Unit-III

Mining Frequent Patterns, Associations, and Correlations: Basic Concepts

Frequent patterns are patterns (e.g., itemsets, subsequences, or substructures) that appear frequently in a data set. For example, a set of items, such as milk and bread, that appear frequently together in a transaction data set is a *frequent itemset*.
A subsequence, such as buying first a PC, then a digital camera, and then a memory card, if it occurs frequently in a shopping history database, is a (*frequent*) *sequential pattern*. A *substructure* can refer to different structural forms, such as subgraphs, subtrees, or sublattices, which may be combined with itemsets or subsequences. If a substructure occurs frequently, it is called a (*frequent*) *structured pattern*. Finding frequent patterns plays an essential role in mining associations, correlations, and many other interesting relationships among data.

Market Basket Analysis: A Motivating Example

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 6.1). The discovery of these 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? This information can lead to increased sales by helping retailers do selective marketing and plan their shelf space.

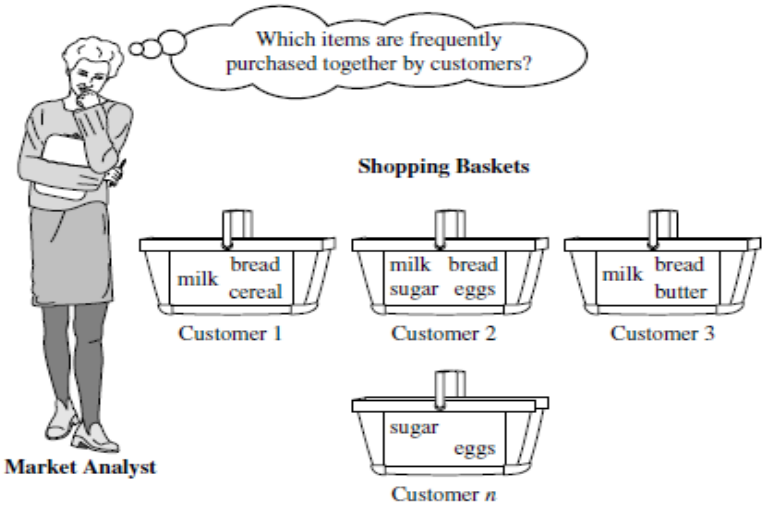


Figure 6.1 Market basket analysis.

"Which groups or sets of items are customers likely to purchase on a given trip to the store?"
To answer your question, market basket analysis may be performed on the retail data of customer transactions at your store. You can then use the results to plan marketing or advertising strategies, or in the design of a new catalog.

Source: Data Mining Concepts and Techniques, 3rd Edition, Han, Kamber and Pei

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 the following association rule:

$$computer \Rightarrow antivirus\_software\ [support = 2\%, confidence = 60\%].$$

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 above Rule means that 2% of all the transactions under 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 both a **minimum support threshold** and a **minimum confidence threshold**. These thresholds can be a set by users or domain experts.

Frequent Itemsets, Closed Itemsets, and Association Rules

$$support(A \Rightarrow B) = P(A \cup B)$$
$$confidence(A \Rightarrow B) = P(B|A).$$

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

A set of items is referred to as an **itemset**. An itemset that contains *k* items is a ***k*-itemset**. The set f*computer, antivirus softwar*eg 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.

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

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.

An itemset *X* is **closed** in a data set *D* if there exists no proper super-itemset *Y* such that *Y* has the same support count as *X* in*D*. An itemset *X* is a **closed frequent itemset** in set *D* if *X* is both closed and frequent in *D*. An itemset *X* is a **maximal frequent itemset** (or **max-itemset**) in a data set *D* if *X* is frequent, and there exists no super-itemset *Y* such that *X* _ *Y* and *Y* is frequent in *D*.

Example. Closed and maximal frequent itemsets. Suppose that a transaction database has only two transactions: {*a*1, *a*2, : : : , *a*100}; {*a*1, *a*2, : : : , *a*50}. Let the minimum support count threshold be *min sup* = 1. We find two closed frequent itemsets and their support counts, that is, C={{*a*1, *a*2, : : : , *a*100} : 1; {*a*1, *a*2, : : : , *a*50} : 2}. There is only one maximal frequent itemset: M={{*a*1, *a*2, : : : , *a*100} : 1}. Notice that we cannot include {*a*1, *a*2, : : : , *a*50} as a maximal frequent itemset because it has a frequent superset, {*a*1, *a*2, : : : , *a*100}. Compare this

to the preceding where we determined that there are 2^{100} -1 frequent itemsets, which are too many to be enumerated!
The set of closed frequent itemsets contains complete information regarding the frequent itemsets. For example, from C, we can derive, say, (1) {*a*2, *a*45 : 2} since {*a*2, *a*45} is a sub-itemset of the itemset {*a*1, *a*2, … , *a*50 : 2}; and (2) {*a*8, *a*55 : 1} since {*a*8, *a*55} is not a sub-itemset of the previous itemset but of the itemset {*a*1, *a*2, : : : , *a*100 : 1}. However, from the maximal frequent itemset, we can only assert that both itemsets ({*a*2, *a*45} and {*a*8, *a*55}) are frequent, but we cannot assert their actual support counts.

