7 Advanced Pattern Mining

Frequent pattern mining has reached far beyond the basics due to substantial research, numerous extensions of the problem scope, and broad application studies. In this chapter, you will learn methods for advanced pattern mining. We begin by laying out a general road map for pattern mining. We introduce methods for mining various kinds of patterns, and discuss extended applications of pattern mining. We include in-depth coverage of methods for mining many kinds of patterns: multilevel patterns, multidimensional patterns, patterns in continuous data, rare patterns, negative patterns, constrained frequent patterns, frequent patterns in high-dimensional data, colossal patterns, and compressed and approximate patterns. Other pattern mining themes, including mining sequential and structured patterns and mining patterns from spatiotemporal, multimedia, and stream data, are considered more advanced topics and are not covered in this book. Notice that *pattern mining* is a more general term than *frequent pattern mining* since the former covers rare and negative patterns as well. However, when there is no ambiguity, the two terms are used interchangeably.

7.1 Pattern Mining: A Road Map

Chapter 6 introduced the basic concepts, techniques, and applications of frequent pattern mining using market basket analysis as an example. Many other kinds of data, user requests, and applications have led to the development of numerous, diverse methods for mining patterns, associations, and correlation relationships. Given the rich literature in this area, it is important to lay out a clear road map to help us get an organized picture of the field and to select the best methods for pattern mining applications.

Figure 7.1 outlines a general road map on pattern mining research. Most studies mainly address three pattern mining aspects: the kinds of patterns mined, mining methodologies, and applications. Some studies, however, integrate multiple aspects; for example, different applications may need to mine different patterns, which naturally leads to the development of new mining methodologies.

Data Mining: Concepts and Techniques. DOI: 10.1016/B978-0-12-381479-1.00007-1

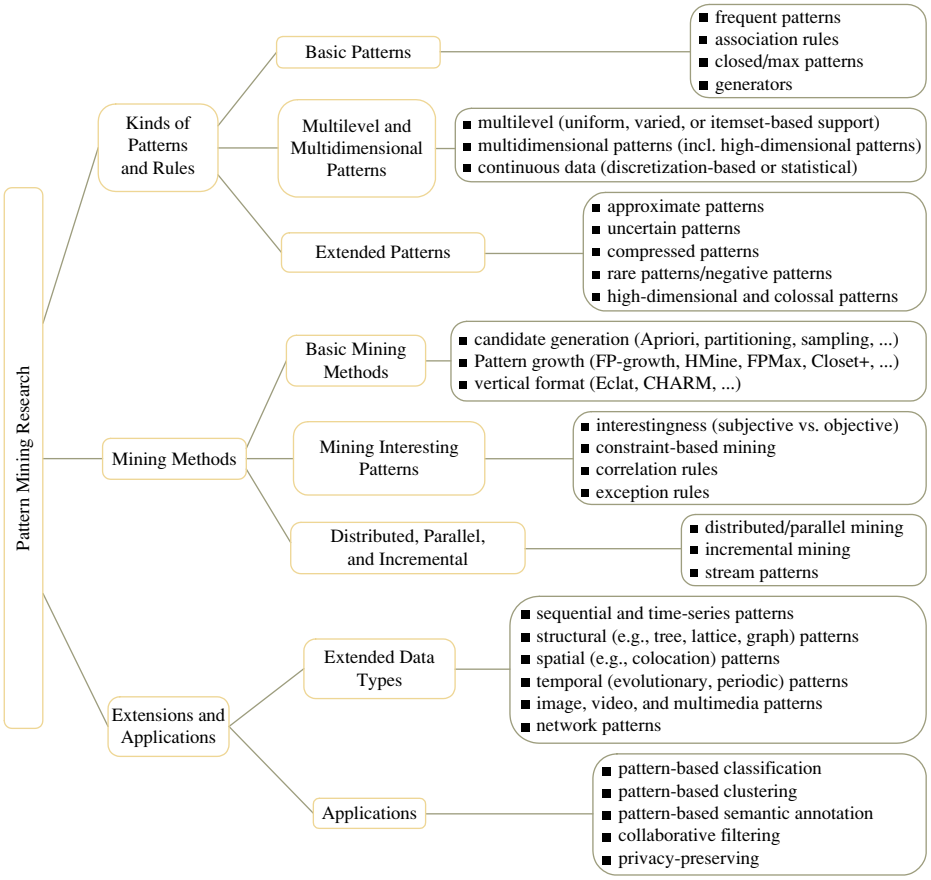

Figure 7.1 A general road map on pattern mining research.

Based on pattern diversity, pattern mining can be classified using the following criteria:

- **Basic patterns:** As discussed in Chapter 6, a frequent pattern may have several alternative forms, including a simple frequent pattern, a closed pattern, or a max-pattern. To review, a **frequent pattern** is a pattern (or itemset) that satisfies a minimum support threshold. A pattern p is a **closed pattern** if there is no superpattern p' with the same support as p. Pattern p is a **max-pattern** if there exists no frequent superpattern of p. Frequent patterns can also be mapped into **association rules**, or other kinds of rules based on interestingness measures. Sometimes we may also be interested in **infrequent** or **rare patterns** (i.e., patterns that occur rarely but are of critical importance, or **negative patterns** (i.e., patterns that reveal a negative correlation between items).

- **Based on the *abstraction* levels involved in a pattern:** Patterns or association rules may have items or concepts residing at high, low, or multiple abstraction levels. For example, suppose that a set of association rules mined includes the following rules where X is a variable representing a customer:

$$buys(X, \text{“}computer\text{”}) \Rightarrow buys(X, \text{“}printer\text{”}) \tag{7.1}$$

$$buys(X, \text{“}laptop\ computer\text{”}) \Rightarrow buys(X, \text{“}color\ laser\ printer\text{”}) \tag{7.2}$$

  In Rules (7.1) and (7.2), the items bought are referenced at different abstraction levels (e.g., “*computer*” is a higher-level abstraction of “*laptop computer*,” and “*color laser printer*” is a lower-level abstraction of “*printer*”). We refer to the rule set mined as consisting of **multilevel association rules**. If, instead, the rules within a given set do not reference items or attributes at different abstraction levels, then the set contains **single-level association rules**.

- **Based on the *number of dimensions* involved in the rule or pattern:** If the items or attributes in an association rule or pattern reference only one dimension, it is a **single-dimensional association rule/pattern.** For example, Rules (7.1) and (7.2) are single-dimensional association rules because they each refer to only one dimension, *buys*.[1]

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

$$age(X, \text{“}20\ldots29\text{”}) \wedge income(X, \text{“}52K\ldots58K\text{”}) \Rightarrow buys(X, \text{“}iPad\text{”}). \tag{7.3}$$

- **Based on the *types of values* handled in the rule or pattern:** If a rule involves associations between the presence or absence of items, it is a **Boolean association rule**. For example, Rules (7.1) and (7.2) are Boolean association rules obtained from market basket analysis.

  If a rule describes associations between quantitative items or attributes, then it is a **quantitative association rule**. In these rules, quantitative values for items or attributes are partitioned into intervals. Rule (7.3) can also be considered a quantitative association rule where the quantitative attributes *age* and *income* have been discretized.

- **Based on the *constraints* or *criteria* used to mine *selective patterns*:** The patterns or rules to be discovered can be **constraint-based** (i.e., satisfying a set of user-defined constraints), **approximate**, **compressed**, **near-match** (i.e., those that tally the support count of the near or almost matching itemsets), **top-*k*** (i.e., the k most frequent itemsets for a user-specified value, k), **redundancy-aware top-*k*** (i.e., the top-k patterns with similar or redundant patterns excluded), and so on.

[1] Following the terminology used in multidimensional databases, we refer to each distinct predicate in a rule as a *dimension*.

Alternatively, pattern mining can be classified with respect to the kinds of data and applications involved, using the following criteria:

- **Based on *kinds of data and features* to be mined:** Given relational and data warehouse data, most people are interested in itemsets. Thus, frequent pattern mining in this context is essentially **frequent itemset mining**, that is, to mine frequent *sets of items*. However, in many other applications, patterns may involve sequences and structures. For example, by studying the order in which items are frequently purchased, we may find that customers tend to first buy a PC, followed by a digital camera, and then a memory card. This leads to **sequential patterns**, that is, frequent *subsequences* (which are often separated by some other events) in a *sequence of ordered events*.

  We may also mine **structural patterns**, that is, frequent *substructures*, in a *structured data set*. Note that *structure* is a general concept that covers many different kinds of structural forms such as directed graphs, undirected graphs, lattices, trees, sequences, sets, single items, or combinations of such structures. Single items are the simplest form of structure. Each element of a general pattern may contain a subsequence, a subtree, a subgraph, and so on, and such containment relationships can be defined recursively. Therefore, structural pattern mining can be considered as the most general form of frequent pattern mining.

- **Based on *application domain-specific semantics*:** Both data and applications can be very diverse, and therefore the patterns to be mined can differ largely based on their domain-specific semantics. Various kinds of application data include spatial data, temporal data, spatiotemporal data, multimedia data (e.g., image, audio, and video data), text data, time-series data, DNA and biological sequences, software programs, chemical compound structures, web structures, sensor networks, social and information networks, biological networks, data streams, and so on. This diversity can lead to dramatically different pattern mining methodologies.

- **Based on *data analysis usages*:** Frequent pattern mining often serves as an intermediate step for improved data understanding and more powerful data analysis. For example, it can be used as a feature extraction step for classification, which is often referred to as **pattern-based classification**. Similarly, **pattern-based clustering** has shown its strength at clustering high-dimensional data. For improved data understanding, patterns can be used for semantic annotation or contextual analysis. Pattern analysis can also be used in **recommender systems**, which recommend information items (e.g., books, movies, web pages) that are likely to be of interest to the user based on similar users' patterns. Different analysis tasks may require mining rather different kinds of patterns as well.

The next several sections present advanced methods and extensions of pattern mining, as well as their application. Section 7.2 discusses methods for mining multilevel patterns, multidimensional patterns, patterns and rules with continuous attributes, rare patterns, and negative patterns. Constraint-based pattern mining is studied in

Section 7.3. Section 7.4 explains how to mine high-dimensional and colossal patterns. The mining of compressed and approximate patterns is detailed in Section 7.5. Section 7.6 discusses the exploration and applications of pattern mining. More advanced topics regarding mining sequential and structural patterns, and pattern mining in complex and diverse kinds of data are briefly introduced in Chapter 13.

7.2 Pattern Mining in Multilevel, Multidimensional Space

This section focuses on methods for mining in multilevel, multidimensional space. In particular, you will learn about mining multilevel associations (Section 7.2.1), multidimensional associations (Section 7.2.2), quantitative association rules (Section 7.2.3), and rare patterns and negative patterns (Section 7.2.4). *Multilevel associations* involve concepts at different abstraction levels. *Multidimensional associations* involve more than one dimension or predicate (e.g., rules that relate what a customer *buys* to his or her *age*). *Quantitative association rules* involve numeric attributes that have an implicit ordering among values (e.g., *age*). *Rare patterns* are patterns that suggest interesting although rare item combinations. *Negative patterns* show negative correlations between items.

7.2.1 Mining Multilevel Associations

For many applications, strong associations discovered at high abstraction levels, though with high support, could be commonsense knowledge. We may want to drill down to find novel patterns at more detailed levels. On the other hand, there could be too many scattered patterns at low or primitive abstraction levels, some of which are just trivial specializations of patterns at higher levels. Therefore, it is interesting to examine how to develop effective methods for mining patterns at multiple abstraction levels, with sufficient flexibility for easy traversal among different abstraction spaces.

Example 7.1 **Mining multilevel association rules.** Suppose we are given the task-relevant set of transactional data in Table 7.1 for sales in an *AllElectronics* store, showing the items purchased for each transaction. The concept hierarchy for the items is shown in Figure 7.2. A concept hierarchy defines a sequence of mappings from a set of low-level concepts to a higher-level, more general concept set. Data can be generalized by replacing low-level concepts within the data by their corresponding higher-level concepts, or *ancestors*, from a concept hierarchy.

Figure 7.2's concept hierarchy has five levels, respectively referred to as levels 0 through 4, starting with level 0 at the root node for **all** (the most general abstraction level). Here, level 1 includes *computer, software, printer and camera,* and *computer accessory*; level 2 includes *laptop computer, desktop computer, office software, antivirus software,* etc.; and level 3 includes *Dell desktop computer, . . . , Microsoft office software,* etc. Level 4 is the most specific abstraction level of this hierarchy. It consists of the raw data values.

Table 7.1 Task-Relevant Data, *D*

TID	*Items Purchased*
T100	Apple 17″ MacBook Pro Notebook, HP Photosmart Pro b9180
T200	Microsoft Office Professional 2010, Microsoft Wireless Optical Mouse 5000
T300	Logitech VX Nano Cordless Laser Mouse, Fellowes GEL Wrist Rest
T400	Dell Studio XPS 16 Notebook, Canon PowerShot SD1400
T500	Lenovo ThinkPad X200 Tablet PC, Symantec Norton Antivirus 2010
…	…

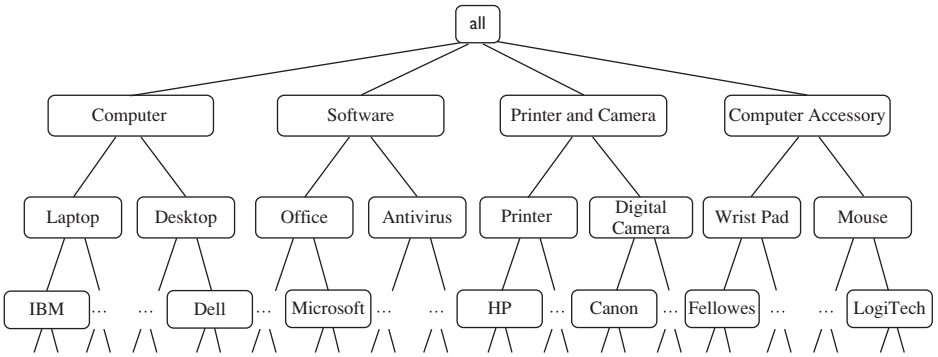

Figure 7.2 Concept hierarchy for *AllElectronics* computer items.

Concept hierarchies for nominal attributes are often implicit within the database schema, in which case they may be automatically generated using methods such as those described in Chapter 3. For our example, the concept hierarchy of Figure 7.2 was generated from data on product specifications. Concept hierarchies for numeric attributes can be generated using discretization techniques, many of which were introduced in Chapter 3. Alternatively, concept hierarchies may be specified by users familiar with the data such as store managers in the case of our example.

