UNIT-III

Data Mining - Frequent Pattern Analysis

Syllabus

Data Mining - Frequent Pattern Analysis: Mining Frequent Patterns, Associations and Correlations, Mining Methods, Pattern Evaluation Method, Pattern Mining in Multilevel, Multi- Dimensional Space – Constraint Based Frequent Pattern Mining, Classification using Frequent Patterns

Association Analysis

Association mining aims to extract interesting correlations, frequent patterns, associations or casual structures among sets of items or objects in transaction databases, relational database or other data repositories. Association rules are widely used in various areas such as telecommunication networks, market and risk management, inventory control, cross-marketing, catalog design, loss-leader analysis, clustering, classification, etc.

Examples:

Rule Form: Body->Head [Support, confidence]

Buys (X, “Computer”) ->Buys (X, “Software”) [40%, 50%]

Association rule: basic concepts:

- Given: (1) database of transaction, (2) each transaction is a list of items (purchased by a customer in visit)
- Find: all rules that correlate the presence of one set of items with that of another set of items.
- E.g., 98% of people who purchase tires and auto accessories also get automotive services done.
- E.g., Market Basket Analysis
- This process analyzes customer buying habits by finding associations between the different items that customers place in their “Shopping Baskets”. The discovery of such associations can help retailers develop marketing strategies by gaining insight into which items are frequently purchased together by customer.

Applications:

- Maintenance agreement (what the store should do to boost maintenance agreement sales)
- Home Electronics (what other products should the store stocks up?)
- Attached mailing in direct marketing

Association Rule:

An association rule is an implication expression of the form X->Y, where X and Y are disjoint itemsets, i.e., $X \cap Y = \emptyset$. The strength of an association rule can be measured in terms of its support and confidence. Support determines how often a rule is applicable to a given data set, while confidence determines how frequently items in Y appear in

transactions that contain X. The formal definition of these metrics are

Support, $s(X->Y) = \frac{(X \cup Y)}{N}$

Confidence, $c(X->Y) = \frac{(X \cup Y)}{\sigma(X)}$

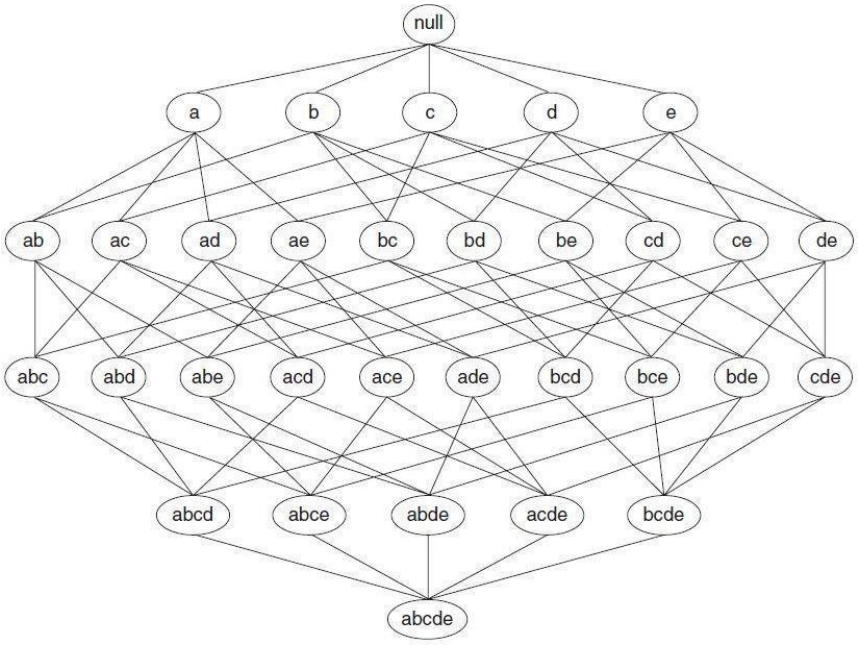
Why Use Support and Confidence? Support is an important measure because a rule that has very low support may occur simply by chance. A low support rule is also likely to be uninteresting from a business perspective because it may not be profitable to promote items that customers seldom buy together. For these reasons, support is often used to eliminate uninteresting rules.

Confidence, on the other hand, measures the reliability of the inference made by a rule. For a given rule $X \rightarrow Y$, the higher the confidence, the more likely it is for Y to be present in transactions that contain X. Confidence also provides an estimate of the conditional probability of Y given X.

Therefore, a common strategy adopted by many association rule mining algorithms is to decompose the problem into two major subtasks:

1. Frequent Itemset Generation, whose objective is to find all the item- sets that satisfy the *minsup* threshold. These itemsets are called frequent itemsets.

2. Rule Generation, whose objective is to extract all the high-confidence rules from the frequent itemsets found in the previous step. These rules are called strong rules.



Frequent Itemset Generation:

A lattice structure can be used to enumerate the list of all possible itemsets. Above Figure shows an itemset lattice for $I = \{a, b, c, d, e\}$. In general, a data set that contains k items can potentially generate up to $2^k - 1$ frequent itemsets, excluding the null set. Because k can be very large in many practical applications, the search space of itemsets that need to be explored is exponentially large.

To find frequent itemsets we have two algorithms,

a) Apriori Algorithm

b) FP-Growth

a) Apriori Algorithm:

Apriori is a seminal algorithm proposed by R. Agrawal and R. Srikant in 1994 for mining frequent itemsets for Boolean association rules. The name of the algorithm is based on the fact that the algorithm uses *prior knowledge* of frequent itemset properties, as we shall see later. Apriori employs an iterative approach known as a *level-wise* search, where *k*- itemsets are used to explore (k+1)-itemsets.

First, the set of frequent 1-itemsets is found by scanning the database to accumulate the count for each item, and collecting those items that satisfy minimum support. The resulting set is denoted by L1. Next, L1 is used to find L2, the set of frequent 2-itemsets, which is used to find L3, and so on, until no more frequent k-itemsets can be found. The finding of each Lk requires one full scan of the database.

To improve the efficiency of the level-wise generation of frequent itemsets, an important property called the Apriori property is used to reduce the search space.

Apriori property: *All nonempty subsets of a frequent itemset must also be frequent.*

The Apriori property is based on the following observation. By definition, if an itemset *I* does not satisfy the minimum support threshold, *min sup*, then *I* is not frequent, that is, *P(I)< min sup*. If an item *A* is added to the itemset *I*, then the resulting itemset (i.e.,IUA) cannot occur more frequently than *I*. Therefore, IUA is not frequent either, that is, *P*(IUA) < *min sup*.

This property belongs to a special category of properties called **antimonotonicity** in the sense that *if a set cannot pass a test, all of its supersets will fail the same test as well*. It is called *antimonotonicity* because the property is monotonic in the context of failing a test.

A two-step process is followed, consisting of **join** and **prune** actions.

1. **The join step**: To find L_k, a set of **candidate** *k*-itemsets is generated by joining L_{k-1} with itself. This set of candidates is denoted C_k.