Frequent Itemset Mining Methods
Apriori Algorithm: Finding Frequent Itemsets by Confined Candidate Generation

Apriori is a seminal algorithm proposed by R. Agrawal and R. Srikant in 1994 for mining frequent itemsets for Boolean association rules [AS94b]. The name of the algorithm is based on the fact that the algorithm uses *prior knowledge* of frequent itemset properties, as we shall see later. Apriori employs an iterative approach known as a *level-wise* search, where *k*-itemsets are used to explore (*k* +1)- itemsets. First, the set of frequent 1-itemsets is found by scanning the database to accumulate the count for each item, and collecting those items that satisfy minimum support. The resulting set is denoted by *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** is used to reduce the search space.

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

"How is the Apriori property used in the algorithm?"
A two-step process is followed, consisting of **join** and **prune** actions.
1. The join step: To find *Lk*, a set of **candidate** *k*-itemsets is generated by joining *Lk*-1 with itself. This set of candidates is denoted *Ck*. Let *l*1 and *l*2 be itemsets in *Lk*-1. The notation *li*[*j*] refers to the *j*th item in *li* (e.g., *l*1[*k* -2] refers to the second to the last item in *l*1). For efficient implementation, Apriori assumes that items within a transaction or itemset are sorted in lexicographic order. For the (*k* -1)-itemset, *li* , this means that the items are sorted such that *li*[1] < *li*[2] < …… < *li*[*k* -1]. The join, *Lk*-1 ⋈ *Lk*-1, is performed, where members of *Lk*-1 are joinable if their first (*k* -2) items are in common. That is, members *l*1 and *l*2 of *Lk*-1 are joined if $(l1[1] = l2[1]) \wedge (l_1[2] = l2[2]) \wedge \ldots \wedge (l_1[k-2] = l2[k-2]) \wedge (l1[k-1] < l2[k-1])$. 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], …. , *l*1[*k* -2], *l*1[*k* -1], *l*2[*k* -1]}.

2. The prune step: *Ck* is a superset of *Lk*, that is, its members may or may not be frequent, but all of the frequent *k*-itemsets are included in *Ck*. A database scan to determine the count of each candidate in *Ck* would result in the determination of *Lk* (i.e., all candidates having a count no less than the minimum support count are frequent by definition, and therefore belong to *Lk*). *Ck*, however, can be huge, and so this could involve heavy computation. To reduce the size of *Ck*, 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 *Lk*-1, then the candidate cannot be frequent either and so can be removed from *Ck*. This **subset testing** can be done quickly by maintaining a hash tree of all frequent itemsets.

Table 6.1 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 6.3 Apriori. Let's look at a concrete example, based on the *AllElectronics* transaction database, *D*, of Table 6.1. There are nine transactions in this database, that is, $|D| = 9$. We use Figure 6.2 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 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 $L1 \bowtie L1$ to generate a candidate set of 2-itemsets, *C*2. *C*2 consists of $(|L1|_{C2})$ 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 6.2.

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 the candidate 3-itemsets, *C*3, is detailed in Figure 6.3. From the join step, we first get $C3 = L2 \bowtie L2$ = {{I1, I2, I3}, {I1, I2, I5}, {I1, I3, I5},{I2, I3, I4}, {I2, I3, I5}, {I2, I4, I5}}.

(a) Join: $C3 = L2 \bowtie L2 = \{\{I1, I2\}, \{I1, I3\}, \{I1, I5\}, \{I2, I3\}, \{I2, I4\}, \{I2, I5\}\} \bowtie \{\{I1, I2\}, \{I1, I3\}, \{I1, I5\}, \{I2, I3\}, \{I2, I4\}, \{I2, I5\}\} = \{\{I1, I2, I3\}, \{I1, I2, I5\}, \{I1, I3, I5\}, \{I2, I3, I4\}, \{I2, I3, I5\}, \{I2, I4, I5\}\}$.

(b) Prune using the Apriori property: All nonempty subsets of a frequent itemset must also be frequent. Do any of the candidates have a subset that is not frequent?

Generation and pruning of Candidate 3-itemsets, C3, from L2 using the apriori property.