The items in Table 7.1 are at the lowest level of Figure 7.2's concept hierarchy. It is difficult to find interesting purchase patterns in such raw or primitive-level data. For instance, if "*Dell Studio XPS 16 Notebook*" or "*Logitech VX Nano Cordless Laser Mouse*" occurs in a very small fraction of the transactions, then it can be difficult to find strong associations involving these specific items. Few people may buy these items together, making it unlikely that the itemset will satisfy minimum support. However, we would expect that it is easier to find strong associations between generalized abstractions of these items, such as between "*Dell Notebook*" and "*Cordless Mouse.*" ■

Association rules generated from mining data at multiple abstraction levels are called **multiple-level** or **multilevel association rules**. Multilevel association rules can be

mined efficiently using concept hierarchies under a support-confidence framework. In general, a top-down strategy is employed, where counts are accumulated for the calculation of frequent itemsets at each concept level, starting at concept level 1 and working downward in the hierarchy toward the more specific concept levels, until no more frequent itemsets can be found. For each level, any algorithm for discovering frequent itemsets may be used, such as Apriori or its variations.

A number of variations to this approach are described next, where each variation involves "playing" with the support threshold in a slightly different way. The variations are illustrated in Figures 7.3 and 7.4, where nodes indicate an item or itemset that has been examined, and nodes with thick borders indicate that an examined item or itemset is frequent.

- **Using uniform minimum support for all levels** (referred to as **uniform support**): The same minimum support threshold is used when mining at each abstraction level. For example, in Figure 7.3, a minimum support threshold of 5% is used throughout (e.g., for mining from "*computer*" downward to "*laptop computer*"). Both "*computer*" and "*laptop computer*" are found to be frequent, whereas "*desktop computer*" is not.

  When a uniform minimum support threshold is used, the search procedure is simplified. The method is also simple in that users are required to specify only

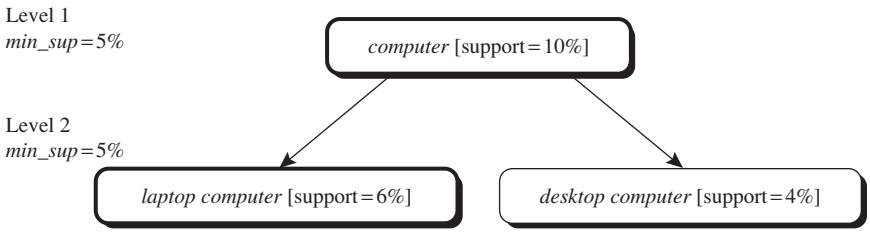

Figure 7.3 Multilevel mining with uniform support.

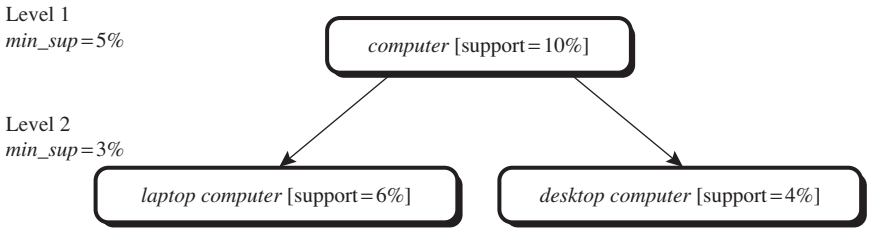

Figure 7.4 Multilevel mining with reduced support.

one minimum support threshold. An Apriori-like optimization technique can be adopted, based on the knowledge that an ancestor is a superset of its descendants: The search avoids examining itemsets containing any item of which the ancestors do not have minimum support.

The uniform support approach, however, has some drawbacks. It is unlikely that items at lower abstraction levels will occur as frequently as those at higher abstraction levels. If the minimum support threshold is set too high, it could miss some meaningful associations occurring at low abstraction levels. If the threshold is set too low, it may generate many uninteresting associations occurring at high abstraction levels. This provides the motivation for the next approach.

- **Using reduced minimum support at lower levels** (referred to as **reduced support**): Each abstraction level has its own minimum support threshold. The deeper the abstraction level, the smaller the corresponding threshold. For example, in Figure 7.4, the minimum support thresholds for levels 1 and 2 are 5% and 3%, respectively. In this way, "*computer,*" "*laptop computer,*" and "*desktop computer*" are all considered frequent.
- **Using item or group-based minimum support (referred to as group-based support)**: Because users or experts often have insight as to which groups are more important than others, it is sometimes more desirable to set up user-specific, item, or group-based minimal support thresholds when mining multilevel rules. For example, a user could set up the minimum support thresholds based on product price or on items of interest, such as by setting particularly low support thresholds for "*camera with price over $1000*" or "*Tablet PC,*" to pay particular attention to the association patterns containing items in these categories.

  For mining patterns with mixed items from groups with different support thresholds, usually the lowest support threshold among all the participating groups is taken as the support threshold in mining. This will avoid filtering out valuable patterns containing items from the group with the lowest support threshold. In the meantime, the minimal support threshold for each individual group should be kept to avoid generating uninteresting itemsets from each group. Other interestingness measures can be used after the itemset mining to extract truly interesting rules.

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

A serious side effect of mining multilevel association rules is its generation of many redundant rules across multiple abstraction levels due to the "ancestor" relationships among items. For example, consider the following rules where "*laptop computer*" is an ancestor of "*Dell laptop computer*" based on the concept hierarchy of Figure 7.2, and

where X is a variable representing customers who purchased items in *AllElectronics* transactions.

$$buys(X, \text{“}laptop\ computer\text{”}) \Rightarrow buys(X, \text{“}HP\ printer\text{”})$$
$$[support = 8\%, confidence = 70\%] \quad (7.4)$$

$$buys(X, \text{“}Dell\ laptop\ computer\text{”}) \Rightarrow buys(X, \text{“}HP\ printer\text{”})$$
$$[support = 2\%, confidence = 72\%] \quad (7.5)$$

“*If Rules (7.4) and (7.5) are both mined, then how useful is Rule (7.5)? Does it really provide any novel information?*” If the latter, less general rule does not provide new information, then it should be removed. Let's look at how this may be determined. A rule $R1$ is an **ancestor** of a rule $R2$, if $R1$ can be obtained by replacing the items in $R2$ by their ancestors in a concept hierarchy. For example, Rule (7.4) is an ancestor of Rule (7.5) because “*laptop computer*” is an ancestor of “*Dell laptop computer.*” Based on this definition, a rule can be considered redundant if its support and confidence are close to their “expected” values, based on an ancestor of the rule.

Example 7.2 **Checking redundancy among multilevel association rules.** Suppose that Rule (7.4) has a 70% confidence and 8% support, and that about one-quarter of all “*laptop computer*” sales are for “*Dell laptop computers.*” We may expect Rule (7.5) to have a confidence of around 70% (since all data samples of “*Dell laptop computer*” are also samples of “*laptop computer*”) and a support of around 2% (i.e., $8\% \times \frac{1}{4}$). If this is indeed the case, then Rule (7.5) is not interesting because it does not offer any additional information and is less general than Rule (7.4). ■

7.2.2 Mining Multidimensional Associations

So far, we have studied association rules that imply a single predicate, that is, the predicate *buys*. For instance, in mining our *AllElectronics* database, we may discover the Boolean association rule

$$buys(X, \text{“}digital\ camera\text{”}) \Rightarrow buys(X, \text{“}HP\ printer\text{”}). \quad (7.6)$$

Following the terminology used in multidimensional databases, we refer to each distinct predicate in a rule as a dimension. Hence, we can refer to Rule (7.6) as a **single-dimensional** or **intradimensional association rule** because it contains a single distinct predicate (e.g., *buys*) with multiple occurrences (i.e., the predicate occurs more than once within the rule). Such rules are commonly mined from transactional data.

Instead of considering transactional data only, sales and related information are often linked with relational data or integrated into a data warehouse. Such data stores are multidimensional in nature. For instance, in addition to keeping track of the items purchased in sales transactions, a relational database may record other attributes associated

with the items and/or transactions such as the item description or the branch location of the sale. Additional relational information regarding the customers who purchased the items (e.g., customer age, occupation, credit rating, income, and address) may also be stored. Considering each database attribute or warehouse dimension as a predicate, we can therefore mine association rules containing *multiple* predicates such as

$$age(X, \text{“20}\ldots\text{29”}) \wedge occupation(X, \text{“}student\text{”}) \Rightarrow buys(X, \text{“}laptop\text{”}). \quad (7.7)$$

Association rules that involve two or more dimensions or predicates can be referred to as **multidimensional association rules**. Rule (7.7) contains three predicates (*age*, *occupation*, and *buys*), each of which occurs *only once* in the rule. Hence, we say that it has **no repeated predicates**. Multidimensional association rules with no repeated predicates are called **interdimensional association rules**. We can also mine multidimensional association rules with repeated predicates, which contain multiple occurrences of some predicates. These rules are called **hybrid-dimensional association rules**. An example of such a rule is the following, where the predicate *buys* is repeated:

$$age(X, \text{“20}\ldots\text{29”}) \wedge buys(X, \text{“}laptop\text{”}) \Rightarrow buys(X, \text{“}HP\ printer\text{”}). \quad (7.8)$$

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

In the first approach, *quantitative attributes are discretized using predefined concept hierarchies*. This discretization occurs before mining. For instance, a concept hierarchy for *income* may be used to replace the original numeric values of this attribute by interval labels such as “0..20K,” “21K..30K,” “31K..40K,” and so on. Here, discretization is *static* and predetermined. Chapter 3 on data preprocessing gave several techniques for discretizing numeric attributes. The discretized numeric attributes, with their interval labels, can then be treated as nominal attributes (where each interval is considered a category). We refer to this as **mining multidimensional association rules using static discretization of quantitative attributes**.

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

Let’s study each of these approaches for mining multidimensional association rules. For simplicity, we confine our discussion to interdimensional association rules. Note that rather than searching for frequent itemsets (as is done for single-dimensional association rule mining), in multidimensional association rule mining we search for

frequent *predicate sets*. A ***k*-predicate set** is a set containing k conjunctive predicates. For instance, the set of predicates {*age, occupation, buys*} from Rule (7.7) is a 3-predicate set. Similar to the notation used for itemsets in Chapter 6, we use the notation L_k to refer to the set of frequent k-predicate sets.

7.2.3 Mining Quantitative Association Rules

As discussed earlier, relational and data warehouse data often involve quantitative attributes or measures. We can discretize quantitative attributes into multiple intervals and then treat them as nominal data in association mining. However, such simple discretization may lead to the generation of an enormous number of rules, many of which may not be useful. Here we introduce three methods that can help overcome this difficulty to discover novel association relationships: (1) a data cube method, (2) a clustering-based method, and (3) a statistical analysis method to uncover exceptional behaviors.

Data Cube–Based Mining of Quantitative Associations

In many cases quantitative attributes can be discretized before mining using predefined concept hierarchies or data discretization techniques, where numeric values are replaced by interval labels. Nominal attributes may also be generalized to higher conceptual levels if desired. If the resulting task-relevant data are stored in a relational table, then any of the frequent itemset mining algorithms we have discussed can easily be modified so as to find all frequent predicate sets. In particular, instead of searching on only one attribute like *buys*, we need to search through all of the relevant attributes, treating each attribute–value pair as an itemset.

Alternatively, the transformed multidimensional data may be used to construct a *data cube*. Data cubes are well suited for the mining of multidimensional association rules: They store aggregates (e.g., counts) in multidimensional space, which is essential for computing the support and confidence of multidimensional association rules. An overview of data cube technology was presented in Chapter 4. Detailed algorithms for data cube computation were given in Chapter 5. Figure 7.5 shows the lattice of cuboids defining a data cube for the dimensions *age, income*, and *buys*. The cells of an n-dimensional cuboid can be used to store the support counts of the corresponding n-predicate sets. The base cuboid aggregates the task-relevant data by *age, income*, and *buys*; the 2-D cuboid, (*age, income*), aggregates by *age* and *income*, and so on; the 0-D (apex) cuboid contains the total number of transactions in the task-relevant data.

Due to the ever-increasing use of data warehouse and OLAP technology, it is possible that a data cube containing the dimensions that are of interest to the user may already exist, fully or partially materialized. If this is the case, we can simply fetch the corresponding aggregate values or compute them using lower-level materialized aggregates, and return the rules needed using a rule generation algorithm. Notice that even in this case, the Apriori property can still be used to prune the search space. If a given k-predicate set has support *sup*, which does not satisfy minimum support, then further

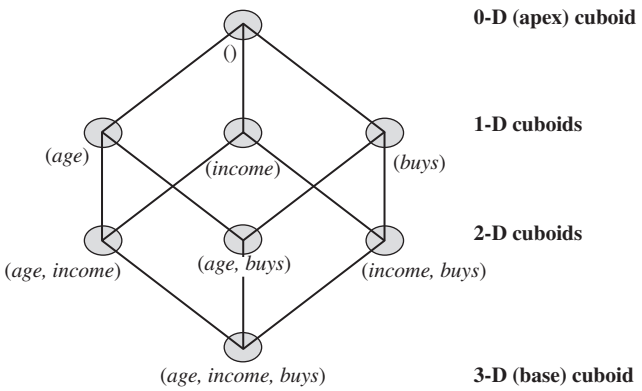

Figure 7.5 Lattice of cuboids, making up a 3-D data cube. Each cuboid represents a different group-by. The base cuboid contains the three predicates *age, income*, and *buys*.

exploration of this set should be terminated. This is because any more-specialized version of the *k*-itemset will have support no greater than *sup* and, therefore, will not satisfy minimum support either. In cases where no relevant data cube exists for the mining task, we must create one on-the-fly. This becomes an iceberg cube computation problem, where the minimum support threshold is taken as the iceberg condition (Chapter 5).

Mining Clustering-Based Quantitative Associations

Besides using discretization-based or data cube–based data sets to generate quantitative association rules, we can also generate *quantitative association rules* by clustering data in the quantitative dimensions. (Recall that objects within a cluster are similar to one another and dissimilar to those in other clusters.) The general assumption is that interesting frequent patterns or association rules are in general found at relatively dense clusters of quantitative attributes. Here, we describe a top-down approach and a bottom-up approach to clustering that finds quantitative associations.

A typical top-down approach for finding clustering-based quantitative frequent patterns is as follows. For each quantitative dimension, a standard clustering algorithm (e.g., *k*-means or a density-based clustering algorithm, as described in Chapter 10) can be applied to find clusters in this dimension that satisfy the minimum support threshold. For each cluster, we then examine the 2-D spaces generated by combining the cluster with a cluster or nominal value of another dimension to see if such a combination passes the minimum support threshold. If it does, we continue to search for clusters in this 2-D region and progress to even higher-dimensional combinations. The Apriori pruning still applies in this process: If, at any point, the support of a combination does not have minimum support, its further partitioning or combination with other dimensions cannot have minimum support either.

