UNIT –II

Mining Frequent Patterns, Association and Correlations

Topics covered:

- Market Basket Analysis
- Association Rule Mining
- Frequent Item set mining methods
- Mining various kinds of Association rule
- Constraint-Based frequent pattern mining

Introduction :

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. Moreover, it helps in data classification, clustering, and other data mining tasks. Thus, frequent pattern mining has become an important data mining task and a focused theme in data mining research.

2.1 Basic Concepts

Frequent pattern mining searches for recurring relationships in a given data set.

Market Basket Analysis: A Motivating Example

Frequent itemset mining leads to the discovery of associations and correlations among items in large transactional or relational data sets. With massive amounts of data continuously being collected and stored, many industries are becoming interested in mining such patterns from their databases. The discovery of interesting correlation relation-ships among huge amounts of business transaction records can help in many business decision-making processes such as catalog design, cross-marketing, and customer shopping behavior 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 below). 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 tripto the supermarket? This information can lead to increased sales by helping retailers do selective marketing and plan their shelf space.

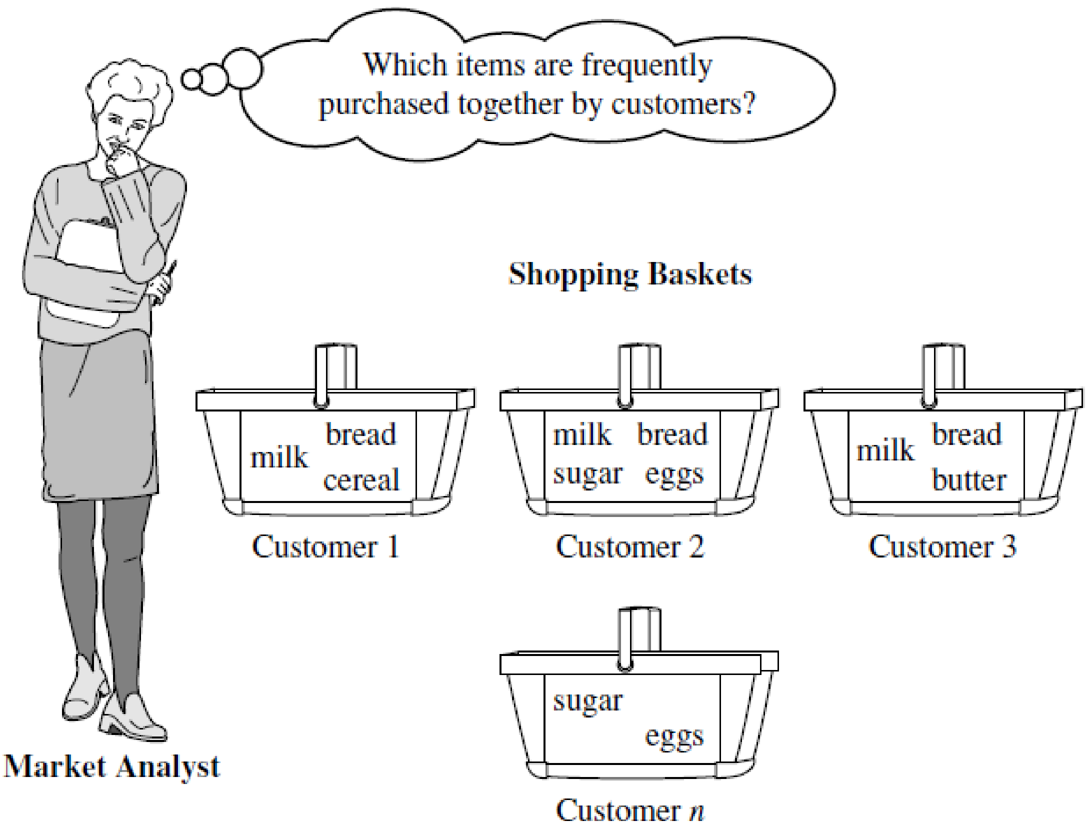


Figure 6.1 Market basket analysis.

Example Market basket analysis.

Suppose, as manager of an *AllElectronics* branch, you would like to learn more about the buying habits of your customers. Specifically, you wonder, *"Which groups or sets of items are customers likely to purchase on a given trip to the store?"*

To answer the question, market basket analysis may be performed on the retail data of customer transactions at your store. We can then use the results to plan marketing or advertising strategies, or in the design of a new catalog. For instance, market basket analysis may help us design different store layouts. In one strategy, items that are frequently purchased together can be placed in proximity to further encourage the combined sale of such items.

If customers who purchase computers also tend to buy antivirus software at the same time, then placing the hardware display close to the software display may help increase the sales of both items.

In an alternative strategy, placing hardware and software at opposite ends of the store may attract customers who purchase such items to pick up other items along the way.

For instance, after deciding on an expensive computer, a customer may observe security systems for sale while heading toward the software display to purchase antivirus software, and may decide to purchase a home security system as well. Market basket analysis can also help retailers plan which items to put on sale at reduced prices. If customers tend to purchase computers and printers together, then having a sale on printers may encourage the sale of printers *as well as* computers.

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 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% 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.

