Chapter 2
Data Mining Paradigms

2.1 Introduction

In data mining, the size of the dataset involved is large. It is convenient to visualize such a dataset as a matrix of size $n \times d$, where n is the number of data points, and d is the number of features. Typically, it is possible that either n or d or both are large. In mining such datasets, important issues are:

- The dataset cannot be accommodated in the main memory of the machine. So, we need to store the data on a secondary storage medium like a disk and transfer the data in parts to the main memory for processing; such an activity could be time-consuming. Because disk access can be more expensive compared to accessing the data from the memory, the number of database scans is an important parameter. So, when we analyze data mining algorithms, it is important to consider the number of database scans required.
- The dimensionality of the data can be very large. In such a case, several of the conventional algorithms that use the Euclidean distance like metrics to characterize proximity between a pair of patterns may not play a meaningful role in such high-dimensional spaces where the data is sparsely distributed. So, different techniques to deal with such high-dimensional datasets become important.
- Three important data mining tasks are:

  1. *Clustering.* Here a collection of patterns is partitioned into two or more clusters. Typically, clusters of patterns are represented using cluster representatives; a centroid of the points in the cluster is one of the most popularly used cluster representatives. Typically, a partition or a clustering is represented by k representatives, where k is the number of clusters; such a process leads to *lossy data compression*. Instead of dealing with all the n data points in the collection, one can just use the k cluster representatives (where $k \ll n$ in the data mining context) for further decision making.
  2. *Classification.* In classification, a machine learning algorithm is used on a given collection of training data to obtain an appropriate *abstraction* of the dataset. Decision trees and probability distributions of points in various classes

T. Ravindra Babu et al., *Compression Schemes for Mining Large Datasets*,
Advances in Computer Vision and Pattern Recognition,
DOI 10.1007/978-1-4471-5607-9_2,

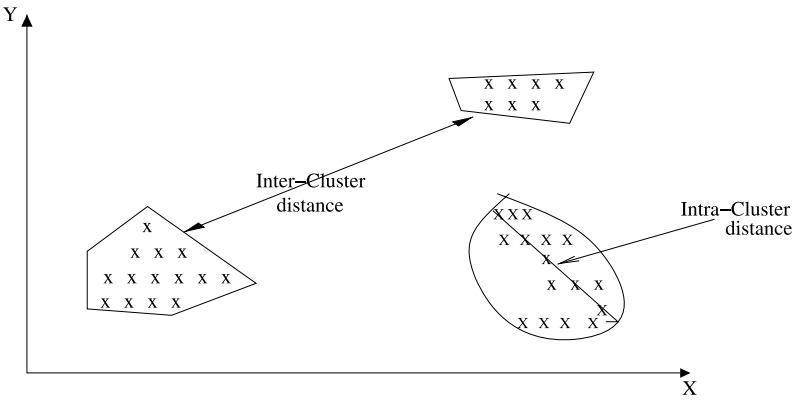

Fig. 2.1 Clustering

are examples of such abstractions. These abstractions are used to classify a test pattern.

3. *Association Rule Mining*. This activity has played a major role in giving a distinct status to the field of data mining itself. By convention, an association rule is an implication of the form $A \rightarrow B$, where A and B are two disjoint itemsets. It was initiated in the context of market-basket analysis to characterize how frequently items in A are bought along with items in B. However, generically it is possible to view classification and clustering rules also as association rules.

In order to run these tasks on large datasets, it is important to consider techniques that could lead to scalable mining algorithms. Before we examine these techniques, we briefly consider some of the popular algorithms for carrying out these data mining tasks.

2.2 Clustering

Clustering is the process of partitioning a set of patterns into cohesive groups or clusters. Such a process is carried out so that intra-cluster patterns are similar and inter-cluster patterns are dissimilar. This is illustrated using a set of two-dimensional points shown in Fig. 2.1. There are three clusters in this figure, and patterns are represented as two-dimensional points. The Euclidean distance between a pair of points belonging to the same cluster is smaller than that between any two points chosen from different clusters.

The Euclidean distance between two points X and Y in the p-dimensional space, where x_i and y_i are the ith components of X and Y, respectively, is given by

$$d(X, Y) = \left[\sum_{i=1}^{p}(x_i - y_i)^2\right]^{\frac{1}{2}}.$$

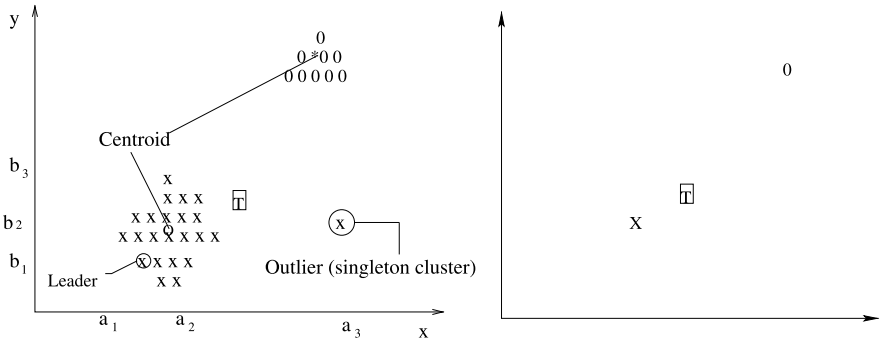

Fig. 2.2 Representing clusters

This notion characterizes similarity; the intra-cluster distance (similarity) is small (high), and the inter-cluster distance (similarity) is large (low). There could be other ways of characterizing similarity.

Clustering is useful in generating data abstraction. The process of data abstraction may be explained using Fig. 2.2. There are two dense clusters; the first has 22 points, and the second has 9 points. Further, there is a singleton cluster in the figure. Here, a cluster of points is represented by its *centroid* or its *leader*. The centroid stands for the sample mean of the points in the cluster, and it need not coincide with any one of the input points as indicated in the figure. There is another point in the figure, which is far off from any of the other points, and it belongs to the third cluster. This could be an *outlier*. Typically, these outliers are ignored, and each of the remaining clusters is represented by one or more points, called the *cluster representatives*, to achieve the abstraction. The most popular cluster representative is its centroid.

Here, if each cluster is represented by its centroid, then there is a reduction in the dataset size. One can use only the two centroids for further decision making. For example, in order to classify a test pattern using the nearest-neighbor classifier, one requires 32 distance computations if all the data points are used. However, using the two centroids requires just two distance computations to compute the nearest centroid of the test pattern. It is possible that classifiers using the cluster centroids can be optimal under some conditions. The above discussion illustrates the role of clustering in *lossy data compression*.

2.2.1 Clustering Algorithms

Typically, a grouping of patterns is meaningful when the within-group similarity is high and the between-group similarity is low. This may be illustrated using the groupings of the seven two-dimensional points shown in Fig. 2.3.

Algorithms for clustering can be broadly grouped into *hierarchical* and *partitional* categories. A hierarchical scheme forms a nested grouping of patterns,

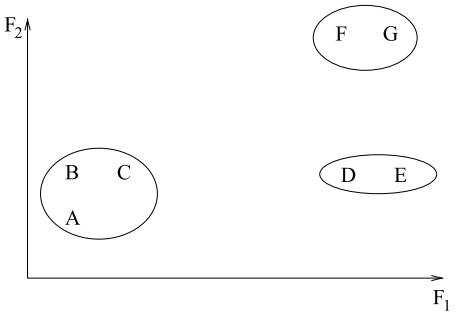

Fig. 2.3 A clustering of the two-dimensional points

whereas a partitional algorithm generates a single partition of the set of patterns. This is illustrated using the two-dimensional data set consisting of points labeled $A : (1, 1)^t$, $B : (2, 2)^t$, $C : (1, 2)^t$, $D : (6, 2)^t$, $E : (6.9, 2)^t$, $F : (6, 6)^t$, and $G : (6.8, 6)^t$ in Fig. 2.3. This figure depicts seven patterns in three clusters. Typically, a partitional algorithm would produce the three clusters shown in Fig. 2.3. A hierarchical algorithm would result in a *dendrogram* representing the nested grouping of patterns and similarity levels at which groupings change.

In this section, we describe two popular clustering algorithms; one of them is hierarchical, and the other is partitional.

2.2.2 Single-Link Algorithm

This algorithm is a bottom-up hierarchical algorithm starting with n singleton clusters when there are n data points to be clustered. It keeps on merging smaller clusters to form bigger clusters based on minimum distance between two clusters. The specific algorithm is described below.

Single-Link Algorithm

1. Input: n data points; Output: A dendrogram depicting the hierarchy.
2. Form the $n \times n$ proximity matrix by using the Euclidean distance between all pairs of points. Assign each point to a separate cluster; this step results in n singleton clusters.
3. Merge a pair of most similar clusters to form a bigger cluster. The distance between two clusters C_i and C_j to be merged is given by

$$Distance(C_i, C_j) = Min_{X,Y}\, d(X, Y) \quad \text{where } X \in C_i \text{ and } Y \in C_j$$

4. Repeat step 3 till the partition of required size is obtained; a k-partition is obtained if the number of clusters k is given; otherwise, merging continues till a single cluster of all the n points is obtained.

We illustrate the single-link algorithm using the data shown in Fig. 2.3. The proximity matrix showing the Euclidean distance between each pair of patterns is shown

Table 2.1 Distance matrix

	A	B	C	D	E	F	G
A	0.0	1.4	1.0	5.1	6.0	7.0	7.6
B	1.4	0.0	1.0	4.0	4.9	5.6	6.3
C	1.0	1.0	0.0	5.0	5.9	6.4	7.0
D	5.1	4.0	5.0	0.0	0.9	4.0	4.1
E	6.0	4.9	5.9	0.9	0.0	4.1	4.0
F	7.0	5.6	6.4	4.0	4.1	0.0	0.8
G	7.6	6.3	7.0	4.1	4.0	0.8	0.0

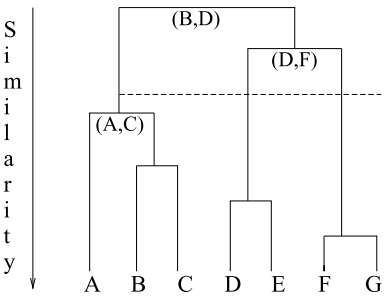

Fig. 2.4 The dendrogram obtained using the single-link algorithm

in Table 2.1. A dendrogram of the seven points in Fig. 2.3 (obtained from the single-link algorithm) is shown in Fig. 2.4. Note that there are seven leaves with each leaf corresponding to a singleton cluster in the tree structure. The smallest distance between a pair of such clusters is 0.8, which leads to merging {F} and {G} to form {F, G}. Next merger leads to {D, E} based on a distance of 0.9 units. This is followed by merging {B} and {C}, then {A} and {B, C} at a distance of 1 unit each. At this point we have three clusters. By merging clusters further we get ultimately a single cluster as shown in the figure. The dendrogram can be broken at different levels to yield different clusterings of the data. The partition of three clusters obtained using the dendrogram is the same as the partition shown in Fig. 2.3. A major issue with the hierarchical algorithm is that computation and storage of the proximity matrix requires $O(n^2)$ time and space.

2.2.3 k-Means Algorithm

The k-means algorithm is the most popular clustering algorithm. It is a partitional clustering algorithm and produces clusters by optimizing a criterion function. The most acceptable criterion function is the squared-error criterion as it can be used to generate compact clusters. The k-means algorithm is the most successfully used squared-error clustering algorithm. The k-means algorithm is popular because it is easy to implement and its time complexity is $O(n)$, where n is the number of patterns. We give a description of the k-means algorithm below.