A bottom-up approach for finding clustering-based frequent patterns works by first clustering in high-dimensional space to form clusters with support that satisfies the minimum support threshold, and then projecting and merging those clusters in the space containing fewer dimensional combinations. However, for high-dimensional data sets, finding high-dimensional clustering itself is a tough problem. Thus, this approach is less realistic.

Using Statistical Theory to Disclose Exceptional Behavior

It is possible to discover quantitative association rules that disclose exceptional behavior, where "exceptional" is defined based on a statistical theory. For example, the following association rule may indicate exceptional behavior:

$$sex = female \Rightarrow meanwage = \$7.90/hr \ (overall\_mean\_wage = \$9.02/hr). \tag{7.9}$$

This rule states that the average wage for females is only \$7.90/hr. This rule is (subjectively) interesting because it reveals a group of people earning a significantly lower wage than the average wage of \$9.02/hr. (If the average wage was close to \$7.90/hr, then the fact that females also earn \$7.90/hr would be "uninteresting.")

An integral aspect of our definition involves applying statistical tests to confirm the validity of our rules. That is, Rule (7.9) is only accepted if a statistical test (in this case, a Z-test) confirms that with high confidence it can be inferred that the mean wage of the female population is indeed lower than the mean wage of the rest of the population. (The above rule was mined from a real database based on a 1985 U.S. census.)

An association rule under the new definition is a rule of the form:

$$population\_subset \Rightarrow \textbf{mean}\_of\_values\_for\_the\_subset, \tag{7.10}$$

where the mean of the subset is significantly different from the mean of its complement in the database (and this is validated by an appropriate statistical test).

7.2.4 Mining Rare Patterns and Negative Patterns

All the methods presented so far in this chapter have been for mining frequent patterns. Sometimes, however, it is interesting to find patterns that are rare instead of frequent, or patterns that reflect a negative correlation between items. These patterns are respectively referred to as rare patterns and negative patterns. In this subsection, we consider various ways of defining rare patterns and negative patterns, which are also useful to mine.

Example 7.3 **Rare patterns and negative patterns.** In jewelry sales data, sales of diamond watches are rare; however, patterns involving the selling of diamond watches could be interesting. In supermarket data, if we find that customers frequently buy Coca-Cola Classic or Diet Coke but not both, then buying Coca-Cola Classic and buying Diet Coke together

is considered a negative (correlated) pattern. In car sales data, a dealer sells a few fuel-thirsty vehicles (e.g., SUVs) to a given customer, and then later sells hybrid mini-cars to the same customer. Even though buying SUVs and buying hybrid mini-cars may be negatively correlated events, it can be interesting to discover and examine such exceptional cases. ■

An **infrequent** (or **rare**) **pattern** is a pattern with a frequency support that is *below* (or *far below*) a user-specified minimum support threshold. However, since the occurrence frequencies of the majority of itemsets are usually below or even far below the minimum support threshold, it is desirable in practice for users to specify other conditions for rare patterns. For example, if we want to find patterns containing at least one item with a value that is over $500, we should specify such a constraint explicitly. Efficient mining of such itemsets is discussed under mining multidimensional associations (Section 7.2.1), where the strategy is to adopt multiple (e.g., item- or group-based) minimum support thresholds. Other applicable methods are discussed under constraint-based pattern mining (Section 7.3), where user-specified constraints are pushed deep into the iterative mining process.

There are various ways we could define a negative pattern. We will consider three such definitions.

Definition 7.1: If itemsets X and Y are both frequent but rarely occur together (i.e., $sup(X \cup Y) < sup(X) \times sup(Y)$), then itemsets X and Y are **negatively correlated**, and the pattern $X \cup Y$ is a **negatively correlated pattern**. If $sup(X \cup Y) \ll sup(X) \times sup(Y)$, then X and Y are **strongly negatively correlated**, and the pattern $X \cup Y$ is a **strongly negatively correlated pattern**. □

This definition can easily be extended for patterns containing k-itemsets for $k > 2$.

A problem with the definition, however, is that it is not *null-invariant*. That is, its value can be misleadingly influenced by null transactions, where a *null-transaction* is a transaction that does not contain any of the itemsets being examined (Section 6.3.3). This is illustrated in Example 7.4.

Example 7.4 Null-transaction problem with Definition 7.1. If there are a lot of null-transactions in the data set, then the number of null-transactions rather than the patterns observed may strongly influence a measure's assessment as to whether a pattern is negatively correlated. For example, suppose a sewing store sells needle packages A and B. The store sold 100 packages each of A and B, but only one transaction contains both A and B. Intuitively, A is negatively correlated with B since the purchase of one does not seem to encourage the purchase of the other.

Let's see how the above Definition 7.1 handles this scenario. If there are 200 transactions, we have $sup(A \cup B) = 1/200 = 0.005$ and $sup(A) \times sup(B) = 100/200 \times 100/200 = 0.25$. Thus, $sup(A \cup B) \ll sup(A) \times sup(B)$, and so Definition 7.1 indicates that A and B are strongly negatively correlated. What if, instead of only 200 transactions in the database, there are 10^6? In this case, there are many null-transactions, that is, many contain neither A nor B. How does the definition hold up? It computes $sup(A \cup B) = 1/10^6$ and $sup(X) \times sup(Y) = 100/10^6 \times 100/10^6 = 1/10^8$.

Thus, $sup(A \cup B) \gg sup(X) \times sup(Y)$, which contradicts the earlier finding even though the number of occurrences of A and B has not changed. The measure in Definition 7.1 is not null-invariant, where *null-invariance* is essential for quality interestingness measures as discussed in Section 6.3.3. ■

Definition 7.2: If X and Y are strongly negatively correlated, then

$$sup(X \cup \overline{Y}) \times sup(\overline{X} \cup Y) \gg sup(X \cup Y) \times sup(\overline{X} \cup \overline{Y}).$$

Is this measure null-invariant? □

Example 7.5 Null-transaction problem with Definition 7.2. Given our needle package example, when there are in total 200 transactions in the database, we have

$$sup(A \cup \overline{B}) \times sup(\overline{A} \cup B) = 99/200 \times 99/200 = 0.245$$
$$\gg sup(A \cup B) \times sup(\overline{A} \cup \overline{B}) = 199/200 \times 1/200 \approx 0.005,$$

which, according to Definition 7.2, indicates that A and B are strongly negatively correlated. What if there are 10^6 transactions in the database? The measure would compute

$$sup(A \cup \overline{B}) \times sup(\overline{A} \cup B) = 99/10^6 \times 99/10^6 = 9.8 \times 10^{-9}$$
$$\ll sup(A \cup B) \times sup(\overline{A} \cup \overline{B}) = 199/10^6 \times (10^6 - 199)/10^6 \approx 1.99 \times 10^{-4}.$$

This time, the measure indicates that A and B are positively correlated, hence, a contradiction. The measure is not null-invariant. ■

As a third alternative, consider Definition 7.3, which is based on the Kulczynski measure (i.e., the average of conditional probabilities). It follows the spirit of interestingness measures introduced in Section 6.3.3.

Definition 7.3: Suppose that itemsets X and Y are both frequent, that is, $sup(X) \geq min\_sup$ and $sup(Y) \geq min\_sup$, where min_sup is the minimum support threshold. If $(P(X|Y) + P(Y|X))/2 < \epsilon$, where ϵ is a negative pattern threshold, then pattern $X \cup Y$ is a **negatively correlated pattern**. □

Example 7.6 Negatively correlated patterns using Definition 7.3, based on the Kulczynski measure. Let's reexamine our needle package example. Let min_sup be 0.01% and $\epsilon = 0.02$. When there are 200 transactions in the database, we have $sup(A) = sup(B) = 100/200 = 0.5 > 0.01\%$ and $(P(B|A) + P(A|B))/2 = (0.01 + 0.01)/2 < 0.02$; thus A and B are negatively correlated. Does this still hold true if we have many more transactions? When there are 10^6 transactions in the database, the measure computes $sup(A) = sup(B) = 100/10^6 = 0.01\% \geq 0.01\%$ and $(P(B|A) + P(A|B))/2 = (0.01 + 0.01)/2 < 0.02$, again indicating that A and B are negatively correlated. This matches our intuition. The measure does not have the null-invariance problem of the first two definitions considered.

Let's examine another case: Suppose that among 100,000 transactions, the store sold 1000 needle packages of A but only 10 packages of B; however, every time package B is

sold, package A is also sold (i.e., they appear in the same transaction). In this case, the measure computes $(P(B|A) + P(A|B))/2 = (0.01 + 1)/2 = 0.505 \gg 0.02$, which indicates that A and B are positively correlated instead of negatively correlated. This also matches our intuition. ■

With this new definition of negative correlation, efficient methods can easily be derived for mining negative patterns in large databases. This is left as an exercise for interested readers.

7.3 Constraint-Based Frequent Pattern Mining

A data mining process may uncover thousands of rules from a given data set, most of which end up being unrelated or uninteresting to users. Often, users have a good sense of which "direction" of mining may lead to interesting patterns and the "form" of the patterns or rules they want to find. They may also have a sense of "conditions" for the rules, which would eliminate the discovery of certain rules that they know would not be of interest. Thus, a good heuristic is to have the users specify such intuition or expectations as *constraints* to confine the search space. This strategy is known as **constraint-based mining**. The constraints can include the following:

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

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

The first four constraint types have already been addressed in earlier sections of this book and this chapter. In this section, we discuss the use of *rule constraints* to focus the mining task. This form of constraint-based mining allows users to describe the rules that they would like to uncover, thereby making the data mining process more *effective*. In addition, a sophisticated mining query optimizer can be used to exploit the constraints specified by the user, thereby making the mining process more *efficient*.

Constraint-based mining encourages interactive exploratory mining and analysis. In Section 7.3.1, you will study metarule-guided mining, where syntactic rule constraints are specified in the form of rule templates. Section 7.3.2 discusses the use of *pattern space pruning* (which prunes patterns being mined) and *data space pruning* (which prunes pieces of the data space for which further exploration cannot contribute to the discovery of patterns satisfying the constraints).

For pattern space pruning, we introduce three classes of properties that facilitate constraint-based search space pruning: *antimonotonicity, monotonicity*, and *succinctness*. We also discuss a special class of constraints, called *convertible constraints*, where by proper data ordering, the constraints can be pushed deep into the iterative mining process and have the same pruning power as monotonic or antimonotonic constraints. For data space pruning, we introduce two classes of properties—*data succinctness* and *data antimonotonicty*—and study how they can be integrated within a data mining process.

For ease of discussion, we assume that the user is searching for association rules. The procedures presented can be easily extended to the mining of correlation rules by adding a correlation measure of interestingness to the support-confidence framework.

7.3.1 Metarule-Guided Mining of Association Rules

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

Example 7.7 **Metarule-guided mining.** Suppose that as a market analyst for *AllElectronics* you have access to the data describing customers (e.g., customer age, address, and credit rating) as well as the list of customer transactions. You are interested in finding associations between customer traits and the items that customers buy. However, rather than finding *all* of the association rules reflecting these relationships, you are interested only in determining which pairs of customer traits promote the sale of office software. A metarule can be used to specify this information describing the form of rules you are interested in finding. An example of such a metarule is

$$P_1(X, Y) \wedge P_2(X, W) \Rightarrow buys(X, \text{"}office\ software\text{"}), \tag{7.11}$$

where P_1 and P_2 are **predicate variables** that are instantiated to attributes from the given database during the mining process, X is a variable representing a customer, and Y and W take on values of the attributes assigned to P_1 and P_2, respectively. Typically, a user will specify a list of attributes to be considered for instantiation with P_1 and P_2. Otherwise, a default set may be used.

In general, a metarule forms a hypothesis regarding the relationships that the user is interested in probing or confirming. The data mining system can then search for

rules that match the given metarule. For instance, Rule (7.12) matches or **complies with** Metarule (7.11):

$$age(X, \text{“30..39”}) \wedge income(X, \text{“41}K\text{..60}K\text{”}) \Rightarrow buys(X, \text{“}office\ software\text{”}). \quad (7.12)$$

■

“*How can metarules be used to guide the mining process?*” Let’s examine this problem closely. Suppose that we wish to mine interdimensional association rules such as in Example 7.7. A metarule is a rule template of the form

$$P_1 \wedge P_2 \wedge \cdots \wedge P_l \Rightarrow Q_1 \wedge Q_2 \wedge \cdots \wedge Q_r, \quad (7.13)$$

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

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

This is a typical case of mining multidimensional association rules. By extending such methods using the constraint-pushing techniques described in the following section, we can derive efficient methods for metarule-guided mining.

7.3.2 Constraint-Based Pattern Generation: Pruning Pattern Space and Pruning Data Space

Rule constraints specify expected set/subset relationships of the variables in the mined rules, constant initiation of variables, and constraints on aggregate functions and other forms of constraints. Users typically employ their knowledge of the application or data to specify rule constraints for the mining task. These rule constraints may be used together with, or as an alternative to, metarule-guided mining. In this section, we examine rule constraints as to how they can be used to make the mining process more efficient. Let’s study an example where rule constraints are used to mine hybrid-dimensional association rules.

Example 7.8 **Constraints for mining association rules.** Suppose that *AllElectronics* has a sales multidimensional database with the following interrelated relations:

- *item(item_ID, item_name, description, category, price)*
- *sales(transaction_ID, day, month, year, store_ID, city)*
- *trans_item(item_ID, transaction_ID)*

Here, the *item* table contains attributes *item_ID, item_name, description, category*, and *price*; the *sales* table contains attributes *transaction_ID day, month, year, store_ID*, and *city*; and the two tables are linked via the foreign key attributes, *item_ID* and *transaction_ID*, in the table *trans_item*.

Suppose our association mining query is "*Find the patterns or rules about the sales of which cheap items (where the sum of the prices is less than $10) may promote (i.e., appear in the same transaction) the sales of which expensive items (where the minimum price is $50), shown in the sales in Chicago in 2010.*"

This query contains the following four constraints: (1) $sum(I.price) < \$10$, where I represents the *item_ID* of a cheap item; (2) $min(J.price) \geq \$50$), where J represents the *item_ID* of an expensive item; (3) $T.city = Chicago$; and (4) $T.year = 2010$, where T represents a *transaction_ID*. For conciseness, we do not show the mining query explicitly here; however, the constraints' context is clear from the mining query semantics. ■

Dimension/level constraints and interestingness constraints can be applied after mining to filter out discovered rules, although it is generally more efficient and less expensive to use them *during* mining to help prune the search space. Dimension/level constraints were discussed in Section 7.2, and interestingness constraints, such as support, confidence, and correlation measures, were discussed in Chapter 6. Let's focus now on rule constraints.