2.1.2 Frequent Itemsets, Closed Itemsets, and Association Rules

Let I ={ *I*1, *I*2, … , *Im }* be an itemset. Let *D*, the task-relevant data. Each transaction is associated with an identifier, called a *TID*. Let *A* be a set of items. An association rule is an implication of the form

$A \Rightarrow B$, where $A \subset \mathcal{I}$, $B \subset \mathcal{I}$, $A \neq \emptyset$, $B \neq \emptyset$, and $A \cap B = \phi$. The rule $A \Rightarrow B$ holds in the transaction set *D* with **support** *s*, where *s* is the percentage of transactions in *D* that contain *A* *B* (i.e., the *union* of sets *A* and *B* say, or, both *A* and *B*). This is taken to be the probability, *P(A B)*. The rule *A* *B* has **confidence** *c* in the transaction set *D*, where *c* is the percentage of transactions in *D* containing *A* that also contain *B*. This is taken to be the conditional probability,
P(B | A). That is,

$$support(A \Rightarrow B) = P(A \cup B) \qquad (6.2)$$

$$confidence(A \Rightarrow B) = P(B|A). \qquad (6.3)$$

Rules that satisfy both a minimum support threshold (*min sup*) and a minimum confidence threshold (*min conf*) are called **strong**. By convention, we write support and confidence values so as to occur between 0% and 100%, rather than 0 to 1.0.

A set of items is referred to as an **itemset**.2 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. Note that the itemset support defined in Eq. (6.2) is sometimes referred to as *relative support*, whereas the occurrence frequency is called the **absolute support**. If the relative support of an itemset I satisfies a prespecified **minimum support threshold** (i.e., the absolute support of I satisfies the corresponding **minimum support count threshold**), then I is a **frequent** itemset. The set of frequent k-itemsets is commonly denoted by L_k.
From Eq. (6.3), we have

$$confidence(A \Rightarrow B) = P(B|A) = \frac{support(A \cup B)}{support(A)} = \frac{support\_count(A \cup B)}{support\_count(A)}. \qquad (6.4)$$

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 $X \quad Y$ and Y is frequent in D.

Example : Closed and maximal frequent itemsets. Suppose that a transaction database has only two transactions: { < $a1$, $a2$, … , $a100$ >; <$a1$, $a2$, … , $a50$> }. Let the minimum support count threshold be *min_ sup* = 1. We find two closed frequent itemsets and their support counts, that is,
C = { {$a1$, $a2$, … , $a100$} : 1 ; {$a1$, $a2$, … , $a50$} : 2 }. There is only one maximal frequent itemset:
M = {{ $a1$, $a2$, … , $a100$} : 1}. Notice that we cannot include {$a1$, $a2$, … , $a50$} as a maximal frequent itemset because it has a frequent superset, {$a1$, $a2$, … , $a100$}. 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) {$a2$, $a45$: 2} since {$a2$, $a45$} is a sub-itemset of the itemset {$a1$, $a2$, … , $a50$: 2}; and (2){$a8$, $a55$: 1} since {$a8$, $a55$} is not a sub-itemset of the previous itemset but of the itemset {$a1$, $a2$, … , $a100$: 1}. However, from the maximal frequent itemset, we can only assert that both itemsets ({$a2$, $a45$} and {$a8$, $a55$}) are frequent, but we cannot assert their actual support counts.

2.2 Frequent Itemset Mining Methods

Apriori, the basic algorithm for finding frequent itemsets

2.2.1 Apriori Algorithm: Finding Frequent Itemsets by Confined Candidate Generation

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

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

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). For efficient implementation, 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 $l_i[1] < l_i[2] < \cdots < l_i[k-1]$. 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 $(l_1[1] = l_2[1]) \wedge (l_1[2] = l_2[2]) \wedge \cdots \wedge (l_1[k-2] = l_2[k-2]) \wedge (l_1[k-1] < l_2[k-1])$. 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 database scan 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 *Ck*. This **subset testing** can be done quickly by maintaining a hash tree of all frequent itemsets.

Example Apriori. Let's consider *AllElectronics* transaction database, *D*, of Table shown below. There are nine transactions in this database, that is, |*D*|= 9.

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

1. In the first iteration of the algorithm, each item is a member of the set of candidate 1-itemsets, $C1$. 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, $L1$, can then be determined. It consists of the candidate 1-itemsets satisfying minimum support. In our example, all of the candidates in $C1$ satisfy minimum support.
3. To discover the set of frequent 2-itemsets, $L2$, the algorithm uses the join $L1$ $L1$ to generate a candidate set of 2-itemset, C2. No candidates are removed from $C2$ during the prune step because each subset of the candidates is also frequent.

