determines new candidate itemsets only immediately prior to each complete database scan. The technique is dynamic in that it estimates the support of all of the itemsets that have been counted so far, adding new candidate itemsets if all of their subsets are estimated to be frequent. The resulting algorithm requires fewer database scans than Apriori.

Other variations involving the mining of multilevel and multidimensional association rules are discussed in the rest of this chapter. The mining of associations related to spatial data, time-series data, and multimedia data are discussed in Chapter 9.

6.2.4 Mining Frequent Itemsets without Candidate Generation

As we have seen, in many cases the Apriori candidate generate-and-test method reduces the size of candidate sets significantly and leads to good performance gain. However, it may suffer from two nontrivial costs.

- **It may need to generate a huge number of candidate sets.** For example, if there are 10^4 frequent 1-itemsets, the Apriori algorithm will need to generate more than 10^7 candidate 2-itemsets and accumulate and test their occurrence frequencies. Moreover, to discover a frequent pattern of size 100, such as $\{a_1, \ldots, a_{100}\}$, it must generate more than $2^{100} \approx 10^{30}$ candidates in total.
- **It may need to repeatedly scan the database and check a large set of candidates by pattern matching.** This is especially the case for mining long patterns.

"Can we design a method that mines the complete set of frequent itemsets without candidate generation?" An interesting method in this attempt is called **frequent-pattern growth**, or simply **FP-growth**, which adopts a *divide-and-conquer* strategy as follows: compress the database representing frequent items into a **frequent-pattern tree**, or **FP-tree**, but retain the itemset association information, and then divide such a compressed database into a set of *conditional databases* (a special kind of projected database), each associated with one frequent item, and mine each such database separately. Let's look at an example.

Example 6.3 We reexamine the mining of transaction database, D, of Figure 6.2 in Example 6.1 using the frequent-pattern growth approach.

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 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. For example, the scan of the first transaction, "T100: I1, I2, I5", which contains three items (I2, I1, I5) in L order, leads to the construction of the first branch of the tree with three nodes: ⟨(I2: 1), (I1: 1), I5: 1)⟩, where I2 is linked as a child of the root, I1 is linked to I2, and I5 is linked to I2. The second transaction, T200, contains the items I2 and I4 in L order, which would result in a branch where I2 is linked to the root and I4 is linked to I2. However, this branch would share a common **prefix,** ⟨I2⟩, with the existing path for T100. Therefore, we instead increment the count of the I2 node by 1, and create a new node, (I4: 1), which is linked as a child of (I2: 2). In general, when considering the branch to be added for a transaction, the count of each node along a common prefix is incremented by 1, and nodes for the items following the prefix are created and linked accordingly.

To facilitate tree traversal, an item header table is built so that each item points to its occurrences in the tree via a chain of **node-links.** The tree obtained after scanning all of the transactions is shown in Figure 6.8 with the associated node-links. Therefore, the problem of mining frequent patterns in databases is transformed to that of mining the FP-tree.

The mining of the FP-tree proceeds as follows. Start from each frequent length-1 pattern (as an initial **suffix pattern**), construct its **conditional pattern base** (a "subdatabase" 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 such a tree. The pattern growth is achieved by the concatenation of the suffix pattern with the frequent patterns generated from a conditional FP-tree.

Mining of the FP-tree is summarized in Table 6.1 and detailed as follows. Let's first consider I5 which is the last item in L, rather than the first. The reasoning behind this will become apparent as we explain the FP-tree mining process. I5

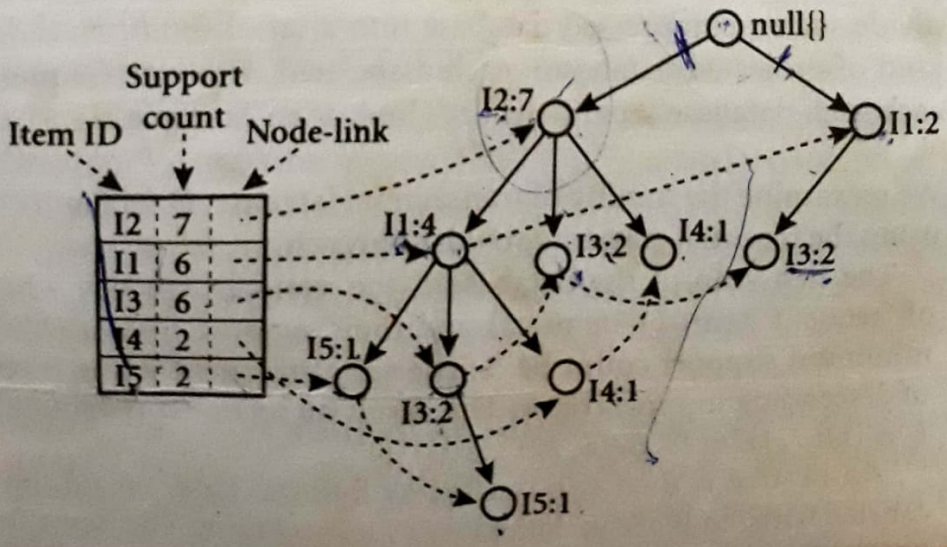

Figure 6.8 An FP-tree that registers compressed, frequent pattern inform

Table 6.1 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: 2, I2 I1 I3: 2
I1	{(I2: 4)}	⟨I2: 4⟩	I2 I1: 4

occurs in two branches of the FP-tree of Figure 6.8. (The occurrences of I5 can easily be found by following its chain of node-links.) The paths formed by these branches are ⟨(I2 I1 I5: 1)⟩ and ⟨(I2 I1 I3 I5: 1)⟩. Therefore, considering I5 as a suffix, its corresponding two prefix paths are ⟨(I2 I1: 1)⟩ and ⟨(I2 I1 I3: 1)⟩, which form its conditional pattern base. Its conditional FP-tree contains only a single path, ⟨I2: 2, I1: 2⟩; I3 is not included because its support count of 1 is less than the minimum support count. The single path generates all the combinations of frequent patterns: I2 I5: 2, I1 I5: 2, I2 I1 I5: 2.

For I4, its two prefix paths form the conditional pattern base, {(I2 I1: 1), (I2: 1)}, which generates a single-node conditional FP-tree ⟨I2: 2⟩ and derives one frequent pattern, I2 I4: 2. Notice that although I5 follows I4 in the first branch, there is no need to include I5 in the analysis here since any frequent pattern involving I5 has been analyzed in the examination of I5. This is the reason that we started processing at the end of L, rather than at the front.

Similar to the above analysis, I3's conditional pattern base is {(I2 I1: 2), (I2: 2), (I1: 2)}. Its conditional FP-tree has two branches, ⟨I2: 4, I1: 2⟩ and ⟨I1: 2⟩, as shown in Figure 6.9, which generates the set of patterns: {I2 I3: 4, I1 I3: 2, I2 I1 I3: 2}. Finally, I1's conditional pattern base is {(I2: 4)}, whose FP-tree contains only one node ⟨I2: 4⟩, which generates one frequent pattern, I2 I1: 4. This mining process is summarized in Figure 6.10. ■

The FP-growth method transforms the problem of finding long frequent patterns to looking for shorter ones recursively and then concatenating the suffix.