"*How can we use rule constraints to prune the search space? More specifically, what kind of rule constraints can be 'pushed' deep into the mining process and still ensure the completeness of the answer returned for a mining query?*"

In general, an efficient frequent pattern mining processor can prune its search space during mining in two major ways: *pruning pattern search space* and *pruning data search space*. The former checks candidate patterns and decides whether a pattern can be pruned. Applying the Apriori property, it prunes a pattern if no superpattern of it can be generated in the remaining mining process. The latter checks the data set to determine whether the particular data piece will be able to contribute to the subsequent generation of satisfiable patterns (for a particular pattern) in the remaining mining process. If not, the data piece is pruned from further exploration. A constraint that may facilitate pattern space pruning is called a *pattern pruning constraint*, whereas one that can be used for data space pruning is called a *data pruning constraint*.

Pruning Pattern Space with Pattern Pruning Constraints

Based on how a constraint may interact with the pattern mining process, there are five categories of pattern mining constraints: (1) *antimonotonic,* (2) *monotonic,* (3) *succinct,* (4) *convertible,* and (5) *inconvertible*. For each category, we use an example to show its characteristics and explain how such kinds of constraints can be used in the mining process.

The first category of constraints is **antimonotonic**. Consider the rule constraint "$sum(I.price) \leq \$100$" of Example 7.8. Suppose we are using the Apriori framework, which explores itemsets of size k at the kth iteration. If the price summation of the items in a candidate itemset is no less than \$100, this itemset can be pruned from the search space, since adding more items into the set (assuming price is no less than zero) will only make it more expensive and thus will never satisfy the constraint. In other words, if an itemset does not satisfy this rule constraint, none of its supersets can satisfy the constraint. If a rule constraint obeys this property, it is **antimonotonic**. Pruning by antimonotonic constraints can be applied at each iteration of Apriori-style algorithms to help improve the efficiency of the overall mining process while guaranteeing completeness of the data mining task.

The Apriori property, which states that all nonempty subsets of a frequent itemset must also be frequent, is antimonotonic. If a given itemset does not satisfy minimum support, none of its supersets can. This property is used at each iteration of the Apriori algorithm to reduce the number of candidate itemsets examined, thereby reducing the search space for association rules.

Other examples of antimonotonic constraints include "$min(J.price) \geq \$50$," "$count(I) \leq 10$," and so on. Any itemset that violates either of these constraints can be discarded since adding more items to such itemsets can never satisfy the constraints. Note that a constraint such as "$avg(I.price) \leq \$10$" is not antimonotonic. For a given itemset that does not satisfy this constraint, a superset created by adding some (cheap) items may result in satisfying the constraint. Hence, pushing this constraint inside the mining process will not guarantee completeness of the data mining task. A list of SQL primitives–based constraints is given in the first column of Table 7.2. The antimonotonicity of the constraints is indicated in the second column. To simplify our discussion, only existence operators (e.g., $=$, $\in$, but not $\neq$, $\notin$) and comparison (or containment) operators with equality (e.g., $\leq$, $\subseteq$) are given.

The second category of constraints is **monotonic**. If the rule constraint in Example 7.8 were "$sum(I.price) \geq \$100$," the constraint-based processing method would be quite different. If an itemset I satisfies the constraint, that is, the sum of the prices in the set is no less than \$100, further addition of more items to I will increase cost and will always satisfy the constraint. Therefore, further testing of this constraint on itemset I becomes redundant. In other words, if an itemset satisfies this rule constraint, so do all of its supersets. If a rule constraint obeys this property, it is **monotonic**. Similar rule monotonic constraints include "$min(I.price) \leq \$10$," "$count(I) \geq 10$," and so on. The monotonicity of the list of SQL primitives–based constraints is indicated in the third column of Table 7.2.

The third category is **succinct constraints**. For this constraints category, we can *enumerate all and only those sets that are guaranteed to satisfy the constraint.* That is, if a rule constraint is **succinct**, we can directly generate precisely the sets that satisfy it, even before support counting begins. This avoids the substantial overhead of the generate-and-test paradigm. In other words, such constraints are *precounting prunable.* For example, the constraint "$min(J.price) \geq \$50$" in Example 7.8 is succinct because we can explicitly and precisely generate all the itemsets that satisfy the constraint.

Table 7.2 Characterization of Commonly Used SQL-Based Pattern Pruning Constraints

Constraint	*Antimonotonic*	*Monotonic*	*Succinct*
$v \in S$	no	yes	yes
$S \supseteq V$	no	yes	yes
$S \subseteq V$	yes	no	yes
$min(S) \leq v$	no	yes	yes
$min(S) \geq v$	yes	no	yes
$max(S) \leq v$	yes	no	yes
$max(S) \geq v$	no	yes	yes
$count(S) \leq v$	yes	no	weakly
$count(S) \geq v$	no	yes	weakly
$sum(S) \leq v\ (\forall a \in S, a \geq 0)$	yes	no	no
$sum(S) \geq v\ (\forall a \in S, a \geq 0)$	no	yes	no
$range(S) \leq v$	yes	no	no
$range(S) \geq v$	no	yes	no
$avg(S)\ \theta\ v, \theta \in \{\leq, \geq\}$	convertible	convertible	no
$support(S) \geq \xi$	yes	no	no
$support(S) \leq \xi$	no	yes	no
$all\_confidence(S) \geq \xi$	yes	no	no
$all\_confidence(S) \leq \xi$	no	yes	no

Specifically, such a set must consist of a nonempty set of items that have a price no less than \$50. It is of the form S, where $S \neq \emptyset$ is a subset of the set of all items with prices no less than \$50. Because there is a precise "formula" for generating all the sets satisfying a succinct constraint, there is no need to iteratively check the rule constraint during the mining process. The succinctness of the list of SQL primitives–based constraints is indicated in the fourth column of Table 7.2.[2]

The fourth category is **convertible constraints**. Some constraints belong to none of the previous three categories. However, if the items in the itemset are arranged in a particular order, the constraint may become monotonic or antimonotonic with regard to the frequent itemset mining process. For example, the constraint "$avg(I.price) \leq \$10$" is neither antimonotonic nor monotonic. However, if items in a transaction are added to an itemset in price-ascending order, the constraint becomes *antimonotonic*, because if an itemset I violates the constraint (i.e., with an average price greater than \$10), then further addition of more expensive items into the itemset will never make it

[2]For constraint $count(S) \leq v$ (and similarly for $count(S) \geq v$), we can have a member generation function based on a cardinality constraint (i.e., $\{X \mid X \subseteq Itemset \wedge |X| \leq v\}$). Member generation in this manner is of a different flavor and thus is called *weakly succinct*.

satisfy the constraint. Similarly, if items in a transaction are added to an itemset in price-descending order, it becomes *monotonic*, because if the itemset satisfies the constraint (i.e., with an average price no greater than \$10), then adding cheaper items into the current itemset will still make the average price no greater than \$10. Aside from "$avg(S) \leq v$" and "$avg(S) \geq v$," given in Table 7.2, there are many other convertible constraints such as "$variance(S) \geq v$" "$standard\_deviation(S) \geq v$," and so on.

Note that the previous discussion does not imply that every constraint is convertible. For example, "$sum(S)\, \theta\, v$," where $\theta \in \{\leq, \geq\}$ and each element in S could be of any real value, is not convertible. Therefore, there is yet a fifth category of constraints, called **inconvertible constraints**. The good news is that although there still exist some tough constraints that are not convertible, most simple SQL expressions with built-in SQL aggregates belong to one of the first four categories to which efficient constraint mining methods can be applied.

Pruning Data Space with Data Pruning Constraints

The second way of search space pruning in constraint-based frequent pattern mining is *pruning data space*. This strategy prunes pieces of data if they will not contribute to the subsequent generation of satisfiable patterns in the mining process. We consider two properties: *data succinctness* and *data antimonotonicity*.

Constraints are **data-succinct** if they can be used *at the beginning of a pattern mining process* to prune the data subsets that cannot satisfy the constraints. For example, if a mining query requires that the mined pattern must contain *digital camera*, then any transaction that does not contain *digital camera* can be pruned at the beginning of the mining process, which effectively reduces the data set to be examined.

Interestingly, many constraints are **data-antimonotonic** in the sense that *during the mining process*, if a data entry cannot satisfy a data-antimonotonic constraint based on the current pattern, then it can be pruned. We prune it because it will not be able to contribute to the generation of any superpattern of the current pattern in the remaining mining process.

Example 7.9 **Data antimonotonicity.** A mining query requires that $C_1 : sum(I.price) \geq \$100$, that is, the sum of the prices of the items in the mined pattern must be no less than \$100. Suppose that the current frequent itemset, S, does not satisfy constraint C_1 (say, because the sum of the prices of the items in S is \$50). If the remaining frequent items in a transaction T_i are such that, say, $\{i_2.price = \$5, i_5.price = \$10, i_8.price = \$20\}$, then T_i will not be able to make S satisfy the constraint. Thus, T_i cannot contribute to the patterns to be mined from S, and thus can be pruned.

Note that such pruning cannot be done at the beginning of the mining because at that time, we do not know yet if the total sum of the prices of all the items in T_i will be over \$100 (e.g., we may have $i_3.price = \$80$). However, during the iterative mining process, we may find some items (e.g., i_3) that are not frequent with S in the transaction data set, and thus they would be pruned. Therefore, such checking and pruning should be enforced at each iteration to reduce the data search space. ■

Notice that constraint C_1 is a monotonic constraint with respect to pattern space pruning. As we have seen, this constraint has very limited power for reducing the search space in pattern pruning. However, the same constraint can be used for effective reduction of the data search space.

For an antimonotonic constraint, such as $C_2 : sum(I.price) \leq \$100$, we can prune both pattern and data search spaces at the same time. Based on our study of pattern pruning, we already know that the current itemset can be pruned if the sum of the prices in it is over \$100 (since its further expansion can never satisfy C_2). At the same time, we can also prune any remaining items in a transaction T_i that cannot make the constraint C_2 valid. For example, if the sum of the prices of items in the current itemset S is \$90, any patterns over \$10 in the remaining frequent items in T_i can be pruned. If none of the remaining items in T_i can make the constraint valid, the entire transaction T_i should be pruned.

Consider pattern constraints that are neither antimonotonic nor monotonic such as "$C_3 : avg(I.price) \leq 10$." These can be data-antimonotonic because if the remaining items in a transaction T_i cannot make the constraint valid, then T_i can be pruned as well. Therefore, data-antimonotonic constraints can be quite useful for constraint-based data space pruning.

Notice that search space pruning by data antimonotonicity is confined only to a pattern growth–based mining algorithm because the pruning of a data entry is determined based on whether it can contribute to a specific pattern. Data antimonotonicity cannot be used for pruning the data space if the Apriori algorithm is used because the data are associated with all of the currently active patterns. At any iteration, there are usually many active patterns. A data entry that cannot contribute to the formation of the superpatterns of a given pattern may still be able to contribute to the superpattern of other active patterns. Thus, the power of data space pruning can be very limited for nonpattern growth–based algorithms.

7.4 Mining High-Dimensional Data and Colossal Patterns

The frequent pattern mining methods presented so far handle large data sets having a small number of dimensions. However, some applications may need to mine *high-dimensional data* (i.e., data with hundreds or thousands of dimensions). Can we use the methods studied so far to mine high-dimensional data? The answer is unfortunately negative because the search spaces of such typical methods grow exponentially with the number of dimensions.

Researchers have overcome this difficulty in two directions. One direction extends a pattern growth approach by further exploring the vertical data format to handle data sets with a large number of *dimensions* (also called *features* or *items*, e.g., genes) but a *small* number of *rows* (also called *transactions* or *tuples*, e.g., samples). This is useful in applications like the analysis of gene expressions in bioinformatics, for example, where we often need to analyze microarray data that contain a *large* number of genes

(e.g., 10,000 to 100,000) but only a *small* number of samples (e.g., 100 to 1000). The other direction develops a new mining methodology, called *Pattern-Fusion*, which mines *colossal patterns*, that is, patterns of very long length.

Let's first briefly examine the first direction, in particular, a pattern growth–based row enumeration approach. Its general philosophy is to explore the *vertical data format*, as described in Section 6.2.5, which is also known as **row enumeration**. Row enumeration differs from traditional column (i.e., item) enumeration (also known as the *horizontal data format*). In traditional column enumeration, the data set, D, is viewed as a set of rows, where each row consists of an itemset. In row enumeration, the data set is instead viewed as an itemset, each consisting of a set of *row_ID*s indicating where the item appears in the traditional view of D. The original data set, D, can easily be transformed into a transposed data set, T. A data set with a small number of rows but a large number of dimensions is then transformed into a transposed data set with a large number of rows but a small number of dimensions. Efficient pattern growth methods can then be developed on such relatively low-dimensional data sets. The details of such an approach are left as an exercise for interested readers.

The remainder of this section focuses on the second direction. We introduce Pattern-Fusion, a new mining methodology that mines *colossal patterns* (i.e., patterns of very long length). This method takes leaps in the pattern search space, leading to a good approximation of the complete set of colossal frequent patterns.

7.4.1 Mining Colossal Patterns by Pattern-Fusion

Although we have studied methods for mining frequent patterns in various situations, many applications have hidden patterns that are tough to mine, due mainly to their immense length or size. Consider bioinformatics, for example, where a common activity is DNA or microarray data analysis. This involves mapping and analyzing very long DNA and protein sequences. Researchers are more interested in finding large patterns (e.g., long sequences) than finding small ones since larger patterns usually carry more significant meaning. We call these large patterns *colossal* patterns, as distinguished from patterns with large support sets. Finding colossal patterns is challenging because incremental mining tends to get "trapped" by an explosive number of midsize patterns before it can even reach candidate patterns of large size. This is illustrated in Example 7.10.

