UNIT-II

ASSOCIATION RULE MINING

Mining Frequent Patterns, Associations and Correlations – Mining Methods – Mining Various Kinds of Association Rules – Correlation Analysis – Constraint Based Association Mining

Basic Concepts

Frequent pattern mining searches for recurring relationships in a given data set. It introduces the basic concepts of frequent pattern mining for the discovery of interesting associations and correlations between itemsets in transactional and relational databases.

Market Basket Analysis: A Motivating Example

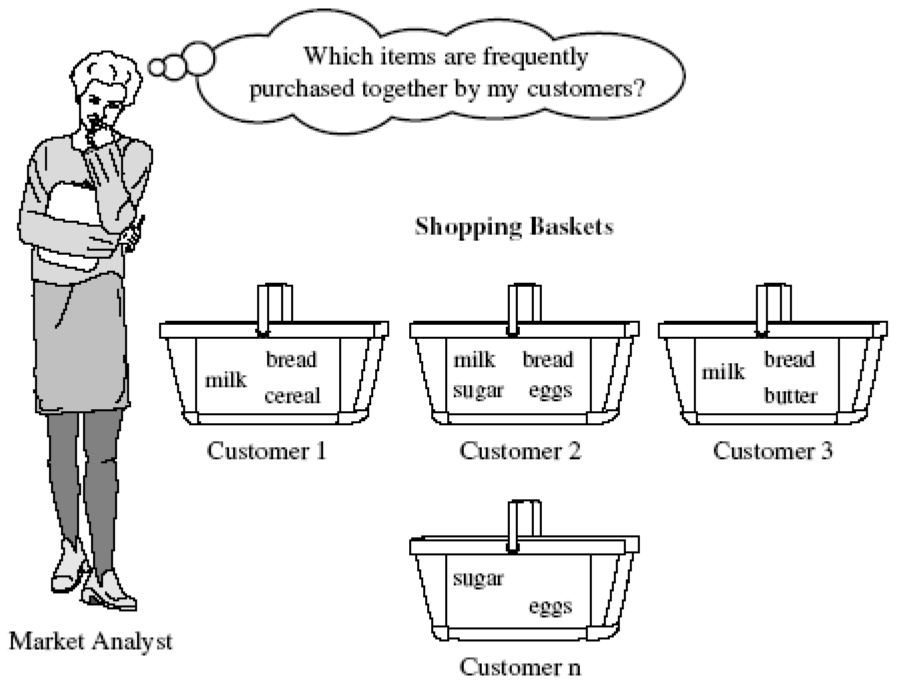


Figure 5.1 Market basket analysis.

A typical example of frequent itemset mining is market basket analysis. This process analyzes customer buying habits by finding associations between the different items that customers place in their ―shopping baskets‖ (Figure 5.1). The discovery of such associations can help retailers develop marketing strategies by gaining insight into which items are frequently purchased together by customers. For instance, if customers are buying milk, how likely are they to also buy bread (and what kind of bread) on the same trip to the supermarket? Such information can lead to increased sales by helping retailers do selective marketing and plan their shelf space.

If we think of the universe as the set of items available at the store, then each item has a Boolean variable representing the presence or absence of that item. Each basket can then be represented by a Boolean vector of values assigned to these variables. The Boolean vectors can be analyzed for buying patterns that reflect items that are frequently *associated* or purchased together. These patterns can be represented in the form of association rules. For example, the information that customers who purchase computers also tend to buy antivirus software at the same time is represented in Association Rule (5.1) below:

Computer =>*antivirus software* [*support* = 2%; *confidence* = 60%] (5.1)

Rule support and confidence are two measures of rule interestingness. They respectively reflect the usefulness and certainty of discovered rules. A support of 2% for Association Rule (5.1) means that 2% of all the transactions und er analysis show that computer and antivirus software are purchased together. A confidence of 60% means that 60% of the customers who purchased a computer also bought the software. Typically, association rules are considered interesting if they satisfy bot h a minimum support threshold and a minimum confidence threshold. Such thresholds can be set by users or domain experts. Additional analysis can be performed to uncover interesting statistical correlations between associated items.

