UNIT-6

UNIT –VI Syllabus

Cluster Analysis: Basic Concepts and Algorithms:Overview: What Is Cluster Analysis? Different Types of Clustering, Different Types of Clusters; K-means: The Basic K-means Algorithm, K-means Additional Issues, Bisecting K-means, Strengths and Weaknesses; Agglomerative Hierarchical Clustering: Basic Agglomerative Hierarchical Clustering Algorithm DBSCAN: Traditional Density Center-Based Approach, DBSCAN Algorithm, Strengths and Weaknesses. **(Tan &Vipin)**

Cluster Analysis

Cluster analysis groups data objects based on information found only in the data that describes the objects and their relationships. The goal is that the objects within a group be similar (or related) to one another and different from (or unrelated to) the objects in other groups. The greater the similarity (or homogeneity) within a group and the greater the difference between groups, the better or more distinct the clustering.

In many applications, the notion of a cluster is not well defined. To better understand the diffculty of deciding what constitutes a cluster, consider Figure 1, which shows 20 points and three different ways of dividing them into clusters. The shapes of the markers indicate cluster membership. Figures 1(b) and 1(d) divide the data into two and six parts, respectively. However, the apparent division of each of the two larger clusters into three subclusters may simply be an artifact of the human visual system. Also, it may not be unreasonable to say that the points form four clusters, as shown in Figure 1(c). This figure illustrates that the definition of a cluster is imprecise and that the best definition depends on the nature of data and the desired results. Cluster analysis is related to other techniques that are used to divide data objects into groups. For instance, clustering can be regarded as a form of classification in that it creates a labeling of objects with class (cluster) labels. However, it derives these labels only from the data. In contrast, classification in the sense of Chapter 3 is **supervised classiftcation**; i.e., new, unlabeled objects are assigned a class label using a model developed from objects with known class labels. For this reason, cluster analysis is sometimes referred to as **unsupervised classiftcation**. When the term classification is used

(a) Original points. (b) Two clusters.

(c) Four clusters. (d) Six clusters.

Figure 1. Three different ways of clustering the same set of points.

without any qualification within data mining, it typically refers to supervised classification.

Also, while the terms segmentation and partitioning are sometimes used as synonyms for clustering, these terms are frequently used for approaches outside the traditional bounds of cluster analysis. For example, the term partitioning is often used in connection with techniques that divide graphs into subgraphs and that are not strongly connected to clustering.

Segmentation often refers to the division of data into groups using simple techniques; e.g., an image can be split into segments based only on pixel intensity and color, or people can be divided into groups based on their income. Nonetheless, some work in graph partitioning and in image and segmentation is related to cluster analysis.

1.1 Different Types of Clusterings

An entire collection of clusters is commonly referred to as a **clustering**, and in this section, we distinguish various types of clusterings: hierarchical (nested) versus partitional (unnested), exclusive versus overlapping versus fuzzy, and complete versus partial.

1.2 Hierarchical versus Partitional The most commonly discussed distinction among deferent types of clusterings is whether the set of clusters is nested or un nested, or in more traditional terminology, hierarchical or partitional. A **partitional clustering** is simply a division of the set of data objects into non-overlapping subsets (clusters) such that each data object is in exactly one subset. Taken individually, each collection of clusters in Figures 1 (b–d) is a partitional clustering. If we permit clusters to have subclusters, then we obtain a **hierarchical clustering**, which is a set of nested clusters that are organized as a tree. Each node (cluster) in the tree (except for the leaf nodes) is the union of its children (subclusters), and the root of the tree is the cluster containing all the objects. Often, but not always, the leaves of the tree are singleton clusters of individual data objects. If we allow clusters to be nested, then one interpretation of Figure 1(a) is that it has two subclusters (Figure 1(b)), each of which, in turn, has three subclusters (Figure 1(d)). The clusters shown in Figures 1 (a–d), when taken in that order, also form a hierarchical (nested) clustering with, respectively, 1, 2, 4, and 6 clusters on each level. Finally, note that a hierarchical clustering can be viewed as a sequence of partitional clusterings and a partitional clustering can be obtained by taking any member of that sequence; i.e., by cutting the hierarchical tree at a particular level.

Exclusive versus Overlapping versus Fuzzy The clusterings shown in Figure 1 are all **exclusive**, as they assign each object to a single cluster. There are many situations in which a point could reasonably be placed in more than one cluster, and these situations are better addressed by non-exclusive clustering. In the most general sense, an **overlapping** or **non-exclusive clustering** is used to reflect the fact that an object can *simultaneously* belong to more than one group (class). For instance, a person at a university can be both an enrolled student and an employee of the university. A non-exclusive clustering is also often used when, for example, an object is "between" two or more clusters and could reasonably be assigned to any of these clusters. Imagine a point halfway between two of the clusters of Figure 1. Rather than make a somewhat arbitrary assignment of the object to a single cluster, it is placed in all of the "equally good" clusters.

In a **fuzzy clustering**, every object belongs to every cluster with a membership weight that is between 0 (absolutely doesn't belong) and 1 (absolutely belongs). In other words, clusters are treated as fuzzy sets. (Mathematically, a fuzzy set is one in which an object belongs to every set with a weight that is between 0 and 1. In fuzzy clustering, we often impose the additional constraint that the sum of the weights for each object must equal 1.) Similarly, probabilistic clustering techniques compute the probability with

which each point belongs to each cluster, and these probabilities must also sum to 1. Because the membership weights or probabilities for any object sum to 1, a fuzzy or probabilistic clustering does not address true multiclass situations, such as the case of a student employee, where an object belongs to multiple classes. Instead, these approaches are most appropriate for avoiding the arbitrariness of assigning an object to only one cluster when it is close to several. In practice, a fuzzy or probabilistic clustering is often converted to an exclusive clustering by assigning each object to the cluster in which its membership weight or probability is highest.

Complete versus Partial A **complete clustering** assigns every object to a cluster, whereas a **partial clustering** does not. The motivation for a partial clustering is that some objects in a data set may not belong to well- defined groups. Many times objects in the data set represent noise, outliers, or "uninteresting background." For example, some newspaper stories share a common theme, such as global warming, while other stories are more generic or one-of-a-kind. Thus, to find the important topics in last month's stories, we often want to search only for clusters of documents that are tightly related by a common theme. In other cases, a complete clustering of the objects is desired. For example, an application that uses clustering to organize documents for browsing needs to guarantee that all documents can be browsed.

1.3 Different Types of Clusters

Clustering aims to find useful groups of objects (clusters), where usefulness is defined by the goals of the data analysis. Not surprisingly, several different notions of a cluster prove useful in practice. In order to visually illustrate the differences among these types of clusters, we use two-dimensional points, as shown in Figure 2, as our data objects. We stress, however, that the types of clusters described here are equally valid for other kinds of data.

Well-Separated A cluster is a set of objects in which each object is closer (or more similar) to every other object in the cluster than to any object not in the cluster. Sometimes a threshold is used to specify that all the objects in a cluster must be sufficiently close (or similar) to one another. This idealistic definition of a cluster is satisfied only when the data contains natural clusters that are quite far from each other. Figure 2(a) gives an example of well- separated clusters that consists of two groups of points in a two-dimensional space. The distance between any two points in different groups is larger than the distance between any two points within a group. Well-separated clusters do not need to be globular, but can have any shape.

Prototype-Based A cluster is a set of objects in which each object is closer (more similar) to the prototype that defines the cluster than to the prototype of any other cluster. For data with continuous attributes, the prototype of a cluster is often a centroid, i.e., the average (mean) of all the points in the cluster. When a centroid is not meaningful, such as when the data has categorical attributes, the prototype is often a medoid, i.e., the most representative point of a cluster. For many types of data, the prototype can be regarded as the most central point, and in such instances, we commonly refer to prototype- based clusters as **center-based clusters**. Not surprisingly, such clusters tend to be globular. Figure 2(b) shows an example of center-based clusters.

Graph-Based If the data is represented as a graph, where the nodes are objects and the links represent connections among objects (see Section 2.1.2), then a cluster can be defined as a **connected component**; i.e., a group of objects that are connected to one another, but that have no connection to objects outside the group. An important example of graph-based clusters is a **contiguity-based cluster**, where two objects are connected only if they

are within a specified distance of each other. This implies that each object in a contiguity-based cluster is closer to some other object in the cluster than to any point in a different cluster. Figure 2(c) shows an example of such clusters for two-dimensional points. This definition of a cluster is useful when clusters are irregular or intertwined. However, this approach can have trouble when noise is present since, as illustrated by the two spherical clusters of Figure 2(c), a small bridge of points can merge two distinct clusters.

Other types of graph-based clusters are also possible. One such approach (Section 2) defines a cluster as a **clique**; i.e., a set of nodes in a graph that are completely connected to each other. Specifically, if we add connections between objects in the order of their distance from one another, a cluster is formed when a set of objects forms a clique. Like prototype-based clusters, such clusters tend to be globular.

Density-Based A cluster is a dense region of objects that is surrounded by a region of low density. Figure 2(d) shows some density-based clusters for data created by adding noise to the data of Figure 2(c). The two circular clusters are not merged, as in Figure 2(c), because the bridge between them fades into the noise. Likewise, the curve that is present in Figure 2(c) also fades into the noise and does not form a cluster in Figure 2(d). A density- based definition of a cluster is often employed when the clusters are irregular or intertwined, and when noise and outliers are present. By contrast, a contiguity- based definition of a cluster would not work well for the data of Figure 2(d) because the noise would tend to form bridges between clusters.