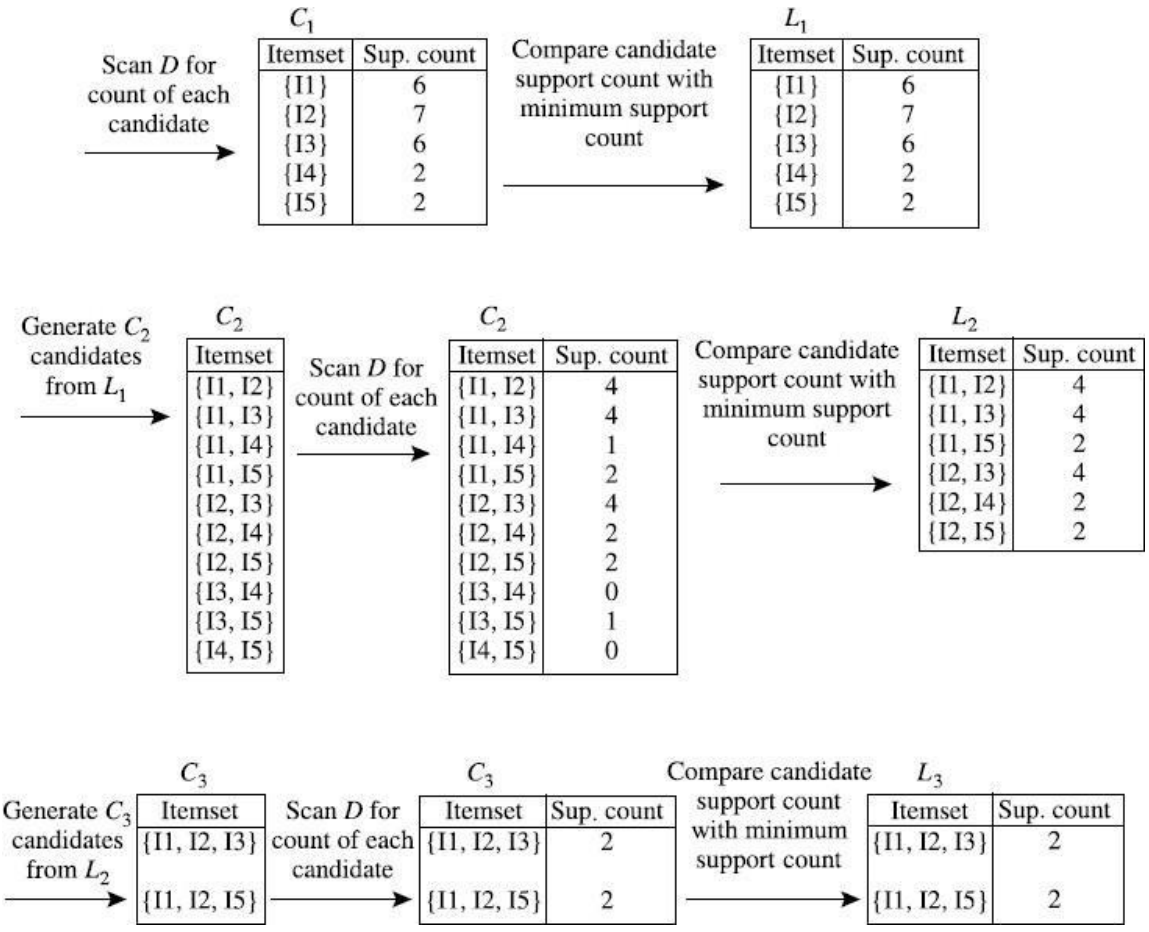
2. **The prune step**: C_k is a superset of L_k, that is, its members may or may not be frequent, but all of the frequent *k*-itemsets are included in *Ck*. A database scan to determine the count of each candidate in C_k would result in the determination of L_k.

Example:

Transactional Data for an *AllElectronics* Branch

TID	*List of item_IDs*
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

1. In the first iteration of the algorithm, each item is a member of the set of candidate 1- itemsets, C_1. The algorithm simply scans all of the transactions to count the number of occurrences of each item.
2. Suppose that the minimum support count required is 2, that is, *min sup* = 2. (Here, we are referring to *absolute* support because we are using a support count. The corresponding relative support is 2/9 = 22%.) The set of frequent 1-itemsets, L_1, can then be determined. It consists of the candidate 1-itemsets satisfying minimum support. In our example, all of the candidates in C_1 satisfy minimum support.
3. To discover the set of frequent 2-itemsets, L_2, the algorithm uses the join $L_1 \bowtie L_1$ to generate a candidate set of 2-itemsets, C_2. C_2 consists of 2-itemsets. Note that no candidates are removed from C_2 during the prune step because each subset of the candidates is also frequent.
4. Next, the transactions in D are scanned and the support count of each candidate itemset in C_2 is accumulated, as shown in the middle table of the second row in Figure
5. The set of frequent 2-itemsets, L_2, is then determined, consisting of those candidate 2- itemsets in C_2 having minimum support.



6. The generation of the set of the candidate 3-itemsets, C_3, is detailed in Figure From the join step, we first get $C_3 = L_2 \bowtie L_2 = \{\{I1, I2, I3\}, \{I1, I2, I5\}, \{I1, I3, I5\}, \{I2, I3, I4\}, \{I2, I3, I5\}, \{I2, I4, I5\}\}$ Based on the Apriori property that all subsets of a frequent itemset must also be frequent, we can determine that the four latter candidates cannot possibly be frequent. We therefore remove them from C_3, thereby saving the effort of unnecessarily obtaining their counts during the subsequent scan of D to determine L_3.
7. The transactions in D are scanned to determine L_3, consisting of those candidate 3-itemsets in C_3 having minimum support.
8. The algorithm uses $L_3 \bowtie L3$ to generate a candidate set of 4-itemsets, C_4. Although the join results in {I1, I2, I3, I5}, itemset {I1, I2, I3, I5} is pruned because its subset {I2, I3, I5} is not frequent. Thus, $C_4 \neq \emptyset$, and the algorithm terminates, having found all of the frequent itemsets.

b) FP-Growth:

FP-growth (finding frequent itemsets without candidate generation). We reexamine the mining of transaction database, *D*, of Table in previous Example using the frequent pattern growth approach.

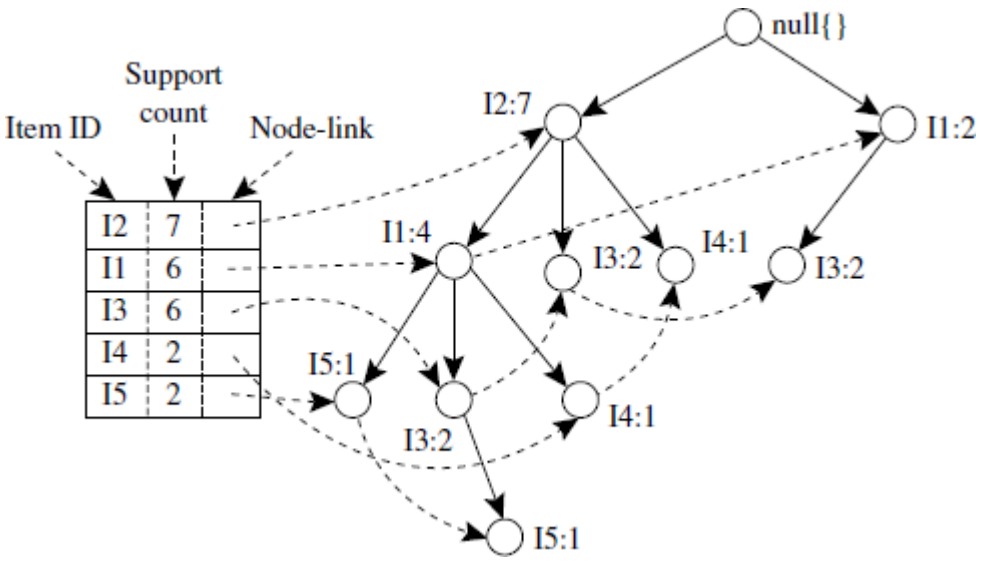
Transactional Data for an *AllElectronics* Branch

TID	*List of item_IDs*
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

The first scan of the database is the same as Apriori, which derives the set of frequent items (1-itemsets) and their support counts (frequencies). Let the minimum support count be

2. The set of frequent items is sorted in the order of descending support count. This resulting set or *list* is denoted by *L*. Thus, we have L = {{I2:7}, {I1:6}, {I3:6}, {I4:2}, {I5:2}}

An FP-tree is then constructed as follows. First, create the root of the tree, labeled with "null." Scan database D a second time. The items in each transaction are processed in L order (i.e., sorted according to descending support count), and a branch is created for each transaction.



The FP-tree is mined as follows. Start from each frequent length-1 pattern (as an initial **suffix pattern**), construct its **conditional pattern base** (a "sub-database," which consists of the set of *prefix paths* in the FP-tree co-occurring with the suffix pattern), then construct its (*conditional*) FP-tree, and perform mining recursively on the tree. The pattern growth is achieved by the concatenation of the suffix pattern with the frequent patterns generated from a conditional FP-tree.

Mining the FP-Tree by Creating Conditional (Sub-)Pattern Bases

Item	*Conditional Pattern Base*	*Conditional FP-tree*	*Frequent Patterns Generated*
I5	{{I2, I1: 1}, {I2, I1, I3: 1}}	⟨I2: 2, I1: 2⟩	{I2, I5: 2}, {I1, I5: 2}, {I2, I1, I5: 2}
I4	{{I2, I1: 1}, {I2: 1}}	⟨I2: 2⟩	{I2, I4: 2}
I3	{{I2, I1: 2}, {I2: 2}, {I1: 2}}	⟨I2: 4, I1: 2⟩, ⟨I1: 2⟩	{I2, I3: 4}, {I1, I3: 4}, {I2, I1, I3: 2}
I1	{{I2: 4}}	⟨I2: 4⟩	{I2, I1: 4}

Finally, we can conclude that frequent itemsets are {I2, I1, I5} and {I2, I1, I3}.