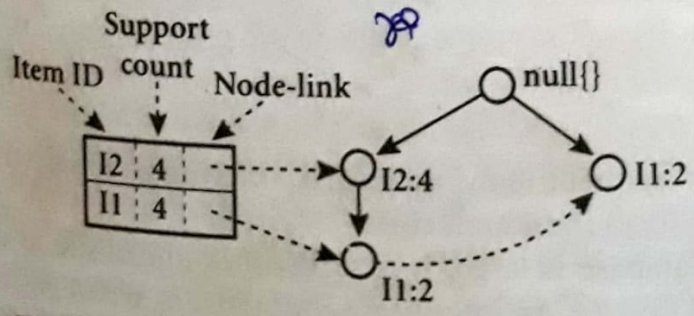

Figure 6.9 The conditional FP-tree associated with the conditional node I3.

Algorithm: FP_growth. Mine frequent patterns using an FP-tree by pattern fragment growth.

Input: A transaction database, D; minimum support threshold, *min_sup*.

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 the set of frequent items F and their supports. Sort F in support 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 $N.item\text{-}name = p.item\text{-}name$, 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. Mining of an FP-tree is performed 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 = minimum support of nodes in β;
(4)  else for each a_i in the header of Tree {
(5)      generate pattern β = a_i ∪ α with support = a_i.support;
(6)      construct β's conditional pattern base and then β's conditional
             FP_tree Tree_β;
(7)      if Tree_β ≠ ∅ then
(8)          call F_growth(Tree_β, β); }
```

Figure 6.10 The FP-growth algorithm for discovering frequent itemsets without candidate generation.

It uses the least frequent items as a suffix, offering good selectivity. The method substantially reduces the search costs.

When the database is large, it is sometimes unrealistic to construct a main memory-based FP-tree. An interesting alternative is to first partition the database into a set of projected databases, and then construct an FP-tree and mine it in

each projected database. Such a process can be recursively applied to any projected database if its FP-tree still cannot fit in main memory.

A study on the performance of the FP-growth method shows that it is efficient and scalable for mining both long and short frequent patterns, and is about an order of magnitude faster than the Apriori algorithm. It is also faster than a Tree-Projection algorithm which projects a database into a tree of projected databases recursively.

prove the overall efficiency of this query answering technique.

6.3 Mining Multilevel Association Rules from Transaction Databases

In this section, you will learn methods for mining multilevel association rules, that is, rules involving items at different levels of abstraction. Methods for checking for redundant multilevel rules are also discussed.

6.3.1 Multilevel Association Rules

For many applications, it is difficult to find strong associations among data items at low or primitive levels of abstraction due to the sparsity of data in multidimensional space. Strong associations discovered at high concept levels may represent common sense knowledge. However, what may represent common sense to one user may seem novel to another. Therefore, data mining systems should provide capabilities to mine association rules at multiple levels of abstraction and traverse easily among different abstraction spaces.

Let's examine the following example.

Table 6.2 Task-relevant data, D.

TID	Items purchased
T1	IBM desktop computer, Sony b/w printer
T2	Microsoft educational software, Microsoft financial management software
T3	Logitech mouse computer accessory, Ergoway wrist pad computer accessory
T4	IBM desktop computer, Microsoft financial management software
T5	IBM desktop computer
⋮	⋮

Example 6.5 Suppose we are given the task-relevant set of transactional data in Table 6.2 for sales at the computer department of an *AllElectronics* branch, showing the items purchased for each transaction TID. The concept hierarchy for the items is shown in Figure 6.11. A concept hierarchy defines a sequence of mappings from a set of low-level concepts to higher-level, more general concepts. Data can be generalized by replacing low-level concepts within the data by their higher-level concepts, or *ancestors*, from a concept hierarchy.[7] The concept hierarchy of Figure 6.11 has four levels, referred to as levels 0, 1, 2, and 3. By convention, levels within a concept hierarchy are numbered from top to bottom, starting with level 0 at the root node for **all** (the most general abstraction level). Here, level 1 includes *computer, software, printer,* and *computer accessory*, level 2 includes *desktop computer, laptop computer, educational software, financial management software, . . .*, and level 3 includes *IBM desktop computer, . . . , Microsoft educational software*, and so on. Level 3 represents the most specific abstraction level of this hierarchy. Concept hierarchies may be specified by users familiar with the data or may exist implicitly in the data.

The items in Table 6.2 are at the lowest level of the concept hierarchy of Figure 6.11. It is difficult to find interesting purchase patterns at such raw or primitive-level data. For instance, if "*IBM desktop computer*" or "*Sony b/w (black and white) printer*" each occurs in a very small fraction of the transactions, then it may be difficult to find strong associations involving such items. Few people may buy such items together, making it unlikely that the itemset "*{IBM desktop computer, Sony b/w printer}*" will satisfy minimum support. However, consider the generalization of "*Sony b/w printer*" to "*b/w printer*". One would expect that it is easier to find strong associations between "*IBM desktop computer*" and "*b/w printer*" than between "*IBM desktop computer*" and "*Sony b/w printer*". Similarly, many people may purchase "*computer*" and "*printer*" together, rather than specifically

[7] Concept hierarchies were described in detail in Chapters 2 and 4. In order to make the chapters of this book as self-contained as possible, we offer their definition again here. Generalization was described in Chapter 5.

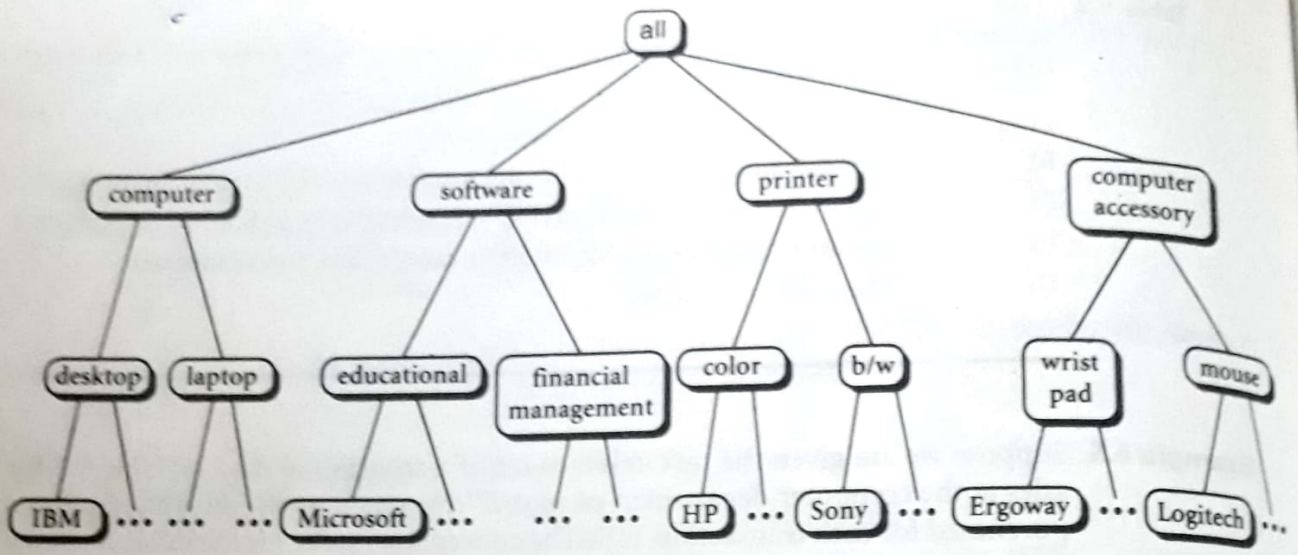

Figure 6.11 A concept hierarchy for *AllElectronics* computer items.

purchasing *"IBM desktop computer"* and *"Sony b/w printer"* together. In other words, itemsets containing generalized items, such as *"{IBM desktop computer, b/w printer}"* and *"{computer, printer}"* are more likely to have minimum support than itemsets containing only primitive-level data, such as *"{IBM desktop computer, Sony b/w printer}"*. Hence, it is easier to find interesting associations among items at *multiple* concept levels, rather than only among low-level data.

Rules generated from association rule mining with concept hierarchies are called **multiple-level** or **multilevel association rules**, since they consider more than one concept level.

6.3.2 Approaches to Mining Multilevel Association Rules