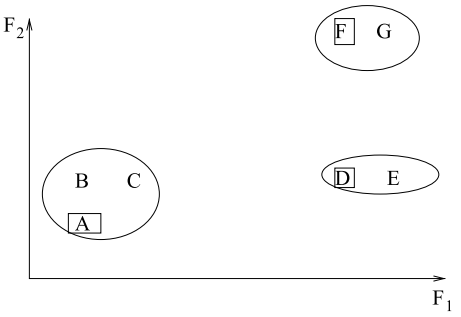

Fig. 2.5 An optimal clustering of the points

k-Means Algorithm

1. Select k initial centroids. One possibility is to select k out of the n points randomly as the initial centroids. Each of them represents a cluster.
2. Assign each of the remaining $n - k$ points to one of these k clusters; a pattern is assigned to a cluster if the centroid of the cluster is the nearest, among all the k centroids, to the pattern.
3. Update the centroids of the clusters based on the assignment of the patterns.
4. Assign each of the n patterns to the nearest cluster using the current set of centroids.
5. Repeat steps 3 and 4 till there is no change in the assignment of points in two successive iterations.

An important feature of this algorithm is that it is sensitive to the selection of the initial centroids and may converge to a local minimum of the squared-error criterion function value if the initial partition is not properly chosen. The squared-error criterion function is given by

$$\sum_{i=1}^{k} \sum_{X \in C_i} \|X - centroid_i\|^2. \tag{2.1}$$

We illustrate the k-means algorithm using the dataset shown in Fig. 2.3. If we consider A, D, and F as the initial centroids, then the resulting partition is shown in Fig. 2.5. For this optimal partition, the centroids of the three clusters are:

- centroid1: $(1.33, 1.66)^t$; centroid2: $(6.45, 2)^t$; centroid3: $(6.4, 2)^t$.
- The corresponding value of the squared error is around 2 units.

The popularity of the k-means algorithm may be attributed to its simplicity. It requires $O(n)$ time as it computes nk distances in each pass and the number of passes may be assumed to be a constant. Also, the number of clusters k is a constant. Further, it needs to store k centroids in the memory. So, the space requirement is also small.

However, it is possible that the algorithm generates a nonoptimal partition by choosing A, B, and C as the initial centroids as depicted in Fig. 2.6. In this case, the three centroids are:

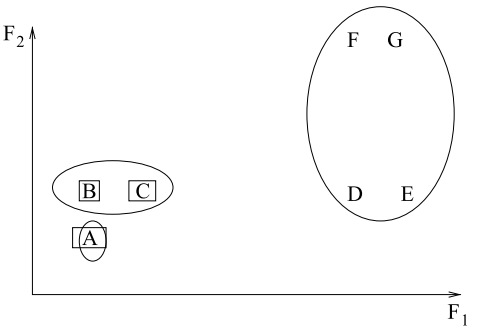

Fig. 2.6 A nonoptimal clustering of the two-dimensional points

- centroid1: $(1, 1)^t$; centroid2: $(1.5, 2)^t$; centroid3: $(6.4, 4)^t$.
- The corresponding squared error value is around 17 units.

2.3 Classification

There are a variety of classifiers. Typically, a set of labeled patterns is used to classify an unlabeled test pattern. Classification involves labeling a test pattern; in the process, either the labeled training dataset is directly used, or an abstraction or model learnt from the training dataset is used. Typically, classifiers learnt from the training dataset are categorized as either generative or discriminative. The Bayes classifier is a well-known generative model where a test pattern X is classified or assigned to class C_i, based on the a posteriori probabilities $P(C_j/X)$ for $j = 1, \ldots, C$ if

$$P(C_i/X) \geq P(C_j/X) \quad \text{for all } j.$$

These posterior probabilities are obtained using the Bayes rule using prior probabilities and the probability distributions of patterns in each of the classes. It is possible to show that the Bayes classifier is optimal; it can minimize the average probability of error. Support Vector Machine (SVM) is a popular discriminative classifier, and it learns a weight vector W and a threshold b from the training patterns from two classes. It assigns the test pattern X to class C_1 (positive class) if $W^t X + b \geq 0$, else it assigns X to class C_2 (negative class).

The Nearest-Neighbor Classifier (NNC) is the simplest and popular classifier; it classifies the test pattern by using the training patterns directly. An important property of the NNC is that its error rate is less than twice the error rate of the Bayes classifier when the number of training patterns is asymptotically large. We briefly describe the NNC, which employs the nearest-neighbor rule for classification.

Nearest-Neighbor Classifier (NNC)

Input: A training set $\mathscr{X} = \{(X_1, C^1), (X_2, C^2), \ldots, (X_n, C^n)\}$ and a test pattern X. Note that X_i, $i = 1, \ldots, n$, and X are some p-dimensional patterns. Further, $C^i \in \{C_1, C_2, \ldots, C_C\}$ where C_i is the ith class label.

Table 2.2 Data matrix

Pattern ID	feature1	feature2	feature3	feature4	Class label
X_1	1.0	1.0	1.0	1.0	C_1
X_2	6.0	6.0	6.0	6.0	C_2
X_3	7.0	7.0	7.0	7.0	C_2
X_4	1.0	1.0	2.0	2.0	C_1
X_5	1.0	2.0	2.0	2.0	C_1
X_6	7.0	7.0	6.0	6.0	C_2
X_7	1.0	2.0	2.0	1.0	C_1
X_8	6.0	6.0	7.0	7.0	C_2

Output: Class label for the test pattern X.
Decision: Assign X to class C^i if $d(X, X_i) = \min_j \ d(X, X_j)$.

We illustrate the NNC using the four-dimensional dataset shown in Table 2.2. There are eight patterns, $X_1, \ldots, X_8$, from two classes C_1 and C_2, four patterns from each class. The patterns are four-dimensional, and the dimensions are characterized by feature1, feature2, feature3, and feature4, respectively. In addition to the four features, there is an additional column that provides the class label of each pattern.

Let the test pattern $X = (2.0, 2.0, 2.0, 2.0)^t$. The Euclidean distances between X and each of the eight patterns are given by

$$d(X, X_1) = 2.0; \quad d(X, X_2) = 8.0; \quad d(X, X_3) = 10.0;$$
$$d(X, X_4) = 1.41; \quad d(X, X_5) = 1.0; \quad d(X, X_6) = 9.05;$$
$$d(X, X_7) = 1.41; \quad d(X, X_8) = 9.05.$$

So, the Nearest Neighbor (NN) of X is X_5 because $d(X, X_5)$ is the smallest (it is 1.0) among all the eight distances. So, $NN(X) = X_5$, and the class label assigned to X is the class label of X_5, which is C_1 here, which means that X is assigned to class C_1. Note that NNC requires eight distances to be calculated in this example. In general, if there are n training patterns, then the number of distances to be calculated to classify a test pattern is $O(n)$.

The nearest-neighbor classifier is popular because:

1. It is easy to understand and implement.
2. There is no learning or training phase; it uses the whole training data to classify the test pattern.
3. Unlike the Bayes classifier, it does not require the probability structure of the classes.
4. It shows good performance. If optimal accuracy is 99.99 %, then with a large training data, it can give at least 99.80 % accuracy.

Even though it is popular, there are some negative aspects. They include:

1. It is sensitive to noise; if the $NN(X)$ is erroneously labeled, then X will be misclassified.
2. It needs to store the entire training data; further, it needs to compute the distances between the test pattern and each of the training patterns. So, the computational requirements can be large.
3. The distance between a pair of points may not be meaningful in high-dimensional spaces. It is known that, as the dimensionality increases, the distance between a point X and its nearest neighbor tends toward the distance between X and its farthest neighbor. As a consequence, NNC may perform poorly in the context of high-dimensional spaces.

Some of the possible solutions to the above problems are:

1. In order to tolerate noise, a modification to NNC is popularly used; it is called the k-Nearest Neighbor Classifier (*kNNC*). Instead of deciding the class label of X using the class label of the $NN(X)$, X is labeled using the class labels of k nearest neighbors of X. In the case of *kNNC*, the class label of X is the label of the class that is the most frequent among the class labels of the k nearest neighbors. In other words, X is assigned to the class to which majority of its k nearest neighbors belong; the value of k is to be fixed appropriately. In the example dataset shown in Table 2.2, the three nearest neighbors of $X = (2.0, 2.0, 2.0, 2.0)^t$ are X_5, X_4, and X_7. All the three neighbors are from class C_1; so X is assigned to class C_1.
2. NNC requires $O(n)$ time to compute the n distances, and also it requires $O(n)$ space. It is possible to reduce the effort by compressing the training data. There are several algorithms for performing this compression; we consider here a scheme based on clustering. We cluster the n patterns into k clusters using the k-means algorithm and use the k resulting centroids instead of the n training patterns. Labeling the centroids is done by using the majority class label in each cluster.

   By clustering the example dataset shown in Table 2.2 using the k-means algorithm, with a value of $k = 2$, we get the following clusters:

   - *Cluster1:* $\{X_1, X_4, X_5, X_7\}$ – Centroid: $(1.0, 1.5, 1.75, 1.5)^t$
   - *Cluster2:* $\{X_2, X_3, X_6, X_8\}$ – Centroid: $(6.5, 6.5, 6.5, 6.5)^t$

   Note that Cluster1 contains four patterns from C_1 and Cluster2 has the four patterns from C_2. So, by using these two representatives instead of the eight training patterns, the number of distance computations and memory requirements will reduce. Specifically, Centroid of Cluster1 is nearer to X than the Centroid of Cluster2. So, X is assigned to C_1 using two distance computations.
3. In order to reduce the dimensionality, several feature selection/extraction techniques are used. We use a feature set partitioning scheme that we explain in detail in the sequel.

Another important classifier is based on *Support Vector Machine*. We consider it next.

Support Vector Machine The support vector machine (SVM) is a very popular classifier. Some of the important properties of the SVM-based classification are:

- The SVM classifier is a discriminative classifier. It can be used to discriminate between two classes. Intrinsically, it supports binary classification.
- It obtains a linear discriminant function of the form $W^t X + b$ from the training data. Here, W is called the weight vector of the same size as the data points, and b is a scalar. Learning the SVM classifier amounts to obtaining the values of W and b from the training data.
- It is ideally associated with a binary classification problem. Typically, one of them is called the negative class, and the other is called the positive class.
- If X is from the positive class, then $W^t X + b > 0$, and if X is from the negative class, then $W^t X + b < 0$.
- It finds the parameters W and b so that the margin between the two classes is maximized.
- It identifies a subset of the training patterns, which are called *support vectors*. These support vectors lie on parallel hyperplanes; negative and positive hyperplanes correspond respectively to the negative and positive classes. A point X on the negative hyperplane satisfies $W^t X + b = -1$, and similarly, a point X on the positive hyperplane satisfies $W^t X + b = 1$.
- The margin between the two support planes is maximized in the process of finding out W and b. In other words, the normal distance between the support planes $W^t X + b = -1$ and $W^t X + b = 1$ is maximized. The distance is $\frac{2}{\|W\|}$. It is maximized using the constraints that every pattern X from the positive class satisfies $W^t X + b \geq +1$ and every pattern X from the negative class satisfies $W^t X + b \leq -1$. Instead of maximizing the margin, we minimize its inverse. This may be viewed as a constrained optimization problem given by

$$\begin{aligned} &\text{Min}_W \quad \|W\|^2 \\ &\text{s.t.} \quad y_i\big(W^t X_i + b\big) \geq 1, \quad i = 1, 2, \ldots, n, \end{aligned}$$

  where $y_i = 1$ if X_i is in the positive class and $y_i = -1$ if X_i is in the negative class.
- The Lagrangian for the optimization problem is

$$L(W, b) = \frac{1}{2}\|W\|^2 - \sum_{i=1}^{n} \alpha_i \big(y_i\big(W^t X - i + b\big) - 1\big).$$

  In order to minimize the Lagrangian, we take the derivative with respect to b and gradient with respect to W, and equating to 0, we get α_is that satisfy

$$\alpha_i \geq 0 \quad \text{and} \quad \sum_{i=1}^{q} \alpha_i y_i = 0,$$

where q is the number of support vectors, and W is given by

$$W = \sum_{i=1}^{q} \alpha_i y_i X_i.$$

- It is possible to view the decision boundary as $W^t X + b = 0$ and W is orthogonal to the decision boundary.

