Chapter 7

Clustering

Clustering is the process of examining a collection of "points," and grouping the points into "clusters" according to some distance measure. The goal is that points in the same cluster have a small distance from one another, while points in different clusters are at a large distance from one another. A suggestion of what clusters might look like was seen in Fig. 1.1. However, there the intent was that there were three clusters around three different road intersections, but two of the clusters blended into one another because they were not sufficiently separated.

Our goal in this chapter is to offer methods for discovering clusters in data. We are particularly interested in situations where the data is very large, and/or where the space either is high-dimensional, or the space is not Euclidean at all. We shall therefore discuss several algorithms that assume the data does not fit in main memory. However, we begin with the basics: the two general approaches to clustering and the methods for dealing with clusters in a non-Euclidean space.

7.1 Introduction to Clustering Techniques

We begin by reviewing the notions of distance measures and spaces. The two major approaches to clustering – hierarchical and point-assignment – are defined. We then turn to a discussion of the "curse of dimensionality," which makes clustering in high-dimensional spaces difficult, but also, as we shall see, enables some simplifications if used correctly in a clustering algorithm.

7.1.1 Points, Spaces, and Distances

A dataset suitable for clustering is a collection of *points*, which are objects belonging to some *space*. In its most general sense, a space is just a universal set of points, from which the points in the dataset are drawn. However, we should be mindful of the common case of a Euclidean space (see Section 3.5.2),

which has a number of important properties useful for clustering. In particular, a Euclidean space's points are vectors of real numbers. The length of the vector is the number of dimensions of the space. The components of the vector are commonly called *coordinates* of the represented points.

All spaces for which we can perform a clustering have a distance measure, giving a distance between any two points in the space. We introduced distances in Section 3.5. The common Euclidean distance (square root of the sums of the squares of the differences between the coordinates of the points in each dimension) serves for all Euclidean spaces, although we also mentioned some other options for distance measures in Euclidean spaces, including the Manhattan distance (sum of the magnitudes of the differences in each dimension) and the L_∞-distance (maximum magnitude of the difference in any dimension).

Figure 7.1: Heights and weights of dogs taken from three varieties

Example 7.1: Classical applications of clustering often involve low-dimensional Euclidean spaces. For example, Fig. 7.1 shows height and weight measurements of dogs of several varieties. Without knowing which dog is of which variety, we can see just by looking at the diagram that the dogs fall into three clusters, and those clusters happen to correspond to three varieties. With small amounts of data, any clustering algorithm will establish the correct clusters, and simply plotting the points and "eyeballing" the plot will suffice as well. □

However, modern clustering problems are not so simple. They may involve Euclidean spaces of very high dimension or spaces that are not Euclidean at all. For example, it is challenging to cluster documents by their topic, based on the occurrence of common, unusual words in the documents. It is challenging to cluster moviegoers by the type or types of movies they like.

We also considered in Section 3.5 distance measures for non-Euclidean spaces. These include the Jaccard distance, cosine distance, Hamming distance,

and edit distance. Recall that the requirements for a function on pairs of points to be a distance measure are that

1. Distances are always nonnegative, and only the distance between a point and itself is 0.
2. Distance is symmetric; it doesn't matter in which order you consider the points when computing their distance.
3. Distance measures obey the triangle inequality; the distance from x to y to z is never less than the distance going from x to z directly.

7.1.2 Clustering Strategies

We can divide (cluster!) clustering algorithms into two groups that follow two fundamentally different strategies.

1. *Hierarchical* or *agglomerative* algorithms start with each point in its own cluster. Clusters are combined based on their "closeness," using one of many possible definitions of "close." Combination stops when further combination leads to clusters that are undesirable for one of several reasons. For example, we may stop when we have a predetermined number of clusters, or we may use a measure of compactness for clusters, and refuse to construct a cluster by combining two smaller clusters if the resulting cluster has points that are spread out over too large a region.
2. The other class of algorithms involve *point assignment*. Points are considered in some order, and each one is assigned to the cluster into which it best fits. This process is normally preceded by a short phase in which initial clusters are estimated. Variations allow occasional combining or splitting of clusters, or may allow points to be unassigned if they are *outliers* (points too far from any of the current clusters).

Algorithms for clustering can also be distinguished by:

(a) Whether the algorithm assumes a Euclidean space, or whether the algorithm works for an arbitrary distance measure. We shall see that a key distinction is that in a Euclidean space it is possible to summarize a collection of points by their *centroid* – the average of the points. In a non-Euclidean space, there is no notion of a centroid, and we are forced to develop another way to summarize clusters.

(b) Whether the algorithm assumes that the data is small enough to fit in main memory, or whether data must reside in secondary memory, primarily. Algorithms for large amounts of data often must take shortcuts, since it is infeasible to look at all pairs of points, for example. It is also necessary to summarize clusters in main memory, since we cannot hold all the points of all the clusters in main memory at the same time.

