, defined in (8.13).

$$d(i,j) = \frac{b+c}{a+b+c+d} \tag{8.13}$$

A binary variable is **asymmetric** if the outcomes of the states are not equally important, such as the *positive* and *negative* outcomes of a disease *test*. By convention, we shall code the most important outcome, which is usually the rarest one, by 1 (e.g., *HIV positive*), and the other by 0 (e.g., *HIV negative*). The agreement of two 1's (a positive match) is then considered more significant than that of two zeros (a negative match). Such a "binary" variable can actually be considered as a "unary variable". The similarity based on such variables is called **noninvariant similarity**. For noninvariant similarities, the most well-known coefficient is the **Jaccard coefficient**, defined in (8.14), where the number of negative matches, d, is considered unimportant and thus is ignored in the computation.

$$d(i,j) = \frac{b+c}{a+b+c} \tag{8.14}$$

When both symmetric and asymmetric binary variables occur in the same data set, the mixed variables approach described in Section 8.2.5 can be applied.

Example 8.1 Dissimilarity between binary variables. Suppose that a patient record table, Table 8.2, contains the attributes *name, gender, fever, cough, test-1, test-2, test-3*, and *test-4*, where *name* is an object-id, *gender* is a symmetric attribute, and the remaining attributes are asymmetric.

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

Table 8.2: A relational table containing mostly binary attributes.

For asymmetric attribute values, let the values Y and P be set to 1, and the value N be set to 0. Suppose that the distance between objects (patients) is computed based only on the asymmetric variables. According to the Jaccard coefficient formula (8.14), the distance between each pair of the three patients, Jack, Mary and Jim, should be,

$$d(jack, mary) = \frac{b+c}{a+b+c} = \frac{0+1}{2+0+1} = 0.33 \tag{8.15}$$

$$d(jack, jim) = \frac{b+c}{a+b+c} = \frac{1+1}{1+1+1} = 0.67 \tag{8.16}$$

$$d(jim, mary) = \frac{b+c}{a+b+c} = \frac{1+2}{1+1+2} = 0.75 \tag{8.17}$$

These measurements suggest that Jim and Mary are unlikely to have a similar disease. Of the three patients, Jack and Mary are the most likely to have a similar disease. □

8.2.4 Nominal, ordinal, and ratio-scaled variables

This section discusses how to compute the dissimilarity between objects described by nominal, ordinal, and ratio-scaled variables.

Nominal variables

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

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

The dissimilarity between two objects i and j can be computed using the **simple matching** approach as in (8.18):

$$d(i,j) = \frac{p-m}{p}, \tag{8.18}$$

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

Nominal variables can be encoded by a large number of asymmetric binary variables by creating a new binary variable for each of the M nominal states. For an object with a given state value, the binary variable representing that state is set to 1, while the remaining binary variables are set to 0. For example, to encode the nominal variable *map_color*, a binary variable can be created for each of the five colors listed above. For an object having the color *yellow*, the *yellow* variable is set to 1, while the remaining four variables are set to 0. The dissimilarity coefficient for this form of encoding can be calculated using the methods discussed in Section 8.2.3.

Ordinal variables

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

The treatment of ordinal variables is quite similar to that of interval-scaled variables when computing the dissimilarity between objects. Suppose that f is one of a set of ordinal variables describing n objects. The dissimilarity computation with respect to f involves the following steps.

1. The value of f for the ith object is x_{if}, and f has M_f ordered states, representing the ranking $1, .., M_f$. Replace each x_{if} by its corresponding rank, $r_{if} \in \{1, \ldots, M_f\}$.

2. Since each ordinal variable can have a different number of states, it is often necessary to map the range of each variable onto [0-1] so that each variable has equal weight. This can be achieved by replacing the rank r_{if} of the i-th object in the f-th variable by

$$z_{if} = \frac{r_{if}-1}{M_f-1}. \tag{8.19}$$

3. Dissimilarity can then be computed using any of the distance measures described in Section 8.2.2 for interval-scaled variables, using z_{if} to represent the f value for the ith object.

Ratio-scaled variables

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

$$Ae^{Bt} \quad or \quad Ae^{-Bt}, \tag{8.20}$$

where A and B are positive constants. Typical examples include the growth of a bacteria population, or the decay of a radio-active element.

There are three methods to handle ratio-scaled variables for computing the dissimilarity between objects.

1. Treat ratio-scaled variables like interval-scaled variables. This, however, is not usually a good choice since it is likely that the scale may be distorted.
2. Apply **logarithmic transformation** to a ratio-scaled variable f having value x_{if} for object i by using the formula $y_{if} = \log(x_{if})$. The y_{if} values can be treated as interval-valued, as described in Section 8.2.2. Notice that for some ratio-scaled variables, one may also apply log-log transformation, or other transformations, depending on the definition and application.
3. Treat x_{if} as continuous ordinal data and treat their ranks as interval-valued.

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

8.2.5 Variables of mixed types

Sections 8.2.2 to 8.2.4 discussed how to compute the dissimilarity between objects described by variables of the same type, where these types may be either *interval-scaled*, *symmetric binary*, *asymmetric binary*, *nominal*, *ordinal*, or *ratio-scaled*. However, in many real databases, objects are described by a *mixture* of variable types. In general, a database can contain all of the six variable types listed above. We need a method to compute the dissimilarity between objects of mixed variable types.

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

A more preferable approach is to process all variable types together, performing a single cluster analysis. One such technique, proposed by (Ducker et al. 1965) and extended by (Kaufman and Rousseeuw 1990), combines the different variables into a single dissimilarity matrix and brings all the meaningful variables onto a common scale of the interval [0,1].

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

$$d(i,j) \quad = \quad \frac{\Sigma_{f=1}^{p}, \delta_{ij}^{(f)} d_{ij}^{(f)}}{\Sigma_{f=1}^{p} \delta_{ij}^{(f)}} \tag{8.21}$$

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

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

Thus, the dissimilarity between objects can be computed even when the variables describing the objects are of different types.

8.3 A categorization of major clustering methods

There exist a large number of clustering algorithms in the literature. The choice of clustering algorithm depends both on the type of data available and on the particular purpose and application. If cluster analysis is used as a descriptive or exploratory tool, it is possible to try several algorithms on the same data to see what the data may disclose.

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

1. **Partitioning methods**.

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

   Given k, the number of partitions to construct, a partitioning method creates an initial partitioning. It then uses an *iterative relocation technique* which attempts to improve the partitioning by moving objects from one group to another. The general criterion of a good partitioning is that objects in the same cluster are "close" or related to each other, whereas objects of different clusters are "far apart" or very different. There are various kinds of other criteria for judging the quality of partitions.

   To achieve global optimality in partitioning-based clustering would require the exhaustive enumeration of all of the possible partitions. Instead, most applications adopt one of two popular heuristic methods: (1) the *k-means* algorithm, where each cluster is represented by the means value of the objects in the cluster; and (2) the *k-medoids* algorithm, where each cluster is represented by one of the objects located near the center of the cluster. These heuristic clustering methods work well for finding spherical-shaped clusters in small to medium sized databases. For finding clusters with complex shapes and for clustering very large data sets, partitioning-based methods need to be extended. Partitioning-based clustering methods are studied in depth in Section 8.4.

2. **Hierarchical methods**.

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

   Hierarchical methods suffer from the fact that once a step (merge or split) is done, it can never be undone. This rigidity of the hierarchical method is both the key to its success because it leads to smaller computation cost without worrying about a combinatorial number of different choices, as well as the key to its main problem because it cannot correct erroneous decisions.

   It can be advantageous to combine iterative relocation and hierarchical agglomeration by first using hierarchical agglomerative algorithm and then refining the result using iterative relocation. Some scalable clustering algorithms, such as BIRCH and CURE, have been developed based on such an integrated approach. Hierarchical clustering methods are studied in Section 8.5.

3. **Density-based methods**.

   Most partitioning methods cluster objects based on the distance between objects. Such methods can find only spherical-shaped clusters and encounter difficulty at discovering clusters of arbitrary shapes. Clustering methods have been developed based on the notion of *density*. The general idea is to continue growing the given cluster so long as the density (number of objects or data points) in the "neighborhood" exceeds some threshold, i.e., for each data point within a given cluster, the neighborhood of a given radius has to contain at least a minimum number of points. Such a method can be used to filter out noise (outliers), and discover clusters of arbitrary shape.

DBSCAN is a typical density-based method which grows clusters according to a density threshold. OPTICS is a density-based method which computes an augmented clustering ordering for automatic and interactive cluster analysis. Density-based clustering methods are studied in Section 8.6.

4. **Grid-based methods**.

   A grid-based method quantizes the object space into a finite number of cells which form a grid structure. It then performs all of the clustering operations on the grid structure (i.e., on the quantized space). The main advantage of this approach is its fast processing time which is typically independent of the number of data objects, and dependent only on the number of cells in each dimension in the quantized space.

   STING is a typical example of a grid-based method. CLIQUE and Wave-Cluster are two clustering algorithms which are both grid-based and density-based. Grid-based clustering methods are studied in Section 8.7.

5. **Model-based methods**.

   A model-based method hypothesizes a model for each of the clusters, and finds the best fit of the data to that model. A model-based algorithm may locate clusters by constructing a density function that reflects the spatial distribution of the data points. It also leads to a way of automatically determining the number of clusters based on standard statistics, taking "noise" or outliers into account and thus yielding robust clustering methods. Model-based clustering methods are studied in Section 8.8.

Some clustering algorithms integrate the ideas of several clustering methods, so that it is sometimes difficult to classify a given algorithm as uniquely belonging to only one clustering method category. Furthermore, some applications may have clustering criteria which require the integration of several clustering techniques.

In the following sections, we examine each of the above five clustering methods in detail. We also introduce algorithms which integrate the ideas of several clustering methods.

8.4 Partitioning methods

Given a database of n objects, and k, the number of clusters to form, a partitioning algorithm organizes the objects into k partitions ($k \leq n$), where each partition represents a cluster. The clusters are formed to optimize an objective partitioning criterion, often called a *similarity function*, such as distance, so that the objects within a cluster are "similar", whereas the objects of different clusters are "dissimilar" in terms of the database attributes.

8.4.1 Classical partitioning methods: k-means and k-medoids

The most well-known and commonly used partitioning methods are **k-means** proposed by (MacQueen 1967), and **k-medoids** proposed by (Kaufman and Rousseeuw 1987), and their variations.

Centroid-based technique: The k-means method

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

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

$$E = \Sigma_{i=1}^{k} \Sigma_{x \in C_i} |x - m_i|^2, \tag{8.22}$$

where x is the point in space representing the given object, and m_i is the mean of cluster C_i (both x and m_i are multidimensional). This criterion tries to make the resulting k clusters as compact and as separate as possible. The k-means procedure is summarized in Figure 8.1.

The algorithm attempts to determine k partitions that minimize the squared-error function. It works well when the clusters are compact clouds that are rather well separated from one another. The method is relatively scalable and

Algorithm 8.4.1 (k-means) The k-means algorithm for partitioning based on the mean value of the objects in the cluster.

Input: The number of clusters k, and a database containing n objects.

Output: A set of k clusters which minimizes the squared-error criterion.

Method: The k-means algorithm is implemented as follows.

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

□

Figure 8.1: k-means algorithm.

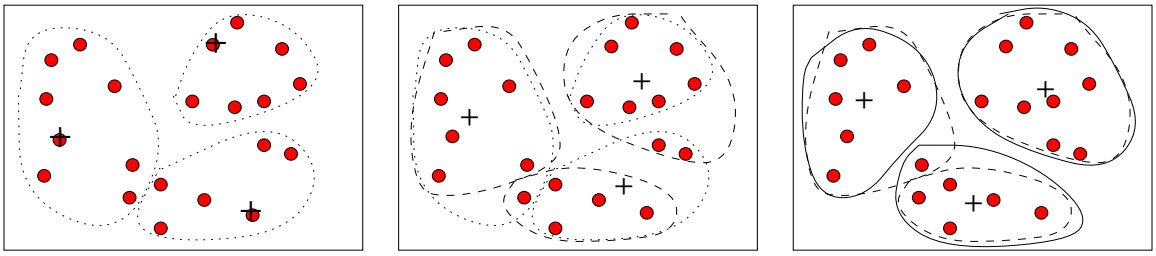


Figure 8.2: Clustering of a set of points based on the k-means method

efficient in processing large data sets because the computational complexity of the algorithm is $O(nkt)$, where n is the total number of objects, k is the number of clusters, and t is the number of iterations. Normally, $k \ll n$ and $t \ll n$. The method often terminates at a local optimum.

