Contents

Chapter 4

Primitives for Data Mining

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

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

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

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

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

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

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

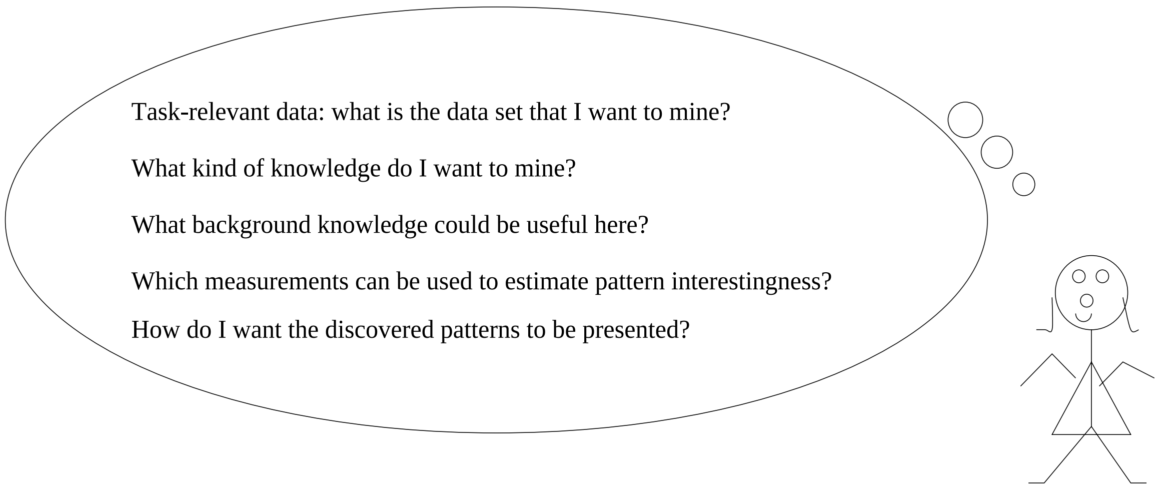

Task-relevant data: what is the data set that I want to mine?
What kind of knowledge do I want to mine?
What background knowledge could be useful here?
Which measurements can be used to estimate pattern interestingness?
How do I want the discovered patterns to be presented?

Figure 4.1: Defining a data mining task or query.

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

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

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

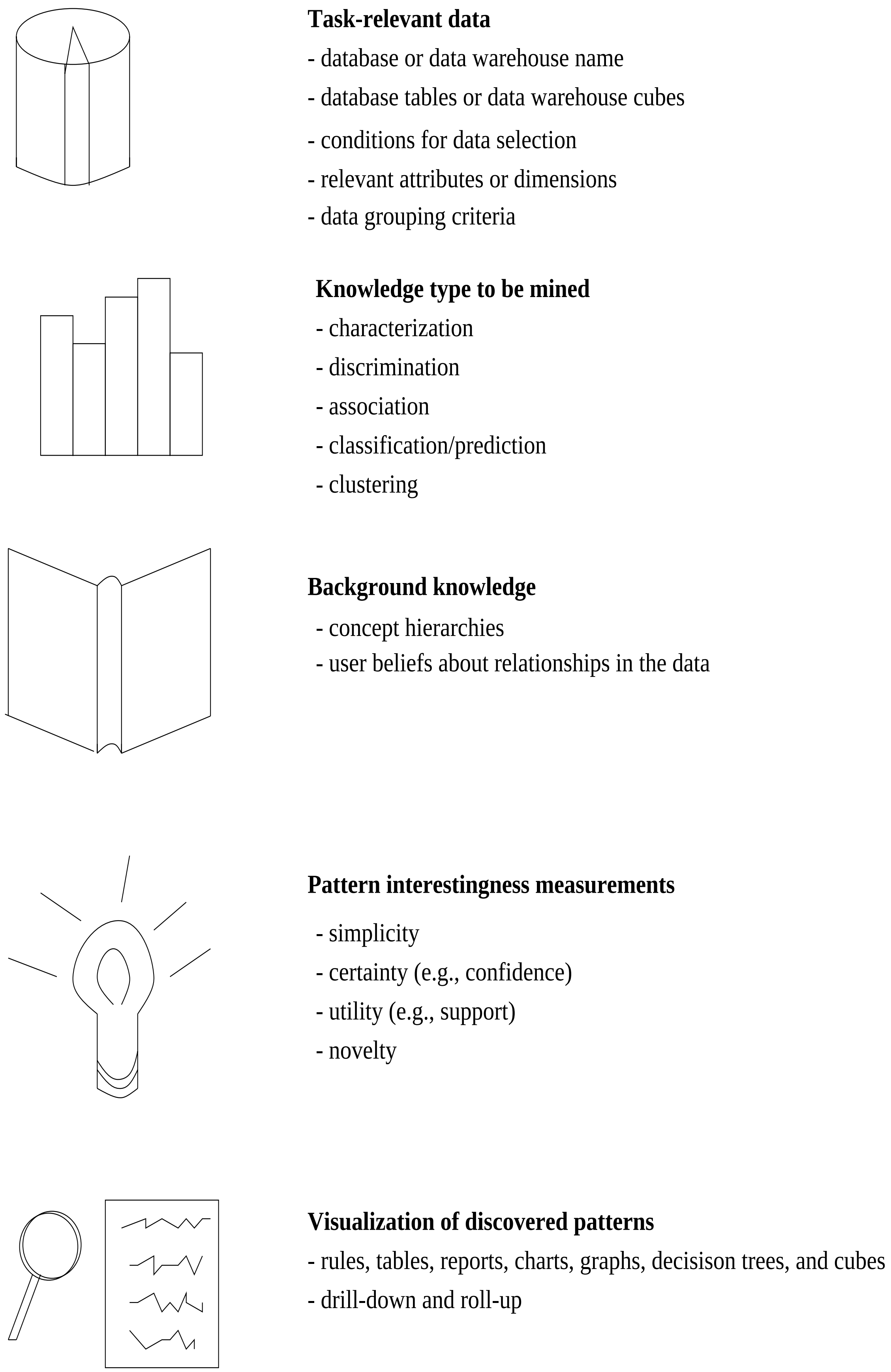
4. **interestingness measures**: These functions are used to separate uninteresting patterns from knowledge. They may be used to guide the mining process, or after discovery, to evaluate the discovered patterns. Different kinds of knowledge may have different interestingness measures. For example, interestingness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

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

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

4.1.1 Task-relevant data

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



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

Knowledge type to be mined
- characterization
- discrimination
- association
- classification/prediction
- clustering

Background knowledge
- concept hierarchies
- user beliefs about relationships in the data

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

Visualization of discovered patterns
- rules, tables, reports, charts, graphs, decisison trees, and cubes
- drill-down and roll-up

Figure 4.2: Primitives for specifying a data mining task.

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

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

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

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

In a data warehouse, data are typically stored in a multidimensional database, known as a **data cube**, which can be implemented using a multidimensional array structure, a relational structure, or a combination of both, as discussed in Chapter 2. The set of task-relevant data can be specified by condition-based data filtering, *slicing* (extracting data for a given attribute value, or "slice"), or *dicing* (extracting the intersection of several slices) of the data cube.

Notice that in a data mining query, the conditions provided for data selection can be at a level that is conceptually higher than the data in the database or data warehouse. For example, a user may specify a selection on items at *AllElectronics* using the concept "*type = home entertainment*", even though individual items in the database may not be stored according to type, but rather, at a lower conceptual, such as "*TV*", "*CD player*", or "*VCR*". A concept hierarchy on item which specifies that "*home entertainment*" is at a higher concept level, composed of the lower level concepts {*"TV", "CD player", "VCR"*} can be used in the collection of the task-relevant data.

The set of relevant attributes specified may involve other attributes which were not explicitly mentioned, but which should be included because they are implied by the concept hierarchy or dimensions involved in the set of relevant attributes specified. For example, a query-relevant set of attributes may contain *city*. This attribute, however, may be part of other concept hierarchies such as the concept hierarchy *street* < *city* < *province_or_state* < *country* for the dimension *location*. In this case, the attributes *street*, *province_or_state*, and *country* should also be included in the set of relevant attributes since they represent lower or higher level abstractions of *city*. This facilitates the mining of knowledge at multiple levels of abstraction by specialization (drill-down) and generalization (roll-up).

Specification of the relevant attributes or dimensions can be a difficult task for users. A user may have only a rough idea of what the interesting attributes for exploration might be. Furthermore, when specifying the data to be mined, the user may overlook additional relevant data having strong semantic links to them. For example, the sales of certain items may be closely linked to particular events such as Christmas or Halloween, or to particular groups of people, yet these factors may not be included in the general data analysis request. For such cases, mechanisms can be used which help give a more precise specification of the task-relevant data. These include functions to evaluate and rank attributes according to their relevancy with respect to the operation specified. In addition, techniques that search for attributes with strong semantic ties can be used to enhance the initial dataset specified by the user.

4.1.2 The kind of knowledge to be mined

It is important to specify the kind of knowledge to be mined, as this determines the data mining function to be performed. The kinds of knowledge include concept description (characterization and discrimination), association, classification, prediction, clustering, and evolution analysis.