Example 7.10 The challenge of mining colossal patterns. Consider a 40×40 square table where each row contains the integers 1 through 40 in increasing order. Remove the integers on the diagonal, and this gives a 40×39 table. Add 20 identical rows to the bottom of the table, where each row contains the integers 41 through 79 in increasing order, resulting in a 60×39 table (Figure 7.6). We consider each row as a transaction and set the minimum support threshold at 20. The table has an exponential number (i.e., $\binom{40}{20}$) of midsize closed/maximal frequent patterns of size 20, but only one that is colossal: $\alpha = (41, 42, \ldots, 79)$ of size 39. None of the frequent pattern mining algorithms that we have introduced so far can complete execution in a reasonable amount of time.

row/col	1	2	3	4	…	38	39
1	2	3	4	5	…	39	40
2	1	3	4	5	…	39	40
3	1	2	4	5	…	39	40
4	1	2	3	5	…	39	40
5	1	2	3	4	…	39	40
…	…	…	…	…	…	…	…
39	1	2	3	4	…	38	40
40	1	2	3	4	…	38	39
41	41	42	43	44	…	78	79
42	41	42	43	44	…	78	79
…	…	…	…	…	…	…	…
60	41	42	43	44	…	78	79

Figure 7.6 A simple colossal patterns example: The data set contains an exponential number of midsize patterns of size 20 but only one that is colossal, namely $(41, 42, \ldots, 79)$.

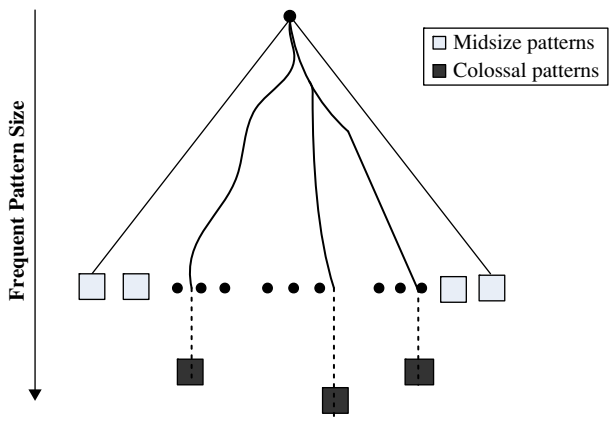

Figure 7.7 Synthetic data that contain some colossal patterns but exponentially many midsize patterns.

The pattern search space is similar to that in Figure 7.7, where midsize patterns largely outnumber colossal patterns. ■

All of the pattern mining strategies we have studied so far, such as Apriori and FP-growth, use an incremental growth strategy by nature, that is, they increase the length of candidate patterns by one at a time. Breadth-first search methods like Apriori cannot bypass the generation of an explosive number of midsize patterns generated,

making it impossible to reach colossal patterns. Even depth-first search methods like FP-growth can be easily trapped in a huge amount of subtrees before reaching colossal patterns. Clearly, a completely new mining methodology is needed to overcome such a hurdle.

A new mining strategy called *Pattern-Fusion* was developed, which fuses a small number of shorter frequent patterns into colossal pattern candidates. It thereby takes leaps in the pattern search space and avoids the pitfalls of both breadth-first and depth-first searches. This method finds a good approximation to the complete set of *colossal* frequent patterns.

The Pattern-Fusion method has the following major characteristics. First, it traverses the tree in a bounded-breadth way. Only a fixed number of patterns in a bounded-size candidate pool are used as starting nodes to search downward in the pattern tree. As such, it avoids the problem of exponential search space.

Second, Pattern-Fusion has the capability to identify "shortcuts" whenever possible. Each pattern's growth is not performed with one-item addition, but with an agglomeration of multiple patterns in the pool. These shortcuts direct Pattern-Fusion much more rapidly down the search tree toward the colossal patterns. Figure 7.8 conceptualizes this mining model.

As Pattern-Fusion is designed to give an approximation to the colossal patterns, a quality evaluation model is introduced to assess the patterns returned by the algorithm. An empirical study verifies that Pattern-Fusion is able to efficiently return high-quality results.

Let's examine the Pattern-Fusion method in more detail. First, we introduce the concept of **core pattern**. For a pattern α, an itemset $\beta \subseteq \alpha$ is said to be a τ*-core pattern* of α if $\frac{|D_\alpha|}{|D_\beta|} \geq \tau$, $0 < \tau \leq 1$, where $|D_\alpha|$ is the number of patterns containing α in database

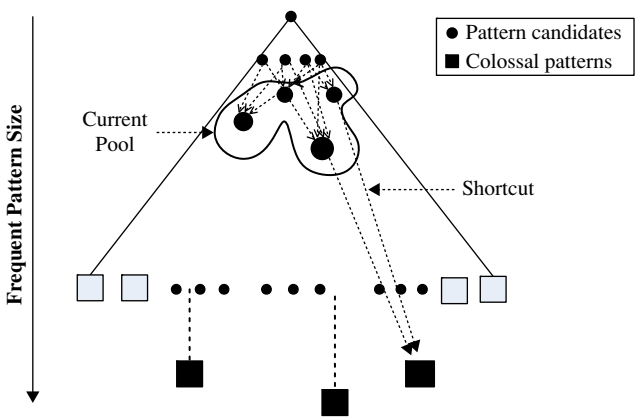

Figure 7.8 Pattern tree traversal: Candidates are taken from a pool of patterns, which results in shortcuts through pattern space to the colossal patterns.

D. τ is called the *core ratio*. A pattern α is (d, τ)-*robust* if d is the maximum number of items that can be removed from α for the resulting pattern to remain a τ-core pattern of α, that is,

$$d = \max_{\beta}\{|\alpha| - |\beta| | \beta \subseteq \alpha, \text{ and } \beta \text{ is a } \tau\text{-core pattern of } \alpha\}.$$

Example 7.11 **Core patterns.** Figure 7.9 shows a simple transaction database of four distinct transactions, each with 100 duplicates: $\{\alpha_1 = (abe),\ \alpha_2 = (bcf),\ \alpha_3 = (acf),\ \alpha_4 = (abcfe)\}$. If we set $\tau = 0.5$, then (ab) is a core pattern of α_1 because (ab) is contained only by α_1 and α_4. Therefore, $\frac{|D_{\alpha_1}|}{|D_{(ab)}|} = \frac{100}{200} \geq \tau$. α_1 is $(2, 0.5)$-robust while α_4 is $(4, 0.5)$-robust. The table also shows that larger patterns (e.g., $(abcfe)$) have far more core patterns than smaller ones (e.g., (bcf)). ■

From Example 7.11, we can deduce that large or colossal patterns have far more core patterns than smaller patterns do. Thus, a colossal pattern is more robust in the sense that *if a small number of items are removed from the pattern, the resulting pattern would have a similar support set.* The larger the pattern size, the more prominent this robustness. Such a robustness relationship between a colossal pattern and its corresponding core patterns can be extended to multiple levels. The lower-level core patterns of a colossal pattern are called **core descendants**.

Given a small c, a colossal pattern usually has far more core descendants of size c than a smaller pattern. This means that if we were to draw randomly from the complete set of patterns of size c, we would be more likely to pick a core descendant of a colossal pattern than that of a smaller pattern. In Figure 7.9, consider the complete set of patterns of size $c = 2$, which contains $\binom{5}{2} = 10$ patterns in total. For illustrative purposes, let's assume that the larger pattern, $abcef$, is colossal. The probability of being able to randomly draw a core descendant of $abcef$ is 0.9. Contrast this to the probability of randomly drawing a core descendent of smaller (noncolossal) patterns, which is at most 0.3. Therefore, a colossal pattern can be generated by merging a proper set of

Transactions (# of Transactions)	*Core Patterns* ($\tau = 0.5$)
(abe) (100)	(abe), (ab), (be), (ae), (e)
(bcf) (100)	(bcf), (bc), (bf)
(acf) (100)	(acf), (ac), (af)
$(abcef)$ (100)	(ab), (ac), (af), (ae), (bc), (bf), (be), (ce), (fe), (e), (abc), (abf), (abe), (ace), (acf), (afe), (bcf), (bce), (bfe), (cfe), $(abcf)$, $(abce)$, $(bcfe)$, $(acfe)$, $(abfe)$, $(abcef)$

Figure 7.9 A transaction database, which contains duplicates, and core patterns for each distinct transaction.

its core patterns. For instance, *abcef* can be generated by merging just two of its core patterns, *ab* and *cef*, instead of having to merge all of its 26 core patterns.

Now, let's see how these observations can help us leap through pattern space more directly toward colossal patterns. Consider the following scheme. First, generate a complete set of frequent patterns up to a user-specified small size, and then randomly pick a pattern, β. β will have a high probability of being a core-descendant of some colossal pattern, α. Identify all of α's core-descendants in this complete set, and merge them. This generates a much larger core-descendant of α, giving us the ability to leap along a path toward α in the core-pattern tree, T_α. In the same fashion we select K patterns. The set of larger core-descendants generated is the candidate pool for the next iteration.

A question arises: Given β, a core-descendant of a colossal pattern α, how can we find the other core-descendants of α? Given two patterns, α and β, the pattern distance between them is defined as $Dist(\alpha, \beta) = 1 - \frac{|D_\alpha \cap D_\beta|}{|D_\alpha \cup D_\beta|}$. Pattern distance satisfies the triangle inequality.

For a pattern, α, let C_α be the set of all its core patterns. It can be shown that C_α is bounded in metric space by a "ball" of diameter $r(\tau)$, where $r(\tau) = 1 - \frac{1}{2/\tau - 1}$. This means that given a core pattern $\beta \in C_\alpha$, we can identify all of α's core patterns in the current pool by posing a range query. Note that in the mining algorithm, each randomly drawn pattern could be a core-descendant of more than one colossal pattern, and as such, when merging the patterns found by the "ball," more than one larger core-descendant could be generated.

From this discussion, the Pattern-Fusion method is outlined in the following two phases:

1. **Initial Pool:** Pattern-Fusion assumes an initial pool of small frequent patterns is available. This is the complete set of frequent patterns up to a small size (e.g., 3). This initial pool can be mined with any existing efficient mining algorithm.
2. **Iterative Pattern-Fusion:** Pattern-Fusion takes as input a user-specified parameter, K, which is the maximum number of patterns to be mined. The mining process is iterative. At each iteration, K seed patterns are randomly picked from the current pool. For each of these K seeds, we find all the patterns within a ball of a size specified by τ. All the patterns in each "ball" are then fused together to generate a set of superpatterns. These superpatterns form a new pool. If the pool contains more than K patterns, the next iteration begins with this pool for the new round of random drawing. As the support set of every superpattern shrinks with each new iteration, the iteration process terminates.

Note that *Pattern-Fusion merges small subpatterns of a large pattern instead of incrementally-expanding patterns with single items.* This gives the method an advantage to circumvent midsize patterns and progress on a path leading to a potential colossal pattern. The idea is illustrated in Figure 7.10. Each point shown in the metric space

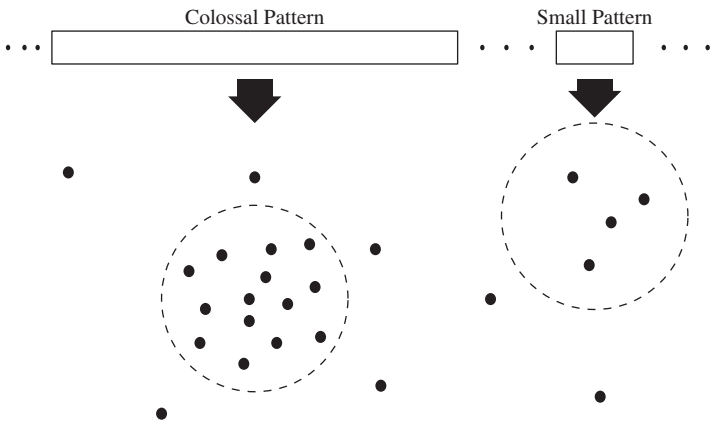

Figure 7.10 Pattern metric space: Each point represents a core pattern. The core patterns of a colossal pattern are denser than those of a small pattern, as shown within the dotted lines.

represents a core pattern. In comparison to a smaller pattern, a larger pattern has far more core patterns that are close to one another, all of which are bounded by a ball, as shown by the dotted lines. When drawing randomly from the initial pattern pool, we have a much higher probability of getting a core pattern of a large pattern, because the ball of a larger pattern is much denser.

It has been theoretically shown that Pattern-Fusion leads to a good approximation of colossal patterns. The method was tested on synthetic and real data sets constructed from program tracing data and microarray data. Experiments show that the method can find most of the colossal patterns with high efficiency.

7.5 Mining Compressed or Approximate Patterns

A major challenge in frequent pattern mining is the huge number of discovered patterns. Using a minimum support threshold to control the number of patterns found has limited effect. Too low a value can lead to the generation of an explosive number of output patterns, while too high a value can lead to the discovery of only commonsense patterns.

To reduce the huge set of frequent patterns generated in mining while maintaining high-quality patterns, we can instead mine a compressed or approximate set of frequent patterns. *Top-k most frequent closed patterns* were proposed to make the mining process concentrate on only the set of k most frequent patterns. Although interesting, they usually do not epitomize the k most representative patterns because of the uneven frequency distribution among itemsets. *Constraint-based mining* of frequent patterns (Section 7.3) incorporates user-specified constraints to filter out uninteresting patterns. Measures of

pattern/rule *interestingness* and *correlation* (Section 6.3) can also be used to help confine the search to patterns/rules of interest.

In this section, we look at two forms of "compression" of frequent patterns that build on the concepts of closed patterns and max-patterns. Recall from Section 6.2.6 that a *closed pattern* is a lossless compression of the set of frequent patterns, whereas a *max-pattern* is a lossy compression. In particular, Section 7.5.1 explores *clustering-based compression of frequent patterns*, which groups patterns together based on their similarity and frequency support. Section 7.5.2 takes a "*summarization*" *approach*, where the aim is to derive redundancy-aware top-k representative patterns that cover the whole set of (closed) frequent itemsets. The approach considers not only the representativeness of patterns but also their mutual independence to avoid redundancy in the set of generated patterns. The k representatives provide compact compression over the collection of frequent patterns, making them easier to interpret and use.

7.5.1 Mining Compressed Patterns by Pattern Clustering

Pattern compression can be achieved by pattern clustering. Clustering techniques are described in detail in Chapters 10 and 11. In this section, it is not necessary to know the fine details of clustering. Rather, you will learn how the concept of clustering can be applied to compress frequent patterns. Clustering is the automatic process of grouping like objects together, so that objects within a cluster are similar to one another and dissimilar to objects in other clusters. In this case, the objects are frequent patterns. The frequent patterns are clustered using a tightness measure called δ-cluster. A representative pattern is selected for each cluster, thereby offering a compressed version of the set of frequent patterns.

Before we begin, let's review some definitions. An itemset X is a **closed frequent itemset** in a data set D if X is frequent and there exists no proper super-itemset Y of X such that Y has the same support count as X in D. An itemset X is a **maximal frequent itemset** in data set D if X is frequent and there exists no super-itemset Y such that $X \subset Y$ and Y is frequent in D. Using these concepts alone is not enough to obtain a good representative compression of a data set, as we see in Example 7.12.