The k-means method, however, can be applied only when the mean of a cluster is defined. This may not be the case in some applications, such as when data with categorical attributes are involved. The necessity for users to specify k, the number of clusters, in advance can be seen as a disadvantage. The k-means method is not suitable for discovering clusters with non-convex shapes, or clusters of very different size. Moreover, it is sensitive to noise and outlier data points since a small number of such data can substantially influence the mean value.

Example 8.2 Suppose there are a set of objects located in a rectangle as shown in Figure 8.2. Let $k = 3$, that is, the user would like to cluster the objects into three clusters.

According to Algorithm 8.4.1, we arbitrarily choose 3 objects (marked by "+") as the initial three cluster centers. Then each object is distributed to the chosen cluster domains based on which cluster center is the nearest. Such a distribution forms a silhouette encircled by dotted curve, as shown in Figure 8.2 a).

This kind of grouping will update the cluster centers. That is, the mean value of each cluster is recalculated based on the objects in the cluster. Relative to these new centers, objects are re-distributed to the chosen cluster domains based on which cluster center is the nearest. Such a re-distribution forms a new silhouette encircled by dashed curve, as shown in Figure 8.2 b).

This process repeats, which leads to Figure 8.2 c). Eventually, no re-distribution of the objects in any cluster happens and the process terminates. The final clusters are the result of the clustering process. □

There are quite a few variants of the k-means method which differ in the selection of initial k means, the calculation of dissimilarity, and the strategies to calculate cluster means. An interesting strategy which often yields good results is to first apply a hierarchical agglomeration algorithm to determine the number of clusters and to find an initial classification, and then use iterative relocation to improve the classification.

Algorithm 8.4.2 The k-medoids algorithm for partitioning based on the central objects.

Input: The number of clusters k, and a database containing n objects.

Output: A set of k clusters which minimizes the sum of the dissimilarities of all the objects to their nearest medoid.

Method: The k-medoids algorithm is implemented as follows.

```
1)    arbitrarily choose k objects as the initial medoids;
2)    repeat
3)         assign each object to the cluster corresponding to the nearest medoid;
4)         calculate the objective function, which is the sum of dissimilarities
               of all the objects to their nearest medoid;
5)         swap the medoid x by an object y if such a swap reduces the objective function;
6)    until no change;
```

□

Figure 8.3: k-medoids algorithm.

Another variant to k-means is **k-modes** method by (Huang 1998) which extends the k-means paradigm to cluster categorical data by replacing the means of clusters with modes, using new dissimilarity measures to deal with categorical objects, and using a frequency-based method to update modes of clusters. The k-means and the k-modes methods can be integrated to cluster data with mixed numeric and categorical values, which is called the **k-prototypes** method.

Another variant to k-means is called EM (Expectation Maximization) algorithm by (Lauritzen 1995), which extends the k-means paradigm in a different way: Instead of assigning each point to a dedicated cluster, it assigns each point to a cluster according to some weight representing the probability of membership. In another word, there are no strict boundaries between clusters. Therefore, new means are then computed based on weighted measures.

A recent effort on scaling k-means algorithm, proposed by (Bradley, Fayyad, and Reina 1998), is based on an idea of identifying regions of the data that are compressible, regions that must be maintained in main memory, and the regions that are discardable. A point is *discardable* if its membership in a cluster is ascertained, and such a point can be discarded and only the summarized clustering feature of such discarded points should be retained; a point is *compressible* if it is not discardable but belongs to a tight subcluster, and such a subcluster is summarized using its summarized clustering feature; and if a point is neither discardable nor compressible, and such a point should be *retained in main memory*. The iterative clustering algorithm will include only the summarized clustering features of the compressible points and the points which must retain in main memory in the iterative cluster analysis, and thus turns a secondary-memory based alogrithm into a main memory-based algorithm to achieve the scalability.

Representative point-based technique: The k-medoids method

The k-means algorithm is sensitive to outliers since an object with some extremely large value may substantially distort the distribution of data. Instead of taking the mean value of the objects in a cluster as a reference point, one may take a representative object in a cluster, called a *medoid*, which is the most centrally located point in a cluster. Thus the partitioning method can still be performed based on the principle of minimizing the sum of the dissimilarities between each object and with its corresponding reference point, which forms the basis of the k-medoids method.

PAM (partition around medoids) is a k-medoids type clustering algorithm. It finds k clusters in n objects by first finding a representative object (*medoid*) for each cluster. The initial set of medoids could be arbitrarily selected. It then iteratively replaces one of the medoids by one of the non-medoids as long as the total distance of the resulting clustering is improved.

The detailed algorithm of PAM is presented in Figure 8.3. The algorithm attempts to determine k partitions for n objects. After initial selection of k *medoids*, the algorithm repeatedly tries to make a better choice of medoids by analyzing all the possible pairs of objects such that one object is a medoid and the other is not. The measure of clustering quality is calculated for each such combination. The best choice of points in one iteration is chosen as the

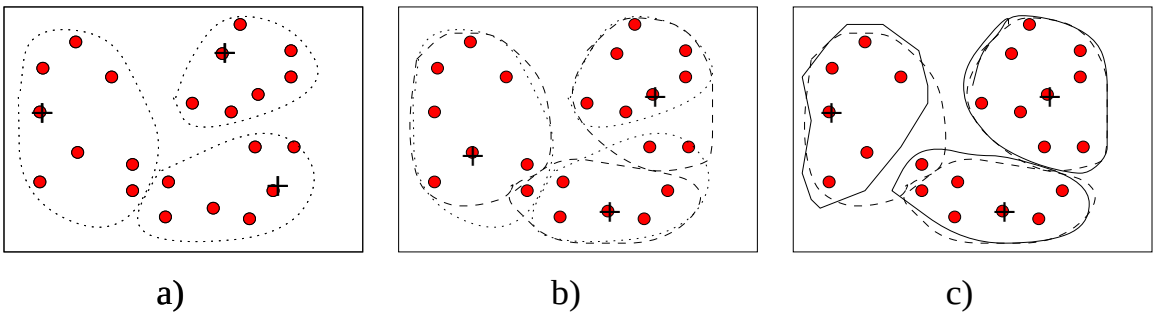


Figure 8.4: Clustering of a set of points based on the k-medoids method

medoids for the next iteration. The cost of a single iteration is $O(k(n-k)^2)$. For large values of n and k, the cost of such computation could be high.

Example 8.3 Suppose there are a set of object located in a rectangle as shown in Figure 8.4. Let $k = 3$, that is, the user would like to cluster the objects into three clusters.

According to Algorithm 8.4.2, we arbitrarily choose 3 objects (marked by "+") as the initial three cluster centers. Then each object is distributed to the chosen cluster domains based on which cluster center is the nearest. Such a distribution forms a silhouette encircled by dotted curve, as shown in Figure 8.2 a).

This kind of grouping will update the cluster centers. That is, the medoid of each cluster is recalculated based on the objects in the cluster. Regarding to these new centers, objects are re-distributed to the chosen cluster domains based on which cluster center is the nearest. Such a re-distribution forms a new silhouette encircled by dashed curve, as shown in Figure 8.4 b).

This process repeats, which leads to Figure 8.4 c). Eventually, no re-distribution of the objects in any cluster happens and the process terminates. The final clusters are the result of the clustering process. □

The k-medoids method is more robust than k-means in the presence of noise and outliers because a medoid is less influenced by outliers or other extreme values than mean. However, its processing is more costly than the k-means method. Moreover, it also needs user to specify k, the number of clusters.

8.4.2 Partitioning methods in large databases: from k-medoids to CLARANS

A typical k-medoids partition algorithm like PAM works effectively for small data sets, but it does not scale well for large data sets. To deal with larger data sets, a *sampling*-based method, called CLARA (clustering large applications) was developed by (Kaufman and Rousseeuw 1990).

The idea of CLARA is as follows: Instead of taking the whole set of data into consideration, only a small portion of the real data is chosen as a representative of the data, and *medoids* are chosen from this sample using PAM. If the sample is selected in a fairly random manner, it correctly represents the whole data set, and the representative objects (medoids) chosen will therefore be similar to those chosen from the whole data set. CLARA draws multiple samples of the data set, applies PAM on each sample, and gives the best clustering as the output. As expected, CLARA can deal with larger data sets than PAM. The complexity of each iteration now becomes $O(kS^2 + k(n-k))$, where S is the size of the sample, k is the number of clusters, and n is the total number of points.

The effectiveness of CLARA depends on the sample size. Notice that PAM searches for the best k medoids among a given data set, whereas CLARA searches for the best k medoids among the *selected* sample of the data set. CLARA cannot find the best clustering if any sampled medoid is not among the best k medoids. For example, if an object O_i is one of the medoids in the best k medoids but it is not selected during sampling, CLARA will never find the best clustering. This is exactly the tradeoff for efficiency. A good clustering based on samples will not necessarily represent a good clustering of the whole data set if the sample is biased.

To improve the quality and scalability of CLARA, another clustering algorithm called CLARANS (Clustering Large Applications based upon RANdomized Search) was proposed by (Ng and Han 1994). It is also a k-medoids type algorithm and combines the sampling technique with PAM. However, unlike CLARA, CLARANS does not confine itself to any sample at any given time. While CLARA has a fixed sample at every stage of the search, CLARANS

draws a sample with some randomness in each step of the search. The clustering process can be presented as searching a graph where every node is a potential solution, i.e., a set of k medoids. The clustering obtained after replacing a single medoid is called the *neighbor* of the current clustering. The number of neighbors to be randomly tried is restricted by a parameter. If a better neighbor is found, CLARANS moves to the neighbor's node and the process starts again; otherwise the current clustering produces a local optimum. If the local optimum is found, CLARANS starts with new randomly selected nodes in search for a new local optimum. CLARANS has been experimentally shown to be more effective than both PAM and CLARA. The computational complexity of every iteration in CLARANS is basically linearly proportional to the number of objects. It should be mentioned that CLARANS can be used to find the most natural number of clusters using *silhouette coefficient* which is a property of an object that specifies how much the object truly belongs to the cluster. CLARANS also enables the detection of outliers, i.e., the points that do not belong to any cluster. The performance of the CLARANS algorithm can been further improved by exploring spatial data structures, such as R*-trees, and some focusing techniques, as reported by (Ester, Kriegel and Xu 1995).

8.5 Hierarchical methods

A hierarchical clustering method works by grouping data objects into a tree of clusters. Hierarchical clustering methods can be further classified into agglomerative and divisive hierarchical clustering, depending on whether the hierarchical decomposition is formed in a bottom-up or top-down fashion. The quality of a pure hierarchical clustering method suffers from its inability to perform adjustment once a merge or split decision is done. Recent studies have been emphasizing the integration of hierarchical agglomeration with iterative relocation methods.

8.5.1 Agglomerative and divisive hierarchical clustering

In general, there are two types of hierarchical clustering methods:

1. **Agglomerative hierarchical clustering**: It starts by placing each object in its own cluster and then merges these atomic clusters into larger and larger clusters until all the objects are in a single cluster or until it satisfies certain termination condition. Most hierarchical clustering methods belong to this category. They differ only in their definition of between-cluster similarity.

   For example, a method called AGNES (Agglomerative Nesting) is introduced in (Kaufman and Rousseeuw 1990) and is also available in S-Plus. The method uses the *single-link* method, where each cluster is represented by all the data points in the cluster, and the similarity between two clusters is measured by the similarity of the *closest* pair of data points belonging to different clusters. AGNES merges the nodes (i.e., individual objects or clusters) that have the least dissimilarity and goes on in a nondescending fashion.

2. **Divisive hierarchical clustering**: It does the reverse by starting with all objects in one cluster, subdividing it into smaller and smaller pieces until each object forms a cluster on its own or until it satisfies certain termination condition, such as a desired number of clusters is obtained or the distance between two closest clusters is above a certain threshold distance. Divisive methods are not generally available and rarely have been applied due to the difficulty of making a right decision of splitting at a high level. A divisive hierarchical clustering method, called DIANA (Divisia Analysis), is introduced in (Kaufman and Rousseeuw 1990), and is also available in S-Plus.

