4 Unsupervised Learning

Supervised learning discovers patterns in the data that relate data attributes to a class attribute. These patterns are then utilized to predict the values of the class attribute of future data instances. These classes indicate some real-world predictive or classification tasks such as determining whether a news article belongs to the category of sports or politics, or whether a patient has a particular disease. However, in some other applications, the data have no class attributes. The user wants to explore the data to find some intrinsic structures in them. Clustering is one technology for finding such structures. It organizes data instances into **similarity groups**, called **clusters** such that the data instances in the same cluster are similar to each other and data instances in different clusters are very different from each other. Clustering is often called **unsupervised learning**, because unlike supervised learning, class values denoting an *a priori* partition or grouping of the data are not given. Note that according to this definition, we can also say that association rule mining is an unsupervised learning task. However, due to historical reasons, clustering is closely associated and even synonymous with unsupervised learning while association rule mining is not. We follow this convention, and describe some main clustering techniques in this chapter.

Clustering has been shown to be one of the most commonly used data analysis techniques. It also has a long history, and has been used in almost every field, e.g., medicine, psychology, botany, sociology, biology, archeology, marketing, insurance, library science, etc. In recent years, due to the rapid increase of online documents and the expansion of the Web, text document clustering too has become a very important task. In Chap. 12, we will also see that clustering is very useful in Web usage mining.

4.1 Basic Concepts

Clustering is the process of organizing data instances into groups whose members are similar in some way. A **cluster** is therefore a collection of data instances which are "similar" to each other and are "dissimilar" to data instances in other clusters. In the clustering literature, a data instance is also called an **object** as the instance may represent an object in the real-world. It is also called a **data point** as it can be seen as a point in an *r*-dimension space, where *r* is the number of attributes in the data.

Fig. 4.1 shows a 2-dimensional data set. We can clearly see three groups of data points. Each group is a cluster. The task of clustering is to find the three clusters hidden in the data. Although it is easy for a human to visually detect clusters in a 2-dimensional or even 3-demensional space, it becomes very hard, if not impossible, to detect clusters visually as the number of dimensions increases. Additionally, in many applications, clusters are not as clear-cut or well separated as the three clusters in Fig. 4.1. Automatic techniques are thus needed for clustering.

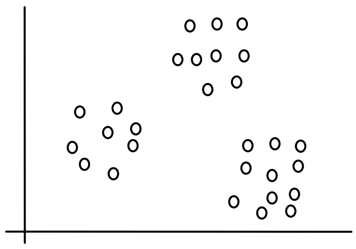


Fig. 4.1. Three natural groups or clusters of data points

After seeing the example in Fig. 4.1, you may ask the question: What is clustering for? To answer it, let us see some application examples from different domains.

Example 1: A company wants to conduct a marketing campaign to promote its products. The most effective strategy is to design a set of personalized marketing materials for each individual customer according to his/her profile and financial situation. However, this is too expensive for a large number of customers. At the other extreme, the company designs only one set of marketing materials to be used for all customers. This one-size-fits-all approach, however, may not be effective. The most cost-effective approach is to segment the customers into a small number of groups according to their similarities and design some targeted marketing materials for each group. This segmentation task is commonly done using clustering algorithms, which **partition** customers into similarity groups. In marketing research, clustering is often called **segmentation**. ■

Example 2: A company wants to produce and sell T-shirts. Similar to the case above, on one extreme, for each customer it can measure his/her size and have a T-shirt tailor-made for him/her. Obviously, this T-shirt is going to be expensive. On the other extreme, only one size of T-shirts is made.

Since this size may not fit most people, the company might not be able to sell as many T-shirts. Again, the most cost effective way is to group people based on their sizes and make a different generalized size of T-shirts for each group. This is why we see small, medium and large size T-shirts in shopping malls, and seldom see T-shirts with only a single size. The method used to group people according to their sizes is clustering. The process is usually as follows: The T-shirt manufacturer first samples a large number of people and measure their sizes to produce a measurement database. It then clusters the data, which **partitions** the data into some similarity subsets, i.e., clusters. For each cluster, it computes the average of the sizes and then uses the average to mass-produce T-shirts for all people of similar size. ■

Example 3: Everyday, news agencies around the world generate a large number of news articles. If a Web site wants to collect these news articles to provide an integrated news service, it has to organize the collected articles according to some topic hierarchy. The question is: What should the topics be, and how should they be organized? One possibility is to employ a group of human editors to do the job. However, the manual organization is costly and very time consuming, which makes it unsuitable for news and other time sensitive information. Throwing all the news articles to the readers with no organization is clearly not an option. Although classification is able to classify news articles according to predefined topics, it is not applicable here because classification needs training data, which have to be manually labeled with topic classes. Since news topics change constantly and rapidly, the training data would need to change constantly as well, which is infeasible via manual labeling. Clustering is clearly a solution for this problem because it automatically groups a stream of news articles based on their content similarities. **Hierarchical clustering algorithms** can also organize documents hierarchically, i.e., each topic may contain sub-topics and so on. Topic hierarchies are particularly useful for texts. ■

The above three examples indicate two types of clustering, **partitional** and **hierarchical**. Indeed, these are the two most important types of clustering approaches. We will study some specific algorithms of these two types of clustering.

Our discussion and examples above also indicate that clustering needs a similarity function to measure how similar two data points (or objects) are, or alternatively a **distance function** to measure the distance between two data points. We will use distance functions in this chapter. The goal of clustering is thus to discover the intrinsic grouping of the input data through the use of a clustering algorithm and a distance function.

```
   Algorithm k-means(k, D)
1  choose k data points as the initial centroids (cluster centers)
2  repeat
3      for each data point x ∈ D do
4          compute the distance from x to each centroid;
5          assign x to the closest centroid        // a centroid represents a cluster
6      endfor
7      re-compute the centroid using the current cluster memberships
8  until the stopping criterion is met
```

Fig. 4.2. The *k*-means algorithm

4.2 K-means Clustering

The *k*-means algorithm is the best known **partitional clustering algorithm.** It is perhaps also the most widely used among all clustering algorithms due to its simplicity and efficiency. Given a set of data points and the required number of *k* clusters (*k* is specified by the user), this algorithm iteratively partitions the data into *k* clusters based on a distance function.

4.2.1 K-means Algorithm

Let the set of data points (or instances) *D* be

$$\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\},$$

where $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{ir})$ is a vector in a real-valued space $X \subseteq \Re^r$, and r is the number of attributes in the data (or the number of dimensions of the **data space**). The *k*-means algorithm partitions the given data into *k* clusters. Each cluster has a cluster **center**, which is also called the cluster **centroid**. The centroid, usually used to represent the cluster, is simply the mean of all the data points in the cluster, which gives the name to the algorithm, i.e., since there are *k* clusters, thus *k* means. Figure 4.2 gives the *k*-means clustering algorithm.

At the beginning, the algorithm randomly selects *k* data points as the **seed** centroids. It then computes the distance between each seed centroid and every data point. Each data point is assigned to the centroid that is closest to it. A centroid and its data points therefore represent a cluster. Once all the data points in the data are assigned, the centroid for each cluster is re-computed using the data points in the current cluster. This process repeats until a stopping criterion is met. The stopping (or convergence) criterion can be any one of the following:

1. no (or minimum) re-assignments of data points to different clusters.
2. no (or minimum) change of centroids.
3. minimum decrease in the **sum of squared error** (SSE),

$$SSE = \sum_{j=1}^{k} \sum_{\mathbf{x} \in C_j} dist(\mathbf{x}, \mathbf{m}_j)^2, \tag{1}$$

where k is the number of required clusters, C_j is the jth cluster, $\mathbf{m}_j$ is the centroid of cluster C_j (the mean vector of all the data points in C_j), and $dist(\mathbf{x}, \mathbf{m}_j)$ is the distance between data point $\mathbf{x}$ and centroid $\mathbf{m}_j$.

The *k*-means algorithm can be used for any application data set where the **mean** can be defined and computed. In **Euclidean space**, the mean of a cluster is computed with:

$$\mathbf{m}_j = \frac{1}{|C_j|} \sum_{\mathbf{x}_i \in C_j} \mathbf{x}_i, \tag{2}$$

where $|C_j|$ is the number of data points in cluster C_j. The distance from a data point $\mathbf{x}_i$ to a cluster mean (centroid) $\mathbf{m}_j$ is computed with

$$\begin{aligned} dist(\mathbf{x}_i, \mathbf{m}_j) &= \| \mathbf{x}_i - \mathbf{m}_j \| \\ &= \sqrt{(x_{i1} - m_{j1})^2 + (x_{i2} - m_{j2})^2 + \ldots + (x_{ir} - m_{jr})^2}. \end{aligned} \tag{3}$$

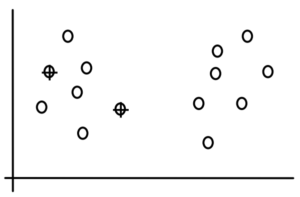
Example 4: Figure 4.3(A) shows a set of data points in a 2-dimensional space. We want to find 2 clusters from the data, i.e., $k = 2$. First, two data points (each marked with a cross) are randomly selected to be the initial centroids (or seeds) shown in Fig. 4.3(A). The algorithm then goes to the first iteration (the repeat-loop).