Frequent Itemsets, Closed Itemsets, and Association Rules

- A set of items is referred to as an **itemset.**
- An itemset that contains *k* items is a ***k*-itemset**.
- The set {*computer, antivirus software}* is a **2-itemset**.
- The occurrence frequency of an itemset is the number of transactions that contain the itemset. This is also known, simply, as the **frequency, support count,** or **count** of the itemset.

$$\begin{aligned} support(A \Rightarrow B) &= P(A \cup B) \\ confidence(A \Rightarrow B) &= P(B|A). \end{aligned}$$

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

- Rules that satisfy both a minimum support threshold (*min sup*) and a minimum confidence threshold (*min conf*) are called **Strong Association Rules.**

In general, association rule mining can be viewed as a **two-step process**:

1. Find all frequent itemsets: By definition, each of these itemsets will occur at least as frequently as a predetermined minimum support count, *min_sup*.
2. Generate strong association rules from the frequent itemsets: By definition, these rules must satisfy minimum support and minimum confidence.

The Apriori Algorithm: Finding Frequent Itemsets Using Candidate Generation

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 following. 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 *L*1.Next, *L*1 is used to find *L*2, the set of frequent 2-itemsets, which is used to find *L*3, 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, presented below, is used to reduce the search space. We will first describe this property, and then show an example illustrating its use.

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

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. Let l_1 and l_2 be itemsets in L_{k-1}. The notation $l_i[j]$ refers to the jth item in l_i (e.g., $l_1[k-2]$ refers to the second to the last item in l_1). By convention, Apriori assumes that items within a transaction or itemset are sorted in lexicographic order. For the $(k-1)$-itemset, l_i, this means that the items are sorted such that $l_i[1] < l_i[2] < \ldots < l_i[k-1]$. The join, $L_{k-1} \bowtie L_{k-1}$, is performed, where members of L_{k-1} are joinable if their first $(k-2)$ items are in common. That is, members l_1 and l_2 of L_{k-1} are joined if $(l_1[1] = l_2[1]) \wedge (l_1[2] = l_2[2]) \wedge \ldots \wedge (l_1[k-2] = l_2[k-2]) \wedge (l_1[k-1] < l_2[k-1])$. The condition $l_1[k-1] < l_2[k-1]$ simply ensures that no duplicates are generated. The resulting itemset formed by joining l_1 and l_2 is $l_1[1], l_1[2], \ldots, l_1[k-2], l_1[k-1], l_2[k-1]$.
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 C_k. A scan of the database to determine the count of each candidate in C_k would result in the determination of L_k (i.e., all candidates having a count no less than the minimum support count are frequent by definition, and therefore belong to L_k). C_k, however, can be huge, and so this could involve heavy computation. To reduce the size of C_k, the Apriori property is used as follows. Any $(k-1)$-itemset that is not frequent cannot be a subset of a frequent k-itemset. Hence, if any $(k-1)$-subset of a candidate k-itemset is not in L_{k-1}, then the candidate cannot be frequent either and so can be removed from C_k. This **subset testing** can be done quickly by maintaining a hash tree of all frequent itemsets.

Table 5.1 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

Example 5.3 **Apriori.** Let's look at a concrete example, based on the *AllElectronics* transaction database, D, of Table 5.1. There are nine transactions in this database, that is, $|D| = 9$. We use Figure 5.2 to illustrate the Apriori algorithm for finding frequent itemsets in D.

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 in order 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.[8] C_2 consists of $\binom{|L_1|}{2}$ 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 5.2.
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 candidate 3-itemsets, C_3, is detailed in Figure 5.3. 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. Note that when given a candidate k-itemset, we only need to check if its $(k-1)$-subsets are frequent since the Apriori algorithm uses a level-wise search strategy. The resulting pruned version of C_3 is shown in the first table of the bottom row of Figure 5.2.
7. The transactions in D are scanned in order to determine L_3, consisting of those candidate 3-itemsets in C_3 having minimum support (Figure 5.2).
8. The algorithm uses $L_3 \bowtie L_3$ to generate a candidate set of 4-itemsets, C_4. Although the join results in $\{\{I1, I2, I3, I5\}\}$, this itemset is pruned because its subset $\{\{I2, I3, I5\}\}$ is not frequent. Thus, $C_4 = \phi$, and the algorithm terminates, having found all of the frequent itemsets. ■

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 Equation (5.4) for confidence, which we show again here for completeness:

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

Algorithm: Apriori. Find frequent itemsets using an iterative level-wise approach based on candidate generation.

