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 ⟨*city, province_or_state, country*⟩. 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 `counts` 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. □

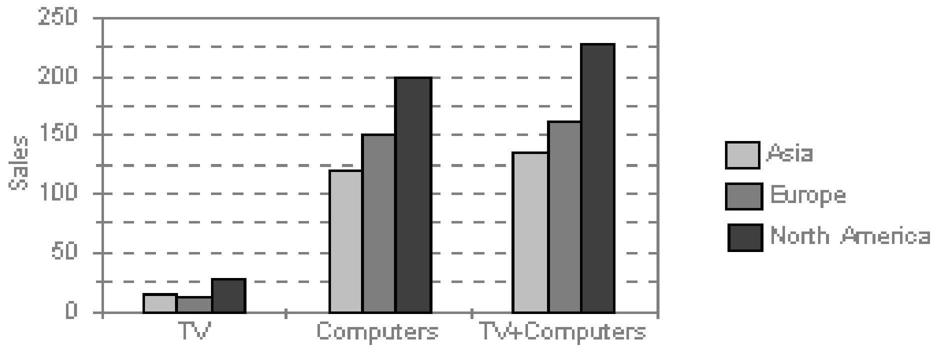

Figure 5.1: Bar chart representation of the sales in 1997.

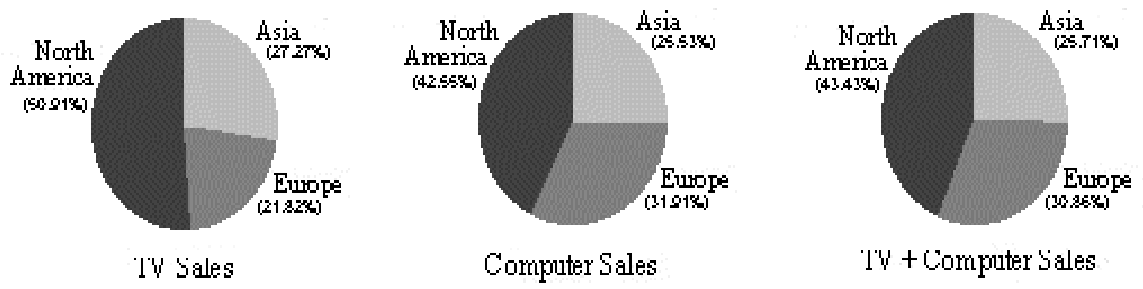

Figure 5.2: Pie chart representation of the sales in 1997.

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.

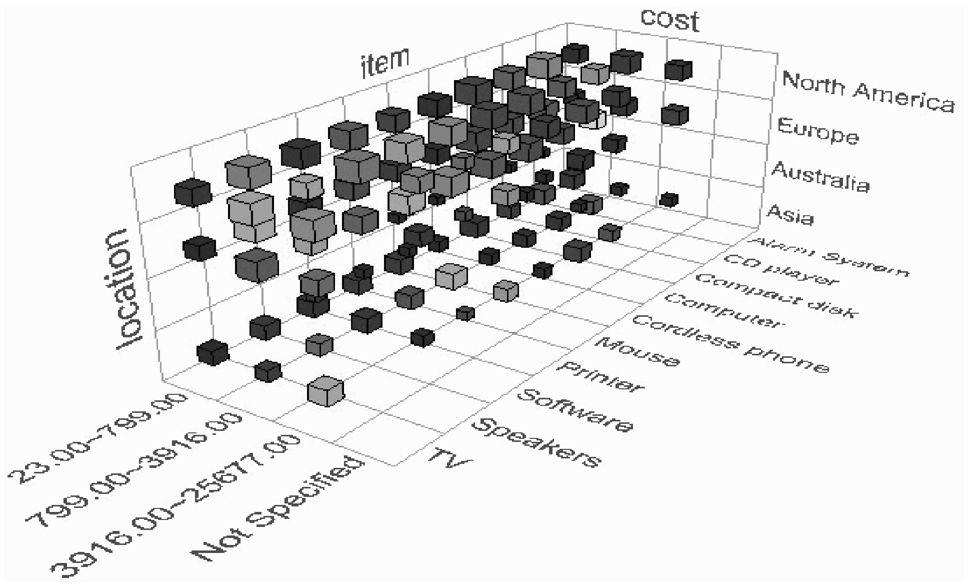

Figure 5.3: A 3-D Cube view representation of the sales in 1997.

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

$$\begin{aligned}&\forall X,\ item(X) = \text{“}computer\text{”} \Rightarrow \\ &\quad (location(X) = \text{“}Asia\text{”})\ [t : 25.00\%] \vee (location(X) = \text{“}Europe\text{”})\ [t : 30.00\%] \vee \\ &\quad (location(X) = \text{“}North\_America\text{”})\ [t : 45.00\%]\end{aligned} \tag{5.3}$$

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

$$I(s_1, s_2, \ldots, s_m) \quad = \quad \Leftrightarrow\sum_{i=1}^{m} \frac{s_i}{s} log_2 \frac{s_i}{s}. \qquad (5.4)$$

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:

$$\forall X,\ graduate\_student(X) \ \Leftarrow\ birth\_country(X) = \text{“}Canada\text{”} \wedge age\_range = \text{“25\_30”} \wedge gpa = \text{“}good\text{”}\,[d : 30\%]. \tag{5.9}$$

□

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

$$\forall X,\ target\_class(X)\ \Rightarrow\ condition_1(X)[t:w_1] \vee \cdots \vee condition_n(X)[t:w_n], \qquad (5.10)$$

  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 $\{1,\dots,n\}$.