The 2-item subsets of {I1, I2, I3} are {I1, I2}, {I1, I3}, and {I2, I3}. All 2-item subsets of {I1, I2, I3} are members of *L*2. Therefore, keep {I1, I2, I3} in *C*3.
The 2-item subsets of {I1, I2, I5} are {I1, I2}, {I1, I5}, and {I2, I5}. All 2-item subsets of {I1, I2, I5} are members of *L*2. Therefore, keep {I1, I2, I5} in *C*3.
The 2-item subsets of {I1, I3, I5} are {I1, I3}, {I1, I5}, and {I3, I5}. {I3, I5} is not a member of *L*2, and so it is not frequent. Therefore, remove {I1, I3, I5} from *C*3.
The 2-item subsets of {I2, I3, I4} are {I2, I3}, {I2, I4}, and {I3, I4}. {I3, I4} is not a member of *L*2, and so it is not frequent. Therefore, remove {I2, I3, I4} from *C*3.
The 2-item subsets of {I2, I3, I5} are {I2, I3}, {I2, I5}, and {I3, I5}. {I3, I5} is not a member of *L*2, and so it is not frequent. Therefore, remove {I2, I3, I5} from *C*3.
The 2-item subsets of {I2, I4, I5} are {I2, I4}, {I2, I5}, and {I4, I5}. {I4, I5} is not a member of *L*2, and so it is not frequent. Therefore, remove {I2, I4, I5} from *C*3.
(c) Therefore, $C3 = \{\{I1, I2, I3\}, \{I1, I2, I5\}\}$ after pruning.

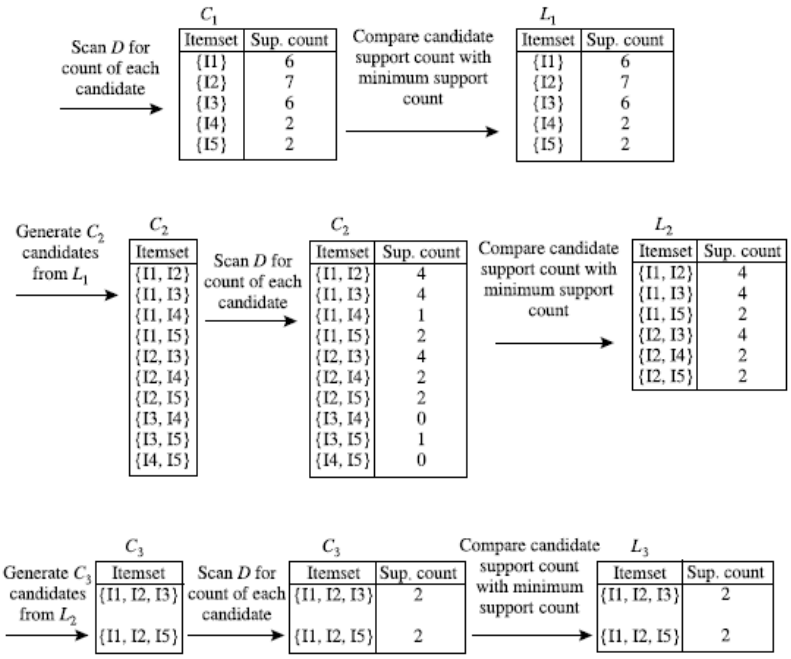


Figure 6.2 Generation of the candidate itemsets and frequent itemsets, where the minimum support count is 2.

7. The transactions in *D* are scanned to determine *L*3, consisting of those candidate 3-itemsets in *C*3 having minimum support (Figure 6.2).

8. The algorithm uses *L*3 1 *L*3 to generate a candidate set of 4-itemsets, *C*4. Although the join results in {{I1, I2, I3, I5}}, itemset {I1, I2, I3, I5} is pruned because its subset {I2, I3, I5} is not frequent. Thus, *C*4 = ϕ, and the algorithm terminates, having found all of the frequent itemsets.

Source: Data Mining Concepts and Techniques, 3rd Edition, Han, Kamber and Pei

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)   L1 = find_frequent_1-itemsets(D);
(2)   for (k = 2; Lk−1 ≠ φ; k++) {
(3)       Ck = apriori_gen(Lk−1);
(4)       for each transaction t ∈ D { // scan D for counts
(5)           Ct = subset(Ck, t); // get the subsets of t that are candidates
(6)           for each candidate c ∈ Ct
(7)               c.count++;
(8)       }
(9)       Lk = {c ∈ Ck | c.count ≥ min_sup}
(10)  }
(11)  return L = ∪k Lk;

procedure apriori_gen(Lk−1:frequent (k − 1)-itemsets)
(1)   for each itemset l1 ∈ Lk−1
(2)       for each itemset l2 ∈ Lk−1
(3)           if (l1[1] = l2[1]) ∧ (l1[2] = l2[2])
                  ∧... ∧ (l1[k − 2] = l2[k − 2]) ∧ (l1[k − 1] < l2[k − 1]) then {
(4)               c = l1 ⋈ l2; // join step: generate candidates
(5)               if has_infrequent_subset(c, Lk−1) then
(6)                   delete c; // prune step: remove unfruitful candidate
(7)               else add c to Ck;
(8)           }
(9)   return Ck;