We illustrate the working of the SVM using an example in the two-dimensional space. Let us consider two points, $X_1 = (2, 1)^t$ from the negative class and $X_2 = (6, 3)^t$ from the positive class. We have the following:

- Using $\alpha_1 y_1 + \alpha_2 y_2 = 0$ and observing that $y_1 = -1$ and $y_2 = 1$, we get $\alpha_1 = \alpha_2$. So, we use α instead of α_1 or α_2.
- As a consequence, $W = -\alpha X_1 + \alpha X_2 = (4\alpha, 2\alpha)^t$.
- We know that $W^t X_1 + b = -1$ and $W^t X_2 + b = 1$; substituting the values of W, X_1, and X_2, we get

$$8\alpha + 2\alpha + b = -1,$$
$$24\alpha + 6\alpha + b = 1.$$

  By solving the above, we get $20\alpha = 2$ or $\alpha = \frac{1}{10}$, from which and from one of the above equations we get $b = -2$.
- From $W = (4\alpha, 2\alpha)^t$ and $\alpha = \frac{1}{10}$ we get $W = (\frac{2}{5}, \frac{1}{5})^t$.
- In this simple example, we have started with two support vectors in the two-dimensional case. So, it was easy to solve for αs. In general, there are efficient schemes for finding these values.
- If we consider a point $X = (x_1, x_2)^t$ on the line $x_2 = -2x_1 + 5$, for example, the point $(1, 3)^t$, then $W^t(1, 3)^t - 2 = -1$ as $W = (\frac{2}{5}, \frac{1}{5})^t$. This line is the support line for the points in the negative class. In a higher-dimensional space, it is a hyperplane.
- In a similar manner, any point on the parallel line $x_2 = -2x_1 + 15$, for example, $(5, 5)^t$ satisfies the property that $W^t(5, 5) - 2 = 1$, and this parallel line is the support plane for the positive class. Again in a higher-dimensional space, it becomes a hyperplane parallel to the negative class plane.
- Note that the decision boundary is given by

$$\left(\frac{2}{5}, \frac{1}{5}\right) X - 2 = 0.$$

  So, the decision boundary $\frac{2}{5}x_1 + \frac{1}{5}x_2 - 2 = 0$ lies exactly in the middle of the two support lines and is parallel to both. Note that $(4, 2)^t$ is located on the decision boundary.
- A point $(7, 6)^t$ is in the positive class as $W^t(7, 6)^t - 2 = 2 > 0$. Similarly, $W^t(1, 1)^t - 2 = -1.4 < 0$; so, $(1, 1)^t$ is in the negative class.
- We have discussed what is known as the linear SVM. If the two classes are linearly separable, then the linear SVM is sufficient.

- If the classes are not linearly separable, then we map the points to a high-dimensional space with a hope to find linear separability in the new space. Fortunately, one can implicitly make computations in the high-dimensional space without having to work explicitly in it. It is possible by using a class of kernel functions that characterize similarity between patterns.
- However, in large-scale applications involving high-dimensional data like in text mining, linear SVMs are used by default for their simplicity in training.

2.4 Association Rule Mining

This is an activity that is not a part of either *pattern recognition* or *machine learning* conventionally. An association rule is an implication of the form $A \rightarrow B$, where A and B are disjoint itemsets; A is called the *antecedent*, and B is called the *consequent*. Typically, this activity became popular in the context of market-basket analysis, where one is concerned with the set of items available in a super market, and transactions are made by various customers. In such a context, an association rule provides information on the association between two sets of items that are frequently bought together; this facilitates in strategic decisions that may have a positive commercial impact in displaying the related items on appropriate shelves to avoid congestion or in terms of offering incentives to customers on some products/items.

Some of the features of the association rule mining activity are:

1. The rule $A \rightarrow B$ is not like the conventional implication used in a classical logic, for example, the propositional logic. Here, the rule does not guarantee the purchase of items in B in the same transaction where items in A are bought; it depicts a kind of frequent association between A and B in terms of buying patterns.
2. It is assumed that there is a global set of items I; in the case of market-basket analysis, I is the set of all items/product lines available for sale in a supermarket. Note that A and B are disjoint subsets of I. So, if the cardinality of I is d, then the number of all possible rules is of $O(3^d)$; this is because an item in I can be a part of A or B or none of the two and there are d items. In order to reduce the mining effort, only a subset of the rules that are based on frequently bought items is examined.
3. Popularly, the quantity of an item bought is not used; it is important to consider whether an item is bought in a transaction or not. For example, if a customer buys 1.2 kilograms of Sugar, 3 loafs of Bread, and a tin of Jam in the same transaction, then the corresponding transaction is represented as {Sugar, Bread, Jam}. Such a representation helps in viewing a transaction as a subset of I.
4. In order to mine useful rules, only rules of the form $A \rightarrow B$, where A and B are subsets of frequent itemsets, are explored. So, it is important to consider algorithms for frequent itemset mining. Once all the frequent itemsets are mined, it is required to obtain the corresponding association rules.

Table 2.3 Transaction data

Transaction	Itemset
t_1	{a,c,d,e}
t_2	$\{a, d, e\}$
t_3	$\{b, d, e\}$
t_4	$\{a, b, c\}$
t_5	$\{a, b, c, d\}$
t_6	$\{a, b, d\}$
t_7	$\{a, d\}$

2.4.1 Frequent Itemsets

A transaction t is a subset of the set of items I. An itemset X is a subset of a transaction t if all the items in X have been bought in t. If $\mathscr{T}$ is a set of transactions where $\mathscr{T} = \{t_1, t_2, \ldots, t_n\}$, then the support-set of X is given by

$$\textit{Support-set}(X) = \{t_i | X \text{ is a subset of } t\}.$$

The *support* of X is given by the cardinality of $\textit{Support-set}(X)$ or $|\textit{Support-set}(X)|$. An itemset X is a *frequent itemset* if $\textit{Support}(X) \geq \textit{Minsup}$, where *Minsup* is a user-provided threshold.

We explain the notion of frequent itemset using the transaction data shown in Table 2.3.

Some of the itemsets with their supports corresponding to the data in Table 2.3 are:

- $\textit{Support}(\{a, b, c\}) = 2$; $\textit{Support}(\{a, d\}) = 5$;
- $\textit{Support}(\{b, d\}) = 3$; $\textit{Support}(\{a, c\}) = 3$.

If we use a *Minsup* value of 4, then the itemset $\{a, d\}$ is frequent. Further, $\{a, b, c\}$ is not frequent; we call such itemsets *infrequent*. There is a systematic way of enumerating all the frequent itemsets; this is done by an algorithm called *Apriori*. This algorithm enumerates a relevant subset of the itemsets for examining whether they are frequent or not. It is based on the following observations.

1. Any subset of a frequent itemset is frequent. This is because if A and B are two itemsets such that A is a subset B, then $\textit{Support}(A) \geq \textit{Support}(B)$ because $\textit{Support-set}(A) \subseteq \textit{Support-set}(B)$. For example, knowing that itemset $\{a, d\}$ is frequent, we can infer that the itemsets $\{a\}$ and $\{d\}$ are frequent. Note that in the data shown in Table 2.3, $\textit{Support}(\{a\}) = 6$ and $\textit{Support}(\{d\}) = 6$ and both exceed the *Minsup* value.
2. Any superset of an infrequent itemset is infrequent. If A and B are two itemsets such that A is a superset B, then $\textit{Support}(A) \leq \textit{Support}(B)$. In the example, $\{a, c\}$ is infrequent; one of its supersets $\{a, c, d\}$ is also infrequent. Note that $\textit{Support}(\{a, c, d\}) = 2$ and it is less than the *Minsup* value.

Table 2.4 Printed characters of 1

0	0	1	1	0	0
0	0	1	1	0	0
0	0	1	1	0	0

2.4.1.1 Apriori Algorithm

The Apriori algorithm iterates over two steps to generate all the frequent itemsets from a transaction dataset. Each iteration requires a database scan. These two steps are as follows.

- *Generating Candidate itemsets of size k*. These itemsets are obtained by looking at frequent itemsets of size $k-1$.
- *Generating Frequent itemsets of size k*. This is achieved by scanning the transaction database once to check whether a candidate of size k is frequent or not.

It starts with the empty set (ϕ), which is frequent because the empty set is a subset of every transaction. So, $Support(\phi) = |\mathscr{T}|$, where $\mathscr{T}$ is the set of transactions. Note that ϕ is a size 0 itemset as there are no items in it. It then generates candidate itemsets of size 1; we call such itemsets 1-*itemsets*. Note that every 1-itemset is a candidate. In the example data shown in Table 2.3, the candidate 1-itemsets are $\{a\}, \{b\}, \{c\}, \{d\}, \{e\}$. Now it scans the database once to obtain the supports of these 1-itemsets. The supports are:

$$Support\big(\{a\}\big) = 6; \qquad Support\big(\{b\}\big) = 4; \qquad Support\big(\{c\}\big) = 3;$$
$$Support\big(\{d\}\big) = 6; \qquad Support\big(\{e\}\big) = 3.$$

Using a *Minsup* value of 4, we can observe that frequent 1-itemsets are $\{a\}, \{b\}$, and $\{d\}$. From these frequent 1-itemsets we generate candidate 2-itemsets. The candidates are $\{a, b\}, \{a, d\}$, and $\{b, d\}$. Note that the other 2-itemsets need not be considered as candidates because they are supersets of infrequent itemsets and hence cannot be frequent. For example, $\{a, c\}$ is infrequent because $\{c\}$ is infrequent. A second database scan is used to find the support values of these candidates. The supports are $Support(\{a, b\}) = 3$, $Support(\{a, d\}) = 5$, and $Support(\{b, d\}) = 3$. So, only $\{a, d\}$ is a frequent 2-itemset. So, there can not be any candidates of size 3. For example, $\{a, b, d\}$ is not frequent because $\{a, b\}$ is infrequent.

It is important to note that transactions need not be associated with supermarket buying patterns only. It is possible to view a wide variety of patterns as transactions. For example, consider printed characters of size 3×3 corresponding to character 1 shown in Table 2.4; there are two 1s. In the left-side one is present in the third column of the matrix and the right-side matrix, the pattern is present in column 1. By labeling the locations in such 3×3 matrices using 1 to 9 in a row-major fashion, the two patterns may be viewed as transactions based on the 9 items. Specifically, the transactions are $t_1 : \{3, 6, 9\}$ and $t_2 : \{1, 4, 7\}$, where t_1 corresponds to the left-side pattern, and t_2 corresponds the right-side pattern in Table 2.4. Let us call the left side 1 as Type1 1, and the right side 1 as Type2 1.