Example 7.12 Shortcomings of closed itemsets and maximal itemsets for compression. Table 7.3 shows a subset of frequent itemsets on a large data set, where *a, b, c, d, e, f* represent individual items. There are no closed itemsets here; therefore, we cannot use closed frequent itemsets to compress the data. The only maximal frequent itemset is P_3. However, we observe that itemsets P_2, P_3, and P_4 are significantly different with respect to their support counts. If we were to use P_3 to represent a compressed version of the data, we would lose this support count information entirely. From visual inspection, consider the two pairs (P_1, P_2) and (P_4, P_5). The patterns within each pair are very similar with respect to their support and expression. Therefore, intuitively, P_2, P_3, and P_4, collectively, should serve as a better compressed version of the data. ■

Table 7.3 Subset of Frequent Itemsets

ID	*Itemsets*	*Support*
P_1	$\{b, c, d, e\}$	205,227
P_2	$\{b, c, d, e, f\}$	205,211
P_3	$\{a, b, c, d, e, f\}$	101,758
P_4	$\{a, c, d, e, f\}$	161,563
P_5	$\{a, c, d, e\}$	161,576

So, let's see if we can find a way of clustering frequent patterns as a means of obtaining a compressed representation of them. We will need to define a good similarity measure, cluster patterns according to this measure, and then select and output only a *representative pattern* for each cluster. Since the set of closed frequent patterns is a lossless compression over the original frequent patterns set, it is a good idea to discover representative patterns over the collection of *closed* patterns.

We can use the following distance measure between closed patterns. Let P_1 and P_2 be two closed patterns. Their supporting transaction sets are $T(P_1)$ and $T(P_2)$, respectively. The **pattern distance** of P_1 and P_2, $Pat\_Dist(P_1, P_2)$, is defined as

$$Pat\_Dist(P_1, P_2) = 1 - \frac{|T(P_1) \cap T(P_2)|}{|T(P_1) \cup T(P_2)|}. \tag{7.14}$$

Pattern distance is a valid distance metric defined on the set of transactions. Note that it incorporates the *support* information of patterns, as desired previously.

Example 7.13 **Pattern distance.** Suppose P_1 and P_2 are two patterns such that $T(P_1) = \{t_1, t_2, t_3, t_4, t_5\}$ and $T(P_2) = \{t_1, t_2, t_3, t_4, t_6\}$, where t_i is a transaction in the database. The distance between P_1 and P_2 is $Pat\_Dist(P_1, P_2) = 1 - \frac{4}{6} = \frac{1}{3}$. ■

Now, let's consider the *expression* of patterns. Given two patterns A and B, we say B can be **expressed** by A if $O(B) \subset O(A)$, where $O(A)$ is the corresponding itemset of pattern A. Following this definition, assume patterns $P_1, P_2, \ldots, P_k$ are in the same cluster. The representative pattern P_r of the cluster should be able to *express* all the other patterns in the cluster. Clearly, we have $\cup_{i=1}^{k} O(P_i) \subseteq O(P_r)$.

Using the distance measure, we can simply apply a clustering method, such as k-means (Section 10.2), on the collection of frequent patterns. However, this introduces two problems. First, the quality of the clusters cannot be guaranteed; second, it may not be able to find a representative pattern for each cluster (i.e., the pattern P_r may not belong to the same cluster). To overcome these problems, this is where the concept of δ-cluster comes in, where δ ($0 \leq \delta \leq 1$) measures the tightness of a cluster.

A pattern P is **δ-covered** by another pattern P' if $O(P) \subseteq O(P')$ and $Pat\_Dist(P, P') \leq \delta$. A set of patterns form a **δ-cluster** if there exists a representative pattern P_r such that for each pattern P in the set, P is δ-covered by P_r.

Note that according to the concept of δ-cluster, a pattern can belong to multiple clusters. Also, using δ-cluster, we only need to compute the distance between each pattern and the representative pattern of the cluster. Because a pattern P is δ-covered by a representative pattern P_r only if $O(P) \subseteq O(P_r)$, we can simplify the distance calculation by considering only the supports of the patterns:

$$Pat\_Dist(P, P_r) = 1 - \frac{|T(P) \cap T(P_r)|}{|T(P) \cup T(P_r)|} = 1 - \frac{|T(P_r)|}{|T(P)|}. \tag{7.15}$$

If we restrict the representative pattern to be frequent, then the number of representative patterns (i.e., clusters) is no less than the number of maximal frequent patterns. This is because a maximal frequent pattern can only be covered by itself. To achieve more succinct compression, we relax the constraints on representative patterns, that is, we allow the support of representative patterns to be *somewhat* less than *min_sup*.

For any representative pattern P_r, assume its support is k. Since it has to *cover* at least one frequent pattern (i.e., P) with support that is at least *min_sup*, we have

$$\delta \geq Pat\_Dist(P, P_r) = 1 - \frac{|T(P_r)|}{|T(P)|} \geq 1 - \frac{k}{min\_sup}. \tag{7.16}$$

That is, $k \geq (1 - \delta) \times min\_sup$. This is the minimum support for a representative pattern, denoted as min_sup_r.

Based on the preceding discussion, the pattern compression problem can be defined as follows: *Given a transaction database, a minimum support* min_sup, *and the cluster quality measure δ, the pattern compression problem is to find a set of representative patterns R such that for each frequent pattern P (with respect to* min_sup), *there is a representative pattern $P_r \in R$ (with respect to* min_sup_r), *which covers P, and the value of $|R|$ is minimized.*

Finding a minimum set of representative patterns is an NP-Hard problem. However, efficient methods have been developed that reduce the number of closed frequent patterns generated by orders of magnitude with respect to the original collection of closed patterns. The methods succeed in finding a high-quality compression of the pattern set.

7.5.2 Extracting Redundancy-Aware Top-*k* Patterns

Mining the top-k most frequent patterns is a strategy for reducing the number of patterns returned during mining. However, in many cases, frequent patterns are not mutually independent but often clustered in small regions. This is somewhat like finding 20 population centers in the world, which may result in cities clustered in a small number of countries rather than evenly distributed across the globe. Instead, most users would prefer to derive the k most interesting patterns, which are not only significant, but also mutually independent and containing little redundancy. A small set of

k representative patterns that have not only high significance but also low redundancy are called **redundancy-aware top-k patterns**.

Example 7.14 **Redundancy-aware top-k strategy versus other top-k strategies.** Figure 7.11 illustrates the intuition behind *redundancy-aware top-k patterns* versus *traditional top-k patterns* and *k-summarized patterns*. Suppose we have the frequent patterns set shown in Figure 7.11(a), where each circle represents a pattern of which the significance is colored in grayscale. The distance between two circles reflects the redundancy of the two corresponding patterns: The closer the circles are, the more redundant the respective patterns are to one another. Let's say we want to find three patterns that will best represent the given set, that is, $k = 3$. Which three should we choose?

Arrows are used to show the patterns chosen if using redundancy-aware top-k patterns (Figure 7.11b), traditional top-k patterns (Figure 7.11c), or k-summarized patterns (Figure 7.11d). In Figure 7.11(c), the **traditional top-k strategy** relies solely on significance: It selects the three most significant patterns to represent the set.

In Figure 7.11(d), the ***k*-summarized pattern strategy** selects patterns based solely on nonredundancy. It detects three clusters, and finds the most representative patterns to

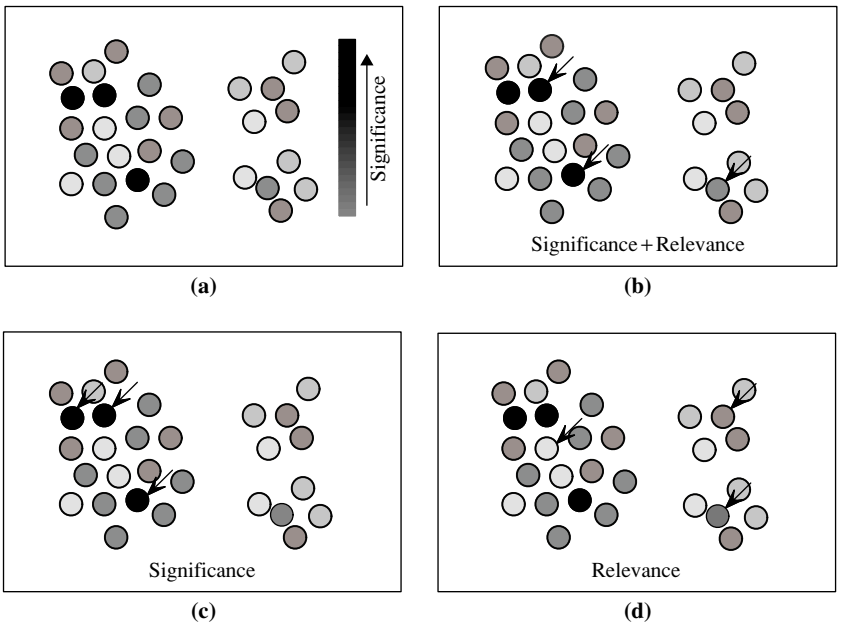

Figure 7.11 Conceptual view comparing top-k methodologies (where gray levels represent pattern significance, and the closer that two patterns are displayed, the more redundant they are to one another): (a) original patterns, (b) redundancy-aware top-k patterns, (c) traditional top-k patterns, and (d) k-summarized patterns.

be the "centermost'" pattern from each cluster. These patterns are chosen to represent the data. The selected patterns are considered "summarized patterns" in the sense that they represent or "provide a summary" of the clusters they stand for.

By contrast, in Figure 7.11(b) the **redundancy-aware top-*k* patterns** make a trade-off between significance and redundancy. The three patterns chosen here have high significance and low redundancy. Observe, for example, the two highly significant patterns that, based on their redundancy, are displayed next to each other. The redundancy-aware top-k strategy selects only one of them, taking into consideration that two would be redundant. To formalize the definition of redundancy-aware top-k patterns, we'll need to define the concepts of significance and redundancy. ■

A **significance measure** S is a function mapping a pattern $p \in \mathcal{P}$ to a real value such that $S(p)$ is the degree of interestingness (or usefulness) of the pattern p. In general, significance measures can be either objective or subjective. *Objective measures* depend only on the structure of the given pattern and the underlying data used in the discovery process. Commonly used objective measures include support, confidence, correlation, and *tf-idf* (or *term frequency* versus *inverse document frequency*), where the latter is often used in information retrieval. *Subjective* measures are based on user beliefs in the data. They therefore depend on the users who examine the patterns. A subjective measure is usually a relative score based on user prior knowledge or a background model. It often measures the unexpectedness of a pattern by computing its divergence from the background model. Let $S(p, q)$ be the **combined significance** of patterns p and q, and $S(p|q) = S(p, q) - S(q)$ be the **relative significance** of p given q. Note that the combined significance, $S(p, q)$, means the collective significance of two individual patterns p and q, not the significance of a single super pattern $p \cup q$.

Given the significance measure S, the **redundancy** R **between two patterns** p and q is defined as $R(p, q) = S(p) + S(q) - S(p, q)$. Subsequently, we have $S(p|q) = S(p) - R(p, q)$.

We assume that the combined significance of two patterns is no less than the significance of any individual pattern (since it is a collective significance of two patterns) and does not exceed the sum of two individual significance patterns (since there exists redundancy). That is, the redundancy between two patterns should satisfy

$$0 \leq R(p, q) \leq \min(S(p), S(q)). \tag{7.17}$$

The ideal redundancy measure $R(p, q)$ is usually hard to obtain. However, we can approximate redundancy using distance between patterns such as with the distance measure defined in Section 7.5.1.

The problem of finding redundancy-aware top-k patterns can thus be transformed into finding a k-pattern set that maximizes the marginal significance, which is a well-studied problem in information retrieval. In this field, a document has high marginal relevance if it is both relevant to the query and contains minimal marginal similarity to previously selected documents, where the marginal similarity is computed by choosing the most relevant selected document. Experimental studies have shown this method to be efficient and able to find high-significance and low-redundancy top-k patterns.

7.6 Pattern Exploration and Application

For discovered frequent patterns, is there any way the mining process can return additional information that will help us to better understand the patterns? What kinds of applications exist for frequent pattern mining? These topics are discussed in this section. Section 7.6.1 looks at the automated generation of **semantic annotations** for frequent patterns. These are dictionary-like annotations. They provide semantic information relating to patterns, based on the context and usage of the patterns, which aids in their understanding. Semantically similar patterns also form part of the annotation, providing a more direct connection between discovered patterns and any other patterns already known to the users.

Section 7.6.2 presents an overview of applications of frequent pattern mining. While the applications discussed in Chapter 6 and this chapter mainly involve market basket analysis and correlation analysis, there are many other areas in which frequent pattern mining is useful. These range from data preprocessing and classification to clustering and the analysis of complex data.

7.6.1 Semantic Annotation of Frequent Patterns

Pattern mining typically generates a huge set of frequent patterns without providing enough information to interpret the meaning of the patterns. In the previous section, we introduced pattern processing techniques to shrink the size of the output set of frequent patterns such as by extracting redundancy-aware top-k patterns or compressing the pattern set. These, however, do not provide any semantic interpretation of the patterns. It would be helpful if we could also generate semantic annotations for the frequent patterns found, which would help us to better understand the patterns.

"*What is an appropriate semantic annotation for a frequent pattern?*" Think about what we find when we look up the meaning of terms in a dictionary. Suppose we are looking up the term *pattern*. A dictionary typically contains the following components to explain the term:

1. *A set of definitions*, such as "a decorative design, as for wallpaper, china, or textile fabrics, etc.; a natural or chance configuration"
2. *Example sentences*, such as "*patterns* of frost on the window; the behavior *patterns* of teenagers, . . . "
3. *Synonyms from a thesaurus*, such as "model, archetype, design, exemplar, motif, ..."

Analogically, what if we could extract similar types of semantic information and provide such structured annotations for frequent patterns? This would greatly help users in interpreting the meaning of patterns and in deciding on how or whether to further explore them. Unfortunately, it is infeasible to provide such precise semantic definitions for patterns without expertise in the domain. Nevertheless, we can explore how to *approximate* such a process for frequent pattern mining.

Pattern: "{*frequent, pattern*}"
context indicators:
"mining," "constraint," "Apriori," "FP-growth,"
"rakesh agrawal," "jiawei han," ...
representative transactions:
1) mining *frequent patterns* without candidate ...
2) ... mining closed *frequent* graph *patterns*
semantically similar patterns:
"{*frequent, sequential, pattern*}," "{*graph, pattern*}"
"{*maximal, pattern*}," "{*frequent, closed, pattern*}," ...