Iteration 1: Each data point is assigned to its closest centroid to form 2 clusters. The resulting clusters are given in Fig. 4.3(B). Then the centroids are re-computed based on the data points in the current clusters (Fig. 4.3(C)). This leads to iteration 2.

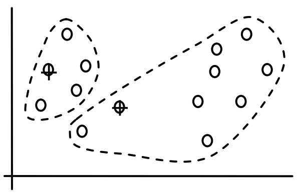
Iteration 2: Again, each data point is assigned to its closest new centroid to form two new clusters shown in Fig. 4.3(D). The centroids are then re-computed. The new centroids are shown in Fig. 4.3(E).

Iteration 3: The same operations are performed as in the first two iterations. Since there is no re-assignment of data points to different clusters in this iteration, the algorithm ends.

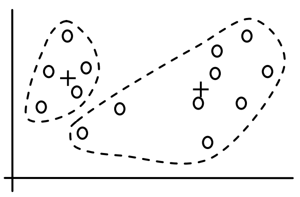
The final clusters are those given in Fig. 4.3(G). The set of data points in each cluster and its centroid are output to the user.



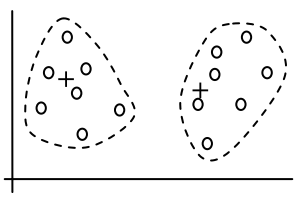
(A). Random selection of *k* seeds (or centroids)



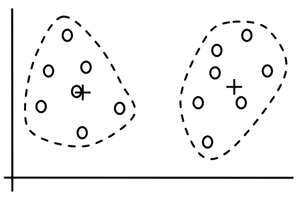
Iteration 1: (B). Cluster assignment



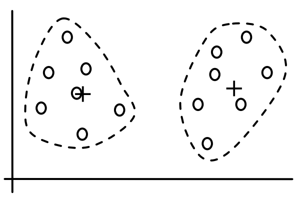
(C). Re-compute centroids



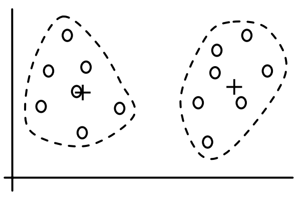
Iteration 2: (D). Cluster assignment



(E). Re-compute centroids



Iteration 3: (F). Cluster assignment



(G). Re-compute centroids

Fig. 4.3. The working of the *k*-means algorithm through an example ■

One problem with the *k*-means algorithm is that some clusters may become empty during the clustering process since no data point is assigned to them. Such clusters are called **empty clusters**. To deal with an empty cluster, we can choose a data point as the replacement centroid, e.g., a data point that is furthest from the centroid of a large cluster. If the sum of the squared error (SSE) is used as the stopping criterion, the cluster with the largest squared error may be used to find another centroid.

4.2.2 Disk Version of the K-means Algorithm

The *k*-means algorithm may be implemented in such a way that it does not need to load the entire data set into the main memory, which is useful for large data sets. Notice that the centroids for the *k* clusters can be computed incrementally in each iteration because the summation in Equation (2) can be calculated separately first. During the clustering process, the number of data points in each cluster can be counted incrementally as well. This gives us a disk based implementation of the algorithm (Fig. 4.4), which produces exactly the same clusters as that in Fig. 4.2, but with the data on disk. In each for-loop, the algorithm simply scans the data once.

The whole clustering process thus scans the data *t* times, where *t* is the number of iterations before convergence, which is usually not very large (< 50). In applications, it is quite common to set a limit on the number of iterations because later iterations typically result in only minor changes to the clusters. Thus, this algorithm may be used to cluster large data sets which cannot be loaded into the main memory. Although there are several special algorithms that scale-up clustering algorithms to large data sets, they all require sophisticated techniques.

```
   Algorithm disk-$k$-means($k$, $D$)
1  Choose $k$ data points as the initial centroids $\mathbf{m}_j, j = 1, \ldots, k$;
2  repeat
3      initialize $\mathbf{s}_j \leftarrow \mathbf{0}, j = 1, \ldots, k$;        // $\mathbf{0}$ is a vector with all 0's
4      initialize $n_j \leftarrow 0, j = 1, \ldots, k$;        // $n_j$ is the number of points in cluster $j$
5      for each data point $\mathbf{x} \in D$ do
6          $j \leftarrow \arg\min_{i \in \{1,2,\ldots k\}} dist(\mathbf{x}, \mathbf{m}_i)$;
7          assign $\mathbf{x}$ to the cluster $j$;
8          $\mathbf{s}_j \leftarrow \mathbf{s}_j + \mathbf{x}$;
9          $n_j \leftarrow n_j + 1$;
10     endfor
11     $\mathbf{m}_j \leftarrow \mathbf{s}_j / n_j, j = 1, \ldots, k$;
12 until the stopping criterion is met
```

Fig. 4.4. A simple disk version of the *k*-means algorithm

Let us give some explanations of this algorithm. Line 1 does exactly the same thing as the algorithm in Fig. 4.2. Line 3 initializes vector $\mathbf{s}_j$ which is used to incrementally compute the sum in Equation (2) (line 8). Line 4 initializes n_j which records the number of data points assigned to cluster j (line 9). Lines 6 and 7 perform exactly the same tasks as lines 4 and 5 in the original algorithm in Fig. 4.2. Line 11 re-computes the centroids, which are used in the next iteration. Any of the three stopping criteria may be used here. If the sum of squared error is applied, we can modify the algorithm slightly to compute the sum of square error incrementally.

4.2.3 Strengths and Weaknesses

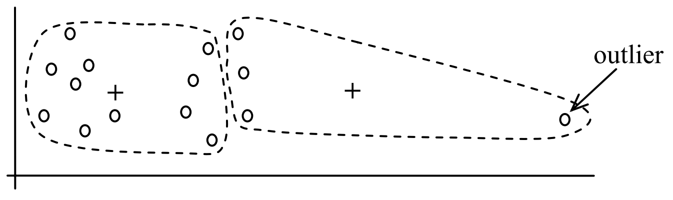
The main strengths of the *k*-means algorithm are its simplicity and efficiency. It is easy to understand and easy to implement. Its time complexity is $O(tkn)$, where n is the number of data points, k is the number of clusters, and t is the number of iterations. Since both k and t are normally much smaller than n. The *k*-means algorithm is considered a linear algorithm in the number of data points.

The weaknesses and ways to address them are as follows:

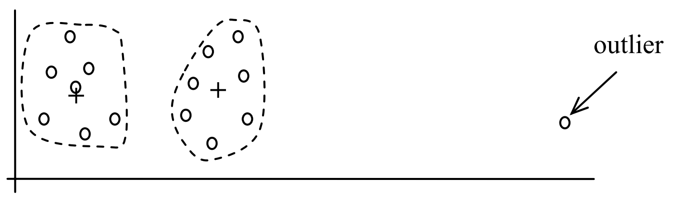
1. The algorithm is only applicable to data sets where the notion of the **mean** is defined. Thus, it is difficult to apply to categorical data sets. There is, however, a variation of the *k*-means algorithm called ***k*-modes**, which clusters categorical data. The algorithm uses the mode instead of the mean as the centroid. Assuming that the data instances are described by r categorical attributes, the mode of a cluster C_j is a tuple $\mathbf{m}_j = (m_{j1}, m_{j2}, \ldots, m_{jr})$ where m_{ji} is the most frequent value of the ith attribute of the data instances in cluster C_j. The similarity (or distance) between a data instance and a mode is the number of values that they match (or do not match).
2. The user needs to specify the number of clusters k in advance. In practice, several k values are tried and the one that gives the most desirable result is selected. We will discuss the evaluation of clusters later.
3. The algorithm is sensitive to **outliers**. Outliers are data points that are very far away from other data points. Outliers could be errors in the data recording or some special data points with very different values. For example, in an employee data set, the salary of the Chief-Executive-Officer (CEO) of the company may be considered as an outlier because its value could be many times larger than everyone else. Since the *k*-means algorithm uses the mean as the centroid of each cluster, outliers may result in undesirable clusters as the following example shows.