Table 2.5 Transactions for characters of 1

TID	1	2	3	4	5	6	7	8	9	Class
t_1	1	0	0	1	0	0	1	0	0	Type1 1
t_2	1	0	0	1	1	0	1	0	0	Type1 1
t_3	1	0	0	1	0	0	1	1	0	Type1 1
t_4	0	0	1	0	0	1	0	0	1	Type2 1
t_5	0	0	1	0	0	1	0	1	1	Type2 1
t_6	0	0	1	0	1	1	0	0	1	Type2 1

So, it is possible to represent data based on categorical features using transactions and mine them to obtain frequent patterns. For example, with a small amount of noise, we can have transaction data corresponding to these 1s as shown in Table 2.5. There are six transactions, each of them corresponding to a 1. By using a *Minsup* value of 3, we get the frequent itemset $\{1, 4, 7\}$ for Type1 1 and the frequent itemset $\{3, 6, 9\}$ for Type2 1. Naturally subsets of these frequent itemsets also are frequent.

2.4.2 Association Rules

In association rule mining there are two important phases:

1. *Generating Frequent Itemsets.* This requires one or more dataset scans. Based on the discussion in the previous subsection, Apriori requires $k + 1$ dataset scans if the largest frequent itemset is of size k.
2. *Obtaining Association Rules.* This step generates association rules based on frequent itemsets. Once frequent itemsets are obtained from the transaction dataset, association rules can be obtained without any more dataset scans, provided that the support of each of the frequent itemsets is stored. So, this step is computationally simpler.

If X is a frequent itemset, then rules of the form $A \rightarrow B$ where $A \subset X$ and $B = X - A$ are considered. Such a rule is accepted if the confidence of the rule exceeds a user-specified confidence value called *Minconf*. The confidence of a rule $A \rightarrow B$ is defined as

$$Confidence(A \rightarrow B) = \frac{Support(A \cup B)}{Support(A)}.$$

So, if the support values of all the frequent itemsets are stored, then it is possible to compute the confidence value of a rule without scanning the dataset.

For example, in the dataset shown in Table 2.3, $\{a, d\}$ is a frequent itemset. So, there are two possible association rules. They are:

1. $\{a\} \rightarrow \{d\}$; its confidence is $\frac{5}{6}$.
2. $\{d\} \rightarrow \{a\}$; its confidence is $\frac{5}{6}$.

So, if the *Minconf* value is 0.5, then both these rules satisfy the confidence threshold.

In the case of character data shown in Table 2.5, it is appropriate to consider rules of the form:

- $\{1, 4, 7\} \rightarrow Type1\ 1$
- $\{3, 6, 9\} \rightarrow Type2\ 1$

Typically, the antecedent of such an association rule or a classification rule is a disjunction of one or more maximally frequent itemsets. A frequent itemset A is maximal if there is no frequent itemset B such that A is a subset of B. This illustrates the role of frequent itemsets in classification.

2.5 Mining Large Datasets

There are several applications where the size of the pattern matrix is large. By *large*, we mean that the entire pattern matrix cannot be accommodated in the main memory of the computer. So, we store the input data on a secondary storage medium like the disk and transfer the data in parts to the main memory for processing. For example, a transaction database of a supermarket chain may consist of trillions of transactions, and each transaction is a sparse vector of a very high dimensionality; the dimensionality depends on the number of product-lines. Similarly, in a network intrusion detection application, the number of connections could be prohibitively large, and the number of packets to be analyzed or classified could be even larger. Another application is the clustering of click-streams; this forms an important part of web usage mining. Other applications include genome sequence mining, where the dimensionality could be running into millions, social network analysis, text mining, and biometrics.

An objective way of characterizing largeness of a data set is by specifying bounds on the number of patterns and features present. For example, a data set having more than billion patterns and/or more than million features is *large*. However, such a characterization is not universally acceptable and is bound to change with the developments in technology. For example, in the 1960s, "large" meant several hundreds of patterns. So, it is good to consider a more pragmatic characterization; large data sets are those that may not fit the main memory of the computer; so, largeness of the data varies with the technological developments. Such large data sets are typically stored on a disk, and each point in the set is accessed from the disk based on processing needs. Note that disk access can be several orders slower compared to the memory access; this property remains in tact even though memory and disk sizes at different points time in the past are different. So, characterizing largeness using this property could be more meaningful.

The above discussion motivates the need for integrating various algorithmic design techniques along with the existing mining algorithms so that they can handle

large data sets. Here, we provide an exhaustive set of design techniques that are useful in this context. More specifically, we offer a unifying framework that is helpful in categorizing algorithms for mining large data sets; further, it provides scope for designing novel efficient mining algorithms.

2.5.1 Possible Solutions

It is important that the mining algorithms that work with large data sets should scale up well. Algorithms having nonlinear time and space complexities are ruled out. Even algorithms requiring linear time and space may not be feasible if the number of dataset scans is large. Based on these observations, it is possible to list the following solutions for mining large data sets.

1. *Incremental Mining.* The basis of incremental mining is that the data is considered sequentially and the data points are processed step by step. In most of the incremental mining algorithms, a small dataset is used to generate an abstraction. New points are processed to update the abstraction currently available without examining the previously seen data points. Also it is important that abstraction generated is as small as possible in size. Such a scheme helps in mining very large-scale datasets.

   We can characterize incremental mining formally as follows. Let

   $$\mathscr{X} = \{(X_1, \theta_1, t_1), (X_2, \theta_2, t_2), \ldots, (X_n, \theta_n, t_n)\}$$

   be the set of n patterns, each represented as a triple, where X_i is the ith pattern, θ_i is the class label of X_i, and t_i is the time-stamp associated with X_i so that $t_i < t_j$ if $i < j$. In incremental mining, as the data is considered sequentially, in a particular order, we may attach time stamps $t_1, t_2, \ldots, t_n$ with the patterns $X_1, X_2, \ldots, X_n$. Let A^k represent the abstraction generated using the first k patterns, and A^n represent the abstraction obtained after all the n patterns are processed. Further, in incremental mining, A^{k+1} is obtained using A^k and X_{k+1} only.
2. *Divide-and-Conquer Approach.* Divide-and-conquer is a well-known algorithm design strategy. It has been used in designing several efficient algorithms. It has been used in efficient data mining. A notable development in this direction is the Map-Reduce framework, which is popular in a variety of data mining applications including text mining.
3. *Mining based on an Intermediate Abstraction*. The idea here is to use one or two database scans to obtain a compact representation of the dataset. Such a representation may fit into main memory. Further processing is based on this abstraction, and it does not require any more dataset scans. For example, as discussed in the previous section, once frequent itemsets are obtained using a small number of database scans, association rules can be obtained without anymore database scans.

In the rest of the section, we examine how these three techniques are used in Clustering, Classification, and Association Rule Mining.

2.5.2 Clustering

2.5.2.1 Incremental Clustering

The basis of incremental clustering is that the data is considered sequentially and the patterns are processed step by step. In most of the incremental clustering algorithms, one of the patterns in the data set (usually the first pattern) is selected to form an initial cluster. Each of the remaining points is assigned to one of the existing clusters or may be used to form a new cluster based on some criterion. Here, a new data item is assigned to a cluster without affecting the existing clusters significantly.

The abstraction A^k varies from algorithm to algorithm, and it can take different forms. One of the popular schemes is when A^k is a set of prototypes or cluster representatives. Leader clustering algorithm is a well-known member of this category. It is described below.

Leader Clustering Algorithm

Input: The dataset to be clustered and a *Threshold value T* provided by the user.
Output: A partition of the dataset such that patterns in each cluster are within a sphere of radius T.

1. Set $k = 1$. Assign the first data point X_1 to cluster C_k. Set the leader of C_k to be $L_k = X_1$.
2. Assign the next data point X to one of the existing clusters or to a new cluster. This assignment is done based on some similarity between the data point and the existing leaders. Specifically, assign the data point X to cluster C_j if $d(X, L_j) < T$; if there are more than one C_j satisfying the threshold requirement, then assign X to one of these clusters arbitrarily. If there is no C_j such that $d(X, L_j) < T$, then increment k, assign X to C_k, and set X to be L_k.
3. Repeat step 2 till all the data points are assigned to clusters.

BIRCH: Balanced Iterative Reducing and Clustering using Hierarchies
BIRCH may be viewed as a hierarchical version of the leader algorithm with some additional representational features to handle large-scale data. It constructs a data structure called the Cluster Feature tree (CF tree), which represents each cluster compactly using a vector called Cluster Feature (CF). We explain these notions using the dataset shown in Table 2.6.

- *Clustering Feature* (*CF*). Let us consider the cluster of 2 points, $\{(1, 1)^t, (2, 2)^t\}$. The CF vector is three-dimensional and is $\langle 2, (3, 3), (5, 5)\rangle$, where the three components of the vector are as follows:
  1. The first component is the number of elements in the cluster, which is 2 here.
  2. The second component is the linear sum of all the points (vectors) in the cluster, which is $(3, 3)$ $(= (1 + 2, 1 + 2))$ in this example.
  3. The third component, squared sum, is the sum of squares of the components of the points in the cluster; here it is $(5, 5)$ $(= (1^2 + 2^2, 1^2 + 2^2))$.

Table 2.6
A two-dimensional dataset

Pattern number	feature1	feature2
1	1	1
2	6	3
3	2	2
4	7	4
5	9	8
6	9	11
7	14	2
8	13	3

- *Merging Clusters.* A major flexibility offered by representing clusters using *CF* vectors is that it is very easy to merge two or more clusters. For example if n_i is the number of elements in C_i, ls_i is the linear sum, and ss_i is the squared sum, then
  CF vector of cluster C_i is $\langle n_i, ls_i, ss_i \rangle$ and
  CF vector of cluster C_j is $\langle n_j, ls_j, ss_j \rangle$, then
  CF vector of the cluster obtained by merging C_i and C_j is
  $\langle n_i + n_j, ls_i + ls_j, ss_i + ss_j \rangle$.
- *Computing Cluster Parameters.* Another important property of the *CF* representation is that several statistics associated with the corresponding cluster can be obtained easily using it. A *statistic* is a function of the samples in the cluster. For example, if a cluster $C = \{X^1, X^2, \ldots, X^q\}$, then

$$\textit{Centroid of } C = \textit{Centroid}_C = \frac{\sum_{j=1}^{q} X_j}{q} = \frac{ls}{q},$$

$$\textit{Radius of } C = R = \left[\frac{\sum_{i=1}^{q}(X_i - \textit{Centroid}_C)^2}{q}\right]^{\frac{1}{2}}$$
$$= \left[\frac{ss_i - 2\frac{ls_i^2}{q} + \frac{ls_i^2}{q^2}}{q}\right]^{\frac{1}{2}}.$$

- At the leaf node level, each cluster is controlled by a user-provided threshold. If T is the threshold, then all the points in the cluster lie in a sphere of radius T. As the clusters are merged to form clusters at a previous level, one can use the merging property of the *CF* vectors.