procedure has_infrequent_subset(c: candidate k-itemset;
          Lk−1: frequent (k − 1)-itemsets); // use prior knowledge
(1)   for each (k − 1)-subset s of c
(2)       if s ∉ Lk−1 then
(3)           return TRUE;
(4)   return FALSE;
```

Figure 6.4 Apriori algorithm for discovering frequent itemsets for mining Boolean association rules.

Generating Association Rules from Frequent Itemsets

Based on this equation, association rules can be generated as follows:

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

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

Example 6.4 Generating association rules. Let's try an example based on the transactional data for *AllElectronics* shown before in Table 6.1. The data contain frequent itemset X = {I1, I2,I5}.What are the association rules that can be generated from X? The nonempty subsets of X

are {I1, I2}, {I1, I5}, {I2, I5}, {I1}, {I2}, and {I5}. The resulting association rules are as shown below, each listed with its confidence:

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

If the minimum confidence threshold is, say, 70%, then only the second, third, and last rules are output, because these are the only ones generated that are **strong**.

Improving the Efficiency of Apriori

"How can we further improve the efficiency of Apriori-based mining?"

Many variations of the Apriori algorithm have been proposed that focus on improving the efficiency of the original algorithm. They are as follows

Hash-based technique (hashing itemsets into corresponding buckets): A hash-based technique can be used to reduce the size of the candidate *k*-itemsets, *Ck*, for $k > 1$. For example, when scanning each transaction in the database to generate the frequent 1-itemsets, *L*1, we can generate all the 2-itemsets for each transaction, hash (i.e., map) them into the different *buckets* of a *hash table* structure, and increase the corresponding bucket counts. A 2-itemset with a corresponding bucket count in the hash table that is below the support threshold cannot be frequent and thus should be removed from the candidate set.

Create hash table H_2 using hash function $h(x, y) = ((order\ of\ x) \times 10 + (order\ of\ y)) \bmod 7$

H_2

bucket address	0	1	2	3	4	5	6
bucket count	2	2	4	2	2	4	4
bucket contents	{I1, I4} {I3, I5}	{I1, I5} {I1, I5}	{I2, I3} {I2, I3} {I2, I3} {I2, I3}	{I2, I4} {I2, I4}	{I2, I5} {I2, I5}	{I1, I2} {I1, I2} {I1, I2} {I1, I2}	{I1, I3} {I1, I3} {I1, I3} {I1, I3}

Figure 6.5 Hash table, H_2, for candidate 2-itemsets. This hash table was generated by scanning Table 6.1's transactions while determining L_1. If the minimum support count is, say, 3, then the itemsets in buckets 0, 1, 3, and 4 cannot be frequent and so they should not be included in C_2.

Transaction reduction (reducing the number of transactions scanned in future iterations): A transaction that does not contain any frequent *k*-itemsets cannot contain any frequent (*k* C1)-itemsets. Therefore, such a transaction can be marked or removed from further consideration because subsequent database scans for *j*-itemsets, where $j > k$, will not need to consider such a transaction.

Partitioning (partitioning the data to find candidate itemsets): A partitioning technique can be used that requires just two database scans to mine the frequent itemsets (Figure 6.6). It consists

of two phases. In phase I, the algorithm divides the transactions of *D* into *n* nonoverlapping partitions.
In phase II, a second scan of *D* is conducted in which the actual support of each candidate is assessed to determine the global frequent itemsets. Partition size and the number of partitions are set so that each partition can fit into main memory and therefore be read only once in each phase.

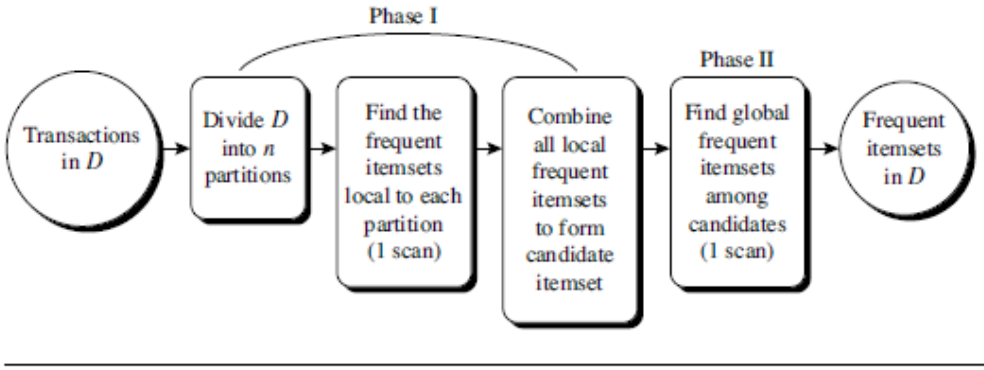


Figure 6.6 Mining by partitioning the data.

Sampling (mining on a subset of the given data): The basic idea of the sampling approach is to pick a random sample *S* of the given data *D*, and then search for frequent itemsets in *S* instead of *D*.

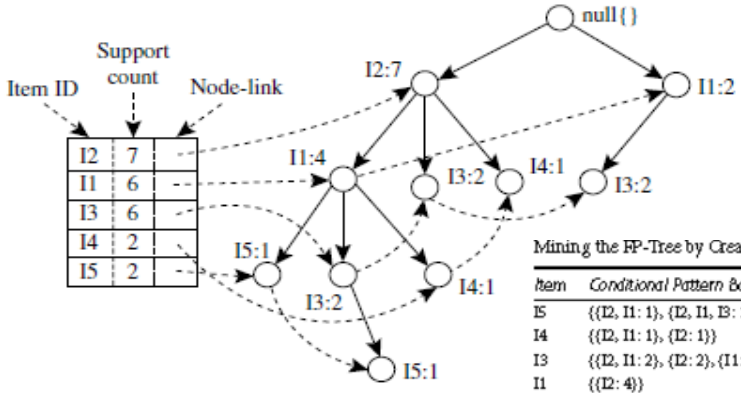
Dynamic itemset counting (adding candidate itemsets at different points during a scan): A dynamic itemset counting technique was proposed in which the database is partitioned into blocks marked by start points. In this variation, new candidate itemsets can be added at any start point, unlike in Apriori, which determines new candidate itemsets only immediately before each complete database scan.

A Pattern-Growth Approach for Mining Frequent Itemsets
"Can we design a method that mines the complete set of frequent itemsets without such a costly candidate generation process?" 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 conditional database each associated with one frequent item or pattern and mines each database seperatly.

Example 6.5 FP-growth (finding frequent itemsets without candidate generation). We reexamine the mining of transaction database, *D*, of Table 6.1 in Example 6.3 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. 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 to

