Module 5

Association Rules Mining: Concepts, Apriori and FP-Growth Algorithm. Cluster Analysis: Introduction, Concepts, Types of data in cluster analysis, Categorization of clustering methods. Partitioning method: K-Means and K-Medoid Clustering

Association Rules Mining: Concepts, Apriori and FP-Growth Algorithm.

Association rule mining finds interesting association or correlation relationships among a large set of data items. With massive amounts of data continuously being collected and stored in databases, many industries are becoming interested in mining association rules from their databases. For example, the discovery of interesting association relationships among huge amounts of business transaction records can help catalog design, cross-marketing, lossleader analysis, and other business decision making processes.

A typical example of association rule 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 such associations can help retailers develop marketing strategies by gaining insight into which items are frequently purchased together by customers. For instance, if customers are buying milk, how likely are they to also buy bread (and what kind of bread) on the same trip to the supermarket? Such information can lead to increased sales by helping retailers to do selective marketing and plan their shelf space. For instance, placing milk and bread within close proximity may further encourage the sale of these items together within single visits to the store.

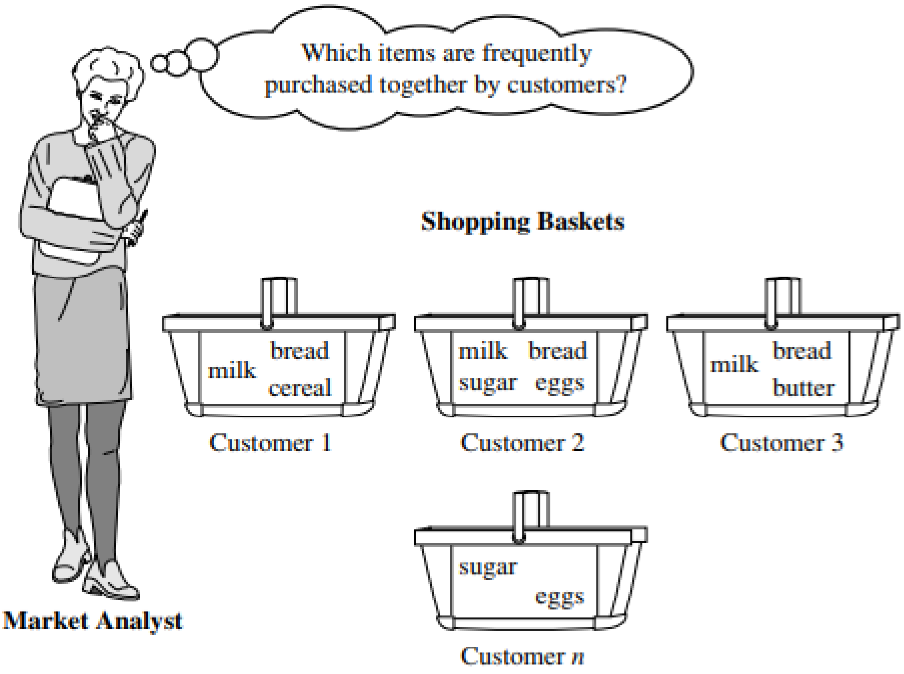


Figure 6.1 Market basket analysis.

How can we find association rules from large amounts of data, where the data are either transactional or relational? Which association rules are the most interesting? How can we help or guide the mining procedure to discover interesting associations?

Association rule mining

Association rule mining searches for interesting relationships among items in a given data set.

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 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. For instance, market basket analysis may help you 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 entice 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 ⇒ antivirus software [support = 2%,confidence = 60%] . (6.1)

Rule support and confidence are two measures of rule interestingness. They respectively reflect the usefulness and certainty of discovered rules. A support of 2% for Rule (6.1) 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. Additional analysis can be performed to discover interesting statistical correlations between associated items.

Basic concepts

Let $I = \{I1, I2, ..., Im\}$ be an itemset {SET OF ITEMS} .

Let D, the task-relevant data, be a set of database transactions where each transaction T is a nonempty itemset such that $T \subseteq I$.

Each transaction is associated with an identifier, called a TID. Let A be a set of items. A transaction T is said to contain A if $A \subseteq T$. An association rule is an implication of the form $A \Rightarrow B$, where $A \subset I$, $B \subset I$, 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). 1 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⇒B) =P(A ∪B) . (6.2)

confidence(A⇒B) =P(B|A). (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. 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 Lk .

From Eq. (6.3), we have

$$\text{confidence(A⇒B) = P(B|A) = } \frac{support(A\cup B)}{support(A)} = \frac{support\_count(A\cup B)}{support\_count(A)}. \quad (6.4)$$

Equation (6.4) shows that the confidence of rule A ⇒ B can be easily derived from the support counts of A and A ∪B. That is, once the support counts of A, B, and A ∪B are found, it is straightforward to derive the corresponding association rules A ⇒ B and B ⇒ A and check whether they are strong. Thus, the problem of mining association rules can be reduced to that of mining frequent itemsets.

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. Additional interestingness measures can be applied for the discovery of correlation relationships between associated items, as will be discussed in Section 6.3. Because the second step is much less costly than the first, the overall performance of mining association rules is determined by the first step.

Association rule mining: A road map

Market basket analysis is just one form of association rule mining. In fact, there are many kinds of association rules.

Association rules can be classifed in various ways, based on the following criteria:

1. Based on the types of values handled in the rule:

If a rule concerns associations between the presence or absence of items, it is a Boolean association rule. For example, Rule (6.1) above is a Boolean association rule obtained from market basket analysis. If a rule describes associations between quantitative items or attributes, then it is a quantitative association rule. In these rules, quantitative values for items or attributes are partitioned into intervals. Rule (6.4) below is an example of a quantitative association rule.

age(X,"30 - 34") ^ income(X,"42K - 48K") ⇒ buys(X,"high resolution TV") (6.4)

Note that the quantitative attributes, age and income, have been discretized.

2. Based on the dimensions of data involved in the rule:

If the items or attributes in an association rule each reference only one dimension, then it is a single- dimensional association rule. Note that Rule (6.1) could be rewritten as

buys(X,"computer") ⇒ buys(X," antivirus software ") (6.5)

Rule (6.1) is therefore a single-dimensional association rule since it refers to only one dimension, i.e., buys. If a rule references two or more dimensions, such as the dimensions buys, time of transaction, and customer category, then it is a multidimensional association rule. Rule (6.4) above is considered a multidimensional association rule since it involves three dimensions, age, income, and buys.

3. Based on the levels of abstractions involved in the rule set:

Some methods for association rule mining can fnd rules at difering levels of abstraction. For example, suppose that a set of association rules mined included Rule (6.6) and (6.7) below.

age(X,"30 - 34") ⇒ buys(X,"laptop computer") (6.6)

age(X,"30 - 34") ⇒ buys(X,"computer") (6.7)

In Rules (6.6) and (6.7), the items bought are referenced at diferent levels of abstraction. (That is, "computer" is a higher level abstraction of "laptop computer"). We refer to the rule set mined as consisting of multilevel association rules. If, instead, the rules within a given set do not reference items or attributes at diferent levels of abstraction, then the set contains single-level association rules.

4. Based on the nature of the association involved in the rule: Association mining can be extended to correlation analysis, where the absence or presence of correlated items can be identifed.

Throughout the rest of this chapter, you will study methods for mining each of the association rule types described.