We show the *CF*-tree generated using the data shown in Table 2.6 in Fig. 2.7. By inserting the first pattern we get the *CF* vector $\langle 1, (1, 1), (1, 1) \rangle$. When two patterns are within a threshold of two units, we put them in the same cluster at the leaf level; for example, $(1, 1)^t$ and $(2, 2)^t$ are placed in the same cluster at the leaf node as shown in the figure. Here we consider nodes that can store two clusters at each

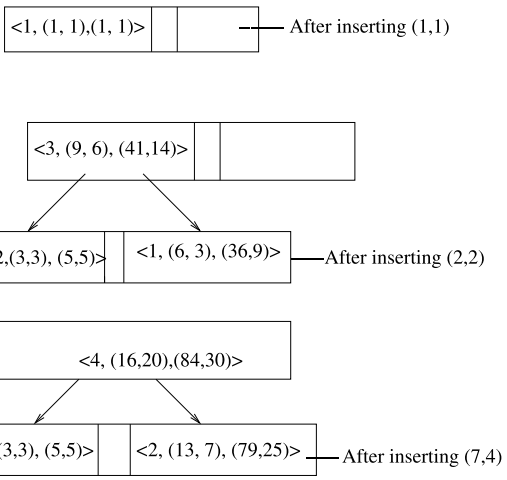

Fig. 2.7 Insertion of the first three patterns

level. By inserting all the eight patterns we get the *CF*-tree shown in Fig. 2.8. Some of the important characteristics of the incremental algorithms are:

1. They require one database scan to generate the clustering of the data. Each pattern is examined only once in the process. In the case of leader clustering algorithm, the clustering is represented by a set of leaders. If the threshold value is small, then a larger number of clusters are generated. Similarly, if the threshold value is large, then the number of clusters is small.
2. BIRCH generates a *CF*-tree using a single database scan. Such an abstraction captures clusters in a hierarchical manner. Merging two smaller clusters to form a bigger cluster is very easy by using the merging property of the corresponding *CF* vectors.
3. The parameters controlling the size of the *CF*-tree are the number of clusters stored in each node of the tree and the threshold value used at the leaf node to fix the size of the clusters.
4. Order-independence is an important property of clustering algorithms. An algorithm is *order-independent* if it generates the same partition for any order in which the data is presented. Otherwise it is *order-dependent*. Unfortunately, in-

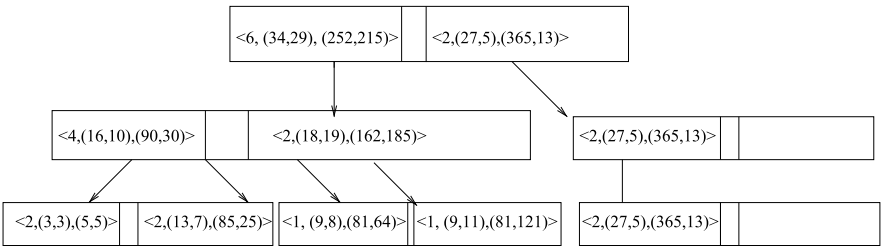

Fig. 2.8 CF-tree for the data

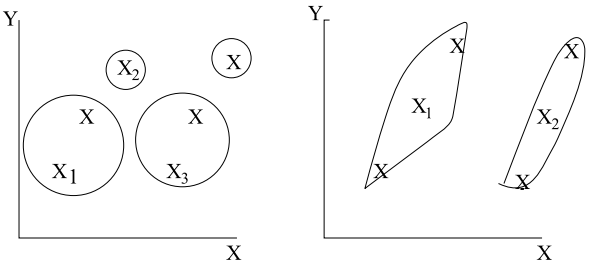

Fig. 2.9 Order-dependence of leader algorithm

cremental algorithms can be order-dependent. This may be illustrated using an example shown in Fig. 2.9.

By choosing the order in different ways, we get different partitions in terms of both the number and size of clusters. For example, by choosing the three points labelled $X_1,\ X_2,\ X_3$ in that order as shown in the left part of the figure, we get four clusters irrespective of the order in which the points $X_4,\ X_5,\ X_6$ are processed. Similarly, by selecting the centrally located points X_4 and X_5 as shown in the right part of the figure as the first two points in the order, we get two clusters irrespective of the order of the remaining four points.

2.5.2.2 Divide-and-Conquer Clustering

Conventionally, designers of clustering algorithms tacitly assume that the data sets fit the main memory. This assumption does not hold when the data sets are large. In such a situation, it makes sense to consider data in parts and cluster each part independently and obtain the corresponding clusters and their representatives. Once we have obtained the cluster representatives for each part, we can cluster these representatives appropriately and realize the clusters corresponding to the entire data set. If two or more representatives from different parts are assigned to some cluster C, then assign the patterns in the corresponding clusters (of these representatives) to C. Specifically, this may be achieved using a two-level clustering scheme depicted in Fig. 2.10. There are n patterns in the data set. All these patterns are stored on a disk. Each part or block of size $\frac{n}{p}$ patterns is considered for clustering at a time. These $\frac{n}{p}$ data points are clustered in the main memory into k clusters using some clustering algorithm. Clustering these p parts can be done either sequentially or in parallel; the number of these clusters corresponding to all the p blocks is pk as there are k cluster in each block. So, we will have pk cluster representatives. By clustering these pk cluster representatives using the same or a different clustering algorithm into k clusters, we can realize a clustering of the entire data set as stated earlier.

It is possible to extend this algorithm to any number of levels. More levels are required if the data set size is very large and the main memory size is small. If single-link algorithm is used to cluster data at both the levels, then we have the following number of distance computations. We consider the number of distances as distance computations form a major part of the computation requirements.

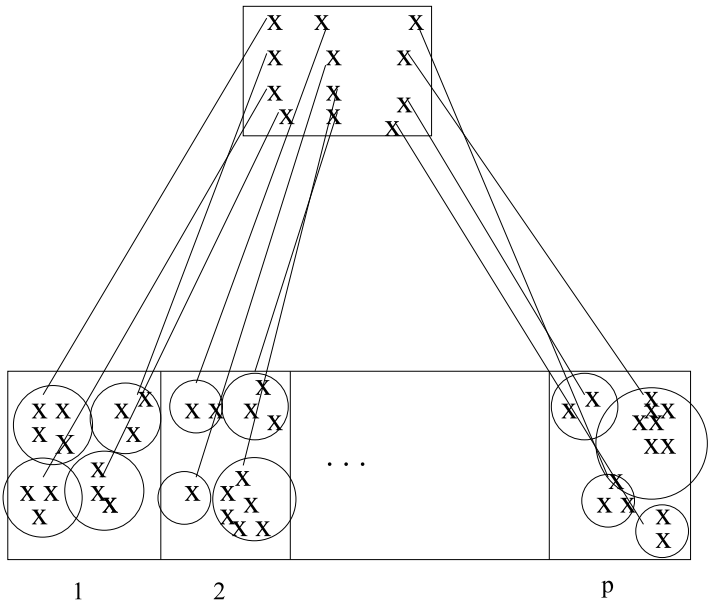

Fig. 2.10 Divide-and-conquer approach to clustering

- *One-level Algorithm.* It does not employ divide-and-conquer. It is the conventional single-link algorithm applied on n data points, which makes $\frac{n(n-1)}{2}$ distance computations.
- *The Two-level Algorithm.* It requires:
  - In each block at the first level, there are $\frac{n}{p}$ points. So, the number of distance computations in each block is $\frac{n}{2p}(\frac{n}{p}-1)$.
  - There are p blocks at the first level. So, the total number of distances at the first level is $\frac{n}{2}(\frac{n}{p}-1)$.
  - There are pk representatives at the second level. So, the number of distances computed at the second level is $\frac{pk(pk-1)}{2}$.
  - So, the total number of distances for the two-level divide-and-conquer algorithm is $\frac{n}{2}(\frac{n}{p}-1) + \frac{pk(pk-1)}{2}$.
- *A Comparison.* The number of distances computed by the conventional single-link and two-level algorithm are shown in Table 2.7 for different values of n, k, and p.

So, there is a great reduction in both time and space requirements if the two-level algorithm is used. Also, the divide-and-conquer algorithm facilitates clustering very large datasets.

Table 2.7 Number of distances computed

No. of data points (n)	No. of blocks (p)	No. of clusters (k)	One-level algorithm	Two-level algorithm
100	2	5	4950	2495
500	20	5	124,750	15,900
1000	20	5	499,500	29,450
10,000	100	5	49,995,000	619,750

2.5.2.3 Clustering Based on an Intermediate Representation

The basic idea here is to generate an abstraction by scanning the dataset once or twice and then use the abstraction, not the original data, for further processing. In order to illustrate the working of this category of algorithms, we use the dataset shown in Table 2.5. We use a database scan to find frequent 1-itemsets. Using a *Minsup* value of 3, we get the following frequent itemsets: {1}, {4}, {7}, {3}, {6}, and {9}; all these items have a support value of 3. We perform one more database scan to construct a tree using only the frequent items. First, we consider transaction t_1 and insert it into a tree as shown in Fig. 2.11. Here, we consider only the frequent items present in t_1; these are 1, 4, and 7. So, these are inserted into the tree by having one node for each item present. The item numbers are indicated inside the nodes; in addition, the count values are also indicated along with the item numbers. For example, in Fig. 2.11(a), $1:1$, $4:1$, and $7:1$ indicate that items 1, 4, and 7 are present in the transaction. Next, we consider t_2, which has the same items as t_1, and so we simply increment the counts as shown in Fig. 2.11(b). After examining all the six transactions, we get the tree shown in Fig. 2.11. In the process, we need to create new branches and nodes appropriately as we encounter new transactions. For example, after considering t_4, we have items 3, 6, and 9 present in it, which prompts us to start a new branch with nodes for the items 3, 6, and 9. At this point, the counts on the right branch of the tree for these items are $3:1$, $6:1$, and $9:1$, respectively. It is possible to store the items in a transaction in any order, but we used the item numbers in increasing order.

Note that the two branches of the tree, which is called Frequent-Pattern tree or FP-tree, correspond to two different clusters; here each cluster corresponds to a different class of 1s. Some of the important features of this class of algorithms are:

1. They require only two scans of the database. This is because each data item is examined only twice. An abstraction is generated, and it is used for further processing. Centroids, leaders, and FP-tree are some example abstractions.
2. The intermediate representation is useful in other important mining tasks like association rule mining, clustering, and classification. For example, the FP-tree has been successfully used in association rule mining, clustering, and classification.
3. Typically, the space required by the intermediate representation could be much smaller than the space required by the entire data set. So, it is possible to store it in compact manner in the main memory.

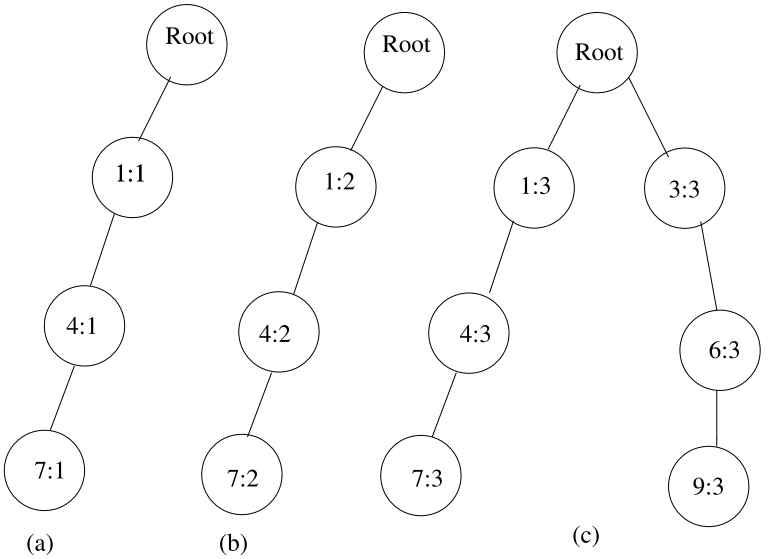

Fig. 2.11 A tree structure for the character patterns

There are several other types of intermediate representations. Some of them are:

- It is possible to reduce the computational requirements of clustering by using a random subset of the dataset.
- An important and not systematically pursued direction is to use a compression scheme to reduce the time and memory required to store the data. The compression scheme may be lossy or nonlossy. Use the compressed data for further processing. This direction will be examined in a great detail in the rest of the book.

2.5.3 Classification

It is also possible to exploit the three paradigms in classification. We discuss these directions next.

2.5.3.1 Incremental Classification

Most of the classifiers can be suitably altered to handle incremental classification. We can easily modify the NNC to perform incremental classification. This can be done by incrementally updating the nearest neighbor of the test pattern. The specific incremental algorithm for NNC is:

1. Let $\mathscr{A}_k$ be the nearest neighbor of the test pattern X after examining training patterns $X_1,\ X_2,\ \ldots,\ X_k$.

