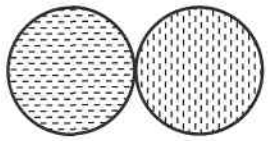
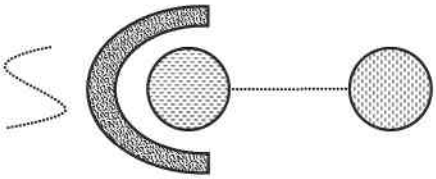
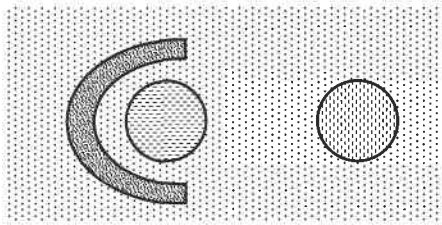
(a) Well-separated clusters. Each point is closer to all of the points in its cluster than to any point in another cluster.



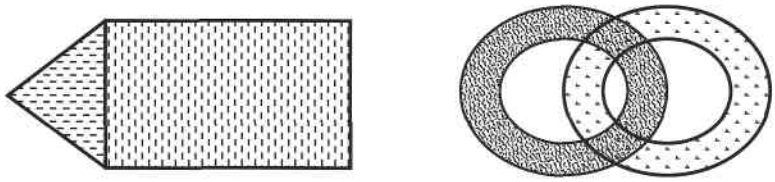
(b) Center-based clusters. Each point is closer to the center of its cluster than to the center of any other cluster.



(c) Contiguity-based clusters. Each point is closer to at least one point in its cluster than to any point in another cluster.



(d) Density-based clusters. Clusters are regions of high density separated by regions of low density.



(e) Conceptual clusters. Points in a cluster share some general property that derives from the entire set of points. (Points in the intersection of the circles belong to both.)

Figure 8.2. Different types of clusters as illustrated by sets of two-dimensional points.

8.2 K-means

Prototype-based clustering techniques create a one-level partitioning of the data objects. There are a number of such techniques, but two of the most prominent are K-means and K-medoid. K-means defines a prototype in terms of a centroid, which is usually the mean of a group of points, and is typically

applied to objects in a continuous n-dimensional space. K-medoid defines a prototype in terms of a medoid, which is the most representative point for a group of points, and can be applied to a wide range of data since it requires only a proximity measure for a pair of objects. While a centroid almost never corresponds to an actual data point, a medoid, by its definition, must be an actual data point. In this section, we will focus solely on K-means, which is one of the oldest and most widely used clustering algorithms.

8.2.1 The Basic K-means Algorithm

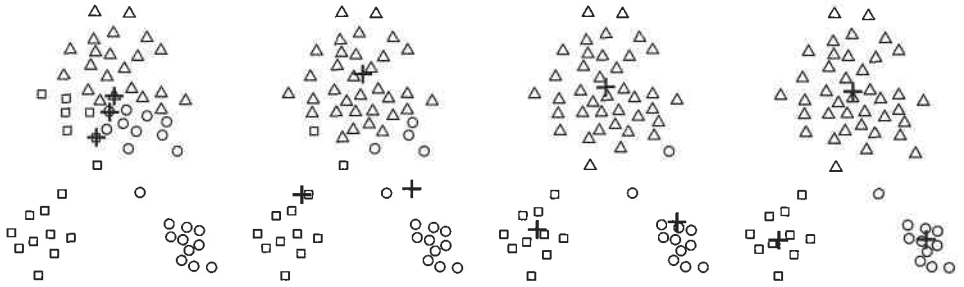
The K-means clustering technique is simple, and we begin with a description of the basic algorithm. We first choose K initial centroids, where K is a user-specified parameter, namely, the number of clusters desired. Each point is then assigned to the closest centroid, and each collection of points assigned to a centroid is a cluster. The centroid of each cluster is then updated based on the points assigned to the cluster. We repeat the assignment and update steps until no point changes clusters, or equivalently, until the centroids remain the same.

K-means is formally described by Algorithm 8.1. The operation of K-means is illustrated in Figure 8.3, which shows how, starting from three centroids, the final clusters are found in four assignment-update steps. In these and other figures displaying K-means clustering, each subfigure shows (1) the centroids at the start of the iteration and (2) the assignment of the points to those centroids. The centroids are indicated by the "+" symbol; all points belonging to the same cluster have the same marker shape.

Algorithm 8.1 Basic K-means algorithm.

1: Select K points as initial centroids.
2: **repeat**
3: Form K clusters by assigning each point to its closest centroid.
4: Recompute the centroid of each cluster.
5: **until** Centroids do not change.

In the first step, shown in Figure 8.3(a), points are assigned to the initial centroids, which are all in the larger group of points. For this example, we use the mean as the centroid. After points are assigned to a centroid, the centroid is then updated. Again, the figure for each step shows the centroid at the beginning of the step and the assignment of points to those centroids. In the second step, points are assigned to the updated centroids, and the centroids



(a) Iteration 1. (b) Iteration 2. (c) Iteration 3. (d) Iteration 4.

Figure 8.3. Using the K-means algorithm to find three clusters in sample data.

are updated again. In steps 2, 3, and 4, which are shown in Figures 8.3 (b), (c), and (d), respectively, two of the centroids move to the two small groups of points at the bottom of the figures. When the K-means algorithm terminates in Figure 8.3(d), because no more changes occur, the centroids have identified the natural groupings of points.