Input:

- D, a database of transactions;
- min_sup, the minimum support count threshold.

Output: L, frequent itemsets in D.

Method:

```
(1)   $L_1$ = find_frequent_1-itemsets(D);
(2)   for ($k = 2; L_{k-1} \neq \phi; k++$) {
(3)       $C_k$ = apriori_gen($L_{k-1}$);
(4)       for each transaction $t \in D$ { // scan $D$ for counts
(5)           $C_t$ = subset($C_k, t$); // get the subsets of $t$ that are candidates
(6)           for each candidate $c \in C_t$
(7)               c.count++;
(8)       }
(9)       $L_k = \{c \in C_k | c.count \geq min\_sup\}$
(10)  }
(11)  return $L = \cup_k L_k$;

procedure apriori_gen($L_{k-1}$:frequent $(k-1)$-itemsets)
(1)   for each itemset $l_1 \in L_{k-1}$
(2)       for each itemset $l_2 \in L_{k-1}$
(3)           if $(l_1[1] = l_2[1]) \wedge (l_1[2] = l_2[2]) \wedge ... \wedge (l_1[k-2] = l_2[k-2]) \wedge (l_1[k-1] < l_2[k-1])$ then {
(4)               $c = l_1 \bowtie l_2$; // join step: generate candidates
(5)               if has_infrequent_subset($c, L_{k-1}$) then
(6)                   delete $c$; // prune step: remove unfruitful candidate
(7)               else add $c$ to $C_k$;
(8)           }
(9)   return $C_k$;

procedure has_infrequent_subset($c$: candidate $k$-itemset;
            $L_{k-1}$: frequent $(k-1)$-itemsets); // use prior knowledge
(1)   for each $(k-1)$-subset $s$ of $c$
(2)       if $s \notin L_{k-1}$ then
(3)           return TRUE;
(4)   return FALSE;
```

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 minimum support. Frequent itemsets can be stored ahead of time in hash tables along with their counts so that they can be accessed quickly.

Example 5.4 **Generating association rules.** Let's try an example based on the transactional data for *AllElectronics* shown in Table 5.1. Suppose the data contain the frequent itemset l = {I1, I2, I5}. What are the association rules that can be generated from l? The nonempty subsets of l are {I1, I2}, {I1, I5}, {I2, I5}, {I1}, {I2}, and {I5}. The resulting association rules are as shown below, each listed with its confidence:

$$\begin{array}{ll}
I1 \wedge I2 \Rightarrow I5, & \textit{confidence} = 2/4 = 50\% \\
I1 \wedge I5 \Rightarrow I2, & \textit{confidence} = 2/2 = 100\% \\
I2 \wedge I5 \Rightarrow I1, & \textit{confidence} = 2/2 = 100\% \\
I1 \Rightarrow I2 \wedge I5, & \textit{confidence} = 2/6 = 33\% \\
I2 \Rightarrow I1 \wedge I5, & \textit{confidence} = 2/7 = 29\% \\
I5 \Rightarrow I1 \wedge I2, & \textit{confidence} = 2/2 = 100\%
\end{array}$$

If the minimum confidence threshold is, say, 70%, then only the second, third, and last rules above 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-hand side of the rule. ■

FP-Growth Method: Mining Frequent Itemsets without Candidate Generation

As we have seen, in many cases the Apriori candidate generate-and-test method significantly reduces the size of candidate sets, leading to good performance gain.

An interesting method in this attempt is called frequent-pattern growth, or simply FP-growth, which adopts a *divide- and-conquer* strategy as follows. First, it compresses the database representing frequent items into a frequent-pattern tree, or FP-tree, which retains the itemset association information. It then divides the compressed database into a set of *conditional databases* (a special kind of projected database), each associated with one frequent item or ―pattern fragment,‖ and mines each such database separately. You'll see how it works with the following example.