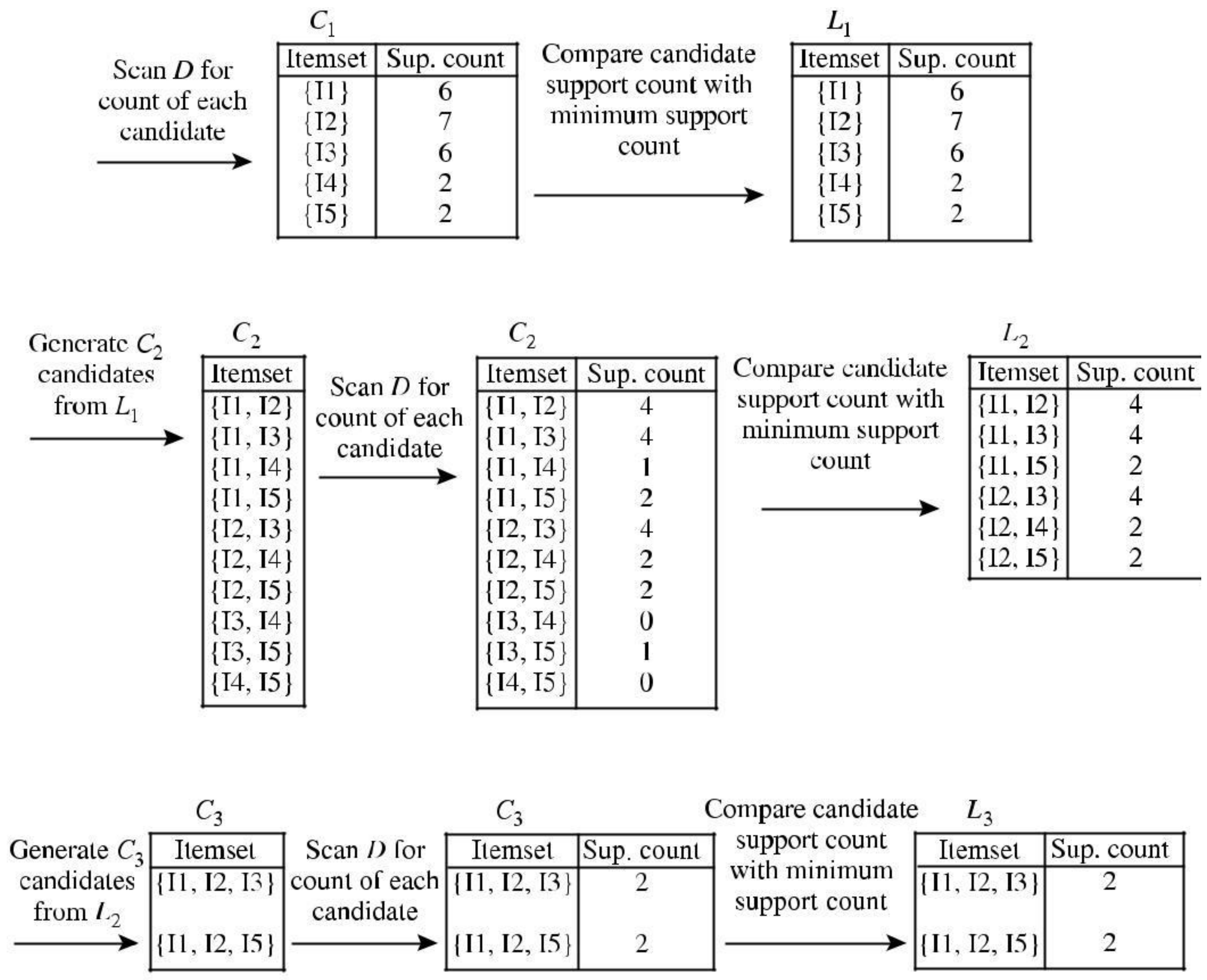
4. Next, the transactions in D are scanned and the support count of each candidate itemset in $C2$ is accumulated, as shown in the middle table of the second row.

5. The set of frequent 2-itemsets, $L2$, is then determined, consisting of those candidate 2-itemsets in $C2$ having minimum support.

6. The generation of the set of the candidate 3-itemsets, $C3$, is given in Fig 6.3. From the join step, we first get $C_3 = L_2 \quad L_2$ = { {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 subsetsof a frequent itemset must also be frequent, we can determine that the four lattercandidates cannot possibly be frequent.We therefore remove them from $C3$, therebysaving the effort of unnecessarily obtaining their counts during the subsequent scanof D to determine $L3$. Note that when given a candidate k-itemset, we only need tocheck if its (k -1)-subsets are frequent since the Apriori algorithm uses a level-wise search strategy. The resulting pruned version of $C3$ is shown in the first table of the bottom row of Figure 6.2.

7. The transactions in D are scanned to determine $L3$, consisting of those candidate 3-itemsets in $C3$ having minimum support (Figure 6.2).

8. The algorithm uses $L3$ $L3$ to generate a candidate set of 4-itemsets, C_4. Although the join results in {{I1, I2, I3, I5}}, itemset {I1, I2, I3, I5} is pruned because its subset {I2, I3, I5 } is not frequent. Thus, C_4 = , and the algorithm terminates, having found all of the frequent itemsets.


Scan D for count of each candidate
Compare candidate support count with minimum support count
Generate C_2 candidates from L_1
Scan D for count of each candidate
Compare candidate support count with minimum support count
Generate C_3 candidates from L_2
Scan D for count of each candidate
Compare candidate support count with minimum support count

C_1

Itemset	Sup. count
{I1}	6
{I2}	7
{I3}	6
{I4}	2
{I5}	2

L_1

Itemset	Sup. count
{I1}	6
{I2}	7
{I3}	6
{I4}	2
{I5}	2

C_2

Itemset
{I1, I2}
{I1, I3}
{I1, I4}
{I1, I5}
{I2, I3}
{I2, I4}
{I2, I5}
{I3, I4}
{I3, I5}
{I4, I5}

C_2

Itemset	Sup. count
{I1, I2}	4
{I1, I3}	4
{I1, I4}	1
{I1, I5}	2
{I2, I3}	4
{I2, I4}	2
{I2, I5}	2
{I3, I4}	0
{I3, I5}	1
{I4, I5}	0

L_2

Itemset	Sup. count
{I1, I2}	4
{I1, I3}	4
{I1, I5}	2
{I2, I3}	4
{I2, I4}	2
{I2, I5}	2