For some combinations of proximity functions and types of centroids, K-means always converges to a solution; i.e., K-means reaches a state in which no points are shifting from one cluster to another, and hence, the centroids don't change. Because most of the convergence occurs in the early steps, however, the condition on line 5 of Algorithm 8.1 is often replaced by a weaker condition, e.g., repeat until only 1% of the points change clusters.

We consider each of the steps in the basic K-means algorithm in more detail and then provide an analysis of the algorithm's space and time complexity.

Assigning Points to the Closest Centroid

To assign a point to the closest centroid, we need a proximity measure that quantifies the notion of "closest" for the specific data under consideration. Euclidean (L_2) distance is often used for data points in Euclidean space, while cosine similarity is more appropriate for documents. However, there may be several types of proximity measures that are appropriate for a given type of data. For example, Manhattan (L_1) distance can be used for Euclidean data, while the Jaccard measure is often employed for documents.

Usually, the similarity measures used for K-means are relatively simple since the algorithm repeatedly calculates the similarity of each point to each centroid. In some cases, however, such as when the data is in low-dimensional

Table 8.1. Table of notation.

Symbol	Description
$\mathbf{x}$	An object.
C_i	The i^{th} cluster.
$\mathbf{c}_i$	The centroid of cluster C_i.
$\mathbf{c}$	The centroid of all points.
m_i	The number of objects in the i^{th} cluster.
m	The number of objects in the data set.
K	The number of clusters.

Euclidean space, it is possible to avoid computing many of the similarities, thus significantly speeding up the K-means algorithm. Bisecting K-means (described in Section 8.2.3) is another approach that speeds up K-means by reducing the number of similarities computed.

Centroids and Objective Functions

Step 4 of the K-means algorithm was stated rather generally as "recompute the centroid of each cluster," since the centroid can vary, depending on the proximity measure for the data and the goal of the clustering. The goal of the clustering is typically expressed by an objective function that depends on the proximities of the points to one another or to the cluster centroids; e.g., minimize the squared distance of each point to its closest centroid. We illustrate this with two examples. However, the key point is this: once we have specified a proximity measure and an objective function, the centroid that we should choose can often be determined mathematically. We provide mathematical details in Section 8.2.6, and provide a non-mathematical discussion of this observation here.

Data in Euclidean Space Consider data whose proximity measure is Euclidean distance. For our objective function, which measures the quality of a clustering, we use the **sum of the squared error (SSE)**, which is also known as scatter. In other words, we calculate the error of each data point, i.e., its Euclidean distance to the closest centroid, and then compute the total sum of the squared errors. Given two different sets of clusters that are produced by two different runs of K-means, we prefer the one with the smallest squared error since this means that the prototypes (centroids) of this clustering are a better representation of the points in their cluster. Using the notation in Table 8.1, the SSE is formally defined as follows:

$$\text{SSE} = \sum_{i=1}^{K} \sum_{x \in C_i} dist(c_i, x)^2 \tag{8.1}$$

where *dist* is the standard Euclidean (L_2) distance between two objects in Euclidean space.

Given these assumptions, it can be shown (see Section 8.2.6) that the centroid that minimizes the SSE of the cluster is the mean. Using the notation in Table 8.1, the centroid (mean) of the i^{th} cluster is defined by Equation 8.2.

$$\mathbf{c}_i = \frac{1}{m_i} \sum_{\mathbf{x} \in C_i} \mathbf{x} \tag{8.2}$$

To illustrate, the centroid of a cluster containing the three two-dimensional points, (1,1), (2,3), and (6,2), is $((1+2+6)/3, ((1+3+2)/3) = (3,2)$.

Steps 3 and 4 of the K-means algorithm directly attempt to minimize the SSE (or more generally, the objective function). Step 3 forms clusters by assigning points to their nearest centroid, which minimizes the SSE for the given set of centroids. Step 4 recomputes the centroids so as to further minimize the SSE. However, the actions of K-means in Steps 3 and 4 are only guaranteed to find a local minimum with respect to the SSE since they are based on optimizing the SSE for specific choices of the centroids and clusters, rather than for all possible choices. We will later see an example in which this leads to a suboptimal clustering.

Document Data To illustrate that K-means is not restricted to data in Euclidean space, we consider document data and the cosine similarity measure. Here we assume that the document data is represented as a document-term matrix as described on page 31. Our objective is to maximize the similarity of the documents in a cluster to the cluster centroid; this quantity is known as the **cohesion** of the cluster. For this objective it can be shown that the cluster centroid is, as for Euclidean data, the mean. The analogous quantity to the total SSE is the total cohesion, which is given by Equation 8.3.

$$\text{Total Cohesion} = \sum_{i=1}^{K} \sum_{\mathbf{x} \in C_i} cosine(\mathbf{x}, \mathbf{c}_i) \tag{8.3}$$

The General Case There are a number of choices for the proximity function, centroid, and objective function that can be used in the basic K-means

Table 8.2. K-means: Common choices for proximity, centroids, and objective functions.