"How can we mine multilevel association rules efficiently using concept hierarchies?" Let's look at some approaches based on a support-confidence framework. In general, a top-down strategy is employed, where counts are accumulated for the calculation of frequent itemsets at each concept level, starting at the concept level 1 and working towards the lower, more specific concept levels, until no more frequent itemsets can be found. That is, once all frequent itemsets at concept level 1 are found, then the frequent itemsets at level 2 are found, and so on. For each level, any algorithm for discovering frequent itemsets may be used, such as Apriori or its variations. A number of variations to this approach are described below and illustrated in Figures 6.12 to 6.16, where nodes indicate an item or itemset that

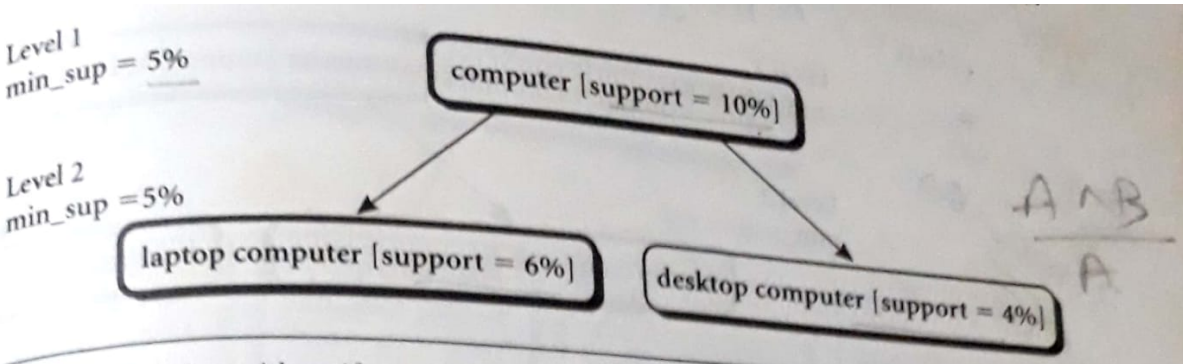

Figure 6.12 Multilevel mining with uniform support.

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 level of abstraction. For example, in Figure 6.12, a minimum support threshold of 5% is used throughout (e.g., for mining from "*computer*" down to "*laptop computer*"). Both "*computer*" and "*laptop computer*" are found to be frequent, while "*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. An optimization technique can be adopted, based on the knowledge that an ancestor is a superset of its descendents: the search avoids examining itemsets containing any item whose ancestors do not have minimum support.

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

- **Using reduced minimum support at lower levels** (referred to as **reduced support**): Each level of abstraction has its own minimum support threshold. The lower the abstraction level, the smaller the corresponding threshold. For example, in Figure 6.13, 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.

For mining multiple-level associations with *reduced support*, there are a number of alternative search strategies:

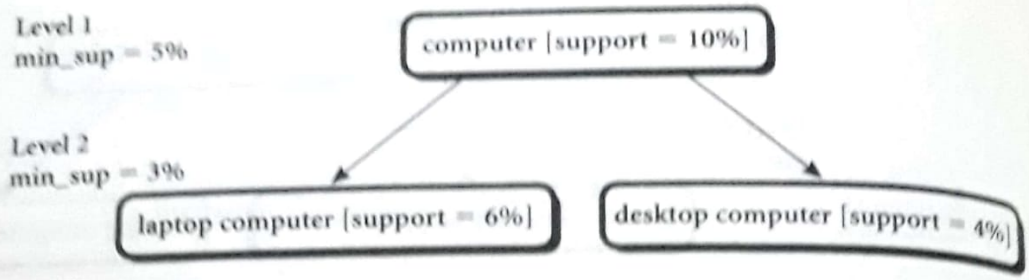

Figure 6.13 Multilevel mining with reduced support.

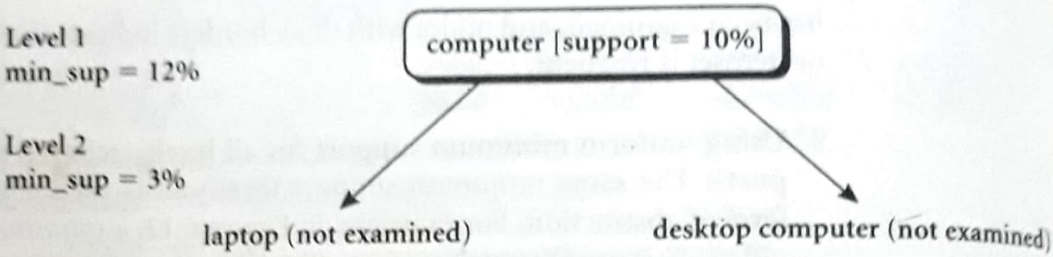

Figure 6.14 Multilevel mining with reduced support, using level-cross filtering by a single item.

- **Level-by-level independent:** This is a full-breadth search, where no background knowledge of frequent itemsets is used for pruning. Each node is examined, regardless of whether or not its parent node is found to be frequent.
- **Level-cross filtering by single item:** An item at the ith level is examined if and only if its parent node at the $(i-1)$th level is frequent. In other words, we investigate a more specific association from a more general one. If a node is frequent, its children will be examined; otherwise, its descendents are pruned from the search. For example, in Figure 6.14, the descendent nodes of "*computer*" (i.e., "*laptop computer*" and "*desktop computer*") are not examined, since "*computer*" is not frequent.
- **Level-cross filtering by k-itemset:** A k-itemset at the ith level is examined if and only if its corresponding parent k-itemset at the $(i-1)$th level is frequent. For example, in Figure 6.15, the 2-itemset "*{computer, printer}*" is frequent, therefore the nodes "*{laptop computer, b/w printer}*", "*{laptop computer, color printer}*", "*{desktop computer, b/w printer}*", and "*{desktop computer, color printer}*" are examined.

"*How do these methods compare?*" The *level-by-level independent* strategy is very relaxed in that it may lead to examining numerous infrequent items at low levels, finding associations between items of little importance. For example, if "*computer*

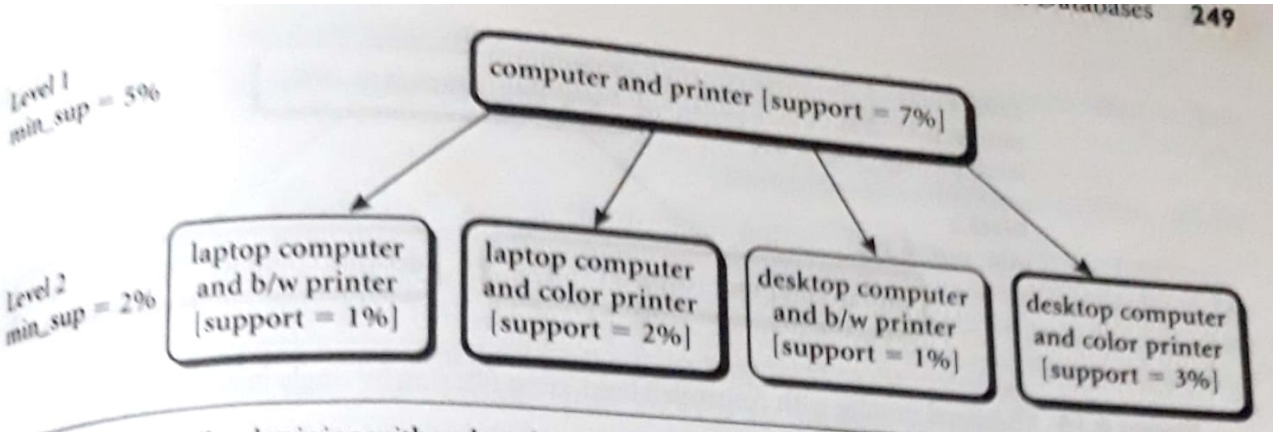

Figure 6.15 Multilevel mining with reduced support, using level-cross filtering by a k-itemset. Here, $k = 2$.

furniture" is rarely purchased, it may not be beneficial to examine whether the more specific "*computer chair*" is associated with "*laptop*". However, if "*computer accessories*" are sold frequently, it may be beneficial to see whether there is an associated purchase pattern between "*laptop*" and "*mouse*".

The *level-cross filtering by k-itemset* strategy allows the mining system to examine only the children of frequent k-itemsets. This restriction is very strong in that there usually are not many k-itemsets that, when combined, are also frequent (especially when $k > 2$). Hence, many valuable patterns may be filtered out using this approach.

The *level-cross filtering by single item* strategy represents a compromise between the two extremes. However, this method may miss associations between low-level items that are frequent based on a reduced minimum support, but whose ancestors do not satisfy minimum support (since the support thresholds at each level can be different). For example, if "*color monitor*" occurring at concept level i is frequent based on the minimum support threshold of level i, but its parent "*monitor*" at level $(i - 1)$ is not frequent according to the minimum support threshold of level $(i - 1)$, then frequent associations such as "*desktop computer* $\Rightarrow$ *color monitor*" will be missed.

A modified version of the *level-cross filtering by single item* strategy, known as the **controlled level-cross filtering by single item** strategy, addresses the above concern as follows. A threshold, called the **level passage threshold**, can be set up for "passing down" relatively frequent items (called **subfrequent items**) to lower levels. In other words, this method allows the children of items that do not satisfy the minimum support threshold to be examined if these items satisfy the level passage threshold. Each concept level can have its own level passage threshold. The level passage threshold for a given level is typically set to a value between the minimum support threshold of the next lower level and the minimum support threshold of the given level. Users may choose to "slide down" or lower the level passage threshold at high concept levels to allow the descendents of the subfrequent items at lower levels to be examined. Sliding the level passage

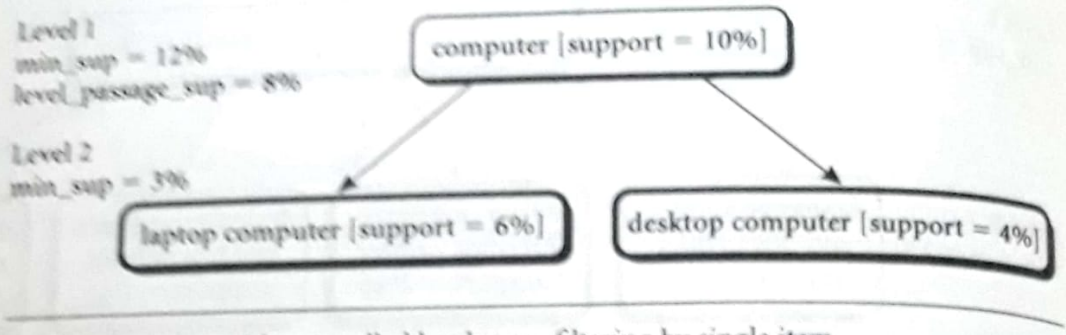

Figure 6.16 Multilevel mining with controlled level-cross filtering by single item.

threshold down to the minimum support threshold of the lowest level would allow the descendents of all of the items to be examined. For example, in Figure 6.16, setting the level passage threshold (*level_passage_sup*) of level 1 to 8% allows the nodes *"laptop computer"* and *"desktop computer"* at level 2 to be examined and found frequent, even though their parent node, *"computer"*, is not frequent. By adding this mechanism, users have the flexibility to further control the mining process at multiple abstraction levels, as well as reduce the number of meaningless associations that would otherwise be examined and generated.