6.2 Mining single-dimensional Boolean association rules from transactional databases - **Apriori Algorithm**

In this section, you will learn methods for mining the simplest form of association rules - single-dimensional, singlelevel, Boolean association rules, such as those discussed for market basket analysis in Section 6.1.1. We begin by presenting **Apriori**, a basic algorithm for fnding frequent itemsets (Section 6.2.1). A procedure for generating strong association rules from frequent itemsets is discussed in Section 6.2.2. Section 6.2.3 describes several variations to the Apriori algorithm for improved efciency and scalability.

6.2.1 **The Apriori algorithm: Finding frequent itemsets**

Apriori is an influential algorithm for mining frequent itemsets for Boolean association rules. The name of the algorithm is based on the fact that the algorithm uses prior knowledge of frequent itemset properties, as we shall see below. Apriori employs an iterative approach known as a level-wise search, where k-itemsets are used to explore s(k+1)-itemsets. First, the set of frequent 1-itemsets is found. This set is denoted L1. L1 is used to fnd L2, the frequent 2-itemsets, which is used to fnd L3, and so on, until no more frequent k-itemsets can be found. The finding of each Lk requires one full scan of the database.

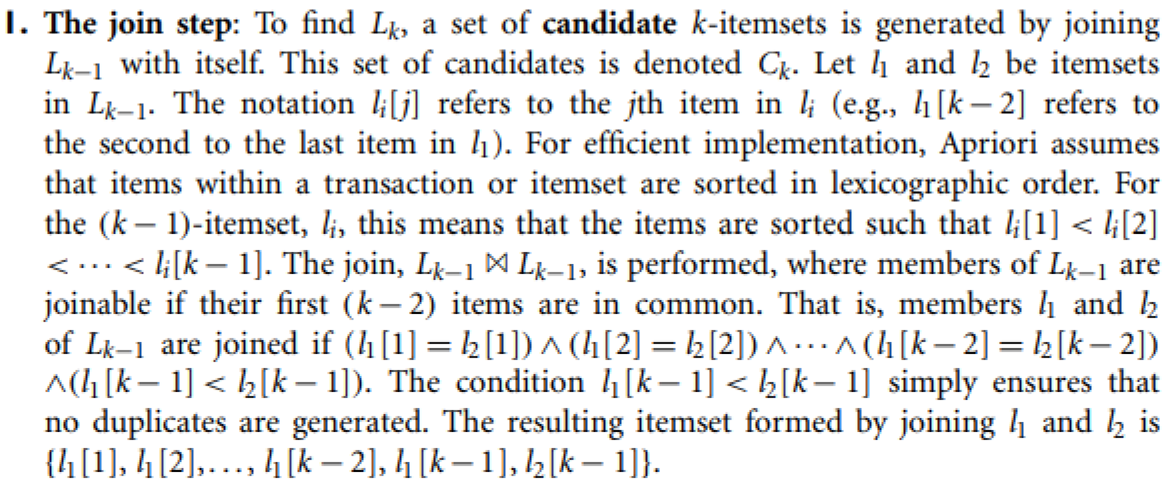
To improve the efficiency of the level-wise generation of frequent itemsets, an important property called the Apriori property, presented below, is used to reduce the search space. We will frst describe this property, and then show an example illustrating its use.

The Apriori property. All non-empty subsets of a frequent itemset must also be frequent.

This property is based on the following observation. By defnition, if an itemset I does not satisfy the minimum support threshold, s, then I is not frequent, i.e., P(I) < s. 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, I U A is not frequent either, i.e., P(I U A) < s.

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

“How is the Apriori property used in the algorithm?" To understand this, we must look at how L k-1 is used to find L k. 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: 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.

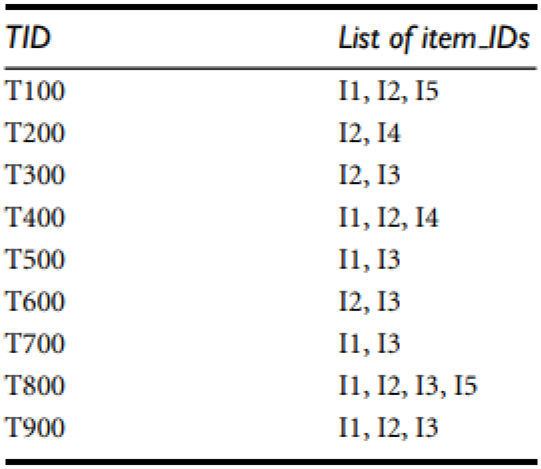
Apriori. Let's look at a concrete example, based on the AllElectronics transaction database, D, of Table 6.1. There are nine transactions in this database, that is, |D| = 9. We use Figure 6.2 to illustrate the Apriori algorithm for finding frequent itemsets in D.

1. In the first iteration of the algorithm, each item is a member of the set of candidate 1-itemsets, C1. The algorithm simply scans all of the transactions to count the number of occurrences of each item.

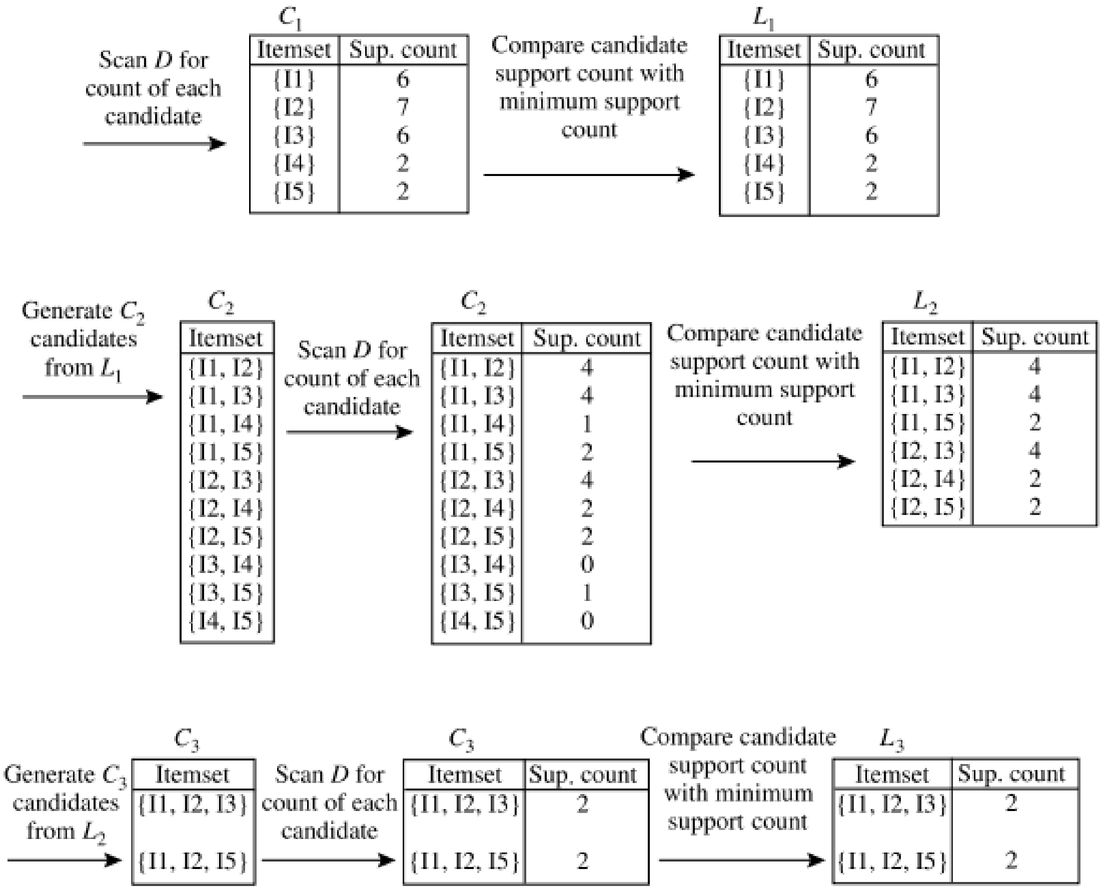
2. Suppose that the minimum support count required is 2, that is, min sup = 2. (Here, we are referring to absolute support because we are using a support count. The corresponding relative support is 2/9 = 22%.) The set of frequent 1-itemsets, L1, can then be determined. It consists of the candidate 1-itemsets 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-itemsets, C2. ${}^{7}C_{2}$ consists of ${}^{|L1|}C_{2}$ 2-itemsets. Note that no candidates are removed from C2 during the prune step because each subset of the candidates is also frequent.