Proximity Function	Centroid	Objective Function
Manhattan (L_1)	median	Minimize sum of the L_1 distance of an object to its cluster centroid
Squared Euclidean (L_2^2)	mean	Minimize sum of the squared L_2 distance of an object to its cluster centroid
cosine	mean	Maximize sum of the cosine similarity of an object to its cluster centroid
Bregman divergence	mean	Minimize sum of the Bregman divergence of an object to its cluster centroid

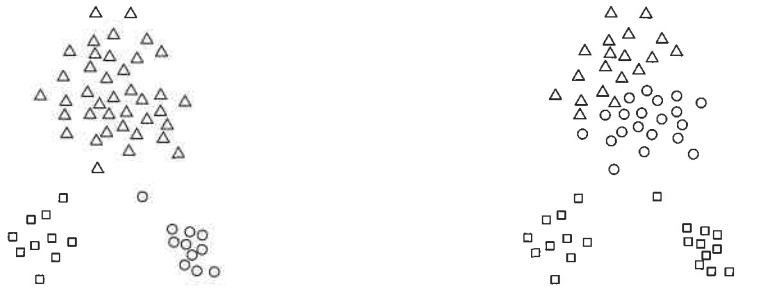
algorithm and that are guaranteed to converge. Table 8.2 shows some possible choices, including the two that we have just discussed. Notice that for Manhattan (L_1) distance and the objective of minimizing the sum of the distances, the appropriate centroid is the median of the points in a cluster.

The last entry in the table, Bregman divergence (Section 2.4.5), is actually a class of proximity measures that includes the squared Euclidean distance, L_2^2, the Mahalanobis distance, and cosine similarity. The importance of Bregman divergence functions is that any such function can be used as the basis of a K-means style clustering algorithm with the mean as the centroid. Specifically, if we use a Bregman divergence as our proximity function, then the resulting clustering algorithm has the usual properties of K-means with respect to convergence, local minima, etc. Furthermore, the properties of such a clustering algorithm can be developed for all possible Bregman divergences. Indeed, K-means algorithms that use cosine similarity or squared Euclidean distance are particular instances of a general clustering algorithm based on Bregman divergences.

For the rest our K-means discussion, we use two-dimensional data since it is easy to explain K-means and its properties for this type of data. But, as suggested by the last few paragraphs, K-means is a very general clustering algorithm and can be used with a wide variety of data types, such as documents and time series.

Choosing Initial Centroids

When random initialization of centroids is used, different runs of K-means typically produce different total SSEs. We illustrate this with the set of two-dimensional points shown in Figure 8.3, which has three natural clusters of points. Figure 8.4(a) shows a clustering solution that is the global minimum of



(a) Optimal clustering.

(b) Suboptimal clustering.

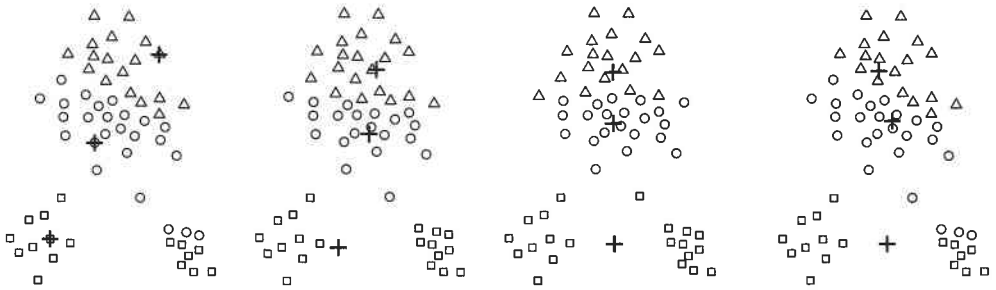
Figure 8.4. Three optimal and non-optimal clusters.

the SSE for three clusters, while Figure 8.4(b) shows a suboptimal clustering that is only a local minimum.

Choosing the proper initial centroids is the key step of the basic K-means procedure. A common approach is to choose the initial centroids randomly, but the resulting clusters are often poor.

Example 8.1 (Poor Initial Centroids). Randomly selected initial centroids may be poor. We provide an example of this using the same data set used in Figures 8.3 and 8.4. Figures 8.3 and 8.5 show the clusters that result from two particular choices of initial centroids. (For both figures, the positions of the cluster centroids in the various iterations are indicated by crosses.) In Figure 8.3, even though all the initial centroids are from one natural cluster, the minimum SSE clustering is still found. In Figure 8.5, however, even though the initial centroids seem to be better distributed, we obtain a suboptimal clustering, with higher squared error. ■

Example 8.2 (Limits of Random Initialization). One technique that is commonly used to address the problem of choosing initial centroids is to perform multiple runs, each with a different set of randomly chosen initial centroids, and then select the set of clusters with the minimum SSE. While simple, this strategy may not work very well, depending on the data set and the number of clusters sought. We demonstrate this using the sample data set shown in Figure 8.6(a). The data consists of two pairs of clusters, where the clusters in each (top-bottom) pair are closer to each other than to the clusters in the other pair. Figure 8.6 (b–d) shows that if we start with two initial centroids per pair of clusters, then even when both centroids are in a single



(a) Iteration 1. (b) Iteration 2. (c) Iteration 3. (d) Iteration 4.