Generating Association Rules from Frequent Itemsets

Once the frequent itemsets from transactions in a database *D* have been found, it is straightforward to generate strong association rules from them (where *strong* association rules satisfy both minimum support and minimum confidence). This can be done using Eq. for confidence, which we show again here for completeness:

$$confidence(A \Rightarrow B) = P(B|A) = \frac{support\_count(A \cup B)}{support\_count(A)}.$$

The conditional probability is expressed in terms of itemset support count, where $support\_count(A \cup B)$ is the number of transactions containing the itemsets $A \cup B$, and $support\_count(A)$ is the number of transactions containing the itemset A. Based on this equation, association rules can be generated as follows:

- For each frequent itemset l, generate all nonempty subsets of l.
- For every nonempty subset s of l, output the rule "$s \Rightarrow (l - s)$" if $\frac{support\_count(l)}{support\_count(s)} \geq min\_conf$, where min_conf is the minimum confidence threshold.

Because the rules are generated from frequent itemsets, each one automatically satisfies the minimum support. Frequent itemsets can be stored ahead of time in hash tables along with their counts so that they can be accessed quickly.

Generating association rules. Let's try an example based on the transactional data for *AllElectronics* shown before in Table 6.1. The data contain frequent itemset $X = \{I1, I2, I5\}$. What are the association rules that can be generated from X? The nonempty subsets of X are $\{I1, I2\}$, $\{I1, I5\}$, $\{I2, I5\}$, $\{I1\}$, $\{I2\}$, and $\{I5\}$. The resulting association rules are as shown below, each listed with its confidence:

$$\{I1, I2\} \Rightarrow I5, \quad confidence = 2/4 = 50\%$$
$$\{I1, I5\} \Rightarrow I2, \quad confidence = 2/2 = 100\%$$
$$\{I2, I5\} \Rightarrow I1, \quad confidence = 2/2 = 100\%$$
$$I1 \Rightarrow \{I2, I5\}, \quad confidence = 2/6 = 33\%$$
$$I2 \Rightarrow \{I1, I5\}, \quad confidence = 2/7 = 29\%$$
$$I5 \Rightarrow \{I1, I2\}, \quad confidence = 2/2 = 100\%$$

If the minimum confidence threshold is, say, 70%, then only the second, third, and last rules are output, because these are the only ones generated that are strong. Note that, unlike conventional classification rules, association rules can contain more than one conjunct in the right side of the rule. ■

Mining Frequent Itemsets Using the Vertical Data Format

Both the Apriori and FP-growth methods mine frequent patterns from a set of trans- actions in *TID-itemset* format (i.e., $\{TID : itemset\}$), where *TID* is a transaction ID and *itemset* is the set of items bought in transaction *TID*. This is known as the **horizontal data format**. Alternatively, data can be presented in *item-TID set* format

Algorithm: **FP growth.** Mine frequent itemsets using an FP-tree by pattern fragment growth.

Input:

D, a transaction database;

min sup, the minimum support count threshold.

Output: The complete set of frequent patterns.

Method:

1. The FP-tree is constructed in the following steps:

(a) Scan the transaction database *D* once. Collect *F*, the set of frequent items,

and their support counts. Sort F in support count descending order as L, the *list* of frequent items.

(b) Create the root of an FP-tree, and label it as "null." For each transaction *Trans* in D do the following.
Select and sort the frequent items in *Trans* according to the order of L. Let the sorted frequent item list in *Trans* be $[p\ P]$, where p is the first element and P is the remaining list. Call insert tree$([p\ P], T)$, which is performed as follows. If T has a child N such that *N.item-name* = *p.item-name*, then increment N's count by 1; else create a new node N, and let its count be 1, its parent link be linked to T, and its node-link to the nodes with the same *item-name* via the node-link structure. If P is nonempty, call insert tree(P, N) recursively.

2. The FP-tree is mined by calling FP growth$(FP\ tree, null)$, which is implemented as follows.

```
procedure FP growth(Tree, α)
(1)   if Tree contains a single path P then
(2)     for each combination (denoted as β) of the nodes in the path P
(3)       generate pattern β ∪ α with support count = minimum support count of nodes in β;
(4)   else for each ai in the header of Tree {
(5)     generate pattern β = ai ∪ α with support count = ai.support count;
(6)     construct β's conditional pattern base and then β's conditional FP tree Treeβ;
(7)     if Treeβ /= ∅ then
(8)       call FP growth(Treeβ, β); }
```

Figure 6.9 FP-growth algorithm for discovering frequent itemsets without candidate generation.

(i.e., *item* : *TID set*), where *item* is an item name, and *TID set* is the set of transaction identifiers containing the item. This is known as the **vertical data format**.
In this subsection, we look at how frequent itemsets can also be mined effi- ciently using vertical data format, which is the essence of the **Eclat** (Equivalence Class Transformation) algorithm.

Example 6.6 Mining frequent itemsets using the vertical data format. Consider the horizontal data format of the transaction database, D, of Table 6.1

in Example 6.3. This can be transformed into the vertical data format shown in Table 6.3 by scanning the dataset once.

Mining can be performed on this data set by intersecting the TID sets of every pair of frequent single items. The minimum support count is 2. Because every single item is

Table 6.3 The Vertical Data Format of the Transaction DataSet *D* of Table 6.1

itemset	*TID set*
I1	{T100, T400, T500, T700, T800, T900}
I2	{T100, T200, T300, T400, T600, T800, T900}I3
	{T300, T500, T600, T700, T800, T900}
I4	{T200, T400}
I5	{T100, T800}

Table 6.4 2-Itemsets in Vertical Data Format

itemset	*TID set*
{I1, I2}	{T100, T400, T800, T900}
{I1, I3}	{T500, T700, T800, T900}
{I1, I4}	{T400}
{I1, I5}	{T100, T800}
{I2, I3}	{T300, T600, T800, T900}
{I2, I4}	{T200, T400}
{I2, I5}	{T100, T800}
{I3, I5}	{T800}

Table 6.5 3-Itemsets in Vertical Data Format

itemset	*TID set*
{I1, I2, I3}	{T800, T900}
{I1, I2, I5}	{T100, T800}

frequent in Table 6.3, there are 10 intersections performed in total, which lead to eight nonempty 2-itemsets, as shown in Table 6.4. Notice that because the itemsets I1, I4 and I3, I5 each contain only one transaction, they do not belong to the set of frequent 2-itemsets.

Based on the Apriori property, a given 3-itemset is a candidate 3-itemset only if every one of its 2-itemset subsets is frequent. The candidate generation process here will generate only two 3-itemsets: {I1, I2, I3} and {I1, I2, I5}. By intersecting the TID sets of any two corresponding 2-itemsets of these candidate 3-itemsets, it derives Table 6.5, where there are only two frequent 3-itemsets: {I1, I2, I3: 2} and {I1, I2, I5: 2}.

Example 6.6 illustrates the process of mining frequent itemsets by exploring the vertical data format. First, we transform the horizontally formatted data into the vertical format by scanning the data set once. The support count of an itemset is simply the length of the TID set of the itemset. Starting with k = 1, the frequent k-itemsets can be used to construct the candidate (k + 1)-itemsets based on the Apriori property.

The computation is done by intersection of the TID sets of the frequent *k*-itemsets to compute the TID sets of the corresponding *(k 1)*-itemsets. This process repeats, with *k* incremented by 1 each time, until no frequent itemsets or candidate itemsets can be found.

Besides taking advantage of the Apriori property in the generation of candidate *(k 1)*-itemset from frequent *k*-itemsets, another merit of this method is that there is no need to scan the database to find the support of *(k 1)*-itemsets (for *k* 1). This is because the TID set of each *k*-itemset carries the complete information required for counting such support. However, the TID sets can be quite long, taking substantial memory space as well as computation time for intersecting the long sets.

To further reduce the cost of registering long TID sets, as well as the subsequent costs of intersections, we can use a technique called *diffset*, which keeps track of only the differences of the TID sets of a *(k 1)*-itemset and a corresponding *k*-itemset. For instance, in Example 6.6 we have $\{I1\} = \{T100, T400, T500, T700, T800, T900\}$ and $\{I1, I2\} = \{T100, T400, T800, T900\}$. The *diffset* between the two is *diffset*$(\{I1, I2\}, \{I1\}) = \{T500, T700\}$. Thus, rather than recording the four TIDs that make up the intersection of I1 and I2, we can instead use *diffset* to record just two TIDs, indicating the difference between I1 and I1, I2. Experiments show that in certain situations, such as when the data set contains many dense and long patterns, this technique can substantially reduce the total cost of vertical format mining of frequent itemsets.

Mining Closed and Max Patterns

In Section 6.1.2 we saw how frequent itemset mining may generate a huge number of frequent itemsets, especially when the *min sup* threshold is set low or when there exist long patterns in the data set. Example 6.2 showed that closed frequent itemsets[9] can substantially reduce the number of patterns generated in frequent itemset mining while preserving the complete information regarding the set of frequent itemsets. That is, from the set of closed frequent itemsets, we can easily derive the set of frequent itemsets and their support. Thus, in practice, it is more desirable to mine the set of closed frequent itemsets rather than the set of all frequent itemsets in most cases.