C_3

Itemset
{I1, I2, I3}
{I1, I2, I5}

C_3

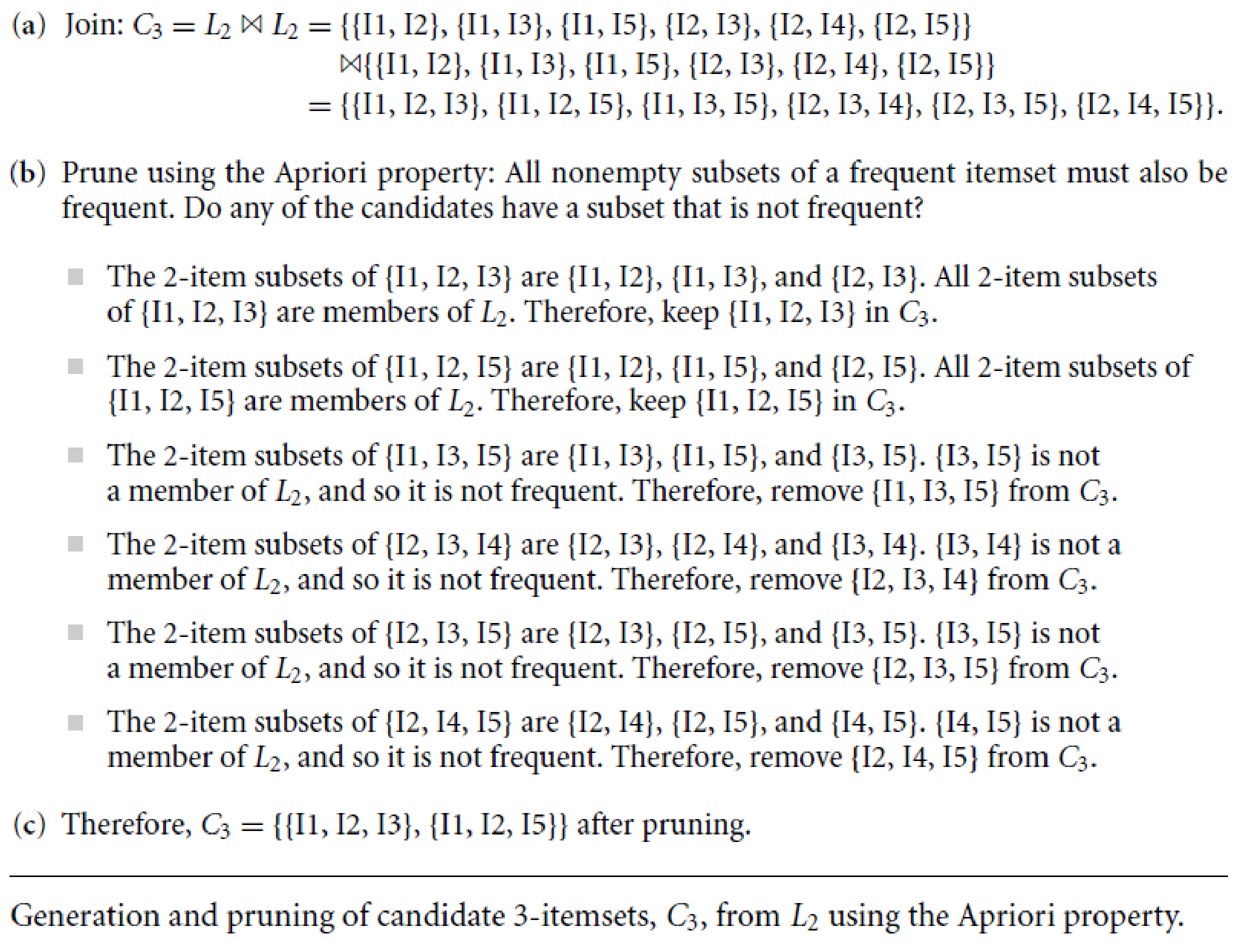
Itemset	Sup. count
{I1, I2, I3}	2
{I1, I2, I5}	2

L_3

Itemset	Sup. count
{I1, I2, I3}	2
{I1, I2, I5}	2

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

Fig:6.3



(a) Join: $C_3 = L_2 \bowtie L_2 = \{\{I1, I2\}, \{I1, I3\}, \{I1, I5\}, \{I2, I3\}, \{I2, I4\}, \{I2, I5\}\}$
$\bowtie \{\{I1, I2\}, \{I1, I3\}, \{I1, I5\}, \{I2, I3\}, \{I2, I4\}, \{I2, I5\}\}$
$= \{\{I1, I2, I3\}, \{I1, I2, I5\}, \{I1, I3, I5\}, \{I2, I3, I4\}, \{I2, I3, I5\}, \{I2, I4, I5\}\}$.

(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?

- 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, $C_3 = \{\{I1, I2, I3\}, \{I1, I2, I5\}\}$ after pruning.

Generation and pruning of candidate 3-itemsets, C_3, from L_2 using the Apriori property.