So far, our discussion has focused on finding frequent itemsets where all items within the itemset must belong to the same concept level. This may result in rules such as *"computer ⇒ printer"* (where *"computer"* and *"printer"* are both at concept level 1) and *"desktop computer ⇒ b/w printer"* (where *"desktop computer"* and *"b/w printer"* are both at level 2 of the given concept hierarchy). Suppose, instead, that we would like to find rules that *cross concept level boundaries*, such as *"computer ⇒ b/w printer"*, where items within the rule are not required to belong to the same concept level. These rules are called **cross-level association rules**.

"How can cross-level associations be mined?" If mining associations from concepts levels i and j, where level j is more specific (i.e., at a lower abstraction level) than i, then the reduced minimum support threshold of level j should be used overall so that items from level j can be included in the analysis.

6.3.3 Checking for Redundant Multilevel Association Rules

Concept hierarchies are useful in data mining since they permit the discovery of knowledge at different levels of abstraction, such as multilevel association rules. However, when multilevel association rules are mined, some of the rules found will be redundant due to "ancestor" relationships between items. For example, consider the following rules where *"desktop computer"* is an ancestor of *"IBM desktop computer"* based on the concept hierarchy of Figure 6.11.

$$\textit{desktop computer} \Rightarrow \textit{b/w printer} \quad [\textit{support} = 8\%, \textit{confidence} = 70\%] \quad (6.9)$$

$$\textit{IBM desktop computer} \Rightarrow \textit{b/w printer} \quad [\textit{support} = 2\%, \textit{confidence} = 72\%] \quad (6.10)$$

"*If Rules (6.9) and (6.10) are both mined, then how useful is the latter rule?*" you may wonder. "*Does it really provide any novel information?*" If the latter, less general rule does not provide new information, it should be removed. Let's have a look at how this may be determined. A rule, $R1$, is an **ancestor** of a rule, $R2$, if $R1$ can be obtained by replacing the items in $R2$ by their ancestors in a concept hierarchy. For example, Rule (6.9) is an ancestor of Rule (6.10) since "*desktop computer*" is an ancestor of "*IBM desktop computer*". Based on this definition, a rule can be considered redundant if its support and confidence are close to their "expected" values, based on an ancestor of the rule. As an illustration, suppose that Rule (6.9) has a 70% confidence and 8% support, and that about one quarter of all "*desktop computer*" sales are for "*IBM desktop computers*". One may expect Rule (6.10) to have a confidence of around 70% (since all data samples of "*IBM desktop computer*" are also samples of "*desktop computer*") and a support of around 2% (i.e., $8\% \times \frac{1}{4}$). If this is indeed the case, then Rule (6.10) is not interesting since it does not offer any additional information and is less general than Rule (6.9).

6.4 Mining Multidimensional Association Rules from Relational Databases and Data Warehouses

In this section, you will learn methods for mining multidimensional association rules, that is, rules involving more than one dimension or predicate (e.g., rules relating what a customer *buys* as well as the customer's *age*). These methods can be organized according to their treatment of quantitative attributes.

6.4.1 Multidimensional Association Rules

So far in this chapter, 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 "*IBM desktop computer* $\Rightarrow$ *Sony b/w printer*", which can also be written as

$$\textit{buys}(X, \text{"}\textit{IBM desktop computer}\text{"}) \Rightarrow \textit{buys}(X, \text{"}\textit{Sony b/w printer}\text{"}) \quad (6.11)$$

where X is a variable representing customers who purchased items in *AllElectronics* transactions. 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 (6.11) as a **single-dimensional** or **intradimension association rule** since it contains a single distinct predicate (e.g., *buys*) with multiple occurrences (i.e., the

predicate occurs more than once within the rule). As we have seen in the previous sections of this chapter, such rules are commonly mined from transactional data.

Suppose, however, that rather than using a transactional database, sales and related information are stored in a relational database or data warehouse. Such data stores are multidimensional, by definition. For instance, in addition to keeping track of the items purchased in sales transactions, a relational database may record other attributes associated with the items, such as the quantity purchased or the price, or the branch location of the sale. Additional relational information regarding the customers who purchased the items, such as customer age, occupation, credit rating, income, and address, may also be stored. Considering each database attribute or warehouse dimension as a predicate, it can therefore be interesting to mine association rules containing *multiple* predicates, such as

$$age(X, \text{"20} \ldots \text{29"}) \wedge occupation(X, \text{"}student\text{"}) \Rightarrow buys(X, \text{"}laptop\text{"}) \quad (6.12)$$

Association rules that involve two or more dimensions or predicates can be referred to as **multidimensional association rules**. Rule (6.12) 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 **interdimension association rules**. We may also be interested in mining multidimensional association rules with repeated predicates, which contain multiple occurrences of some predicates. These rules are called **hybrid-dimension association rules**. An example of such a rule is the following, where the predicate *buys* is repeated:

$$age(X, \text{"20} \ldots \text{29"}) \wedge buys(X, \text{"}laptop\text{"}) \Rightarrow buys(X, \text{"}b/w\ printer\text{"}) \quad (6.13)$$

Note that database attributes can be categorical or quantitative. **Categorical** attributes have a finite number of possible values, with no ordering among the values (e.g., *occupation*, *brand*, *color*). Categorical attributes are also called **nominal** attributes, since their values are "names of things." **Quantitative** attributes are numeric and have an implicit ordering among values (e.g., *age*, *income*, *price*). Techniques for mining multidimensional association rules can be categorized according to three basic approaches regarding the treatment of quantitative attributes.