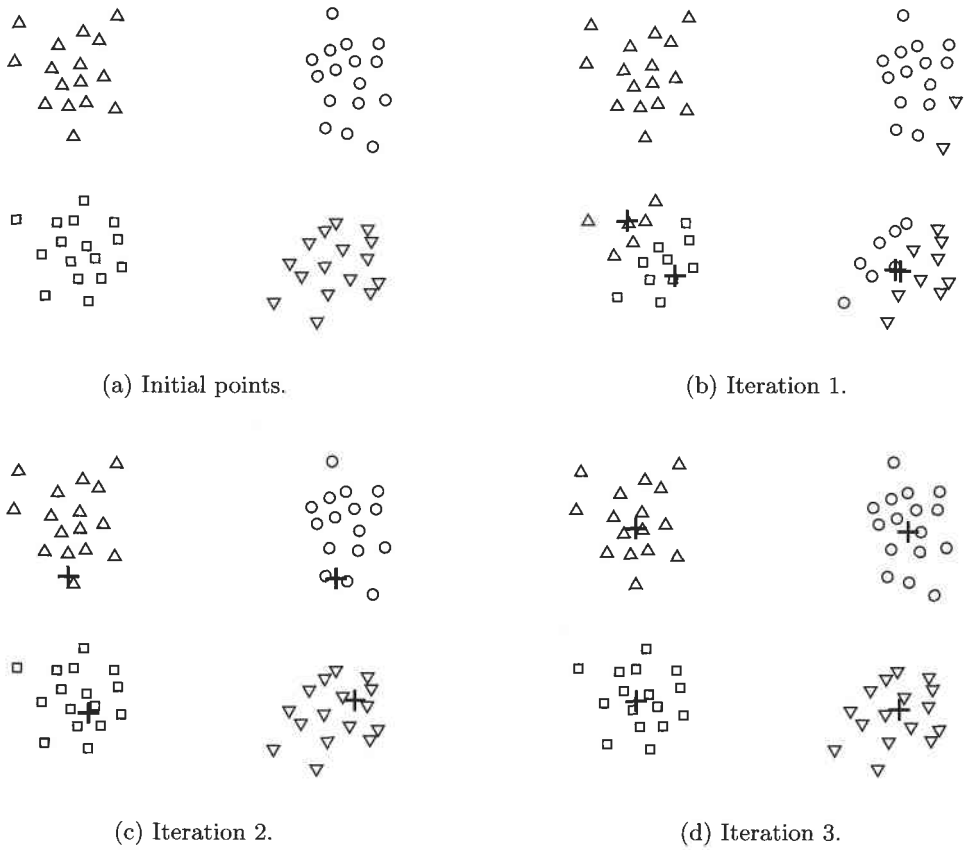
Figure 8.5. Poor starting centroids for K-means.

cluster, the centroids will redistribute themselves so that the "true" clusters are found. However, Figure 8.7 shows that if a pair of clusters has only one initial centroid and the other pair has three, then two of the true clusters will be combined and one true cluster will be split.

Note that an optimal clustering will be obtained as long as two initial centroids fall anywhere in a pair of clusters, since the centroids will redistribute themselves, one to each cluster. Unfortunately, as the number of clusters becomes larger, it is increasingly likely that at least one pair of clusters will have only one initial centroid. (See Exercise 4 on page 559.) In this case, because the pairs of clusters are farther apart than clusters within a pair, the K-means algorithm will not redistribute the centroids between pairs of clusters, and thus, only a local minimum will be achieved. ■

Because of the problems with using randomly selected initial centroids, which even repeated runs may not overcome, other techniques are often employed for initialization. One effective approach is to take a sample of points and cluster them using a hierarchical clustering technique. K clusters are extracted from the hierarchical clustering, and the centroids of those clusters are used as the initial centroids. This approach often works well, but is practical only if (1) the sample is relatively small, e.g., a few hundred to a few thousand (hierarchical clustering is expensive), and (2) K is relatively small compared to the sample size.

The following procedure is another approach to selecting initial centroids. Select the first point at random or take the centroid of all points. Then, for each successive initial centroid, select the point that is farthest from any of the initial centroids already selected. In this way, we obtain a set of initial



(a) Initial points.

(b) Iteration 1.

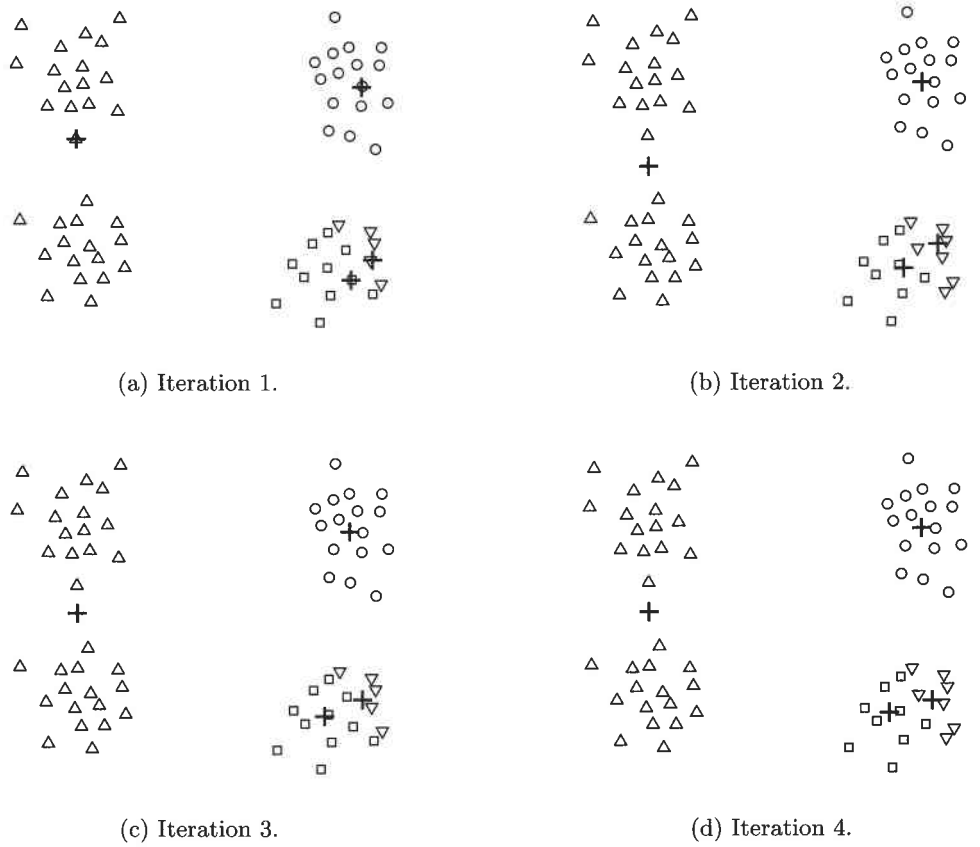
(c) Iteration 2.