Merging of clusters is often based on the distance between clusters. The widely used measures for distance between clusters are as follows, where m_i is the mean for cluster C_i, n_i is the number of points in C_i, and $|p - p'|$ is the distance between two points p and p'.

$$
\begin{aligned}
d_{min}(C_i, C_j) &= min_{p \in Ci, p' \in C_j} |p - p'| \\
d_{mean}(C_i, C_j) &= |m_i - m_j| \\
d_{avg}(C_i, C_j) &= 1/(n_i n_j) \Sigma_{p \in C_i} \Sigma_{p' \in C_j} |p - p'| \\
d_{max}(C_i, C_j) &= max_{p \in Ci, p' \in C_j} |p - p'|
\end{aligned}
$$

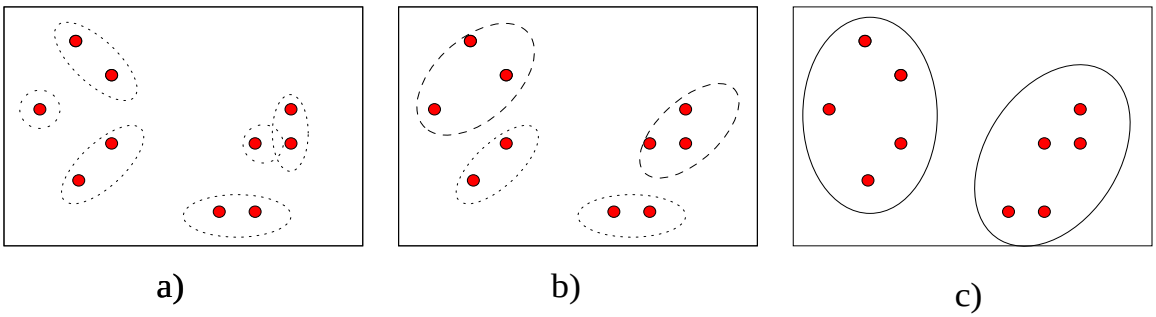


Figure 8.5: Clustering of a set of points based on the "Agglomerative Nesting" method

Example 8.4 Suppose there are a set of objects located in a rectangle as shown in Figure 8.5. An agglomerative hierarchical clustering method, called AGNES, works as follows. Initially, every object is placed into a cluster of its own. Then the clusters are merged step-by-step according to some principle such as merging the clusters with the minimum Euclidean distance between the closest objects in the cluster. Figure 8.5 *a*) shows that the closest (i.e., with minimum Euclidean distance) single object clusters are first merged into two object clusters. This cluster merging process repeats, and the closest clusters are further merged, as shown in Figure 8.5 *b*) and *c*). Eventually, all the objects are merged into one big cluster.

A divisive hierarchical clustering method, called DIANA (Divisia Analysis), works in the reverse order. That is, initially, all the objects are placed in one cluster. Then the cluster is split according to some principle, such as splitting the clusters according to the maximum Euclidean distance between the closest neighboring objects in the cluster. Figure 8.5 *c*) can be viewed as the result of the first split. This cluster splitting process repeats, and each cluster is further split according to the same criteria. Figure 8.5 *b*) and *a*) can be viewed as snapshots of such splitting. Eventually, every cluster will contain only a single object.

In either agglomerative or divisive hierarchical clustering, one can specify the desired number of clusters as a termination condition so that the hierarchical clustering process will terminate when the process reaches the desired number of clusters. □

The hierarchical clustering method, though simple, often encounters difficulties at making critical decisions for selection of correct merge or split points. Such a decision is critical because once a group of objects is merged or split, the process at the next step will work on the newly generated clusters. That is, it will never undo what was done previously nor perform object swapping between clusters. Thus merge or split decisions, if not wise enough at some step, may lead to low quality clusters. Moreover, the method does not scale well since the decision of merge or split needs to examine and evaluate a good number of objects or clusters.

One promising direction to improve the clustering quality of hierarchical method is to integrate hierarchical clustering with other clustering techniques for multiple phase clustering. A few such methods are introduced in the following subsections. The first, called BIRCH, first partitions objects hierarchically using tree structures and then applies other clustering algorithms to form refined clusters. The second, called CURE, represents each cluster by a certain fixed number of representative points and then shrinks them toward the center of the cluster by a specified fraction. The third, called ROCK, merges clusters based on their inter-connectivity. The fourth, called CHAMELEON, explores dynamic modeling in hierarchical clustering.

8.5.2 BIRCH: Balanced Iterative Reducing and Clustering using Hierarchies

An interesting integrated hierarchical clustering method, called BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) was developed by (Zhang, Ramakrishnan and Livny 1996). It introduces two concepts, **clustering feature** and **CF tree** (**Clustering Feature tree**), and uses *CF tree* as a summarize cluster representation to achieve good speed and clustering scalability in large databases. It is also good for incremental and dynamical clustering of incoming data points.

A **clustering feature CF** is a triplet summarizing information about subclusters of points. Given N d-

dimensional points $\{X_i\}$ in a subcluster, CF is defined as

$$CF = (N, \vec{LS}, SS) \tag{8.23}$$

where N is the number of points in the subcluster, $\vec{LS}$ is the linear sum on N points, i.e., $\sum_{i=1}^{N} \vec{X_i}$, and SS is the square sum of data points, i.e., $\sum_{i=1}^{N} \vec{X_i}^2$.

Clustering feature is essentially summary statistics for the cluster: the zero-th, first, and second moments of the subcluster from statistical point of view. It registers crucial measurements for computing clusters and utilizes storage efficiently since it summarizes the information about the subclusters of points instead of storing all points.

A **CF tree** is a height-balanced tree which stores the clustering features. It has two parameters: branching factor B and threshold T. The branching factor specifies the maximum number of children. The threshold parameter specifies the maximum diameter of subclusters stored at the leaf nodes. By changing the threshold value one can change the size of the tree. The non-leaf nodes store sums of their children's CFs, and thus, summarize the information about their children.

The BIRCH algorithm has the following two phases:

- **Phase 1**: Scan database to build an initial in-memory CF tree, which can be viewed as a multi-level compression of the data that tries to preserve the inherent clustering structure of the data.
- **Phase 2**: Apply a (selected) clustering algorithm to cluster the leaf nodes of the CF-tree.

For Phase 1, the *CF tree* is built dynamically as data points are inserted. Thus, the method is an incremental one. A point is inserted to the closest leaf entry (subcluster). If the diameter of the subcluster stored in the leaf node after insertion is larger than the threshold value, then the leaf node and possibly other nodes are split. After the insertion of the new point, the information about it is passed towards the root of the tree. One can change the size of the *CF tree* by changing the threshold. If the size of the memory that is needed for storing the *CF tree* is larger than the size of the main memory, then a smaller value of threshold is specified and the *CF tree* is rebuilt. The rebuild process is performed by building a new tree from the leaf nodes of the old tree. Thus, the process of rebuilding the tree is done without the necessity of reading all the points. This is similar to the insertion and node split in the construction of B+-trees. Therefore, for building the tree, data has to be read just once. Some heuristics and methods are also introduced to deal with outliers and improve the quality of CF trees by additional scans of the data.

After CF tree is built, any clustering algorithm, such as a typical partitioning algorithm, can be used with the CF tree in Phase 2.

BIRCH tries to produce the best clusters with the available resources. With limited amount of main memory, an important consideration is to minimize the time required for I/O. It applies a multi-phase clustering technique: A single scan of data set yields a basic good clustering, and one or more additional scans can (optionally) be used to further improve the quality. So the computation complexity of the algorithm is $O(N)$, where N is the number of objects to be clustered.

Experiments have shown the linear scalability of the algorithm with respect to the number of points and good quality of clustering of the data. However, since each node in CF-tree can only hold a limited number of entries due to its size, a CF-tree node does not always correspond to a natural cluster. Moreover, if the clusters are not spherical in shape, BIRCH does not perform well because it uses the notion of radius or diameter to control the boundary of a cluster.

8.5.3 CURE: Clustering Using REpresentatives

Most clustering algorithms either favors clusters with spherical shape and similar sizes, or are fragile in the presence of outliers. An interesting method, called CURE (Clustering Using REpresentatives), by (Guha, Rastogi and Shim 1998), integrates hierarchical and partitioning algorithms and overcomes the problem of favoring clusters with spherical shape and similar sizes.

CURE employs a novel hierarchical clustering algorithm that adopts a middle ground between the centroid-based and the all-point extremes. Instead of using a single centroid to represent a cluster, a fixed number of representative points are chosen to represent a cluster. These representative points are generated by first selecting well-scattered

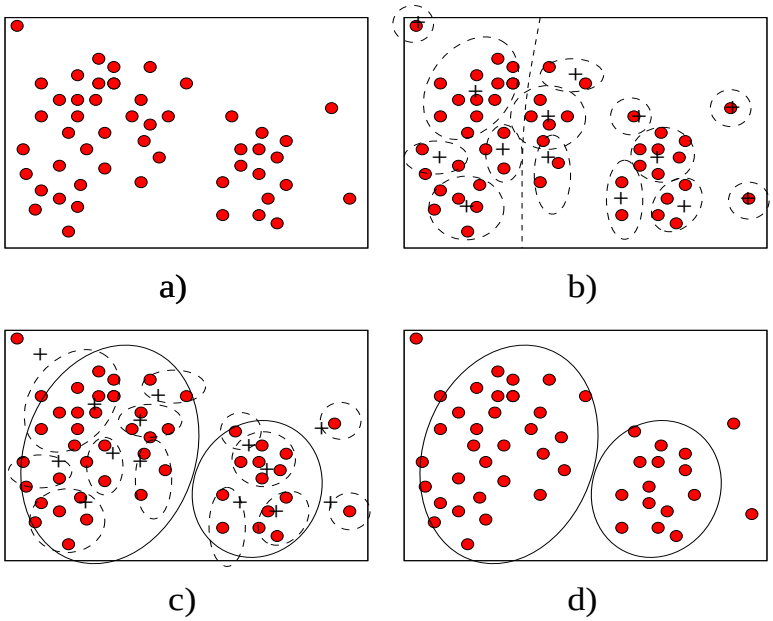


Figure 8.6: Clustering of a set of points by CURE

points from the cluster and then shrinking them toward the center of the cluster by a specified fraction (shrinking factor). The clusters with the closest pair of representative points are the clusters that are merged at each step of the algorithm.

Having more than one representative point per cluster allows CURE to adjust well to the geometry of non-spherical shapes and avoid single-link effect. The shrinking helps dampen the effects of outliers. Therefore, CURE is more robust to outliers and identifies clusters having non-spherical shapes and wide variance in size.

To handle large databases, CURE employs a combination of random sampling and partitioning: A random sample is first partitioned and each partition is partially clustered. The partial clusters are then clustered in a second pass to yield the desired clusters.

The major steps of the CURE algorithm are briefly outlined as follows: (1) draw a random sample s, (2) partition sample s to p partitions, each of size s/p, (3) partially cluster partitions into s/pq clusters, for some $q > 1$, (4) eliminate outliers by random sampling: if a cluster grows too slow, eliminate it; (5) cluster partial clusters, a process of shrinking multiple representative points towards the gravity center by a fraction of α, specified by a user, where the multiple representatives capture the shape of the cluster; and (6) mark data with the corresponding cluster labels.

Here we present an example to show how CURE works.

Example 8.5 Suppose there are a set of objects located in a rectangle region. Let $p = 2$, that is, the user would like to cluster the objects into two clusters.

First, 50 objects are sampled as shown in Figure 8.6 *a*). Then these objects are partitioned initially into two clusters, each containing 25 points. We partially cluster the partitions into 10 clusters based on minimal mean distance. Each cluster representative is marked by a small cross, as shown in Figure 8.6 *b*). These representatives are shifted towards the center of gravity by a fraction α, as shown in Figure 8.6 *c*). They capture the shape of the cluster and form two clusters. Thus, the objects are partitioned into two clusters, with the outliers removed, as shown in Figure 8.6 *d*). □

CURE produces high quality clusters in the existence of outliers, complex shapes of clusters with different size. It scales well for large databases without sacrificing clustering quality. CURE needs a few user-specified parameters, such as the size of the random sample, the number of desired clusters, and the shrinking fraction α. One question is the sensitivity of these parameters to the results of clustering. A sensitivity analysis has been provided on the effect of changing parameters. It shows although some parameters can be varied without impacting the quality of clustering, the parameter setting in general does have a significant influence on the results.