Shared-Property (Conceptual Clusters) More generally, we can define a cluster as a set of objects that share some property. This definition en- compasses all the previous definitions of a cluster; e.g., objects in a center- based cluster share the property that they are all closest to the same centroid or medoid. However, the shared-property approach also includes new types of clusters. Consider the clusters shown in Figure 2(e). A triangular area (cluster) is adjacent to a rectangular one, and there are two intertwined circles (clusters). In both cases, a clustering algorithm would need a very specific concept of a cluster to successfully detect these clusters. The process of finding such clusters is called conceptual clustering. However, too sophisticated a notion of a cluster would take us into the area of pattern recognition, and thus, we only consider simpler types of clusters in this book.

Road Map

In this topic, we use the following three simple, but important techniques to introduce many of the concepts involved in cluster analysis.

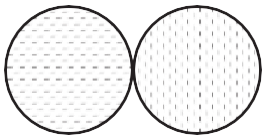
K-means. This is a prototype-based, partitional clustering technique that attempts to find a user-specified number of clusters (K), which are represented by their centroids.

Agglomerative Hierarchical Clustering. This clustering approach refers to a collection of closely related clustering techniques that produce a hierarchical clustering by starting with each point as a singleton cluster and then repeatedly merging the two closest clusters until a single, all- encompassing cluster remains. Some of these techniques have a natural interpretation in terms of graph-based clustering, while others have an interpretation in terms of a prototype-based approach.

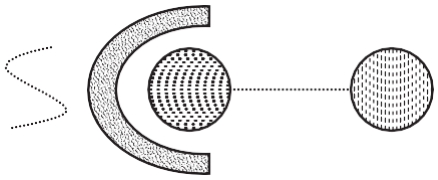
DBSCAN. This is a density-based clustering algorithm that produces a partitional clustering, in which the number of clusters is automatically determined by the algorithm. Points in low-density regions are classified as noise and omitted; thus, DBSCAN does not produce a complete clustering.



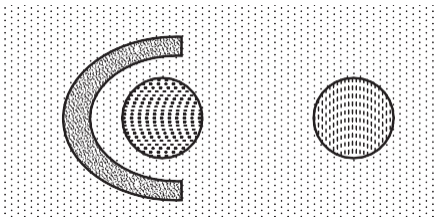
(a) Well-separated clusters. Each point is closer to all of the points in its cluster than to any point in another cluster.



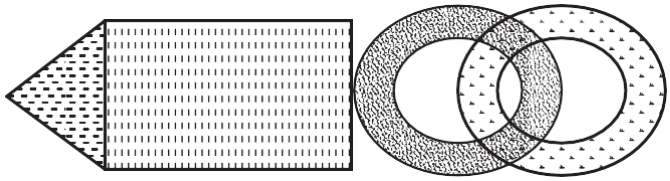
(b) Center-based clusters. Each point is closer to the center of its cluster than to the center of any other cluster.



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



Density-based clusters. Clusters are regions of high density sepa- rated by regions of low density.



(a) 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 2. Different types of clusters as illustrated by sets of two-dimensional points.

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.

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 1. The operation of K-means is illustrated in Figure 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 1 Basic K-means algorithm.

1: Select *K* points as initial centroids.

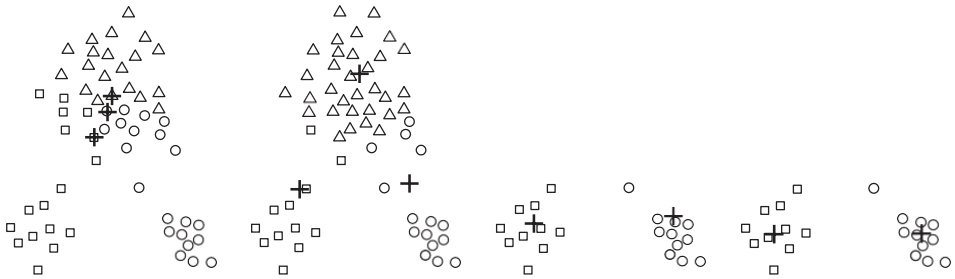
2: **repeat**

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 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 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 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 3(d), because no more changes occur, the centroids have identified the natural groupings of points.

For a number of combinations of proximity functions and types of cen- troids, 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 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, several types of proximity measures can be 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 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 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: after we have specified a proximity measure and an objective function, the centroid that we should choose can often be determined mathematically. We provide mathe- matical details in Section 2.6, and provide a non-mathematical discussion of this observation here.

Data in Euclidean Space Consider data whose proximity measure is Eu- clidean 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 1, the SSE is formally defined as follows:

$$\text{SSE} = \sum_{i=1}^{K} \sum_{\mathbf{x} \in C_i} dist(\mathbf{c}_i, \mathbf{x})^2 \qquad (1)$$

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

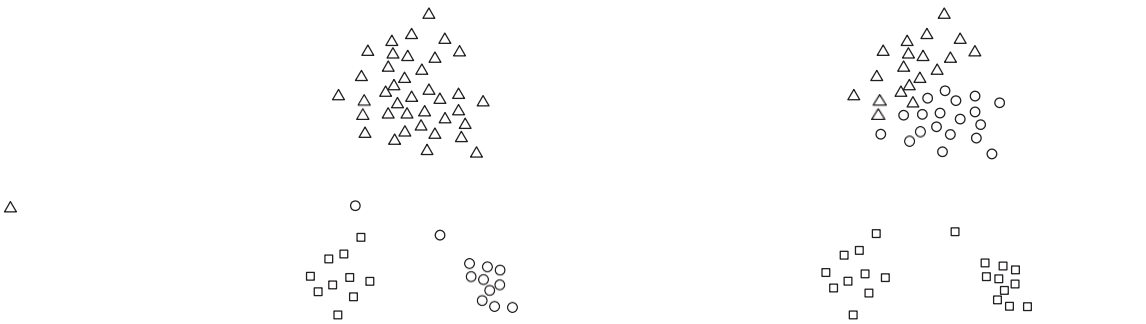
Given these assumptions, it can be shown (see Section 2.6) that the centroid that minimizes the SSE of the cluster is the mean. Using the notation in Table 1, the centroid (mean) of the i^{th} cluster is defined by Equation 2

$$\mathbf{c}_i = \frac{1}{m_i} \sum_{\mathbf{x} \in C_i} \mathbf{x} \qquad (2)$$

algorithm and that are guaranteed to converge. Table 2 shows some possible choices, including the two that we have just discussed. Notice that for Man- hattan (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 2.4.8), is actually a class of proximity measures that includes the squared Euclidean distance, L^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 con- vergence, local minima, etc. Furthermore, the properties of such a clustering algorithm can be developed for all possible Bregman divergences. Forexample, 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 of 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 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 3, which has three natural clusters of points. Figure 4(a) shows a clustering solution that is the global minimum of



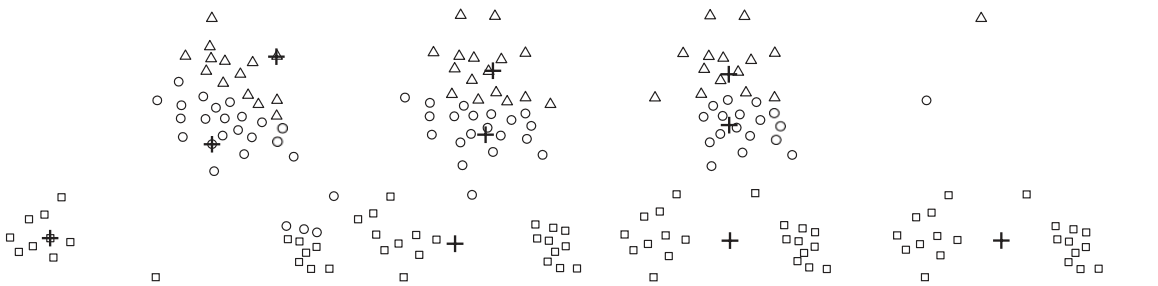
(a) Optimal clustering. (b) Suboptimal clustering.

Figure 4. Three optimal and non-optimal clusters.

the SSE for three clusters, while Figure 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 1 (Poor Initial Centroids)**.** Randomly selected initial centroids can be poor. We provide an example of this using the same data set used in Figures 3 and 4. Figures 3 and 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 3, even though all the initial centroids are from one natural cluster, the minimum SSE clustering is still found. In Figure 5, however, even though the initial centroids seem to be better distributed, we obtain a suboptimal clustering, with higher squared error.