   Example 5: In Fig. 4.5(A), due to an outlier data point, the two resulting clusters do not reflect the natural groupings in the data. The ideal clusters are shown in Fig. 4.5(B). The outlier should be identified and reported to the user. ■

   There are several methods for dealing with outliers. One simple method is to remove some data points in the clustering process that are much further away from the centroids than other data points. To be safe, we may want to monitor these possible outliers over a few iterations and then decide whether to remove them. It is possible that a very small cluster of data points may be outliers. Usually, a threshold value is used to make the decision.



(A): Undesirable clusters



(B): Ideal clusters

Fig. 4.5. Clustering with and without the effect of outliers ■

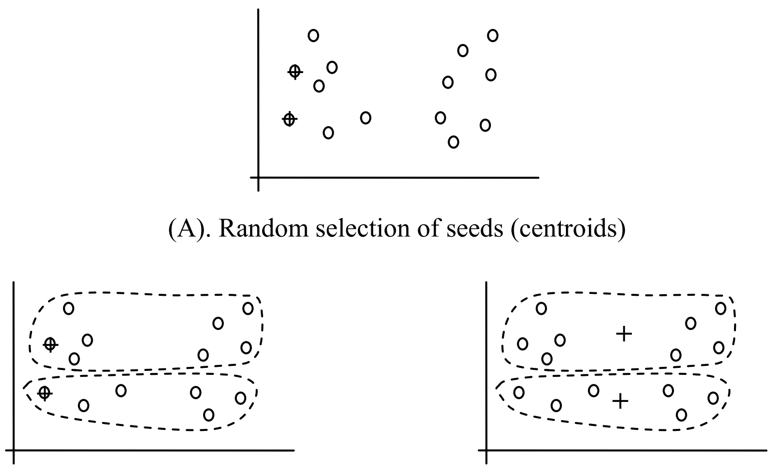
Another method is to perform random sampling. Since in sampling we only choose a small subset of the data points, the chance of selecting an outlier is very small. We can use the sample to do a pre-clustering and then assign the rest of the data points to these clusters, which may be done in any of the three ways below:

- Assign each remaining data point to the centroid closest to it. This is the simplest method.
- Use the clusters produced from the sample to perform supervised learning (classification). Each cluster is regarded as a class. The clustered sample is thus treated as the training data for learning. The resulting classifier is then applied to classify the remaining data points into appropriate classes or clusters.
- Use the clusters produced from the sample as seeds to perform **semi-supervised learning**. Semi-supervised learning is a new learning model that learns from a small set of labeled examples (with classes) and a large set of unlabeled examples (without classes). In our case, the clustered sample data are used as the labeled set and the remaining data points are used as the unlabeled set. The results of the learn-

ing naturally cluster all the remaining data points. We will study this technique in the next chapter.

4. The algorithm is sensitive to **initial seeds**, which are the initially selected centroids. Different initial seeds may result in different clusters. Thus, if the sum of squared error is used as the stopping criterion, the algorithm only achieves **local optimal**. The global optimal is computationally infeasible for large data sets.

Example 6: Figure 4.6 shows the clustering process of a 2-dimensional data set. The goal is to find two clusters. The randomly selected initial seeds are marked with crosses in Fig. 4.6(A). Figure 4.6(B) gives the clustering result of the first iteration. Figure 4.6(C) gives the result of the second iteration. Since there is no re-assignment of data points, the algorithm stops.



(A). Random selection of seeds (centroids)

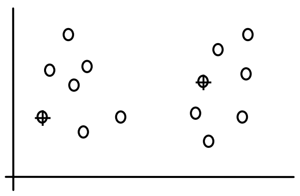
(B). Iteration 1

(C). Iteration 2

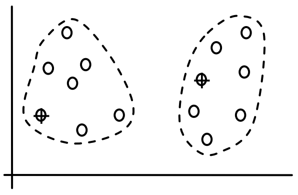
Fig. 4.6. Poor initial seeds (centroids)

If the initial seeds are different, we may obtain entirely different clusters as Fig. 4.7 shows. Figure 4.7 uses the same data as Fig. 4.6, but different initial seeds (Fig. 4.7(A)). After two iterations, the algorithm ends, and the final clusters are given in Fig. 4.7(C). These two clusters are more reasonable than the two clusters in Fig. 4.6(C), which indicates that the choice of the initial seeds in Fig. 4.6(A) is poor.

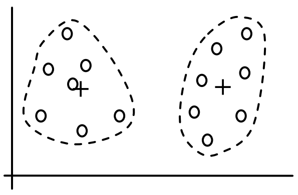
To select good initial seeds, researchers have proposed several methods. One simple method is to first compute the mean **m** (the centroid) of the entire data set (any random data point rather than the mean can be used as well). Then the first seed data point $\mathbf{x}_1$ is selected to be the furthest from the mean $\mathbf{m}$. The second data point $\mathbf{x}_2$ is selected to be the furthest from $\mathbf{x}_1$. Each subsequent data point $\mathbf{x}_i$ is selected such that the sum of distances from $\mathbf{x}_i$ to those already selected data points is the largest. However, if the data has outliers, the method will not work well. To deal with outliers, again, we can randomly select a small sample of the data and perform the same operation on the sample. As we discussed above, since the number of outliers is small, the chance that they show up in the sample is very small.



(A). Random selection of *k* seeds (centroids)



(B). Iteration 1



(C). Iteration 2

Fig. 4.7. Good initial seeds (centroids) ■

Another method is to sample the data and use the sample to perform hierarchical clustering, which we will discuss in Sect. 4.4. The centroids of the resulting *k* clusters are used as the initial seeds.

Yet another approach is to manually select seeds. This may not be a difficult task for text clustering applications because it is easy for human users to read some documents and pick some good seeds. These seeds may help improve the clustering result significantly and also enable the system to produce clusters that meet the user's needs.

5. The *k*-means algorithm is not suitable for discovering clusters that are not hyper-ellipsoids (or hyper-spheres).

Example 7: Figure 4.8(A) shows a 2-dimensional data set. There are two irregular shaped clusters. However, the two clusters are not hyper-ellipsoids, which means that the *k*-means algorithm will not be able to find them. Instead, it may find the two clusters shown in Fig. 4.8(B).

The question is: are the two clusters in Fig. 4.8(B) necessarily bad? The answer is no. It depends on the application. It is not true that a clustering algorithm that is able to find arbitrarily shaped clusters is always better. We will discuss this issue in Sect. 4.3.2.

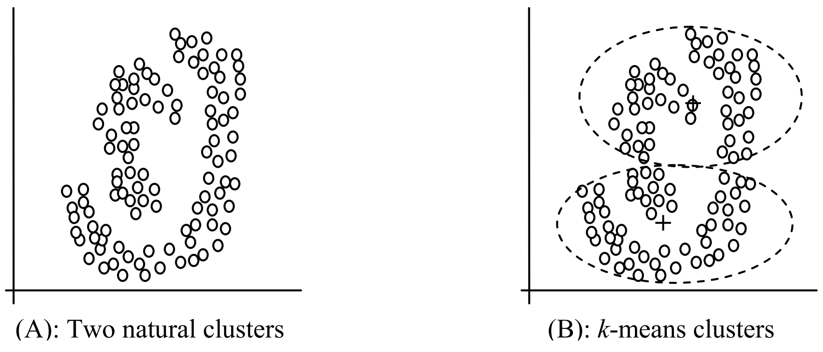


Fig. 4.8. Natural (but irregular) clusters and *k*-means clusters ■

Despite these weaknesses, *k*-means is still the most popular algorithm in practice due to its simplicity, efficiency and the fact that other clustering algorithms have their own lists of weaknesses. There is no clear evidence showing that any other clustering algorithm performs better than the *k*-means algorithm in general, although it may be more suitable for some specific types of data or applications than *k*-means. Note also that comparing different clustering algorithms is a very difficult task because unlike supervised learning, nobody knows what the correct clusters are, especially in high dimensional spaces. Although there are several cluster evaluation methods, they all have drawbacks. We will discuss the evaluation issue in Sect. 4.9.

4.3 Representation of Clusters

Once a set of clusters is found, the next task is to find a way to represent the clusters. In some applications, outputting the set of data points that makes up the cluster to the user is sufficient. However, in other applications that involve decision making, the resulting clusters need to be represented in a compact and understandable way, which also facilitates the evaluation of the resulting clusters.