2. Next, when X_{k+1} is encountered, we update the nearest neighbor of X to get $\mathscr{A}_{k+1}$ using $\mathscr{A}_k$ and X_{k+1}.
3. Repeat step 2 till $\mathscr{A}_n$ is obtained.

We illustrate it using the dataset shown in Table 2.2. Consider the test pattern $X = (2.0, 2.0, 2.0, 2.0)^t$ and $X_1,\ X_2,\ X_3,\ X_4$. The nearest neighbor of X out of these four points is X_4, which is at a distance of 1.414 units; So, $\mathscr{A}_4$ is X_4. Now, if we encounter X_5, then $\mathscr{A}_5$ gets updated, and it is X_5 because $d(X, X_5) = 1.0$, and it is smaller than $d(X, \mathscr{A}_4)$, which is 1.414. Proceeding further in this manner, we note that $\mathscr{A}_8$ is X_5; so, X is assigned to C_1 as the class label of X_5 is C_1.

In a similar manner, it is possible to visualize an incremental version of the *kNNC*. For example, the three nearest neighbors of X after examining the first four patterns in Table 2.2 are $X_4,\ X_1$, and X_2. Now if we encounter X_5, then the three nearest neighbors are $X_5,\ X_4$, and X_1. After examining all the eight patterns, we get the three nearest neighbors of X to be $X_5,\ X_4$, and X_7. All the three neighbors are from class C_1; so, we assign X to C_1.

2.5.3.2 Divide-and-Conquer Classification

It is also possible to exploit the divide-and-conquer paradigm in classification. Even though it is possible to use it along with a variety of classifiers, we consider it in the context of NNC. It is possible to use the division across either rows or columns of the data matrix.

- *Division across the rows.* It may be described as follows:

  1. Let the n rows of the data matrix be partitioned into p blocks, where there are $\frac{n}{p}$ rows in each block.
  2. Obtain the nearest neighbor of the test pattern X in each block using $\frac{n}{p}$ distances. Let the nearest neighbor of X in the ith block be X^i, and its distance from X be di.
  3. Let dj be the minimum of the values $d1,\ d2,\ \ldots, dp$. Then $NN(X) = X^j$. Ties may be arbitrarily broken.

  Note that computations in steps 2 and 3 can be parallelized to a large extent. We illustrate this algorithm using the data shown in Table 2.2. Let us consider two ($p = 2$) blocks such that:

  – $Block1 = \{X_1,\ X_2,\ X_3,\ X_4\}$;
  – $Block2 = \{X_5,\ X_6,\ X_7,\ X_8\}$.

  Now for the test pattern $X = (2.0, 2.0, 2.0, 2.0)^t$, the nearest neighbors in the two blocks are $X^1 = X_4$ and $X^2 = X_5$. Note that their distances from X are $d1 = 1.414$ and $d2 = 1.0$, respectively. So, X^2, which is equal to X_5, is the nearest neighbor of X as the distance $d2$ is the smaller of the two. It is possible to consider unequal-size partitions also and still obtain the nearest neighbor. Also, it is possible to have a divide-and-conquer *kNNC* using a variant of the above algorithm.

- *Division among the columns.* An interesting situation emerges when the columns are grouped together. It can lead to novel pattern generation or *pattern synthesis*. The specific algorithm is given below:

  1. Divide the number of features d into p blocks, where each block has $\frac{d}{p}$ features. Consider data corresponding to each of these blocks in each of the classes.
  2. Divide the test pattern X into p blocks; let the corresponding subpatterns be $X^1, X^2, \ldots, X^p$, respectively.
  3. Find the nearest neighbor of each X^i for $i = 1, 2, \ldots, p$ from the corresponding ith block of each class.
  4. Concatenate these nearest subpatterns of the corresponding subpattern of the test pattern obtained for each class separately. Among these concatenated patterns, obtain the nearest pattern to X; assign the class label of the nearest concatenated pattern to X.

  We explain the working of this scheme using the example data shown in Table 2.2 and the test pattern $X = (2.0, 2.0, 2.0, 2.0)^t$. Let $p = 2$. Let the two feature set blocks be

  – $Block1 = \{feature1, feature2\}$;
  – $Block2 = \{feature3, feature4\}$.

  Correspondingly, the test pattern has two blocks, $X^1 = (2.0, 2.0)^t$ and $X^2 = (2.0, 2.0)^t$. The training data after partitioning into two feature blocks and reorganizing so that all the patterns in class are put together is shown in Table 2.8. Note that, for X^1, the nearest neighbor from C_1 can be either the first subpattern of X_5, which is denoted by X_5^1, or X_7^1; we resolve the tie in favor of the first pattern, which is X_5^1 as X_5 appears before X_7 in the table. Further, the nearest subpattern from C_2 for X^1 is X_2^1. Similarly, for the second subpattern X^2 of X, the nearest neighbors from C_1 and C_2 respectively are X_4^2 and X_2^2. Now, concatenating the nearest subpatterns from the two classes, we have

  – $C_1 - X_5^1 : X_4^2$, which is $(1.0, 2.0, 2.0, 2.0)^t$;
  – $C_2 - X_2^1 : X_2^2$, which is $(6.0, 6.0, 6.0, 6.0)^t$.

  Out of these two patterns, the pattern from C_1 is nearer to X than the pattern from C_2, the corresponding distances being 1.0 and 8.0, respectively. So, we assign X to C_1.

  There are some important points to be considered here:

  1. In the above example, both the concatenated patterns are already present in the data. However, it is possible that novel patterns are generated by concatenating the nearest subpatterns. For example, consider the test pattern $Y = (1.0, 2.0, 1.0, 1.0)^t$. In this case, the nearest subpatterns from C_1 and C_2 for $Y^1 = (1.0, 1.0)^t$ and $Y^2 = (1.0, 1.0)^t$ are given below:

     – The nearest neighbors of Y^1 from C_1 and C_2 respectively are X_5^1 and X_2^1.

- The nearest neighbors of Y^2 from C_1 and C_2 respectively are X_1^2 and X_2^2.
- Concatenating the nearest subpatterns from C_1, we get $(1.0, 2.0, 1.0, 1.0)^t$.
- Concatenating the nearest subpatterns from C_2, we get $(6.0, 6.0, 6.0, 6.0)^t$.

So, Y is classified as belonging to C_1 because the concatenated pattern $(1.0, 2.0, 1.0, 1.0)^t$ is closer to Y than the pattern from C_2. Note that in this case, the concatenated pattern is the novel pattern $(1.0, 2.0, 1.0, 1.0)^t$, which is not a part of the training data from C_1. So, this scheme has the potential to generate novel patterns from each of the classes and use them in decision making. In general, if there are p blocks and n_i patterns in class C_i, the space size of all possible concatenated patterns in the class is n_i^p, which can be much larger than n_i.

2. Even though the effective search space size or number of patterns examined from the ith class is n_i^p, the actual effort involved in finding the nearest concatenated pattern is of $O(n_i p)$, which is linear.
3. There is no need to compute the distance between X and concatenated nearest subpatterns from each class separately if an appropriate distance function is used. For example, if we use the squared Euclidean distance, then the distance between the test pattern X and the concatenated subpatterns from a class is the sum of the distances between the corresponding subpatterns. Specifically,

$$d^2\big(X, CN_i(X)\big) = \sum_{j=1}^{p} d^2(X^j, NN_i\big(X^j\big),$$

where $CN_i(X)$ is the concatenated nearest subpattern of $X^j s$ from class C_i, and $NN_i(X^j)$ is the nearest subpattern of X^j from C_i. For example,

- The nearest subpattern of X^1 from C_1 is X_5^1, and that of X^2 is X_4^2.
- The corresponding squared Euclidean distances are $d^2(X^1, X_5^1) = 1.0$ and $d^2(X^2, X_4^2) = 0.0$.
- So, the distance between X and the concatenated pattern $(1.0, 2.0, 2.0, 2.0)^t$ is $1.0 + 0.0 = 1.0$.
- Similarly, for C_2, the nearest subpatterns of X^1 and X^2 are X_2^1 and X_2^2, respectively.
- The corresponding distances are $d^2(X^1, X_2^1) = 32$ and $d^2(X^2, X_2^2) = 32$. So, $d^2(X, CN_2(X)) = 32 + 32 = 64$.

4. It is possible to extend this partition-based scheme to the *kNNC*.

2.5.3.3 Classification Based on Intermediate Abstraction

Here we also consider the *NNC*. There could be different intermediate representations possible. Some of them are:

Table 2.8 Reorganized data matrix

Pattern ID	feature1	feature2	feature3	feature4	Class label
X_1	1.0	1.0	1.0	1.0	C_1
X_4	1.0	1.0	2.0	2.0	C_1
X_5	1.0	2.0	2.0	2.0	C_1
X_7	1.0	2.0	2.0	1.0	C_1
X_2	6.0	6.0	6.0	6.0	C_2
X_3	7.0	7.0	7.0	7.0	C_2
X_6	7.0	7.0	6.0	6.0	C_2
X_8	6.0	6.0	7.0	7.0	C_2

1. *Clustering-based.* Cluster the training data and use the cluster representatives as the intermediate abstraction. Clustering could be carried out in each class separately. The resulting clusters may be interpreted as subclasses of the respective classes. For example, consider the two-class four-dimensional dataset shown in Table 2.2. By clustering the data in each class separately using the k-means algorithm with $k = 2$ we get the following centroids:
   - C_1. By selecting X_1 and X_5 as the initial centroids, the clusters obtained using the k-means algorithm are $C_{11} = \{X_1\}$ and $C_{12} = \{X_4, X_5, X_7\}$, and the centroids of these clusters are $(1.0, 1.0, 1.0, 1.0)^t$ and $(1.0, 1.66, 2.0, 1.66)^t$, respectively. Here, C_{11} and C_{12} are the first and second clusters obtained by grouping data in C_1.
   - C_2. By selecting X_2 and X_3 as the initial centroids, using the k-means algorithm, we get the clusters $C_{21} = \{X_2, X_6, X_8\}$ and $C_{22} = \{X_3\}$, and the respective centroids are $(6.33, 6.33, 6.33, 6.33)^t$ and $(7.0, 7.0, 7.0, 7.0)^t$.
   - *Classification of* X. Using the four centroids, two from each class, instead of using all the eight training points, we classify the test pattern $X = (2.0, 2.0, 2.0, 2.0)^t$. The distances between X and these four centroids are $d(X, C_{11}) = 2.0$, $d(X, C_{12}) = 1.22$, $d(X, C_{21}) = 8.66$, and $d(X, C_{22}) = 10.0$. So, X is closer to C_{12}, which is a cluster (or a subclass) in C_1; as a consequence, X is assigned to C_1.
2. *FP-Tree based Abstraction*. Here we consider using an abstraction based on frequent itemsets in classification. For example, consider the transaction dataset shown in Table 2.5 and the corresponding FP-tree structure shown in Fig. 2.11(c). Such an abstraction can be used in classification. The data in the table has two classes, Type1 1 and Type2 1. Now consider a test pattern, which is a noisy version of Type1 1 given by $(1, 0, 1, 1, 0, 0, 1, 0, 0)^t$, which means the corresponding itemset using frequent items with *Minsup* value of 3 is $\{1, 3, 4, 7\}$. This pattern aligns better with the left branch of the tree in Fig. 2.11 than the right branch. So, we assign it to *Type1* 1 as the left branch represents *Type* 1 class. In the process of alignment, we find out the nearest branch in the tree in terms of the common items present in the branch and in the transaction.