Example 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 might 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 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 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 cluster, the centroids



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

Figure 5. Poor starting centroids for K-means.

will redistribute themselves so that the "true" clusters are found. However, Figure 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 603. 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 might 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 centroids

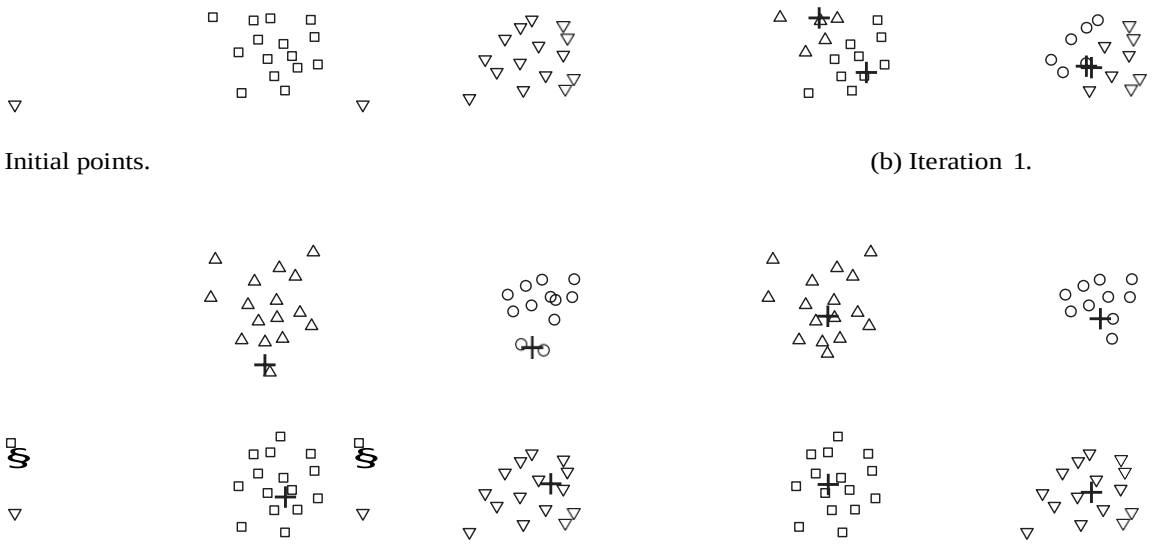
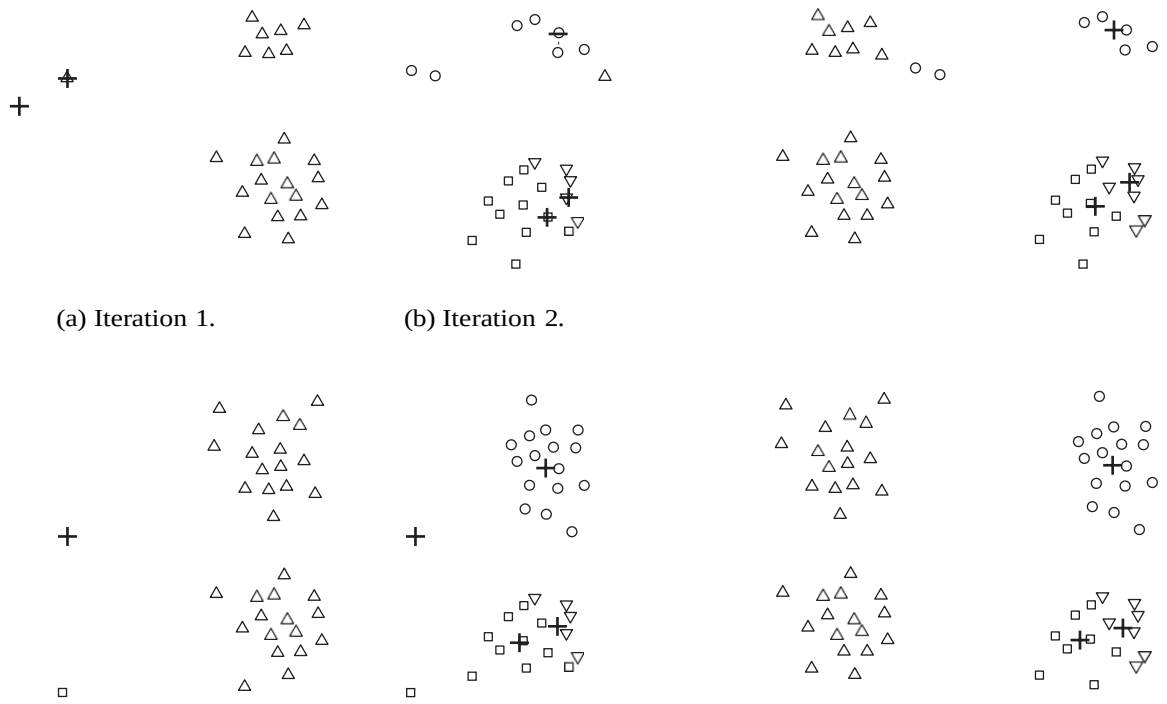


Figure 6. Two pairs of clusters with a pair of initial centroids within each pair of clusters.

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), which can lead to a situation where many clusters have just one point—an outlier—which reduces the number of centroids for forming clusters for the majority of points. 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. Because 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.



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

(c) Iteration 3. (d) Iteration 4.

Figure 7. Two pairs of clusters with more or fewer than two initial centroids within a pair of clusters.

K-means++ More recently, a new approach for initializing K-means, called K-means++, has been developed. This procedure is guaranteed to find a K-means clustering solution that is optimal to within a factor of $O \log(k)$, which in practice translates into noticeably better clustering results in terms of lower SSE. This technique is similar to the idea just discussed of picking the first centroid at random and then picking each remaining centroid as the point as far from the remaining centroids as possible. Specifically, K-means++ picks centroids incrementally until k centroids have been picked. At every such step, each point has a probability of being picked as the new centroid that is proportional to the square of its distance to its closest centroid. It might seem that this approach might tend to choose outliers for centroids, but because outliers are rare, by definition, this is unlikely.

The details of K-means++ initialization are given by Algorithm 2. The rest of the algorithm is the same as ordinary K-means.

Algorithm 2 K-means++ initialization algorithm.

1: For the first centroid, pick one of the points at random.

2: **for** i = 1 to *number of trials* **do**

3: Compute the distance, $d(x)$, of each point to its closest centroid.

4: Assign each point a probability proportional to each point's $d(x)^2$.

5: Pick new centroid from the remaining points using the weighted probabilities.

6: **end for**

Later, we will discuss two other approaches that are also useful for pro- ducing better-quality

(lower SSE) clusterings: using a variant of K-means that is less susceptible to initialization problems (bisecting K-means) and using postprocessing to "fix up" the set of clusters produced. K-means++ could be combined with either approach.

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 $O(I \times K \times m \times n)$, 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.

K-means: Additional Issues

Handling Empty Clusters

One of the problems with the basic K-means algorithm 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. (A K-means++ approach could be used as well.) Another approach is to choose the replacement centroid at random 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) are typically not as representative as they oth- erwise would be and thus, the SSE will be higher. 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 compres- sion, 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 9. 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 often want to eliminate small clusters because 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 terms *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 employ 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 with respect to their closest centroids.

Two strategies that decrease the number of clusters, while trying to mini- mize 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 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 because 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 can 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 5.2. On the negative side, updating centroids incrementally introduces an order dependency. In

other words, the clusters produced usually 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.

Bisecting K-means

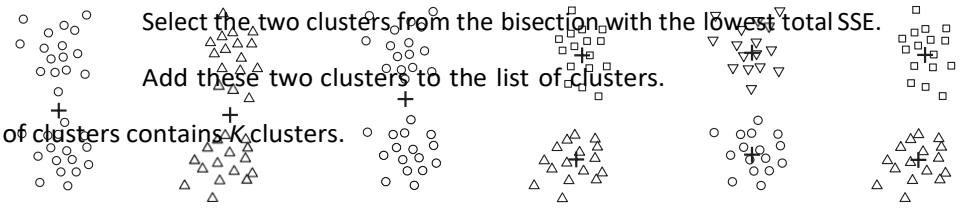
The bisecting K-means algorithm is a straightforward extension of the basic K-means algorithm that is based on a simple idea: to obtain *K* clusters, split the set of all points into two clusters, select one of these clusters to split, and so on, until *K* clusters have been produced. The details of bisecting K-means are given by Algorithm 3.

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.

Because we are using the K-means algorithm "locally," i.e., to bisect individual clusters, the final set of clusters does not represent a clustering that is a local minimum with respect to the total SSE. Thus, we often refine the resulting clusters by using their cluster centroids as the initial centroids for the standard K-means algorithm.

Algorithm 3 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** The list of clusters contains *K* clusters.



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

Figure 8. Bisecting K-means on the four clusters example.

Example 3 (Bisecting K-means and Initialization)**.** To illustrate that bi- secting K-means is less susceptible to initialization problems, we show, in Figure 8, how bisecting K-means

finds four clusters in the data set originally shown in Figure 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.

K-means and Different Types of Clusters

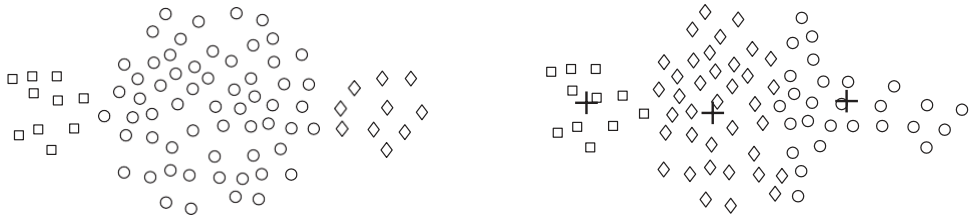
K-means and its variations have a number of limitations with respect to finding different types of clusters. In particular, K-means has difficulty detecting the "natural" clusters, when clusters have non-spherical shapes or widely different sizes or densities. This is illustrated by Figures 9, 10, and 11. In Figure 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 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 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 because 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 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.

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 suscepti- ble to initialization problems. K- means is not suitable for all types of data, 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.