"How can we mine closed frequent itemsets?" A naïve approach would be to first mine the complete set of frequent itemsets and then remove every frequent itemset that is a proper subset of, and carries the same support as, an existing frequent itemset. However, this is quite costly. As shown in Example 6.2, this method would have to first derive $2^{100} - 1$ frequent itemsets to obtain a length-100 frequent itemset, all before it could begin to eliminate redundant itemsets. This is prohibitively expensive. In fact, there exist only a very small number of closed frequent itemsets in Example 6.2's data set.

A recommended methodology is to search for closed frequent itemsets directly during the mining process. This requires us to prune the search space as soon as we can identify the case of closed itemsets during mining. Pruning strategies include the following:

Item merging: If every transaction containing a frequent itemset X also contains an itemset Y but not any proper superset of Y, then $X \cup Y$ forms a frequent closed itemset and there is no need to search for any itemset containing X but no Y.

For example, in Table 6.2 of Example 6.5, the projected conditional database for prefix itemset {I5:2} is {{I2, I1}, {I2, I1, I3}}, from which we can see that each of its transactions contains itemset {I2, I1} but no proper superset of {I2, I1}. Itemset {I2, I1} can be merged with {I5} to form the closed itemset, {I5, I2, I1: 2}, and we do not need to mine for closed itemsets that contain I5 but not {I2, I1}.

Sub-itemset pruning: If a frequent itemset X is a proper subset of an already found frequent closed itemset Y and support count(X) = support count(Y), then X and all of X's descendants in the set enumeration tree cannot be frequent closed itemsets and thus can be pruned.

Similar to Example 6.2, suppose a transaction database has only two transactions: {(a1, a2, . . . , a100), (a1, a2, . . . , a50)}, and the minimum support count is min sup = 2. The projection on the first item, a1, derives the frequent itemset,

{a1, a2, . . . , a50 : 2}, based on the itemset merging optimization. Because support({a2}) = support({a1, a2, . . . , a50}) = 2, and {a2} is a proper subset of {a1, a2, . . . , a50}, there is no need to examine a2 and its projected database. Similar pruning can be done for a3, . . . , a50 as well. Thus, the mining of closed frequent itemsets in this data set terminates after mining a1's projected database.

Item skipping: In the depth-first mining of closed itemsets, at each level, there will be a prefix itemset X associated with a header table and a projected database. If a local frequent item p has the same support in several header tables at different levels, we can safely prune p from the header tables at higher levels.
Consider, for example, the previous transaction database having only two transactions: {(a1, a2, . . . , a100), (a1, a2, . . . , a50)}, where min sup = 2. Because a2 in a1's projected database has the same support as a2 in the global header table, a2 can be pruned from the global header table. Similar pruning can be done for a3, . . . , a50. There is no need to mine anything more after mining a1's projected database.

Besides pruning the search space in the closed itemset mining process, another important optimization is to perform efficient checking of each newly derived frequent itemset to see whether it is closed. This is because the mining process cannot ensure that every generated frequent itemset is closed.
When a new frequent itemset is derived, it is necessary to perform two kinds of closure checking: (1) superset checking, which checks if this new frequent itemset is a superset of some already found closed itemsets with the same support, and (2) subset checking, which checks whether the newly found itemset is a subset of an already found closed itemset with the same support.
If we adopt the item merging pruning method under a divide-and$_{\nabla}$conquer framework, then the superset checking is actually built-in and there is no need to explicitly perform superset checking. This is because if a frequent itemset X U Y is found later than itemset X, and carries the same support as X, it must be in X's projected database and must have been generated during itemset merging.
To assist in subset checking, a compressed **pattern-tree** can be constructed to main- tain the set of closed itemsets mined so far. The pattern-tree is similar in structure to the FP-tree except that all the closed itemsets found are stored explicitly in the corresponding tree branches. For efficient subset checking, we can use the following property: If the current itemset Sc can be subsumed by another already found closed itemset Sa, then (1) Sc and Sa have the same support, (2) the length of Sc is smaller than that of Sa, and (3) all of the items in Sc are contained in Sa.

Based on this property, a **two-level hash index structure** can be built for fast access- ing of the pattern-tree: The first level uses the identifier of the last item in Sc as a hash key (since this identifier must be within the branch of Sc), and the second level uses the sup- port of Sc as a hash key (since Sc and Sa have the same support). This will substantially speed up the subset checking process.

This discussion illustrates methods for efficient mining of closed frequent itemsets. "Can we extend these methods for efficient mining of maximal frequent itemsets?" Because maximal frequent itemsets share many similarities with closed frequent itemsets, many of the optimization techniques developed here can be extended to mining maximal frequent itemsets. However, we leave this method as an exercise for interested readers.

Which Patterns Are Interesting?—PatternEvaluation Methods

Most association rule mining algorithms employ a support–confidence framework. Although minimum support and confidence thresholds help weed out or exclude the exploration of a good number of uninteresting rules, many of the rules generated are still not interesting to the users. Unfortunately, this is especially true when mining at low support thresholds or mining for long patterns. This has been a major bottleneck for successful application of association rule mining.

In this section, we first look at how even strong association rules can be uninteresting and misleading (Section 6.3.1). We then discuss how the support–confidence frame- work can be supplemented with additional interestingness measures based on correlation analysis (Section 6.3.2). Section 6.3.3 presents additional pattern evaluation measures. It then provides an overall comparison of all the measures discussed here. By the end, you will learn which pattern evaluation measures are most effective for the discovery of only interesting rules.

Strong Rules Are Not Necessarily Interesting

Whether or not a rule is interesting can be assessed either subjectively or objectively. Ultimately, only the user can judge if a given rule is interesting, and this judgment, being ubjective, may differ from one user to another. However, objective interestingness mea- sures, based on the statistics "behind" the data, can be used as one step toward the goal of weeding out uninteresting rules that would otherwise be presented to the user.

"How can we tell which strong association rules are really interesting?" Let's examine the following example.

Example 6.7 A misleading "strong" association rule. Suppose we are interested in analyzing trans- actions at AllElectronics with respect to the purchase of computer games and videos. Let game refer to the transactions containing computer games, and video refer to those containing videos. Of the 10,000 transactions analyzed, the data show that 6000 of the customer transactions included computer games, while 7500 included videos, and 4000 included both computer games and videos. Suppose that a data mining program for discovering association rules is run on the data, using a minimum support of, say, 30% and a minimum confidence of 60%. The following association rule is discovered:

$$buys(X, \text{"}computer\ games\text{"}) \Rightarrow buys(X, \text{"}videos\text{"}) \quad [support = 40\%, confidence = 66\%]. \qquad (6.6)$$

Rule (6.6) is a strong association rule and would therefore be reported, since its support value of $\frac{4000}{10,000} = 40\%$ and confidence value of $\frac{4000}{6000} = 66\%$ satisfy the minimum support and minimum confidence thresholds, respectively. However, Rule (6.6) is misleading because the probability of purchasing videos is 75%, which is even larger than 66%. In fact, computer games and videos are negatively associated because the purchase of one of these items actually decreases the likelihood of purchasing the other. Without fully understanding this phenomenon, we could easily make unwise business decisions based on Rule (6.6).

Example 6.7 also illustrates that the confidence of a rule $A \Rightarrow B$ can be deceiving. It does not measure the *real strength* (or lack of strength) of the *correlation* and *implica- tion* between *A* and *B*. Hence, alternatives to the support–confidence framework can be useful in mining interesting data relationships.

From Association Analysis to Correlation Analysis

As we have seen so far, the support and confidence measures are insufficient at filtering out uninteresting association rules. To tackle this weakness, a correlation measure can be used to augment the support–confidence framework for association rules. This leads to correlation rules of the form

$$A \Rightarrow B \text{ [support, confidence, correlation]}. \qquad (6.7)$$

That is, a correlation rule is measured not only by its support and confidence but also by the correlation between itemsets A and B. There are many different

correlation mea- sures from which to choose. In this subsection, we study several correlation measures to determine which would be good for mining large data sets.

Lift is a simple correlation measure that is given as follows. The occurrence of itemset A is **independent** of the occurrence of itemset B if P(A B) P(A)P(B); otherwise, itemsets A and B are **dependent** and **correlated** as events. This definition can easily be extended to more than two itemsets. The **lift** between the occurrence of A and B can be measured by computing

$$lift(A, B) = \frac{P(A \cup B)}{P(A)P(B)}. \qquad (6.8)$$

If the resulting value of Eq. (6.8) is less than 1, then the occurrence of A is negatively correlated with the occurrence of B, meaning that the occurrence of one likely leads to the absence of the other one. If the resulting value is greater than 1, then A and B are positively correlated, meaning that the occurrence of one implies the occurrence of the other. If the resulting value is equal to 1, then A and B are independent and there is no correlation between them.