4.3.1 Common Ways of Representing Clusters

There are three main ways to represent clusters:

1. Use the centroid of each cluster to represent the cluster. This is the most popular way. The centroid tells where the center of the cluster is. One may also compute the radius and standard deviation of the cluster to determine the spread in each dimension. The centroid representation alone works well if the clusters are of the hyper-spherical shape. If clusters are elongated or are of other shapes, centroids may not be suitable.
2. Use classification models to represent clusters. In this method, we treat each cluster as a class. That is, all the data points in a cluster are regarded as having the same class label, e.g., the cluster ID. We then run a supervised learning algorithm on the data to find a classification model. For example, we may use the decision tree learning to distinguish the clusters. The resulting tree or set of rules provide an understandable representation of the clusters.

   Figure 4.9 shows a partitioning produced by a decision tree algorithm. The original clustering gave three clusters. Data points in cluster 1 are represented by 1's, data points in cluster 2 are represented by 2's, and data points in cluster 3 are represented by 3's. We can see that the three clusters are separated and each can be represented with a rule.

   $x \leq 2 \rightarrow$ cluster 1
   $x > 2, y > 1.5 \rightarrow$ cluster 2
   $x > 2, y \leq 1.5 \rightarrow$ cluster 3

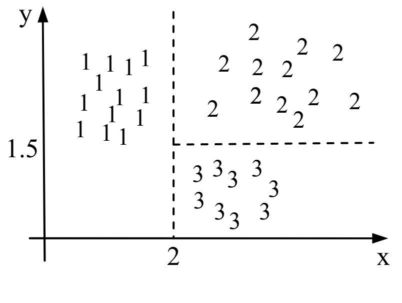


Fig. 4.9. Description of clusters using rules

We make two remarks about this representation method:

- The partitioning in Fig. 4.9 is an ideal case as each cluster is represented by a single rectangle (or rule). However, in most applications, the situation may not be so ideal. A cluster may be split into a few

hyper-rectangles or rules. However, there is usually a dominant or large rule which covers most of the data points in the cluster.
- One can use the set of rules to evaluate the clusters to see whether they conform to some existing domain knowledge or intuition.

3. Use frequent values in each cluster to represent it. This method is mainly for clustering of categorical data (e.g., in the *k*-modes clustering). It is also the key method used in text clustering, where a small set of frequent words in each cluster is selected to represent the cluster.

4.3.2 Clusters of Arbitrary Shapes

Hyper-elliptical and hyper-spherical clusters are usually easy to represent, using their centroids together with spreads (e.g., standard deviations), rules, or a combination of both. However, other arbitrary shaped clusters, like the natural clusters shown in Fig. 4.8(A), are hard to represent especially in high dimensional spaces.

A common criticism about an algorithm like *k*-means is that it is not able to find arbitrarily shaped clusters. However, this criticism may not be as bad as it sounds because whether one type of clustering is desirable or not depends on the application. Let us use the natural clusters in Fig. 4.8(A) to discuss this issue together with an artificial application.

Example 8: Assume that the data shown in Fig. 4.8(A) is the measurement data of people's physical sizes. We want to group people based on their sizes into only two groups in order to mass-produce T-shirts of only 2 sizes (say large and small). Even if the measurement data indicate two natural clusters as in Fig. 4.8(A), it is difficult to use the clusters because we need centroids of the clusters to design T-shirts. The clusters in Fig. 4.8(B) are in fact better because they provide us the centroids that are representative of the surrounding data points. If we use the centroids of the two natural clusters as shown in Fig. 4.10 to make T-shirts, it is clearly inappropriate because they are too near to each other in this case. In general, it does not make sense to define the concept of center or centroid for an irregularly shaped cluster. ■

Note that clusters of arbitrary shapes can be found by neighborhood search algorithms such as some hierarchical clustering methods (see the next section), and density-based clustering methods [164]. Due to the difficulty of representing an arbitrarily shaped cluster, an algorithm that finds such clusters may only output a list of data points in each cluster, which are not as easy to use. These kinds of clusters are more useful in spatial and image processing applications, but less useful in others.

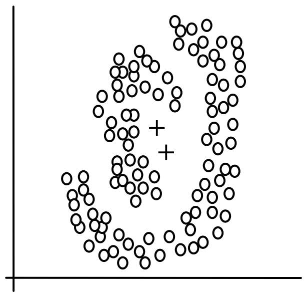


Fig. 4.10. Two natural clusters and their centroids

4.4 Hierarchical Clustering

Hierarchical clustering is another major clustering approach. It has a number of desirable properties which make it popular. It clusters by producing a nested sequence of clusters like a **tree** (also called a **dendrogram**). Singleton clusters (individual data points) are at the bottom of the tree and one root cluster is at the top, which covers all data points. Each internal cluster node contains child cluster nodes. Sibling clusters partition the data points covered by their common parent. Figure 4.11 shows an example.

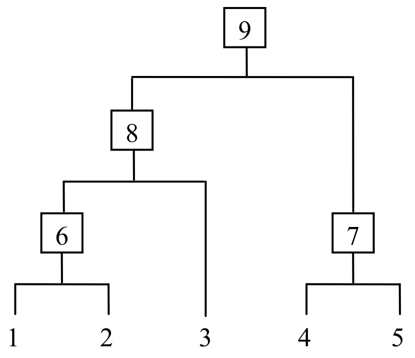


Fig. 4.11. An illustration of hierarchical clustering

At the bottom of the tree, there are 5 clusters (5 data points). At the next level, cluster 6 contains data points 1 and 2, and cluster 7 contains data points 4 and 5. As we move up the tree, we have fewer and fewer clusters. Since the whole clustering tree is stored, the user can choose to view clusters at any level of the tree.

There are two main types of hierarchical clustering methods:

Agglomerative (**bottom up**) **clustering:** It builds the dendrogram (tree) from the bottom level, and merges the most similar (or nearest) pair of clusters at each level to go one level up. The process continues until all the data points are merged into a single cluster (i.e., the root cluster).

Divisive (**top down**) **clustering:** It starts with all data points in one cluster, the root. It then splits the root into a set of child clusters. Each child cluster is recursively divided further until only singleton clusters of individual data points remain, i.e., each cluster with only a single point.

Agglomerative methods are much more popular than divisive methods. We will focus on agglomerative hierarchical clustering. The general agglomerative algorithm is given in Fig. 4.12.

```
   Algorithm Agglomerative($D$)
1  Make each data point in the data set $D$ a cluster,
2  Compute all pair-wise distances of $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n \in D$;
2  repeat
3      find two clusters that are nearest to each other;
4      merge the two clusters form a new cluster $c$;
5      compute the distance from $c$ to all other clusters;
12 until there is only one cluster left
```

Fig. 4.12. The agglomerative hierarchical clustering algorithm

Example 9: Figure 4.13 illustrates the working of the algorithm. The data points are in a 2-dimensional space. Figure 4.13(A) shows the sequence of nested clusters, and Fig. 4.13(B) gives the dendrogram. ■

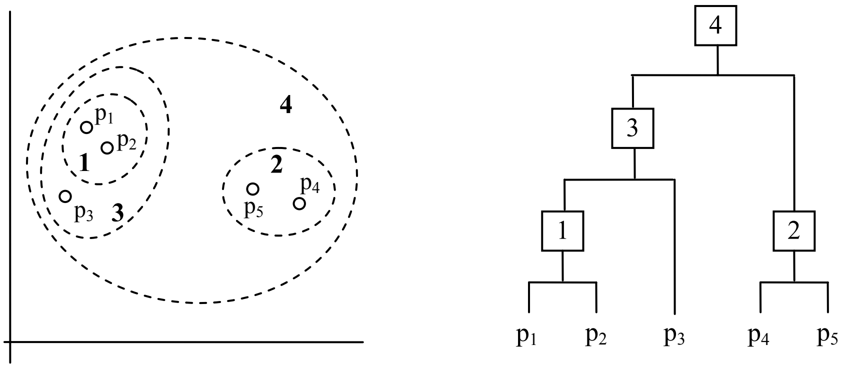


Fig. 4.13. The working of an agglomerative hierarchical clustering algorithm

Unlike the *k*-means algorithm, which uses only the centroids in distance computation, hierarchical clustering may use anyone of several methods to determine the distance between two clusters. We introduce them next.

4.4.1 Single-Link Method

In **single-link** (or **single linkage**) hierarchical clustering, the distance between two clusters is the distance between two closest data points in the two clusters (one data point from each cluster). In other words, the single-link clustering merges the two clusters in each step whose two nearest data points (or members) have the smallest distance, i.e., the two clusters with the **smallest minimum** pair-wise distance. The single-link method is suitable for finding non-elliptical shape clusters. However, it can be sensitive to noise in the data, which may cause the **chain effect** and produce straggly clusters. Figure 4.14 illustrates this situation. The noisy data points (represented with filled circles) in the middle connect two natural clusters and split one of them.