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

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;
```

2.2.2 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).

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

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

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

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

Example:6.4

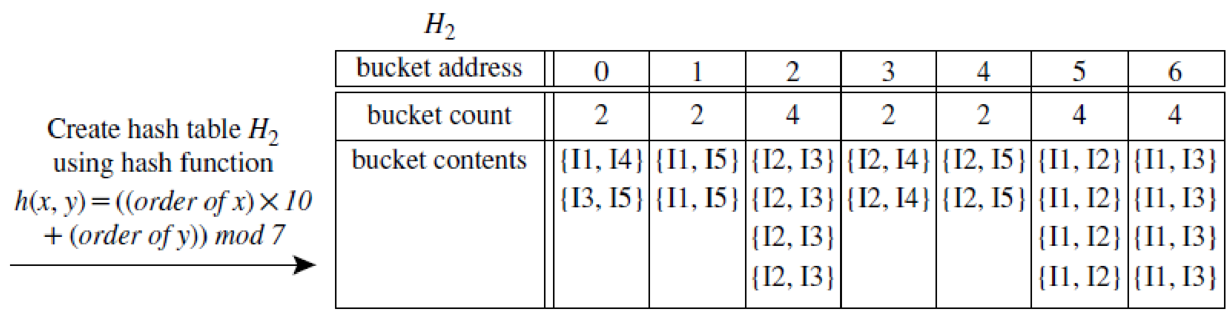
Generating association rules. Let's try an example based on the transactional data for *AllElectronics* shown before in Table 6.1. The data contain frequent itemset $X = \{I1, I2, I5\}$. What are the association rules that can be generated from X? The nonempty subsets of X are {I1, I2}, {I1, I5}, {I2,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

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

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

2.2.3 Improving the Efficiency of Apriori

Many variations of the Apriori algorithm have been proposed that focus on improving the efficiency of the original algorithm. Several of these variations are summarized as follows:



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

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}

Fig 6.5 Hash table, $H2$, for candidate 2-itemsets. This hash table was generated by scanning Table 6.1's transactions while determining $L1$. 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 $C2$.

1. **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, $L1$, 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. Such a hash-based technique may substantially reduce the number of candidate k-itemsets examined (especially when $k = 2$).

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 $(k + 1)$-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.

3. **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. It consists of two phases. In phase I, the algorithm divides the trans-actions of D into n nonoverlapping partitions. If the minimum relative support threshold for transactions in D is *min sup*, then the minimum support count for a partition is *min sup the number of transactions in that partition*. For each partition, all the *local frequent itemsets* (i.e., the itemsets frequent within the partition) are found.

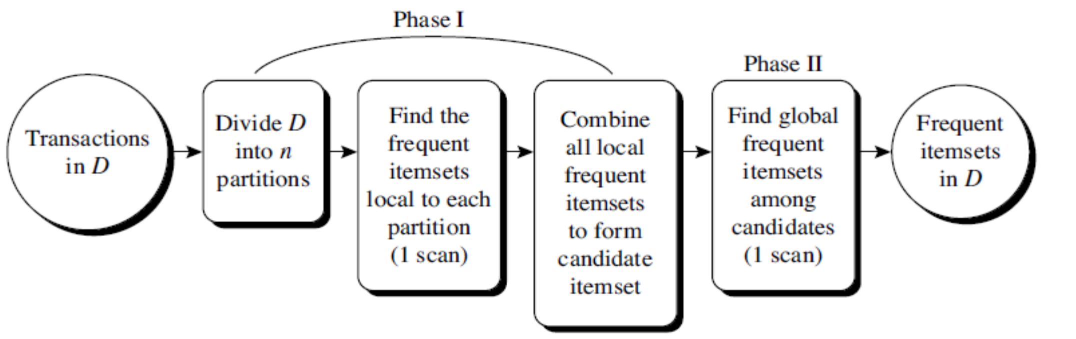

Phase I
Phase II
Transactions in D
Divide D into n partitions
Find the frequent itemsets local to each partition (1 scan)
Combine all local frequent itemsets to form candidate itemset
Find global frequent itemsets among candidates (1 scan)
Frequent itemsets in D

Figure 6.6 Mining by partitioning the data.

4. **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. In this way, we trade off some degree of accuracy against efficiency. The S sample size is such that the search for frequent itemsets in S can be done in main memory, and so only one scan of the transactions in S is required overall. Because we are searching for frequent itemsets in S rather than in D, it is possible that we will miss some of the global frequent itemsets.

5. **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. The tech-nique uses the count-so-far as the lower bound of the actual count. If the count-so-far passes the

minimum support, the itemset is added into the frequent itemset collection and can be used to generate longer candidates. This leads to fewer database scans than with Apriori for finding all the frequent itemsets.

2.2.4 A Pattern-Growth Approach for Mining Frequent Itemsets

The Apriori candidate generate-and-test method significantly reduces the size of candidate sets, leading to good performance gain. However, it can suffer from two nontrivial costs:

1. It may still need to generate a huge number of candidate sets. For example, if there are 10^4 frequent 1-itemsets, the Apriori algorithm will need to generate more than 10^7 candidate 2-itemsets.

2. It may need to repeatedly scan the whole database and check a large set of candidates by pattern matching. It is costly to go over each transaction in the database to determine the support of the candidate itemsets.

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 database separately. For each "pattern fragment," only its associated data sets need to be examined. Therefore, this approach may substantially reduce the size of the data sets to be searched, along with the "growth" of patterns being examined.

Example 6.5: FP-growth (finding frequent itemsets without candidate generation).

The first scan of the database is the same as Apriori, which derives the set of frequent items (1-itemsets) and their support counts (frequencies). Let the minimum support count be 2. The set of frequent items is sorted in the order of descending support count. This resulting set or *list* is denoted by *L*. Thus, we have *L* = ***{{*** I2: 7}, {I1: 6}, {I3: 6}, {I4: 2}, **{**I5: 2}}.