Equation (6.8) is equivalent to P(B A)/P(B), or conf(A B)/sup(B), which is also referred to as the lift of the association (or correlation) rule A B. In other words, it assesses the degree to which the occurrence of one "lifts" the occurrence of the other. For example, if A corresponds to the sale of computer games and B corresponds to the sale of videos, then given the current market conditions, the sale of games is said to increase or "lift" the likelihood of the sale of videos by a factor of the value returned by Eq. (6.8).

Example 6.8 Correlation analysis using lift. To help filter out misleading "strong" associations of the form $A \Rightarrow B$ from the data of Example 6.7, we need to study how the two item- sets, *A* and *B*, are correlated. Let *game* refer to the transactions of Example 6.7 that do not contain computer games, and *video* refer to those that do not contain videos. The transactions can be summarized in a *contingency table*, as shown in Table 6.6.

From the table, we can see that the probability of purchasing a computer game is $P(game) = 0.60$, the probability of purchasing a video is $P(video) = 0.75$, and the probability of purchasing both is $P(game, video) = 0.40$. By Eq. (6.8), the lift of Rule (6.6) is $P(game, video)/(P(game) \times P(video)) = 0.40/(0.60 \times 0.75) = 0.89$. Because this value is less than 1, there is a negative correlation between the occur- rence of

■

game and *video* . The numerator is the likelihood of a customer purchasing both, while the denominator is what the likelihood would have been if the two pur- chases were completely independent. Such a negative correlation cannot be identified by a support–confidence framework.

The second correlation measure that we study is the χ^2 measure, which was introduced in Chapter 3 (Eq. 3.1). To compute the χ^2 value, we take the squared difference between the observed and expected value for a slot (*A* and *B* pair) in the contin- gency table, divided by the expected value. This amount is summed for all slots of the contingency table. Let's perform a χ^2 analysis of Example 6.8.

Table 6.6 2 × 2 Contingency Table Summarizing the Transactions with Respect to Game and Video Purchases

	game	$\overline{game}$	Σ_{row}
video	4000	3500	7500
$\overline{video}$	2000	500	2500
Σ_{col}	6000	4000	10,000

Table 6.7 Table 6.6 Contingency Table, Now with the Expected Values

	game	$\overline{game}$	Σ_{row}
video	4000 (4500)	3500 (3000)	7500
$\overline{video}$	2000 (1500)	500 (1000)	2500
Σ_{col}	6000	4000	10,000

Example 6.9 Correlation analysis using χ^2. To compute the correlation using χ^2 analysis for nominal data, we need the observed value and expected value (displayed in parenthesis) for each slot of the contingency table, as shown in Table 6.7. From the table, we can compute the χ^2 value as follows:

$$\chi^2 = \sum \frac{(observed - expected)^2}{expected} = \frac{(4000 - 4500)^2}{4500} + \frac{(3500 - 3000)^2}{3000} + \frac{(2000 - 1500)^2}{1500} + \frac{(500 - 1000)^2}{1000} = 555.6.$$

Because the χ^2 value is greater than 1, and the observed value of the slot (*game*, *video*)

4000, which is less than the expected value of 4500, *buying game* and *buying video* are *negatively correlated*. This is consistent with the conclusion derived from the analysis of the *lift* measure in Example 6.8.

A Comparison of Pattern Evaluation Measures

The above discussion shows that instead of using the simple support–confidence framework to evaluate frequent patterns, other measures, such as *lift* and χ^2, often disclose more intrinsic pattern relationships. How effective are these measures? Should we also consider other alternatives?

Researchers have studied many pattern evaluation measures even before the start of in-depth research on scalable methods for mining frequent patterns. Recently, several other pattern evaluation measures have attracted interest. In this subsection, we present four such measures: *all confidence*, *max confidence*, *Kulczynski*, and *cosine*. We'll then compare their effectiveness with respect to one another and with respect to the *lift* and χ^2 measures.

Given two itemsets, A and B, the **all confidence** measure of A and B is defined as

$$all\_conf(A,B) = \frac{sup(A \cup B)}{max\{sup(A), sup(B)\}} = min\{P(A|B), P(B|A)\}, \quad (6.9)$$

where $max\_sup(A)$, $sup(B)$ is the maximum support of the itemsets A and B. Thus, $all\_conf(A,B)$ is also the minimum confidence of the two association rules related to A and B, namely, "$A \Rightarrow B$" and "$B \Rightarrow A$."

Given two itemsets, A and B, the **max confidence** measure of A and B is defined as

$$max\_conf(A,B) = max\{P(A|B), P(B|A)\}. \quad (6.10)$$

The *max conf* measure is the maximum confidence of the two association rules, "$A \Rightarrow B$" and "$B \Rightarrow A$."

Given two itemsets, A and B, the **Kulczynski** measure of A and B (abbreviated as **Kulc**) is defined as

$$Kulc(A, B) = \frac{1}{2}(P(A|B) + P(B|A)). \quad (6.11)$$

It was proposed in 1927 by Polish mathematician S. Kulczynski. It can be viewed as an average of two confidence measures. That is, it is the average of two conditional probabilities: the probability of itemset *B* given itemset *A*, and the probability of itemset *A* given itemset *B*.

Finally, given two itemsets, *A* and *B*, the **cosine** measure of *A* and *B* is defined as

$$cosine(A, B) = \frac{P(A \cup B)}{\sqrt{P(A) \times P(B)}} = \frac{sup(A \cup B)}{\sqrt{sup(A) \times sup(B)}} = \sqrt{P(A|B) \times P(B|A)}. \quad (6.12)$$

The *cosine* measure can be viewed as a *harmonized lift* measure: The two formulae are similar except that for cosine, the *square root* is taken on the product of the probabilities of *A* and *B*. This is an important difference, however, because by taking the square root, the cosine value is only influenced by the supports of *A*, *B*, and $A \cup B$, and not by the total number of transactions.

Each of these four measures defined has the following property: Its value is only influenced by the supports of *A*, *B*, and $A \cup B$, or more exactly, by the conditional probabilities of $P(A|B)$ and $P(B|A)$, but not by the total number of transactions. Another common property is that each measure ranges from 0 to 1, and the higher the value, the closer the relationship between *A* and *B*.

Now, together with *lift* and χ^2, we have introduced in total six pattern evaluation measures. You may wonder, *"Which is the best in assessing the discovered pattern relationships?"* To answer this question, we examine their performance on some typical data sets.

Table 6.8 2 × 2 Contingency Table for Two Items

	milk	$\overline{milk}$	Σ_{row}
coffee	*mc*	$\overline{m}c$	*c*
$\overline{coffee}$	$m\overline{c}$	$\overline{mc}$	$\overline{c}$
Σ_{col}	*m*	$\overline{m}$	Σ

Table 6.9 Comparison of Six Pattern Evaluation Measures Using Contingency Tables for a Variety of Data Sets

Data Set	mc	$\overline{m}c$	$m\overline{c}$	$\overline{m}\overline{c}$	χ^2	*lift*	*all_conf.*	*max_conf.*	*Kulc.*	*cosine*
D_1	10,000	1000	1000	100,000	90557	9.26	0.91	0.91	0.91	0.91
D_2	10,000	1000	1000	100	0	1	0.91	0.91	0.91	0.91
D_3	100	1000	1000	100,000	670	8.44	0.09	0.09	0.09	0.09
D_4	1000	1000	1000	100,000	24740	25.75	0.5	0.5	0.5	0.5
D_5	1000	100	10,000	100,000	8173	9.18	0.09	0.91	0.5	0.29
D_6	1000	10	100,000	100,000	965	1.97	0.01	0.99	0.5	0.10

Example 6.10 Comparison of six pattern evaluation measures on typical data sets. The relationships between the purchases of two items, *milk* and *coffee*, can be examined by summarizing their purchase history in Table 6.8, a 2 2 contingency table, where an entry such as *mc* represents the number of transactions containing both milk and coffee.

Table 6.9 shows a set of transactional data sets with their corresponding contingency tables and the associated values for each of the six evaluation measures. Let's first examine the first four data sets, D_1 through D_4. From the table, we see that *m* and *c* are positively associated in D_1 and D_2, negatively associated in D_3, and neutral in D_4. For D_1 and D_2, *m* and *c* are positively associated because *mc* (10,000) is considerably greater than *mc* (1000) and *mc* (1000). Intuitively, for people who bought milk (*m* 10, 000 1000 11, 000), it is very likely that they also bought coffee (*mc/m* 10/11 91%), and vice versa.