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 to <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.



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}

Figure 6.7 An FP-tree registers compressed, frequent pattern information.

Mining Frequent Itemsets Using the Vertical Data Format

Both the Apriori and FP-growth methods mine frequent patterns from a set of transactions in *TID-itemset* format (i.e., {*TID* : *itemset*}), where *TID* is a transaction ID and *itemset* is the set of items bought in transaction *TID*. This is known as the **horizontal data format**. Alternatively, data can be presented in *item-TID set* format (i.e., {*item* : *TID set*}), where *item* is an item name, and *TID set* is the set of transaction identifiers containing the item. This is known as the **vertical data format** [**Eclat** (Equivalence Class Transformation) algorithm].

Example 6.6 Mining frequent itemsets using the vertical data format. Consider the horizontal data format of the transaction database, *D*, of Table 6.1 in Example 6.3. This can be transformed into the vertical data format shown in Table 6.3 by scanning the data set once.

Mining can be performed on this data set by intersecting the TID sets of every pair of frequent single items. The minimum support count is 2. Because every single item is frequent in Table 6.3, there are 10 intersections performed in total, which lead to eight nonempty 2-itemsets, as shown in Table 6.4. Notice that because the itemsets {I1, I4} and {I3, I5} each contain only one transaction, they do not belong to the set of frequent 2-itemsets. Based on the Apriori property, a given 3-itemset is a candidate 3-itemset only if every one of its 2-itemset subsets is frequent. The candidate generation process here will generate only two 3-itemsets: {I1, I2, I3} and {I1, I2, I5}. By intersecting the TID sets of any two corresponding 2-itemsets of these candidate 3-itemsets, it derives only two frequent 3-itemsets: {I1, I2, I3: 2} and {I1, I2, I5: 2}.

Table 6.3 The Vertical Data Format of the Transaction Data Set D of Table 6.1

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

Table 6.4 2-Itemsets in Vertical Data Format

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

Table 6.5 3-Itemsets in Vertical Data Format

itemset	TID_set
{I1, I2, I3}	{T800, T900}
{I1, I2, I5}	{T100, T800}

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

Mining Closed and Max Patterns

"How can we mine closed frequent itemsets?"

A naïve approach would be to first mine the complete set of frequent itemsets and then remove every frequent itemset that is a proper subset of, and carries the same support as, an existing frequent itemset. However, this is quite costly.

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

1. **Itemmerging.**
2. **Sub-itemset pruning.**
3. **Item skipping.**

1. **Itemmerging**: *If every transaction containing a frequent itemset X also contains an itemset Y but not any proper superset of Y, then X* $\cup$ *Y forms a frequent closed itemset and there is no need to search for any itemset containing X but no Y.*
   For example, in Table 6.2 of Example 6.5, the projected conditional database for prefix itemset {I5:2} is {{I2, I1}, {I2, I1, I3}}, from which we can see that each of its transactions contains itemset {I2, I1} but no proper superset of {I2, I1}. Itemset {I2, I1} can be merged with {I5} to form the closed itemset, {I5, I2, I1: 2}, and we do not need to mine for closed itemsets that contain I5 but not {I2, I1}.

2. **Sub-itemset pruning**: *If a frequent itemset* X *is a proper subset of an already found frequent closed itemset* Y *and* support count(X) = support count(Y)*, then* X *and all of* X*'s descendants in the set enumeration tree cannot be frequent closed itemsets and thus can be pruned.*
   suppose a transaction database has only two transactions: {($a1$, $a2$, : : : , $a100$), ($a1$, $a2$, : : : , $a50$)}, and the minimum support count is *min sup* = 2. The projection on the first item, $a1$, derives the frequent itemset, {$a1$, $a2$, : : : , $a50$: 2}, based on the *itemset merging* optimization. Because *support*({$a2$}) = *support*({$a1$, $a2$, : : : , $a50$}) = 2, and {$a2$} is a proper subset of {$a1$, $a2$, : : : , $a50$}, there is no need to examine $a2$ and its projected database. Similar pruning can be done for $a3$,….. , $a50$ as well. Thus, the mining of closed frequent itemsets in this data set terminates after mining $a1$'s projected database.
3. **Item skipping**: *In the depth-first mining of closed itemsets, at each level, there will be a prefix itemset* X *associated with a header table and a projected database. If a local frequent item* p *has the same support in several header tables at different levels, we can safely prune* p *from the header tables at higher levels*.
   Consider, for example, the previous transaction database having only two transactions: {($a1$, $a2$, : : : , $a100$), ($a1$, $a2$, : : : , $a50$)}, where *min sup* = 2. Because $a2$ in $a1$'s projected database has the same support as $a2$ in the global header table, $a2$ can be pruned from the global header table. Similar pruning can be done for $a3$, …. , $a50$. There is no need to mine anything more after mining $a1$'s projected database.