In addition to specifying the kind of knowledge to be mined for a given data mining task, the user can be more specific and provide pattern templates that all discovered patterns must match. These templates, or **metapatterns** (also called **metarules** or **metaqueries**), can be used to guide the discovery process. The use of metapatterns is illustrated in the following example.

Example 4.2 A user studying the buying habits of *AllElectronics* customers may choose to mine *association rules* of the form

$$P(X : customer, W) \wedge Q(X, Y) \;\Rightarrow\; buys(X, Z)$$

where X is a key of the *customer* relation, P and Q are **predicate variables** which can be instantiated to the relevant attributes or dimensions specified as part of the task-relevant data, and W, Y, and Z are **object variables** which can take on the values of their respective predicates for customers X.

The search for association rules is confined to those matching the given metarule, such as

$$age(X, \text{“30\_39”}) \wedge income(X, \text{“40\_50}K\text{”}) \;\Rightarrow\; buys(X, \text{“}VCR\text{”}) \qquad [2.2\%, 60\%] \qquad (4.1)$$

and

$$occupation(X, \text{“}student\text{”}) \wedge age(X, \text{“20\_29”}) \;\Rightarrow\; buys(X, \text{“}computer\text{”}) \qquad [1.4\%, 70\%]. \qquad (4.2)$$

The former rule states that customers in their thirties, with an annual income of between 40K and 50K, are likely (with 60% confidence) to purchase a VCR, and such cases represent about 2.2% of the total number of transactions. The latter rule states that customers who are students and in their twenties are likely (with 70% confidence) to purchase a computer, and such cases represent about 1.4% of the total number of transactions. □

4.1.3 Background knowledge: concept hierarchies

Background knowledge is information about the domain to be mined that can be useful in the discovery process. In this section, we focus our attention on a simple yet powerful form of background knowledge known as *concept hierarchies*. Concept hierarchies allow the discovery of knowledge at multiple levels of abstraction.

As described in Chapter 2, a **concept hierarchy** defines a sequence of mappings from a set of low level concepts to higher level, more general concepts. A concept hierarchy for the dimension *location* is shown in Figure 4.3, mapping low level concepts (i.e., cities) to more general concepts (i.e., countries).

Notice that this concept hierarchy is represented as a set of **nodes** organized in a tree, where each node, in itself, represents a concept. A special node, all, is reserved for the root of the tree. It denotes the most generalized value of the given dimension. If not explicitly shown, it is implied. This concept hierarchy consists of four **levels**. By convention, levels within a concept hierarchy are numbered from top to bottom, starting with level 0 for the all node. In our example, level 1 represents the concept *country*, while levels 2 and 3 respectively represent the concepts *province_or_state* and *city*. The leaves of the hierarchy correspond to the dimension's raw data values (**primitive level data**). These are the most specific values, or concepts, of the given attribute or dimension. Although a concept hierarchy often defines a taxonomy represented in the shape of a tree, it may also be in the form of a general lattice or partial order.

Concept hierarchies are a useful form of background knowledge in that they allow raw data to be handled at higher, generalized levels of abstraction. Generalization of the data, or **rolling up** is achieved by replacing primitive level data (such as city names for *location*, or numerical values for *age*) by higher level concepts (such as continents for *location*, or ranges like "20-39", "40-59", "60+" for *age*). This allows the user to view the data at *more meaningful* and explicit abstractions, and makes the discovered patterns easier to understand. Generalization has an added advantage of compressing the data. Mining on a compressed data set will require fewer input/output operations and be more efficient than mining on a larger, uncompressed data set.

If the resulting data appear overgeneralized, concept hierarchies also allow specialization, or **drilling down**, whereby concept values are replaced by lower level concepts. By rolling up and drilling down, users can view the data from different perspectives, gaining further insight into hidden data relationships.

Concept hierarchies can be provided by system users, domain experts, or knowledge engineers. The mappings are typically data- or application-specific. Concept hierarchies can often be automatically discovered or dynamically refined based on statistical analysis of the data distribution. The automatic generation of concept hierarchies is discussed in detail in Chapter 3.

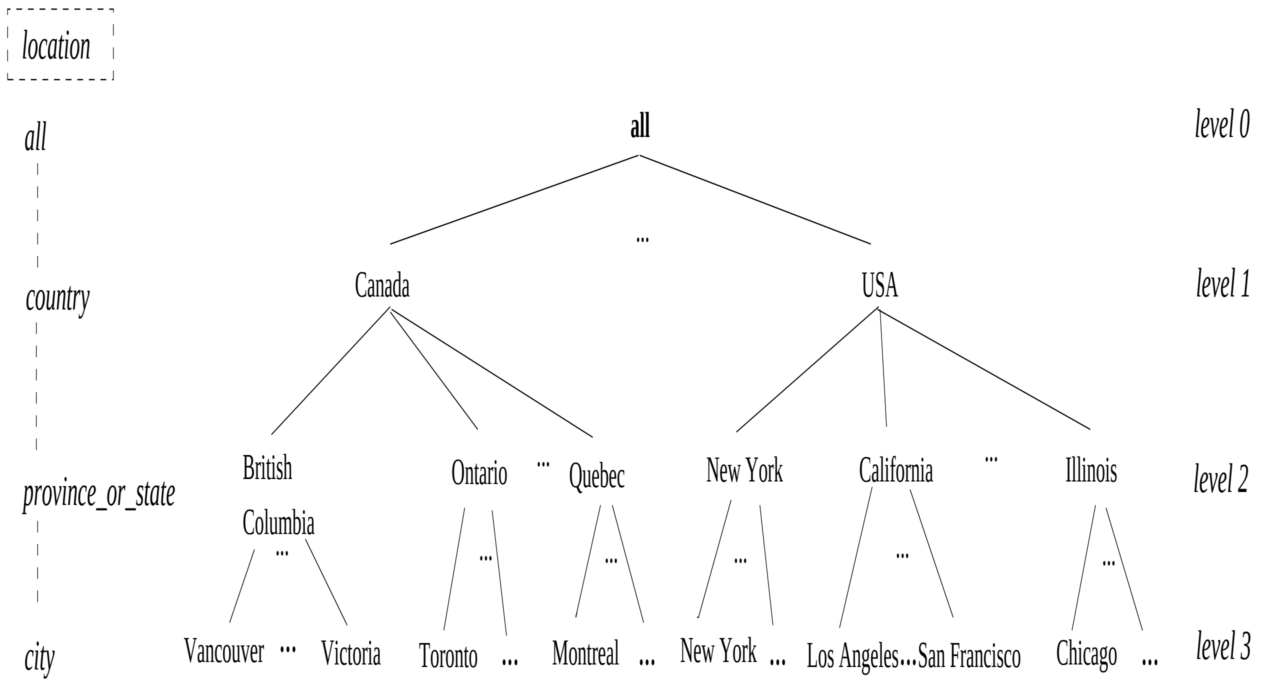


Figure 4.3: A concept hierarchy for the dimension *location*.

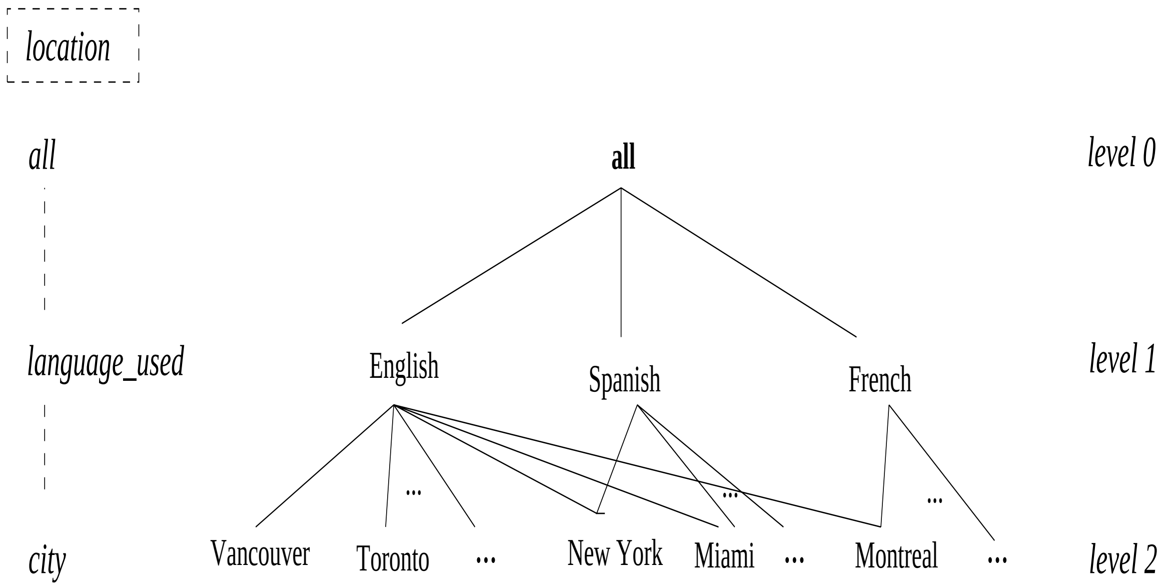


Figure 4.4: Another concept hierarchy for the dimension *location*, based on language.