(d) Iteration 3.

Figure 8.6. Two pairs of clusters with a pair of initial centroids within each pair of clusters.

centroids that is guaranteed to be not only randomly selected but also well separated. Unfortunately, such an approach can select outliers, rather than points in dense regions (clusters). Also, it is expensive to compute the farthest point from the current set of initial centroids. To overcome these problems, this approach is often applied to a sample of the points. Since outliers are rare, they tend not to show up in a random sample. In contrast, points from every dense region are likely to be included unless the sample size is very small. Also, the computation involved in finding the initial centroids is greatly reduced because the sample size is typically much smaller than the number of points.

Later on, we will discuss two other approaches that are useful for producing better-quality (lower SSE) clusterings: using a variant of K-means that



(a) Iteration 1.

(b) Iteration 2.

(c) Iteration 3.

(d) Iteration 4.

Figure 8.7. Two pairs of clusters with more or fewer than two initial centroids within a pair of clusters.

is less susceptible to initialization problems (bisecting K-means) and using postprocessing to "fixup" the set of clusters produced.

Time and Space Complexity

The space requirements for K-means are modest because only the data points and centroids are stored. Specifically, the storage required is $O((m + K)n)$, where m is the number of points and n is the number of attributes. The time requirements for K-means are also modest—basically linear in the number of data points. In particular, the time required is $O(I * K * m * n)$, where I is the number of iterations required for convergence. As mentioned, I is often small and can usually be safely bounded, as most changes typically occur in the

first few iterations. Therefore, K-means is linear in m, the number of points, and is efficient as well as simple provided that K, the number of clusters, is significantly less than m.

8.2.2 K-means: Additional Issues

Handling Empty Clusters

One of the problems with the basic K-means algorithm given earlier is that empty clusters can be obtained if no points are allocated to a cluster during the assignment step. If this happens, then a strategy is needed to choose a replacement centroid, since otherwise, the squared error will be larger than necessary. One approach is to choose the point that is farthest away from any current centroid. If nothing else, this eliminates the point that currently contributes most to the total squared error. Another approach is to choose the replacement centroid from the cluster that has the highest SSE. This will typically split the cluster and reduce the overall SSE of the clustering. If there are several empty clusters, then this process can be repeated several times.

Outliers

When the squared error criterion is used, outliers can unduly influence the clusters that are found. In particular, when outliers are present, the resulting cluster centroids (prototypes) may not be as representative as they otherwise would be and thus, the SSE will be higher as well. Because of this, it is often useful to discover outliers and eliminate them beforehand. It is important, however, to appreciate that there are certain clustering applications for which outliers should not be eliminated. When clustering is used for data compression, every point must be clustered, and in some cases, such as financial analysis, apparent outliers, e.g., unusually profitable customers, can be the most interesting points.

An obvious issue is how to identify outliers. A number of techniques for identifying outliers will be discussed in Chapter 10. If we use approaches that remove outliers before clustering, we avoid clustering points that will not cluster well. Alternatively, outliers can also be identified in a postprocessing step. For instance, we can keep track of the SSE contributed by each point, and eliminate those points with unusually high contributions, especially over multiple runs. Also, we may want to eliminate small clusters since they frequently represent groups of outliers.

Reducing the SSE with Postprocessing

An obvious way to reduce the SSE is to find more clusters, i.e., to use a larger K. However, in many cases, we would like to improve the SSE, but don't want to increase the number of clusters. This is often possible because K-means typically converges to a local minimum. Various techniques are used to "fix up" the resulting clusters in order to produce a clustering that has lower SSE. The strategy is to focus on individual clusters since the total SSE is simply the sum of the SSE contributed by each cluster. (We will use the terminology *total SSE* and *cluster SSE*, respectively, to avoid any potential confusion.) We can change the total SSE by performing various operations on the clusters, such as splitting or merging clusters. One commonly used approach is to use alternate cluster splitting and merging phases. During a splitting phase, clusters are divided, while during a merging phase, clusters are combined. In this way, it is often possible to escape local SSE minima and still produce a clustering solution with the desired number of clusters. The following are some techniques used in the splitting and merging phases.

Two strategies that decrease the total SSE by increasing the number of clusters are the following:

Split a cluster: The cluster with the largest SSE is usually chosen, but we could also split the cluster with the largest standard deviation for one particular attribute.

Introduce a new cluster centroid: Often the point that is farthest from any cluster center is chosen. We can easily determine this if we keep track of the SSE contributed by each point. Another approach is to choose randomly from all points or from the points with the highest SSE.

Two strategies that decrease the number of clusters, while trying to minimize the increase in total SSE, are the following:

Disperse a cluster: This is accomplished by removing the centroid that corresponds to the cluster and reassigning the points to other clusters. Ideally, the cluster that is dispersed should be the one that increases the total SSE the least.

Merge two clusters: The clusters with the closest centroids are typically chosen, although another, perhaps better, approach is to merge the two clusters that result in the smallest increase in total SSE. These two merging strategies are the same ones that are used in the hierarchical

clustering techniques known as the centroid method and Ward's method, respectively. Both methods are discussed in Section 8.3.