The results of the four newly introduced measures show that *m* and *c* are strongly positively associated in both data sets by producing a measure value of 0.91. However, *lift* and χ^2 generate dramatically different measure values for D_1 and D_2 due to their sensitivity to *mc*. In fact, in many real-world scenarios, *mc* is usually huge and unstable. For example, in a market basket database, the total number of transactions could fluctuate on a daily basis and overwhelmingly exceed the number of transactions containing any particular itemset. Therefore, a good interestingness measure should not be affected by transactions that do not contain the itemsets of interest; otherwise, it would generate unstable results, as illustrated in D_1 and D_2.

Similarly, in D_3, the four new measures correctly show that *m* and *c* are strongly negatively associated because the *m* to *c* ratio equals the *mc* to *m* ratio, that is, 100/1100 = 9.1%. However, *lift* and χ^2 both contradict this in an incorrect way: Their values for D_2 are between those for D_1 and D_3.

For data set D_4, both *lift* and χ^2 indicate a highly positive association between *m* and *c*, whereas the others indicate a "neutral" association because the ratio of *mc* to *mc* equals the ratio of *mc* to *mc*, which is 1. This means that if a customer buys coffee (or milk), the probability that he or she will also purchase milk (or coffee) is exactly 50%.

"Why are lift *and* χ^2 *so poor at distinguishing pattern association relationships in the previous transactional data sets?"* To answer this, we have to consider the *null- transactions*. A **null-transaction** is a transaction that does not contain any of the item-sets being examined. In our example, *mc* represents the number of null-transactions. *Lift* and χ^2 have difficulty distinguishing interesting pattern association relationships because they are both strongly influenced by *mc*. Typically, the number of null- transactions can outweigh the number of individual purchases because, for example, many people may buy neither milk nor coffee. On the other hand, the other four measures are good indicators of interesting pattern associations because their defi- nitions remove the influence of *mc* (i.e., they are not influenced by the number of null-transactions).

This discussion shows that it is highly desirable to have a measure that has a value that is independent of the number of null-transactions. A measure is **null-invariant** if its value is free from the influence of null-transactions. Null-invariance is an impor- tant property for measuring association patterns in large transaction databases. Among the six discussed measures in this subsection, only *lift* and χ^2 are not null-invariant measures.

"*Among the* all confidence, max confidence, Kulczynski, *and* cosine *measures, which is best at indicating interesting pattern relationships*?"

To answer this question, we introduce the **imbalance ratio** (**IR**), which assesses the imbalance of two itemsets, *A* and *B*, in rule implications. It is defined as

$$IR(A, B) = \frac{|sup(A) - sup(B)|}{sup(A) + sup(B) - sup(A \cup B)}. \quad (6.13)$$

where the numerator is the absolute value of the difference between the support of the itemsets A and B, and the denominator is the number of transactions containing A or

A. If the two directional implications between A and B are the same, then $IR(A, B)$ will be zero. Otherwise, the larger the difference between the two, the larger the imbalance ratio. This ratio is independent of the number of null-transactions and independent of the total number of transactions.

Let's continue examining the remaining data sets in Example 6.10.

Example 6.11 Comparing null-invariant measures in pattern evaluation. Although the four measures introduced in this section are null-invariant, they may present dramatically different values on some subtly different data sets. Let's examine data sets D_5 and D_6, shown earlier in Table 6.9, where the two events m and c have unbalanced conditional probabilities. That is, the ratio of mc to c is greater than 0.9. This means that knowing that c occurs should strongly suggest that m occurs also. The ratio of mc to m is less than 0.1, indicating that m implies that c is quite unlikely to occur. The *all confidence* and *cosine* measures view both cases as negatively associated and the *Kulc* measure views both as neutral. The *max confidence* measure claims strong positive associations for these cases. The measures give very diverse results!

"*Which measure intuitively reflects the true relationship between the purchase of milk and coffee?*" Due to the "*balanced*" skewness of the data, it is difficult to argue whether the two data sets have positive or negative association. From one point of view, only $mc/(mc + mc) = 1000/(1000 + 10,000) = 9.09\%$ of milk-related transactions contain coffee in D_5 and this percentage is $1000/(1000 + 100,000) = 0.99\%$ in D_6, both indicating a negative association. On the other hand, 90.9% of transactions in D_5 (i.e., $mc/(mc$ $mc)$ $1000/(1000$ $100)$) and 9% in D_6 (i.e., $1000/(1000$ $10)$) containing coffee contain milk as well, which indicates a positive association between milk and coffee. These draw very different conclusions.

For such "balanced" skewness, it could be fair to treat it as neutral, as *Kulc* does, and in the meantime indicate its skewness using the *imbalance ratio (IR)*. According to Eq. (6.13), for D_4 we have $IR(m, c) = 0$, a perfectly balanced case; for D_5, $IR(m, c) = 0.89$, a rather imbalanced case; whereas for D_6, $IR(m, c)$ 0.99, a very skewed case. Therefore, the two measures, *Kulc* and *IR*, work together, presenting a clear picture for all three data sets, D_4 through D_6.

.

➢ Pattern Mining: A Road Map

Chapter 6 introduced the basic concepts, techniques, and applications of frequent pat- tern 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 stud- ies 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.

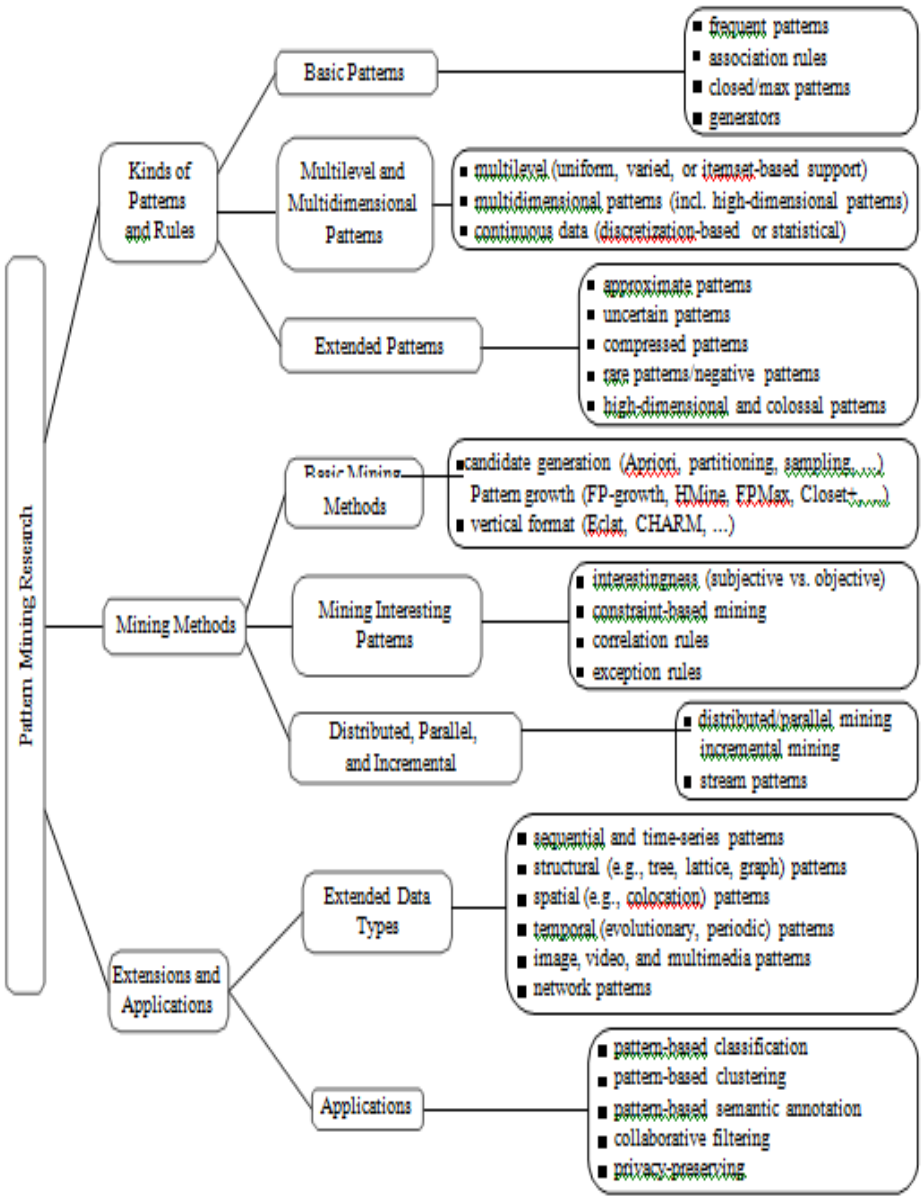


Figure 7.1 A general road map on pattern mining research.