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



Scan D for count of each candidate
Compare candidate support count with minimum support count
Generate C2 candidates from L1
Scan D for count of each candidate
Compare candidate support count with minimum support count
Generate C3 candidates from L2
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

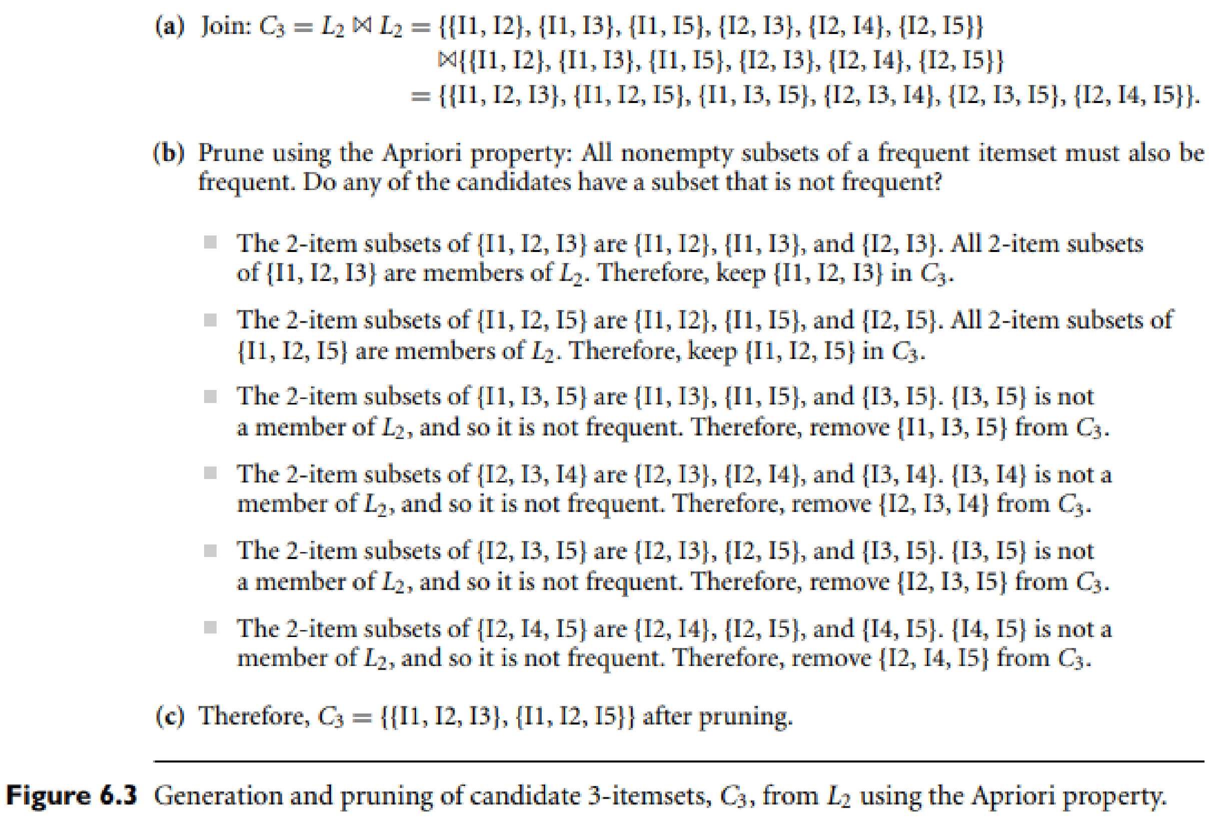
Figure 6.2 Generation of the candidate itemsets and frequent itemsets, where the minimum support count is 2.

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 in Figure 6.2.

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 detailed in Figure 6.3. From the join step, we first get C3 = L2 ⋈ L2 = {{I1, I2, I3}, {I1, I2, I5}, {I1, I3, I5}, {I2, I3, I4}, {I2, I3, I5}, {I2, I4, I5}}. Based on the Apriori property that all subsets of a frequent itemset must also be frequent, we can determine that the four latter candidates cannot possibly be frequent. We therefore remove them from C3, thereby saving the effort of unnecessarily obtaining their counts during the subsequent scan of D to determine L3. Note that when given a candidate k-itemset, we only need to check if its (k − 1)-subsets are frequent since the Apriori algorithm uses a level-wisesearch 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).



(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.

Figure 6.3 Generation and pruning of candidate 3-itemsets, C_3, from L_2 using the Apriori property.

8. The algorithm uses L3 ⋈ L3 to generate a candidate set of 4-itemsets, C4. 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, C4 = ϕ, and the algorithm terminates, having found all of the frequent itemsets.

Figure 6.4 shows pseudocode for the Apriori algorithm and its related procedures. Step 1 of Apriori finds the frequent 1-itemsets, L1. In steps 2 through 10, Lk−1 is used to generate candidates Ck to find Lk for k ≥ 2. The apriori gen procedure generates the candidates and then uses the Apriori property to eliminate those having a subset that is not frequent (step 3). This procedure is described later. Once all of the candidates have been generated, the database is scanned (step 4). For each transaction, a subset function is used to find all subsets of the transaction that are candidates (step 5), and the count for each of these candidates is accumulated (steps 6 and 7). Finally, all the candidates satisfying the minimum support (step 9) form the set of frequent itemsets, L (step 11).