Table 2.9 Transaction data for incremental mining

Transaction	Itemset given	Itemset in frequency order
t_1	$\{a, c, d, e\}$	$\{a, d\}$
t_2	$\{a, d, e\}$	$\{a, d\}$
t_3	$\{b, d, e\}$	$\{d, b\}$
t_4	$\{a, c\}$	$\{a\}$
t_5	$\{a, b, c, d\}$	$\{a, d, b\}$
t_6	$\{a, b, d\}$	$\{a, d, b\}$
t_7	$\{a, b, d\}$	$\{a, d, b\}$

2.5.4 Frequent Itemset Mining

In association rule mining, an important and time-consuming step is frequent itemset generation. So, we consider frequent itemset mining here.

2.5.4.1 Incremental Frequent Itemset Mining

There are incremental algorithms for frequent itemset mining. They do not follow the incremental mining definition given earlier. They may require an additional database scan. We discuss the incremental algorithm next.

1. Consider a block of m transactions, $Block1$, to find the frequent itemsets. Store the frequent itemsets along with their supports. If an itemset is infrequent, but all its subsets are frequent, then it is a *border set*. Obtain the set of such border sets. Let F_1 and B_1 be the frequent and border sets from $Block1$.
2. Now let the database be extended by adding a block, $Block2$, of transactions. Find the frequent itemsets and border set in $Block2$. Let them be F_2 and B_2.
3. We update the frequent itemsets as follows:

   - If an itemset is present in both F_1 and F_2, then it is frequent.
   - If an itemset is infrequent in both the blocks, then it is infrequent.
   - If an itemset is frequent in F_1 but not in F_2, it can be eliminated by using the support values.
   - Itemsets absent in F_1 but frequent in the union of the two blocks can be obtained by using the notion of *promoted border*. This happens when an itemset that is a border set in $Block1$ becomes frequent in the union of the two blocks. If such a thing happens, then additional candidates are generated and tested using another database scan.

   We illustrate the algorithm using the dataset shown in Table 2.9 and *Minsup* value of 4; note that the second column in the table gives the transactions. Let $Block1$ consist the first four transactions, that is, from t_1 to t_4. The various sets along with the frequencies are:

- $F_1 = \{a:3\}, \{c:2\}, \{d:3\}, \{e:3\}, \{a,c:2\}, \{a,d:2\}, \{a,e:2\}, \{d,e:3\}, \{a,d,e:2\}$.
- $B_1 = \{b:1\}, \{c,d:1\}, \{c,e:1\}$.

Now we encounter the incremental portion or $Block2$ consisting of remaining three transactions from Table 2.9. For this part, the sets F_2 and B_2 are:

- $F_2 = \{a:3\}, \{b:3\}, \{d:3\}, \{a,b:3\}, \{b,d:3\}, \{a,d:3\}, \{a,b,d:3\}$.
- $B_2 = \{c:1\}$.

Now we know from F_1 and F_2 that $\{a:6\}$, $\{d:6\}$, and $\{a,d:5\}$ are present in both F_1 and F_2. So, they are frequent. Further note that $\{b:4\}$, a border set in $Block1$ gets promoted to become frequent. So, we add it to the frequent itemsets. We also need to consider $\{a,b\}$, $\{b,d\}$, and $\{a,b,d\}$, which may become frequent. However, $\{b,c\}$ and $\{b,e\}$ need not be considered because $\{c\}$ and $\{e\}$ are infrequent. Now we need to make a scan of the database to decide that $\{a,b:4\}$ and $\{b,d:4\}$ are frequent, but not $\{a,b,d:3\}$.

2.5.4.2 Divide-and-Conquer Frequent Itemset Mining

The divide-and-conquer strategy has been used in mining frequent itemsets. The specific algorithm is as follows.

Input: Transaction Data Matrix and *Minsup* value
Output: Frequent Itemsets

1. Divide the transaction data into p blocks so that each block has $\frac{n}{p}$ transactions.
2. Obtain frequent itemsets in each of the blocks. Let F_i be the set of frequent itemsets in the ith block.
3. Take the union of all the frequent itemsets; let it be F. That means $F = \bigcup_{i=1}^{p} F_i$.
4. Use one more database scans to find the supports of itemsets in F. Those satisfying the *Minsup* threshold are the frequent itemsets. Collect them in F_{final}, which is the set of all the frequent itemsets.

Some of the features of this algorithm are as follows:

1. The most important feature is that if an itemset is infrequent in all the p blocks, then it cannot be frequent.
2. This is a two-level algorithm, and it considers only those itemsets that are frequent at the first level for the possibility of being frequent.
3. The worst-case scenario emerges when at the end of the first level all the itemsets are members of F. This can happen in the case of datasets where the transaction are dense or nonsparse. In such a case, using an *FP*-tree that stores the itemsets in a compact manner can be used.

We explain this algorithm using the data shown in Table 2.9. Let us consider two blocks and *Minsup* value of 4, which means a value of 2 in each block.

- Let $Block1 = \{t_1, t_2, t_3, t_4\}$; $Block2 = \{t_5, t_6, t_7\}$.

- $F_1 = \{a\}, \{c\}, \{d\}, \{e\}, \{a, c\}, \{a, d\}, \{a, e\}, \{d, e\}, \{a, d, e\}$.
- $F_2 = \{a\}, \{b\}, \{d\}, \{a, b\}, \{b, d\}, \{a, d\}, \{a, b, d\}$.
- $F = \{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{a, b\}, \{a, c\}, \{a, d\}, \{a, e\}, \{b, d\}, \{d, e\}, \{a, b, d\}$.
- We examine the elements of F and another dataset scan to get F_{final}:
  $F_{\text{final}} = \{a : 6\}, \{b : 4\}, \{d : 6\}, \{a, b : 4\}, \{a, d : 5\}, \{b, d : 4\}$.

2.5.4.3 Intermediate Abstraction for Frequent Itemset Mining

It is possible to read the database once or twice and produce an abstraction and use this abstraction for obtaining frequent itemsets. The most popular abstraction in this context is the Frequent Pattern Tree or FP-tree. It is constructed using two database scans. It has been used in Clustering and Classification. However, it was originally proposed for obtaining the frequent itemsets. The detailed algorithm for constructing an FP-tree is given below:

Input: Transaction Database and *Minsup*.
Output: FP-tree.

1. Scan the dataset once to get the frequent 1-itemsets using the *Minsup* value.
2. Scan the database once more and in each transaction ignore the infrequent items and insert the remaining part of the transaction in decreasing order of support of the items. Also maintain the frequency counts along with items such that if multiple transactions share the same subsets of items, then they are inserted into the same branch of the tree as shown in Fig. 2.11. The frequency counts of the items in the branch are updated appropriately instead of storing them in multiple branches.

Construction of the FP-tree was discussed using the data shown in Table 2.5 and Fig. 2.11. By examining the FP-tree shown in Fig. 2.11(c), it is possible to show that $\{1, 4, 7\}$ and $\{3, 6, 9\}$ are the two maximal frequent itemsets. Each of them corresponds to a type of 1 (character 1), and also each itemset shares a branch in the tree. Once the tree is obtained, frequent itemsets are found by going through the tree in a bottom-up manner. It starts with a suffix based on less frequent items present in the tree. This is efficiently done using an index structure.

We illustrate the frequent itemset mining using the data shown in Table 2.9. The corresponding FP-tree is shown in Fig. 2.12. Some of the details related to the construction of the tree and finding frequent itemsets are:

- The frequent 1-itemsets are $\{a : 6\}$, $\{d : 6\}$, and $\{b : 4\}$ by using a value of 4 for *Minsup* and data in Table 2.9.
- We rewrite the transactions using the frequency order and *Minsup* information. Infrequent items are deleted, and frequent items are ordered in decreasing order of the frequency. Ties are broken based on lexicographic order. The modified transactions are shown in column 3 of the table.
- By inserting the modified transactions, we get the FP-tree shown in Fig. 2.12.

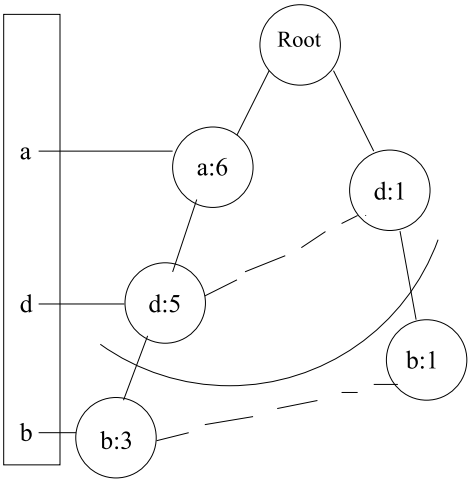

Fig. 2.12 An example FP-tree

- In order to mine the frequent itemsets from the tree, we start with the least frequent among the frequent items, which is b in this case, and mine for all the itemsets from the tree with b as the suffix. For this, we consider the FP-tree above the item b as shown by the curved line segment. Item d occurs in both the branches with frequencies 5 and 1, respectively. However, in terms of co-occurrence along with b, which has a frequency of 3 in the left branch, we need to consider a frequency of 3 for d and a. This is because they concurred in three transactions only along with b. Similarly, from the right branch we know that b and d co-occurred once. From this we get the frequencies of $\{b, d\}$ and $\{a, b\}$ to be 3 from the left branch; in addition, from the right branch we get a frequency of 1 for $\{b, d\}$. This means that the cumulative frequency of $\{b, d\}$ is 4, and so it is frequent, but not $\{a, b\}$ with a frequency of 3 using the value of 4 for *Minsup*.
- Next, we consider item d that appears after b in the bottom-up order of frequency. Note that d has a frequency of 6, and by using it as the suffix we get the itemset $\{a, d\}$, which has a frequency of 5 from the left branch, and so it is frequent.
- Finally, we consider a, which has a frequency of 6, and so the itemset $\{a\}$ is frequent.
- Based on the above-mentioned conditional mining, we get the following frequent itemsets: $\{a : 6\}$, $\{d : 6\}$, $\{a, d : 5\}$, $\{b : 4\}$, and $\{b, d : 4\}$.

2.6 Summary

Data mining deals with large-scale datasets, which may not fit into the main memory. So, the data is stored on a secondary storage, and it is transferred in parts into the memory based on need. Multiple scans of such large databases can be prohibitive in terms of computation time. So, in order to perform some of the data mining tasks like clustering, classification, and frequent itemset mining, it is important to

have some scalable approaches. Specifically, schemes requiring a small number of database scans are important. In this chapter, conventional algorithms used for data mining were discussed first.

There are three different directions for dealing with large-scale datasets. These are based on incremental mining, divide-and-conquer approaches, and an intermediate representation. In an incremental algorithm, each data point is processed only once; so, a single database scan is required for mining. Divide-and-conquer is a well-known algorithm design strategy, and it can be exploited in the context of the data mining tasks including clustering, classification, and frequent itemset mining. The third direction deals with generating an intermediate representation by scanning the database once or twice and uses the abstraction, instead of the data, for further processing. Tree structures like CF-tree and FP-tree can be good examples of intermediate representations. Such trees can be built from the data very efficiently.

2.7 Bibliographic Notes

Important data mining tools including clustering, classification, and association rule mining are discussed in Pujari (2001). A good discussion on clustering is provided in the books by Anderberg (1973) and Jain and Dubes (1988)). They discuss the single-link algorithm and k-means algorithm in a detailed manner. Analysis of these algorithms is provided in Jain et al. (1999).The k-means algorithm was originally proposed by MacQueen (1967). Initial seed selection is an important step in the k-means algorithm. Babu and Murty (1993) use genetic algorithms for initial seed selection. Arthur and Vassilvitskii (2007) presented a probabilistic seed selection scheme. The single-link algorithm was proposed by Sneath (1957). An analysis of the convergence properties of the k-means algorithm is provided by Selim and Ismail (1984). Using the k-means step in genetic algorithm-based clustering, which converges to the global optimum, is proposed and discussed by Krishna and Murty (1999).