7.1.3 The Curse of Dimensionality

High-dimensional Euclidean spaces have a number of unintuitive properties that are sometimes referred to as the "curse of dimensionality." Non-Euclidean spaces usually share these anomalies as well. One manifestation of the "curse" is that in high dimensions, almost all pairs of points are equally far away from one another. Another manifestation is that almost any two vectors are almost orthogonal. We shall explore each of these in turn.

The Distribution of Distances in a High-Dimensional Space

Let us consider a d-dimensional Euclidean space. Suppose we choose n random points in the unit cube, i.e., points $[x_1, x_2, \ldots, x_d]$, where each x_i is in the range 0 to 1. If $d = 1$, we are placing random points on a line of length 1. We expect that some pairs of points will be very close, e.g., consecutive points on the line. We also expect that some points will be very far away – those at or near opposite ends of the line. The average distance between a pair of points is $1/3$.[1]

Suppose that d is very large. The Euclidean distance between two random points $[x_1, x_2, \ldots, x_d]$ and $[y_1, y_2, \ldots, y_d]$ is

$$\sqrt{\sum_{i=1}^{d}(x_i - y_i)^2}$$

Here, each x_i and y_i is a random variable chosen uniformly in the range 0 to 1. Since d is large, we can expect that for some i, $|x_i - y_i|$ will be close to 1. That puts a lower bound of 1 on the distance between almost any two random points. In fact, a more careful argument can put a stronger lower bound on the distance between all but a vanishingly small fraction of the pairs of points. However, the maximum distance between two points is $\sqrt{d}$, and one can argue that all but a vanishingly small fraction of the pairs do not have a distance close to this upper limit. In fact, almost all points will have a distance close to the average distance.

If there are essentially no pairs of points that are close, it is hard to build clusters at all. There is little justification for grouping one pair of points and not another. Of course, the data may not be random, and there may be useful clusters, even in very high-dimensional spaces. However, the argument about random data suggests that it will be hard to find these clusters among so many pairs that are all at approximately the same distance.

Angles Between Vectors

Suppose again that we have three random points A, B, and C in a d-dimensional space, where d is large. Here, we do not assume points are in the unit cube;

[1] You can prove this fact by evaluating a double integral, but we shall not do the math here, as it is not central to the discussion.

they can be anywhere in the space. What is angle ABC? We may assume that A is the point $[x_1, x_2, \ldots, x_d]$ and C is the point $[y_1, y_2, \ldots, y_d]$, while B is the origin. Recall from Section 3.5.4 that the cosine of the angle ABC is the dot product of A and C divided by the product of the lengths of the vectors A and C. That is, the cosine is

$$\frac{\sum_{i=1}^{d} x_i y_i}{\sqrt{\sum_{i=1}^{d} x_i^2}\sqrt{\sum_{i=1}^{d} y_i^2}}$$

As d grows, the denominator grows linearly in d, but the numerator is a sum of random values, which are as likely to be positive as negative. Thus, the expected value of the numerator is 0, and as d grows, its standard deviation grows only as $\sqrt{d}$. Thus, for large d, the cosine of the angle between any two vectors is almost certain to be close to 0, which means the angle is close to 90 degrees.

An important consequence of random vectors being orthogonal is that if we have three random points A, B, and C, and we know the distance from A to B is d_1, while the distance from B to C is d_2, we can assume the distance from A to C is approximately $\sqrt{d_1^2 + d_2^2}$. That rule does not hold, even approximately, if the number of dimensions is small. As an extreme case, if $d = 1$, then the distance from A to C would necessarily be $d_1 + d_2$ if A and C were on opposite sides of B, or $|d_1 - d_2|$ if they were on the same side.

7.1.4 Exercises for Section 7.1

! **Exercise 7.1.1:** Prove that if you choose two points uniformly and independently on a line of length 1, then the expected distance between the points is 1/3.

!! **Exercise 7.1.2:** If you choose two points uniformly in the unit square, what is their expected Euclidean distance?

! **Exercise 7.1.3:** Suppose we have a d-dimensional Euclidean space. Consider vectors whose components are only $+1$ or -1 in each dimension. Note that each vector has length $\sqrt{d}$, so the product of their lengths (denominator in the formula for the cosine of the angle between them) is d. If we chose each component independently, and a component is as likely to be $+1$ as -1, what is the distribution of the value of the numerator of the formula (i.e., the sum of the products of the corresponding components from each vector)? What can you say about the expected value of the cosine of the angle between the vectors, as d grows large?

7.2 Hierarchical Clustering

We begin by considering hierarchical clustering in a Euclidean space. This algorithm can only be used for relatively small datasets, but even so, there

are some efficiencies we can make by careful implementation. When the space is non-Euclidean, there are additional problems associated with hierarchical clustering. We therefore consider "clustroids" and the way we can represent a cluster when there is no centroid or average point in a cluster.