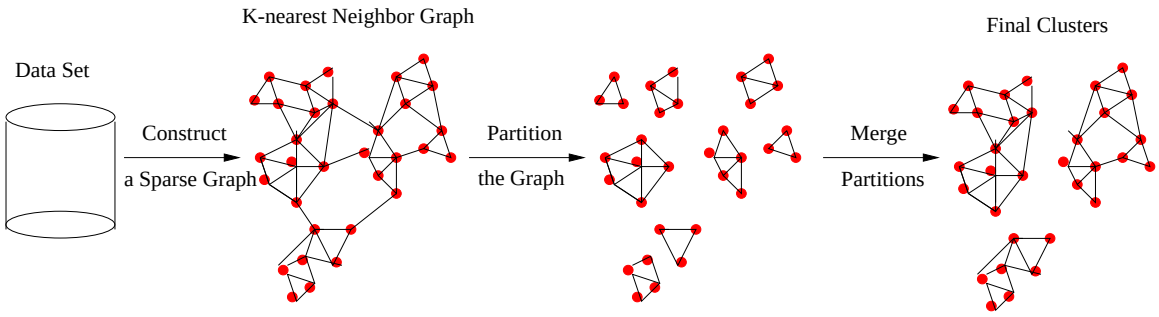


Figure 8.7: CHAMELEON: Hierarchical clustering based on k-nearest neighbors and dynamic modeling

Another agglomerative hierarchical clustering algorithm developed by (Guha, Rastogi and Shim 1999), call ROCK, is suited for clustering categorical attributes. It measures the similarity of two clusters by comparing the aggregate inter-connectivity of two clusters against a user-specified static inter-connectivity model, where the inter-connectivity of two clusters C_1 and C_2 are defined by the number of cross links between the two clusters, and $link(p_i, p_j)$ are the number of common neighbors between two points p_i and p_j.

ROCK first constructs a sparse graph from a given data similarity matrix using a similarity threshold and the concept of shared neighbors, and then perform a hierarchical clustering algorithm on the sparse graph.

8.5.4 CHAMELEON: A hierarchical clustering algorithm using dynamic modeling

Another interesting clustering algorithm, called CHAMELEON, which explores dynamic modeling in hierarchical clustering, has been developed by Karypis, Han and Kumar (1999). In its clustering process, two clusters are merged if the inter-connectivity and closeness (proximity) between two clusters are highly related to the internal inter-connectivity and closeness of objects within the clusters. The merge process based on the dynamic model facilitates the discovery of natural and homogeneous clusters, and it applies to all types of data as long as a similarity function is specified.

CHAMELEON is derived based on the observation of the weakness of two hierarchical clustering algorithms: CURE and ROCK. CURE and related schemes ignore the information about the aggregate inter-connectivity of objects in two clusters; whereas ROCK, the group averaging method, and related schemes ignore the information about the closeness of two clusters while emphasizing on their inter-connectivity.

CHAMELEON first uses a graph partitioning algorithm to cluster the data items into a large number of relatively small subclusters. Then it uses an agglomerative hierarchical clustering algorithm to find the genuine clusters by repeatedly combining together these clusters. To determine the pairs of most similar subclusters, it takes into account both the inter-connectivity as well as the closeness of the clusters, especially the internal characteristics of the clusters themselves. Thus it does not depend on a static, user-supplied model, and can automatically adapt to the internal characteristics of the clusters being merged.

As shown in Figure 8.7, CHAMELEON represents its objects based on the commonly used k-nearest neighbor graph approach. Each vertex of the k-nearest neighbor graph represents a data object, and there exists an edge between two vertices (objects), if one object is among the k-most similar objects of the other. The k-nearest neighbor graph G_k captures the concept of neighborhood dynamically: The neighborhood radius of a data point is determined by the density of the region in which this object resides. In a dense region, the neighborhood is defined narrowly and in a sparse region, the neighborhood is defined more widely. Comparing with the model defined by density-based method like DBSCAN, which will be introduced in Section 8.6 and which uses a global neighborhood density, G_k captures more natural neighborhood. Moreover, the density of the region is recorded as the weight of the edges. That is, the edge of a dense region tends to weight more than that of a sparse region.

CHAMELEON determines the similarity between each pair of clusters C_i and C_j according to their relative inter-connectivity $RI(C_i, C_j)$ and their relative closeness $RC(C_i, C_j)$.

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

and C_j. That is,

$$RI(C_i, C_j) = \frac{|EC_{\{C_i,C_j\}}|}{\frac{1}{2}(|EC_{C_i}| + |EC_{C_j}|)} \tag{8.24}$$

where $EC_{\{C_i,C_j\}}$ is the *edge-cut* of the cluster containing both C_i and C_j so that the cluster is broken into C_i and C_j, and similarly, EC_{C_i} (or EC_{C_j}) is the *size of its min-cut bisector* (i.e., the weighted sum of edges that partition the graph into two roughly equal parts).

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

$$RC(C_i, C_j) = \frac{\overline{S}_{EC_{\{C_i,C_j\}}}}{\frac{|C_i|}{|C_i|+|C_j|}\overline{S}_{EC_{C_i}} + \frac{|C_j|}{|C_i|+|C_j|}\overline{S}_{EC_{C_j}}} \tag{8.25}$$

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

It has been shown that CHAMELEON has more power at discovering arbitrary shaped clusters in high quality than DBSCAN and CURE. However, the processing cost for high dimensional data may take $O(n^2)$ time for n objects in the worst case.

8.6 Density-based clustering methods

To discover clusters with arbitrary shape, density-based clustering methods have been developed, which either connects regions with sufficiently high density into clusters or clusters objects based on density function distribution.

8.6.1 DBSCAN: A density-based clustering method based on connected regions with sufficiently high density

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a density-based clustering algorithm, developed by Ester, Kriegel, Sander and Xu (1996). The algorithm grows regions with sufficiently high density into clusters and discovers clusters of arbitrary shape in spatial database with noise. A cluster is defined as a maximal set of density-connected points.

The basic idea of density-based clustering is as follows: For each object of a cluster, the neighborhood of a given radius (ϵ) (called *ϵ-neighborhood*) has to contain at least a minimum number of objects ($MinPts$).

An object, within a given radius (ϵ) containing no less than a minimum number of neighborhood objects ($MinPts$), is called a **core object** (with respect to a radius ϵ and a minimum number of points $MinPts$).

An object p is **directly density-reachable** from object q with respect to a radius ϵ and a minimum number of points $MinPts$ in a set of objects D if p is within the ϵ-neighborhood of q which contains at least a minimum number of points, $MinPts$.

An object p is **density-reachable** from object q with respect to ϵ and $MinPts$ in a set of objects D if there is a chain of objects $p_1, \ldots, p_n$, $p_1 = q$ and $p_n = p$ such that for $1 \leq i \leq n$, $p_i \in D$ and p_{i+1} is directly density-reachable from p_i with respect to ϵ and $MinPts$.

An object p is **density-connected** to object q with respect to ϵ and $MinPts$ in a set of objects D if there is an object $o \in D$ such that both p and q are density-reachable from o with respect to ϵ and $MinPts$.

Example 8.6 In Figure 8.8, based on a given ϵ represented by the radius of the circles, and let $MinPts = 3$, M is directly density-reachable from P, and Q is (indirectly) density-reachable from P. However, P is not density-reachable from Q. Similarly, R and S are density-reachable from O; and O, R and S are all density-connected. ■

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

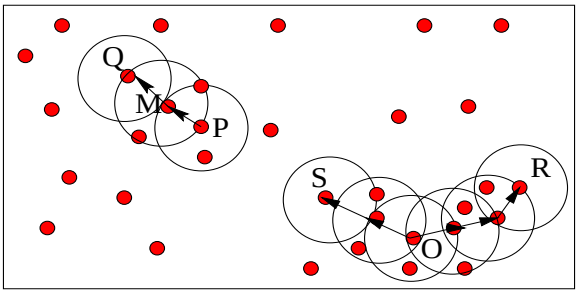


Figure 8.8: Density reachability and density connectivity in density-based clustering

A **density-based cluster** is a set of density-connected objects which is maximal with respect to density-reachability, and every object not contained in any cluster is *noise*.

Based on the notion of density-reachability, a density-based clustering algorithm, DBSCAN, is developed for clustering data in a database. It checks the ϵ-neighborhood of each point in the database. If the ϵ-neighborhood of a point p contains more than $MinPts$, a new cluster with p as a core object is created. It then iteratively collects directly density-reachable objects from these core objects, which may involve the merge of a few density-reachable clusters. The process terminates when no new point can be added to any cluster.

8.6.2 OPTICS: Ordering Points To Identify the Clustering Structure

Although the density-based clustering algorithm, DBSCAN, can discover cluster objects with the selection of some input parameters, such as ϵ and $MinPts$, it still puts the responsibility to users to select good parameter values in order to find correct clusters. Actually, this is the problem associated with many other clustering algorithms. Such parameter settings are usually pretty hard to determine, especially for real-world, high dimensional data sets. Most algorithms are very sensitive to such parameter values: slightly different settings may lead to very different partitions of the data sets. Moreover, high-dimensional real data sets often have very skewed distribution, and there does not even exist a global parameter setting for which the result of a clustering algorithm may describe the intrinsic clustering structure accurately.

To overcome this difficulty, a cluster ordering method, called OPTICS (Ordering Points To Identify the Clustering Structure) has been developed by (Ankerst, Breunig, Kriegel and Sander 1999). It computes an augmented clustering-ordering for automatic and interactive cluster analysis. This cluster ordering contains information which is equivalent to density-based clustering corresponding to a broad range of parameter settings.

By examining density-based clustering algorithm, DBSCAN, one can easily see that for a constant $MinPts$-value, density-based clusters with respect to a higher density (i.e., a lower value for ϵ) are completely contained in density-connected sets with respects to a lower density. Therefore, in order to produce density-based clusters with a set of distance parameters, one need to select the objects for processing in a specific order so that the object which is density-reachable with respect to the lowest ϵ value is finished first.

Based on this idea, two value need to be stored for each object: *core-distance* and *reachability-distance*.

The **core-distance** of an object p is the smallest distance ϵ' between p and an object in its ϵ-neighborhood such that p would be a core object with respect to ϵ' if this neighbor is contained in the ϵ-neighborhood of p. Otherwise, the core-distance is UNDEFINED.

The **reachability-distance** of an object p with respect to another object o is the smallest distance such that p is directly density-reachable from o if o is a core object. If o is not a core object, even at the generating distance ϵ, the reachability-distance of an object p with respect to o is UNDEFINED.

The OPTICS algorithm creates an ordering of a database, additionally storing the core-distance and a suitable reachability-distance for each object. Such information is sufficient for extraction of all density-based clusterings with respect to any distance ϵ' smaller than the generating distance ϵ from this order.

The cluster-ordering of a data set can be presented and understood graphically. For example, Figure 8.9 is the reachability-plot for a simple 2-dimensional data set, which presents a general overview about how the data is structured and clustered. Methods are also developed for viewing clustering structures for high dimensional data.

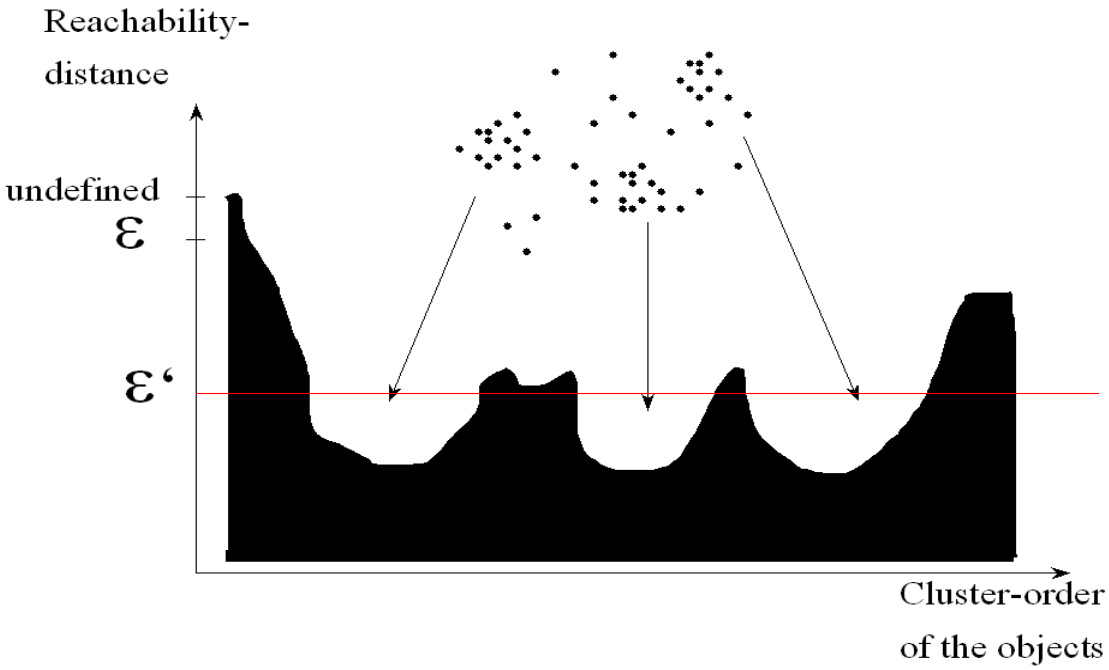


Figure 8.9: Cluster ordering in OPTICS

Because of the structural equivalence of the OPTICS algorithm to DBSCAN, the OPTICS algorithm has the same run-time complexity as that of DBSCAN. Spatial indexing structures can be used to further enhance its performance.