Updating Centroids Incrementally

Instead of updating cluster centroids after all points have been assigned to a cluster, the centroids can be updated incrementally, after each assignment of a point to a cluster. Notice that this requires either zero or two updates to cluster centroids at each step, since a point either moves to a new cluster (two updates) or stays in its current cluster (zero updates). Using an incremental update strategy guarantees that empty clusters are not produced since all clusters start with a single point, and if a cluster ever has only one point, then that point will always be reassigned to the same cluster.

In addition, if incremental updating is used, the relative weight of the point being added may be adjusted; e.g., the weight of points is often decreased as the clustering proceeds. While this can result in better accuracy and faster convergence, it can be difficult to make a good choice for the relative weight, especially in a wide variety of situations. These update issues are similar to those involved in updating weights for artificial neural networks.

Yet another benefit of incremental updates has to do with using objectives other than "minimize SSE." Suppose that we are given an arbitrary objective function to measure the goodness of a set of clusters. When we process an individual point, we can compute the value of the objective function for each possible cluster assignment, and then choose the one that optimizes the objective. Specific examples of alternative objective functions are given in Section 8.5.2.

On the negative side, updating centroids incrementally introduces an order dependency. In other words, the clusters produced may depend on the order in which the points are processed. Although this can be addressed by randomizing the order in which the points are processed, the basic K-means approach of updating the centroids after all points have been assigned to clusters has no order dependency. Also, incremental updates are slightly more expensive. However, K-means converges rather quickly, and therefore, the number of points switching clusters quickly becomes relatively small.

8.2.3 Bisecting K-means

The bisecting K-means algorithm is a straightforward extension of the basic K-means algorithm that is based on a simple idea: to obtain K clusters, split the set of all points into two clusters, select one of these clusters to split, and

so on, until K clusters have been produced. The details of bisecting K-means are given by Algorithm 8.2.

Algorithm 8.2 Bisecting K-means algorithm.

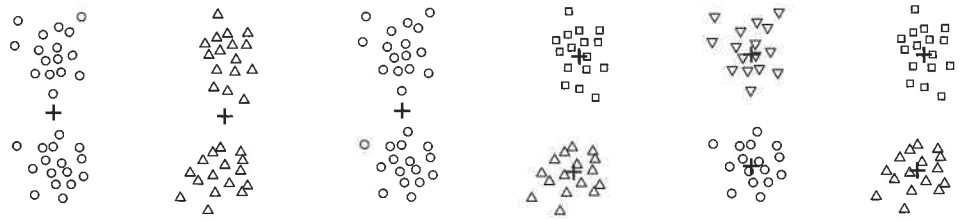
```
1: Initialize the list of clusters to contain the cluster consisting of all points.
2: repeat
3:    Remove a cluster from the list of clusters.
4:    {Perform several "trial" bisections of the chosen cluster.}
5:    for i = 1 to number of trials do
6:       Bisect the selected cluster using basic K-means.
7:    end for
8:    Select the two clusters from the bisection with the lowest total SSE.
9:    Add these two clusters to the list of clusters.
10: until Until the list of clusters contains K clusters.
```

There are a number of different ways to choose which cluster to split. We can choose the largest cluster at each step, choose the one with the largest SSE, or use a criterion based on both size and SSE. Different choices result in different clusters.

We often refine the resulting clusters by using their centroids as the initial centroids for the basic K-means algorithm. This is necessary because, although the K-means algorithm is guaranteed to find a clustering that represents a local minimum with respect to the SSE, in bisecting K-means we are using the K-means algorithm "locally," i.e., to bisect individual clusters. Therefore, the final set of clusters does not represent a clustering that is a local minimum with respect to the total SSE.

Example 8.3 (Bisecting K-means and Initialization). To illustrate that bisecting K-means is less susceptible to initialization problems, we show, in Figure 8.8, how bisecting K-means finds four clusters in the data set originally shown in Figure 8.6(a). In iteration 1, two pairs of clusters are found; in iteration 2, the rightmost pair of clusters is split; and in iteration 3, the leftmost pair of clusters is split. Bisecting K-means has less trouble with initialization because it performs several trial bisections and takes the one with the lowest SSE, and because there are only two centroids at each step. ∎

Finally, by recording the sequence of clusterings produced as K-means bisects clusters, we can also use bisecting K-means to produce a hierarchical clustering.



(a) Iteration 1. (b) Iteration 2. (c) Iteration 3.

Figure 8.8. Bisecting K-means on the four clusters example.

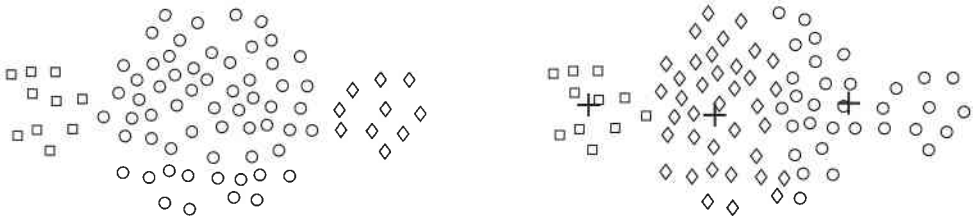
8.2.4 K-means and Different Types of Clusters

K-means and its variations have a number of limitations with respect to finding different types of clusters. In particular, K-means has difficulty detecting the "natural" clusters, when clusters have non-spherical shapes or widely different sizes or densities. This is illustrated by Figures 8.9, 8.10, and 8.11. In Figure 8.9, K-means cannot find the three natural clusters because one of the clusters is much larger than the other two, and hence, the larger cluster is broken, while one of the smaller clusters is combined with a portion of the larger cluster. In Figure 8.10, K-means fails to find the three natural clusters because the two smaller clusters are much denser than the larger cluster. Finally, in Figure 8.11, K-means finds two clusters that mix portions of the two natural clusters because the shape of the natural clusters is not globular.