Example 5.5: FP-growth (finding frequent itemsets without candidate generation). We re-examine the mining of transaction database, *D*, of Table 5.1 in Example 5.3 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⟩, and ⟨I5: 1⟩, where I2 is linked as a child of the root, I1 is linked to I2, and I5 is linked to I1. 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.

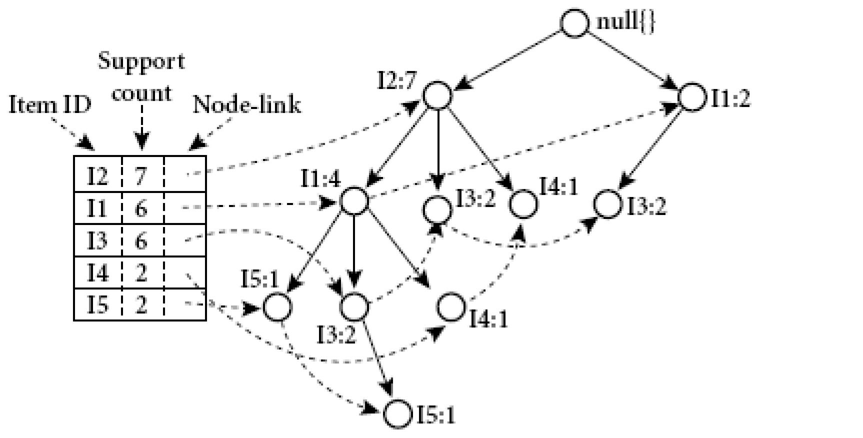


Figure 5.7 An FP-tree registers compressed, frequent pattern information.

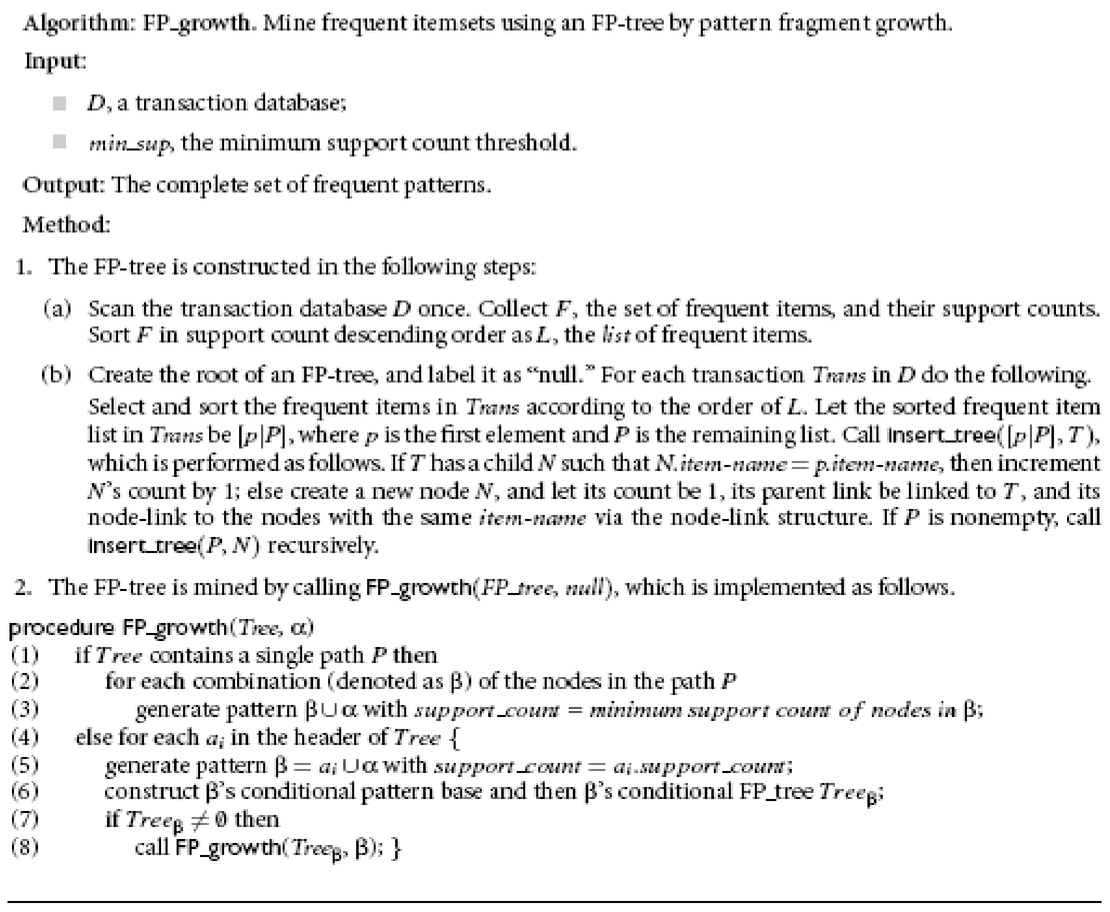
Mining the FP-tree by creating conditional (sub-)pattern bases.