8.6.3 DENCLUE: Clustering based on density distribution functions

DENCLUE (an abbreviation for DENsity-based CLUstEring), a clustering method based on a set of density distribution functions, has been developed by (Hinneburg and Keim 1998).

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

The **influence function** of a data point $y \in F^d$, where F^d is a d-dimensional feature space, is a basic function $f_B^y : F^d \rightarrow R_0^+$, which is defined in terms of a basic influence function f_B,

$$f_B^y = f_B(x, y). \tag{8.26}$$

In principle, the influence function can be an arbitrary function, but it should be reflexive and symmetric. It can be a *Euclidean distance function*, a *square wave influence function*,

$$f_{Square}(x, y) = \begin{cases} 0 & if\ d(x, y) > \sigma \\ 1 & otherwise \end{cases} \tag{8.27}$$

or a *Gaussian influence function*,

$$f_{Gauss}(x, y) = e^{-\frac{d(x,y)^2}{2\sigma^2}} \tag{8.28}$$

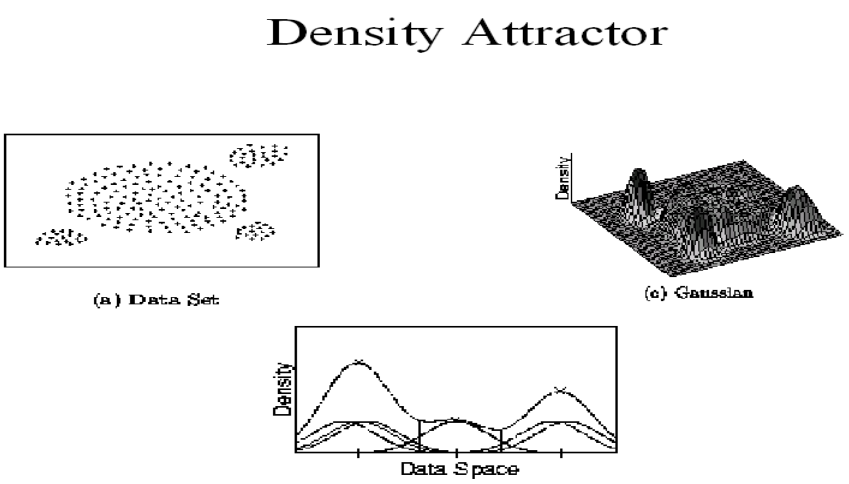


Figure 8.10: Density function and density-attractor

The **density function** is defined as the sum of influence functions of all data points. Given N data objects described by a set of feature vectors $D = \{x_1, \ldots, x_N\} \subset F^d$, the density function is defined as,

$$f_B^D = \Sigma_{i=1}^N f_B^{x_i}(x). \tag{8.29}$$

For example, the density function which results from the Gaussian influence function (8.28) is,

$$f_{Gaussian}^D(x) = \Sigma_{i=1}^N e^{-\frac{d(x,y)^2}{2\sigma^2}} \tag{8.30}$$

From the density function, one can define the *gradient* of a function and the *density attractor* which is the local maxima of the overall density function. For a continuous and differentiable influence function, a hill climbing algorithm, guided by the gradient, can be used to determine the density-attractor of a set of data points.

Based on these notions, both *center-defined cluster* and *arbitrary-shape cluster* can be defined formally. A *center-defined cluster* is a subset C being density-extracted, with its density function no less than a threshold ξ; otherwise (i.e., if density function is less than ξ), it is an outlier. An *arbitrary-shape cluster* is a set of C's, each being density-extracted, with the density function no less than a threshold ξ, and there exists a path P from each region to another and the density function for each point along the path is no less than ξ.

DENCLUE has the following major advantages in comparison with many clustering algorithms: (1) it has a solid mathematical foundation, and generalizes other clustering methods, including partition-based, hierarchical, and locality-based methods, (2) it has good clustering properties for data sets with large amounts of noise, (3) it allows a compact mathematical description of arbitrarily shaped clusters in high dimensional data sets, and (4) it uses grid cells but only keeps information about grid cells that do actually contain data points and manages these cells in a tree-based access structure, and thus it is significantly faster than some influential algorithms, such as it is faster than DBSCAN by a factor of up to 45. However, the method needs careful selection of the parameters, density parameter σ and noise threshold ξ, and the selection of such parameters may significantly influence the quality of clustering results.

8.7 Grid-based clustering methods

A grid-based approach uses multi-resolution grid data structure. It first quantizes the space into a finite number of cells which form a grid structure and then performs all the operations in such a grid structure. The main advantage of the approach is its fast processing time which is typically independent of the number of data objects but dependent only on the number of cells in each dimension in the quantized space.

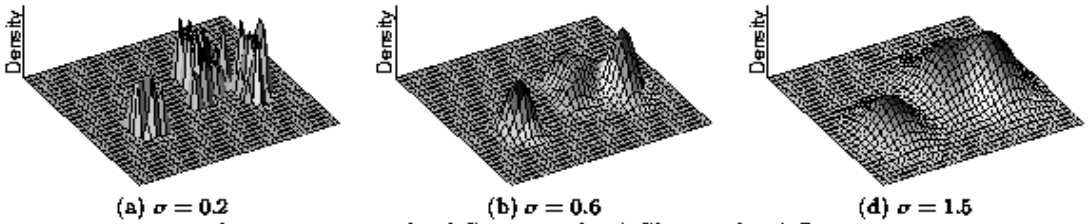


Figure 3: Example of Center-Defined Clusters for different σ

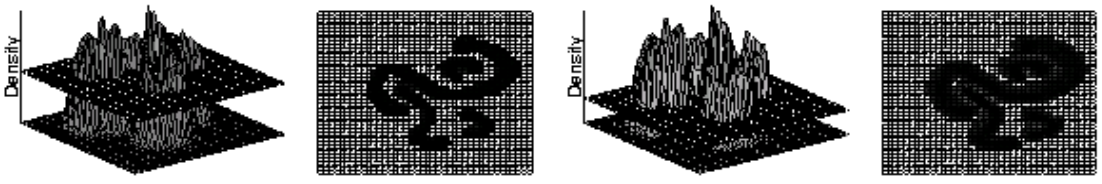


Figure 4: Example of Arbitray-Shape Clusters for different ξ

Figure 8.11: Examples of center-defined clusters and arbitrary-shaped clusters

Some typical example of the grid-based approach include STING, which explores statistical information stored in the grid cells; WaveCluster, which clusters objects using a wavelet transform method; and CLIQUE, which represents a grid- and density-based approach for clustering in high-dimensional data space.

8.7.1 STING: A Statistical Information Grid Approach

STING (STatistical INformation Grid) is a grid-based multi-resolution approach developed by (Wang, Yang and Muntz 1997). In this approach, the spatial area is divided into rectangular cells. There are usually several levels of such rectangular cells corresponding to different levels of resolution and these cells form a hierarchical structure: each cell at a high level is partitioned to form a number of cells at the next lower level. Moreover, important pieces of statistical information, such as mean, max, min, count, standard deviation, etc. associated with the attribute values in each grid cell are precomputed and stored before a query is submitted to a system.

Figure 8.12 shows a hierarchical structure for STING clustering.

The set of statistics-based parameters includes the following: the attribute-independent parameter, n (*count*), and attribute-dependent parameters, m (*mean*), s (standard deviation), *min* (minimum), *max* (maximum), and the *type of distribution* that the attribute value in the cell follows, such as *normal, uniform*, *exponential*, or *none* (if the distribution is unknown). When the data is loaded into the database, the set of parameters, n, m, s, *min*, *max* of the bottom level cells are calculated directly from data. The value of *distribution* could be either assigned by the user if the distribution type is known beforehand or obtained by hypothesis tests such as χ^2-test. Parameters of higher level cells can be easily computed from the parameters of the lower level cells. The type of distribution of a higher level cell can be computed based on the majority distribution types of its corresponding lower level cells plus a threshold filtering process. If the distributions of the lower level cell disagree with each other and fails the threshold test, the distribution type of the high level cell is set to "*none*".

The statistical information collected will be very useful at answering queries. A top-down, statistics information grid-based query answering method can be outlined as follows. First, one can determine a layer to start with, which usually contains a small number of cells. For each cell in the current layer, we calculate the confidence interval (or estimated range) of probability that this cell is relevant to the query. The irrelevant cells will be removed from further consideration, and the deeper layer processing will examine the relevant cells only. This process repeats until it reaches the bottom layer. At this time, if the query specification is met, return the regions of relevant cells that meet the requirement of the query; otherwise, retrieve the data that fall into the relevant cells, do further processing; and return the results that meet the requirement of the query.

The approach offers several advantages over some other clustering methods: (1) the grid-based computation is

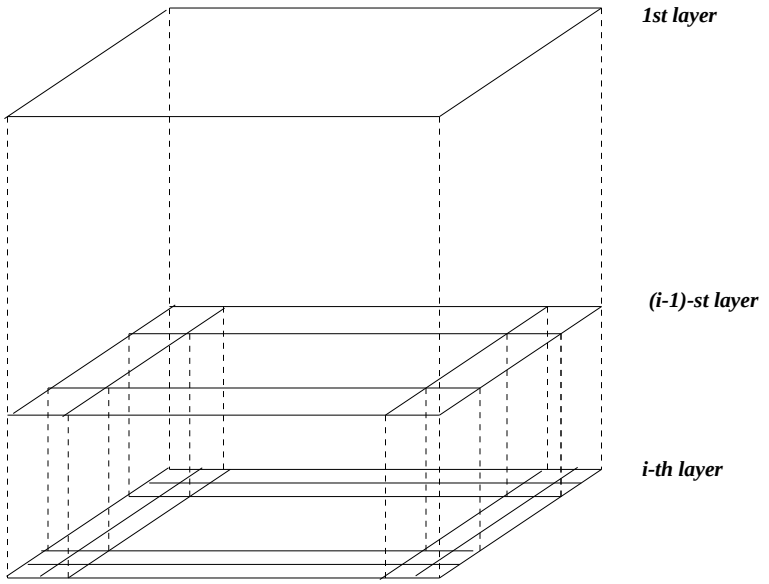


Figure 8.12: A hierarchical structure for STING clustering

query-independent since the statistical information stored in each cell represents the summary information of the data in the grid cell, independent of query; (2) the grid structure facilitates parallel processing and incremental updating; and (3) a major advantage of the method is the efficiency of the method: STING goes through the database once to compute the statistical parameters of the cells, and hence the time complexity of generating clusters in $O(N)$, where N is the total number of objects. After generating the hierarchical structure, the query processing time is $O(G)$, where G is the total number of grid cells at the lowest level, which is usually much smaller than N, the total number of objects.

However, since STING uses multi-resolution approach to perform cluster analysis, the quality of STING clustering will be dependent on the granularity of the lowest level of the grid structure. If the granularity is very fine, the cost of process will increase substantially; however, if the bottom level of the grid structure is too coarse, it may reduce the fine quality of cluster analysis. Moreover, STING does not consider the spatial relationship between the children and their neighboring cells to construct the parent cell. As a result, the shapes of the resulting clusters are isothetic, that is, all the cluster boundaries are either horizontal or vertical, and no diagonal boundary is detected. This may lower the quality and accuracy of the clusters despite the fast processing time of the approach.