In the first approach, *quantitative attributes are discretized using predefined concept hierarchies*. This discretization occurs prior to mining. For instance, a concept hierarchy for *income* may be used to replace the original numeric values of this attribute by ranges, such as "0 . . . 20K", "21K . . . 30K", "31K . . . 40K", and so on. Here, discretization is *static* and predetermined. The discretized numeric attributes, with their range values, can then be treated as categorical attributes (where each range is considered a category). We refer to this as **mining multidimensional association rules using static discretization of quantitative attributes**.

In the second approach, *quantitative attributes are discretized into "bins" based on the distribution of the data*. These bins may be further combined during the

mining process. The discretization process is *dynamic* and established so as to satisfy some mining criteria, such as maximizing the confidence of the rules mined. Because this strategy treats the numeric attribute values as quantities rather than as predefined ranges or categories, association rules mined from this approach are also referred to as **quantitative association rules**.

In the third approach, *quantitative attributes are discretized so as to capture the semantic meaning of such interval data*. This dynamic discretization procedure considers the distance between data points. Hence, such quantitative association rules are also referred to as **distance-based association rules**.

Let's study each of these approaches for mining multidimensional association rules. For simplicity, we confine our discussion to interdimension 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 (6.12) is a 3-predicate set. Similar to the notation used for itemsets, we use the notation L_k to refer to the set of frequent k-predicate sets.

6.4.2 Mining Multidimensional Association Rules Using Static Discretization of Quantitative Attributes

Quantitative attributes, in this case, are discretized prior to mining using predefined concept hierarchies, where numeric values are replaced by ranges. Categorical attributes may also be generalized to higher conceptual levels if desired. If the resulting task-relevant data are stored in a relational table, then the Apriori algorithm requires just a slight modification so as to find all frequent predicate sets rather than frequent itemsets (i.e., by searching through all of the relevant attributes, instead of searching only one attribute, like *buys*). Finding all frequent k-predicate sets will require k or $k + 1$ scans of the table. Other strategies, such as hashing, partitioning, and sampling may be employed to improve the performance.

Alternatively, the transformed task-relevant data may be stored in a *data cube*. Data cubes are well suited for the mining of multidimensional association rules, since they are multidimensional by definition. Data cubes and their computation were discussed in detail in Chapter 2. To review, a data cube consists of a lattice of cuboids that are multidimensional data structures. These structures can hold the given task-relevant data, as well as aggregate, group-by information. Figure 6.17 shows the lattice of cuboids defining a data cube for the dimensions *age*, *income*, and *buys*. The cells of an n-dimensional cuboid are used to store the 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*; the 0-D (apex) cuboid contains the total number of transactions in the task-relevant data, and so on.

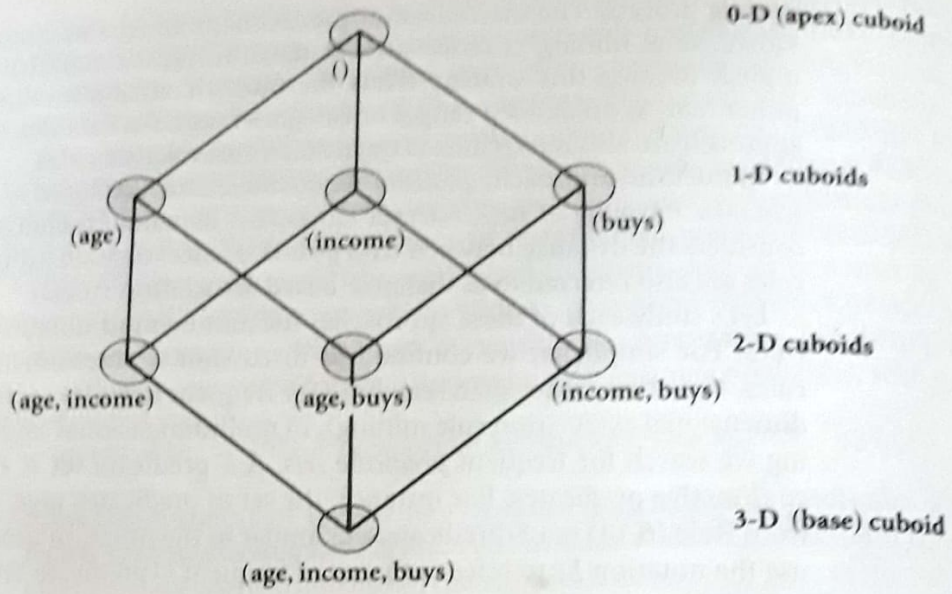

Figure 6.17 Lattice of cuboids, making up a 3-D data cube. Each cuboid represents a different group-by. The base cuboid contains the three predicates *age*, *income*, and *buys*.

Due to the ever-increasing use of data warehousing and OLAP technology, it is possible that a data cube containing the dimensions of interest to the user may already exist, fully materialized. "*If this is the case, how can we go about finding the frequent predicate sets?*" A strategy similar to that employed in Apriori can be used, based on prior knowledge that *every subset of a frequent predicate set must also be frequent*. This property can be used to reduce the number of candidate predicate sets generated.

In cases where no relevant data cube exists for the mining task, one must be created. Chapter 2 describes algorithms for fast, efficient computation of data cubes. These can be modified to search for frequent itemsets during cube construction.

6.4.3 Mining Quantitative Association Rules

Quantitative association rules are multidimensional association rules in which the numeric attributes are *dynamically* discretized during the mining process so as to satisfy some mining criteria, such as maximizing the confidence or compactness of the rules mined. In this section, we will focus specifically on how to mine quantitative association rules having two quantitative attributes on the left-hand side of the rule, and one categorical attribute on the right-hand side of the rule, for example,

$$A_{quan1} \wedge A_{quan2} \Rightarrow A_{cat}$$

where A_{quan1} and A_{quan2} are tests on quantitative attribute ranges (where the ranges are dynamically determined), and A_{cat} tests a categorical attribute from the task-relevant data. Such rules have been referred to as **two-dimensional quantitative association rules**, since they contain two quantitative dimensions. For instance, suppose you are curious about the association relationship between pairs of quantitative attributes, like customer age and income, and the type of television that customers like to buy. An example of such a 2-D quantitative association rule is

$$age(X, \text{"30} \ldots \text{39"}) \wedge income(X, \text{"42K} \ldots \text{48K"}) \Rightarrow buys(X, \text{"high resolution TV"}) \quad (6.14)$$

"How can we find such rules?" Let's look at an approach used in a system called **ARCS** (Association Rule Clustering System), which borrows ideas from image processing. Essentially, this approach maps pairs of quantitative attributes onto a 2-D grid for tuples satisfying a given categorical attribute condition. The grid is then searched for clusters of points, from which the association rules are generated. The following steps are involved in ARCS:

Binning: Quantitative attributes can have a very wide range of values defining their domain. Just think about how big a 2-D grid would be if we plotted *age* and *income* as axes, where each possible value of *age* was assigned a unique position on one axis, and similarly, each possible value of *income* was assigned a unique position on the other axis! To keep grids down to a manageable size, we instead partition the ranges of quantitative attributes into intervals. These intervals are dynamic in that they may later be further combined during the mining process. The partitioning process is referred to as **binning**, that is, where the intervals are considered "bins." Three common binning strategies are