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 sup- port threshold. A pattern p is a **closed pattern** if there is no superpattern p^r 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 impor- tance, 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:

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

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

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:

$$age(X, \text{“}20 \ldots 29\text{”}) \wedge income(X, \text{“}52K \ldots 58K\text{”}) \Rightarrow buys(X, \text{“}iPad\text{”}). \quad (7.3)$$

- **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.

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 ware- house 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.

➢ **Pattern Mining in Multilevel, Multidimensional Space**

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.

- **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 trans- actional 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 con- cept 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
...	...

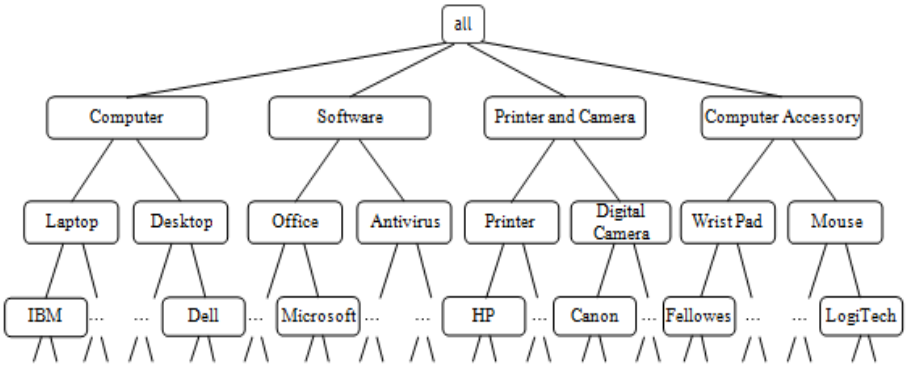


Figure 7.2 Concept hierarchy for *AllElectronics* computer items.

Concept hierarchies for nominal attributes are often implicit within the database schema, in which case they may be automatically generated using methods such. 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 calcu- lation 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 one minimum support threshold.

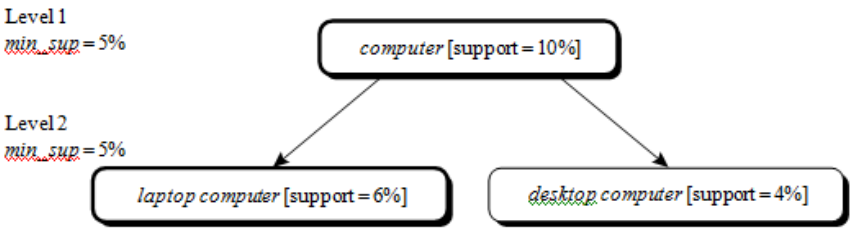


Figure 7.3 Multilevel mining with uniform support.

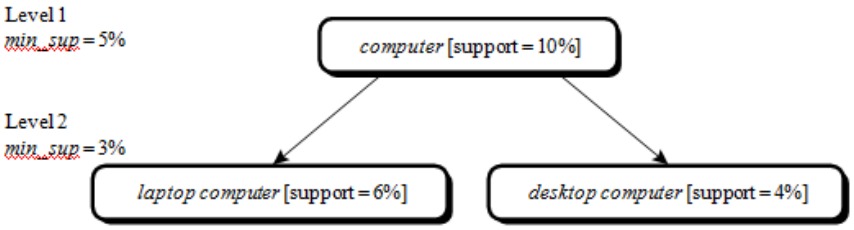


Figure 7.4 Multilevel mining with reduced support.

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 sup- port): 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 thresh- olds, 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%] (7.4)

$$buys(X, \text{“}Dell\ laptop\ computer\text{”}) \Rightarrow buys(X, \text{“}HP\ printer\text{”})$$

[*support* =2%, *confidence* =72%] (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 infor-mation, then it should be removed. Let’s look at how this may be determined. A rule *R*1 is an **ancestor** of a rule *R*2, if *R*1 can be obtained by replacing the items in *R*2 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 defini- tion, 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% $\frac{1}{4}$). 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).

➢ 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

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

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:

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

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.

➢ 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

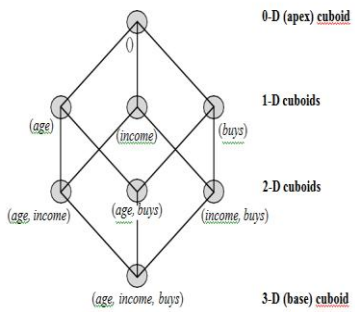


Figure 7.5 Lattice of cuboids, making up a 3-D data cube. Each cuboid represents a different group-by.

The base cuboid contains the three predicates *age, income,* and *buys*. The exploration of this set should be terminated. This is because any more-specialized ver- sion 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 quantita- tive 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 pat- terns 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 thresh- old. 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 prun- ing 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:

$$sex = female \Rightarrow meanwage = \$7.90/hr \ (overall\ mean\ wage = \$9.02/hr). \quad (7.9)$$

This rule states that the average wage for females is only \$7.90/hr. This rule is (subjec- tively) 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:

$$\textit{population subset} \Rightarrow \textbf{mean } \textit{of values for the subset}, \qquad (7.10)$$

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).

➢ 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 neg- atively 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 occur- rence 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 con- ditions 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 explic- itly. 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 $sup(X \cup Y) \; sup(X) \times sup(Y)$, then X and Y are **strongly negatively correlated**, and the pattern $X \cup Y$ is a **strongly negatively correlated pattern**. Q

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, $sup(A \cup B) < sup(A) \times sup(B)$, 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, $sup(A \cup B) < sup(X) \times sup(Y)$, 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

$$sup(X \cup Y) \times sup(X \cup Y) < sup(X \cup \bar{Y}) \times sup(\bar{X} \cup Y).$$

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

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

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

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

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**. Q

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 ϵ 0.02. When there are 200 transactions in the database, we have $sup(A)$ $sup(\bar{B})$ $100/200$ $0.5 > 0.01\%$ and $(P(B|A)$ $P(\bar{A}|B))/2$ $(0.01$ $0.01)/2 < 0.02$; 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% 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 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.

➢ **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 metaruleguided 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 *succinct- ness*. 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.

➢ 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 deter- mining 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

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

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, "30..39"*)* ∧ *income(X,* "41*K* ..60*K*"*)* ⇒ *buys(X,* "*office software*"*).* (7.12)

"*How can metarules be used to guide the mining process?*" Let's examine this prob- lem 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

$P_1 \wedge P_2 \wedge \cdots \wedge P_l \Rightarrow Q_1 \wedge Q_2 \wedge \cdots \wedge Q_r,$ (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 metaruleguided mining.

➢ Constraint-Based Pattern Generation: PruningPattern 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) ≥ $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 sup- port, confidence, and correlation measures, were discussed in Chapter 6. Let's focus now on rule constraints.

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 pat- tern 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)* ≤ $100" of Example 7.8. Suppose we are using the Apriori framework, which explores itemsets of size *k* at the *k*th 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 algo- rithms 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)* \$50," "*count(I)* 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)* \$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., , , but not ,) and comparison (or containment) operators with equality (e.g., ,) are given.

The second category of constraints is **monotonic**. If the rule constraint in Example 7.8 were "*sum(I.price)* \$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 con- straint, so do all of its supersets. If a rule constraint obeys this property, it is **monotonic**. Similar rule monotonic constraints include "*min(I.price)* \$10," "*count(I)* 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)* \$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 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)* $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

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 con- straint (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) v*" and "*avg(S) v*," given in Table 7.2, there are many other convertible constraints such as "*variance(S) v*" "*standard deviation(S) v*," and so on.

Note that the previous discussion does not imply that every constraint is convertible. For example,

"*sum(S)* θv," where θ , 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 pat- tern 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 usu- ally 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.

➢ Classification Using Frequent Patterns

Frequent patterns show interesting relationships between attribute–value pairs that occur frequently in a given data set. For example, we may find that the attribute–value pairs *age youth* and *credit OK* occur in 20% of data tuples describing *AllElectronics* customers who buy a computer. We can think of each attribute–value pair as an *item*, so the search for these frequent patterns is known as *frequent pattern mining* or *frequent itemset mining*. In Chapters 6 and 7, we saw how **association rules** are derived from frequent patterns, where the associations are commonly used to analyze the purchas- ing patterns of customers in a store. Such analysis is useful in many decision-making processes such as product placement, catalog design, and cross-marketing.