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

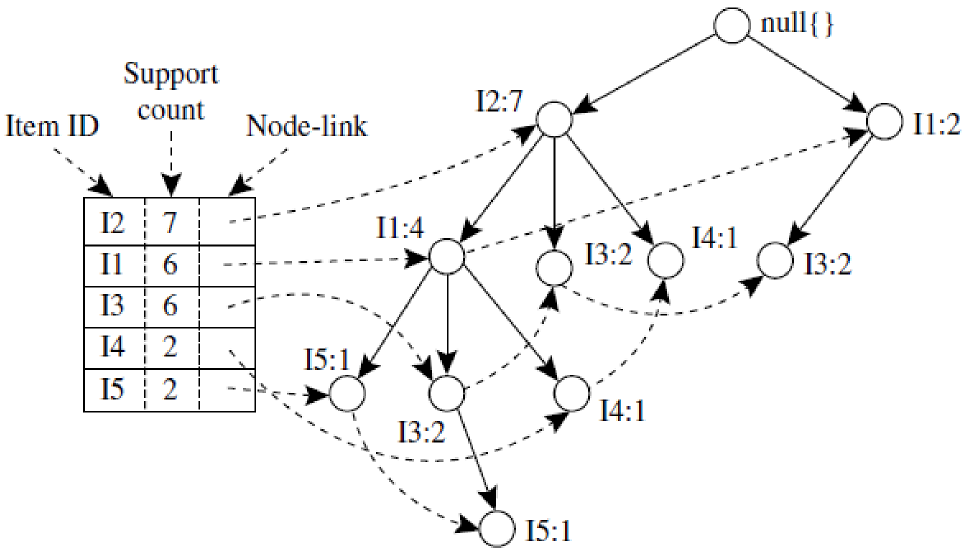


Figure 6.7 An FP-tree registers compressed, frequent pattern information.

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

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

Mining of the FP-tree is summarized in Table below 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 FP-tree branches of Figure above. (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. Using this conditional pattern base as a transaction database, we build an I5-conditional FP-tree, which contains only a single path, <I2: 2, I1: 2>; I3 is not included because its support count of 1 is less than the minimum support count. The single path generates all the combinations of frequent patterns: {I2, I5: 2}, {I1, I5: 2}, {I2, I1, I5: 2}.

For I4, its two prefix paths form the conditional pattern base, {{I2 I1: 1}, {I2: 1}}, which generates a single-node conditional FP-tree, <I2: 2>, and derives one frequent pattern, <I2, I4: 2>.

Table 6.2 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}

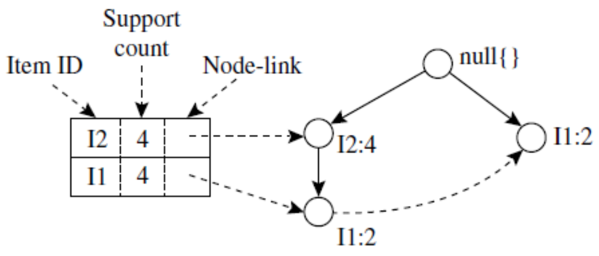


Figure 6.8 The conditional FP-tree associated with the conditional node I3.

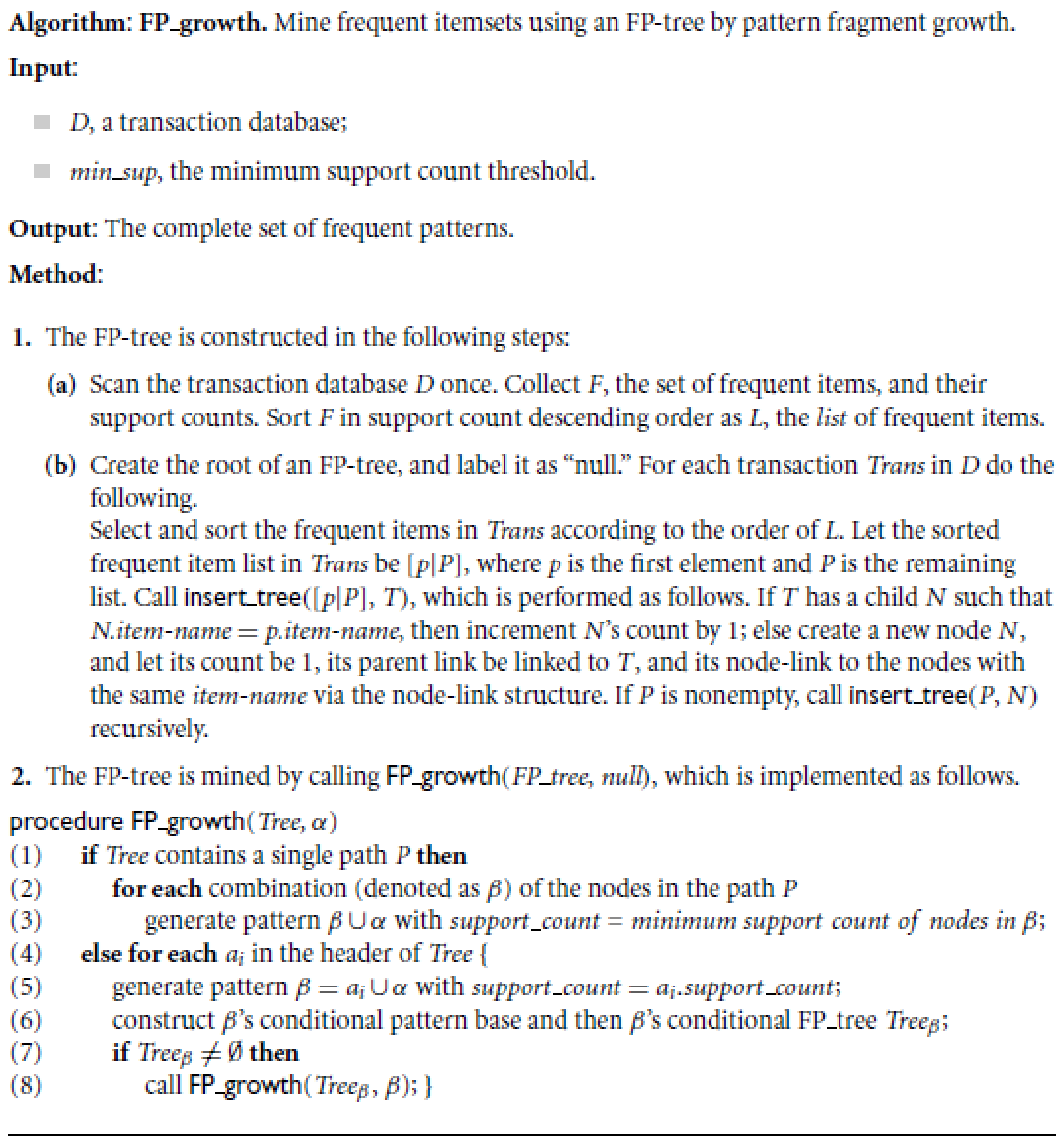
Similar to the preceding analysis, I3's conditional pattern base is {{ I2, I1: 2 }, { I2: 2 },
{I1: 2 } }. Its conditional FP-tree has two branches, <I2: 4, I1: 2 > and **<**I1: 2 >, as shown in Figure 6.8, which generates the set of patterns {{I2, I3: 4 }, {I1, I3: 4 }, {I 2, I1, I3: 2 }}. Finally, I1's conditional pattern base is {{I2: 4 }}, with an FP-tree that contains only one node, <I2: 4 >, which generates one frequent pattern, {I2, I1: 4}. This mining process is summarized in Figure 6.9.

The FP-growth method transforms the problem of finding long frequent patterns into searching for shorter ones in much smaller conditional databases recursively and then concatenating the suffix. It uses the least frequent items as a suffix, offering good selectivity. The method substantially reduces the search costs.

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

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

A study of the FP-growth method performance shows that it is efficient and scalable for mining both long and short frequent patterns, and is about an order of magnitude faster than the Apriori algorithm.



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

2.2.5 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**.
Frequent itemsets can also be mined efficiently using vertical data format, which is the essence of the **Eclat** (Equivalence Class Transformation) algorithm.

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

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}

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 gen-erate only two 3-itemsets: **{**I1, I2, I3**}** and **{**I1, I2, I5**}**. By intersecting the TID sets of any two corresponding 2-itemsets of these candidate 3-itemsets, it derives Table 6.5, where there are only two frequent 3-itemsets:
{I1, I2, I3: 2**}** and **{**I1, I2, I5: 2**}**.

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

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

Besides taking advantage of the Apriori property in the generation of candidate (k + 1) -itemset from frequent k-itemsets, another merit of this method is that there is no need to scan the database to find the support of (k+ 1)-itemsets (for $k \geq 1$). 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.

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

2.2.6 Mining Closed and Max Patterns

In practice, it is more desirable to mine the set of closed frequent itemsets rather than the set of all frequent itemsets in most cases.

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

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

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

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

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

Similar to Example 6.2, suppose a transaction database has only two trans-actions: {<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}) D support({a1, a2, …, a50}) = 2, and {a2 } is a proper subset of {a1, a2, …, a50}, there is no need to examine a_2 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 a_1's projected database.

Item skipping: *In the depth-first mining of closed itemsets, at each level, there will be a prefix itemset* X *associated with a header table and a projected database. If a local frequent item* p *has the same support in several header tables at different levels, we can safely prune* p *from the header tables at higher levels*.