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}

Mining of the FP-tree is summarized in Table 5.2 and detailed as follows. We first consider I5, which is the last item in *L*, rather than the first. The reason for starting at the end of the list will become apparent as we explain the FP-tree mining process. I5 occurs in two branches of the FP-tree of Figure 5.7. (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, I1: 2}. Notice that although I5 follows I4 in the first branch, there is no need to include I5 in the analysis here because any frequent pattern involving I5 is analyzed in the examination of I5.

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 5.8, which generates the set of patterns, {{I2, I3: 4}, {I1, I3: 4}, {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 5.9. ■



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, $\alpha$)
(1)   if Tree contains a single path P then
(2)       for each combination (denoted as $\beta$) of the nodes in the path P
(3)           generate pattern $\beta \cup \alpha$ with support_count = minimum support count of nodes in $\beta$;
(4)   else for each $a_i$ in the header of Tree {
(5)       generate pattern $\beta = a_i \cup \alpha$ with support_count = $a_i$.support_count;
(6)       construct $\beta$'s conditional pattern base and then $\beta$'s conditional FP_tree $Tree_\beta$;
(7)       if $Tree_\beta \neq \emptyset$ then
(8)           call FP_growth($Tree_\beta$, $\beta$); }
```

Figure 5.9 The FP-growth algorithmfor discovering frequent itemsets without candidategeneration.

Also Read Example problems which we solved in Class Lecture

<u>Mining Various Kinds of Association Rules:</u>

1) Mining 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 at those levels. Strong associations discovered at high levels of abstraction may represent commonsense knowledge. Moreover, what may represent common sense to one user may seem novel to another. Therefore, data mining systems should provide capabilities for mining association rules at multiple levels of abstraction, with sufficient flexibility for easy traversal among different abstraction spaces.

Let's examine the following example.

Mining multilevel association rules. Suppose we are given the task-relevant set of transactional data in Table for sales in an *AllElectronics* store, showing the items purchased for each transaction. The concept hierarchy for the items is shown in Figure 5.10. 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 concepthierarchy.

TID	*Items Purchased*
T100	IBM-ThinkPad-T40/2373, HP-Photosmart-7660
T200	Microsoft-Office-Professional-2003, Microsoft-Plus!-Digital-Media
T300	Logitech-MX700-Cordless-Mouse, Fellowes-Wrist-Rest
T400	Dell-Dimension-XPS, Canon-PowerShot-S400
T500	IBM-ThinkPad-R40/P4M, Symantec-Norton-Antivirus-2003
...	...

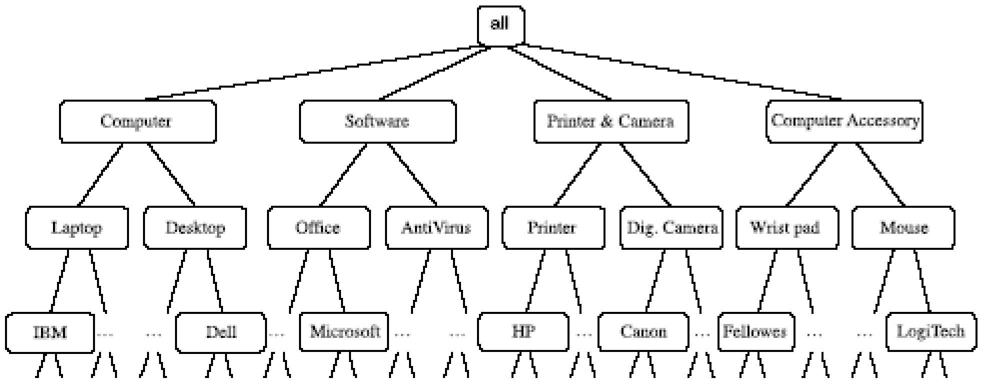
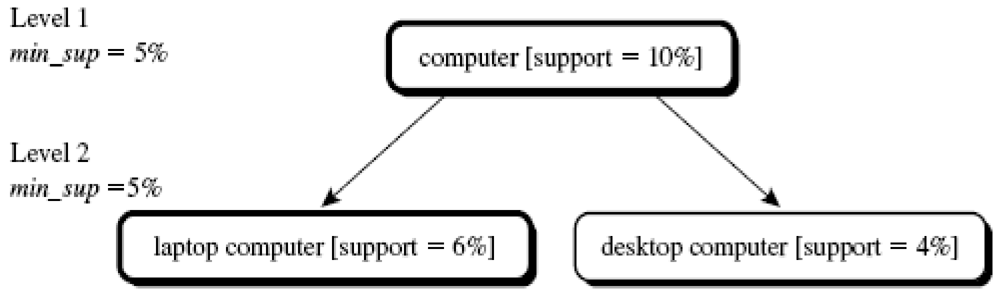


Figure 5.10 A concept hierarchy for *AllElectronics* computeritems.