- **Equiwidth binning**, where the interval size of each bin is the same,
- **Equidepth binning**, where each bin has approximately the same number of tuples assigned to it, and
- **Homogeneity-based binning**, where bin size is determined so that the tuples in each bin are uniformly distributed.

ARCS uses equiwidth binning, where the bin size for each quantitative attribute is input by the user. A 2-D array for each possible bin combination involving both quantitative attributes is created. Each array cell holds the corresponding count distribution for each possible class of the categorical attribute of the rule right-hand side. By creating this data structure, the task-relevant data need only be scanned once. The same 2-D array can be used to generate rules for any value of the categorical attribute, based on the same two quantitative attributes. Binning is also discussed in Chapter 3.

Finding frequent predicate sets: Once the 2-D array containing the count distribution for each category is set up, this can be scanned in order to find the

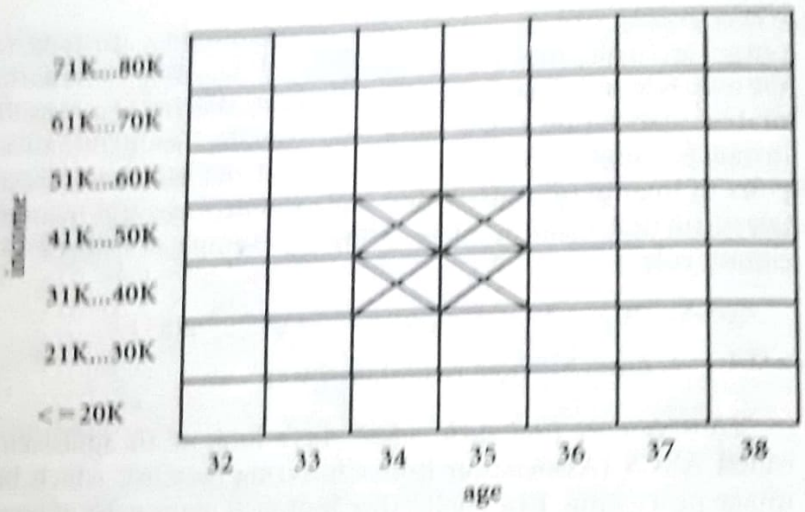

Figure 6.18 A 2-D grid for tuples representing customers who purchase high-resolution TVs.

frequent predicate sets (those satisfying minimum support) that also satisfy minimum confidence. Strong association rules can then be generated from these predicate sets, using a rule generation algorithm like that described in Section 6.2.2.

Clustering the association rules: The strong association rules obtained in the previous step are then mapped to a 2-D grid. Figure 6.18 shows a 2-D grid for 2-D quantitative association rules predicting the condition *buys*(*X*, "*high resolution TV*") on the rule right-hand side, given the quantitative attributes *age* and *income*. The four Xs correspond to the rules

$$age(X, 34) \wedge income(X, \text{"31K} \ldots \text{40K"}) \Rightarrow buys(X, \text{"high resolution TV"}) \quad (6.15)$$

$$age(X, 35) \wedge income(X, \text{"31K} \ldots \text{40K"}) \Rightarrow buys(X, \text{"high resolution TV"}) \quad (6.16)$$

$$age(X, 34) \wedge income(X, \text{"41K} \ldots \text{50K"}) \Rightarrow buys(X, \text{"high resolution TV"}) \quad (6.17)$$

$$age(X, 35) \wedge income(X, \text{"41K} \ldots \text{50K"}) \Rightarrow buys(X, \text{"high resolution TV"}) \quad (6.18)$$

"*Can we find a simpler rule to replace the above four rules?*" Notice that these rules are quite "close" to one another, forming a rule cluster on the grid. Indeed, the four rules can be combined or "clustered" together to form the following simpler rule, which subsumes and replaces the above four rules:

$$age(X, \text{"34} \ldots \text{35"}) \wedge income(X, \text{"31K} \ldots \text{50K"}) \Rightarrow buys(X, \text{"high resolution TV"}) \quad (6.19)$$

Price ($)	Equiwidth (width $10)	Equidepth (depth 2)	Distance-based
7	[0, 10]	[7, 20]	[7, 7]
20	[11, 20]	[22, 50]	[20, 22]
22	[21, 30]	[51, 53]	[50, 53]
50	[31, 40]		
51	[41, 50]		
53	[51, 60]		

Table 6.3 Binning methods like equiwidth and equidepth do not always capture the semantics of interval data.

ARCS employs a clustering algorithm for this purpose. The algorithm scans the grid, searching for rectangular clusters of rules. In this way, bins of the quantitative attributes occurring within a rule cluster may be further combined, and hence, further dynamic discretization of the quantitative attributes occurs.

The grid-based technique described here assumes that the initial association rules can be clustered into rectangular regions. Prior to performing the clustering, smoothing techniques can be used to help remove noise and outliers from the data. Rectangular clusters may oversimplify the data. Alternative approaches have been proposed, based on other shapes of regions that tend to better fit the data, yet require greater computation effort.

A non-grid-based technique has been proposed to find quantitative association rules that are more general, where any number of quantitative and categorical attributes can appear on either side of the rules. In this technique, quantitative attributes are dynamically partitioned using equidepth binning, and the partitions are combined based on a measure of *partial completeness*, which quantifies the information lost due to partitioning. For references on these alternatives to ARCS, see the bibliographic notes.

6.4.4 Mining Distance-Based Association Rules

The previous section described quantitative association rules where quantitative attributes are discretized initially by binning methods, and the resulting intervals are then combined. Such an approach, however, may not capture the semantics of interval data since they do not consider the relative distance between data points or between intervals.

Consider, for example, Table 6.3 which shows data for the attribute *price*, partitioned according to equiwidth and equidepth binning versus a distance-

based partitioning. The distance-based partitioning seems the most intuitive, since it groups values that are close together within the same interval (e.g., [20, 22]). In contrast, equidepth partitioning groups distant values together (e.g., [22, 50]). Equiwidth may split values that are close together and create intervals for which there are no data. Clearly, a distance-based partitioning that considers the density or number of points in an interval, as well as the "closeness" of points in an interval, helps produce a more meaningful discretization. Intervals for each quantitative attribute can be established by *clustering* the values for the attribute.

A disadvantage of association rules is that they do not allow for approximations of attribute values. Consider the following association rule:

$$item\_type(X, \text{"}electronic\text{"}) \wedge manufacturer(X, \text{"}foreign\text{"}) \Rightarrow price(X, \$200) \quad (6.20)$$

where X is a variable describing items at *AllElectronics*. In reality, it is more likely that the prices of foreign electronic items are *close to or approximately* \$200, rather than exactly \$200. It would be useful to have association rules that can express such a notion of closeness. Note that the support and confidence measures do not consider the closeness of values for a given attribute. This motivates the mining of **distance-based association rules**, which capture the semantics of interval data while allowing for approximation in data values. A two-phase algorithm can be used to mine distance-based association rules. The first phase employs clustering to find the intervals or clusters, adapting to the amount of available memory. The second phase obtains distance-based association rules by searching for groups of clusters that occur frequently together.