There may be more than one concept hierarchy for a given attribute or dimension, based on different user viewpoints. Suppose, for instance, that a regional sales manager of *AllElectronics* is interested in studying the buying habits of customers at different locations. The concept hierarchy for *location* of Figure 4.3 should be useful for such a mining task. Suppose that a marketing manager must devise advertising campaigns for *AllElectronics*. This user may prefer to see *location* organized with respect to linguistic lines (e.g., including English for Vancouver, Montreal and New York; French for Montreal; Spanish for New York and Miami; and so on) in order to facilitate the distribution of commercial ads. This alternative hierarchy for *location* is illustrated in Figure 4.4. Note that this concept hierarchy forms a lattice, where the node "New York" has two parent nodes, namely "English" and "Spanish".

There are four major types of concept hierarchies. Chapter 2 introduced the most common types — *schema hierarchies* and *set-grouping hierarchies*, which we review here. In addition, we also study *operation-derived hierarchies* and *rule-based hierarchies*.

1. A **schema hierarchy** (or more rigorously, a *schema-defined hierarchy*) is a total or partial order among attributes in the database schema. Schema hierarchies may formally express existing semantic relationships between attributes. Typically, a schema hierarchy specifies a data warehouse dimension.

   Example 4.3 Given the schema of a relation for *address* containing the attributes *street, city, province_or_state*, and *country*, we can define a *location* schema hierarchy by the following total order:

   $$street < city < province\_or\_state < country$$

   This means that *street* is at a conceptually lower level than *city*, which is lower than *province_or_state*, which is conceptually lower than *country*. A schema hierarchy provides metadata information, i.e., data about the data. Its specification in terms of a total or partial order among attributes is more concise than an equivalent definition that lists all instances of streets, provinces or states, and countries.

   Recall that when specifying the task-relevant data, the user specifies relevant attributes for exploration. If a user had specified only one attribute pertaining to location, say, *city*, other attributes pertaining to any schema hierarchy containing *city* may automatically be considered relevant attributes as well. For instance, the attributes *street, province_or_state*, and *country* may also be automatically included for exploration. □

2. A **set-grouping hierarchy** organizes values for a given attribute or dimension into groups of constants or range values. A total or partial order can be defined among groups. Set-grouping hierarchies can be used to refine or enrich schema-defined hierarchies, when the two types of hierarchies are combined. They are typically used for defining small sets of object relationships.

   Example 4.4 A set-grouping hierarchy for the attribute *age* can be specified in terms of ranges, as in the following.

   $$\{20-39\} \subset young$$
   $$\{40-59\} \subset middle\_aged$$
   $$\{60-89\} \subset senior$$
   $$\{young,\ middle\_aged,\ senior\} \subset \mathsf{all}(age)$$

   Notice that similar range specifications can also be generated automatically, as detailed in Chapter 3. □

   Example 4.5 A set-grouping hierarchy may form a portion of a schema hierarchy, and vice versa. For example, consider the concept hierarchy for *location* in Figure 4.3, defined as *city* < *province_or_state* < *country*. Suppose that possible constant values for *country* include "*Canada*", "*USA*", "*Germany*", "*England*", and "*Brazil*". Set-grouping may be used to refine this hierarchy by adding an additional level above *country*, such as *continent*, which groups the country values accordingly. □

3. **Operation-derived hierarchies** are based on operations specified by users, experts, or the data mining system. Operations can include the decoding of information-encoded strings, information extraction from complex data objects, and data clustering.

Example 4.6 An e-mail address or a URL of the WWW may contain hierarchy information relating departments, universities (or companies), and countries. Decoding operations can be defined to extract such information in order to form concept hierarchies.

For example, the e-mail address "dmbook@cs.sfu.ca" gives the partial order, "*login-name* < *department* < *university* < *country*", forming a concept hierarchy for e-mail addresses. Similarly, the URL address "http://www.cs.sfu.ca/research/DB/DBMiner" can be decoded so as to provide a partial order which forms the base of a concept hierarchy for URLs. □

Example 4.7 Operations can be defined to extract information from complex data objects. For example, the string "Ph.D. in Computer Science, UCLA, 1995" is a complex object representing a university degree. This string contains rich information about the type of academic degree, major, university, and the year that the degree was awarded. Operations can be defined to extract such information, forming concept hierarchies. □

Alternatively, mathematical and statistical operations, such as data clustering and data distribution analysis algorithms, can be used to form concept hierarchies, as discussed in Section 3.5

4. **A rule-based hierarchy** occurs when either a whole concept hierarchy or a portion of it is defined by a set of rules, and is evaluated dynamically based on the current database data and the rule definition.

Example 4.8 The following rules may be used to categorize *AllElectronics* items as *low_profit_margin* items, *medium_profit_margin* items, and *high_profit_margin* items, where the profit margin of an item X is defined as the difference between the retail price and actual cost of X. Items having a profit margin of less than \$50 may be defined as *low_profit_margin* items, items earning a profit between \$50 and \$250 may be defined as *medium_profit_margin* items, and items earning a profit of more than \$250 may be defined as *high_profit_margin* items.

$$\begin{aligned}
low\_profit\_margin(X) &\Leftarrow price(X, P1) \wedge cost(X, P2) \wedge ((P1 - P2) < \$50)\\
medium\_profit\_margin(X) &\Leftarrow price(X, P1) \wedge cost(X, P2) \wedge ((P1 - P2) > \$50) \wedge ((P1 - P2) \leq \$250)\\
high\_profit\_margin(X) &\Leftarrow price(X, P1) \wedge cost(X, P2) \wedge ((P1 - P2) > \$250)
\end{aligned}$$

□

The use of concept hierarchies for data mining is described in the remaining chapters of this book.

4.1.4 Interestingness measures

Although specification of the task-relevant data and of the kind of knowledge to be mined (e.g, characterization, association, etc.) may substantially reduce the number of patterns generated, a data mining process may still generate a large number of patterns. Typically, only a small fraction of these patterns will actually be of interest to the given user. Thus, users need to further confine the number of uninteresting patterns returned by the process. This can be achieved by specifying interestingness measures which estimate the simplicity, certainty, utility, and novelty of patterns.

In this section, we study some objective measures of pattern interestingness. Such objective measures are based on the structure of patterns and the statistics underlying them. In general, each measure is associated with a *threshold* that can be controlled by the user. Rules that do not meet the threshold are considered uninteresting, and hence are not presented to the user as knowledge.

- **Simplicity.** A factor contributing to the interestingness of a pattern is the pattern's overall simplicity for human comprehension. Objective measures of pattern simplicity can be viewed as functions of the pattern structure, defined in terms of the pattern size in bits, or the number of attributes or operators appearing in the pattern. For example, the more complex the structure of a rule is, the more difficult it is to interpret, and hence, the less interesting it is likely to be.

  Rule length, for instance, is a simplicity measure. For rules expressed in conjunctive normal form (i.e., as a set of conjunctive predicates), rule length is typically defined as the number of conjuncts in the rule.

Association, discrimination, or classification rules whose lengths exceed a user-defined threshold are considered uninteresting. For patterns expressed as decision trees, simplicity may be a function of the number of tree leaves or tree nodes.

- **Certainty.** Each discovered pattern should have a measure of certainty associated with it which assesses the validity or "trustworthiness" of the pattern. A certainty measure for association rules of the form "$A \Rightarrow B$," is **confidence**. Given a set of task-relevant data tuples (or transactions in a transaction database) the confidence of "$A \Rightarrow B$" is defined as:

$$\text{Confidence}(A \Rightarrow B) = P(B|A) = \frac{\#\_tuples\_containing\_both\_A\_and\_B}{\#\_tuples\_containing\_A}. \quad (4.3)$$

Example 4.9 Suppose that the set of task-relevant data consists of transactions from the computer department of *AllElectronics*. A confidence of 85% for the association rule

$$buys(X, \text{"}computer\text{"}) \Rightarrow buys(X, \text{"}software\text{"}) \quad (4.4)$$

means that 85% of all customers who purchased a computer also bought software. □

A confidence value of 100%, or 1, indicates that the rule is always correct on the data analyzed. Such rules are called **exact**.

For classification rules, confidence is referred to as **reliability** or **accuracy**. Classification rules propose a model for distinguishing objects, or tuples, of a target class (say, *bigSpenders*) from objects of contrasting classes (say, *budgetSpenders*). A low reliability value indicates that the rule in question incorrectly classifies a large number of contrasting class objects as target class objects. Rule reliability is also known as **rule strength, rule quality, certainty factor**, and **discriminating weight**.

- **Utility.** The potential usefulness of a pattern is a factor defining its interestingness. It can be estimated by a utility function, such as support. The **support** of an association pattern refers to the percentage of task-relevant data tuples (or transactions) for which the pattern is true. For association rules of the form "$A \Rightarrow B$", it is defined as

$$\text{Support}(A \Rightarrow B) = P(A \cup B) = \frac{\#\_tuples\_containing\_both\_A\_and\_B}{total\_\#\_of\_tuples}. \quad (4.5)$$

Example 4.10 Suppose that the set of task-relevant data consists of transactions from the computer department of *AllElectronics*. A support of 30% for the association rule (4.4) means that 30% of all customers in the computer department purchased both a computer and software. □

Association rules that satisfy both a user-specified *minimum confidence threshold* and user-specified *minimum support threshold* are referred to as **strong association rules**, and are considered interesting. Rules with low support likely represent noise, or rare or exceptional cases.