- 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 $\{1,\dots,n\}$.

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

$$\forall X,\ target\_class(X) \Leftrightarrow condition_1(X)[t:w_1, d:w_1'] \vee \cdots \vee condition_n(X)[t:w_n, d:w_n']. \qquad (5.11)$$

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.

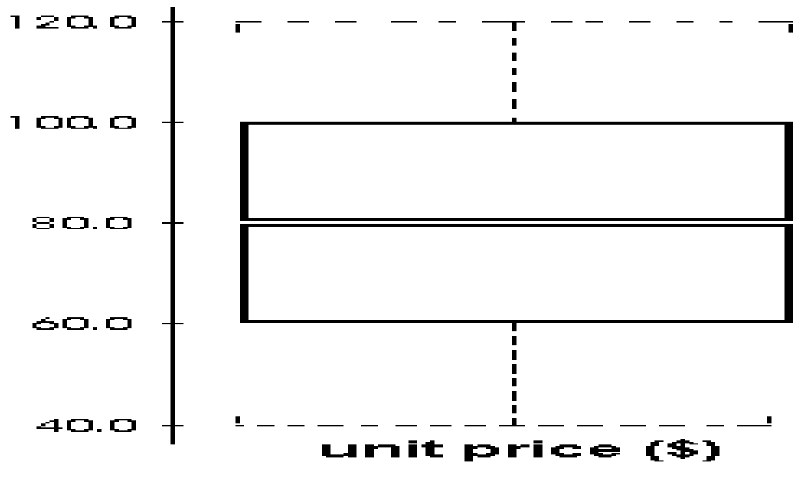

Figure 5.4: A boxplot for the data set of Table 5.11.

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

$$s^2 \quad = \quad \frac{1}{n \Leftrightarrow 1}\sum_{i=1}^{n}(x_i \Leftrightarrow \bar{x})^2 = \frac{1}{n \Leftrightarrow 1}[\sum x_i^2 \Leftrightarrow \frac{1}{n}(\sum x_i)^2] \tag{5.18}$$

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

$$f_i = \frac{i \Leftrightarrow 0.5}{n}.$$

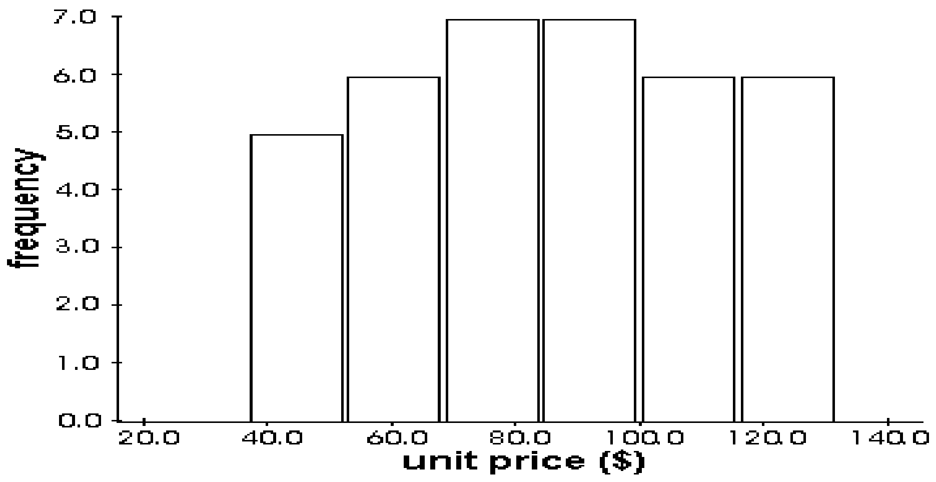

Figure 5.5: A histogram for the data set of Table 5.11.

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.