Association rules generated from mining data at multiple levels of abstraction are called multiple-level or multilevel association rules. Multilevel association rules can be mined efficiently using concept hierarchies under a support-confidence framework. In general, a top-down strategy is employed, where counts are accumulated for the calculation of frequent itemsets at each concept level, starting at the 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.

- **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 5.11, 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 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 whose ancestors do not have minimum support.


Level 1
min_sup = 5%
computer [support = 10%]
Level 2
min_sup =5%
laptop computer [support = 6%]
desktop computer [support = 4%]

Multilevel mining with uniform support.

- **Using reduced minimum support at lower levels (referred to as reduced support):** Each level of abstraction has its own minimum support threshold. The deeper the level of abstraction, the smaller the corresponding threshold is. For example, in Figure, the minimum support thresholds for levels 1 and 2 are 5% and 3%, respectively. In this way, *"computer," "laptop computer,"* and *"desktop computer"* are all considered frequent.

- **Using item or group-based minimum support (referred to as group-based support):** Because users or experts often have insight as to which groups are more important than others, it is sometimes more desirable to set up user-specific, item, or group based minimal support thresholds when mining multilevel rules. For example, a user could set up the minimum support thresholds based on pro duct price, or on items of interest, such as by setting particularly low support thresholds for *laptop computers* and *flash drives* in order to pay particular attention to the association patterns containing items in these categories.

2) Mining Multidimensional Association Rules from Relational Databases and DataWarehouses

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

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

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 above as a single dimensional or intra dimensional 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). As we have seen in the previous sections of this chapter, such rules are commonly mined from transactional data.

Considering each database attribute or warehouse dimension as a predicate, we can therefore mine association rules containing *multiple* predicates, such as

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

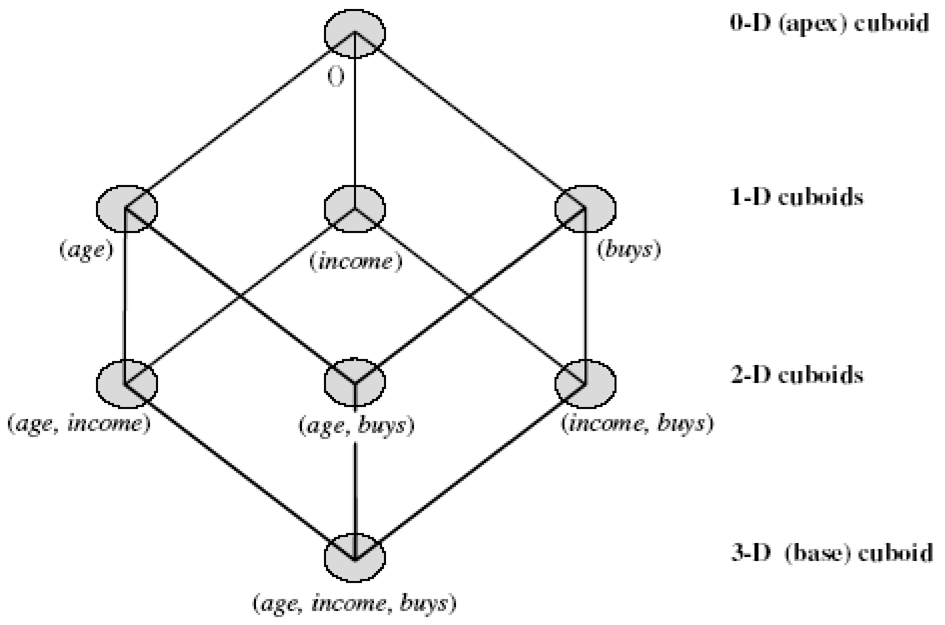
Association rules that involve two or more dimensions or predicates can be referred to as multidimensional association rules. Rule above contains three predicates (*age, occupation*, and *buys*), each of which occurs *only once* in the rule. Hence, we say that it has no repeated predicates. Multidimensional association rules with no repeated predicates are c alled inter dimensional 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:

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

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, because 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 into two basic approaches regarding the treatment of quantitative attributes.