The numerator of the support equation is also known as the rule **count**. Quite often, this number is displayed instead of support. Support can easily be derived from it.

Characteristic and discriminant descriptions are, in essence, generalized tuples. Any generalized tuple representing less than Y% of the total number of task-relevant tuples is considered noise. Such tuples are not displayed to the user. The value of Y is referred to as the **noise threshold**.

- **Novelty.** Novel patterns are those that contribute new information or increased performance to the given pattern set. For example, a data exception may be considered novel in that it differs from that expected based on a statistical model or user beliefs. Another strategy for detecting novelty is to remove redundant patterns. If a discovered rule can be implied by another rule that is already in the knowledge base or in the derived rule set, then either rule should be re-examined in order to remove the potential redundancy.

Mining with concept hierarchies can result in a large number of redundant rules. For example, suppose that the following association rules were mined from the *AllElectronics* database, using the concept hierarchy in Figure 4.3 for *location*:

$$location(X, \text{“}Canada\text{”}) \Rightarrow buys(X, \text{“}SONY\_TV\text{”}) \quad [8\%, 70\%] \tag{4.6}$$

$$location(X, \text{“}Montreal\text{”}) \Rightarrow buys(X, \text{“}SONY\_TV\text{”}) \quad [2\%, 71\%] \tag{4.7}$$

Suppose that Rule (4.6) has 8% support and 70% confidence. One may expect Rule (4.7) to have a confidence of around 70% as well, since all the tuples representing data objects for Montreal are also data objects for Canada. Rule (4.6) is more general than Rule (4.7), and therefore, we would expect the former rule to occur more frequently than the latter. Consequently, the two rules should not have the same support. Suppose that about one quarter of all sales in Canada comes from Montreal. We would then expect the support of the rule involving Montreal to be one quarter of the support of the rule involving Canada. In other words, we expect the support of Rule (4.7) to be $8\% \times \frac{1}{4} = 2\%$. If the actual confidence and support of Rule (4.7) are as expected, then the rule is considered redundant since it does not offer any additional information and is less general than Rule (4.6). These ideas are further discussed in Chapter 6 on association rule mining.

The above example also illustrates that when mining knowledge at multiple levels, it is reasonable to have different support and confidence thresholds, depending on the degree of granularity of the knowledge in the discovered pattern. For instance, since patterns are likely to be more scattered at lower levels than at higher ones, we may set the minimum support threshold for rules containing low level concepts to be lower than that for rules containing higher level concepts.

Data mining systems should allow users to flexibly and interactively specify, test, and modify interestingness measures and their respective thresholds. There are many other objective measures, apart from the basic ones studied above. Subjective measures exist as well, which consider user beliefs regarding relationships in the data, in addition to objective statistical measures. Interestingness measures are discussed in greater detail throughout the book, with respect to the mining of characteristic, association, and classification rules, and deviation patterns.

4.1.5 Presentation and visualization of discovered patterns

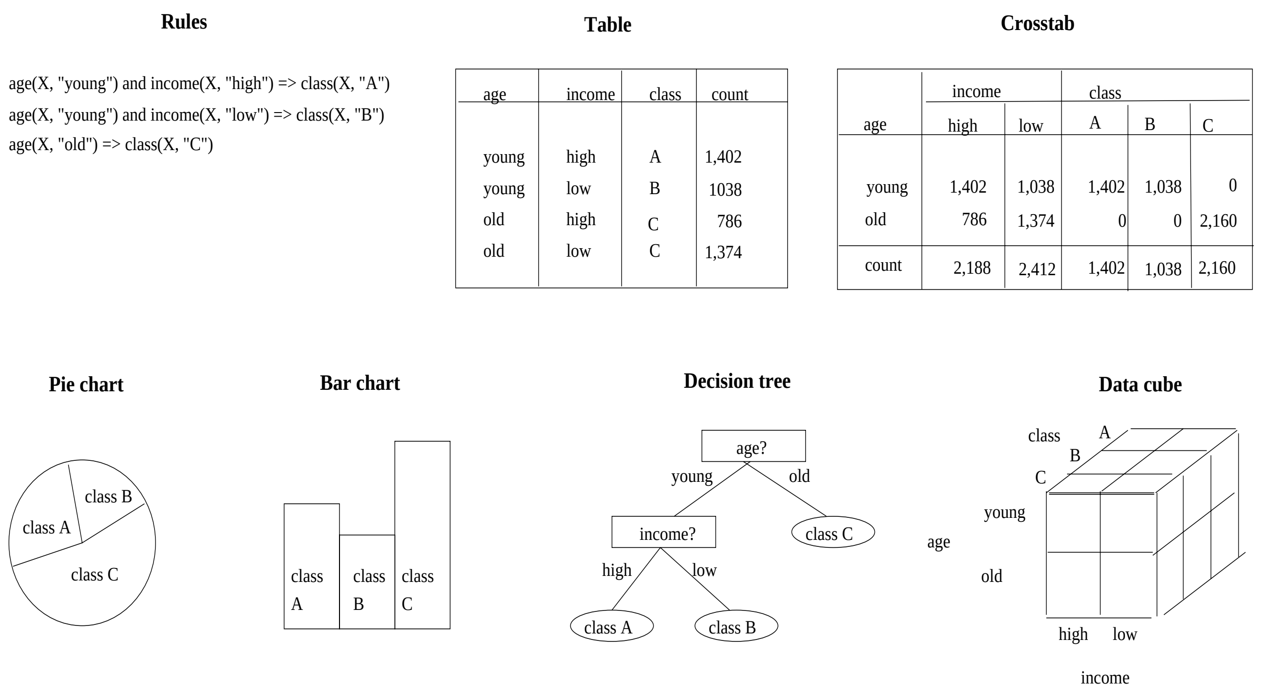
For data mining to be effective, data mining systems should be able to display the discovered patterns in multiple forms, such as rules, tables, crosstabs, pie or bar charts, decision trees, cubes, or other visual representations (Figure 4.5). Allowing the visualization of discovered patterns in various forms can help users with different backgrounds to identify patterns of interest and to interact or guide the system in further discovery. A user should be able to specify the kinds of presentation to be used for displaying the discovered patterns.

The use of concept hierarchies plays an important role in aiding the user to visualize the discovered patterns. Mining with concept hierarchies allows the representation of discovered knowledge in high level concepts, which may be more understandable to users than rules expressed in terms of primitive (i.e., raw) data, such as functional or multivalued dependency rules, or integrity constraints. Furthermore, data mining systems should employ concept hierarchies to implement drill-down and roll-up operations, so that users may inspect discovered patterns at multiple levels of abstraction. In addition, pivoting (or rotating), slicing, and dicing operations aid the user in viewing generalized data and knowledge from different perspectives. These operations were discussed in detail in Chapter 2. A data mining system should provide such interactive operations for any dimension, as well as for individual values of each dimension.

Some representation forms may be better suited than others for particular kinds of knowledge. For example, generalized relations and their corresponding crosstabs (cross-tabulations) or pie/bar charts are good for presenting characteristic descriptions, whereas decision trees are a common choice for classification. Interestingness measures should be displayed for each discovered pattern, in order to help users identify those patterns representing useful knowledge. These include confidence, support, and count, as described in Section 4.1.4.

4.2 A data mining query language

Why is it important to have a data mining query language? Well, recall that a desired feature of data mining systems is the ability to *support ad-hoc and interactive data mining* in order to facilitate flexible and effective knowledge discovery. Data mining query languages can be designed to support such a feature.



Rules

age(X, "young") and income(X, "high") => class(X, "A")
age(X, "young") and income(X, "low") => class(X, "B")
age(X, "old") => class(X, "C")

Table

age	income	class	count
young	high	A	1,402
young	low	B	1038
old	high	C	786
old	low	C	1,374

Crosstab

	income		class		
age	high	low	A	B	C
young	1,402	1,038	1,402	1,038	0
old	786	1,374	0	0	2,160
count	2,188	2,412	1,402	1,038	2,160

Figure 4.5: Various forms of presenting and visualizing the discovered patterns.

The importance of the design of a good data mining query language can also be seen from observing the history of relational database systems. Relational database systems have dominated the database market for decades. The standardization of relational query languages, which occurred at the early stages of relational database development, is widely credited for the success of the relational database field. Although each commercial relational database system has its own graphical user interace, the underlying core of each interface is a standardized relational query language. The standardization of relational query languages provided a foundation on which relational systems were developed, and evolved. It facilitated information exchange and technology transfer, and promoted commercialization and wide acceptance of relational database technology. The recent standardization activities in database systems, such as work relating to SQL-3, OMG, and ODMG, further illustrate the importance of having a standard database language for success in the development and commercialization of database systems. Hence, having a good query language for data mining may help standardize the development of platforms for data mining systems.

Designing a comprehensive data mining language is challenging because data mining covers a wide spectrum of tasks, from data characterization to mining association rules, data classification, and evolution analysis. Each task has different requirements. The design of an effective data mining query language requires a deep understanding of the power, limitation, and underlying mechanisms of the various kinds of data mining tasks.