The difficulty in these three situations is that the K-means objective function is a mismatch for the kinds of clusters we are trying to find since it is minimized by globular clusters of equal size and density or by clusters that are well separated. However, these limitations can be overcome, in some sense, if the user is willing to accept a clustering that breaks the natural clusters into a number of subclusters. Figure 8.12 shows what happens to the three previous data sets if we find six clusters instead of two or three. Each smaller cluster is pure in the sense that it contains only points from one of the natural clusters.

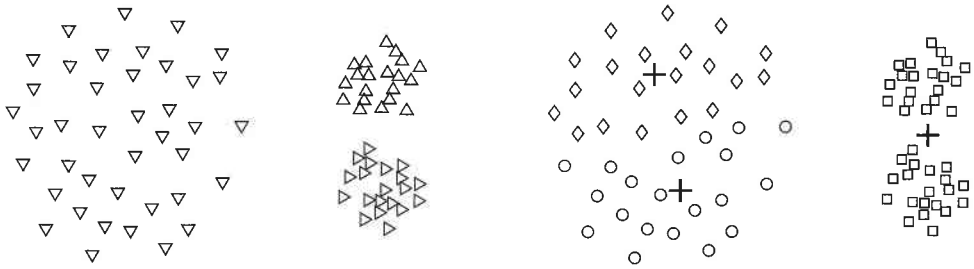
8.2.5 Strengths and Weaknesses

K-means is simple and can be used for a wide variety of data types. It is also quite efficient, even though multiple runs are often performed. Some variants, including bisecting K-means, are even more efficient, and are less susceptible to initialization problems. K-means is not suitable for all types of data,



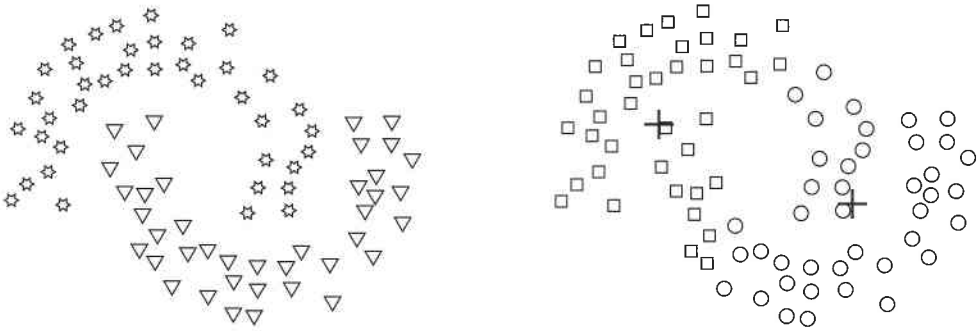
(a) Original points. (b) Three K-means clusters.

Figure 8.9. K-means with clusters of different size.



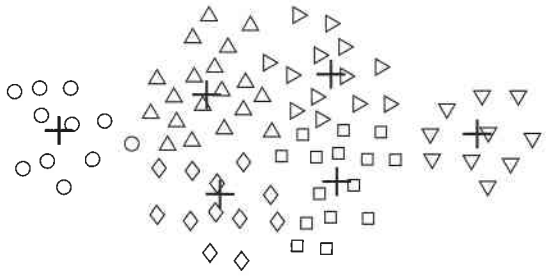
(a) Original points. (b) Three K-means clusters.

Figure 8.10. K-means with clusters of different density.

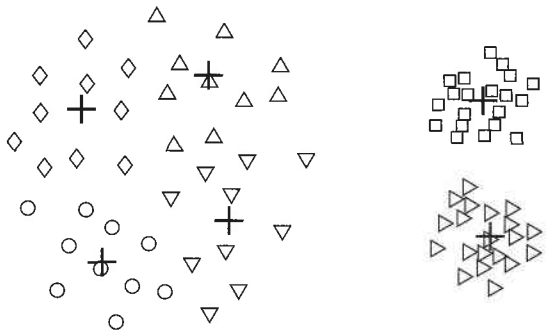


(a) Original points. (b) Two K-means clusters.

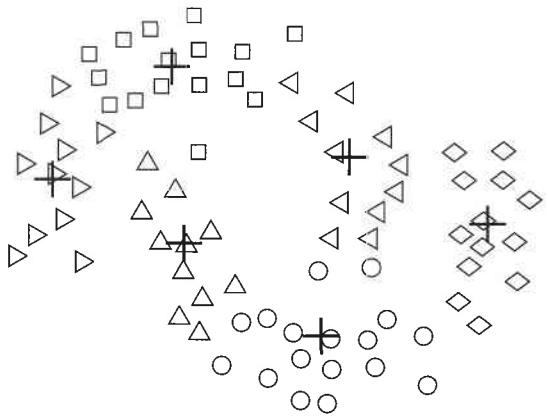
Figure 8.11. K-means with non-globular clusters.



(a) Unequal sizes.



(b) Unequal densities.



(c) Non-spherical shapes.