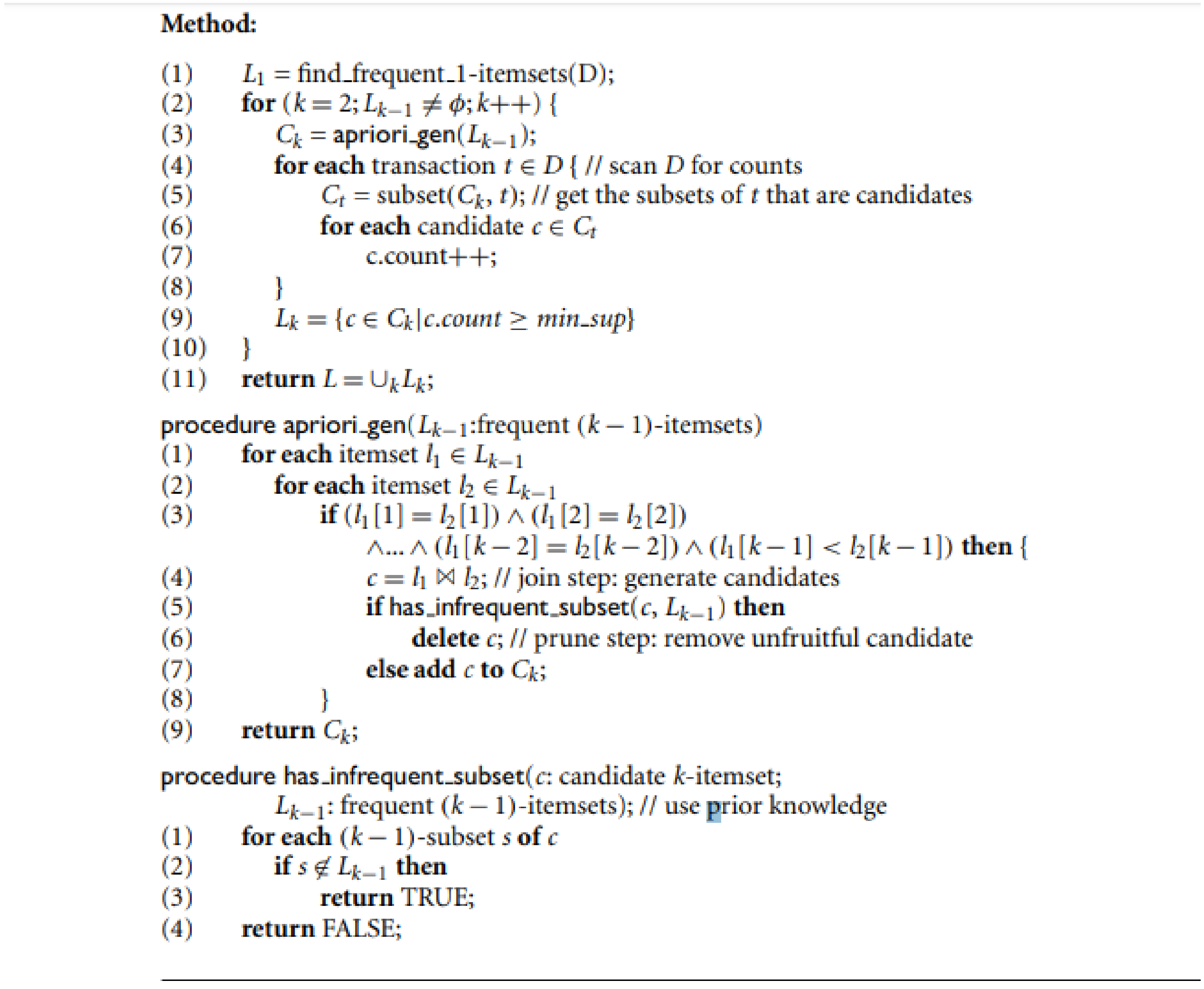
Algorithm: Apriori. Find frequent itemsets using an iterative level-wise approach based on candidate generation.