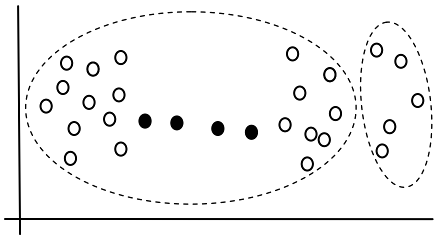


Fig. 4.14. The chain effect of the single-link method

With suitable data structures, single-link hierarchical clustering can be done in $O(n^2)$ time, where n is the number of data points. This is much slower than the *k*-means method, which performs clustering in linear time.

4.4.2 Complete-Link Method

In **complete-link** (or **complete linkage**) clustering, the distance between two clusters is the **maximum** of all pair-wise distances between the data points in the two clusters. In other words, the complete-link clustering merges the two clusters in each step whose two furthest data points have the smallest distance, i.e., the two clusters with the **smallest maximum** pair-wise distance. Figure 4.15 shows the clusters produced by complete-link clustering using the same data as in Fig. 4.14.

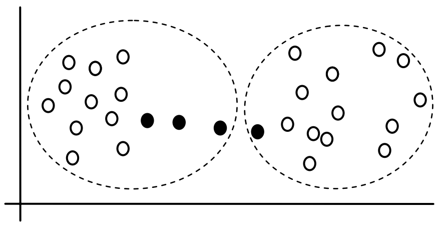


Fig. 4.15. Clustering using the complete-link method

Although the complete-link method does not have the problem of chain effects, it can be sensitive to outliers. Despite this limitation, it has been observed that the complete-link method usually produces better clusters than the single-link method. The worse case time complexity of the complete-link clustering is $O(n^2\log n)$, where n is the number of data points.

4.4.3 Average-Link Method

This is a compromise between the sensitivity of complete-link clustering to outliers and the tendency of single-link clustering to form long chains that do not correspond to the intuitive notion of clusters as compact, spherical objects. In this method, the distance between two clusters is the average distance of all pair-wise distances between the data points in two clusters. The time complexity of this method is also $O(n^2\log n)$.

Apart from the above three popular methods, there are several others. The following two methods are also commonly used:

Centroid method: In this method, the distance between two clusters is the distance between their centroids.

Ward's method: In this method, the distance between two clusters is defined as the increase in the sum of squared error (distances) from that of two clusters to that of one merged cluster. Thus, the clusters to be merged in the next step are the ones that will increase the sum the least. Recall that the sum of squared error (SSE) is one of the measures used in the *k*-means clustering (Equation (1)).

4.4.4. Strengths and Weaknesses

Hierarchical clustering has several advantages compared to the *k*-means and other partitioning clustering methods. It is able to take any form of distance or similarity function. Moreover, unlike the *k*-means algorithm which only gives *k* clusters at the end, the hierarchy of clusters from hierarchical clustering enables the user to explore clusters at any level of detail (or granularity). In many applications, this resulting hierarchy can be very useful in its own right. For example, in text document clustering, the cluster hierarchy may represent a topic hierarchy in the documents.

Some studies have shown that agglomerative hierarchical clustering often produces better clusters than the *k*-means method. It can also find clusters of arbitrary shapes, e.g., using the single-link method.

Hierarchical clustering also has several weaknesses. As we discussed with the individual methods, the single-link method may suffer from the chain effect, and the complete-link method is sensitive to outliers. The main shortcomings of all hierarchical clustering methods are their computation complexities and space requirements, which are at least quadratic. Compared to the *k*-means algorithm, this is very inefficient and not practical for large data sets. One can use sampling to deal with the problems. A small sample is taken to do clustering and then the rest of the data points are assigned to each cluster either by distance comparison or by supervised learning (see Sect. 4.3.1). Some **scale-up methods** may also be applied to large data sets. The main idea of the scale-up methods is to find many small clusters first using an efficient algorithm, and then to use the centroids of these small clusters to represent the clusters to perform the final hierarchical clustering (see the BIRCH method in [610]).

4.5 Distance Functions

Distance or similarity functions play a central role in all clustering algorithms. Numerous distance functions have been reported in the literature and used in applications. Different distance functions are also used for different types of attributes (also called **variables**).

4.5.1 Numeric Attributes

The most commonly used distance functions for numeric attributes are the **Euclidean distance** and **Manhattan** (**city block**) **distance**. Both distance measures are special cases of a more general distance function called the **Minkowski distance**. We use $dist(\mathbf{x}_i, \mathbf{x}_j)$ to denote the distance between two data points of *r* dimensions. The Minkowski distance is:

$$dist(\mathbf{x}_i, \mathbf{x}_j) = (|x_{i1} - x_{j1}|^h + |x_{i2} - x_{j2}|^h + ... + |x_{ir} - x_{jr}|^h)^{\frac{1}{h}}, \qquad (4)$$

where *h* is a positive integer.

If $h = 2$, it is the **Euclidean distance**,

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \ldots + (x_{ir} - x_{jr})^2}. \tag{5}$$

If $h = 1$, it is the **Manhattan distance**,

$$dist(\mathbf{x}_i, \mathbf{x}_j) = | x_{i1} - x_{j1} | + | x_{i2} - x_{j2} | + \ldots + | x_{ir} - x_{jr} |. \tag{6}$$

Other common distance functions include:

Weighted Euclidean distance: A weight is associated with each attribute to express its importance in relation to other attributes.

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{w_1(x_{i1} - x_{j1})^2 + w_2(x_{i2} - x_{j2})^2 + \ldots + w_r(x_{ir} - x_{jr})^2}. \tag{7}$$

Squared Euclidean distance: the standard Euclidean distance is squared in order to place progressively greater weights on data points that are further apart. The distance is

$$dist(\mathbf{x}_i, \mathbf{x}_j) = (x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \ldots + (x_{ir} - x_{jr})^2. \tag{8}$$

Chebychev distance: This distance measure is appropriate in cases where one wants to define two data points as "different" if they are different on any one of the attributes. The Chebychev distance is

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \max(| x_{i1} - x_{j1} |, | x_{i2} - x_{j2} |, \ldots, | x_{ir} - x_{jr} |). \tag{9}$$

4.5.2 Binary and Nominal Attributes

The above distance measures are only appropriate for numeric attributes. For **binary** and **nominal** attributes (also called **unordered categorical** attributes), we need different functions. Let us discuss binary attributes first.

A **binary attribute** has two states or values, usually represented by 1 and 0. The two states have no numerical ordering. For example, Gender has two values, male and female, which have no ordering relations but are just different. Existing distance functions for binary attributes are based on the proportion of value matches in two data points. A match means that, for a particular attribute, both data points have the same value. It is easy to use a **confusion matrix** to introduce these measures. Given the *i*th and *j*th data points, $\mathbf{x}_i$ and $\mathbf{x}_j$, we can construct the confusion matrix in Fig. 4.16.

To give the distance functions, we further divide binary attributes into **symmetric** and **asymmetric** attributes. For different types of attributes, different distance functions need to be used [271]:

		Data point $\mathbf{x}_j$		
		1	0	
	1	a	b	$a+b$
Data point $\mathbf{x}_i$	0	c	d	$c+d$
		$a+c$	$b+d$	$a+b+c+d$

a: the number of attributes with the value of 1 for both data points.
b: the number of attributes for which $x_{if} = 1$ and $x_{jf} = 0$, where x_{if} (x_{jf}) is the value of the fth attribute of the data point $\mathbf{x}_i$ ($\mathbf{x}_j$).
c: the number of attributes for which $x_{if} = 0$ and $x_{jf} = 1$.
d: the number of attributes with the value of 0 for both data points.

Fig. 4.16. Confusion matrix of two data points with only binary attributes

Symmetric attributes: A binary attribute is **symmetric** if both of its states (0 and 1) have equal importance, and carry the same weight, e.g., male and female of the attribute Gender. The most commonly used distance function for symmetric attributes is the **simple matching distance**, which is the proportion of mismatches (Equation (10)) of their values. We assume that every attribute in the data set is a symmetric attribute.