How would you design a data mining query language? Earlier in this chapter, we looked at primitives for defining a data mining task in the form of a data mining query. The primitives specify:

- the set of task-relevant data to be mined,
- the kind of knowledge to be mined,
- the background knowledge to be used in the discovery process,
- the interestingness measures and thresholds for pattern evaluation, and
- the expected representation for visualizing the discovered patterns.

Based on these primitives, we design a query language for data mining called DMQL which stands for **D**ata **M**ining **Q**uery **L**anguage. DMQL allows the ad-hoc mining of several kinds of knowledge from relational databases and data warehouses at multiple levels of abstraction [2].

[2]DMQL syntax for defining data warehouses and data marts is given in Chapter 2.

```
⟨DMQL⟩              ::=    ⟨DMQL_Statement⟩; {⟨DMQL_Statement⟩}
⟨DMQL_Statement⟩ ::=    ⟨Data_Mining_Statment⟩
                       |    ⟨Concept_Hierarchy_Definition_Statement⟩
                       |    ⟨Visualization_and_Presentation⟩
⟨Data_Mining_Statement⟩    ::=
                            use database ⟨database_name⟩ | use data warehouse ⟨data_warehouse_name⟩
                            {use hierarchy ⟨hierarchy_name⟩ for ⟨attribute_or_dimension⟩}
                            ⟨Mine_Knowledge_Specification⟩
                            in relevance to ⟨attribute_or_dimension_list⟩
                            from ⟨relation(s)/cube⟩
                            [where ⟨condition⟩]
                            [order by ⟨order_list⟩]
                            [group by ⟨grouping_list⟩]
                            [having ⟨condition⟩]
                            {with [⟨interest_measure_name⟩] threshold = ⟨threshold_value⟩ [for ⟨attribute(s)⟩]}
⟨Mine_Knowledge_Specification⟩::= ⟨Mine_Char⟩ | ⟨Mine_Discr⟩ | ⟨Mine_Assoc⟩ | ⟨Mine_Class⟩ | ⟨Mine_Pred⟩
⟨Mine_Char⟩  ::=            mine characteristics [as ⟨pattern_name⟩]
                            analyze ⟨measure(s)⟩
⟨Mine_Discr⟩  ::=           mine comparison [as ⟨pattern_name⟩]
                            for ⟨target_class⟩ where ⟨target_condition⟩
                            {versus  ⟨contrast_class_i⟩ where ⟨contrast_condition_i⟩}
                            analyze ⟨measure(s)⟩
⟨Mine_Assoc⟩  ::=           mine associations [as ⟨pattern_name⟩]
                            [matching ⟨metapattern⟩]
⟨Mine_Class⟩  ::=           mine classification [as ⟨pattern_name⟩]
                            analyze ⟨classifying_attribute_or_dimension⟩
⟨Mine_Pred⟩  ::=            mine prediction [as ⟨pattern_name⟩]
                            analyze ⟨prediction_attribute_or_dimension⟩
                            {set {⟨attribute_or_dimension_i⟩= ⟨value_i⟩}}
⟨Concept_Hierarchy_Defintion_Statement⟩   ::=
                            define hierarchy ⟨hierarchy_name⟩
                            [for ⟨attribute_or_dimension⟩]
                            on ⟨relation_or_cube_or_hierarchy⟩
                            as ⟨hierarchy_description⟩
                            [where ⟨condition⟩]
⟨Visualization_and_Presentation⟩    ::=
                            display as ⟨result_form⟩
                            |        roll up on ⟨attribute_or_dimension ⟩
                            |        drill down on ⟨attribute_or_dimension⟩
                            |        add ⟨attribute_or_dimension⟩
                            |        drop ⟨attribute_or_dimension⟩
```

Figure 4.6: Top-level syntax of a data mining query language, DMQL.

The language adopts an SQL-like syntax, so that it can easily be integrated with the relational query language, SQL. The syntax of DMQL is defined in an extended BNF grammar, where "[ ]" represents 0 or one occurrence, "{ }" represents 0 or more occurrences, and words in sans serif font represent keywords.

In Sections 4.2.1 to 4.2.5, we develop DMQL syntax for each of the data mining primitives. In Section 4.2.6, we show an example data mining query, specified in the proposed syntax. A top-level summary of the language is shown in Figure 4.6.

4.2.1 Syntax for task-relevant data specification

The first step in defining a data mining task is the specification of the task-relevant data, i.e., the data on which mining is to be performed. This involves specifying the database and tables or data warehouse containing the relevant data, conditions for selecting the relevant data, the relevant attributes or dimensions for exploration, and instructions regarding the ordering or grouping of the data retrieved. DMQL provides clauses for the specification of such information, as follows.

- use database ⟨database_name⟩, or use data warehouse ⟨data_warehouse_name⟩: The use clause directs the mining task to the database or data warehouse specified.
- from ⟨relation(s)/cube(s)⟩ [where ⟨condition⟩]: The from and where clauses respectively specify the database tables or data cubes involved, and the conditions defining the data to be retrieved.
- in relevance to ⟨att_or_dim_list⟩: This clause lists the attributes or dimensions for exploration.
- order by ⟨order_list⟩: The order by clause specifies the sorting order of the task-relevant data.
- group by ⟨grouping_list⟩: The group by clause specifies criteria for grouping the data.
- having ⟨condition⟩: The having clause specifies the condition by which groups of data are considered relevant.

These clauses form an SQL query to collect the task-relevant data.

Example 4.11 This example shows how to use DMQL to specify the task-relevant data described in Example 4.1 for the mining of associations between items frequently purchased at *AllElectronics* by Canadian customers, with respect to customer *income* and *age*. In addition, the user specifies that she would like the data to be grouped by date. The data are retrieved from a relational database.

```
use database AllElectronics_db
in relevance to I.name, I.price, C.income, C.age
from customer C, item I, purchases P, items_sold S
where I.item_ID = S.item_ID and S.trans_ID = P.trans_ID and P.cust_ID = C.cust_ID
      and C.address = "Canada"
group by P.date
```

□

4.2.2 Syntax for specifying the kind of knowledge to be mined

The ⟨Mine_Knowledge_Specification⟩ statement is used to specify the kind of knowledge to be mined. In other words, it indicates the data mining functionality to be performed. Its syntax is defined below for characterization, discrimination, association, classification, and prediction.

1. **Characterization.**

```
⟨Mine_Knowledge_Specification⟩ ::=
        mine characteristics [as ⟨pattern_name⟩]
        analyze ⟨measure(s)⟩
```

This specifies that characteristic descriptions are to be mined. The analyze clause, when used for characterization, specifies aggregate measures, such as `count`, `sum`, or `count%` (percentage count, i.e., the percentage of tuples in the relevant data set with the specified characteristics). These measures are to be computed for each data characteristic found.

Example 4.12 The following specifies that the kind of knowledge to be mined is a characteristic description describing customer purchasing habits. For each characteristic, the percentage of task-relevant tuples satisfying that characteristic is to be displayed.

```
mine characteristics as customerPurchasing
analyze count%
```

□

2. **Discrimination.**

```
⟨Mine_Knowledge_Specification⟩ ::=
        mine comparison [as ⟨pattern_name⟩]
        for ⟨target_class⟩ where ⟨target_condition⟩
        {versus ⟨contrast_class_i⟩ where ⟨contrast_condition_i⟩}
        analyze ⟨measure(s)⟩
```

This specifies that discriminant descriptions are to be mined. These descriptions compare a given target class of objects with one or more other contrasting classes. Hence, this kind of knowledge is referred to as a comparison. As for characterization, the analyze clause specifies aggregate measures, such as `count`, `sum`, or `count%`, to be computed and displayed for each description.

Example 4.13 The user may define categories of customers, and then mine descriptions of each category. For instance, a user may define *bigSpenders* as customers who purchase items that cost $100 or more on average, and *budgetSpenders* as customers who purchase items at less than $100 on average. The mining of discriminant descriptions for customers from each of these categories can be specified in DMQL as shown below, where I refers to the *item* relation. The count of task-relevant tuples satisfying each description is to be displayed.

```
mine comparison as purchaseGroups
for bigSpenders where avg(I.price) ≥ $100
versus budgetSpenders where avg(I.price) < $100
analyze count
```

□

3. **Association.**

```
⟨Mine_Knowledge_Specification⟩ ::=
        mine associations [as ⟨pattern_name⟩]
        [matching ⟨metapattern⟩]
```

This specifies the mining of patterns of association. When specifying association mining, the user has the option of providing templates (also known as *metapatterns* or *metarules*) with the matching clause. The metapatterns can be used to focus the discovery towards the patterns that match the given metapatterns, thereby enforcing additional syntactic constraints for the mining task. In addition to providing syntactic constraints, the metapatterns represent data hunches or hypotheses that the user finds interesting for investigation. Mining with the use of metapatterns, or **metarule-guided mining**, allows additional flexibility for ad-hoc rule mining. While metapatterns may be used in the mining of other forms of knowledge, they are most useful for association mining due to the vast number of potentially generated associations.