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;
```

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

A procedure can then be called to generate association rules from the frequent itemsets.

The apriori_gen procedure performs two kinds of actions, namely, join and prune, as described before. In the join component, Lk−1 is joined with Lk−1 to generate potential candidates (steps 1–4). The prune component (steps 5–7) employs the Apriori property to remove candidates that have a subset that is not frequent. The test for infrequent subsets is shown in procedure has_infrequent_subset.

Generating Association Rules from Frequent Itemsets

Once the frequent itemsets from transactions in a database D have been found, it is straightforward to generate strong association rules from them (where strong association rules satisfy both minimum support and minimum confidence). This can be done using Eq. (6.4) for confidence, which we show again here for completeness:

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

The conditional probability is expressed in terms of itemset support count, where support count(AUB) is the number of transactions containing the itemsets AUB, 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 ⇒ (l – s)" if $\frac{support\_count(l)}{support\_count(s)}$≥ 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} ⇒ I5, confidence = 2/4 = 50%

{I1, I5} ⇒ I2, confidence = 2/2 = 100%

{I2, I5} ⇒ I1, confidence = 2/2 = 100%

I1 ⇒ {I2, I5}, confidence = 2/6 = 33%

I2 ⇒ {I1, I5}, confidence = 2/7 = 29%

I5 ⇒ {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. Note that, unlike conventional classification rules, association rules can contain more than one conjunct in the right side of the rule.

A Pattern-Growth Approach for Mining Frequent Itemsets - Frequent pattern growth, or simply FP-growth Algorithm

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

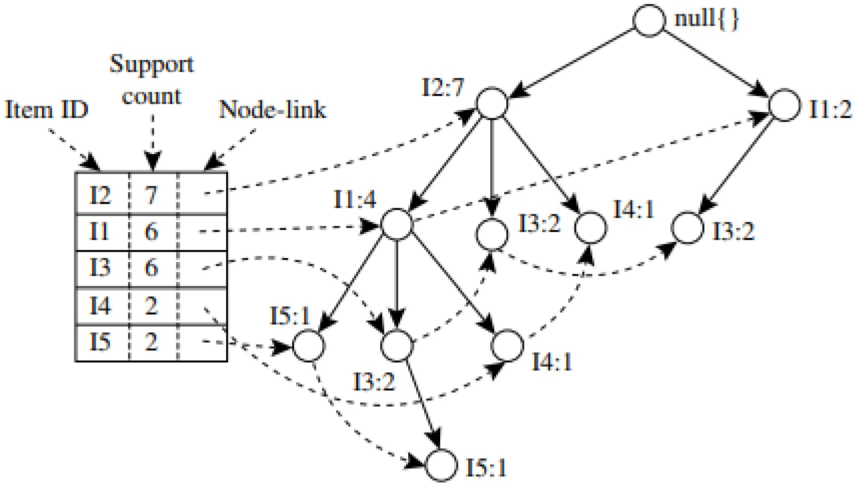
- It may still need to generate a huge number of candidate sets. For example, if there are 104 frequent 1-itemsets, the Apriori algorithm will need to generate more than 107 candidate 2-itemsets.
- It may need to repeatedly scan the whole database and check a large set of candidates by pattern matching. It is costly to go over each transaction in the database to determine the support of the candidate itemsets.

"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 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. You will see how it works in Example 6.5.

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. 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).



Item ID	Support count	Node-link
I2	7	
I1	6	
I3	6	
I4	2	
I5	2	

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

In general, when considering the branch to be added for a transaction, the count of each node along a common prefix is incremented by 1, and nodes for the items following the prefix are created and linked accordingly.

To facilitate tree traversal, an item header table is built so that each item points to its occurrences in the tree via a chain of node-links. The tree obtained after scanning all the transactions is shown in Figure

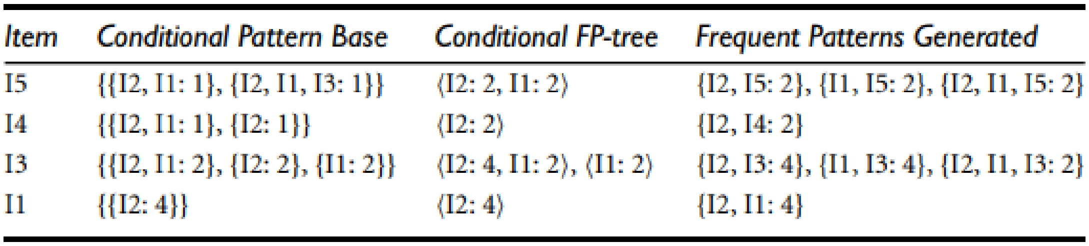
6.7 with the associated node-links. In this way, the problem of mining frequent patterns in databases is transformed into that of mining the FP-tree.

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 6.2 and detailed as follows. We first consider I5, which is the last item in L, rather than the first. The reason for starting at the end of the list will become apparent as we explain the FP-tree mining process. I5 occurs in two FP-tree branches of Figure 6.7. (The occurrences of I5 can easily be found by following its chain of node-links.) The paths formed by these branches are (I2, I1, I5: 1) and (I2, I1, I3, I5: 1). Therefore, considering I5 as a suffix, its corresponding two prefix paths are (I2, I1: 1) and (I2, I1, I3: 1), which form its conditional pattern base. 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, hI2: 2i, 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}

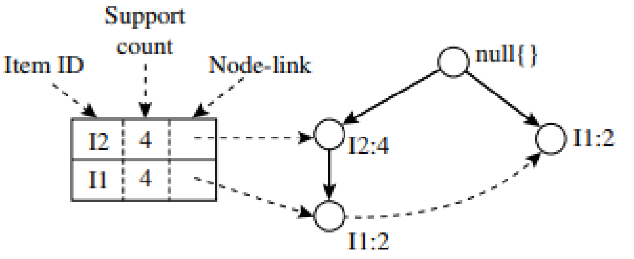


Figure 6.8 The conditional FP-tree associated with the conditional node I3.

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}, {I2, 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.

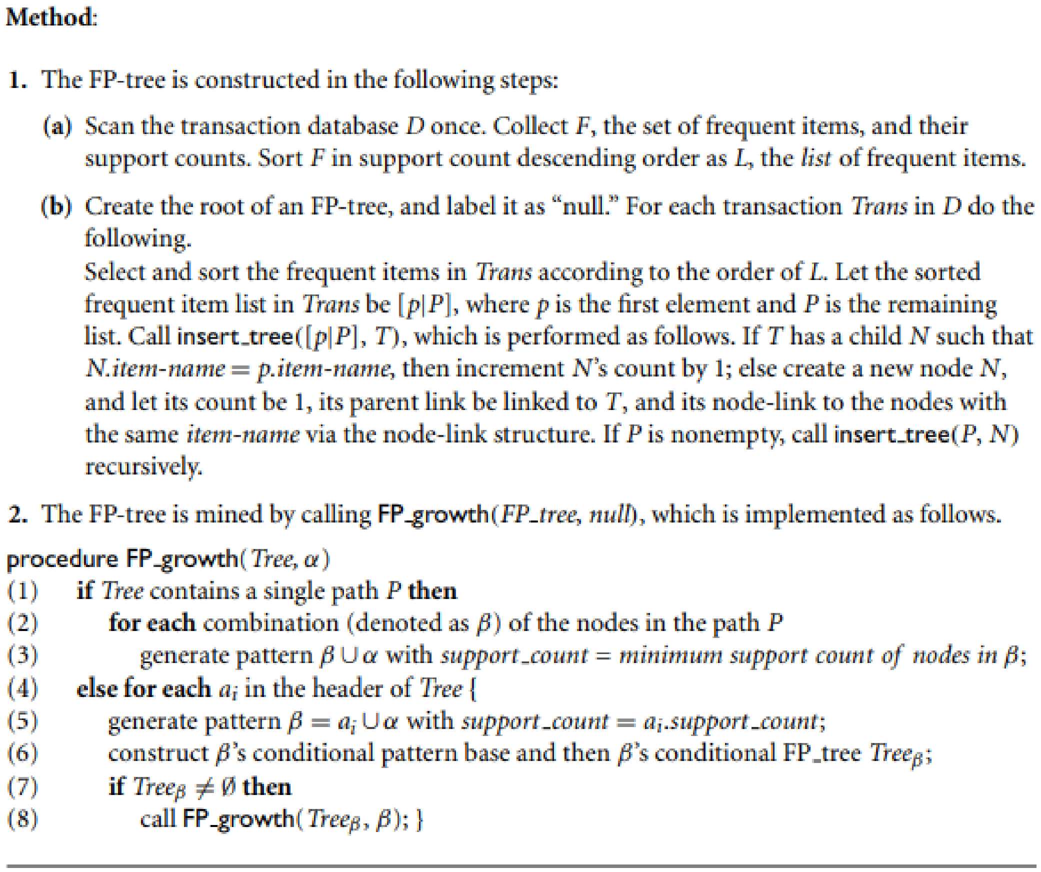
Algorithm: FP growth. Mine frequent itemsets using an FP-tree by pattern fragment growth.

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 $N.item\text{-}name = p.item\text{-}name$, then increment N's count by 1; else create a new node N, and let its count be 1, its parent link be linked to T, and its node-link to the nodes with the same *item-name* via the node-link structure. If P is nonempty, call insert_tree(P, N) recursively.

2. The FP-tree is mined by calling FP_growth(*FP_tree*, *null*), which is implemented as follows.

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

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

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 memorybased 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.

Cluster Analysis

Imagine that you are given a set of data objects for analysis where, unlike in classi_cation, the class label of each object is not known. Clustering is the process of grouping the data into classes or clusters so that objects within a cluster have high similarity in comparison to one another, but are very dissimilar to objects in other clusters.

Dissimilarities are assessed based on the attribute values describing the objects. Often, distance measures are used.Clustering has its roots in many areas, including data mining, statistics, biology, and machine learning.

What is cluster analysis?

The process of grouping a set of physical or abstract objects into classes of similar objects is called clustering. A cluster is a collection of data objects that are similar to one another within the same cluster and are dissimilar to the objects in other clusters. A cluster of data objects can be treated collectively as one group in many applications.

Cluster analysis is an important human activity. Early in childhood, one learns how to distinguish between cats and dogs, or between animals and plants, by continuously improving subconscious classification schemes. Cluster analysis has been widely used in numerous applications, including pattern recognition, data analysis, image processing, and market research. By clustering, one can identify crowded and sparse regions, and therefore, discover overall distribution patterns and interesting correlations among data attributes.

“What are some typical applications of clustering?"

In business, clustering may help marketers discover distinct groups in their customer bases and characterize customer groups based on purchasing patterns. In biology, it can be used to derive plant and animal taxonomies, categorize genes with similar functionality, and gain insight into structures inherent in populations. Clustering may also help in the identification of areas of similar land use in an earth observation database, and in the identification of groups of motor insurance policy holders with a high average claim cost, as well as the identification of groups of houses in a city according to house type, value, and geographical location. It may also help classify documents on the WWW for information discovery. As a data mining function, cluster analysis can be used as a stand-alone tool to gain insight into the distribution of data, to observe the characteristics of each cluster, and to focus on a particular set of clusters for further analysis. Alternatively, it may serve as a preprocessing step for other algorithms, such as classification and characterization, operating on the detected clusters.

Data clustering is under vigorous development. Contributing areas of research include data mining, statistics, machine learning, spatial database technology, biology, and marketing. Owing to the huge amounts of data collected in databases, cluster analysis has recently become a highly active topic in data mining research.