8.7.2 WaveCluster: Clustering using wavelet transformation

WaveCluster, developed by (Sheikholeslami, Chatterjee and Zhang 1998), is a multi-resolution clustering approach which first summarize the data by imposing a multidimensional grid structure on the data space, and then transforms the original feature space by wavelet transform and find dense regions in the transsformed space.

In this approach, each grid cell summarizes the ifnromation of a group of points which map into the cell, and this summary information typically fits into main memory for multi-resolution wavelet transform and the subsequent cluster analysis. In the grid structure, the numerical attributes of a spatial object can be represented by a *feature vector* where each element of the vector corresponds to one numerical attribute, or *feature*. For an object with n numerical attributes, the feature vector will be one point in the n-dimensional feature space.

Wavelet transform is a signal processing technique that decomposes a signal into different frequency subbands. The wavelet model also works on n-dimensional signals by applying one-dimensional transform n times.

In wavelet transform, spatial data are converted into the frequency domain. Convolution with an appropriate kernel function results in a transformed space where the natural clusters in the data become more distinguishable. Clusters can then be identified by finding the dense regions in the transformed domain.

Wavelet transform provides the following interesting features. First, it provides unsupervised clustering. The hat-shape filters emphasize regions where the points cluster, but simultaneously tend to suppress weaker information

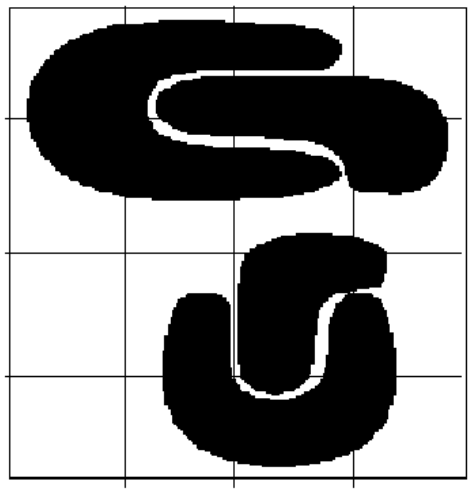


Figure 8.13: A sample of 2-dimensional feature space. Figure is from Sheikholeslami, Chatterjee and Zhang (1998).

in their boundary. Thus, dense regions in the original feature space act as attractors for nearby points and as inhibitors for the points that are not close enough. This means the clusters in the data automatically stand out and clear the regions around them. Second, the low-pass filters used in the wavelet transform will automatically remove outliers. Moreover, the multi-resolution property of wavelet transform can help detecting the clusters at different levels of accuracy. Finally, it is very fast to apply wavelet transform and such a process can also be performed in parallel.

The wavelet-based clustering algorithm can be outlined as follows:

Algorithm 8.7.1 The wavelet-based clustering algorithm for multi-resolution clustering by wavelet transform.

Input: The feature vectors of multidimensional data objects.

Output: Clustered objects.

Method: The wavelet-based clustering algorithm is implemented as follows.

```
(1)    Quantize feature space and then assign objects to the units;
(2)    Apply wavelet transform on the feature space;
(3)    Find the connected components (clusters) in the subbands of
       transformed feature space, at different levels;
(4)    Assign labels to the units;
(5)    Make the look up tables and map the objects to the clusters. ■
```

The computational complexity of the algorithm is $O(N)$, where N is the number of objects in the database.

For example, Fig. 8.13 shows a sample of 2-dimensional feature space, where each point in the image represents the feature values of one object in the spatial data sets. Fig. 8.14 shows the result of wavelet transforms at different scales from fine (scale 1) to coarse (scale 3). At each level, sub-band LL (average) is shown at the upper-left quadrant, sub-band LH (horizontal edges) is shown at the upper-right quadrant, sub-band HL (vertical edges) is shown at the lower-left quadrant, and sub-band HH (corners) is shown at the lower-right quadrant.

WaveCluster is a grid-based and density-based algorithm. WaveCluster conforms with all the requirements of good clustering algorithms: It handles large datasets efficiently, discovers clusters with arbitrary shape, successfully handles outliers, and is insensitive to the order of input. The study also compares WaveCluster with BIRCH, CLARANS and DBSCAN, and shows that WaveCluster outperforms these methods in both efficiency and clustering quality.

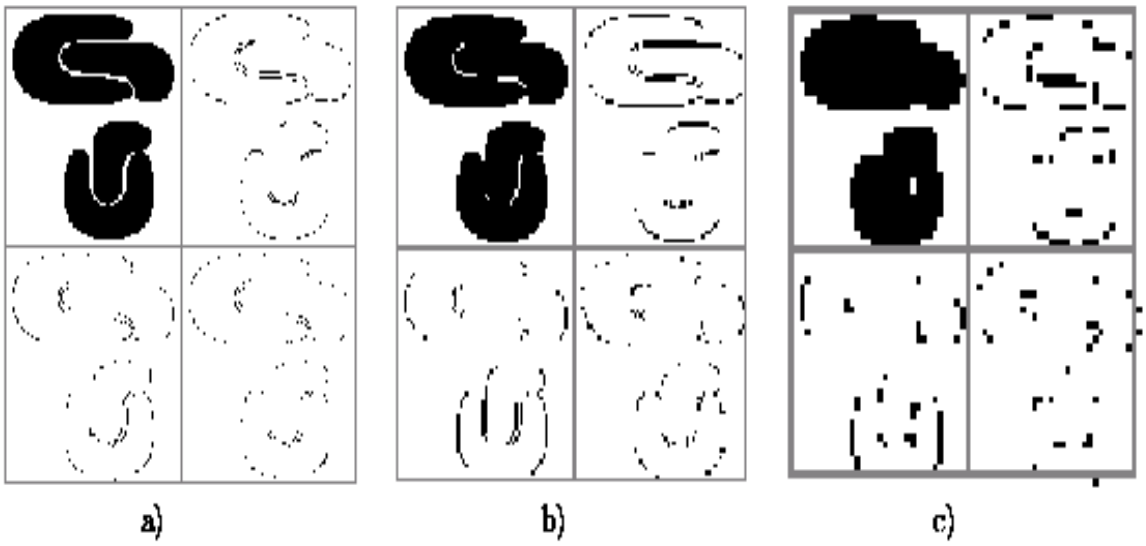


Figure 8.14: Multi-resolution of the feature space in Figure 8.13 at a) scale 1; b) scale 2; c) scale 3. Figure is from Sheikholeslami, Chatterjee and Zhang (1998).

8.7.3 CLIQUE: Clustering high-dimensional space

Another clustering algorithm, CLIQUE, developed by (Agrawal et al. 1998), is another integrated, density-based and grid-based clustering method. It is good for clustering high dimensional data in large databases.

Given a large set of multidimensional data points, the data space is usually not uniformly occupied by the data points. Data clustering identifies the sparse and the crowded places, and hence discovers the overall distribution patterns of the data set.

A unit is **dense** if the fraction of total data points contained in the unit exceeds an input model parameter. A cluster is a maximal set of connected dense units.

CLIQUE partitions the m-dimensional data space into non-overlapping rectangular units, identifies dense units, and finds clusters in all subspaces of the original data space, using a candidate generation method similar to the Apriori algorithm for mining association rules.

CLIQUE performs multidimensional clustering in two steps:

First, CLIQUE identifies clusters by determining dense units in all the subspaces of interests and then determining connected dense units in all subspaces of interests.

An important heuristic that CLIQUE adopts is the Apriori principle in high dimensional clustering: *If a k-dimensional unit is dense, then so are its projects in $(k-1)$-dimensional space.* That is, if any of the $(k-1)$-th units is not dense, its corresponding k-th dimensional unit cannot be a candidate dense unit. Therefore, all the candidate k-dimensional dense units can be generated from $(k-1)$-dimensional dense ones.

Second, CLIQUE generates minimal description for the clusters as follows. It first determines the maximal regions that cover a cluster of connected dense units for each cluster and then determines minimal cover for each cluster.

CLIQUE automatically finds subspaces of the highest dimensionality such that high density clusters exist in those subspaces. It is insensitive to the order of records in input and does not presume some canonical data distribution. It scales linearly with the size of input and has good scalability as the number of dimensions in the data is increased. However, the accuracy of the clustering result may be degraded at the expense of simplicity of the method.

8.8 Model-based clustering methods

Model-based clustering methods make use of certain models for the clusters and attempt to optimize the fit between the data and the model. It is often based on the assumption that the data are generated by a mixture of underlying probability distributions.

The model-based clustering methods have two major approaches: *statistical approach* and *neural network approach*.

Statistical approach

Clustering in machine learning is usually referred to as unsupervising learning or concept (clustering) formation. Most of work on concept formation adopt a probability-based approach, which uses probability measurements, such as category utility used in (Fisher 1987) and (Gennari, Langley and Fisher 1989), for clustering and represents concepts or clusters with probability descriptions.

For example, COBWEB (Fisher 1987) performs a top-down, unsupervised, incremental classification of concepts on categorical data. It uses one measure, called *category utility*, to score each newly inserted node and determine where to put in the hierarchy. The method has several limitations. First, it is based on the assumption that probability distributions on separate attributes are statistically independent of each other. This assumption is, however, not always true since correlation between attributes often exists. Moreover, the probability distribution representation of clusters makes it quite expensive to update and store the clusters. This is especially so when the attributes have a large number of values since their time and space complexities depend not only on the number of attributes, but also on the number of values for each attribute. Furthermore, the probability-based tree (such as (Fisher 1987)) that is built to identify clusters is not height-balanced for skewed input data, which may cause the time and space complexity to degrade dramatically.

Another system called CLASSIT (Gennari, Langley and Fisher 1989) for incremental clustering of continuous (or real valued) data. It stores a continuous normal distribution (i.e., mean and standard deviation) for each individual attribute in each node and uses a modified category utility measure which is an integral over continuous attributes instead of sum over discrete attributes as in COBWEB. However, it suffers the similar problem as COBWEB and thus it is not suitable for clustering large database data.

AutoClass (Cheeseman and Stutz 1996) implements a Bayesian clustering method based on mixture models. It uses the Bayesian statistical analysis to estimate the number of clusters.

Neural network approach

The best known neural network approach in clustering is SOM (self-organizing feature map) proposed by Kohonen in 1981. It can be viewed as a nonlinear projection from an m-dimensional input space onto a lower-order (typically 2-dimensional) regular lattice of cells. Such a mapping is used to identify clusters of elements that are similar (in an Euclidean sense) in the original space.

8.9 Outlier analysis

Very often, there exist data objects which do not comply with the general behavior or models of the data. Such sets of data objects, which are grossly different from or inconsistent with the remaining set of data, are called *outliers* of the data set.

Outliers could be caused by measurement or execution error or by inherent data variability. For example, the display of a person's age as -999 could be caused by a program default setting of an unrecorded age. However, the salary of the chief executive offier of a company could naturally stand out as an outlier among the salaries of the employees in a company.

Many data mining algorithms are trying to eliminate outliers or minimizing the influence of outliers. However, "one person's noise could be another person's signal". In many cases, the outliers themselves might be of particular interest to a user. Thus, outlier detection and analysis is an interesting data mining task.

Outlier mining has wide applications. It can be used in fraud detection for finding unusual usage of credit cards or tele-communication services, in customized marketing for finding spending behavior of extremely rich or poor

people, or in medical analysis for finding unusual response to certain medicine or treatment, etc.

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

The problem of defining outlier is nontrivial. If a regression model is used for data modeling, analysis of residuals can give good estimation for data "extremeness". The task become tricky when finding outlier in time series, as they might be hidden in trend, seasonal, or other cyclic changes. When multi-dimensional data is analyzed, not any particular one but a combination of dimension values might be extreme. Also, defining outliers in categorical data requires separate consideration.

We will examine in detail the issues for defining and mining outliers.

The most obvious and often quite effective ways for outlier detection are *data visualization methods*. Human eyes are very fast and effective at noticing data inconsistencies. However, this does not apply to data containing cyclic plots where apparently outlying values could be perfectly valid values in reality. It will also encounter difficulties to detect outliers with many categorical attributes or data of high dimensionality since human eyes are only good at visualizing numerical data of two to three dimensions.