Example 4.14 The metapattern of Example 4.2 can be specified as follows to guide the mining of association rules describing customer buying habits.

mine associations as buyingHabits
matching $P(X : customer, W) \wedge Q(X, Y) \Rightarrow buys(X, Z)$

□

4. **Classification.**

   ⟨Mine_Knowledge_Specification⟩ ::=
   mine classification [as ⟨pattern_name⟩]
   analyze ⟨classifying_attribute_or_dimension⟩

   This specifies that patterns for data classification are to be mined. The analyze clause specifies that the classification is performed according to the values of ⟨classifying_attribute_or_dimension⟩. For categorical attributes or dimensions, typically each value represents a class (such as "Vancouver", "New York", "Chicago", and so on for the dimension *location*). For numeric attributes or dimensions, each class may be defined by a range of values (such as "20-39", "40-59", "60-89" for *age*). Classification provides a concise framework which best describes the objects in each class and distinguishes them from other classes.

   Example 4.15 To mine patterns classifying customer credit rating where credit rating is determined by the attribute *credit_info*, the following DMQL specification is used:

   mine classification as classifyCustomerCreditRating
   analyze credit_info

   □

5. **Prediction.**

   ⟨Mine_Knowledge_Specification⟩ ::=
   mine prediction [as ⟨pattern_name⟩]
   analyze ⟨prediction_attribute_or_dimension⟩
   {set {⟨attribute_or_dimension_*i*⟩= ⟨value_*i*⟩}}

   This DMQL syntax is for prediction. It specifies the mining of missing or unknown continuous data values, or of the data distribution, for the attribute or dimension specified in the analyze clause. A predictive model is constructed based on the analysis of the values of the other attributes or dimensions describing the data objects (tuples). The set clause can be used to fix the values of these other attributes.

   Example 4.16 To predict the retail price of a new item at *AllElectronics*, the following DMQL specification is used:

   mine prediction as predictItemPrice
   analyze price
   set category = "TV" and brand = "SONY"

   The set clause specifies that the resulting predictive patterns regarding price are for the subset of task-relevant data relating to SONY TV's. If no set clause is specified, then the prediction returned would be a data distribution for all categories and brands of *AllElectronics* items in the task-relevant data. □

The data mining language should also allow the specification of other kinds of knowledge to be mined, in addition to those shown above. These include the mining of data clusters, evolution rules or sequential patterns, and deviations.

4.2.3 Syntax for concept hierarchy specification

Concept hierarchies allow the mining of knowledge at multiple levels of abstraction. In order to accommodate the different viewpoints of users with regards to the data, there may be more than one concept hierarchy per attribute or dimension. For instance, some users may prefer to organize branch locations by provinces and states, while others may prefer to organize them according to languages used. In such cases, a user can indicate which concept hierarchy is to be used with the statement

use hierarchy ⟨hierarchy⟩ for ⟨attribute_or_dimension⟩.

Otherwise, a default hierarchy per attribute or dimension is used.

How can we define concept hierarchies, using DMQL? In Section 4.1.3, we studied four types of concept hierarchies, namely schema, set-grouping, operation-derived, and rule-based hierarchies. Let's look at the following syntax for defining each of these hierarchy types.

1. **Definition of schema hierarchies.**

Example 4.17 Earlier, we defined a schema hierarchy for a relation *address* as the total order $street < city < province\_or\_state < country$. This can be defined in the data mining query language as:

```
define hierarchy location_hierarchy on address as [street, city, province_or_state, country]
```

The ordering of the listed attributes is important. In fact, a total order is defined which specifies that *street* is conceptually one level lower than *city*, which is in turn conceptually one level lower than *province_or_state*, and so on. □

Example 4.18 A data mining system will typically have a predefined concept hierarchy for the schema *date* (*day*, *month*, *quarter*, *year*), such as:

```
define hierarchy time_hierarchy on date as [day, month, quarter, year]
```

□

Example 4.19 Concept hierarchy definitions can involve several relations. For example, an *item_hierarchy* may involve two relations, *item* and *supplier*, defined by the following schema.

$item(item\_ID, brand, type, place\_made, supplier)$
$supplier(name, type, headquarter\_location, owner, size, assets, revenue)$

The hierarchy *item_hierarchy* can be defined as follows:

```
define hierarchy item_hierarchy on item, supplier as
        [item_ID, brand, item.supplier, item.type, supplier.type]
        where item.supplier = supplier.name
```

If the concept hierarchy definition contains an attribute name that is shared by two relations, then the attribute is prefixed by its relation name, using the same dot (".") notation as in SQL (e.g., item.supplier). The join condition of the two relations is specified by a where clause. □

2. **Definition of set-grouping hierarchies.**

Example 4.20 The set-grouping hierarchy for *age* of Example 4.4 can be defined in terms of ranges as follows:

```
define hierarchy age_hierarchy for age on customer as
        level1: {young, middle_aged, senior} < level0: all
        level2: {20, ..., 39} < level1: young
        level2: {40, ..., 59} < level1: middle_aged
        level2: {60, ..., 89} < level1: senior
```

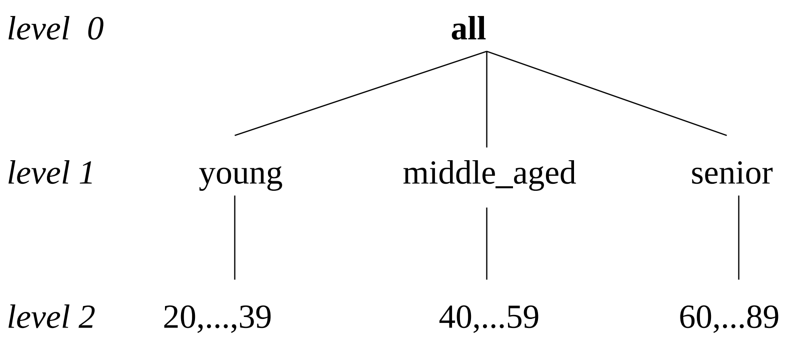


Figure 4.7: A concept hierarchy for the attribute *age*.

The notation "..." implicitly specifies all the possible values within the given range. For example, "{20, ..., 39}" includes all integers within the range of the endpoints, 20 and 39. Ranges may also be specified with real numbers as endpoints. The corresponding concept hierarchy is shown in Figure 4.7. The most general concept for *age* is all, and is placed at the root of the hierarchy. By convention, the all value is always at level 0 of any hierarchy. The all node in Figure 4.7 has three child nodes, representing more specific abstractions of *age*, namely *young*, *middle_aged*, and *senior*. These are at level 1 of the hierarchy. The age ranges for each of these level 1 concepts are defined at level 2 of the hierarchy. □

Example 4.21 The schema hierarchy in Example 4.17 for *location* can be refined by adding an additional concept level, *continent*.

```
define hierarchy on location_hierarchy as
        country: {Canada, USA, Mexico} < continent: NorthAmerica
        country: {England, France, Germany, Italy} < continent: Europe
        ...
        continent: {NorthAmerica, Europe, Asia} < all
```

By listing the countries (for which *AllElectronics* sells merchandise) belonging to each continent, we build an additional concept layer on top of the schema hierarchy of Example 4.17. □

3. **Definition of operation-derived hierarchies**

Example 4.22 As an alternative to the set-grouping hierarchy for *age* in Example 4.20, a user may wish to define an operation-derived hierarchy for *age* based on data clustering routines. This is especially useful when the values of a given attribute are not uniformly distributed. A hierarchy for *age* based on clustering can be defined with the following statement:

```
define hierarchy age_hierarchy for age on customer as
        {age_category(1), ..., age_category(5)} := cluster(default, age, 5) < all(age)
```

This statement indicates that a default clustering algorithm is to be performed on all of the *age* values in the relation *customer* in order to form five clusters. The clusters are ranges with names explicitly defined as "age_category(1), ..., age_category(5)", organized in ascending order. □

4. **Definition of rule-based hierarchies**

Example 4.23 A concept hierarchy can be defined based on a set of rules. Consider the concept hierarchy of Example 4.8 for *items* at *AllElectronics*. This hierarchy is based on item profit margins, where the profit margin of an item is defined as the difference between the retail price of the item, and the cost incurred by *AllElectronics* to purchase the item for sale. The hierarchy organizes items into *low_profit_margin* items, *medium-profit_margin* items, and *high_profit_margin* items, and is defined in DMQL by the following set of rules.

```
define hierarchy profit_margin_hierarchy on item as
        level_1: low_profit_margin < level_0: all
                if (price − cost) < $50
        level_1: medium-profit_margin < level_0: all
                if ((price − cost) > $50) and ((price − cost) ≤ $250))
        level_1: high_profit_margin < level_0: all
                if (price − cost) > $250
```

□

4.2.4 Syntax for interestingness measure specification

The user can control the number of uninteresting patterns returned by the data mining system by specifying measures of pattern interestingness and their corresponding thresholds. Interestingness measures include the confidence, support, noise, and novelty measures described in Section 4.1.4. Interestingness measures and thresholds can be specified by the user with the statement:

with [⟨interest_measure_name⟩] threshold = ⟨threshold_value⟩

Example 4.24 In mining association rules, a user can confine the rules to be found by specifying a minimum support and minimum confidence threshold of 0.05 and 0.7, respectively, with the statements:

with support threshold = 0.05
with confidence threshold = 0.7

□

The interestingness measures and threshold values can be set and modified interactively.