As a branch of statistics, cluster analysis has been studied extensively for many years, focusing mainly on distance-based cluster analysis. Cluster analysis tools based on k-means, k-medoids, and several other methods have alsobeen built into many statistical analysis software packages or systems, such as S-Plus, SPSS, and SAS.

In machine learning, clustering is an example of unsupervised learning. Unlike classification, clustering and unsupervised learning do not rely on predefined classes and class-labeled training examples. For this reason, clustering is a form of learning by observation, rather than learning by examples. In conceptual clustering, a group of objects forms a class only if it is describable by a concept. This differs from conventional clustering which measures similarity based on geometric distance. Conceptual clustering consists of two components: (1) it discovers the appropriate classes, and (2) it forms descriptions for each class, as in classification. The guideline of striving for high intraclass similarity and low interclass similarity still applies.

In data mining, efforts have focused on finding methods for efficient and effective cluster analysis in large databases.

Active themes of research focus on the scalability of clustering methods, the effectiveness of methods for clustering complex shapes and types of data, high-dimensional clustering techniques, and methods for clustering mixed numerical and categorical data in large databases.

Clustering is a challenging field of research where its potential applications pose their own special requirements.

The following are typical requirements of clustering in data mining.

1. Scalability: Many clustering algorithms work well in small data sets containing less than 200 data objects; however, a large database may contain millions of objects. Clustering on a sample of a given large data set may lead to biased results. Highly scalable clustering algorithms are needed.

2. Ability to deal with di_erent types of attributes: Many algorithms are designed to cluster interval-based (numerical) data. However, applications may require clustering other types of data, such as binary, categorical (nominal), and ordinal data, or mixtures of these data types.

3. Discovery of clusters with arbitrary shape: Many clustering algorithms determine clusters based on Euclidean or Manhattan distance measures. Algorithms based on such distance measures tend to find spherical clusters with similar size and density. However, a cluster could be of any shape. It is important to develop algorithms which can detect clusters of arbitrary shape.

4. Minimal requirements for domain knowledge to determine input parameters: Many clustering algorithms require users to input certain parameters in cluster analysis (such as the number of desired clusters). The clustering results are often quite sensitive to input parameters. Parameters are often hard to determine, especially for data sets containing high-dimensional objects. This not only burdens users, but also makes the quality of clustering difficult to control.

5. Ability to deal with noisy data: Most real-world databases contain outliers or missing, unknown, or erroneous data. Some clustering algorithms are sensitive to such data and may lead to clusters of poor quality.

6. Insensitivity to the order of input records: Some clustering algorithms are sensitive to the order of input data, e.g., the same set of data, when presented with di_erent orderings to such an algorithm, may generate dramatically different clusters. It is important to develop algorithms which are insensitive to the order of input.

7. High dimensionality: A database or a data warehouse may contain several dimensions or attributes. Many clustering algorithms are good at handling low-dimensional data, involving only two to three dimensions. Human eyes are good at judging the quality of clustering for up to three dimensions. It is challenging to cluster data objects in high-dimensional space, especially considering that data in high-dimensional space can be very sparse and highly skewed.

8. Constraint-based clustering: Real-world applications may need to perform clustering under various kinds of constraints. Suppose that your job is to choose the locations for a given number of new automatic cash stations (ATMs) in a city. To decide upon this, you may cluster households while considering constraints such as the city's rivers and highway networks, and customer requirements per region. A challenging task is to find groups of data with good clustering behavior that satisfy speci_ed constraints.

9. Interpretability and usability: Users expect clustering results to be interpretable, comprehensible, and usable. That is, clustering may need to be tied up with speci_c semantic interpretations and applications. It is important to study how an application goal may inuence the selection of clustering methods.With these requirements in mind, our study of cluster analysis proceeds as follows. First, we study different types of data and how they can inuence clustering methods. Second, we present a general categorization of clustering methods. We then study each clustering method in detail, including partitioning methods, hierarchical methods, density-based methods, grid-based methods, and model-based methods. We also examine clustering in high-dimensional space and outlier analysis.

8.2 Types of data in clustering analysis

We study the types of data which often occur in clustering analysis and how to preprocess them for such an analysis. Suppose that a data set to be clustered contains n objects which may represent persons, houses, documents, countries, etc. Main memory-based clustering algorithms typically operate on either of the following two data structures.

1. Data matrix (or object-by-variable structure): This represents n objects, such as persons, with p variables (also called measurements or attributes), such as age, height, weight, gender, race, and so on. The structure is in the form of a relational table, or n-by-p matrix (n objects X p variables), as shown in (8.1).

$$\begin{bmatrix} x_{11} & \cdots & x_{1f} & \cdots & x_{1p} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{i1} & \cdots & x_{if} & \cdots & x_{ip} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{n1} & \cdots & x_{nf} & \cdots & x_{np} \end{bmatrix}$$

2. Dissimilarity matrix (or object-by-object structure): This stores a collection of proximities that are available for all pairs of n objects. It is often represented by an n-by-n table as shown below,

$$\begin{bmatrix} 0 & & & & \\ d(2,1) & 0 & & & \\ d(3,1) & d(3,2) & 0 & & \\ \vdots & \vdots & \vdots & & \\ d(n,1) & d(n,2) & \cdots & \cdots & 0 \end{bmatrix}$$

where d(i,j) is the measured difference or dissimilarity between objects i and j. In general, d(i, j) is a nonnegative number that is close to 0 when objects i and j are highly similar or “near" each other, and becomes larger the more they differ. Since d(i, j) = d(j,i), and d(i, i) = 0, we have the matrix in (8.2).

Measures of dissimilarity are discussed throughout this section.

The data matrix is often called a two-mode matrix, whereas the dissimilarity matrix is called a one-mode matrix,since the rows and columns of the former represent different entities, while those of the latter represent the same entity. Many clustering algorithms operate on a dissimilarity matrix. If the data are presented in the form of a data matrix, it can first be transformed into a dissimilarity matrix before applying such clustering algorithms.
“How can dissimilarity, d(i,j), be assessed?", you may wonder.

In this section, we discuss how object dissimilarity can be computed for objects described by interval-scaled variables, by binary variables, by nominal, ordinal, and ratio-scaled variables, or combinations of these variable types. The dissimilarity data can later be used to compute clusters of objects.

A categorization of major clustering methods

There exist a large number of clustering algorithms in the literature. The choice of clustering algorithm depends both on the type of data available and on the particular purpose and application. If cluster analysis is used as a descriptive or exploratory tool, it is possible to try several algorithms on the same data to see what the data may disclose.