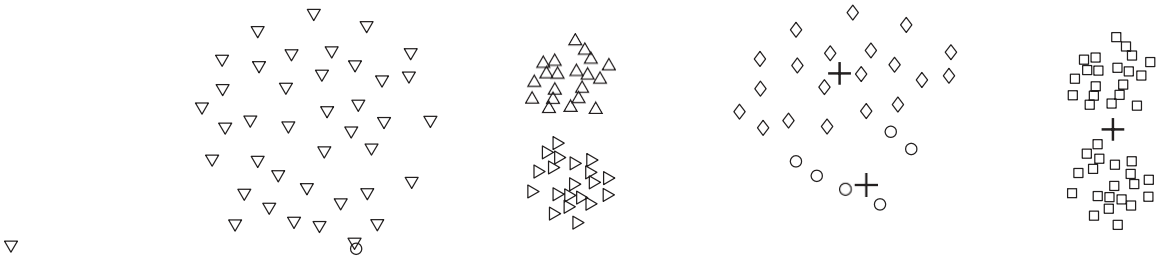
K-means as an Optimization Problem

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, can 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



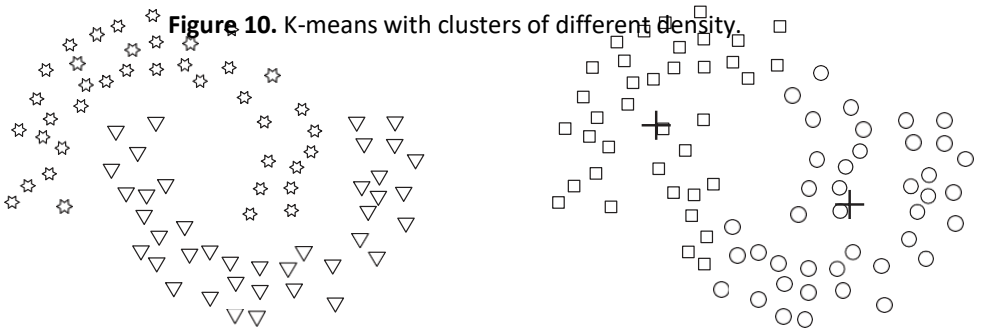
(a) Original points. (b) Three K-means clusters.

Figure 9. K-means with clusters of different size.



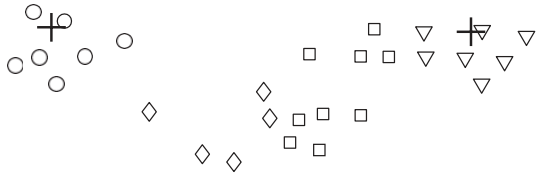
(a) Original points. (b) Three K-means clusters.

Figure 10. K-means with clusters of different density.

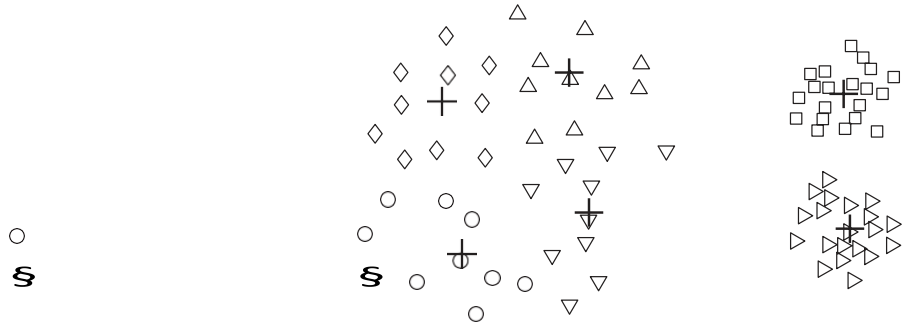


(a) Original points. (b) Two K-means clusters.

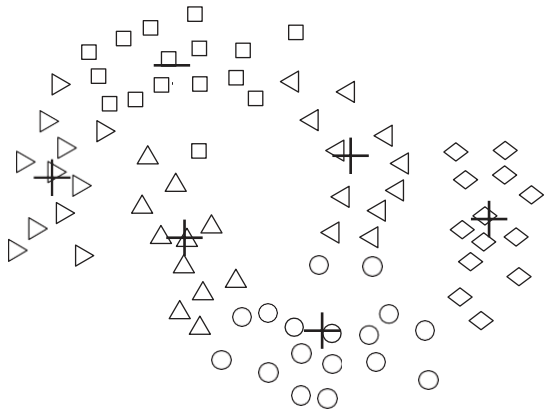
Figure 11. K-means with non-globular clusters.



(a) Unequal sizes.



(b) Unequal densities.



(c) Non-spherical shapes.

Figure 12. Using K-means to find clusters that are subclusters of the natural clusters.

Proble: 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 1, which we repeat here, specialized for one-dimensional data.

$$\text{SSE} = \sum_{i=1}^{K} \sum_{x \in C_i} (c_i - x)^2 \qquad (4)$$

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 1 for a complete list of notation. We can solve for the k^{th} centroid c_k, which minimizes Equation 4, by differentiating the SSE, setting it equal to 0, and solving, as indicated below.

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

1

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., $dist_{L_1} = |c_i - x|$.

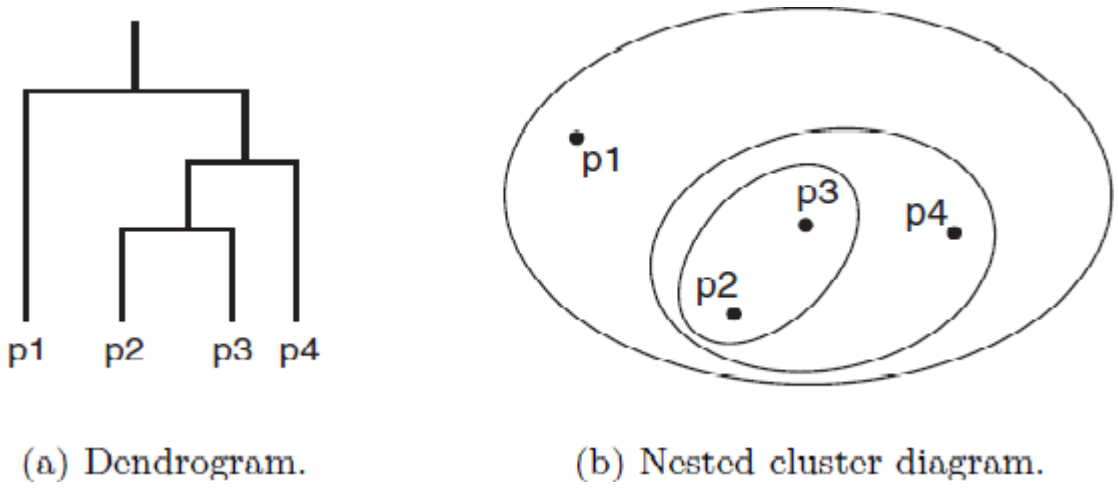
$$\mathrm{SAE} = \sum_{i=1}^{K} \sum_{x \in C_i} dist_{\mathrm{L}_1}(c_i, x) \qquad (5)$$

We can solve for the k^{th} centroid c_k, which minimizes equation 5, by differentiating the SAE, setting it equal to 0, and solving.

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

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

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.



(a) Dendrogram. (b) Nested cluster diagram.

Figure 13. A hierarchical clustering of four points shown as a dendrogram and as nested clusters.

Agglomerative Hierarchical Clustering

Hierarchical clustering techniques are a second important category of cluster- ing 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 8.4.2.

A hierarchical clustering is often displayed graphically using a tree-like diagram called a **dendrogram**, which displays both the cluster-subcluster relationships and the order in which the clusters were merged (agglomerative view) or split (divisive view). For sets of two-dimensional points, such as those that we will use as examples, a hierarchical clustering can also be graphically represented using a nested cluster diagram. Figure 13 shows an example of these two types of figures for a set of four two-dimensional points. These points were clustered using the single-link technique that is described in Section 3.2.

3.1 Basic Agglomerative Hierarchical Clustering Algorithm

Many agglomerative hierarchical clustering techniques are variations on a sin- gle approach: starting with individual points as clusters, successively merge the two closest clusters until only one cluster remains. This approach is expressed more formally in Algorithm 4.

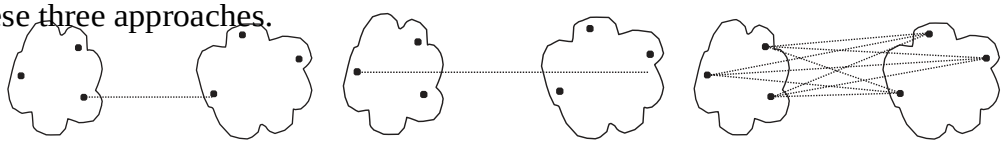
Algorithm 4 Basic agglomerative hierarchical clustering algorithm.

1: Compute the proximity matrix, if necessary.
2: **repeat**
3: Merge the closest two clusters.
4: Update the proximity matrix to reflect the proximity between the new cluster and the original clusters.
5: **until** Only one cluster remains.