7.2.1 Hierarchical Clustering in a Euclidean Space

Any hierarchical clustering algorithm works as follows. We begin with every point in its own cluster. As time goes on, larger clusters will be constructed by combining two smaller clusters, and we have to decide in advance:

1. How will clusters be represented?
2. How will we choose which two clusters to merge?
3. When will we stop combining clusters?

Once we have answers to these questions, the algorithm can be described succinctly as:

```
WHILE it is not time to stop DO
    pick the best two clusters to merge;
    combine those two clusters into one cluster;
END;
```

To begin, we shall assume the space is Euclidean. That allows us to represent a cluster by its centroid or average of the points in the cluster. Note that in a cluster of one point, that point is the centroid, so we can initialize the clusters straightforwardly. We can then use the merging rule that the distance between any two clusters is the Euclidean distance between their centroids, and we should pick the two clusters at the shortest distance. Other ways to define intercluster distance are possible, and we can also pick the best pair of clusters on a basis other than their distance. We shall discuss some options in Section 7.2.3.

Example 7.2: Let us see how the basic hierarchical clustering would work on the data of Fig. 7.2. These points live in a 2-dimensional Euclidean space, and each point is named by its (x, y) coordinates. Initially, each point is in a cluster by itself and is the centroid of that cluster. Among all the pairs of points, there are two pairs that are closest: (10,5) and (11,4) or (11,4) and (12,3). Each is at distance $\sqrt{2}$. Let us break ties arbitrarily and decide to combine (11,4) with (12,3). The result is shown in Fig. 7.3, including the centroid of the new cluster, which is at (11.5, 3.5).

You might think that (10,5) gets combined with the new cluster next, since it is so close to (11,4). But our distance rule requires us to compare only cluster centroids, and the distance from (10,5) to the centroid of the new cluster is $1.5\sqrt{2}$, which is slightly greater than 2. Thus, now the two closest clusters are

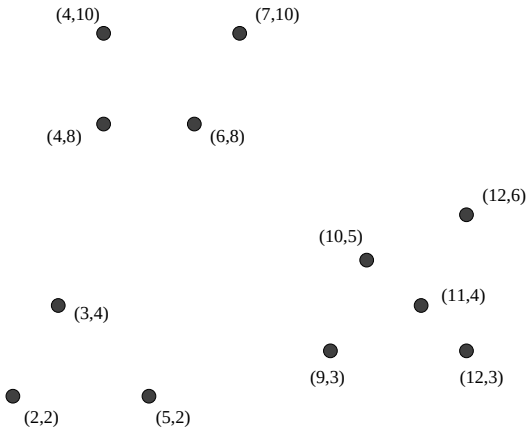

Figure 7.2: Twelve points to be clustered hierarchically

those of the points (4,8) and (4,10). We combine them into one cluster with centroid (4,9).

At this point, the two closest centroids are (10,5) and (11.5, 3.5), so we combine these two clusters. The result is a cluster of three points (10,5), (11,4), and (12,3). The centroid of this cluster is (11,4), which happens to be one of the points of the cluster, but that situation is coincidental. The state of the clusters is shown in Fig. 7.4.

Now, there are several pairs of centroids that are at distance $\sqrt{5}$, and these are the closest centroids. We show in Fig. 7.5 the result of picking three of these:

1. (6,8) is combined with the cluster of two elements having centroid (4,9).
2. (2,2) is combined with (3,4).
3. (9,3) is combined with the cluster of three elements having centroid (11,4).

We can proceed to combine clusters further. We shall discuss alternative stopping rules next. □

There are several approaches we might use to stopping the clustering process.

1. We could be told, or have a belief, about how many clusters there are in the data. For example, if we are told that the data about dogs is taken from Chihuahuas, Dachshunds, and Beagles, then we know to stop when there are three clusters left.
2. We could stop combining when at some point the best combination of existing clusters produces a cluster that is inadequate. We shall discuss

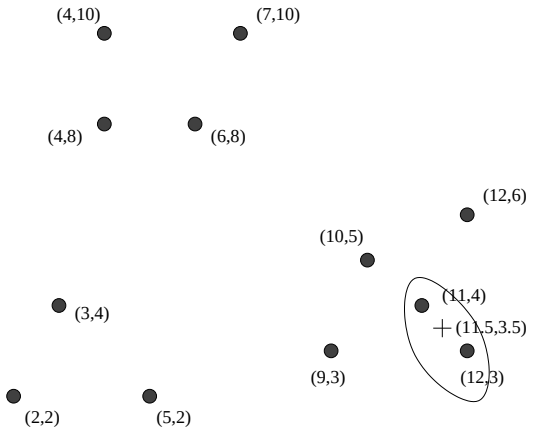

Figure 7.3: Combining the first two points into a cluster

various tests for the adequacy of a cluster in Section 7.2.3, but for an example, we could insist that any cluster have an average distance between the centroid and its points no greater than some limit. This approach is only sensible if we have a reason to believe that no cluster extends over too much of the space.