In general, major clustering methods can be classified into the following categories.

1. Partitioning methods.

Given a database of n objects or data tuples, a partitioning method constructs k partitions of the data, where each partition represents a cluster, and k ≤ n. That is, it classifies the data into k groups, which together satisfy the following requirements: (1) each group must contain at least one object, and (2) each object must belong to exactly one group. Notice that the second requirement can be relaxed in some fuzzy partitioning techniques. References to such techniques are given in the bibliographic notes.

Given k, the number of partitions to construct, a partitioning method creates an initial partitioning. It then uses an iterative relocation technique which attempts to improve the partitioning by moving objects from one group to another. The general criterion of a good partitioning is that objects in the same cluster are “close" or related to each other, whereas objects of different clusters are “far apart" or very different. There are various kinds of other criteria for judging the quality of partitions.

To achieve global optimality in partitioning-based clustering would require the exhaustive enumeration of all of the possible partitions. Instead, most applications adopt one of two popular heuristic methods:

(1) the k-means algorithm, where each cluster is represented by the mean value of the objects in the cluster; and (2) the k-medoids algorithm, where each cluster is represented by one of the objects located near the center of the cluster. These heuristic clustering methods work well for _nding spherical-shaped clusters in small to medium sized databases. For finding clusters with complex shapes and for clustering very large data sets, partitioning-based methods need to be extended.

2. Hierarchical methods.

A hierarchical method creates a hierarchical decomposition of the given set of data objects. A hierarchical method can be classified as being either agglomerative or divisive, based on how the hierarchical decomposition is formed. The agglomerative approach, also called the \bottom-up" approach, starts with each object forming a separate group. It successively merges the objects or groups close to one another, until all of the groups are merged into one (the topmost level of the hierarchy), or until a termination condition holds. The divisive approach, also called the “top-down" approach, starts with all the objects in the same cluster. In each successive iteration, a cluster is split up into smaller clusters, until eventually each object is in one cluster, or until a termination condition holds.

Hierarchical methods suffer from the fact that once a step (merge or split) is done, it can never be undone. This rigidity is useful in that it leads to smaller computation costs by not worrying about a combinatorial number of different choices. However, a major problem of such techniques is that they cannot correct erroneous decisions.

It can be advantageous to combine iterative relocation and hierarchical agglomeration by _rst using a hierarchical agglomerative algorithm and then re_ning the result using iterative relocation. Some scalable clustering algorithms, such as BIRCH and CURE, have been developed based on such an integrated approach. Hierarchical clustering methods are studied in Section 8.5.

3. Density-based methods.

Most partitioning methods cluster objects based on the distance between objects. Such methods can _nd only spherical-shaped clusters and encounter dificulty at discovering clusters of arbitrary shapes. Other clustering methods have been developed based on the notion of density. Their general idea is to continue growing the given cluster as long as the density (number of objects or data points) in the “neighborhood" exceeds some threshold, i.e., for each data point within a given cluster, the neighborhood of a given radius has to contain at least a minimum number of points. Such a method can be used to _lter out noise (outliers), and discover clusters of arbitrary shape.

DBSCAN is a typical density-based method which grows clusters according to a density threshold. OPTICS is a density-based method which computes an augmented clustering ordering for automatic and interactive cluster analysis. Density-based clustering methods are studied in Section 8.6.

4. Grid-based methods.

Grid-based methods quantize the object space into a finite number of cells which form a grid structure. All of the clustering operations are performed on the grid structure (i.e., on the quantized space). The main advantage of this approach is its fast processing time which is typically independent of the number of data objects, and dependent only on the number of cells in each dimension in the quantized space.

STING is a typical example of a grid-based method. CLIQUE and Wave-Cluster are two clustering algorithms which are both grid-based and density-based. Grid-based clustering methods are studied in Section 8.7.

5. Model-based methods.

Model-based methods hypothesize a model for each of the clusters, and find the best fit of the data to the given model. A model-based algorithm may locate clusters by constructing a density function that reects the spatial distribution of the data points. It also leads to a way of automatically determining the number of clusters based on standard statistics, taking \noise" or outliers into account and thus yielding robust clustering methods.

Some clustering algorithms integrate the ideas of several clustering methods, so that it is sometimes difficult to classify a given algorithm as uniquely belonging to only one clustering method category.

Furthermore, some applications may have clustering criteria which require the integration of several clustering techniques.

In the following sections, we examine each of the above five clustering methods in detail. We also introduce algorithms which integrate the ideas of several clustering methods. Outlier analysis, which typically involves clustering,is described in Section 8.9.

8.4 Partitioning methods

Given a database of n objects, and k, the number of clusters to form, a partitioning algorithm organizes the objects into k partitions (k ≤ n), where each partition represents a cluster. The clusters are formed to optimize an objective partitioning criterion, often called a similarity function, such as distance, so that the objects within a cluster are “similar", whereas the objects of di_erent clusters are \dissimilar" in terms of the database attributes.

8.4.1 Classical partitioning methods: k-means and k-medoids

The most well-known and commonly used partitioning methods are k-means, k-medoids, and their variations.

Centroid-based technique: The k-means method

The k-means algorithm takes the input parameter, k, and partitions a set of n objects into k clusters so that the resulting intra-cluster similarity is high whereas the inter-cluster similarity is low. Cluster similarity is measured in regard to the mean value of the objects in a cluster, which can be viewed as the cluster's “center of gravity".

“How does the k-means algorithm work?" It proceeds as follows. First, it randomly selects k of the objects, which initially each represent a cluster mean or center. For each of the remaining objects, an object is assigned to the cluster to which it is the most similar, based on the distance between the object and the cluster mean. It then computes the new mean for each cluster. This process iterates until the criterion function converges. Typically, the squared-error criterion is used, defined as

$$E = \sum_{i=1}^{k} \sum_{p \in C_i} dist(p, c_i)^2,$$

(8.18)

where E is the sum of square-error for all objects in the database, p is the point in space representing the given object, and mi is the mean of cluster Ci (both p and mi are multidimensional). This criterion tries to make the resulting k clusters as compact and as separate as possible. The k-means procedure is summarized below.

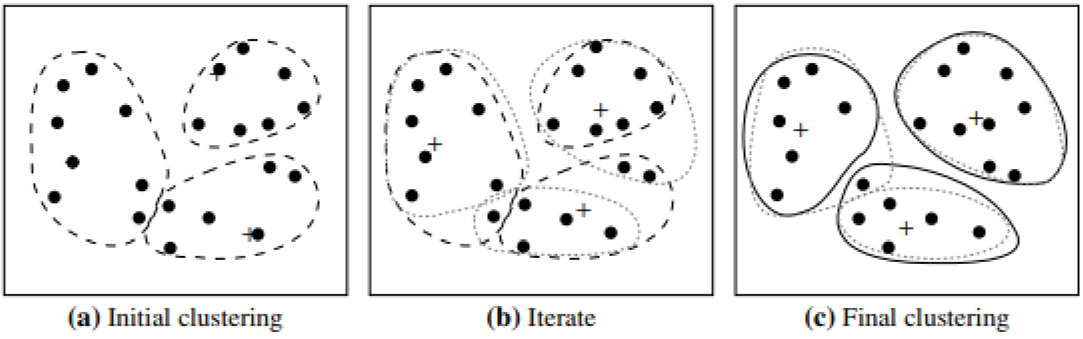
Algorithm 8.4.1 (k-means) The k-means algorithm for partitioning based on the mean value of the objects in the cluster.