In this section, we examine how frequent patterns can be used for classification. Section 9.4.1 explores **associative classification**, where association rules are generated from frequent patterns and used for classification. The general idea is that we can search for strong associations between frequent patterns (conjunctions of attribute–valuepairs) and class labels. Section 9.4.2 explores **discriminative frequent pattern–based classification**, where frequent patterns serve as combined features, which are considered in addition to single features when building a classification model. Because frequent patterns explore highly confident associations among multiple attributes, frequent pattern–based classification may overcome some constraints introduced by decision tree induction, which considers only one attribute at a time. Studies have shown many frequent pattern–based classification methods to have greater accuracy and scalability than some traditional classification methods such as C4.5.

Associative Classification

In this section, you will learn about associative classification. The methods discussed are CBA, CMAR, and CPAR.

Before we begin, however, let's look at association rule mining in general. Association rules are mined in a two-step process consisting of *frequent itemset mining* followed by *rule generation*. The first step searches for patterns of attribute–value pairs that occur repeatedly in a data set, where each attribute–value pair is considered an *item*. The resulting attribute–value pairs form *frequent itemsets* (also referred to as *frequent pat- terns*). The second step analyzes the frequent itemsets to generate association rules. All association rules must satisfy certain criteria regarding their "accuracy" (or *confidence*) and the proportion of the data set that they actually represent

(referred to as *support*). For example, the following is an association rule mined from a data set, *D*, shown with its confidence and support:

$$age = youth \wedge credit\_OK = yes \Rightarrow buys\ computer = yes \quad [support = 20\%, confidence = 93\%], \tag{9.21}$$

where represents a logical "AND." We will say more about confidence and support later.

More formally, let *D* be a data set of tuples. Each tuple in *D* is described by *n* attributes, A_1, A_2, . . . , A_n, and a class label attribute, A_{class}. All continuous attributes are discretized and treated as categorical (or nominal) attributes. An **item**, *p*, is an attribute–value pair of the form (A_i, v), where A_i is an attribute taking a value, *v*. A data tuple $\boldsymbol{X} = (x_1, x_2, \ldots, x_n)$ satisfies an item, $p = (A_i, v)$, if and only if $x_i = v$, where x_i is the value of the *i*th attribute of $\boldsymbol{X}$. Association rules can have any number of items in the rule antecedent (left side) and any number of items in the rule consequent (right side). However, when mining association rules for use in classification, we are only interested in association rules of the form $p_1 \wedge p_2 \wedge \ldots p_l \Rightarrow A_{class} = C$, where the rule antecedent is a conjunction of items, $p_1, p_2, \ldots, p_l$ ($l \leq n$), associated with a class label, *C*. For a given rule, *R*, the percentage of tuples in *D* satisfying the rule antecedent that also have the class label *C* is called the **confidence** of *R*.

From a classification point of view, this is akin to rule accuracy. For example, a confidence of 93% for Rule (9.21) means that 93% of the customers in *D* who are young and have an OK credit rating belong to the class *buys computer = yes*. The percentage of tuples in *D* satisfying the rule antecedent and having class label *C* is called the **support** of *R*. A support of 20% for Rule (9.21) means that 20% of the customers in *D* are young, have an OK credit rating, and belong to the class *buys computer = yes*.

In general, associative classification consists of the following steps:

1. Mine the data for frequent itemsets, that is, find commonly occurring attribute–value pairs in the data.
2. Analyze the frequent itemsets to generate association rules per class, which satisfy confidence and support criteria.
3. Organize the rules to form a rule-based classifier.

Methods of associative classification differ primarily in the approach used for frequent itemset mining and in how the derived rules are analyzed and used for classification. We now look at some of the various methods for associative classification.

One of the earliest and simplest algorithms for associative classification is **CBA** (Classification Based on Associations). CBA uses an iterative approach to frequent itemset mining, similar to that described for Apriori in Section 6.2.1, where multiple passes are made over the data and the derived frequent itemsets are used to generate and test longer itemsets. In general, the number of passes made is equal to the length of the longest rule found. The complete set of rules satisfying minimum confidence and minimum support thresholds are found and then analyzed for inclusion in the classifier. CBA uses a heuristic method to construct the classifier, where the rules are ordered according to decreasing precedence based on their confidence and support. If a set of rules has the same antecedent, then the rule with the highest confidence is selected to represent the set. When classifying a new tuple, the first rule satisfying the tuple is used to classify it. The classifier also contains a default rule, having lowest precedence, which specifies a default class for any new tuple that is not satisfied by any other rule in the classifier. In this way, the set of rules making up the classifier form a *decision list*. In

general, CBA was empirically found to be more accurate than C4.5 on a good number of data sets.

CMAR (Classification based on Multiple Association Rules) differs from CBA in its strategy for frequent itemset mining and its construction of the classifier. It also employs several rule pruning strategies with the help of a tree structure for efficient storage and retrieval of rules. CMAR adopts a variant of the *FP-growth* algorithm to find the complete set of rules satisfying the minimum confidence and minimum support thresholds. FP-growth was described in Section 6.2.4. FP-growth uses a tree structure, called an *FP-tree*, to register all the frequent itemset information contained in the given data set, *D*. This requires only two scans of *D*. The frequent itemsets are then mined from the FP-tree. CMAR uses an enhanced FP-tree that maintains the distribution of class labels among tuples satisfying each frequent itemset. In this way, it is able to combine rule generation together with frequent itemset mining in a single step.

CMAR employs another tree structure to store and retrieve rules efficiently and to prune rules based on confidence, correlation, and database coverage. Rule pruning strategies are triggered whenever a rule is inserted into the tree. For example, given two rules, $R1$ and $R2$, if the antecedent of $R1$ is more general than that of $R2$ and $conf(R1) \geq conf(R2)$, then $R2$ is pruned. The rationale is that highly specialized rules with low confidence can be pruned if a more generalized version with higher confidence exists. CMAR also prunes rules for which the rule antecedent and class are not positively correlated, based on an χ^2 test of statistical significance.

"*If more than one rule applies, which one do we use?*" As a classifier, CMAR operates differently than CBA. Suppose that we are given a tuple $\boldsymbol{X}$ to classify and that only one rule satisfies or matches $\boldsymbol{X}$.[4] This case is trivial—we simply assign the rule's class label. Suppose, instead, that more than one rule satisfies $\boldsymbol{X}$. These rules form a set, *S*. Which rule would we use to determine the class label of $\boldsymbol{X}$? CBA would assign the class label of the most confident rule among the rule set, *S*. CMAR instead considers multiple rules when making its class prediction. It divides the rules into groups according to class labels. All rules within a group share the same class label and each group has a distinct class label.

CMAR uses a weighted χ^2 measure to find the "strongest" group of rules, based on the statistical correlation of rules within a group. It then assigns $\boldsymbol{X}$ the class label of the strongest group. In this way it considers multiple rules, rather than a single rule with highest confidence, when predicting the class label of a new tuple. In experiments, CMAR had slightly higher average accuracy in comparison with CBA. Its runtime, scalability, and use of memory were found to be more efficient.

"*Is there a way to cut down on the number of rules generated?*" CBA and CMAR adopt methods of frequent itemset mining to generate *candidate* association rules, which include all conjunctions of attribute–value pairs (items) satisfying minimum support. These rules are then examined, and a subset is chosen to represent the classifier. However, such methods generate quite a large number of rules. CPAR (Classification based on Predictive Association Rules) takes a different approach to rule generation, based on a rule generation algorithm for classification known as FOIL (Section 8.4.3). FOIL builds rules to distinguish positive tuples (e.g., *buys computer = yes*) from negative tuples (e.g., *buys computer = no*). For multiclass problems, FOIL is applied to each class. That is, for a class, *C*, all tuples of class *C* are considered positive tuples, while the rest are considered negative tuples. Rules are generated to distinguish *C* tuples from all others. Each time a rule is generated, the positive samples it satisfies (or *covers*) are removed until all the positive tuples in the data set are covered. In this way, fewer rules are generated. CPAR relaxes this step by allowing the covered tuples to

remain under consideration, but reducing their weight. The process is repeated for each class. The resulting rules are merged to form the classifier rule set.

During classification, CPAR employs a somewhat different multiple rule strategy than CMAR. If more than one rule satisfies a new tuple, $\boldsymbol{X}$, the rules are divided into groups according to class, similar to CMAR. However, CPAR uses the best k rules of each group to predict the class label of $\boldsymbol{X}$, based on expected accuracy. By considering the best k rules rather than all of a group's rules, it avoids the influence of lower-ranked rules. CPAR's accuracy on numerous data sets was shown to be close to that of CMAR. However, since CPAR generates far fewer rules than CMAR, it shows much better efficiency with large sets of training data.

In summary, associative classification offers an alternative classification scheme by building rules based on conjunctions of attribute–value pairs that occur frequently in data.