"*How are clusters formed in the first phase?*" Here, we give an intuitive description of how clusters can be formed. Interested readers may wish to read Chapter 8, as well as the references for distance-based association rules given in the bibliographic notes of this chapter. Let $S[X]$ be a set of N tuples $t_1, t_2, \ldots, t_N$ projected on the attribute set X. A **diameter** measure is defined to assess the closeness of tuples. The diameter of $S[X]$ is the average pairwise distance between the tuples projected on X. Distance measures such as the Euclidean distance or Manhattan distance may be used.[8] The smaller the diameter of $S[X]$ is, the "closer" its tuples are when projected on X. Hence, the diameter metric assesses the *density* of a cluster. A **cluster** C_X is a set of tuples defined on an attribute set X, where the tuples satisfy a **density threshold**, as well as a **frequency threshold**, which specifies the minimum number of tuples in a cluster. Clustering methods such as those described in Chapter 8 may be modified for use in this first phase of the mining process.

[8] The Euclidean and Manhattan distances between two tuples $t_1 = (x_{11}, x_{12}, \ldots, x_{1m})$ and $t_2 = (x_{21}, x_{22}, \ldots, x_{2m})$ are, respectively, $Euclidean\_d(t_1, t_2) = \sqrt{\sum_{i=1}^{m}(x_{1i} - x_{2i})^2}$ and $Manhattan\_d(t_1, t_2) = \sum_{i=1}^{m} |x_{1i} - x_{2i}|$.

In the second phase, clusters are combined to form distance-based association rules. Consider a simple distance-based association rule of the form $C_X \Rightarrow C_Y$. Suppose that X is the attribute set *{age}* and Y is the attribute set *{income}*. We want to ensure that the implication between the cluster C_X for *age* and C_Y for *income* is strong. This means that when the age-clustered tuples C_X are projected onto the attribute *income*, their corresponding *income* values lie within the income-cluster C_Y, or close to it. A cluster C_X projected onto the attribute set Y is denoted $C_X[Y]$. Therefore, the distance between $C_X[Y]$ and $C_Y[Y]$ must be small. This distance measures the *degree of association* between C_X and C_Y. The smaller the distance between $C_X[Y]$ and $C_Y[Y]$ the stronger the degree of association between C_X and C_Y. The degree of association measure can be defined using standard statistical measures, such as the average intercluster distance, or the centroid Manhattan distance, where the centroid of a cluster represents the "average" tuple of the cluster.

In general, clusters can be combined to find distance-based association rules of the form

$$C_{X_1} C_{X_2} \ldots C_{X_x} \Rightarrow C_{Y_1} C_{Y_2} \ldots C_{Y_y}$$

where X_i and Y_j are pairwise disjoint sets of attributes, and the following three conditions are met: (1) The clusters in the rule antecedent are each strongly associated with each cluster in the consequent; (2) the clusters in the antecedent collectively occur together; and (3) the clusters in the consequent collectively occur together. The degree of association replaces the confidence framework in non-distance-based association rules, while the density threshold replaces the notion of support.

6.5 From Association Mining to Correlation Analysis

"When mining association rules, how can the data mining system tell which rules are likely to be interesting to the user?" Most association rule mining algorithms employ a support-confidence framework. In spite of using minimum support and confidence thresholds to help weed out or exclude the exploration of uninteresting rules, many rules that are not interesting to the user may still be produced. In this section, we first look at how even strong association rules can be uninteresting and misleading, and then discuss additional measures based on statistical independence and correlation analysis.

6.5.1 Strong Rules Are Not Necessarily Interesting: An Example

"In data mining, are all of the strong *association rules discovered (i.e., those rules satisfying the minimum support and minimum confidence thresholds) interesting enough to present to the user?"* Not necessarily. Whether a rule is interesting or not

can be judged either subjectively or objectively. Ultimately, only the user can judge if a given rule is interesting or not, and this judgment, being subjective, may differ from one user to another. However, objective interestingness measures, based on the statistics "behind" the data, can be used as one step towards the goal of weeding out uninteresting rules from presentation to the user.

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

Example 6.6 Suppose we are interested in analyzing transactions 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:

$$buys(X, \text{“}computer\ games\text{”}) \Rightarrow buys(X, \text{“}videos\text{”})$$
$$[support = 40\%, confidence = 66\%] \tag{6.21}$$

Rule (6.21) 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 (6.21) is misleading since the probability of purchasing videos is 75%, which is even larger than 66%. In fact, computer games and videos are negatively associated because the purchase of one of these items actually decreases the likelihood of purchasing the other. Without fully understanding this phenomenon, one could make unwise business decisions based on the rule derived. ∎

The above example also illustrates that the confidence of a rule $A \Rightarrow B$ can be deceiving in that it is only an *estimate* of the conditional probability of itemset B given itemset A. It does not measure the real strength (or lack of strength) of the implication between A and B. Hence, alternatives to the support-confidence framework can be useful in mining interesting data relationships.

6.5.2 From Association Analysis to Correlation Analysis

Association rules mined using a support-confidence framework are useful for many applications. However, the support-confidence framework can be misleading in that it may identify a rule $A \Rightarrow B$ as interesting when, in fact, the occurrence of A does not imply the occurrence of B. In this section, we consider an alternative framework for finding interesting relationships between data itemsets based on correlation.

The occurrence of itemset A is **independent** of the occurrence of itemset B if $P(A \cup B) = P(A)P(B)$; otherwise itemsets A and B are **dependent** and **correlated** as events. This definition can easily be extended to more than two itemsets. The correlation between the occurrence of A and B can be measured by computing