4.2.5 Syntax for pattern presentation and visualization specification

How can users specify the forms of presentation and visualization to be used in displaying the discovered patterns? Our data mining query language needs syntax which allows users to specify the display of discovered patterns in one or more forms, including rules, tables, crosstabs, pie or bar charts, decision trees, cubes, curves, or surfaces. We define the DMQL display statement for this purpose:

display as ⟨result_form⟩

where the ⟨result_form⟩ could be any of the knowledge presentation or visualization forms listed above.

Interactive mining should allow the discovered patterns to be viewed at different concept levels or from different angles. This can be accomplished with roll-up and drill-down operations, as described in Chapter 2. Patterns can be rolled-up, or viewed at a more general level, by *climbing up* the concept hierarchy of an attribute or dimension (replacing lower level concept values by higher level values). Generalization can also be performed by dropping attributes or dimensions. For example, suppose that a pattern contains the attribute *city*. Given the *location* hierarchy *city* < *province_or_state* < *country* < *continent*, then dropping the attribute *city* from the patterns will generalize the data to the next lowest level attribute, *province_or_state*. Patterns can be drilled-down on, or viewed at a less general level, by *stepping down* the concept hierarchy of an attribute or dimension. Patterns can also be made less general by adding attributes or dimensions to their description. The attribute added must be one of the attributes listed in the in relevance to clause for task-relevant specification. The user can alternately view the patterns at different levels of abstractions with the use of the following DMQL syntax:

```
⟨Multilevel_Manipulation⟩ ::=   roll up on ⟨attribute_or_dimension⟩
                               | drill down on ⟨attribute_or_dimension⟩
                               | add ⟨attribute_or_dimension⟩
                               | drop ⟨attribute_or_dimension⟩
```

Example 4.25 Suppose descriptions are mined based on the dimensions *location*, *age*, and *income*. One may "roll up on location" or "drop age" to generalize the discovered patterns. □

age	type	place_made	count%
30-39	home security system	USA	19
40-49	home security system	USA	15
20-29	CD player	Japan	26
30-39	CD player	USA	13
40-49	large screen TV	Japan	8
...	...	...	...
			100%

Figure 4.8: Characteristic descriptions in the form of a table, or generalized relation.

4.2.6 Putting it all together — an example of a DMQL query

In the above discussion, we presented DMQL syntax for specifying data mining queries in terms of the five data mining primitives. For a given query, these primitives define the task-relevant data, the kind of knowledge to be mined, the concept hierarchies and interestingness measures to be used, and the representation forms for pattern visualization. Here we put these components together. Let's look at an example for the full specification of a DMQL query.

Example 4.26 Mining characteristic descriptions. Suppose, as a marketing manager of *AllElectronics*, you would like to characterize the buying habits of customers who purchase items priced at no less than $100, with respect to the customer's age, the type of item purchased, and the place in which the item was made. For each characteristic discovered, you would like to know the percentage of customers having that characteristic. In particular, you are only interested in purchases made in Canada, and paid for with an American Express ("AmEx") credit card. You would like to view the resulting descriptions in the form of a table. This data mining query is expressed in DMQL as follows.

```
use database AllElectronics_db
use hierarchy location_hierarchy for B.address
mine characteristics as customerPurchasing
analyze count%
in relevance to C.age, I.type, I.place_made
from customer C, item I, purchases P, items_sold S, works_at W, branch B
where I.item_ID = S.item_ID and S.trans_ID = P.trans_ID and P.cust_ID = C.cust_ID
        and P.method_paid = "AmEx" and P.empl_ID = W.empl_ID and W.branch_ID = B.branch_ID
        and B.address = "Canada" and I.price ≥ 100
with noise threshold = 0.05
display as table
```

The data mining query is parsed to form an SQL query which retrieves the set of task-relevant data from the *AllElectronics* database. The concept hierarchy *location_hierarchy*, corresponding to the concept hierarchy of Figure 4.3 is used to generalize branch locations to high level concept levels such as "Canada". An algorithm for mining characteristic rules, which uses the generalized data, can then be executed. Algorithms for mining characteristic rules are introduced in Chapter 5. The mined characteristic descriptions, derived from the attributes *age, type* and *place_made*, are displayed as a table, or generalized relation (Figure 4.8). The percentage of task-relevant tuples satisfying each generalized tuple is shown as `count%`. If no visualization form is specified, a default form is used. The noise threshold of 0.05 means any generalized tuple found that represents less than 5% of the total count is omitted from display. □

Similarly, the complete DMQL specification of data mining queries for discrimination, association, classification, and prediction can be given. Example queries are presented in the following chapters which respectively study the mining of these kinds of knowledge.

4.3 Designing graphical user interfaces based on a data mining query language

A data mining query language provides necessary primitives which allow users to communicate with data mining systems. However, inexperienced users may find data mining query languages awkward to use, and the syntax difficult to remember. Instead, users may prefer to communicate with data mining systems through a Graphical User Interface (GUI). In relational database technology, SQL serves as a standard "core" language for relational systems, on top of which GUIs can easily be designed. Similarly, a data mining query language may serve as a "core language" for data mining system implementations, providing a basis for the development of GUI's for effective data mining.

A data mining GUI may consist of the following functional components.

1. **Data collection and data mining query composition**: This component allows the user to specify task-relevant data sets, and to compose data mining queries. It is similar to GUIs used for the specification of relational queries.

2. **Presentation of discovered patterns**: This component allows the display of the discovered patterns in various forms, including tables, graphs, charts, curves, or other visualization techniques.

3. **Hierarchy specification and manipulation:** This component allows for concept hierarchy specification, either manually by the user, or automatically (based on analysis of the data at hand). In addition, this component should allow concept hierarchies to be modified by the user, or adjusted automatically based on a given data set distribution.

4. **Manipulation of data mining primitives**: This component may allow the dynamic adjustment of data mining thresholds, as well as the selection, display, and modification of concept hierarchies. It may also allow the modification of previous data mining queries or conditions.

5. **Interactive multilevel mining**: This component should allow roll-up or drill-down operations on discovered patterns.

6. **Other miscellaneous information**: This component may include on-line help manuals, indexed search, debugging, and other interactive graphical facilities.

Do you think that data mining query languages may evolve to form a standard for designing data mining GUIs? If such an evolution is possible, the standard would facilitate data mining software development and system communication. Some GUI primitives, such as pointing to a particular point in a curve or graph, however, are difficult to specify using a text-based data mining query language like DMQL. Alternatively, a standardized GUI-based language may evolve and replace SQL-like data mining languages. Only time will tell.

4.4 Summary

- We have studied five **primitives** for specifying a data mining task in the form of a **data mining query**. These primitives are the specification of task-relevant data (i.e., the data set to be mined), the kind of knowledge to be mined (e.g., characterization, discrimination, association, classification, or prediction), background knowledge (typically in the form of concept hierarchies), interestingness measures, and knowledge presentation and visualization techniques to be used for displaying the discovered patterns.

- In defining the **task-relevant data**, the user specifies the database and tables (or data warehouse and data cubes) containing the data to be mined, conditions for selecting and grouping such data, and the attributes (or dimensions) to be considered during mining.

- **Concept hierarchies** provide useful background knowledge for expressing discovered patterns in concise, high level terms, and facilitate the mining of knowledge at multiple levels of abstraction.

- Measures of **pattern interestingness** assess the simplicity, certainty, utility, or novelty of discovered patterns. Such measures can be used to help reduce the number of uninteresting patterns returned to the user.

- Users should be able to specify the desired form for **visualizing** the discovered patterns, such as rules, tables, charts, decision trees, cubes, graphs, or reports. Roll-up and drill-down operations should also be available for the inspection of patterns at multiple levels of abstraction.
- **Data mining query languages** can be designed to support ad-hoc and interactive data mining. A data mining query language, such as DMQL, should provide commands for specifying each of the data mining primitives, as well as for concept hierarchy generation and manipulation. Such query languages are SQL-based, and may eventually form a standard on which graphical user interfaces for data mining can be based.