Which Patterns Are Interesting?—Pattern Evaluation Methods

a major bottleneck association rule mining. many of the rules generated are still not interesting to the users. Unfortunately, this is especially true *when mining at low support thresholds or mining for long patterns*.

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

Strong Rules Are Not Necessarily Interesting

"How can we tell which strong association rules are really interesting?"

Example 6.7 A misleading "strong" association rule. 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:

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

It is a strong association rule and would therefore be reported, since its support value of 4000/10,000 = 40% and confidence value of 4000/6000 = 66% satisfy the minimum support and minimum confidence thresholds, respectively. However, this rule is misleading because the

probability of purchasing videos is 75%, which is even larger than 66%. In fact, computer games and videos are negatively associated because the purchase of one of these items actually decreases the likelihood of purchasing the other.

the confidence of a rule $A \Rightarrow B$ can be deceiving. It does not measure the *real strength* (or lack of strength) of the *correlation* and *implication* between *A* and *B*.

From Association Analysis to Correlation Analysis

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*.

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

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

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

In other words, **lift** assesses the degree to which the occurrence of one "lifts" the occurrence of the other.

Example 6.8 Correlation analysis using lift.

From the table, we can see that the probability of purchasing a computer game is *P.({game})* = 0.60, the probability of purchasing a video is *P({video})* = 0.75, and the probability of purchasing both is *P({game, video})*= 0.40. By Eq. **lift**, the lift of given Association rule is *P({game, video})/(P({game}) *P.({video}))* = 0.40/(0.60*0.75) = 0.89. Because this value is less than 1, there is a negative correlation between the occurrence of {*game}* and {*video}*.

The second correlation measure that we study is the χ^2 measure

Source: Data Mining Concepts and Techniques, 3rd Edition, Han, Kamber and Pei

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

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

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

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

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

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

Because the χ^2 value is greater than 1, and the observed value of the slot (*game*, *video*) = 4000, which is less than the expected value of 4500, *buying game* and *buying video* are *negatively correlated*.

A Comparison of Pattern Evaluation Measures

How effective are the measures above discussed so far? Should we also consider other alternatives?

Yes- four such measures: *all confidence*, *max confidence*, *Kulczynski*, and *cosine*.

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

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

where $max\{sup(A), sup(B)\}$ is the maximum support of the itemsets *A* and *B*.

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

$$max\_conf(A, B) = max\{P(A \mid B), P(B \mid A)\}.$$

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

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

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

It can be viewed as an average of two confidence measures. That is, it is the average of two conditional probabilities: the probability of itemset *B* given itemset *A*, and the probability of itemset *A* given itemset *B*.

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

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

The *cosine* measure can be viewed as a *harmonized lift* measure: The two formulae are similar except that for cosine, the *square root* is taken on the product of the probabilities of *A* and *B*.
"Which is the best in assessing the discovered pattern relationships?"
To answer this question, consider the following example

Table 6.8 2×2 Contingency Table for Two Items

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

Comparison of six pattern evaluation measures on typical data sets. The relationships between the purchases of two items, *milk* and *coffee*, can be examined by summarizing their purchase history in Table 6.8, a 2_2 contingency table, where an entry such as *mc* represents the number of transactions containing both milk and coffee.
Table 6.9 shows a set of transactional data sets with their corresponding contingency tables and the associated values for each of the six evaluation measures.