$$corr_{A,B} = \frac{P(A \cup B)}{P(A)P(B)}. \tag{6.22}$$

If the resulting value of Equation (6.22) is less than 1, then the occurrence of A is negatively correlated with (or discourages) the occurrence of B. If the resulting value is greater than 1, then A and B are positively correlated, meaning 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 (6.22) is equivalent to $P(B|A)/P(B)$, which is also called the lift of the association rule $A \Rightarrow B$. 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 Equation (6.22).

Let's go back to the computer game and video data of Example 6.6.

Example 6.7 To help filter out misleading "strong" associations of the form $A \Rightarrow B$, we need to study how the two itemsets, A and B, are correlated. Let $\overline{game}$ refer to the transactions of Example 6.6 that do not contain computer games, and $\overline{video}$ refer to those that do not contain videos. The transactions can be summarized in a contingency table. A contingency table for the data of Example 6.6 is shown in Table 6.4. 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 Equation (6.22), $P(\{game, video\})/(P(\{game\}) \times P(\{video\})) = 0.40/(0.60 \times 0.75) = 0.89$. Since this value is less than 1, there is a negative correlation between the occurrence 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

Table 6.4 A 2 × 2 contingency table summarizing the transactions with respect to computer game and video purchases.

	game	$\overline{game}$	Σ_{row}
video	4,000	3,500	7,500
$\overline{video}$	2,000	500	2,500
Σ_{col}	6,000	4,000	10,000

purchases were completely independent. Such a negative correlation cannot be identified by a support-confidence framework.

This motivates the mining of rules that identify correlations, or *correlation rules*. A **correlation rule** is of the form $\{i_1, i_2, \ldots, i_m\}$ where the occurrences of the items $\{i_1, i_2, \ldots, i_m\}$ are correlated. Given a correlation value determined by Equation (6.22), the χ^2 statistic can be used to determine if the correlation is statistically significant. The χ^2 statistic can also determine negative implication.

An advantage of correlation is that it is *upward closed*. This means that if a set S of items is correlated (i.e., the items in S are correlated), then every superset of S is also correlated. In other words, adding items to a set of correlated items does not remove the existing correlation. The χ^2 statistic is also upward closed within each significance level.

When searching for sets of correlations to form correlation rules, the upward closure property of correlation and χ^2 can be used. Starting with the empty set, we may explore the itemset space (or *itemset lattice*), adding one item at a time, looking for **minimal correlated itemsets**—itemsets that are correlated although no subset of them is correlated. These itemsets form a **border** within the lattice. Because of closure, no itemset below this border will be correlated. Since all supersets of a minimal correlated itemset are correlated, we can stop searching upward. An algorithm that performs a series of such "walks" through itemset space is called a **random walk algorithm**. Such an algorithm can be combined with tests of support in order to perform additional pruning. Random walk algorithms can easily be implemented using data cubes. It is an open problem to adapt the procedure described here to very large databases. Another limitation is that the χ^2 statistic is less accurate when the contingency table data are sparse, and can be misleading for contingency tables larger than 2×2. Proposed alternatives to the support-confidence framework for assessing the interestingness of association rules are given in the bibliographic notes.

6.6 Constraint-Based Association Mining

For a given set of task-relevant data, the data mining process may uncover thousands of rules, many of which are uninteresting to the user. In **constraint-based mining**, mining is performed under the guidance of various kinds of constraints provided by the user. These constraints include the following:

- **Knowledge type constraints:** These specify the type of knowledge to be mined, such as association.
- **Data constraints:** These specify the set of task-relevant data.
- **Dimension/level constraints:** These specify the dimension of the data, or levels of the concept hierarchies, to be used.

- **Interestingness constraints:** These specify thresholds on statistical measures of rule interestingness, such as support and confidence.
- **Rule constraints:** These specify the form of 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.

The above constraints can be specified using a high-level declarative data mining query language, such as that described in Chapter 4.

The first four of the above types of constraints have already been addressed in earlier parts of this book and chapter. In this section, we discuss the use of *rule constraints* to focus the mining task. This form of constraint-based mining allows users to specify the rules to be mined according to their intention, thereby making the data mining process more *effective*. In addition, a sophisticated mining query optimizer can be used to exploit the constraints specified by the user, thereby making the mining process more *efficient*. Constraint-based mining encourages interactive exploratory mining and analysis. In Section 6.6.1, you will study metarule-guided mining, where syntactic rule constraints are specified in the form of rule templates. Section 6.6.2 discusses the use of additional rule constraints, specifying set/subset relationships, constant initiation of variables, and aggregate functions.

6.6.1 Metarule-Guided Mining of Association Rules

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

Example 6.8 Suppose that as a market analyst for *AllElectronics*, you have access to the data describing customers (such as customer age, address, and credit rating) as well as the list of customer transactions. You are interested in finding associations between customer traits and the items that customers buy. However, rather than finding *all* of the association rules reflecting these relationships, you are particularly interested only in determining which pairs of customer traits promote the sale of educational software. A metarule can be used to specify this information describing the form of rules you are interested in finding. An example of such a metarule is

$$P_1(X, Y) \wedge P_2(X, W) \Rightarrow buys(X, \text{“}educational\ software\text{”}) \tag{6.23}$$

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.