Defining Proximity between Clusters

The key operation of Algorithm 4 is the computation of the proximity between two clusters, and it is the definition of cluster proximity that differentiates the various agglomerative hierarchical techniques that we will dis- cuss. Cluster proximity is typically defined with a particular type of cluster in mind—see Section 1.3. For example, many agglomerative hierarchical clustering techniques, such as MIN, MAX, and Group Average, come from a graph-based view of clusters. **MIN** defines cluster proximity as the proximity between the closest two points that are in different clusters, or using graph terms, the shortest edge between two nodes in different subsets of nodes. This yields contiguity-based clusters as shown in Figure 2(c). Alternatively, **MAX** takes the proximity between the farthest two points in different clusters to be the cluster proximity, or using graph terms, the longest edge between two nodes in different subsets of nodes. (If our proximities are distances, then the names, MIN and MAX, are short and suggestive. For similarities, however, where higher values indicate closer points, the names seem reversed. For that reason, we usually prefer to use the alternative names, **single link** and **complete link**, respectively.) Another graph-based approach, the **group average** technique, defines cluster proximity to be the average pairwise proximities (average length of edges) of all pairs of points from different clusters. Figure

14 illustrates these three approaches.



(a) MIN (single link). (b) MAX (complete link). (c) Group average.

Figure 14. Graph-based definitions of cluster proximity.

f, instead, we take a prototype-based view, in which each cluster is rep- resented by a centroid, different definitions of cluster proximity are more natural. When using centroids, the cluster proximity is commonly defined as the proximity between cluster centroids. An alternative technique, **Ward's** method, also assumes that a cluster is represented by its centroid, but it measures the proximity between two clusters in terms of the increase in the SSE that results from merging the two clusters. Like K-means, Ward's method attempts to minimize the sum of the squared distances of points from their cluster centroids.

Time and Space Complexity

The basic agglomerative hierarchical clustering algorithm just presented uses a proximity matrix. This requires the storage of $\frac{1}{2} m^2$ proximities (assuming the proximity matrix is symmetric) where m is the number of data points. The space needed to keep track of the clusters is proportional to the number of clusters, which is m 1, excluding singleton clusters. Hence, the total space complexity is $O(m^2)$.

The analysis of the basic agglomerative hierarchical clustering algorithm is also straightforward with respect to computational complexity. $O(m^2)$ time is required to compute the proximity matrix. After that step, there are $m - 1$ iterations involving steps 3 and 4 because there are m clusters at the start and two clusters are merged during each iteration. If performed as a linear search of the proximity matrix, then for the i^{th} iteration, Step 3 requires $O((m-i+1)^2)$ time, which is proportional to the current number of clusters squared. Step 4 requires $O(m - i + 1)$ time to update the proximity matrix after the merger of two clusters. (A cluster merger affects $O(m - i + 1)$ proximities for the techniques that we consider.) Without modification, this would yield a time complexity of $O(m^3)$. If the distances from each cluster to all other clusters are stored as a sorted list (or heap), it is possible to reduce the cost of finding the two closest clusters to $O(m - i + 1)$. However, because of the additional complexity of keeping data in a sorted list or heap, the overall time required for a hierarchical clustering based on Algorithm 4 is $O(m^2 \log m)$.

The space and time complexity of hierarchical clustering severely limits the size of data sets that can be processed. We discuss scalability approaches for clustering algorithms, including hierarchical clustering techniques, in Section 8.5. Note, however, that the bisecting K-means algorithm presented in Section 2.3 is a scalable algorithm that produces a hierarchical clustering.

Specific Techniques

Sample Data To illustrate the behavior of the various hierarchical clustering algorithms, we will use sample data that consists of six two-dimensional points, which are shown in Figure 15. The *x* and *y* coordinates of the points and the Euclidean distances between them are shown in Tables 3 and 4, respectively.

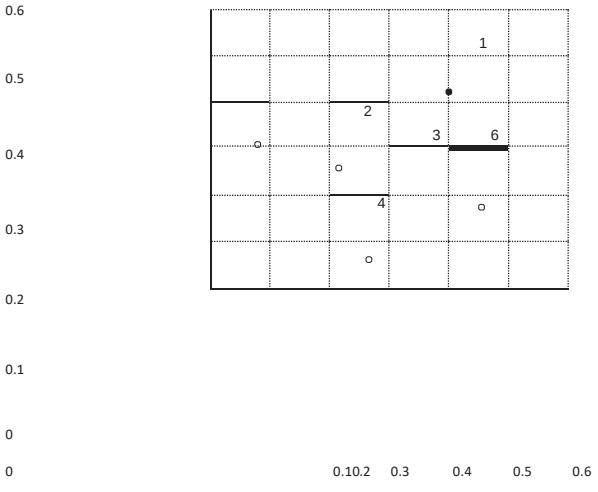


Figure 15. Set of six two-dimensional points.

Point	*x* Coordinate	*y* Coordinate
p1	0.4005	0.5306
p2	0.2148	0.3854
p3	0.3457	0.3156
p4	0.2652	0.1875
p5	0.0789	0.4139

Table 3. *xy*-coordinates of six points.

	p1	p2	p3	p4	p5	p6
p1	0.00	0.24	0.22	0.37	0.34	0.23
p2	0.24	0.00	0.15	0.20	0.14	0.25
p3	0.22	0.15	0.00	0.15	0.28	0.11
p4	0.37	0.20	0.15	0.00	0.29	0.22
p5	0.34	0.14	0.28	0.29	0.00	0.39
p6	0.23	0.25	0.11	0.22	0.39	0.00

Table 4. Euclidean distance matrix for six points.

Single Link or MIN

For the single link or MIN version of hierarchical clustering, the proximity of two clusters is defined as the minimum of the distance (maximum of the similarity) between any two points in the two different clusters. Using graph terminology, if you start with all points as singleton clusters and add links between points one at a time, shortest links first, then these single links combine the points into clusters. The single link technique is good at handling non-elliptical shapes, but is sensitive to noise and outliers.

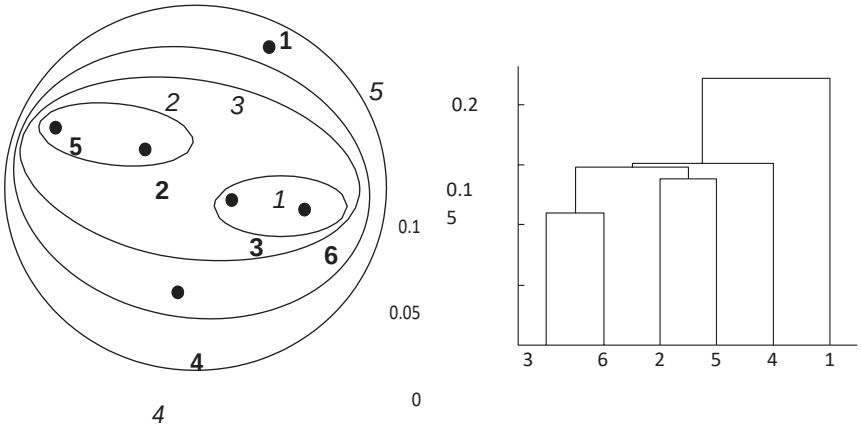
Example 4 (Single Link). Figure 16 shows the result of applying the single link technique to our example data set of six points. Figure 16(a) shows the nested clusters as a sequence of nested ellipses, where the numbers associated with the ellipses indicate the order of the clustering. Figure 16(b) shows the same information, but as a dendrogram. The height at which two clusters are merged in the dendrogram reflects the distance of the two clusters. For instance, from Table 4, we see that the distance between points 3 and 6 is 0.11, and that is the height at which they are joined into one cluster in the dendrogram. As another example, the distance between clusters {3, 6} and {2, 5} is given by

$$\begin{aligned} dist(\{3,6\},\{2,5\}) &= \min(dist(3,2), dist(6,2), dist(3,5), dist(6,5)) \\ &= \min(0.15, 0.25, 0.28, 0.39) \\ &= 0.15. \end{aligned}$$

Complete Link or MAX or CLIQUE

For the complete link or MAX version of hierarchical clustering, the proximity of two clusters is defined as the maximum of the distance (minimum of the similarity) between any two points in the two different clusters. Using graph terminology, if you start with all points as singleton clusters and add links between points one at a time, shortest links first, then a group of points is not a cluster until all the points in it are completely linked, i.e., form a *clique*. Complete link is less susceptible to noise and outliers, but it can break large clusters and it favors globular shapes.

Example 5 (Complete Link). Figure 17 shows the results of applying MAX to the sample data set of six points. As with single link, points 3 and 6



(a) Single link clustering.

(b) Single link dendrogram.

Figure 16. Single link clustering of the six points shown in Figure 15.