Kulc in conjunction with the imbalance ratio(IR) is best, among the given, to present pattern relationships among itemsets..

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

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

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

Pattern Mining in Multilevel, Multidimensional Space.
Mining Multilevel Associations

For many applications, strong associations discovered at high abstraction levels, though with high support, could be commonsense knowledge. We may want to drill down to find novel patterns at more detailed levels.

Example 7.1 Miningmultilevel association rules. A concept hierarchy defines a sequence of mappings from a set of low-level concepts to a higher-level, more general concept set. Data can be generalized by replacing low-level concepts within the data by their corresponding higher-level concepts, or *ancestors*, from a concept hierarchy.

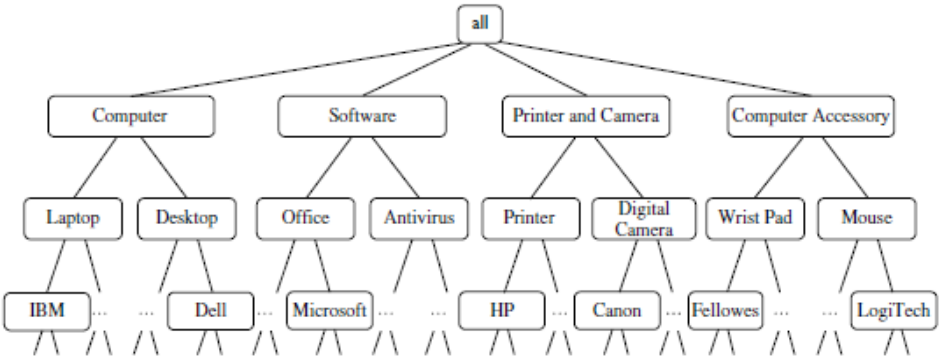


Figure 7.2 Concept hierarchy for *AllElectronics* computer items.

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

Table 7.1 Task-Relevant Data, *D*

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

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

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

A number of variations to this approach are described next, where each variation involves "playing" with the support threshold in a slightly different way. The variations are

Using uniform minimum support for all levels (referred to as **uniform support**): The same minimum support threshold is used when mining at each abstraction level. For example, in Figure 7.3, a minimum support threshold of 5% is used throughout (e.g., for mining from"*computer*" downward to "*laptop computer*"). Both "*computer*" and "*laptop computer*" are found to be frequent, whereas "*desktop computer*" is not. When a uniform minimum support threshold is used, the search procedure is simplified. The method is also simple in that users are required to specify only one minimum support threshold.

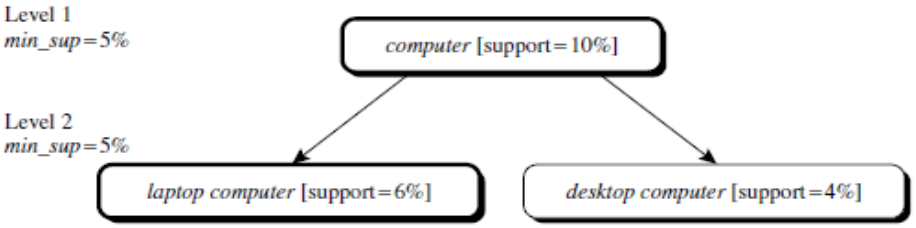


Figure 7.3 Multilevel mining with uniform support.

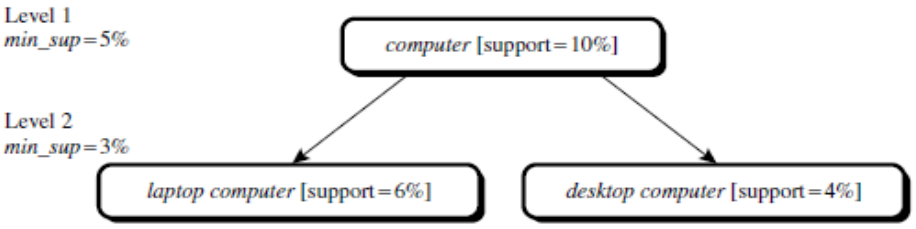


Figure 7.4 Multilevel mining with reduced support.

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

Using reduced minimum support at lower levels (referred to as **reduced support**): Each abstraction level has its own minimum support threshold. The deeper the abstraction level, the smaller the corresponding threshold. For example, in Figure 7.4, the minimum support thresholds for levels 1 and 2 are 5% and 3%, respectively.

Using item or group-based minimum support (referred to as group-based support): it is sometimes more desirable to set up user-specific, item, or group-based minimal support thresholds when mining multilevel rules.

For mining patterns with mixed items from groups with different support thresholds, usually the lowest support threshold among all the participating groups is taken as the support threshold in mining.

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

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

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

"*If Rules (7.4) and (7.5) are both mined, then how useful is Rule (7.5)? Does it really provide any novel information?*" If the latter, less general rule does not provide new information, then it should be removed. Let's look at how this may be determined. A rule *R*1 is an **ancestor** of a rule *R*2, if *R*1 can be obtained by replacing the items in *R*2 by their ancestors in a concept hierarchy. For example, Rule (7.4) is an ancestor of Rule (7.5) because "*laptop computer*" is an ancestor of "*Dell laptop computer.*" Based on this 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.

Mining Multidimensional Associations

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

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

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

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

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

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

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

Database attributes can be nominal or quantitative. The values of **nominal** (or categorical) attributes are "names of things." Nominal attributes have a finite number of possible values, with no ordering among the values (e.g., *occupation*, *brand*, *color*). **Quantitative** attributes are numeric and have an implicit ordering among values (e.g., *age*, *income*, *price*).

In the first approach, *quantitative attributes are discretized using predefined concept hierarchies*. This discretization occurs before mining. For instance, a concept hierarchy for *income* may be used to replace the original numeric values of this attribute by interval labels such as "0..20K," "21K..30K," "31K..40K," and so on.