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \frac{b+c}{a+b+c+d} \tag{10}$$

We can also weight some components in Equation (10) according to application needs. For example, we may want mismatches to carry twice the weight of matches, or vice versa:

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \frac{2(b+c)}{a+d+2(b+c)} \tag{11}$$

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \frac{b+c}{2(a+d)+b+c} \tag{12}$$

Example 10: Given the following two data points, where each attribute is a symmetric binary attribute,

$\mathbf{x}_1$	1	1	1	0	1	0	0
$\mathbf{x}_2$	0	1	1	0	0	1	0

the distance computed based on the simple matching distance is

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \frac{2+1}{2+2+1+2} = \frac{3}{7} = 0.429. \tag{13}$$

■

Asymmetric attributes: A binary attribute is asymmetric if one of the states is more important or valuable than the other. By convention, we use state 1 to represent the more important state, which is typically the rare or infrequent state. The most commonly used distance measure for asymmetric attributes is the **Jaccard distance**:

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \frac{b+c}{a+b+c}. \tag{14}$$

Similarly, we can vary the Jaccard distance by giving more weight to (b+c) or more weight to a to express different emphases.

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \frac{2(b+c)}{a+2(b+c)}. \tag{15}$$

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \frac{b+c}{2a+b+c}. \tag{16}$$

Note that there is also a **Jaccard coefficient**, which measures similarity (rather than distance) and is defined as a / (a+b+c).

For general **nominal attributes** with more than two states or values, the commonly used distance measure is also based on the simple matching distance. Given two data points $\mathbf{x}_i$ and $\mathbf{x}_j$, let the number of attributes be r, and the number of values that match in $\mathbf{x}_i$ and $\mathbf{x}_j$ be q:

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \frac{r-q}{r}. \tag{17}$$

As that for binary attributes, we can give higher weights to different components in Equation (17) according to different application characteristics.

4.5.3 Text Documents

Although a text document consists of a sequence of sentences and each sentence consists of a sequence of words, a document is usually considered as a "bag" of words in document clustering. The sequence and the position information of words are ignored. Thus a document can be represented as a vector just like a normal data point. However, we use similarity to compare two documents rather than distance. The most commonly used similarity function is the **cosine similarity**. We will study this similarity measure in Sect. 6.2.2 when we discuss information retrieval and Web search.

4.6 Data Standardization

One of the most important steps in data pre-processing for clustering is to standardize the data. For example, using the Euclidean distance, standardization of attributes is highly recommended so that all attributes can have equal impact on the distance computation. This is to avoid obtaining clusters that are dominated by attributes with the largest amounts of variation.

Example 11: In a 2-dimensional data set, the value range of one attribute is from 0 to 1, while the value range of the other attribute is from 0 to 1000. Consider the following pair of data points $\mathbf{x}_i$: (0.1, 20) and $\mathbf{x}_j$: (0.9, 720). The Euclidean distance between the two points is

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{(0.9-0.1)^2 + (720-20)^2} = 700.000457, \tag{18}$$

which is almost completely dominated by (720–20) = 700. To deal with the problem, we standardize the attributes, e.g., to force the attributes to have a common value range. If both attributes are forced to have a scale within the range 0–1, the values 20 and 720 become 0.02 and 0.72. The distance on the first dimension becomes 0.8 and the distance on the second dimension 0.7, which are more equitable. Then, $dist(\mathbf{x}_i, \mathbf{x}_j) = 1.063$. ■

This example shows that standardizing attributes is important. In fact, different types of attributes require different treatments. We list these treatments below.

Interval-scaled attributes: These are numeric/continuous attributes. Their values are real numbers following a linear scale. Examples of such attributes are age, height, weight, cost, etc. The idea is that intervals keep the same importance through out the scale. For example, the difference in age between 10 and 20 is the same as that between 40 and 50.

There are two main approaches to standardize interval scaled attributes, **range** and ***z*-score**. The range method divides each value by the range of valid values of the attribute so that the transformed value ranges between 0 and 1. Given the value x_{if} of the *f*th attribute of the *i*th data point, the new value $rg(x_{if})$ is,

$$rg(x_{if}) = \frac{x_{if} - \min(f)}{\max(f) - \min(f)}, \tag{19}$$

where min(*f*) and max(*f*) are the minimum value and maximum value of attribute *f* respectively. max(*f*) – min(*f*) is the value range of the valid values of attribute *f*.

The *z*-score method transforms an attribute value based on the mean and the standard deviation of the attribute. That is, the *z*-score of the value indicates how far and in what direction the value deviates from the mean of the attribute, expressed in units of the standard deviation of the attribute. The standard deviation of attribute *f*, denoted by σ_f, is computed with:

$$\sigma_f = \sqrt{\frac{\sum_{i=1}^{n}(x_{if} - \mu_f)^2}{n-1}}, \tag{20}$$

where n is the number of data points in the data set, x_{if} is the same as above, and μ_f is the mean of attribute *f*, which is computed with:

$$\mu_f = \frac{1}{n}\sum_{i=1}^{n} x_{if}. \tag{21}$$

Given the value x_{if}, its *z*-score (the new value after transformation) is $z(x_{if})$,

$$z(x_{if}) = \frac{x_{if} - \mu_f}{\sigma_f}. \tag{22}$$

Ratio-Scaled Attributes: These are also numeric attributes taking real values. However, unlike interval-scaled attributes, their scales are not linear. For example, the total amount of microorganisms that evolve in a time t is approximately given by

$$Ae^{Bt},$$

where A and B are some positive constants. This formula is referred to as exponential growth. If we have such attributes in a data set for clustering, we have one of the following two options:

1. Treat it as an interval-scaled attribute. This is often not recommended due to scale distortion.
2. Perform logarithmic transformation to each value, x_{if}, i.e.,

   $$\log(x_{if}). \tag{23}$$

   After the transformation, the attribute can be treated as an interval-scaled attribute.

Nominal (Unordered Categorical) Attributes: As we discussed in Sect. 4.5.2, the value of such an attribute can take anyone of a set of states (also called categories). The states have no logical or numerical ordering. For example, the attribute *fruit* may have the possible values, Apple, Orange, and Pear, which have no ordering. A **binary attribute** is a special case of a nominal attribute with only two states or values.

Although nominal attributes are not standardized as numeric attributes, it is sometime useful to convert a nominal attribute to a set of binary attributes. Let the number of values of a nominal attribute be *v*. We can then create *v* binary attributes to represent them, i.e., one binary attribute for each value. If a data instance for the nominal attribute takes a particular value, the value of its corresponding binary attribute is set to 1, otherwise it is set to 0. The resulting binary attributes can be used as numeric attributes. We will discuss this again in Sect. 4.7.

Example 12: For the nominal attribute *fruit*, we create three binary attributes called, Apple, Orange, and Pear in the new data. If a particular data instance in the original data has Apple as the value for *fruit*, then in the transformed data, we set the value of the attribute Apple to 1, and the values of attributes Orange and Pear to 0. ■

Ordinal (Ordered Categorical) Attributes: An ordinal attribute is like a nominal attribute, but its values have a numerical ordering. For example, the age attribute may have the values, Young, Middle-Age and Old. The common approach to distance computation is to treat ordinal attributes as interval-scaled attributes and use the same methods as for interval-scaled attributes to standardize the values of ordinal attributes.

4.7 Handling of Mixed Attributes

So far, we have assumed that a data set contains only one type of attributes. However, in practice, a data set may contain mixed attributes. That is, it may contain any subset of the six types of attributes, **interval-scaled**, **symmetric binary**, **asymmetric binary**, **ratio-scaled**, **ordinal** and **nominal** attributes. Clustering a data set involving mixed attributes is a challenging problem.

One way to deal with such a data set is to choose a dominant attribute type and then convert the attributes of other types to this type. For example, if most attributes in a data set are interval-scaled, we can convert ordinal attributes and ratio-scaled attributes to interval-scaled attributes as discussed above. It is also appropriate to treat symmetric binary attributes as interval-scaled attributes. However, it does not make much sense to convert a nominal attribute with more than two values or an asymmetric binary attribute to an interval-scaled attribute, but it is still frequently done in practice by assigning some numbers to them according to some hidden ordering. For instance, in the example of Apple, Orange, and Pear, one may order them according to their prices, and thus make the attribute *fruit* an ordinal attribute or even an interval-scaled attribute. In the previous section, we also saw that a nominal attribute can be converted to a set of (symmetric) binary attributes, which in turn can be regarded as interval-scaled attributes.

Another method of handling mixed attributes is to compute the distance of each attribute of the two data points separately and then combine all the individual distances to produce an overall distance. We describe one such method, which is due to Gower [205] and is also described in [218, 271]. We describe the combination formula first (Equation (24)) and then present the methods to compute individual distances.