Exercises

1. List and describe the five primitives for specifying a data mining task.
2. Suppose that the university course database for *Big-University* contains the following attributes: the name, address, status (e.g., undergraduate or graduate), and major of each student, and their cumulative grade point average (GPA).
   (a) Propose a concept hierarchy for the attributes *status*, *major*, *GPA*, and *address*.
   (b) For each concept hierarchy that you have proposed above, what type of concept hierarchy have you proposed?
   (c) Define each hierarchy using DMQL syntax.
   (d) Write a DMQL query to find the characteristics of students who have an excellent GPA.
   (e) Write a DMQL query to compare students majoring in science with students majoring in arts.
   (f) Write a DMQL query to find associations involving course instructors, student grades, and some other attribute of your choice. Use a metarule to specify the format of associations you would like to find. Specify minimum thresholds for the confidence and support of the association rules reported.
   (g) Write a DMQL query to predict student grades in "Computing Science 101" based on student GPA to date and course instructor
3. Consider association rule 4.8 below, which was mined from the student database at *Big-University*.

$$major(X, \text{"}science\text{"}) \Rightarrow status(X, \text{"}undergrad\text{"}). \tag{4.8}$$

   Suppose that the number of students at the university (that is, the number of task-relevant data tuples) is 5000, that 56% of undergraduates at the university major in science, that 64% of the students are registered in programs leading to undergraduate degrees, and that 70% of the students are majoring in science.
   (a) Compute the confidence and support of Rule (4.8).
   (b) Consider Rule (4.9) below.

$$major(X, \text{"}biology\text{"}) \Rightarrow status(X, \text{"}undergrad\text{"}) \qquad [17\%, 80\%] \tag{4.9}$$

   Suppose that 30% of science students are majoring in biology. Would you consider Rule (4.9) to be novel with respect to Rule (4.8)? Explain.
4. The ⟨Mine_Knowledge_Specification⟩ statement can be used to specify the mining of characteristic, discriminant, association, classification, and prediction rules. Propose a syntax for the mining of clusters.
5. Rather than requiring users to manually specify concept hierarchy definitions, some data mining systems can generate or modify concept hierarchies automatically *based on the analysis of data distributions.*
   (a) Propose concise DMQL syntax for the automatic generation of concept hierarchies.
   (b) A concept hierarchy may be automatically adjusted to reflect changes in the data. Propose concise DMQL syntax for the automatic adjustment of concept hierarchies.

(c) Give examples of your proposed syntax.

6. In addition to concept hierarchy creation, DMQL should also provide syntax which allows users to modify previously defined hierarchies. This syntax should allow the *insertion* of new nodes, the *deletion* of nodes, and the *moving* of nodes within the hierarchy.

   - To insert a new node N into level L of a hierarchy, one should specify its parent node P in the hierarchy, unless N is at the topmost layer.
   - To delete node N from a hierarchy, all of its descendent nodes should be removed from the hierarchy as well.
   - To move a node N to a different location within the hierarchy, the parent of N will change, and all of the descendents of N should be moved accordingly.

   (a) Propose DMQL syntax for each of the above operations.

   (b) Show examples of your proposed syntax.

   (c) For each operation, illustrate the operation by drawing the corresponding concept hierarchies ("before" and "after").

Bibliographic Notes

A number of objective interestingness measures have been proposed in the literature. Simplicity measures are given in Michalski [23]. The confidence and support measures for association rule interestingness described in this chapter were proposed in Agrawal, Imielinski, and Swami [1]. The strategy we described for identifying redundant multilevel association rules was proposed in Srikant and Agrawal [31, 32]. Other objective interestingness measures have been presented in [1, 6, 12, 17, 27, 19, 30]. Subjective measures of interestingness, which consider user beliefs regarding relationships in the data, are discussed in [18, 21, 20, 26, 29].

The DMQL data mining query language was proposed by Han et al. [11] for the DBMiner data mining system. *Discovery Board* (formerly *Data Mine*) was proposed by Imielinski, Virmani, and Abdulghani [13] as an application development interface prototype involving an SQL-based operator for data mining query specification and rule retrieval. An SQL-like operator for mining single-dimensional association rules was proposed by Meo, Psaila, and Ceri [22], and extended by Baralis and Psaila [4]. Mining with metarules is described in Klemettinen et al. [16], Fu and Han [9], Shen et al. [28], and Kamber et al. [14]. Other ideas involving the use of templates or predicate constraints in mining have been discussed in [3, 7, 18, 29, 33, 25].

For a comprehensive survey of visualization techniques, see *Visual Techniques for Exploring Databases* by Keim [15].

Bibliography

[1] R. Agrawal, T. Imielinski, and A. Swami. Database mining: A performance perspective. *IEEE Trans. Knowledge and Data Engineering*, 5:914–925, 1993.

[2] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proc. 1995 Int. Conf. Data Engineering*, pages 3–14, Taipei, Taiwan, March 1995.

[3] T. Anand and G. Kahn. Opportunity explorer: Navigating large databases using knowledge discovery templates. In *Proc. AAAI-93 Workshop Knowledge Discovery in Databases*, pages 45–51, Washington DC, July 1993.

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

[5] R.G.G. Cattell. *Object Data Management: Object-Oriented and Extended Relational Databases, Rev. Ed.* Addison-Wesley, 1994.

[6] M. S. Chen, J. Han, and P. S. Yu. Data mining: An overview from a database perspective. *IEEE Trans. Knowledge and Data Engineering*, 8:866–883, 1996.

[7] V. Dhar and A. Tuzhilin. Abstract-driven pattern discovery in databases. *IEEE Trans. Knowledge and Data Engineering*, 5:926–938, 1993.

[8] M. Ester, H.-P. Kriegel, and X. Xu. Knowledge discovery in large spatial databases: Focusing techniques for efficient class identification. In *Proc. 4th Int. Symp. Large Spatial Databases (SSD'95)*, pages 67–82, Portland, Maine, August 1995.

[9] Y. Fu and J. Han. Meta-rule-guided mining of association rules in relational databases. In *Proc. 1st Int. Workshop Integration of Knowledge Discovery with Deductive and Object-Oriented Databases (KDOOD'95)*, pages 39–46, Singapore, Dec. 1995.

[10] J. Han, Y. Cai, and N. Cercone. Data-driven discovery of quantitative rules in relational databases. *IEEE Trans. Knowledge and Data Engineering*, 5:29–40, 1993.

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

[12] J. Hong and C. Mao. Incremental discovery of rules and structure by hierarchical and parallel clustering. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 177–193. AAAI/MIT Press, 1991.

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

[14] M. Kamber, J. Han, and J. Y. Chiang. Metarule-guided mining of multi-dimensional association rules using data cubes. In *Proc. 3rd Int. Conf. Knowledge Discovery and Data Mining (KDD'97)*, pages 207–210, Newport Beach, California, August 1997.

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

[16] M. Klemettinen, H. Mannila, P. Ronkainen, H. Toivonen, and A.I. Verkamo. Finding interesting rules from large sets of discovered association rules. In *Proc. 3rd Int. Conf. Information and Knowledge Management*, pages 401–408, Gaithersburg, Maryland, Nov. 1994.

[17] A. J. Knobbe and P. W. Adriaans. Analysing binary associations. In *Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining (KDD'96)*, pages 311–314, Portland, OR, Aug. 1996.

[18] B. Liu, W. Hsu, and S. Chen. Using general impressions to analyze discovered classification rules. In *Proc. 3rd Int.. Conf. on Knowledge Discovery and Data Mining (KDD'97)*, pages 31–36, Newport Beach, CA, August 1997.

[19] J. Major and J. Mangano. Selecting among rules induced from a hurricane database. *Journal of Intelligent Information Systems*, 4:39–52, 1995.

[20] C. J. Matheus and G. Piatesky-Shapiro. An application of KEFIR to the analysis of healthcare information. In *Proc. AAAI'94 Workshop Knowledge Discovery in Databases (KDD'94)*, pages 441–452, Seattle, WA, July 1994.

[21] C.J. Matheus, G. Piatetsky-Shapiro, and D. McNeil. Selecting and reporting what is interesting: The KEFIR application to healthcare data. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 495–516. AAAI/MIT Press, 1996.

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

[23] R. S. Michalski. A theory and methodology of inductive learning. In Michalski et al., editor, *Machine Learning: An Artificial Intelligence Approach, Vol. 1*, pages 83–134. Morgan Kaufmann, 1983.

[24] R. Ng and J. Han. Efficient and effective clustering method for spatial data mining. In *Proc. 1994 Int. Conf. Very Large Data Bases*, pages 144–155, Santiago, Chile, September 1994.

[25] R. Ng, L. V. S. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained associations rules. In *Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data*, pages 13–24, Seattle, Washington, June 1998.

[26] G. Piatesky-Shapiro and C. J. Matheus. The interestingness of deviations. In *Proc. AAAI'94 Workshop Knowledge Discovery in Databases (KDD'94)*, pages 25–36, Seattle, WA, July 1994.

[27] G. Piatetsky-Shapiro. Discovery, analysis, and presentation of strong rules. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 229–238. AAAI/MIT Press, 1991.

[28] W. Shen, K. Ong, B. Mitbander, and C. Zaniolo. Metaqueries for data mining. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 375–398. AAAI/MIT Press, 1996.

[29] A. Silberschatz and A. Tuzhilin. What makes patterns interesting in knowledge discovery systems. *IEEE Trans. on Knowledge and Data Engineering*, 8:970–974, Dec. 1996.

[30] P. Smyth and R.M. Goodman. An information theoretic approch to rule induction. *IEEE Trans. Knowledge and Data Engineering*, 4:301–316, 1992.

[31] R. Srikant and R. Agrawal. Mining generalized association rules. In *Proc. 1995 Int. Conf. Very Large Data Bases*, pages 407–419, Zurich, Switzerland, Sept. 1995.

[32] R. Srikant and R. Agrawal. Mining quantitative association rules in large relational tables. In *Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data*, pages 1–12, Montreal, Canada, June 1996.

[33] R. Srikant, Q. Vu, and R. Agrawal. Mining association rules with item constraints. In *Proc. 3rd Int. Conf. Knowledge Discovery and Data Mining (KDD'97)*, pages 67–73, Newport Beach, California, August 1997.

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

[35] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: an efficient data clustering method for very large databases. In *Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data*, pages 103–114, Montreal, Canada, June 1996.