```
Input: The number of clusters k, and a database containing n objects.

Output: A set of k clusters which minimizes the squared-error criterion.

Method: 1) arbitrarily choose k objects as the initial cluster centers;

2) repeat

3) (re)assign each object to the cluster to which the object is the most similar,

        based on the mean value of the objects in the cluster;

4) update the cluster means, i.e., calculate the mean value of the objects for each cluster;

5) until no change;
```

The k-means algorithm.

The algorithm attempts to determine k partitions that minimize the squared-error function. It works well when the clusters are compact clouds that are rather well separated from one another. The method is relatively scalable and efficient in processing large data sets because the computational complexity of the algorithm is O(nkt), where n is the total number of objects, k is the number of clusters, and t is the number of iterations. Normally, k ≤ n and t ≤n. The method often terminates at a local optimum.

The k-means method, however, can be applied only when the mean of a cluster is defined. This may not be the case in some applications, such as when data with categorical attributes are involved. The necessity for users tospecify k, the number of clusters, in advance can be seen as a disadvantage. The k-means method is not suitable for discovering clusters with non-convex shapes, or clusters of very different size. Moreover, it is sensitive to noise and outlier data points since a small number of such data can substantially influence the mean value.


(a) Initial clustering

(b) Iterate

(c) Final clustering

Clustering of a set of objects using the *k*-means method; for (b) update cluster centers and reassign objects accordingly (the mean of each cluster is marked by a +).

Figure 8.2

Example 8.2 Suppose that there are a set of objects located in space as depicted in the rectangle shown in Figure 8.2a). Let k = 3, that is, the user would like to cluster the objects into three clusters.

According to Algorithm 8.4.1, we arbitrarily choose three objects as the three initial cluster centers, where cluster centers are marked by a “+". Each object is distributed to a cluster based on the cluster center to which it is the nearest. Such a distribution forms silhouettes encircled by dotted curves, as shown in Figure 8.2a).

This kind of grouping will update the cluster centers. That is, the mean value of each cluster is recalculated based on the objects in the cluster. Relative to these new centers, objects are re-distributed to the cluster domains based on which cluster center is the nearest. Such a re-distribution forms new silhouettes encircled by dashed curves, as shown in Figure 8.2b).

This process iterates, leading to Figure 8.2c). Eventually, no re-distribution of the objects in any cluster occurs and so the process terminates. The resulting clusters are returned by the clustering process.

There are quite a few variants of the k-means method. These can differ in the selection of the initial k means, the calculation of dissimilarity, and the strategies for calculating cluster means. An interesting strategy which often yields good results is to first apply a hierarchical agglomeration algorithm to determine the number of clusters and to find an initial classification, and then use iterative relocation to improve the classification.

Another variant to k-means is the k-modes method which extends the k-means paradigm to cluster categorical data by replacing the means of clusters with modes, using new dissimilarity measures to deal with categorical objects, and using a frequency-based method to update modes of clusters. The k-means and the k-modes methods can be integrated to cluster data with mixed numeric and categorical values, resulting in the k-prototypes method.

The EM (Expectation Maximization) algorithm extends the k-means paradigm in a different way: Instead of assigning each object to a dedicated cluster, it assigns each object to a cluster according to a weight representing the probability of membership. In other words, there are no strict boundaries between clusters. Therefore, new means are computed based on weighted measures.

“How can we make the k⍰means algorithm more scalable?" A recent e_ort on scaling the k-means algorithm is based on the idea of identifying three kinds of regions in data: regions that are compressible, regions that must be maintained in main memory, and regions that are discardable. An object is discardable if its membership in a cluster is ascertained. An object is compressible if it is not discardable but belongs to a tight subcluster. A data structure known as a clustering feature is used to summarize objects that have been discarded or compressed. If an object is neither discardable nor compressible, then it should be retained in main memory. To achieve scalability, the iterative clustering algorithm only includes the clustering features of the compressible objects and the objects which must be retained in main memory, thereby turning a secondary-memory based algorithm into a main memory-based algorithm.

Representative object-based technique: The k-medoids method

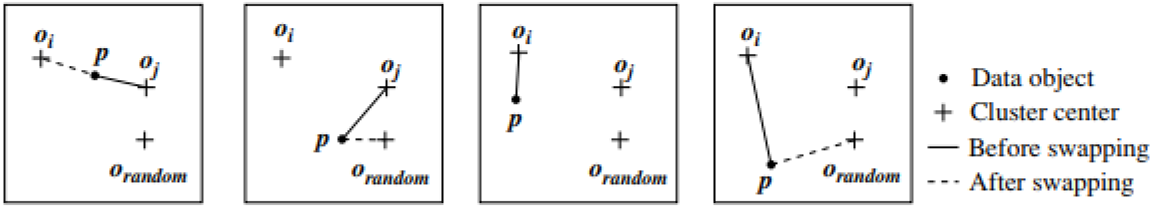
The k-means algorithm is sensitive to outliers since an object with an extremely large value may substantially distort the distribution of data.

“How might the algorithm be modified to diminish such sensitivity?", you may wonder.

Instead of taking the mean value of the objects in a cluster as a reference point, the medoid can be used, which is the most centrally located object in a cluster. Thus the partitioning method can still be performed based on the principle of minimizing the sum of the dissimilarities between each object and its corresponding reference point. This forms the basis of the k-medoids method.

The basic strategy of k-medoids clustering algorithms is to find k clusters in n objects by first arbitrarily finding a representative object (the medoid) for each cluster. Each remaining object is clustered with the medoid to which it is the most similar. The strategy then iteratively replaces one of the medoids by one of the non-medoids as long as the quality of the resulting clustering is improved. This quality is estimated using a cost function which measures the average dissimilarity between an object and the medoid of its cluster. To determine whether a non-medoid object, o_random, is a good replacement for a current medoid, oj , the following four cases are examined for each of the non-medoid objects, p.

- Case 1: p currently belongs to medoid oj. If oj is replaced by o_{random} as a medoid and p is closest to one of Oi, i ≠ j, then p is re-assigned to oi.
- Case 2: p currently belongs to medoid oj. If oj is replaced by o_{random} as a medoid and p is closest to o_{random}, then p is re-assigned to o_{random}.
- Case 3: p currently belongs to medoid oi, i 6= j. If oj is replaced by o_{random} as a medoid and and p is still closest to oi, then the assignment does not change.
- Case 4: p currently belongs to medoid oi, i 6= j. If oj is replaced by o_{random} as a medoid and p is closest to o_{random}, then p is re-assigned to o_{random}.



(a) Reassigned to o_i

(b) Reassigned to o_{random}

(c) No change

(d) Reassigned to o_{random}

Four cases of the cost function for *k*-medoids clustering.

Figure 8.3 illustrates the four cases. Each time a re-assignment occurs, a difference in square-error E is contributed to the cost function. Therefore, the cost function calculates the difference in square- error value if a current medoid is replaced by a non-medoid object. The total cost of swapping is the sum of costs incurred by all non-medoid objects.

If the total cost is negative, then oj is replaced or swapped with orandom since the actual square-error E would be reduced. If the total cost is positive, the current medoid oj is considered acceptable, and nothing is changed in the iteration. A general k-medoids algorithm is presented in Figure 8.4.

PAM (Partitioning Around Medoids) was one of the first k-medoids algorithms introduced. It attempts to determine k partitions for n objects. After an initial random selection of k medoids, the algorithm repeatedly tries to make a better choice of medoids. All of the possible pairs of objects are analyzed, where one object in each pair is considered a medoid and the other is not. The quality of the resulting clustering is calculated for each such combination. An object, oj, is replaced with the object causing the greatest reduction in square-error. The set of best objects for each cluster in one iteration form the medoids for the next iteration. For large values of n and k, such computation becomes very costly.

“Which method is more robust | k - means or k - medians?" The k-medoids method is more robust than k-means in the presence of noise and outliers because a medoid is less influenced by outliers or other extreme values than a mean. However, its processing is more costly than the k-means method. Both methods require the user to specify k, the number of clusters.

Algorithm 8.4.2 (k-medoids) A general k-medoids algorithm for partitioning based on medoid or central objects.

Input: The number of clusters k, and a database containing n objects.

Output: A set of k clusters which minimizes the sum of the dissimilarities of all the objects to their nearest medoid.

Method:

```
1) arbitrarily choose k objects as the initial medoids;
2) repeat
3)     assign each remaining object to the cluster with the nearest medoid;
4)     randomly select a non-medoid object, $o_{random}$;
5)     compute the total cost, S, of swapping oj with $o_{random}$;
6)     if $S < 0$ then swap oj with $o_{random}$ to form the new set of k medoids;
7) until no change;
```

Figure 8.4: The k-medoids algorithm.