Figure 8.12. Using K-means to find clusters that are subclusters of the natural clusters.

however. It cannot handle non-globular clusters or clusters of different sizes and densities, although it can typically find pure subclusters if a large enough number of clusters is specified. K-means also has trouble clustering data that contains outliers. Outlier detection and removal can help significantly in such situations. Finally, K-means is restricted to data for which there is a notion of a center (centroid). A related technique, K-medoid clustering, does not have this restriction, but is more expensive.

8.2.6 K-means as an Optimization Problem

Here, we delve into the mathematics behind K-means. This section, which can be skipped without loss of continuity, requires knowledge of calculus through partial derivatives. Familiarity with optimization techniques, especially those based on gradient descent, may also be helpful.

As mentioned earlier, given an objective function such as "minimize SSE," clustering can be treated as an optimization problem. One way to solve this problem—to find a global optimum—is to enumerate all possible ways of dividing the points into clusters and then choose the set of clusters that best satisfies the objective function, e.g., that minimizes the total SSE. Of course, this exhaustive strategy is computationally infeasible and as a result, a more practical approach is needed, even if such an approach finds solutions that are not guaranteed to be optimal. One technique, which is known as **gradient descent**, is based on picking an initial solution and then repeating the following two steps: compute the change to the solution that best optimizes the objective function and then update the solution.

We assume that the data is one-dimensional, i.e., $dist(x, y) = (x - y)^2$. This does not change anything essential, but greatly simplifies the notation.

Derivation of K-means as an Algorithm to Minimize the SSE

In this section, we show how the centroid for the K-means algorithm can be mathematically derived when the proximity function is Euclidean distance and the objective is to minimize the SSE. Specifically, we investigate how we can best update a cluster centroid so that the cluster SSE is minimized. In mathematical terms, we seek to minimize Equation 8.1, which we repeat here, specialized for one-dimensional data.

$$\text{SSE} = \sum_{i=1}^{K} \sum_{x \in C_i} (c_i - x)^2 \tag{8.4}$$

Here, C_i is the i^{th} cluster, x is a point in C_i, and c_i is the mean of the i^{th} cluster. See Table 8.1 for a complete list of notation.

We can solve for the k^{th} centroid c_k, which minimizes Equation 8.4, by differentiating the SSE, setting it equal to 0, and solving, as indicated below.

$$\begin{aligned}
\frac{\partial}{\partial c_k}\text{SSE} &= \frac{\partial}{\partial c_k}\sum_{i=1}^{K}\sum_{x\in C_i}(c_i - x)^2 \\
&= \sum_{i=1}^{K}\sum_{x\in C_i}\frac{\partial}{\partial c_k}(c_i - x)^2 \\
&= \sum_{x\in C_k} 2 * (c_k - x_k) = 0
\end{aligned}$$

$$\sum_{x\in C_k} 2 * (c_k - x_k) = 0 \Rightarrow m_k c_k = \sum_{x\in C_k} x_k \Rightarrow c_k = \frac{1}{m_k}\sum_{x\in C_k} x_k$$

Thus, as previously indicated, the best centroid for minimizing the SSE of a cluster is the mean of the points in the cluster.

Derivation of K-means for SAE

To demonstrate that the K-means algorithm can be applied to a variety of different objective functions, we consider how to partition the data into K clusters such that the sum of the Manhattan (L_1) distances of points from the center of their clusters is minimized. We are seeking to minimize the sum of the L_1 absolute errors (SAE) as given by the following equation, where $dist_{L_1}$ is the L_1 distance. Again, for notational simplicity, we use one-dimensional data, i.e., $dist_{L_1} = |c_i - x|$.

$$\text{SAE} = \sum_{i=1}^{K}\sum_{x\in C_i} dist_{L_1}(c_i, x) \tag{8.5}$$

We can solve for the k^{th} centroid c_k, which minimizes Equation 8.5, by differentiating the SAE, setting it equal to 0, and solving.

$$\begin{aligned}\frac{\partial}{\partial c_k}\text{SAE} &= \frac{\partial}{\partial c_k}\sum_{i=1}^{K}\sum_{x\in C_i}|c_i - x| \\ &= \sum_{i=1}^{K}\sum_{x\in C_i}\frac{\partial}{\partial c_k}|c_i - x| \\ &= \sum_{x\in C_k}\frac{\partial}{\partial c_k}|c_k - x| = 0\end{aligned}$$

$$\sum_{x\in C_k}\frac{\partial}{\partial c_k}|c_k - x| = 0 \Rightarrow \sum_{x\in C_k} sign(x - c_k) = 0$$

If we solve for c_k, we find that $c_k = median\{x \in C_k\}$, the median of the points in the cluster. The median of a group of points is straightforward to compute and less susceptible to distortion by outliers.

8.3 Agglomerative Hierarchical Clustering

Hierarchical clustering techniques are a second important category of clustering methods. As with K-means, these approaches are relatively old compared to many clustering algorithms, but they still enjoy widespread use. There are two basic approaches for generating a hierarchical clustering:

Agglomerative: Start with the points as individual clusters and, at each step, merge the closest pair of clusters. This requires defining a notion of cluster proximity.

Divisive: Start with one, all-inclusive cluster and, at each step, split a cluster until only singleton clusters of individual points remain. In this case, we need to decide which cluster to split at each step and how to do the splitting.

Agglomerative hierarchical clustering techniques are by far the most common, and, in this section, we will focus exclusively on these methods. A divisive hierarchical clustering technique is described in Section 9.4.2.

A hierarchical clustering is often displayed graphically using a tree-like diagram called a **dendrogram**, which displays both the cluster-subcluster