Consider, for example, the previous transaction database having only two trans-actions: { <*a*1, *a*2, …, *a*100>, <*a*1, *a*2, …, *a*50 >}, where *min sup* = 2. Because *a*2 in *a*1's projected database has the same support as *a*2 in the global header table, *a*2 can be pruned from the global header table. Similar pruning can be done for *a*3, …, *a*50. There is no need to mine anything more after mining *a*1's projected database.

Besides pruning the search space in the closed itemset mining process, another important optimization is to perform efficient checking of each newly derived frequent itemset to see whether it is closed. This is because the mining process cannot ensure that every generated frequent itemset is closed.

When a new frequent itemset is derived, it is necessary to perform two kinds of closure checking: (1) *superset checking*, which checks if this new frequent itemset is a superset of some already found closed itemsets with the same support, and (2) *subset checking*, which checks whether the newly found itemset is a subset of an already found closed itemset with the same support.

If we adopt the *item merging* pruning method under a divide-and-conquer frame-work, then the superset checking is actually built-in and there is no need to explicitly perform superset checking. This is because if a frequent itemset *X* ***[*** *Y* is found later than itemset *X*, and carries the same support as *X*, it must be in *X*'s projected database and must have been generated during itemset merging.

To assist in subset checking, a compressed **pattern-tree** can be constructed to main-tain the set of closed itemsets mined so far. The pattern-tree is similar in structure to the FP-tree except that all the closed itemsets found are stored explicitly in the corresponding tree branches. For efficient subset checking, we can use the following property: *If the current itemset* Sc *can be subsumed by another already found closed itemset* Sa*, then (1)* Sc *and* Sa *have the same support, (2) the length of* Sc *is smaller than that of* Sa*, and (3) all of the items in* Sc *are contained in* Sa*.*

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

This discussion illustrates methods for efficient mining of closed frequent itemsets. *"Can we extend these methods for efficient mining of maximal frequent itemsets?"* Because maximal frequent itemsets share many similarities with closed frequent itemsets, many of the optimization techniques developed here can be extended to mining maximal frequent itemsets.

2.3 Which Patterns Are Interesting?—Pattern Evaluation Methods

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

In this section, we first look at how even strong association rules can be uninteresting and misleading (Section 3.3.1). We then discuss how the support–confidence frame-work can be supplemented with additional interestingness measures based on *correlation analysis* (Section 3.3.2). Section 3.3.3 presents additional pattern evaluation measures. It then provides an overall comparison of all the measures discussed here. By the end, you will learn which pattern evaluation measures are most effective for the discovery of only interesting rules.

2.3.1 Strong Rules Are Not Necessarily Interesting

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

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

Example 6.7 A misleading "strong" association rule. Suppose we are interested in analyzing 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, "*computer games*")   *buys(X*, "*videos*") [*support = 40%*, *confidence= 66%*]. (6.6)

Rule (6.6) 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, Rule (6.6) is misleading because the probability of purchasing videos is 75%, which is even larger than 66%. In fact, computer games and videos are negatively associated because the purchase of one of these items actually decreases the likelihood of purchasing the other. Without fully understanding this phenomenon, we could easily make unwise business decisions based on Rule (6.6).

2.3.2 From Association Analysis to Correlation Analysis

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

A   *B* [*support*, *confidence*, *correlation*]. (6.7)

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

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

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

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

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

Example 6.8

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

From the table, we can see that the probability of purchasing a computer game is $P(\{game\}) = 0.60$, the probability of purchasing a video is $P(\{video\}) = 0.75$, and the probability of purchasing both is $P(\{game, video\}) = 0.40$. By Eq. (6.8), the lift of Rule (6.6) is $P(\{game, video\})/(P(\{game\}) \times P(\{video\})) = 0.40/(0.60 \times 0.75) = 0.89$. Because this value is less than 1, there is a negative correlation between the occurrence of {*game*} and {*video*}. The numerator is the likelihood of a customer purchasing both, while the denominator is what the likelihood would have been if the two purchases were completely independent. Such a negative correlation cannot be identified by a support–confidence framework. ■