In the second approach, *quantitative attributes are discretized or clustered into "bins" based on the data distribution*. These bins may be further combined during the mining process. The discretization process is *dynamic* and established so as to satisfy some mining criteria such as maximizing the confidence of the rules mined.

Mining Quantitative Association Rules

three methods that can help overcome this difficulty to discover novel association relationships:

(1) a data cube method,

(2) a clustering-based method, and

(3) a statistical analysis method to uncover exceptional behaviors.

Data Cube–Based Mining of Quantitative Associations

Alternatively, the transformed multidimensional data may be used to construct a *data cube*. Data cubes are well suited for the mining of multidimensional association rules: They store aggregates (e.g., counts) in multidimensional space, which is essential for computing the support and confidence of multidimensional association rules.

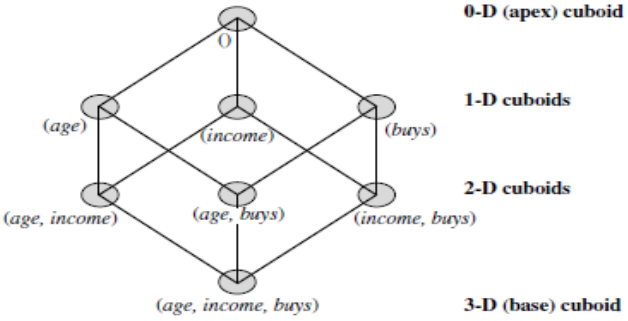


Figure 7.5 Lattice of cuboids, making up a 3-D data cube. Each cuboid represents a different group-by. The base cuboid contains the three predicates *age*, *income*, and *buys*.

Mining Clustering-Based Quantitative Associations

Besides using discretization-based or data cube–based data sets to generate quantitative association rules, we can also generate *quantitative association rules* by clustering data in the quantitative dimensions.

A typical top-down approach for finding clustering-based quantitative frequent patterns is as follows. For each quantitative dimension, a standard clustering algorithm (e.g., *k*-means or a density-based clustering algorithm, as described in Chapter 10) can be applied to find clusters in this dimension that satisfy the minimum support threshold. For each cluster, we then examine the 2-D spaces generated by combining the cluster with a cluster or nominal value of another dimension to see if such a combination passes

the minimum support threshold. If it does, we continue to search for clusters in this 2-D region and progress to even higher-dimensional combinations. The Apriori pruning still applies in this process: If, at any point, the support of a combination does not have minimum support, its further partitioning or combination with other dimensions cannot have minimum support either.

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

Using Statistical Theory to Disclose Exceptional Behavior

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

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

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

That is, Rule (7.9) is only accepted if a statistical test (in this case, a Z-test) confirms that with high confidence it can be inferred that the mean wage of the female population is indeed lower than the mean wage of the rest of the population.

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

$$population\_subset \Rightarrow \textbf{mean}\_of\_values\_for\_the\_subset, \qquad (7.10)$$

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

Mining Rare Patterns and Negative Patterns

Sometimes, however, it is interesting to find patterns that are rare instead of frequent, or patterns that reflect a negative correlation between items. These patterns are respectively referred to as rare patterns and negative patterns.

Example. Rare patterns and negative patterns. In jewelry sales data, sales of diamond watches are rare; however, patterns involving the selling of diamond watches could be interesting. In supermarket data, if we find that customers frequently buy Coca-Cola Classic or Diet Coke but not both, then buying Coca-Cola Classic and buying Diet Coke together is considered a negative (correlated) pattern. In car sales data, a dealer sells a few fuel-thirsty vehicles (e.g., SUVs) to a given customer, and then later sells hybrid mini-cars to the same customer. Even though buying SUVs and buying hybrid mini-cars may be negatively correlated events, it can be interesting to discover and examine such exceptional cases.

An **infrequent** (or **rare**) **pattern** is a pattern with a frequency support that is *below* (or *far below*) a user-specified minimum support threshold.

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

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

problem with the definition, however, is that it is not *null-invariant*. That is, its value can be misleadingly influenced by null transactions, where a *null-transaction* is a transaction that does not contain any of the itemsets being examined.

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

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

Is this measure null-invariant?

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

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

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

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

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

As a third alternative, consider Definition 7.3, which is based on the Kulczynski measure (i.e., the average of conditional probabilities).

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