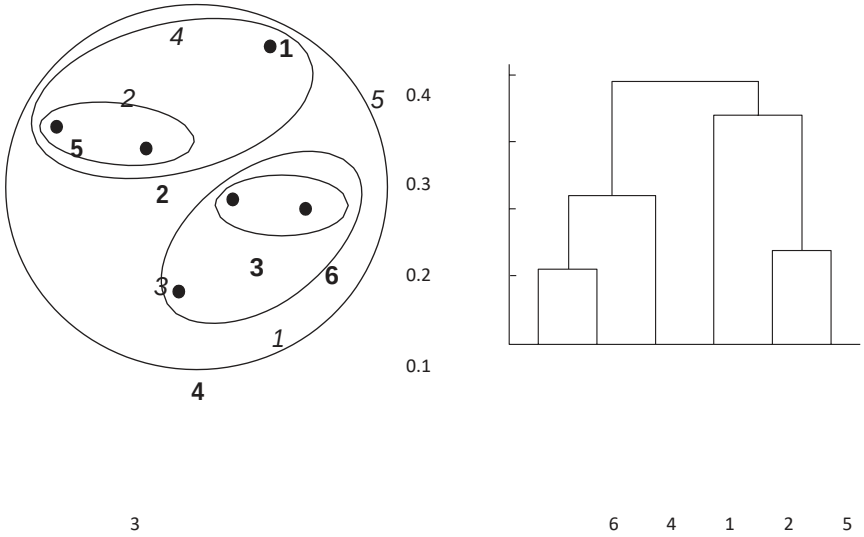
are merged first. However, $\{3, 6\}$ is merged with

because

$$
\begin{aligned}
dist(\{3,6\},\{4\}) &= \max(dist(3,4), dist(6 \\
&= \max(0.15, 0.22) \\
&= 0.22. \\
dist(\{3,6\},\{2,5\}) &= \max(dist(3,2), dist(6 \\
&= \max(0.15, 0.25, 0.28, (\\
&= 0.39. \\
dist(\{3,6\},\{1\}) &= \max(dist(3,1), dist(6 \\
&= \max(0.22, 0.23) \\
&= 0.23.
\end{aligned}
$$

Group Average

For the group average version of hierarchical clustering, the proximity of two clusters is defined as the average pairwise proximity among all pairs of points in the different clusters. This is an intermediate approach between the single and complete link approaches. Thus, for group average, the cluster



(a) Complete link clustering. (b) Complete link dendrogram.

Figure 17. Complete link clustering of the six points shown in Figure 15.

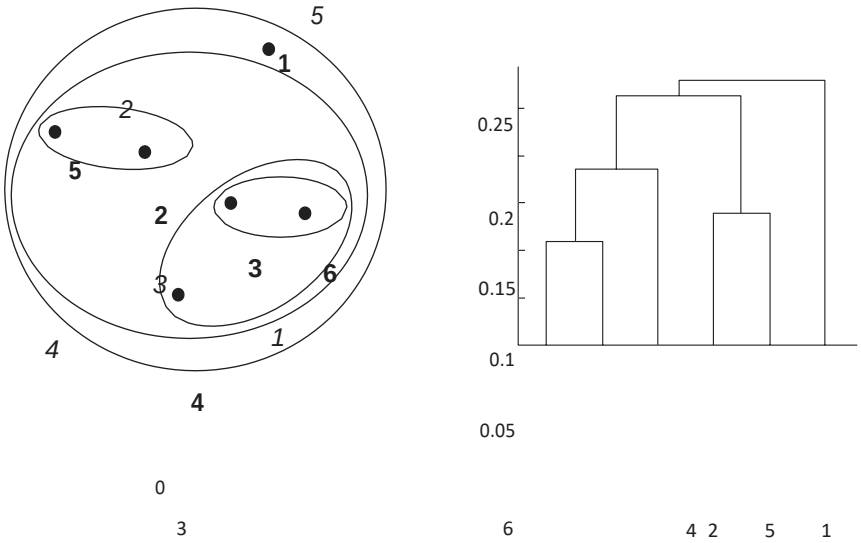
proximity *proximity*(C_i, C_j) of clusters C_i and C_j, which are of size m_i and m_j, respectively, is expressed by the following equation:

$$proximity(C_i, C_j) = \frac{\sum_{\mathbf{x} \in C_i, \mathbf{y} \in C_j} proximity(\mathbf{x}, \mathbf{y})}{m_i \times m_j}. \quad (6)$$

$$\begin{aligned} dist(\{3,6,4\},\{1\}) &= (0.22 + 0.37 + 0.23)/(3 \times 1) \\ &= 0.28 \\ dist(\{2,5\},\{1\}) &= (0.24 + 0.34)/(2 \times 1) \\ &= 0.29 \\ dist(\{3,6,4\},\{2,5\}) &= (0.15 + 0.28 + 0.25 + 0.39 + 0.20 + 0.29)/(3 \times 2) \\ &= 0.26 \end{aligned}$$

Example 6 (Group Average). Figure 18 shows the results of applying the group average approach to the sample data set of six points. To illustrate how group average works, we calculate the distance between some clusters.

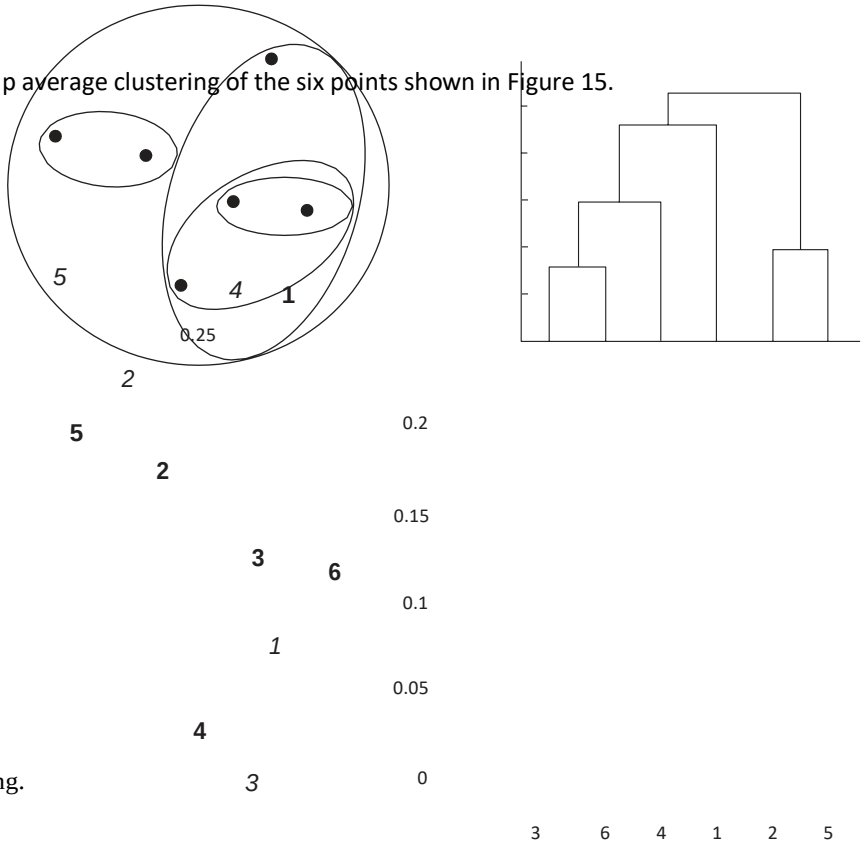
Because *dist*({3, 6, 4}, {2, 5}) is smaller than *dist*({3, 6, 4}, {1}) and *dist*({2, 5}, {1}), clusters {3, 6, 4} and {2, 5} are merged at the fourth stage.



Agglomerative Hierarchical Clustering

(a) Group average clustering. (b) Group average dendrogram.

Figure 18. Group average clustering of the six points shown in Figure 15.



(a) Ward's clustering.

(b) Ward's dendrogram.

Figure 19. Ward's clustering of the six points shown in Figure 15.

Ward's Method and Centroid Methods

For Ward's method, the proximity between two clusters is defined as the increase in the squared error that results when two clusters are merged. Thus, this method uses the same objective function as K-means clustering. While it might seem that this feature makes Ward's method somewhat distinct from other hierarchical techniques, it can be shown mathematically that Ward's method is very similar to the group average method when the proximity between two points is taken to be the square of the distance between them.

Example 7 (Ward's Method). Figure 19 shows the results of applying Ward's method to the sample data set of six points. The clustering that is produced is different from those produced by single link, complete link, and group average.

Centroid methods calculate the proximity between two clusters by calcu- lating the distance between the centroids of clusters. These techniques may seem similar to K-means, but as we have remarked, Ward's method is the correct hierarchical analog. Centroid methods also have a characteristic—often considered bad—that is not possessed by the other hierarchical clustering techniques that we have discussed: the possibility of inversions. Specifically, two clusters that are merged can be more similar (less distant) than the pair of clusters that were merged in a previous step. For the other methods, the distance between merged clusters monotonically increases (or is, at worst, non-increasing) as we proceed from singleton clusters to one all-inclusive cluster.

The Lance-Williams Formula for Cluster Proximity

Any of the cluster proximities that we have discussed in this section can be viewed as a choice of different parameters (in the Lance-Williams formula shown below in Equation 7) for the proximity between clusters Q and R, where R is formed by merging clusters A and B. In this equation, $p(.,.)$ is a proximity function, while m_A, m_B, and m_Q are the number of points in clusters A, B, and Q, respectively. In other words, after we merge clusters A and B to form cluster R, the proximity of the new cluster, R, to an existing cluster, Q, is a linear function of the proximities of Q with respect to the original clusters A and B. Table 5 shows the values of these coefficients for the techniques that we have discussed.

$$p(R, Q) = \alpha_A p(A, Q) + \alpha_B p(B, Q) + \beta p(A, B) + \gamma |p(A, Q) - p(B, Q)| \quad (7)$$

Any hierarchical clustering technique that can be expressed using the Lance-Williams formula does not need to keep the original data points. In- stead, the proximity matrix is updated as clustering occurs. While a general formula is appealing, especially for implementation, it is easier to understand the different hierarchical methods by looking directly at the definition of cluster proximity that each method uses.

Table 5. Table of Lance-Williams coefficients for common hierarchical clustering approaches.