The computer-based outlier detection methods can be categorized into three approaches: *statistical approach*, *distance-based approach*, and *deviation analysis approach*. Notice also that many *clustering algorithms* discard outliers as noise, however, they can be modified to have outlier detection as a byproduct of their execution.

8.9.1 Statistical approach for outlier detection

The statistical approach assumes the model underlying distribution that generates data set (e.g., normal distribution) and then identifies outliers using a test called *discordancy test*. The test construction requires knowledge about parameters of the data set, such as data distribution and knowledge of distribution parameters, such as mean, variance, and the number of expected outliers.

A statistical discordancy test examines two hypotheses: a *working hypothesis* and an *alternative hypothesis*.

The working hypothesis is retained if there is no statistically significant evidence supports its rejection. The working hypothesis H is a statement that the whole data set comes from an initial probability model F, i.e.,

$$H : o_j \in F, where\ j = 1, 2, \ldots, n.$$

Statistical test is performed to see whether an object o_i is significantly large (or small) in relation to the distribution complying with the model F. If o_i is shown to be discordant at the level of test, then it is not reasonable to believe that o_i comes from F and an alternative model that o_i comes from another model G is adopted. The result is very much dependent on which F model is chosen because o_i might be an outlier under one model and a perfectly valid value under the other.

The form of alternative distribution is also very important in determining the power of the test, i.e. the probability that working hypothesis is rejected when o_i is really an outlier. There are several forms of alternative hypotheses.

- **Inherent alternative**. In this case, the working hypothesis that all elements come from distribution F is rejected in favor of an alternative hypothesis that all the observations arise from another distribution G:

  $$\overline{H} : o_j \in G, where\ j = 1, \ldots, n.$$

  F and G may be different distributions or differ only in parameters of the same distribution. But there are constraints on the form of G distribution in that it must have potential to give outliers. For example, it may have different mean, or dispersion, or a longer tail.

- **Mixture alternative**. The mixture alternative states that discordant values are not outliers in F population but contaminants from some other population. In this case the alternative hypothesis is:

  $$\overline{H} : o_j \in (1 - \lambda)F + \lambda G, where\ j = 1, \ldots, n.$$

- **Slippage alternative**. This alternative states that all the observations apart from some prescribed small number arise independently from the initial model F with parameters μ and σ^2 while the remaining elements are independent observations from a modified version of F in which the parameters have been shifted.

Different test statistics have been proposed which allow to accept or reject a working hypothesis. Choice is to be made in regard to which test to prefer in a particular situation. Assuming that some statistic T has been chosen for discordancy testing and value of the statistic for element o_i is v, distribution of T is built to find whether value v is discordant. Significant probability $SP(v) = Prob(T > v)$ is evaluated. If $SP(v)$ is sufficiently small, then o_i is discordant and the working hypothesis is rejected.

In case of multiple outliers there are two basic types of procedures for detecting outliers:

- *Block procedures.* In this case either all suspect elements treated as outliers or all accepted as consistent.
- *Consecutive (or sequential) procedures.* A procedure, called *inside-out*, is more accepted. Its main idea is that the least outlier is tested first. If it is found to be outlier, then all more extreme values are also outliers; otherwise, the next more extreme element is tested, and so on.

A major drawback of the statistical approach is that most tests are for single attributes, but many data mining problems require to find outliers at multidimensional space. Moreover, the statistical approach requires the knowledge about parameters of the data set, such as the data distribution. However, in many cases, data distribution may not be known. Statistical methods do not guarantee that all outliers will be found for the cases where no specific test was developed or observed distribution cannot be adequately modeled with any standard distribution.

8.9.2 Distance-based outlier detection

To counter main limitations imposed by statistical methods, the notion of distance-based (DB) outlier is introduced in (Knorr and Ng 1997).

A distance-based (DB) outlier is defined as follows.

> An object o in a dataset T is a distance-based outlier with parameters p and d, i.e., $DB(p, d)$, if at least a fraction p of the objects in T lie at a distance greater than d from o.

Distance-based outlier generalizes other notions provided by numerous discordancy tests and replaces many of these tests with a single algorithm detecting only outliers of the proposed types. It avoids a lot of computation associated with fitting the observed distribution into some standard distribution and choosing discordancy tests. Therefore, a distance-based outlier is also called a *unified outlier*, or *UO-outlier*.

Based on this notion of outlier, one can show that for many discordancy tests if an object o is an outlier according to a specific discordancy test, o is also a $DB(p, d)$ outlier for some suitably defined p and d. For example, if for a normal distribution observations that lie 3 or more standard deviations from the mean considered to be outliers, then this definition can be unified by a $DB(0.9988, 0.13\sigma)$-outlier.

Several efficient algorithms for mining distance-based outliers have been developed. They are outlined as follows.

- *index-based algorithm.* The index-based algorithm uses multidimensional indexing structures, such as R-trees or k-d trees, to search for neighbors of each object o in a dataset within radius d around that object. As soon as $m + 1$ neighbors are found, where m is the maximum number of objects within the d-neighborhood of an outlier, it is clear that object o is not an outlier. This algorithm has the worst case complexity of $O(k * N^2)$, where k is the dimensionality and N is the number of items in the dataset, and it scales well as k grows. But this complexity evaluation takes into account search time only, while building an index itself can make this approach uncompetitive.
- *nested-loop algorithm.* The nested-loop algorithm has the same computational complexity as the index-based algorithm but it avoids index structure construction and tries to minimize the number of I/O's. It divides the memory buffer space into two halves and the dataset into several logical blocks. By carefully choosing the order of loading the blocks into different halves, I/O efficiency can be achieved.
- *cell-based algorithm.* To avoid N^2 computational complexity, a cell-based algorithm is developed for memory-resident datasets, with complexity $O(c^k + N)$, where c is some constant depending on the number of cells, and k is dimensionality. In this method, the data space is partitioned into cells with a side length equal to $\frac{d}{2\sqrt{k}}$. Every cell has two layers surrounding it, the first is one cell thick and the second is $2\sqrt{k}$ cells thick rounded up to the closest integer. The algorithm counts outliers on a cell-by-cell rather than object-by-object basis. The

algorithm counts number of items in the current cell itself, in the cell and the first layer together, and in the cell and both layers together. Some items in the current cell can be outliers only if the total number of items in the cell and the first layer is less than or equal to the maximum number M of outliers that can be in the d-neighborhood of an outlier. If this condition does not hold, then all the items in the cell can be removed from further investigation as they cannot be outliers. If the total number of the items in the cell and both layers is less than or equal to M, then all items in a cell are outliers; and if it is more than M, then only some of the items in the cell are outliers. To detect these outliers object-by-object processing is used again and for every object in the cell objects from the second layer are checked for being d-neighbors. Only those objects in the cell having less than M neighbors in both first and second layers are considered to be outliers.

A variation to the algorithm is linear with respect to N and guarantees that no more than 3 passes over the dataset are required. It works for large, disk-resident datasets. The cell-based algorithm does not scale well for high dimensions.

Distance-based outlier requires user to set both p and d parameters for testing and checking that every outlier discovered by the algorithm is a "real" outlier. Looking for the suitable parameters can involve many iterations.

8.9.3 Deviation-based outlier detection

(Arning, Agrawal and Raghavan 1996) introduced a notion of outliers that is different from both statistical and distance-based methods as it does not require defining distance measure between data elements. It introduces a linear algorithm for deviation (outlier) detection using implicit redundancy of the data. It defines sequential exception as an element observed in a series of similar data and disturbing this series.

The process of deviation detection described simulates the way people can distinguish unusual object after seeing a series of similar objects and learning their main characteristics. The authors introduce such terms as exception set, dissimilarity function, cardinality function, smoothing factor. An exception set is defined as the subset of items whose removal results in the greatest reduction of the dissimilarity of the residual set, which is equivalent to the greatest reduction in Kolmogorov complexity for the amount of data discarded. Dissimilarity function can be any function that returns a low value if the elements of the set are similar to each other and a higher value if the elements are dissimilar and does not require a metrical distance between the items. Cardinality function can be any function whose value is greater for a subset of the given set with a higher cardinality. The smoothing factor is defined for any subset of the given set and indicates how much the dissimilarity can be reduced by removing the subset from the set. Finding an exception set can be NP-hard and this motivated the definition of the sequential exception problem. Instead of finding dissimilarity of the current subset and the complementary set the algorithm selects a sequence of subsets from the set and for every subset it finds the dissimilarity difference with the preceding subset. Several iterations with random order of the subsets in a sequence are performed to alleviate possible impact of the input order on the results. For every iteration the element with the maximal value of dissimilarity function is selected. The maximum dissimilarity among all iterations scaled by the cardinality function gives exception (outlying) set. The paper gives an example showing that the effectiveness of the algorithm depends on the dissimilarity function used, and the task of the function definition is complicated by the fact that the nature of exception is not known in advance. The option of looking for a universal dissimilarity function was rejected by the authors after working with some real life databases. The algorithm has linear complexity $O(N)$, where N is the number of items in the input, provided that the number of iterations is not very big and dissimilarity function is such that for the next subset in the sequence it can be computed incrementally from the function used for the previous subset.

(Sarawagi, Agrawal and Megiddo 1998) uses OLAP data cubes to identify regions of anomalies in large multi-dimensional data. To make the process more efficient outlier detection is overlapped with cube computation, the algorithm replaces hypothesis-driven exploration, when an analyst is simply looking at the data at different levels of aggregation to discover an anomaly, with discovery-driven exploration, where indicators of exceptions are pre-computed comparing real data in a cube cell with the anticipated value. The computation of the anticipated value takes into account the cell's position in the data cube and trends along different dimensions the cell belongs to. The method allows to find anomalies at different levels of aggregation what is very difficult to do manually when search space is very large because of the number of dimension and several level deep hierarchies in every dimension. The algorithm considers a value in a cell of a data cube to be an exception if it is significantly different from the value anticipated based on a statistical model.

Anticipated value in a cell is considered to be a function of all aggregates computed using the cell value. If some

dimension has hierarchy defined on it, the cell value also depends on its parents along the hierarchies. A cell value is considered to be an exception, if its standardized residual, calculated as absolute difference between actual and anticipated values of the cell normalized by the normal deviation, is more than some threshold.

8.10 Summary

- A **cluster** is a collection of data objects that are *similar* to one another within the same cluster and are *dissimilar* to the objects in other clusters. The process of grouping a set of physical or abstract objects into classes of *similar* objects is called **clustering**.
- Cluster analysis has wide **applications** including market or customer segmentation, pattern recognition, biological studies, spatial data analysis, Web document classification, and many others. Cluster analysis can be used as a stand-alone data mining tool to gain insight into the data distribution, or serve as a preprocessing step for other data mining algorithms operating on the detected clusters.
- The quality of clustering can be assessed based on a measure of **dissimilarity** of objects, which can be computed for **various types of data**, including *interval-scaled*, variables, *binary* variables, *nominal, ordinal*, and *ratio-scaled* variables, or combinations of these variable types.
- Clustering is a dynamic field of research in data mining, with a large number of clustering algorithms developed. These algorithms can be **categorized** into *partitioning methods, hierarchical methods, density-based methods, grid-based methods*, and *model-based methods*.
- **A partitioning method** first creates an initial k partition, where k is the number of partitions to construct, then it uses an *iterative relocation technique* which attempts to improve the partitioning by moving objects from one group to another. Typical partition methods include k-means, k-medoids, CLARANS, and their improvements.
- **A hierarchical method** creates a hierarchical decomposition of the given set of data objects. The method can be classified as being either *agglomerative* (*bottom-up*) or *divisive* (*top-down*), based on how the hierarchical decomposition is formed. To compensate the rigidity of merge or split, hierarchical agglomeration often integrates other clustering techniques, such as iterative relocation. Typical such methods include BIRCH, CURE, and Chameleon.
- **A density-based method** cluster objects based on the notion of density. It either grows clusters according to density of neighborhood objects (such as in DBSCAN) or according to some density function (such as in DENCLUE). Typical density-based method include DBSCAN, OPTICS, and DENCLUE.
- **A grid-based method** first quantizes the object space into a finite number of cells which form a grid structure, and then performs clustering on the grid structure. STING is a typical example of a grid-based method based on statistical information stored in grid cells. CLIQUE and Wave-Cluster are two clustering algorithms which are both grid-based and density-based.
- **A model-based method** hypothesizes a model for each of the clusters and finds the best fit of the data to that model. Typical model-based methods include statistical approach, such as AutoClass, COBWEB and CLASSIT, and neural network approach, such as SOM.
- One person's noise could be another person's signal. **Outlier detection and analysis** is very useful for fraud detection, customized marketing, medical analysis, and many other tasks. Typical computer-based outlier analysis methods include *statistical approach, distance-based approach*, and *deviation analysis approach*.