Figure 7.12 Semantic annotation of the pattern "{*frequent, pattern*}."

In general, the hidden meaning of a pattern can be inferred from patterns with similar meanings, data objects co-occurring with it, and transactions in which the pattern appears. Annotations with such information are analogous to dictionary entries, which can be regarded as annotating each term with structured semantic information. Let's examine an example.

Example 7.15 Semantic annotation of a frequent pattern. Figure 7.12 shows an example of a semantic annotation for the pattern "{*frequent, pattern*}." This dictionary-like annotation provides semantic information related to "{*frequent, pattern*}," consisting of its strongest *context indicators*, the most *representative data transactions*, and the most *semantically similar patterns*. This kind of semantic annotation is similar to natural language processing. The semantics of a word can be inferred from its context, and words sharing similar contexts tend to be semantically similar. The context indicators and the representative transactions provide a view of the context of the pattern from different angles to help users understand the pattern. The semantically similar patterns provide a more direct connection between the pattern and any other patterns already known to the users. ■

"*How can we perform automated semantic annotation for a frequent pattern?*" The key to high-quality semantic annotation of a frequent pattern is the successful context modeling of the pattern. For context modeling of a pattern, p, consider the following.

- A **context unit** is a basic object in a database, D, that carries semantic information and co-occurs with at least one frequent pattern, p, in at least one transaction in D. A context unit can be an item, a pattern, or even a transaction, depending on the specific task and data.
- The **context of a pattern**, p, is a selected set of weighted context units (referred to as **context indicators**) in the database. It carries semantic information, and co-occurs with a frequent pattern, p. The context of p can be modeled using a vector space model, that is, the context of p can be represented as $C(p) = \langle w(u_1),$

$w(u_2), \ldots, w(u_n)\rangle$, where $w(u_i)$ is a weight function of term u_i. A transaction t is represented as a vector $\langle v_1, v_2, \ldots, v_m\rangle$, where $v_i = 1$ if and only if $v_i \in t$, otherwise $v_i = 0$.

Based on these concepts, we can define the basic task of **semantic pattern annotation** as follows:

1. Select context units and design a strength weight for each unit to model the contexts of frequent patterns.
2. Design similarity measures for the contexts of two patterns, and for a transaction and a pattern context.
3. For a given frequent pattern, extract the most significant context indicators, representative transactions, and semantically similar patterns to construct a structured annotation.

"*Which context units should we select as context indicators?*" Although a context unit can be an item, a transaction, or a pattern, typically, frequent patterns provide the most semantic information of the three. There are usually a large number of frequent patterns associated with a pattern, p. Therefore, we need a systematic way to select only the important and nonredundant frequent patterns from a large pattern set.

Considering that the closed patterns set is a lossless compression of frequent pattern sets, we can first derive the closed patterns set by applying efficient closed pattern mining methods. However, as discussed in Section 7.5, a closed pattern set is not compact enough, and pattern compression needs to be performed. We could use the pattern compression methods introduced in Section 7.5.1 or explore alternative compression methods such as microclustering using the Jaccard coefficient (Chapter 2) and then selecting the most representative patterns from each cluster.

"*How, then, can we assign weights for each context indicator?*" A good weighting function should obey the following properties: (1) the best semantic indicator of a pattern, p, is itself, (2) assign the same score to two patterns if they are equally strong, and (3) if two patterns are independent, neither can indicate the meaning of the other. The meaning of a pattern, p, can be inferred from either the appearance or absence of indicators.

Mutual information is one of several possible weighting functions. It is widely used in information theory to measure the mutual independency of two random variables. Intuitively, it measures how much information a random variable tells about the other. Given two frequent patterns, p_α and p_β, let $X = \{0,1\}$ and $Y = \{0,1\}$ be two random variables representing the appearance of p_α and p_β, respectively. **Mutual information** $I(X;Y)$ is computed as

$$I(X;Y) = \sum_{x \in X} \sum_{y \in Y} P(x,y) log \frac{P(x,y)}{P(x)P(y)}, \tag{7.18}$$

where $P(x=1,y=1)=\frac{|D_\alpha \cap D_\beta|}{|D|}$, $P(x=0,y=1)=\frac{|D_\beta|-|D_\alpha \cap D_\beta|}{|D|}$, $P(x=1,y=0)=\frac{|D_\alpha|-|D_\alpha \cap D_\beta|}{|D|}$, and $P(x=0,y=0)=\frac{|D|-|D_\alpha \cup D_\beta|}{|D|}$. Standard Laplace smoothing can be used to avoid zero probability.

Mutual information favors strongly correlated units and thus can be used to model the indicative strength of the context units selected. With context modeling, pattern annotation can be accomplished as follows:

1. To extract the most significant context indicators, we can use cosine similarity (Chapter 2) to measure the semantic similarity between pairs of context vectors, rank the context indicators by the weight strength, and extract the strongest ones.
2. To extract representative transactions, represent each transaction as a context vector. Rank the transactions with semantic similarity to the pattern p.
3. To extract semantically similar patterns, rank each frequent pattern, p, by the semantic similarity between their context models and the context of p.

Based on these principles, experiments have been conducted on large data sets to generate semantic annotations. Example 7.16 illustrates one such experiment.

Example 7.16 Semantic annotations generated for frequent patterns from the DBLP Computer Science Bibliography. Table 7.4 shows annotations generated for frequent patterns from a portion of the DBLP data set.[3] The DBLP data set contains papers from the proceedings of 12 major conferences in the fields of database systems, information retrieval, and data mining. Each transaction consists of two parts: the authors and the title of the corresponding paper.

Consider two types of patterns: (1) *frequent author* or *coauthorship*, each of which is a frequent itemset of authors, and (2) *frequent title terms*, each of which is a frequent sequential pattern of the title words. The method can automatically generate dictionary-like annotations for different kinds of frequent patterns. For frequent itemsets like coauthorship or single authors, the strongest context indicators are usually the other coauthors and discriminative title terms that appear in their work. The semantically similar patterns extracted also reflect the authors and terms related to their work. However, these similar patterns may not even co-occur with the given pattern in a paper. For example, the patterns "*timos_k_selli*," "*ramakrishnan_srikant*," and so on, do not co-occur with the pattern "*christos_faloutsos*," but are extracted because their contexts are similar since they all are database and/or data mining researchers; thus the annotation is meaningful.

For the title term "*information retrieval*," which is a sequential pattern, its strongest context indicators are usually the authors who tend to use the term in the titles of their papers, or the terms that tend to coappear with it. Its semantically similar patterns usually provide interesting concepts or descriptive terms, which are close in meaning (e.g., "*information retrieval* → *information filter*)."

[3] *www.informatik.uni-trier.de/~ley/db/*.

Table 7.4 Annotations Generated for Frequent Patterns in the DBLP Data Set

Pattern	*Type*	*Annotations*
christos_faloutsos	Context indicator	spiros_papadimitriou; fast; use fractal; graph; use correlate
	Representative transactions	multi-attribute hash use gray code
	Representative transactions	recovery latent time-series observe sum network tomography particle filter
	Representative transactions	index multimedia database tutorial
	Semantic similar patterns	spiros_papadimitriou&christos_faloutsos; spiros_papadimitriou; flip_korn; timos_k_selli;
		ramakrishnan_srikant; ramakrishnan_srikant&rakesh_agrawal
information retrieval	Context indicator	w_bruce_croft; web information; monika_ rauch_henzinger;
		james_p_callan; full-text
	Representative transactions	web information retrieval
	Representative transactions	language model information retrieval
	Semantic similar patterns	information use; web information; probabilistic information; information filter;
		text information

In both scenarios, the representative transactions extracted give us the titles of papers that effectively capture the meaning of the given patterns. The experiment demonstrates the effectiveness of semantic pattern annotation to generate a dictionary-like annotation for frequent patterns, which can help a user understand the meaning of annotated patterns. ■

The context modeling and semantic analysis method presented here is general and can deal with any type of frequent patterns with context information. Such semantic annotations can have many other applications such as ranking patterns, categorizing and clustering patterns with semantics, and summarizing databases. Applications of the pattern context model and semantical analysis method are also not limited to pattern annotation; other example applications include pattern compression, transaction clustering, pattern relations discovery, and pattern synonym discovery.

7.6.2 Applications of Pattern Mining

We have studied many aspects of frequent pattern mining, with topics ranging from efficient mining algorithms and the diversity of patterns to pattern interestingness, pattern

compression/approximation, and semantic pattern annotation. Let's take a moment to consider why this field has generated so much attention. What are some of the application areas in which frequent pattern mining is useful? This section presents an overview of applications for frequent pattern mining. We have touched on several application areas already, such as market basket analysis and correlation analysis, yet frequent pattern mining can be applied to many other areas as well. These range from data preprocessing and classification to clustering and the analysis of complex data.

To summarize, frequent pattern mining is a data mining task that discovers patterns that occur frequently together and/or have some distinctive properties that distinguish them from others, often disclosing something inherent and valuable. The patterns may be itemsets, subsequences, substructures, or values. The task also includes the discovery of rare patterns, revealing items that occur very rarely together yet are of interest. Uncovering frequent patterns and rare patterns leads to many broad and interesting applications, described as follows.

Pattern mining is widely used for **noise filtering and data cleaning** as **preprocessing** in many data-intensive applications. We can use it to analyze microarray data, for instance, which typically consists of tens of thousands of dimensions (e.g., representing genes). Such data can be rather noisy. Frequent pattern data mining can help us distinguish between what is noise and what isn't. We may assume that items that occur frequently together are less likely to be random noise and should not be filtered out. On the other hand, those that occur very frequently (similar to stopwords in text documents) are likely indistinctive and may be filtered out. Frequent pattern mining can help in background information identification and noise reduction.

Pattern mining often helps in the **discovery of inherent structures and clusters hidden in the data**. Given the DBLP data set, for instance, frequent pattern mining can easily find interesting clusters like coauthor clusters (by examining authors who frequently collaborate) and conference clusters (by examining the sharing of many common authors and terms). Such structure or cluster discovery can be used as preprocessing for more sophisticated data mining.

Although there are numerous classification methods (Chapters 8 and 9), research has found that frequent patterns can be used as building blocks in the construction of high-quality classification models, hence called **pattern-based classification**. The approach is successful because (1) the appearance of very infrequent item(s) or itemset(s) can be caused by random noise and may not be reliable for model construction, yet a relatively frequent pattern often carries more information gain for constructing more reliable models; (2) patterns in general (i.e., itemsets consisting of multiple attributes) usually carry more information gain than a single attribute (feature); and (3) the patterns so generated are often intuitively understandable and easy to explain. Recent research has reported several methods that mine interesting, frequent, and discriminative patterns and use them for effective classification. Pattern-based classification methods are introduced in Chapter 9.

Frequent patterns can also be used effectively for **subspace clustering in high-dimensional space**. Clustering is challenging in high-dimensional space, where the distance between two objects is often difficult to measure. This is because such a distance is dominated by the different sets of dimensions in which the objects are residing.

Thus, instead of clustering objects in their full high-dimensional spaces, it can be more meaningful to find clusters in certain subspaces. Recently, researchers have developed subspace-based pattern growth methods that cluster objects based on their common frequent patterns. They have shown that such methods are effective for clustering microarray-based gene expression data. Subspace clustering methods are discussed in Chapter 11.

Pattern analysis is useful in the **analysis of spatiotemporal data, time-series data, image data, video data, and multimedia data**. An area of *spatiotemporal data analysis* is the discovery of **colocation patterns**. These, for example, can help determine if a certain disease is geographically colocated with certain objects like a well, a hospital, or a river. In *time-series data analysis*, researchers have discretized time-series values into multiple intervals (or levels) so that tiny fluctuations and value differences can be ignored. The data can then be summarized into sequential patterns, which can be indexed to facilitate similarity search or comparative analysis. In *image analysis and pattern recognition*, researchers have also identified frequently occurring visual fragments as "visual words," which can be used for effective clustering, classification, and comparative analysis.

Pattern mining has also been used for the **analysis of sequence or structural data** such as trees, graphs, subsequences, and networks. In software engineering, researchers have identified consecutive or gapped subsequences in program execution as sequential patterns that help identify software bugs. Copy-and-paste bugs in large software programs can be identified by extended sequential pattern analysis of source programs. Plagiarized software programs can be identified based on their essentially identical program flow/loop structures. Authors' commonly used sentence substructures can be identified and used to distinguish articles written by different authors.

Frequent and discriminative patterns can be used as primitive **indexing structures** (known as graph indices) to help search large, complex, structured data sets and networks. These support a similarity search in graph-structured data such as chemical compound databases or XML-structured databases. Such patterns can also be used for data compression and summarization.

Furthermore, frequent patterns have been used in **recommender systems**, where people can find correlations, clusters of customer behaviors, and classification models based on commonly occurring or discriminative patterns (Chapter 13).

Finally, studies on efficient computation methods in pattern mining mutually enhance many other studies on **scalable computation**. For example, the computation and materialization of **iceberg cubes** using the BUC and Star-Cubing algorithms (Chapter 5) respectively share many similarities to computing frequent patterns by the Apriori and FP-growth algorithms (Chapter 6).

7.7 Summary

- The **scope** of frequent pattern mining research reaches far beyond the basic concepts and methods introduced in Chapter 6 for mining frequent itemsets and associations. This chapter presented a road map of the field, where topics are organized

with respect to the kinds of patterns and rules that can be mined, mining methods, and applications.

- In addition to mining for basic frequent itemsets and associations, **advanced forms of patterns** can be mined such as multilevel associations and multidimensional associations, quantitative association rules, rare patterns, and negative patterns. We can also mine high-dimensional patterns and compressed or approximate patterns.
- **Multilevel associations** involve data at more than one abstraction level (e.g., "*buys computer*" and "*buys laptop*"). These may be mined using multiple minimum support thresholds. **Multidimensional associations** contain more than one dimension. Techniques for mining such associations differ in how they handle repetitive predicates. **Quantitative association rules** involve quantitative attributes. Discretization, clustering, and statistical analysis that discloses exceptional behavior can be integrated with the pattern mining process.
- **Rare patterns** occur rarely but are of special interest. **Negative patterns** are patterns with components that exhibit negatively correlated behavior. Care should be taken in the definition of negative patterns, with consideration of the null-invariance property. Rare and negative patterns may highlight exceptional behavior in the data, which is likely of interest.
- **Constraint-based mining** strategies can be used to help direct the mining process toward patterns that match users' intuition or satisfy certain constraints. Many user-specified constraints can be pushed deep into the mining process. Constraints can be categorized into **pattern-pruning** and **data-pruning** constraints. Properties of such constraints include *monotonicity, antimonotonicity*, *data-antimonotonicity*, and *succinctness.* Constraints with such properties can be properly incorporated into efficient pattern mining processes.
- Methods have been developed for mining patterns in **high-dimensional space**. This includes a pattern growth approach based on *row enumeration* for mining data sets where the number of dimensions is large and the number of data tuples is small (e.g., for microarray data), as well as mining **colossal patterns** (i.e., patterns of very long length) by a *Pattern-Fusion* method.
- To reduce the number of patterns returned in mining, we can instead mine compressed patterns or approximate patterns. *Compressed patterns* can be mined with representative patterns defined based on the concept of clustering, and *approximate patterns* can be mined by extracting **redundancy-aware top-*k* patterns** (i.e., a small set of *k*-representative patterns that have not only high significance but also low redundancy with respect to one another).
- **Semantic annotations** can be generated to help users understand the meaning of the frequent patterns found, such as for textual terms like "{*frequent, pattern*}." These are dictionary-like annotations, providing semantic information relating to the term. This information consists of *context indicators* (e.g., terms indicating the context of that pattern), the most *representative data transactions* (e.g., fragments or sentences

containing the term), and the most *semantically similar patterns* (e.g., "{*maximal*, *pattern*}" is semantically similar to "{*frequent*, *pattern*}"). The annotations provide a view of the pattern's context from different angles, which aids in their understanding.