An authoritative treatment on classification is provided in the popular book by Duda et al. (2000). A comprehensive treatment of the nearest-neighbor classifiers is provided by Dasarathy (1990). Prototype selection is important in reducing the computational effort of the nearest-neighbor classifier. Ravindra Babu and Murty (2001) study prototype selection using genetic algorithms. Jain and Chandrasekaran (1982) discuss the problems associated with dimensionality and sample size. Sun et al. (2013) propose a feature selection based on dynamic weights for classification. The problems associated with computing nearest neighbors in high-dimensional spaces is discussed by François et al. (2007) and Radovanović et al. (2009) . Even though Vapnik (1998) is the proponent of SVMs, they were popularized by the tutorial paper by Burges (1998).

Apriori algorithm for efficient mining of frequent itemsets and association rules was introduced by Agrawal and Srikant (1994). The FP-tree for mining frequent itemsets without candidate generation was proposed by Han et al. (2000). Compression of frequent itemsets using clustering was carried out by Xin et al. (2005).

Ananthanarayana et al. (2003) use a variant of the FP-tree, which can be built using one database scan. The role of frequent itemsets in clustering was examined by Fung (2002). Yin and Han (2003) use frequent itemsets in classification. The role of discriminative frequent patterns in classification is analyzed by Cheng et al. (2007).

The survey paper by Berkhin (2002) discusses a variety of clustering algorithms and approaches that can handle large datasets. Different paradigms for clustering large datasets was presented by Murty (2002). The book by Xu and Wunsch (2009) on clustering offers a good discussion on clustering large datasets. A major problem with distance-based clustering and classification algorithms is that discrimination becomes difficult in high-dimensional spaces. Clustering paradigms for high-dimensional data are discussed by Kriegel et al. (2009). The Leader algorithm for incremental data clustering is described in Spath (1980). BIRCH is an incremental hierarchical platform for clustering, and it is proposed by Zhang (1997). Vijaya et al. (2005) propose another efficient hierarchical clustering algorithm based on leaders. Efficient clustering using frequent itemsets was presented by Ananthanarayana et al. (2001). Murty and Krishna (1980) propose a divide-and-conquer framework for efficient clustering. Guha et al. (2003) proposed a divide-and-conquer algorithm for clustering stream data. Ng and Han (1994) propose two efficient randomized algorithms in the context of partitioning around medoids.

Viswanath et al. (2004) use a divide-and-conquer strategy on the columns of the data matrix to improve the performance of the *kNNC*. Fan et al. (2008) have developed a library, called LIBLINEAR, for dealing with large-scale classification using logistic regression and linear SVMs. Yu et al. (2003) use CF-tree based clustering in training linear SVM classifiers efficiently. Asharaf et al. (2006) use a modified version of the CF-tree for training kernel SVMs. Ravindra Babu et al. (2007) have reported results on *KNNC* using run-length-coded data. Random forests proposed by Breiman (2001) is one of the promising classifiers to deal with high-dimensional datasets.

The book by Han et al. (2012) provides a wider and state-of-the-art coverage of several data mining tasks and applications. Topic analysis has become a popular activity after the proposal of latent Dirichlet allocation by Blei (2012). Yin et al. (2012) combine community detection with topic modeling in analyzing latent communities. In text mining and information retrieval, Wikipedia is used (Hu et al. (2009)) as an external knowledge source. Currently, there is a growing interest in analyzing Big Data (Russom (2011)) and Map-Reduce (Pavlo et al. (2009)) framework to deal with large datasets.

References

R. Agrawal, R. Srikant, Fast algorithms for mining association rules, in *Proceedings of International Conference on VLDB* (1994)

V.S. Ananthanarayana, M.N. Murty, D.K. Subramanian, Efficient clustering of large data sets. Pattern Recognit. **34**(12), 2561–2563 (2001)

V.S. Ananthanarayana, M.N. Murty, D.K. Subramanian, Tree structure for efficient data mining using rough sets. Pattern Recognit. Lett. **24**(6), 851–862 (2003)

M.R. Anderberg, *Cluster Analysis for Applications* (Academic Press, New York, 1973)
D. Arthur, S. Vassilvitskii, K-means++: the advantages of careful seeding, in *Proceedings of ACM-SODA* (2007)
S. Asharaf, S.K. Shevade, M.N. Murty, Scalable non-linear support vector machine using hierarchical clustering, in *ICPR, vol. 1* (2006) pp. 908–911
G.P. Babu, M.N. Murty, A near-optimal initial seed value selection for k-means algorithm using genetic algorithm. Pattern Recognit. Lett. **14**(10) 763–769 (1993)
P. Berkhin, Survey of clustering data mining techniques. Technical Report, Accrue Software, San Jose, CA (2002)
D.M. Blei, Introduction to probabilistic topic models. Commun. ACM **55**(4), 77–84 (2012)
L. Breiman, Random forests. Mach. Learn. **45**(1), 5–32 (2001)
C.J.C. Burges, A tutorial on support vector machines for pattern recognition. Data Min. Knowl. Discov. **2**, 121–168 (1998)
H. Cheng, X. Yan, J. Han, C.-W. Hsu, Discriminative frequent pattern analysis for effective classification, in *Proceedings of ICDE* (2007)
B.V. Dasarathy, *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques* (IEEE Press, Los Alamitos, 1990)
R.O. Duda, P.E. Hart, D.J. Stork, *Pattern Classification* (Wiley-Interscience, New York, 2000)
R.-E. Fan, K.-W. Chang, C.-J. Hsich, X.-R. Wang, C.-J. Lin, LIBLINEAR: a library for large linear classification. J. Mach. Learn. Res. **9**, 1871–1874 (2008)
D. François, V. Wertz, M. Verleysen, The concentration of fractional distances. IEEE Trans. Knowl. Data Eng. **19**(7), 873–885 (2007)
B.C.M. Fung, Hierarchical document clustering using frequent itemsets. M.Sc. Thesis, Simon Fraser University (2002)
S. Guha, A. Meyerson, N. Mishra, R. Motwani, L. O'Callaghan, Clustering data streams: theory and practice. IEEE Trans. Knowl. Data Eng. **15**(3), 515–528 (2003)
J. Han, J. Pei, Y. Yin, Mining frequent patterns without candidate generation, in *Proc. of ACM-SIGMOD* (2000)
J. Han, M. Kamber, J. Pei, *Data Mining—Concepts and Techniques* (Morgan-Kauffman, San Mateo, 2012)
X. Hu, X. Zhang, C. Lu, E.K. Park, X. Zhou, Exploiting Wikipedia as external knowledge for document clustering, in *ACM SIGKDD, KDD* (2009)
A.K. Jain, B. Chandrasekaran, Dimensionality and sample size considerations in pattern recognition practice, in *Handbook of Statistics*, ed. by P.R. Krishnaiah, L. Kanal (1982), pp. 835–855
A.K. Jain, R.C. Dubes, *Algorithms for Clustering Data* (Prentice-Hall, Englewood Cliffs, 1988)
A.K. Jain, M.N. Murty, P.J. Flynn, Data clustering: a review. ACM Comput. Surv. **31**(3), 264–323 (1999)
H.-P Kriegel, P. Kroeger, A. Zimek, Clustering high-dimensional data: a survey on subspace clustering, pattern-based clustering and correlation clustering. ACM Trans. Knowl. Discov. Data **3**(1), 1–58 (2009)
K. Krishna, M.N. Murty, Genetic k-means algorithm. IEEE Trans. Syst. Man Cybern., Part B, Cybern. **29**(3), 433–439 (1999)
J. MacQueen, Some methods for classification and analysis of multivariate observations, in *Proceedings of the Fifth Berkeley Symposium* (1967)
M.N. Murty, Clustering large data sets, in *Soft Computing Approach to Pattern Recognition and Image Processing*, ed. by A. Ghosh, S.K. Pal (World-Scientific, Singapore, 2002), pp. 41–63
M.N. Murty, G. Krishna, A computationally efficient technique for data-clustering. Pattern Recognit. **12**(3), 153–158 (1980)
R.T. Ng, J. Han, Efficient and effective clustering methods for spatial data mining, in *Proc. of the VLDB Conference* (1994)
A. Pavlo, E. Paulson, A. Rasin, D.J. Abadi, D.J. Dewit, S. Madden, M. Stonebraker, A comparison of approaches to large-scale data analysis, in *Proceedings of ACM SIGMOD* (2009)
A.K. Pujari, *Data Mining Techniques* (Universities Press, Hyderabad, 2001)

M. Radovanović, A. Nanopoulos, M. Ivanović, Nearest neighbors in high-dimensional data: the emergence and influence of hubs, in *Proceedings of ICML* (2009)
T. Ravindra Babu, M.N. Murty, Comparison of genetic algorithm based prototype selection schemes. Pattern Recognit. **34**(2), 523–525 (2001)
T. Ravindra Babu, M.N. Murty, V.K. Agrawal, Classification of run-length encoded binary data. Pattern Recognit. **40**(1), 321–323 (2007)
P. Russom, Big data analytics. TDWI Research Report (2011)
S.Z. Selim, M.A. Ismail, K-means-type algorithms: a generalized convergence theorem and characterization of local optimality. IEEE Trans. Pattern Anal. Mach. Intell. **6**(1), 81–87 (1984)
P. Sneath, The applications of computers to taxonomy. J. Gen. Microbiol. **17**(2), 201–226 (1957)
H. Spath, *Cluster Analysis Algorithms for Data Reduction and Classification* (Ellis Horwood, Chichester, 1980)
X. Sun, Y. Liu, M. Xu, H. Chen, J. Han, K. Wang, Feature selection using dynamic weights for classification. Knowl.-Based Syst. **37**, 541–549 (2013)
V.N. Vapnik, *Statistical Learning Theory* (Wiley, New York, 1998)
P.A. Vijaya, M.N. Murty, D.K. Subramanian, Leaders–subleaders: an efficient hierarchical clustering algorithm for large data sets. Pattern Recognit. Lett. **25**(4), 505–513 (2005)
P. Viswanath, M.N. Murty, S. Bhatnagar, Fusion of multiple approximate nearest neighbor classifiers for fast and efficient classification. Inf. Fusion **5**(4), 239–250 (2004)
D. Xin, J. Han, X. Yan, H. Cheng, Mining compressed frequent-pattern sets, in *Proceedings of VLDB Conference* (2005)
R. Xu, D.C. Wunsch II, *Clustering* (IEEE Press/Wiley, Los Alamitos/New York, 2009)
X. Yin, J. Han, CPAR: classification based on predictive association rules, in *Proceedings of SDM* (2003)
Z. Yin, L. Cao, Q. Gu, J. Han, Latent community topic analysis: integration of community discovery with topic modeling. ACM Trans. Intell. Syst. Technol. **3**(4), 63:1–63:23 (2012).
H. Yu, J. Yang, J. Han, Classifying large data sets using SVM with hierarchical clusters, in *Proc. of ACM SIGKDD (KDD)* (2003)
T. Zhang, Data clustering for very large datasets plus applications. Ph.D. Thesis, University of Wisconsin–Madison (1997)

http://www.springer.com/978-1-4471-5606-2

Compression Schemes for Mining Large Datasets
A Machine Learning Perspective
Ravindra Babu, T.; Murty, M.N.; Subrahmanya, S.V.
2013, XVI, 197 p. 62 illus., 3 illus. in color., Hardcover
ISBN: 978-1-4471-5606-2