Exercises

1. What is clustering? How is it different from classification?
2. Given are the following measurements for the variable *age*:

   18, 22, 25, 42, 28, 43, 33, 35, 56, 28.

   Standardize the variable by the following:

(a) Compute the mean absolute deviation of *age*.

(b) Compute the z-score for the first four measurements.

3. Given two objects represented by the following 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 $q = 3$.

4. Table 8.3 contains the attributes *name, gender, trait-1, trait-2, trait-3*, and *trait-4*, where *name* is an object-id, *gender* is a symmetric attribute, and the remaining *trait* attributes are asymmetric, describing personal traits of individuals who desire a penpal. Suppose that a service exists which attempts to find pairs of compatible penpals.

name	gender	trait-1	trait-2	trait-3	trait-4
Kevan	M	N	P	P	N
Caroline	F	N	P	P	N
Erik	M	P	N	N	P
⋮	⋮	⋮	⋮	⋮	⋮

Table 8.3: A relational table containing binary attributes for a penpal service.

   For asymmetric attribute values, let the values Y and P be set to 1, and the value N be set to 0.

   Suppose that the distance between objects (potential penpals) is computed based only on the asymmetric variables.

   (a) Show the contingency matrix for each pair given Kevan, Caroline, and Erik.

   (b) Compute the *simple matching coefficient* for each pair.

   (c) Compute the *Jaccard coefficient* for each pair.

   (d) Who do you suggest would make the best pair of penpals? Which pair of individuals would be the least compatible?

   (e) Suppose that we are to include the symmetric variable *gender* in our analysis. Based on the Jaccard coefficient, who would be the most compatable pair, and why?

5. Clustering has been popularly recognized as an important data mining task with broad applications. Give one application example of the following cases:

   (a) An application which takes clustering as a major data mining function.

   (b) An application which takes clustering as a preprocessing tool for data preparation for other data mining tasks.

6. Briefly describe the following approaches to clustering methods: paritioning methods, hierarchical methods, density-based methods, grid-based methods, and model-based methods. Give examples in each case.

7. Human eyes are fast and effective at judging the quality of clustering methods for two-dimensional data. Can you design a data visualization method which may help humans visualize data clusters and judge the clustering quality for three-dimensional data? What about for even higher dimensional data?

8. Give an example of how specific clustering methods may be integrated, e.g., where one clustering algorithm is used as a preprocessing step for another.

9. Data cubes and multi-dimensional databases contain categorical, ordinal and numerical data in hierarchical or aggregate forms. Based on what you have learned about the clustering methods, design a clustering method which finds clusters in large data cubes effectively and efficiently.

10. Suppose that you are to allocate a number of Automatic Teller Machines (ATMs) in a given region so as to satisfy a number of constraints. Households or places of work may be clustered so that typically one ATM is assigned per cluster. The clustering, however, may be constrained by factors involving the location of bridges, rivers, and highways which can affect ATM accessibility. Additional constraints may involve limitations on the number of ATM's per district forming the region. Given such constraints, how can clustering algorithms be modified to allow for constraint-based clustering?

Bibliographic notes

Clustering methods have been introduced in several textbooks, such as Jain and Dubes (1988) [JD88], Kaufman and Rousseeuw (1990) [KR90]. The methods for combining different variables into a single dissimilarity matrix was first proposed by Ducker et al. (1965?) [?] and later extended by Kaufman and Rousseeuw (1990) [KR90].

For partitioning methods, the k-means algorithm was first introduced by MacQueen (1967) [Mac67], the k-medoids algorithms such as PAM and CLARA were proposed by Kaufman and Rousseeuw (1990) [KR90]. The EM (Expectation Maximization) algorithm was introduced by Lauritzen (1995), and the k-mode algorithm was proposed by Huang (1998) [Hua98]. Ng and Han (1994) [NH94] proposed an algorithm called CLARANS which improves the quality and efficiency of the sampling-based k-medoid clustering method based upon randomized search. Ester *et al.* (1995) [EKX95] propose techniques for further improvement of the performance of the CLARANS algorithm using efficient spatial access methods, such as R*-tree and focusing techniques.

For hierarchical methods, agglomerative hierarchical clustering, represented by AGNES, and divisive hierarchical clustering, represented by DIANA, were introduced by Kaufman and Rousseeuw (1990) [KR90]. An interesting direction for improving the clustering quality of hierarchical clustering methods is to integrate hierarchical clustering with distance-based clustering, iterative relocation, or other nonhierarchical clustering methods. Typical studies in this direction include BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies), proposed by Zhang, Ramakrishnan, and Livny (1996) [ZRL96], CURE (Clustering Using REpresentatives), proposed by Guha, Rastogi, and Shim (1998) [GRS98], ROCK for clustering categorical attributes, proposed by Guha, Rastogi, and Shim (1999) and CHAMELEON by Karypis, Han, and Kumar (1999) [KHK99].

For density-based clustering methods, Ester et al. (1996) [EKSX96] proposed a density-based DBSCAN (Density-Based Spatial Clustering of Applications with Noise) algorithm, and Ankerst et al. (1999) [ABKS99] developed a cluster ordering method, called Optics (Ordering Points To Identify the Clustering Structure). Hinneburg and Keim (1998) [HK98] proposed DENCLUE, a DENsity-based CLUstEring method based on a set of density distribution functions.

Grid-based clustering methods have been studied recently, with a few interesting algorithms proposed. STING (STatistical INformation Grid) is a grid-based multi-resolution approach proposed by Wang, Yang, and Muntz (1997) [WYM97]. WaveCluster is a multi-resolution clustering approach which transforms the original feature space by wavelet transform, developed by Sheikholeslami, Chatterjee and Zhang (1998) [SCZ98]. CLIQUE, developed by (Agrawal et al. 1998), is an integrated, density-based and grid-based clustering method for clustering high dimensional data.

Typical model-based methods include the statistical clustering approach and the neural network approach. Works of the statistical clustering approach include AutoClass by Cheeseman and Stutz (1996) [CS96], COBWEB by Fisher (1987) , CLASSIT by Gennari, Langley and Fisher (1989) [GLF89]. Studies of the neural network approach include SOM (self-organizing feature map) by Kohonen (1982) [Koh82].

Outlier detection and analysis can be categorized into three approaches: *statistical approach*, *distance-based approach*, and *deviation analysis approach*. The statistical approach has been studied in Distance-based (DB) outlier, as introduced by Knorr and Ng (1997) [KN97, KN98]. For the deviation analysis approach, Arning, Agrawal and Raghavan (1996) introduced a linear algorithm for outlier detection using implicit redundancy of the data. Sarawagi, Agrawal, and Megiddo (1998) introduced a discovery-driven method to identify regions of anomalies in large multi-dimensional data in OLAP data cubes.

Bibliography

[AAR96] A. Arning, R. Agrawal, and P. Raghavan. A linear method for deviation detection in large databases. In *Proc. 1996 Int. Conf. Data Mining and Knowledge Discovery (KDD'96)*, pages 164–169, Portland, Oregon, August 1996.

[ABKS99] M. Ankerst, M. Breunig, H.-P. Kriegel, and J. Sander. Optics: Ordering points to identify the clustering structure. In *Proc. 1999 ACM-SIGMOD Int. Conf. Management of Data*, Philadelphia, PA, June 1999.

[AGGR98] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data*, pages 94–105, Seattle, Washington, June 1998.

[BFR98] P. Bradley, U. Fayyad, and C. Reina. Scaling clustering algorithms to large databases. In *Proc. 4th Int. Conf. Knowledge Discovery and Data Mining (KDD'98)*, pages 9–15, New York, NY, August 1998.

[BKSS90] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *Proc. 1990 ACM-SIGMOD Int. Conf. Management of Data*, pages 322–331, Atlantic City, NJ, June 1990.

[CS96] P. Cheeseman and J. Stutz. Bayesian classification (AutoClass): Theory and results. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 153–180. AAAI/MIT Press, 1996.

[EKSX96] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases. In *Proc. 2nd Int. Conf. Knowledge Discovery and Data Mining (KDD'96)*, pages 226–231, Portland, Oregon, August 1996.

[EKX95] M. Ester, H.-P. Kriegel, and X. Xu. Knowledge discovery in large spatial databases: Focusing techniques for efficient class identification. In *Proc. 4th Int. Symp. Large Spatial Databases (SSD'95)*, pages 67–82, Portland, Maine, August 1995.

[Fis87] D. Fisher. Improving inference through conceptual clustering. In *Proc. 1987 AAAI Conf.*, pages 461–465, Seattle, Washington, July 1987.

[Fis95] D. Fisher. Optimization and simplification of hierarchical clusterings. In *Proc. 1st Int. Conf. Knowledge Discovery and Data Mining (KDD'95)*, pages 118–123, Montreal, Canada, Aug. 1995.

[GLF89] J. Gennari, P. Langley, and D. Fisher. Models of incremenral concept formation. *Artificial Intelligence*, 40:11–61, 1989.

[GRS98] S. Guha, R. Rastogi, and K. Shim. Cure: An efficient clustering algorithm for large databases. In *Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data*, pages 73–84, Seattle, Washington, June 1998.

[GRS99] S. Guha, R. Rastogi, and K. Shim. Rock: A robust clustering algorithm for categorical attributes. In *Proc. 1999 Int. Conf. Data Engineering*, pages 512–521, Sydney, Australia, March 1999.

[HK98] A. Hinneburg and D. A. Keim. An efficient approach to clustering in large multimedia databases with noise. In *Proc. 1998 Int. Conf. Knowledge Discovery and Data Mining (KDD'98)*, pages 58–65, New York, NY, August 1998.

[Hua98] Z. Huang. Extensions to the k-means algorithm for clustering large data sets with categorical values. *Data Mining and Knowledge Discovery*, 2:283–304, 1998.

[JD88] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Printice Hall, 1988.

[KHK99] G. Karypis, E.-H. Han, and V. Kumar. CHAMELEON: A hierarchical clustering algorithm using dynamic modeling. *COMPUTER*, 32:68–75, 1999.

[KN97] E. Knorr and R. Ng. A unified notion of outliers: Properties and computation. In *Proc. 3rd Int. Conf. Knowledge Discovery and Data Mining (KDD'97)*, pages 219–222, Newport Beach, California, August 1997.

[KN98] E. Knorr and R. Ng. Algorithms for mining distance-based outliers in large datasets. In *Proc. 1998 Int. Conf. Very Large Data Bases*, pages 392–403, New York, NY, August 1998.

[Koh82] T. Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59–69, 1982.

[KR90] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: an Introduction to Cluster Analysis*. John Wiley & Sons, 1990.

[Lau95] S. L. Lauritzen. The EM algorithm for graphical association models with missing data. *Computational Statistics and Data Analysis*, 19:191–201, 1995.

[Mac67] J. MacQueen. Some methods for classification and analysis of multivariate observations. *Proc. 5th Berkeley Symp. Math. Statist, Prob.*, 1:281–297, 1967.

[NH94] R. Ng and J. Han. Efficient and effective clustering method for spatial data mining. In *Proc. 1994 Int. Conf. Very Large Data Bases*, pages 144–155, Santiago, Chile, September 1994.

[SAM98] S. Sarawagi, R. Agrawal, and N. Megiddo. Discovery-driven exploration of OLAP data cubes. In *Proc. Int. Conf. of Extending Database Technology (EDBT'98)*, pages 168–182, Valencia, Spain, March 1998.

[SCZ98] G. Sheikholeslami, S. Chatterjee, and A. Zhang. WaveCluster: A multi-resolution clustering approach for very large spatial databases. In *Proc. 1998 Int. Conf. Very Large Data Bases*, pages 428–439, New York, NY, August 1998.

[WYM97] W. Wang, J. Yang, and R. Muntz. STING: A statistical information grid approach to spatial data mining. In *Proc. 1997 Int. Conf. Very Large Data Bases*, pages 186–195, Athens, Greece, Aug. 1997.

[ZRL96] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: an efficient data clustering method for very large databases. In *Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data*, pages 103–114, Montreal, Canada, June 1996.

Contents

Contents