- Frequent pattern mining has many diverse applications, ranging from pattern-based data cleaning to pattern-based classification, clustering, and outlier or exception analysis. These methods are discussed in the subsequent chapters in this book.

7.8 Exercises

7.1 Propose and outline a **level-shared mining** approach to mining multilevel association rules in which each item is encoded by its level position. Design it so that an initial scan of the database collects the count for each item *at each concept level*, identifying frequent and subfrequent items. Comment on the processing cost of mining multilevel associations with this method in comparison to mining single-level associations.

7.2 Suppose, as manager of a chain of stores, you would like to use sales transactional data to analyze the effectiveness of your store's advertisements. In particular, you would like to study how specific factors influence the effectiveness of advertisements that announce a particular category of items on sale. The factors to study are the *region* in which customers live and the *day-of-the-week* and *time-of-the-day* of the ads. Discuss how to design an efficient method to mine the transaction data sets and explain how **multidimensional** and **multilevel mining** methods can help you derive a good solution.

7.3 **Quantitative association rules** may disclose exceptional behaviors within a data set, where "exceptional" can be defined based on statistical theory. For example, Section 7.2.3 shows the association rule

$$sex = female \Rightarrow mean\_wage = \$7.90/hr \ (overall\_mean\_wage = \$9.02/hr),$$

which suggests an exceptional pattern. The rule states that the average wage for females is only $7.90 per hour, which is a significantly lower wage than the overall average of $9.02 per hour. Discuss how such quantitative rules can be discovered systematically and efficiently in large data sets with quantitative attributes.

7.4 In multidimensional data analysis, it is interesting to extract pairs of *similar* cell characteristics associated with substantial changes in measure in a data cube, where cells are considered *similar* if they are related by roll-up (i.e, *ancestors*), drill-down (i.e, *descendants*), or 1-D mutation (i.e, *siblings*) operations. Such an analysis is called **cube gradient analysis**.

Suppose the measure of the cube is *average*. A user poses a set of *probe cells* and would like to find their corresponding sets of *gradient cells*, each of which satisfies a certain gradient threshold. For example, find the set of corresponding gradient cells that have an average sale price greater than 20% of that of the given probe cells. Develop an algorithm than mines the set of constrained gradient cells efficiently in a large data cube.

7.5 Section 7.2.4 presented various ways of defining negatively correlated patterns. Consider Definition 7.3: "Suppose that itemsets X and Y are both frequent, that is, $sup(X) \geq min\_sup$ and $sup(Y) \geq min\_sup$, where min_sup is the minimum support threshold. If $(P(X|Y) + P(Y|X))/2 < \epsilon$, where ϵ is a negative pattern threshold, then pattern $X \cup Y$ is a **negatively correlated pattern**." Design an efficient pattern growth algorithm for mining the set of negatively correlated patterns.

7.6 Prove that each entry in the following table correctly characterizes its corresponding **rule constraint** for frequent itemset mining.

	Rule Constraint	*Antimonotonic*	*Monotonic*	*Succinct*
(a)	$v \in S$	no	yes	yes
(b)	$S \subseteq V$	yes	no	yes
(c)	$min(S) \leq v$	no	yes	yes
(d)	$range(S) \leq v$	yes	no	no
(e)	$variance(S) \leq v$	convertible	convertible	no

7.7 The price of each item in a store is non-negative. The store manager is only interested in rules of certain forms, using the constraints given in (a)–(b). For each of the following cases, identify the kinds of **constraints** they represent and briefly discuss how to mine such association rules using **constraint-based pattern mining**.

(a) Containing at least one Blu-ray DVD movie.

(b) Containing items with a sum of the prices that is less than $150.

(c) Containing one free item and other items with a sum of the prices that is at least $200.

(d) Where the average price of all the items is between $100 and $500.

7.8 Section 7.4.1 introduced a core Pattern-Fusion method for **mining high-dimensional data**. Explain why a long pattern, if one exists in the data set, is likely to be discovered by this method.

7.9 Section 7.5.1 defined a **pattern distance measure** between closed patterns P_1 and P_2 as

$$Pat\_Dist(P_1, P_2) = 1 - \frac{|T(P_1) \cap T(P_2)|}{|T(P_1) \cup T(P_2)|},$$

where $T(P_1)$ and $T(P_2)$ are the supporting transaction sets of P_1 and P_2, respectively. Is this a valid distance metric? Show the derivation to support your answer.

7.10 Association rule mining often generates a large number of rules, many of which may be similar, thus not containing much novel information. Design an efficient algorithm that **compresses** a large set of patterns into a small compact set. Discuss whether your mining method is robust under different pattern similarity definitions.

7.11 Frequent pattern mining may generate many superfluous patterns. Therefore, it is important to develop methods that mine compressed patterns. Suppose a user would like to obtain only k patterns (where k is a small integer). Outline an efficient method that generates the ***k* most representative patterns**, where more distinct patterns are preferred over very similar patterns. Illustrate the effectiveness of your method using a small data set.

7.12 It is interesting to generate **semantic annotations** for mined patterns. Section 7.6.1 presented a pattern annotation method. Alternative methods are possible, such as by utilizing type information. In the DBLP data set, for example, authors, conferences, terms, and papers form multi-typed data. Develop a method for automated semantic pattern annotation that makes good use of typed information.

7.9 Bibliographic Notes

This chapter described various ways in which the basic techniques of frequent itemset mining (presented in Chapter 6) have been extended. One line of extension is mining multilevel and multidimensional association rules. Multilevel association mining was studied in Srikant and Agrawal [SA95] and Han and Fu [HF95]. In Srikant and Agrawal [SA95], such mining was studied in the context of *generalized association rules*, and an R-interest measure was proposed for removing redundant rules. Mining multidimensional association rules using static discretization of quantitative attributes and data cubes was studied by Kamber, Han, and Chiang [KHC97].

Another line of extension is to mine patterns on numeric attributes. Srikant and Agrawal [SA96] proposed a nongrid-based technique for mining quantitative association rules, which uses a measure of partial completeness. Mining quantitative association rules based on rule clustering was proposed by Lent, Swami, and Widom [LSW97]. Techniques for mining quantitative rules based on x-monotone and rectilinear regions were presented by Fukuda, Morimoto, Morishita, and Tokuyama [FMMT96] and Yoda, Fukuda, Morimoto, et al. [YFM+97]. Mining (distance-based) association rules over interval data was proposed by Miller and Yang [MY97]. Aumann and Lindell [AL99] studied the mining of quantitative association rules based on a statistical theory to present only those rules that deviate substantially from normal data.

Mining rare patterns by pushing group-based constraints was proposed by Wang, He, and Han [WHH00]. Mining negative association rules was discussed by Savasere, Omiecinski, and Navathe [SON98] and by Tan, Steinbach, and Kumar [TSK05].

Constraint-based mining directs the mining process toward patterns that are likely of interest to the user. The use of metarules as syntactic or semantic filters defining the form of interesting single-dimensional association rules was proposed in Klemettinen, Mannila, Ronkainen, et al. [KMR+94]. Metarule-guided mining, where the metarule consequent specifies an action (e.g., Bayesian clustering or plotting) to be applied to the data satisfying the metarule antecedent, was proposed in Shen, Ong, Mitbander,

and Zaniolo [SOMZ96]. A relation-based approach to metarule-guided mining of association rules was studied in Fu and Han [FH95].

Methods for constraint-based mining using pattern pruning constraints were studied by Ng, Lakshmanan, Han, and Pang [NLHP98]; Lakshmanan, Ng, Han, and Pang [LNHP99]; and Pei, Han, and Lakshmanan [PHL01]. Constraint-based pattern mining by data reduction using data pruning constraints was studied by Bonchi, Giannotti, Mazzanti, and Pedreschi [BGMP03] and Zhu, Yan, Han, and Yu [ZYHY07]. An efficient method for mining constrained correlated sets was given in Grahne, Lakshmanan, and Wang [GLW00]. A dual mining approach was proposed by Bucila, Gehrke, Kifer, and White [BGKW03]. Other ideas involving the use of templates or predicate constraints in mining have been discussed in Anand and Kahn [AK93]; Dhar and Tuzhilin [DT93]; Hoschka and Klösgen [HK91]; Liu, Hsu, and Chen [LHC97]; Silberschatz and Tuzhilin [ST96]; and Srikant, Vu, and Agrawal [SVA97].

Traditional pattern mining methods encounter challenges when mining high-dimensional patterns, with applications like bioinformatics. Pan, Cong, Tung, et al. [PCT+03] proposed CARPENTER, a method for finding closed patterns in high-dimensional biological data sets, which integrates the advantages of vertical data formats and pattern growth methods. Pan, Tung, Cong, and Xu [PTCX04] proposed COBBLER, which finds frequent closed itemsets by integrating row enumeration with column enumeration. Liu, Han, Xin, and Shao [LHXS06] proposed TDClose to mine frequent closed patterns in high-dimensional data by starting from the maximal rowset, integrated with a row-enumeration tree. It uses the pruning power of the minimum support threshold to reduce the search space. For mining rather long patterns, called *colossal patterns*, Zhu, Yan, Han, et al. [ZYH+07] developed a core Pattern-Fusion method that leaps over an exponential number of intermediate patterns to reach colossal patterns.

To generate a reduced set of patterns, recent studies have focused on mining compressed sets of frequent patterns. Closed patterns can be viewed as a lossless compression of frequent patterns, whereas maximal patterns can be viewed as a simple lossy compression of frequent patterns. Top-k patterns, such as by Wang, Han, Lu, and Tsvetkov [WHLT05], and error-tolerant patterns, such as by Yang, Fayyad, and Bradley [YFB01], are alternative forms of interesting patterns. Afrati, Gionis, and Mannila [AGM04] proposed to use k-itemsets to cover a collection of frequent itemsets. For frequent itemset compression, Yan, Cheng, Han, and Xin [YCHX05] proposed a profile-based approach, and Xin, Han, Yan, and Cheng [XHYC05] proposed a clustering-based approach. By taking into consideration both pattern significance and pattern redundancy, Xin, Cheng, Yan, and Han [XCYH06] proposed a method for extracting redundancy-aware top-k patterns.

Automated semantic annotation of frequent patterns is useful for explaining the meaning of patterns. Mei, Xin, Cheng, et al. [MXC+07] studied methods for semantic annotation of frequent patterns.

An important extension to frequent itemset mining is mining sequence and structural data. This includes mining sequential patterns (Agrawal and Srikant [AS95]; Pei, Han, Mortazavi-Asl, et al. [PHM-A+01, PHM-A+04]; and Zaki [Zak01]); mining frequent espisodes (Mannila, Toivonen, and Verkamo [MTV97]); mining structural

patterns (Inokuchi, Washio, and Motoda [IWM98]; Kuramochi and Karypis [KK01]; and Yan and Han [YH02]); mining cyclic association rules (Özden, Ramaswamy, and Silberschatz [ORS98]); intertransaction association rule mining (Lu, Han, and Feng [LHF98]); and calendric market basket analysis (Ramaswamy, Mahajan, and Silberschatz [RMS98]). Mining such patterns is considered an advanced topic and readers are referred to these sources.

Pattern mining has been extended to help effective data classification and clustering. Pattern-based classification (Liu, Hsu, and Ma [LHM98] and Cheng, Yan, Han, and Hsu [CYHH07]) is discussed in Chapter 9. Pattern-based cluster analysis (Agrawal, Gehrke, Gunopulos, and Raghavan [AGGR98] and H. Wang, W. Wang, Yang, and Yu [WWYY02]) is discussed in Chapter 11.

Pattern mining also helps many other data analysis and processing tasks such as cube gradient mining and discriminative analysis (Imielinski, Khachiyan, and Abdulghani [IKA02]; Dong, Han, Lam, et al. [DHL+04]; Ji, Bailey, and Dong [JBD05]), discriminative pattern-based indexing (Yan, Yu, and Han [YYH05]), and discriminative pattern-based similarity search (Yan, Zhu, Yu, and Han [YZYH06]).

Pattern mining has been extended to mining spatial, temporal, time-series, and multimedia data, and data streams. Mining spatial association rules or spatial collocation rules was studied by Koperski and Han [KH95]; Xiong, Shekhar, Huang, et al. [XSH+04]; and Cao, Mamoulis, and Cheung [CMC05]. Pattern-based mining of time-series data is discussed in Shieh and Keogh [SK08] and Ye and Keogh [YK09]. There are many studies on pattern-based mining of multimedia data such as Zaïane, Han, and Zhu [ZHZ00] and Yuan, Wu, and Yang [YWY07]. Methods for mining frequent patterns on stream data have been proposed by many researchers, including Manku and Motwani [MM02]; Karp, Papadimitriou, and Shenker [KPS03]; and Metwally, Agrawal, and El Abbadi [MAE05]. These pattern mining methods are considered advanced topics.

Pattern mining has broad applications. Application areas include computer science such as software bug analysis, sensor network mining, and performance improvement of operating systems. For example, CP-Miner by Li, Lu, Myagmar, and Zhou [LLMZ04] uses pattern mining to identify copy-pasted code for bug isolation. PR-Miner by Li and Zhou [LZ05] uses pattern mining to extract application-specific programming rules from source code. Discriminative pattern mining is used for program failure detection to classify software behaviors (Lo, Cheng, Han, et al. [LCH+09]) and for troubleshooting in sensor networks (Khan, Le, Ahmadi, et al. [KLA+08]).