Clustering Method	α_A	α_B	β	γ
Single Link	$1/2$	$1/2$	0	$-1/2$
Complete Link	$1/2$	$1/2$	0	$1/2$
Group Average	$\frac{m_A}{m_A+m_B}$	$\frac{m_B}{m_A+m_B}$	0	0
Centroid	$\frac{m_A}{m_A+m_B}$	$\frac{m_B}{m_A+m_B}$	$\frac{-m_A m_B}{(m_A+m_B)^2}$	0
Ward's	$\frac{m_A+m_Q}{m_A+m_B+m_Q}$	$\frac{m_B+m_Q}{m_A+m_B+m_Q}$	$\frac{-m_Q}{m_A+m_B+m_Q}$	0

Key Issues in Hierarchical Clustering

Lack of a Global Objective Function We previously mentioned that agglomerative hierarchical clustering cannot be viewed as globally optimizing an objective function. Instead, agglomerative hierarchical clustering techniques use various criteria to decide locally, at each step, which clusters should be merged (or split for divisive approaches). This approach yields clustering algorithms that avoid the difficulty of attempting to solve a hard combinatorial optimization problem. (It can be shown that the general clustering problem for an objective function such as "minimize SSE" is computationally infeasible.) Furthermore, such approaches do not have difficulties in choosing initial points. Nonetheless, the time complexity of $O(m^2 \log m)$ and the space complexity of $O(m^2)$ are prohibitive in many cases.

Ability to Handle Different Cluster Sizes

One aspect of agglomerative hierarchical clustering that we have not yet discussed is how to treat the relative sizes of the pairs of clusters that are merged. (This discussion applies only to cluster proximity schemes that involve sums, such as centroid, Ward's, and group average.) There are two approaches: weighted, which treats all clusters equally, and unweighted, which takes the number of points in each cluster into account. Note that the terminology of weighted or unweighted refers to the data points, not the clusters. In other words, treating clusters of unequal size equally—the weighted approach—gives different weights to the points in different clusters, while taking the cluster size into account—the unweighted approach— gives points in different clusters the same weight. We will illustrate this using the group average technique discussed in Section 3.2, which is the unweighted version of the group average technique. In the clustering literature, the full name of this approach is the Unweighted Pair Group Method using Arithmetic averages (UPGMA). In Table 5, which gives the formula for updating cluster similarity, the coefficients for UPGMA involve the size, m_A and m_B of each of the clusters, A and B that were merged: $\alpha_A = m_A/(m_A + m_B)$, $\alpha_B = m_B/(m_A + m_B)$, $\beta = 0$, $\gamma = 0$. For the weighted version of group average—known as WPGMA—the coefficients are constants that are independent of the cluster sizes: $\alpha_A = 1/2$, $\alpha_B = 1/2$, $\beta = 0$, $\gamma = 0$. In general, unweighted approaches are preferred unless there is reason to believe that individual points should have different weights; e.g., perhaps classes of objects have been unevenly sampled.

Merging Decisions Are Final

Agglomerative hierarchical clustering algorithms tend to make good local decisions about combining two clusters because they can use information about the pairwise similarity of all points. However, once a decision is made to merge two clusters, it cannot be undone at a later time. This approach prevents a local optimization criterion from becoming a global optimization criterion. For example, although the "minimize squared error" criterion from K- means is used in deciding which clusters to merge in Ward's method, the clusters at each level do not represent local minima with respect to the total SSE. Indeed, the clusters are not even stable, in the sense that a point in one cluster can be closer to the centroid of some other cluster than it is to the centroid of its current cluster. Nonetheless, Ward's method is often used as a robust method of initializing a K-means clustering, indicating that a local "minimize squared error" objective function does have a connection to a global "minimize squared error" objective function.

There are some techniques that attempt to overcome the limitation that merges are final. One approach attempts to fix up the hierarchical clustering by moving branches of the tree around so as to improve a global objective function. Another approach uses a partitional clustering technique such as K- means to create many small clusters, and then performs hierarchical clustering using these small clusters as the starting point.

Outliers

Outliers pose the most serious problems for Ward's method and centroid-based hierarchical clustering approaches because they increase SSE and distort cen- troids. For clustering approaches, such as single link, complete link, and group average, outliers are potentially less problematic. As hierarchical clustering proceeds for these algorithms, outliers or small groups of outliers tend to form singleton or small clusters that do not merge with any other clusters until much later in the merging process. By discarding singleton or small clusters that are not merging with other clusters, outliers can be removed.

Strengths and Weaknesses The strengths and weaknesses of specific agglomerative hierarchical clustering algorithms were discussed above. More generally, such algorithms are typically used because the underlying application, e.g., creation of a taxonomy, requires a hierarchy. Also, some studies have suggested that these algorithms can produce better-quality clusters. However, agglomerative hierarchical clustering algorithms are expensive in terms of their computational and storage require- ments. The fact that all merges are final can also cause trouble for noisy, high-dimensional data, such as document data. In turn, these two problems can be addressed to some degree by first partially clustering the data using another technique, such as K-means.

DBSCAN

Density-based clustering locates regions of high density that are separated from one another by regions of low density. DBSCAN is a simple and effective density-based clustering algorithm that illustrates a number of important concepts that are important for any density-based clustering approach. In this section, we focus solely on DBSCAN after first considering the key notion of density. Other algorithms for finding density-based clusters are described in the next chapter.

Traditional Density: Center-Based Approach

Although there are not as many approaches for defining density as there are for defining similarity, there are several distinct methods. In this section we discuss the center-based approach on which DBSCAN is based. Other definitions of density will be presented in Chapter 8. In the center-based approach, density is estimated for a particular point in the data set by counting the number of points within a specified radius, *Eps*, of that point. This includes the point itself. This technique is graphically illustrated by Figure 20. The number of points within a radius of *Eps* of point *A* is 7, including *A* itself. This method is simple to implement, but the density of any point will depend on the specified radius. For instance, if the radius is large enough,

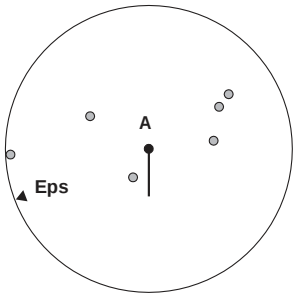


Figure 20. Center-based density.

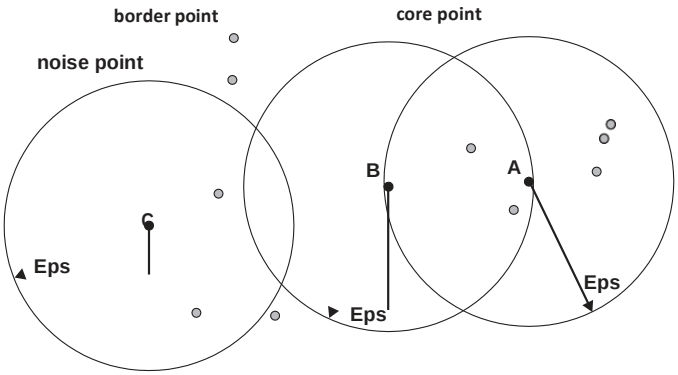


Figure 21. Core, border, and noise points.

then all points will have a density of *m*, the number of points in the data set. Likewise, if the radius is too small, then all points will have a density of 1. An approach for deciding on the appropriate radius for low-dimensional data is given in the next section in the context of our discussion of DBSCAN.

Classiftcation of Points According to Center-Based Density

The center-based approach to density allows us to classify a point as being (1) in the interior of a dense region (a core point), (2) on the edge of a dense region (a border point), or (3) in a sparsely occupied region (a noise or background point). Figure 21 graphically illustrates the concepts of core, border, and noise points using a collection of two-dimensional points. The following text provides a more precise description.

Core points: These points are in the interior of a density-based cluster. A point is a core point if there are at least *MinPts* within a distance of *Eps*, where *MinPts* and *Eps* are user-specified parameters. In Figure 21, point *A* is a core point for the radius (*Eps*) if $MinPts \geq 7$.

Border points: A border point is not a core point, but falls within the neighborhood of a core point. In Figure 21, point *B* is a border point. A border point can fall within the neighborhoods of several core points.

Noise points: A noise point is any point that is neither a core point nor a border point. In Figure 21, point *C* is a noise point.

The DBSCAN Algorithm

Given the previous definitions of core points, border points, and noise points, the DBSCAN algorithm can be informally described as follows. Any two core points that are close enough—within a distance *Eps* of one another—are put in the same cluster. Likewise, any border point that is close enough to a core point is put in the same cluster as the core point. (Ties need to be resolved if a border point is close to core points from different clusters.) Noise points are discarded. The formal details are given in Algorithm 5. This algorithm uses the same concepts and finds the same clusters as the original DBSCAN, but is optimized for simplicity, not efficiency.

Algorithm 5 DBSCAN algorithm.

1: Label all points as core, border, or noise points.

2: Eliminate noise points.

3: Put an edge between all core points within a distance *Eps* of each other.

4: Make each group of connected core points into a separate cluster.

5: Assign each border point to one of the clusters of its associated core points.