The second correlation measure that we study is the χ2 measure .To compute the χ 2 value, we take the squared difference between the observed and expected value for a slot (*A* and *B* pair) in the contingency table, divided by the expected value.

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*. This is consistent with the conclusion derived from the analysis of the *lift* measure in Example 6.8. ■

2.3.3 A Comparison of Pattern Evaluation Measures

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

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

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

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

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

$$max\_conf(A, B) = max\{P(A \mid B), P(B \mid A)\}. \tag{6.10}$$

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

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

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

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

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

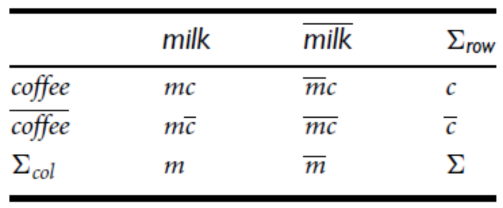
$$\begin{aligned} cosine(A, B) &= \frac{P(A \cup B)}{\sqrt{P(A) \times P(B)}} = \frac{sup(A \cup B)}{\sqrt{sup(A) \times sup(B)}} \\ &= \sqrt{P(A|B) \times P(B|A)}. \end{aligned} \tag{6.12}$$

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

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

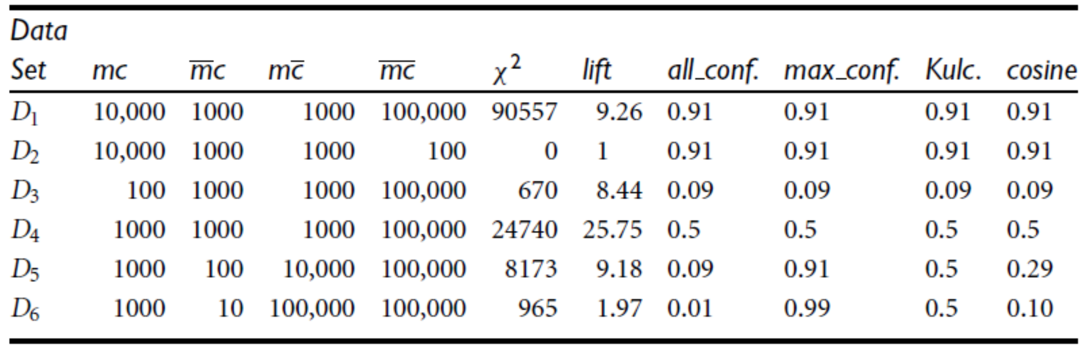
To know Which method is the best in assessing the discovered pattern relationships we examine their performance on some typical data sets.

Table 6.8 2 × 2 Contingency Table for Two Items



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

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



Data Set	mc	$\overline{m}c$	$m\overline{c}$	$\overline{mc}$	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

"Why are lift *and χ2 so poor at distinguishing pattern association relationships in the previous transactional data sets?"* To answer this, we have to consider the *nulltransactions*. A **null-transaction** is a transaction that does not contain any of the itemsets being examined. In our example, *mc* represents the number of null-transactions. *Lift* and χ2 have difficulty distinguishing interesting pattern association relationships because they are both strongly influenced by *mc*. Typically, the number of nulltransactions can outweigh the number of individual purchases because, for example,
many people may buy neither milk nor coffee. On the other hand, the other four measures are good indicators of interesting pattern associations because their definitions remove the influence of *mc* (i.e., they are not influenced by the number of null-transactions).

A measure is **null-invariant** if its value is free from the influence of null-transactions. Null-invariance is an important property for measuring association patterns in large transaction databases. Among the six discussed measures in this subsection, only *lift* and χ2 are not null- invariant measures.

"*Among the* all confidence, max confidence, Kulczynski, *and* cosine *measures, which is best at indicating interesting pattern relationships*?"
To answer this question, we introduce the **imbalance ratio** (**IR**), which assesses the imbalance of two itemsets, *A* and *B*, in rule implications. It is defined as

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

where the numerator is the absolute value of the difference between the support of the itemsets *A* and *B*, and the denominator is the number of transactions containing *A* or *B*. If the two directional implications between *A* and *B* are the same, then *IR(A,B)* will be zero. Otherwise, the larger the difference between the two, the larger the imbalance ratio. This ratio is independent of the number of null-transactions and independent of the total number of transactions.

In summary, the use of only support and confidence measures to mine associations may generate a large number of rules, many of which can be uninteresting to users. Instead, we can augment the support–confidence framework with a pattern interestingness measure, which helps focus the mining toward rules with strong pattern relationships. The added measure substantially reduces the number of rules generated and leads to the discovery of more meaningful rules. Among the four null-invariant measures studied here, namely *all confidence*, *max confidence*, *Kulc*, and *cosine*, we recommend using *Kulc* in conjunction with the imbalance ratio.