Mining Multidimensional Association Rules Using Static Discretization of Quantitative Attributes

Quantitative attributes, in this case, are discretized before mining using predefined concept hierarchies or data discretization techniques, where numeric values are replaced by interval labels. 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 any of the frequent itemset mining algorithms we have discussed can be modified easily so as to find all frequent predicate sets rather than freq uent itemsets. 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.



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*.

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 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. That is,

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

where *Aquan*1 and *Aquan*2 are tests on quantitative attribute intervals (where the intervals are dynamically determined), and *Acat* tests a categorical attribute from the task-relevant data. Such rules have been referred to as two-dimensional quantitative association rules, because 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 (such as *high-definition TV,* i.e., *HDTV*) that customers like to buy. An example of such a 2-D quantitative association rule is

$$age(X, \text{“30...39”}) \wedge income(X, \text{“42K...48K”}) \Rightarrow buys(X, \text{“HDTV”})$$

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, wh ere the interv als are conside re d ―bin s.‖ Three comm o n binnin g strate gie s area as follow s:

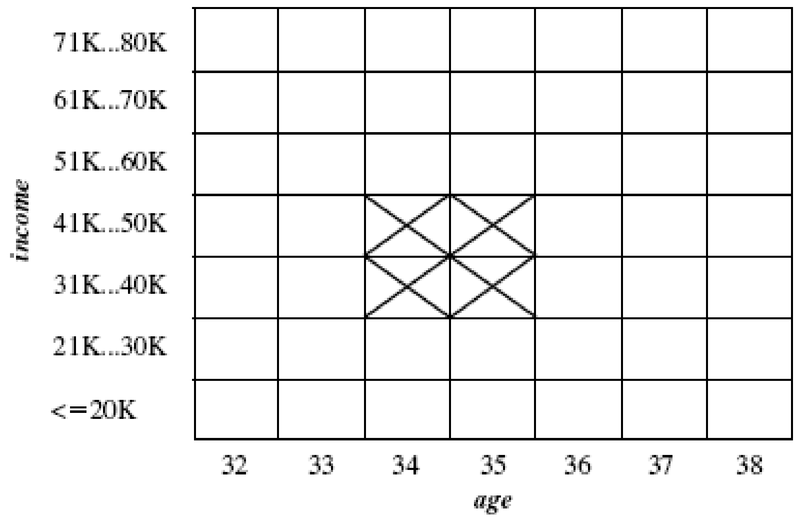
- **Equal-width binning**, where the interval size of each bin is the same
- **Equal-frequency binning**, where each bin has approximately the same number of tuples assigned to it,
- **Clustering-based binning**, where clustering is performed on the quantitative attribute to group *neighboring points* (judged based on various distance measures) into the same bin

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

Clustering the association rules: The strong association rules obtained in the previous step are then mapped to a 2-D grid. Figure 5.14 shows a 2-D grid for 2-D quantitative association rules predicting the condition *buys*(*X*, "*HDTV*") on the rule right-hand side, given the quantitative attributes *age* and *income*. The four Xs correspond to the rules

$$age(X, 34) \wedge income(X, \text{“31K...40K”}) \Rightarrow buys(X, \text{“HDTV”}) \quad (5.16)$$
$$age(X, 35) \wedge income(X, \text{“31K...40K”}) \Rightarrow buys(X, \text{“HDTV”}) \quad (5.17)$$
$$age(X, 34) \wedge income(X, \text{“41K...50K”}) \Rightarrow buys(X, \text{“HDTV”}) \quad (5.18)$$
$$age(X, 35) \wedge income(X, \text{“41K...50K”}) \Rightarrow buys(X, \text{“HDTV”}). \quad (5.19)$$

"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:



A 2-D grid for tuples representing customers who purchase high-definition TVs.

From Association Mining to CorrelationAnalysis

Most association rule mining algorithms employ a support-confidence framework. Often, many interesting rules can be found using low support thresholds. Although minimum support and confidence thresholds *help* weed out or exclude the exploration of a good number of uninteresting rules, many rules so 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 one of the major bottlenecks for successful application of association rulemining.