Time and Space Complexity

The basic time complexity of the DBSCAN algorithm is $O(m \times$ time to find points in the *Eps*-neighborhood), where *m* is the number of points. In the worst case, this complexity is $O(m^2)$. However, in low-dimensional spaces (especially 2D space), data structures such as kd- trees allow efficient retrieval of all points within a given distance of a specified point, and the time complexity can be as low as $O(m \log m)$ in the average case. The space requirement of DBSCAN, even for high-dimensional data, is $O(m)$ because it is necessary to keep only a small amount of data for each point, i.e., the cluster label and the identification of each point as a core, border, or noise point.

Selection of DBSCAN Parameters

There is, of course, the issue of how to determine the parameters *Eps* and *MinPts*. The basic approach is to look at the behavior of the distance from a point to its k^{th} nearest neighbor, which we will call the *k*-dist. For points that belong to some cluster, the value of *k*-dist will be small if *k* is not larger than the cluster size. Note that there will be some variation, depending on the density of the cluster and the random distribution of points, but on average, the range of variation will not be huge if the cluster densities are not radically different.

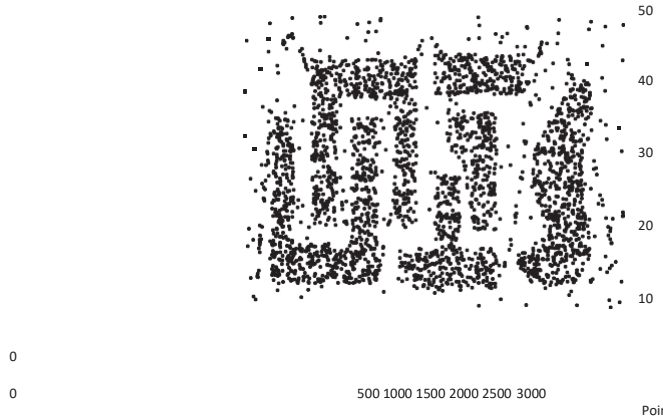


Figure 22. Sample data.

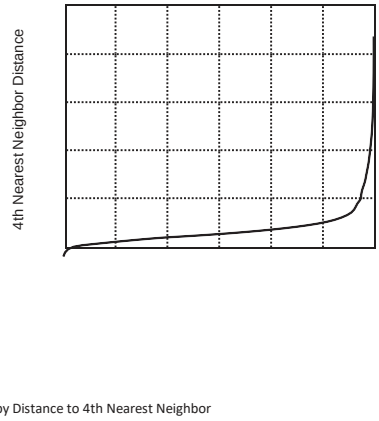


Figure 23. K-dist plot for sample data.

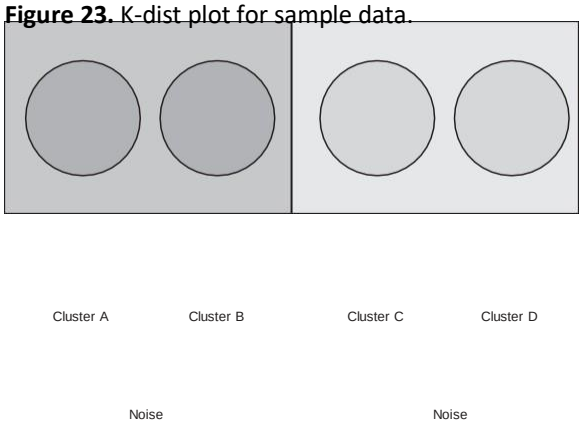


Figure 24. Four clusters embedded in noise.

However, for points that are not in a cluster, such as noise points, the *k*-dist will be relatively large. Therefore, if we compute the *k*-dist for all the data points for some *k*, sort them in increasing order, and then plot the sorted values, we expect to see a sharp change at the value of *k*-dist that corresponds to a suitable value of *Eps*. If we select this distance as the *Eps* parameter and take the value of *k* as the *MinPts* parameter, then points for which *k*-dist is less than *Eps* will be labeled as core points, while other points will be labeled as noise or border points.

Figure 22 shows a sample data set, while the *k*-dist graph for the data is given in Figure 23. The value of *Eps* that is determined in this way depends on *k*, but does not change dramatically as *k* changes. If the value of *k* is too small, then even a small number of closely spaced points that are noise or outliers will be incorrectly labeled as clusters. If the value of *k* is too large, then small clusters (of size less than *k*) are likely to be labeled as noise. The original DBSCAN algorithm used a value of $k = 4$, which appears to be a reasonable value for most two-dimensional data sets.

Clusters of Varying Density

DBSCAN can have trouble with density if the density of clusters varies widely. Consider Figure 24, which shows four clusters embedded in noise. The den- sity of the clusters and noise regions is indicated by their darkness. The noise around the pair of denser clusters, *A* and *B*, has the same density as clusters *C* and *D*. For a fixed *MinPts*, if the *Eps* threshold is chosen so that DBSCAN finds *C* and *D* as distinct clusters, with the points surrounding them as noise, then *A* and *B* and the points surrounding them will become a single cluster. If the *Eps* threshold is set so that DBSCAN finds *A* and *B* as separate clusters, and the points surrounding them are marked as noise, then *C*, *D*, and the points surrounding them will also be marked as noise.

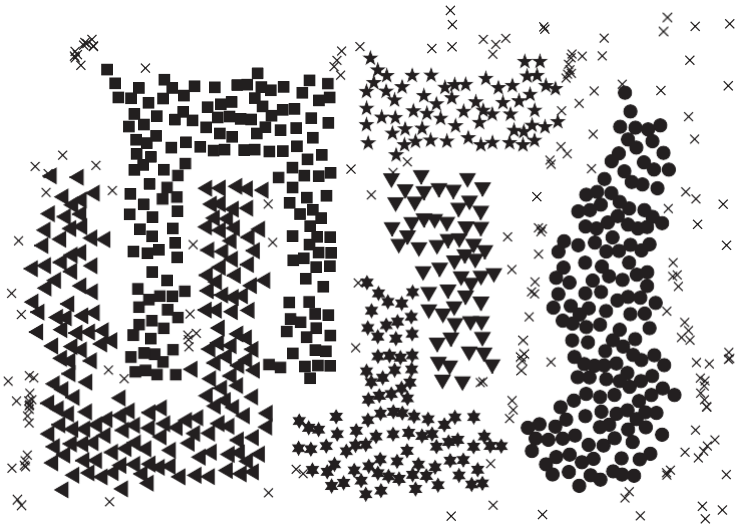
An Example

To illustrate the use of DBSCAN, we show the clusters that it finds in the relatively complicated two-dimensional data set shown in Figure 22. This data set consists of 3000 two-dimensional points. The *Eps* threshold for this data was found by plotting the sorted distances of the fourth nearest neighbor of each point (Figure 23) and identifying the value at which there is a sharp increase. We selected $Eps = 10$, which corresponds to the knee of the curve. The clusters found by DBSCAN using these parameters, i.e., $MinPts = 4$ and $Eps = 10$, are shown in Figure 25(a). The core points, border points, and noise points are displayed in Figure 25(b).

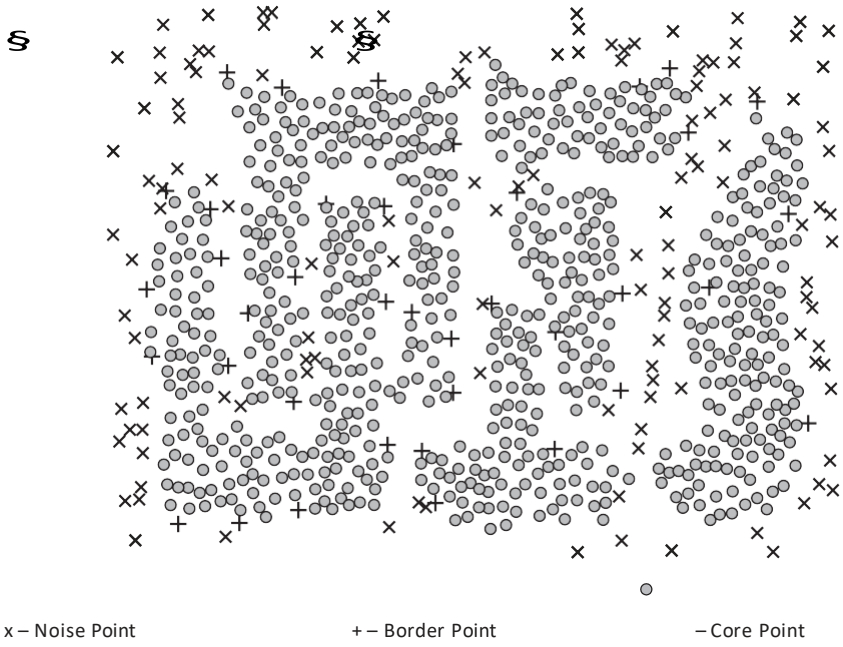
Strengths and Weaknesses

Because DBSCAN uses a density-based definition of a cluster, it is relatively resistant to noise and can handle clusters of arbitrary shapes and sizes. Thus, DBSCAN can find many clusters that could not be found using K-means, such as those in Figure 22. As indicated previously, however, DBSCAN has trouble when the clusters have widely varying densities. It also has trouble with high-dimensional data because density is more difficult to define for such data. Finally, DBSCAN can be expensive when the computation of nearest neighbors requires computing all pairwise proximities, as is usually the case for high-dimensional data.

Copy protected with Online-PDF-No-Copy.com



(a) Clusters found by DBSCAN.



(b) Core, border, and noise points.

Figure 25. DBSCAN clustering of 3000 two-dimensional points.

Copy protected with Online-PDF-No-Copy.com