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \frac{\sum_{f=1}^{r} \delta_{ij}^f d_{ij}^f}{\sum_{f=1}^{r} \delta_{ij}^f}. \tag{24}$$

This distance value is between 0 and 1. r is the number of attributes in the data set. The indicator δ_{ij}^f is 1 if both values x_{if} and x_{jf} for attribute f are non-missing, and it is set to 0 otherwise. It is also set to 0 if attribute f is asymmetric and the match is 0–0. Equation (24) cannot be computed if all δ_{ij}^f 's are 0. In such a case, some default value may be used or one of the data points is removed. d_{ij}^f is the distance contributed by attribute f, and it is in the range 0–1. If f is a binary or nominal attribute,

$$d_{ij}^f = \begin{cases} 1 & if\ x_{if} \neq x_{jf} \\ 0 & otherwise \end{cases} \tag{25}$$

If all the attributes are nominal, Equation (24) reduces to Equation (17). The same is true for symmetric binary attributes, in which we recover the simple matching distance (Equation (10)). When the attributes are all asymmetric, we obtain the Jaccard distance (Equation (14)).

If attribute f is interval-scaled, we use

$$d_{ij}^f = \frac{| x_{if} - x_{jf} |}{R_f} \tag{26}$$

where R_f is the value range of attribute f, which is

$$R_f = \max(f) - \min(f) \tag{27}$$

Ordinal attributes and ratio-scaled attributes are handled in the same way after conversion.

If all the attributes are interval-scaled, Equation (24) becomes the Manhattan distance assuming that all attribute values are standardized by dividing their values with the ranges of their corresponding attributes.

4.8 Which Clustering Algorithm to Use?

Clustering research and application has a long history. Over the years, a vast collection of clustering algorithms has been designed. This chapter only introduced several of the main algorithms.

Given an application data set, choosing the "best" clustering algorithm to cluster the data is a challenge. Every clustering algorithm has limitations and works well with only certain data distributions. However, it is very hard, if not impossible, to know what distribution the application data follows. Worse still, the application data set may not fully follow any "ideal" structure or distribution required by the algorithms. Apart from choosing a suitable clustering algorithm from a large collection of available algorithms, deciding how to standardize the data, to choose a suitable distance function and to select other parameter values (e.g., *k* in the *k*-means algorithm) are complex as well. Due to these complexities, the common practice is to run several algorithms using different distance functions and parameter settings, and then to carefully analyze and compare the results.

The interpretation of the results should be based on insight into the meaning of the original data together with knowledge of the algorithms used. That is, it is crucial that the user of a clustering algorithm fully understands the algorithm and its limitations. He/she should also have the domain expertise to examine the clustering results. In many cases, generating cluster descriptions using a supervised learning method (e.g., decision tree induction) can be particularly helpful to the analysis and comparison.

4.9 Cluster Evaluation

After a set of clusters is found, we need to assess the goodness of the clusters. Unlike classification, where it is easy to measure accuracy using labeled test data, for clustering nobody knows what the correct clusters are given a data set. Thus, the quality of a clustering is much harder to evaluate. We introduce a few commonly used evaluation methods below.

User Inspection: A panel of experts is asked to inspect the resulting clusters and to score them. Since this process is subjective, we take the average of the scores from all the experts as the final score of the clustering. This manual inspection is obviously a labor intensive and time consuming task. It is subjective as well. However, in most applications, some level of manual inspection is necessary because no other existing evaluation methods are able to guarantee the quality of the final clusters. It should be noted that direct user inspection may be easy for certain types of data, but not for others. For example, user inspection is not hard for text documents because one can read them easily. However, for a relational table with only numbers, staring at the data instances in each cluster makes no sense. The user can only meaningfully study the centroids of the clusters, or rules that characterize the clusters generated by a decision tree algorithm or some other supervised learning methods (see Sect. 4.3.1).

Ground Truth: In this method, classification data sets are used to evaluate clustering algorithms. Recall that a classification data set has several classes, and each data instance/point is labeled with one class. Using such a data set for cluster evaluation, we make the assumption that each class corresponds to a cluster. For example, if a data set has three classes, we assume that it has three clusters, and we request the clustering algorithm to also produce three clusters. After clustering, we compare the cluster memberships with the class memberships to determine how good the clustering is. A variety of measures can be used to assess the clustering quality, e.g., entropy, purity, precision, recall, and F-score.

To facilitate evaluation, a confusion matrix can be constructed from the resulting clusters. From the matrix, various measurements can be computed. Let the set of classes in the data set D be $C = (c_1, c_2, \ldots, c_k)$. The clustering method also produces k clusters, which partition D into k disjoint subsets, $D_1, D_2, \ldots, D_k$.

Entropy: For each cluster, we can measure its entropy as follows:

$$entropy(D_i) = -\sum_{j=1}^{k} \Pr_i(c_j) \log_2 \Pr_i(c_j), \tag{28}$$

where $\Pr_i(c_j)$ is the proportion of class c_j data points in cluster i or D_i. The total entropy of the whole clustering (which considers all clusters) is

$$entropy_{total}(D) = \sum_{i=1}^{k} \frac{|D_i|}{|D|} \times entropy(D_i). \tag{29}$$

Purity: This measures the extent that a cluster contains only one class of data. The purity of each cluster is computed with

$$purity(D_i) = \max_j(\Pr_i(c_j)). \tag{30}$$

The total purity of the whole clustering (considering all clusters) is

$$purity_{total}(D) = \sum_{i=1}^{k} \frac{|D_i|}{|D|} \times purity(D_i). \tag{31}$$

Precision, recall, and F-score can be computed as well for each cluster based on the class that is the most frequent in the cluster. Note that these measures are based on a single class (see Sect. 3.3.2).

Example 13: Assume we have a text collection D of 900 documents from three topics (or three classes), Science, Sports, and Politics. Each class has 300 documents, and each document is labeled with one of the topics (classes). We use this collection to perform clustering to find three clusters. Class/topic labels are not used in clustering. After clustering, we want to measure the effectiveness of the clustering algorithm.

First, a confusion matrix (Fig. 4.17) is constructed based on the clustering results. From Fig. 4.17, we see that cluster 1 has 250 Science documents, 20 Sports documents, and 10 Politics documents. The entries of the other rows have similar meanings. The last two columns list the entropy and purity values of each cluster and also the total entropy and purity of the whole clustering (last row). We observe that cluster 1, which contains mainly Science documents, is a much better (or purer) cluster than the other two. This fact is also reflected by both their entropy and purity values.

Cluster	Science	Sports	Politics		Entropy	Purity
1	250	20	10		0.589	0.893
2	20	180	80		1.198	0.643
3	30	100	210		1.257	0.617
Total	300	300	300		1.031	0.711

Fig. 4.17. Confusion matrix with entropy and purity values

Obviously, we can use the total entropy or the total purity to compare different clustering results from the same algorithm with different parameter settings or from different algorithms.

Precision and recall may be computed similarly for each cluster. For example, the precision of Science documents in cluster 1 is 0.89. The recall of Science documents in cluster 1 is 0.83. The F-score for Science documents is thus 0.86. ■

A final remark about this evaluation method is that although an algorithm may perform well on some labeled data sets, there is no guarantee that it will perform well on the actual application data at hand, which has no class labels. However, the fact that it performs well on some labeled data sets does give us some confidence on the quality of the algorithm. This evaluation method is said to be based on **external data** or **information**.

There are also methods that evaluate clusters based on the **internal information** in the clusters (without using external data with class labels). These methods measure **intra-cluster cohesion** (compactness) and **inter-cluster separation** (isolation). Cohesion measures how near the data points in a cluster are to the cluster centroid. Sum of squared error (SSE) is a commonly used measure. Separation measures how far apart different cluster centroids are from one another. Any distance functions can be used for the purpose. We should note, however, that good values for these measurements do not always mean good clusters. In most applications, expert judgments are still the key. Clustering evaluation remains to be a very difficult problem.

Indirect Evaluation: In some applications, clustering is not the primary task. Instead, it is used to help perform another more important task. Then, we can use the performance on the primary task to determine which clustering method is the best for the task. For instance, in a Web usage mining application, the primary task is to recommend books to online shoppers. If the shoppers can be clustered according to their profiles and their past purchasing history, we may be able to provide better recommendations. A few clustering methods can be tried, and their results are then evaluated based on how well they help with the recommendation task. Of course, here we assume that the recommendation results can be reliably evaluated.

4.10 Discovering Holes and Data Regions

In this section, we wander a little to discuss something related but quite different from the preceding algorithms. We show that unsupervised learning tasks may be performed by using supervised learning techniques [350].