1) Strong Rules Are Not Necessarily Interesting: An Example

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 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 toward the goal of weeding out uninteresting rules from presentation to the user.

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

$$A \Rightarrow B\ [support, confidence. correlation].$$

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 measures from which to choose. In this section, we study various correlation measures to determine which would be good for mining large data sets.

Constraint-Based Association Mining:

A data mining process may uncover thousands of rules from a given set of data, most of which end up being unrelated or uninteresting to the users. Often, users have a good sense of which ―direction‖

of mining may lead to interesting patterns and the —form‖ of the patte rn s or rules they would like to find. Thus, a good heuristic is to have the users specif y such intuitio n 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 or correlation.
- **Data constraints:** These specify the set of task-relevant data.
- **Dimension/level constraints:** These specify the desired dimensions (or attributes) of the data, or levels 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 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.

1) Metarule-Guided Mining of Association Rules

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

Metarule-guided mining:- 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 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

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

where $P1$ and $P2$ 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 $P1$ and $P2$, respectively. Typically, a user will specify a list of attributes to be considered for instantiation with $P1$ and $P2$. Otherwise, a default set may be used.

2) Constraint Pushing: Mining Guided by Rule Constraints

Rule constraints specify expected set/subset relationships of the variables in the mined rules, constant initiation of variables, and aggregate functions. 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 min ing process more efficient. Let's study an example where rule constraints are used to mine hybrid-dimensional association rules.

Our association mining query is to *"Find the sales of which cheap items (where the sum of the prices is less than* $100) *may promote the sales of which expensive items (where the minimum price is* $500) *of the same group for Chicago customers in 2004."* This can be expressed in the DMQL data mining query language as follows,

```
(1) mine associations as
(2) $lives\_in(C, \_, \text{"Chicago"}) \wedge sales^{+}(C, ?\{I\}, \{S\}) \Rightarrow sales^{+}(C, ?\{J\}, \{T\})$
(3) from sales
(4) where S.year = 2004 and T.year = 2004 and I.group = J.group
(5) group by C, I.group
(6) having sum(I.price) < 100 and min(J.price) ≥ 500
(7) with support threshold = 1%
(8) with confidence threshold = 50%
```