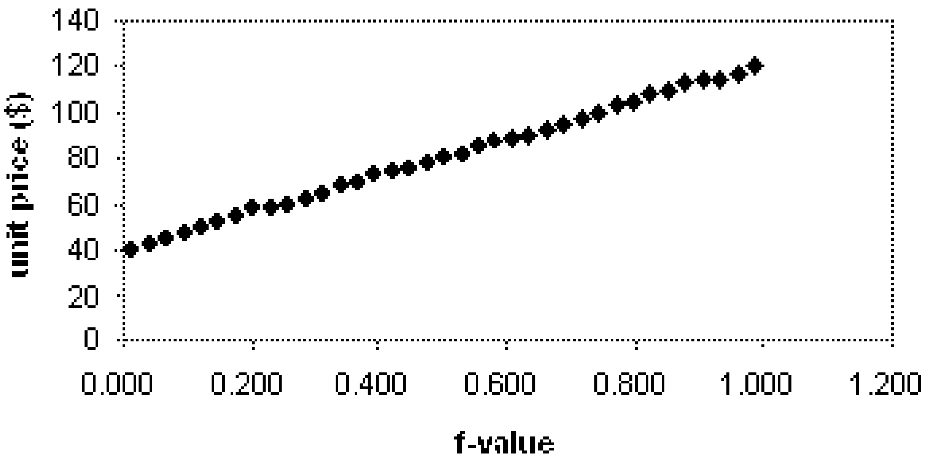

Figure 5.6: A quantile plot for the data set of Table 5.11.

- 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.

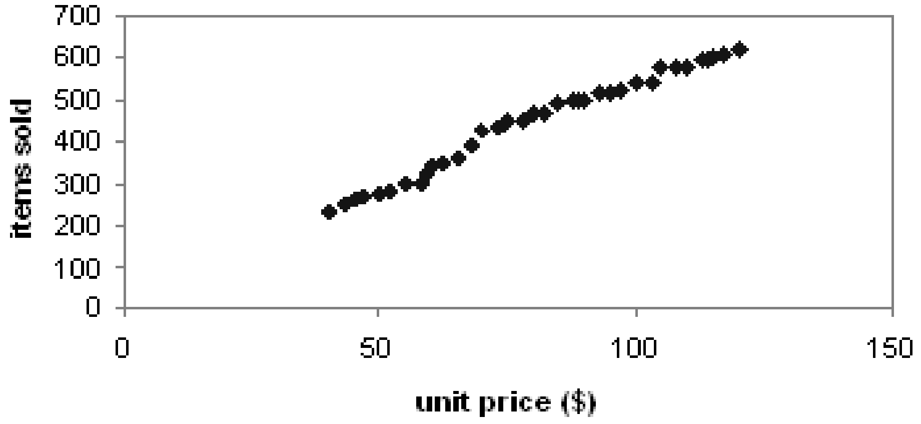

Figure 5.7: A quantile-quantile plot for the data set of Table 5.11.

- 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.

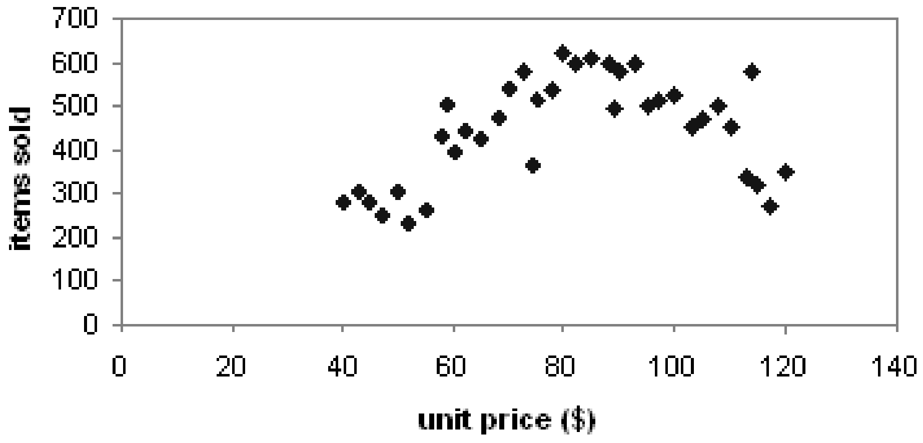

Figure 5.8: A scatter plot for the data set of Table 5.11.

- 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.

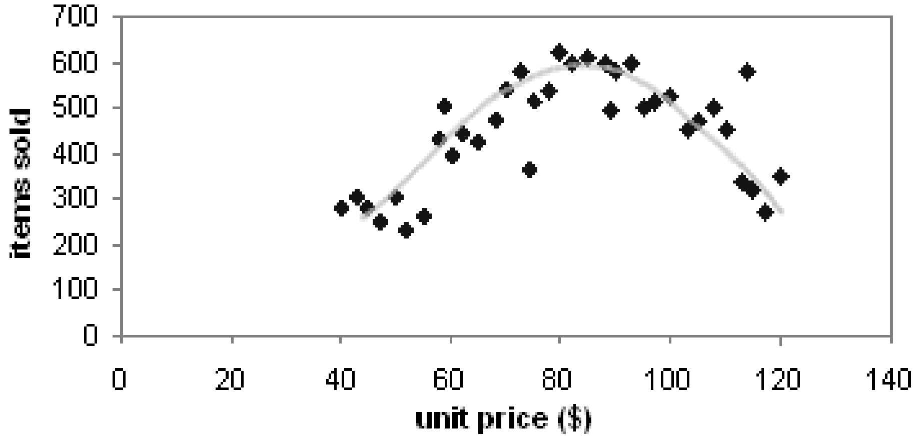

Figure 5.9: A loess curve for the data set of Table 5.11.

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.