In clustering, data points are grouped into clusters according to their distances (or similarities). However, clusters only represent one aspect of the hidden knowledge in data. Another aspect that we have not studied is the **holes**. If we treat data instances as points in an *r*-dimensional space, a hole is simply a region in the space that contains no or few data points. The existence of holes is due to the following two reasons:

1. insufficient data in certain areas, and/or
2. certain attribute-value combinations are not possible or seldom occur.

Although clusters are important, holes in the space can be quite useful too. For example, in a disease database we may find that certain symptoms and/or test values do not occur together, or when a certain medicine is used, some test values never go beyond certain ranges. Discovery of such information can be of great importance in medical domains because it could mean the discovery of a cure to a disease or some biological laws.

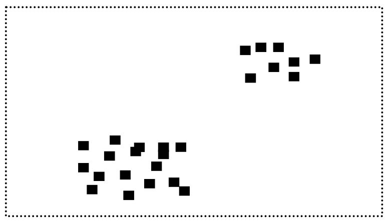
The technique discussed in this section aims to divide the data space into two types of regions, **data regions** (also called **dense regions**) and **empty regions** (also called **sparse regions**). A data region is an area in the space that contains a concentration of data points and can be regarded as a cluster. An empty region is a **hole**. A supervised learning technique similar to decision tree induction is used to separate the two types of regions. The algorithm (called CLTree for CLuser Tree [350]) works for numeric attributes, but can be extended to discrete or categorical attributes.

Decision tree learning is a popular technique for classifying data of various classes. For a decision tree algorithm to work, we need at least two classes of data. A clustering data set, however, has no class label for each data point. Thus, the technique is not directly applicable. However, the problem can be dealt with by a simple idea.

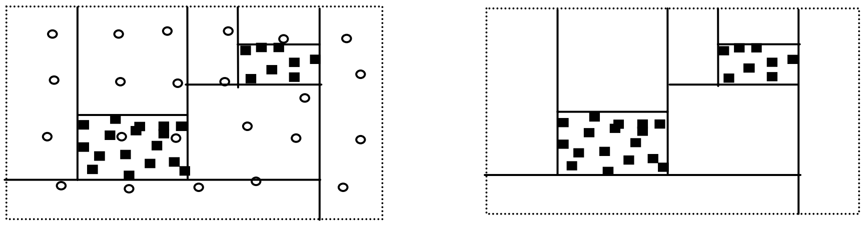
We can regard each data instance/point in the data set as having a class label *Y*. We assume that the data space is uniformly distributed with another type of points, called **non-existing points**, which we will label *N*. With the *N* points added to the original data space, our problem of partitioning the data space into data regions and empty regions becomes a supervised classification problem. The decision tree algorithm can be adapted to solve the problem. Let us use an example to illustrate the idea.

Example 14: Figure 4.18(A) gives a 2-dimensional space with 24 data (*Y*) points. Two data regions (clusters) exist in the space. We then add some uniformly distributed *N* points (represented by "o") to the data space (Fig. 4.18(B)). With the augmented data set, we can run a decision tree algorithm to obtain the partitioning of the space in Fig. 4.18(B). Data regions and empty regions are separated. Each region is a rectangle, which can be expressed as a rule. ■

The reason that this technique works is that if there are clusters (or dense data regions) in the data space, the data points cannot be uniformly distributed in the entire space. By adding some uniformly distributed *N* points, we can isolate data regions because within each data region there are significantly more *Y* points than *N* points. The decision tree technique is well known for this partitioning task.



(A): The original data space



(B). Partitioning with added *N* points

(C). Partitioning without adding *N* points.

Fig. 4.18. Separating data and empty regions using a decision tree

An interesting question is: can the task be performed without physically adding the *N* points to the original data? The answer is yes. Physically adding *N* points increases the size of the data and thus the running time. A more important issue is that it is unlikely that we can have points truly uniformly distributed in a high-dimensional space as we would need an exponential number of them. Fortunately, we do not need to physically add any *N* points. We can compute them when needed. The CLTree method is able to produce the partitioning in Fig. 4.18(C) with no *N* points added. The details are quite involved. Interested readers can refer to [350]. This method has some interesting characteristics:

- It provides descriptions or representations of the resulting data regions and empty regions in terms of hyper-rectangles, which can be expressed as rules as we have seen in Sect. 3.2 of Chap. 3 and in Sect. 4.3.1. Many applications require such descriptions, which can be easily interpreted by users.
- It automatically detects outliers, which are data points in empty regions.
- It may not use all attributes in the data just as in decision tree building

for supervised learning. That is, it can automatically determine what attributes are important and what are not. This means that it can perform subspace clustering, i.e., finding clusters that exist in some subspaces (represented by some subsets of the attributes) of the original space.

This method also has limitations. The main limitation is that data regions of irregular shapes are hard to handle since decision tree learning only generates hyper-rectangles (formed by axis-parallel hyper-planes), which are rules. Hence, an irregularly shaped data or empty region may be split into several hyper-rectangles. Post-processing is needed to join them if desired (see [350] for additional details).

Bibliographic Notes

Clustering or unsupervised learning has a long history and a very large body of work. This chapter described only some widely used core algorithms. Most other algorithms are variations or extensions of these methods. For a comprehensive coverage of clustering, please refer to several books dedicated to clustering, e.g., those by Everitt [167], Hartigan [222], Jain and Dubes [252], Kaufman and Rousseeuw [271], and Mirkin [383]. Most data mining texts also have excellent coverage of clustering techniques, e.g., Han and Kamber [218] and Tan et al. [512], which have influenced the writing of this chapter. Below, we review some more recent developments on clustering and give some further readings.

A density-based clustering algorithm based on local data densities was proposed by Ester et al. [164] and Xu et al. [564] for finding clusters of arbitrary shapes. Hinneburg and Keim [239], Sheikholeslami et al. [485] and Wang et al. [538] proposed several grid-based clustering methods which first partition the space into small grids. A popular neural network clustering algorithm is the Self-Organizing Map (SOM) by Kohonen [287]. Fuzzy clustering (e.g., fuzzy c-means) was studied by Bezdek [50] and Dunn [157]. Cheeseman et al. [94] and Moore [396] studied clustering using mixture models. The method assumes that clusters are a mixture of Gaussians and uses the EM algorithm [127] to learn a mixture density. We will see in Chap. 5 that EM based partially supervised learning algorithms are basically clustering methods with some given initial seeds.

Most clustering algorithms work on numeric data. Categorical data and/or transaction data clustering were investigated by Barbará et al. [36], Ganti et al. [193], Gibson et al. [197], Guha et al. [212], Wang et al. [537], etc. A related area in artificial intelligence is the conceptual clustering, which was studied by Fisher [178], Misha et al. [384] and many others.

Many clustering algorithms, e.g., hierarchical clustering algorithms, have high time complexities and are thus not suitable for large data sets. Scaling up such algorithms becomes an important issue for large applications. Several researchers have designed techniques to scale up clustering algorithms, e.g., Bradley et al. [61], Guha et al. [211], Ng and Han [406], and Zhang et al. [610].

In recent years, there were quite a few new developments in clustering. The first one is **subspace clustering**. Traditional clustering algorithms use the whole space to find clusters, but natural clusters may exist in only some sub-spaces. That is, some clusters may only use certain subsets of the attributes. This problem was investigated by Agrawal et al. [8], Aggarwal et al. [4], Aggarwal and Yu [5], Cheng et al. [99], Liu et al. [350], Zaki et al. [590], and many others.

The second new research is **semi-supervised clustering**, which means that the user can provide some initial information to guide the clustering process. For example, the user can select some initial seeds [39] and/or specify some constraints, e.g., **must-link** (two points must be in the same cluster) and **cannot-link** (two points cannot be in the same cluster) [528].

The third is the **spectral clustering**, which emerged from several fields, e.g., VLSI [17] and computer vision [481, 489, 542]. It clusters data points by computing eigenvectors of the similarity matrix. Recently, it was also studied in machine learning and data mining [141, 404, 594].

Yet another new research is **co-clustering**, which simultaneously clusters both rows and columns. This approach was studied by Cheng and Church [100], Dhillon [134], Dhillon et al. [135], and Hartigan [223].

Regarding document and Web page clustering, most implementations are still based on *k*-means and hierarchical clustering methods, or their variations but using text specific similarity or distance functions. Steinbach et al. [506], and Zhao and Karypis [614, 615] experimented with *k*-means and agglomerative hierarchical clustering methods and also proposed some improvements. Many researchers also worked on **clustering of search engine results** (or snippets) to organize search results into different topics, e.g., Hearst and Pedersen [233], Kummamuru et al. [294], Leouski and Croft [311], Zamir and Etzioni [591, 592], and Zeng et al. [593].