3. We could continue clustering until there is only one cluster. However, it is meaningless to return a single cluster consisting of all the points. Rather, we return the tree representing the way in which all the points were combined. This form of answer makes good sense in some applications, such as one in which the points are genomes of different species, and the distance measure reflects the difference in the genome.[2] Then, the tree represents the evolution of these species, that is, the likely order in which two species branched from a common ancestor.

Example 7.3: If we complete the clustering of the data of Fig. 7.2, the tree describing how clusters were grouped is the tree shown in Fig. 7.6. □

7.2.2 Efficiency of Hierarchical Clustering

The basic algorithm for hierarchical clustering is not very efficient. At each step, we must compute the distances between each pair of clusters, in order to find the best merger. The initial step takes $O(n^2)$ time, but subsequent steps take time proportional to $(n-1)^2, (n-2)^2, \ldots$. The sum of squares up to n is $O(n^3)$, so this algorithm is cubic. Thus, it cannot be run except for fairly small numbers of points.

[2]This space would not be Euclidean, of course, but the principles regarding hierarchical clustering carry over, with some modifications, to non-Euclidean clustering.

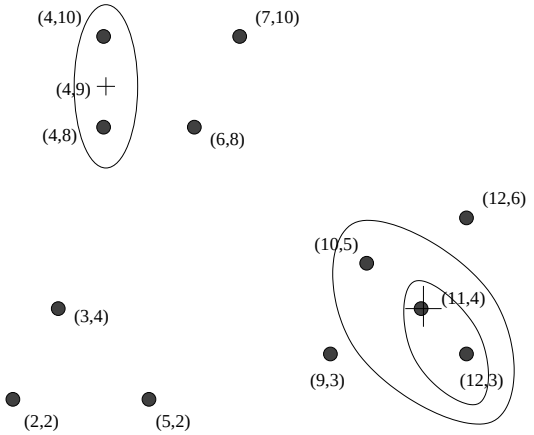

Figure 7.4: Clustering after two additional steps

However, there is a somewhat more efficient implementation of which we should be aware.

1. We start, as we must, by computing the distances between all pairs of points, and this step is $O(n^2)$.

2. Form the pairs and their distances into a priority queue, so we can always find the smallest distance in one step. This operation is also $O(n^2)$.

3. When we decide to merge two clusters C and D, we remove all entries in the priority queue involving one of these two clusters; that requires work $O(n \log n)$ since there are at most $2n$ deletions to be performed, and priority-queue deletion can be performed in $O(\log n)$ time.

4. We then compute all the distances between the new cluster and the remaining clusters. This work is also $O(n \log n)$, as there are at most n entries to be inserted into the priority queue, and insertion into a priority queue can also be done in $O(\log n)$ time.

Since the last two steps are executed at most n times, and the first two steps are executed only once, the overall running time of this algorithm is $O(n^2 \log n)$. That is better than $O(n^3)$, but it still puts a strong limit on how large n can be before it becomes infeasible to use this clustering approach.

7.2.3 Alternative Rules for Controlling Hierarchical Clustering

We have seen one rule for picking the best clusters to merge: find the pair with the smallest distance between their centroids. Some other options are:

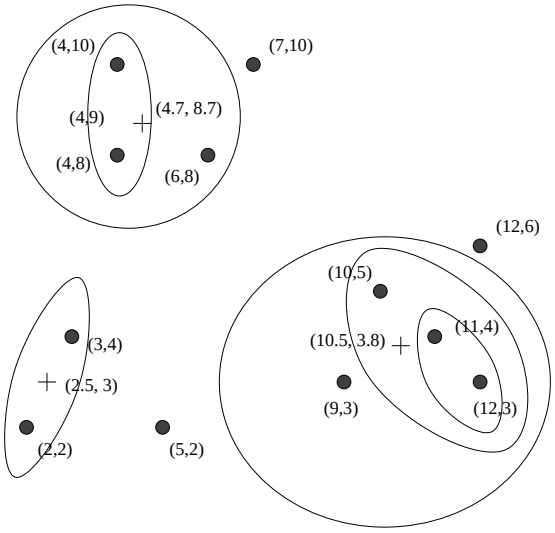

Figure 7.5: Three more steps of the hierarchical clustering

1. Take the distance between two clusters to be the minimum of the distances between any two points, one chosen from each cluster. For example, in Fig. 7.3 we would next chose to cluster the point (10,5) with the cluster of two points, since (10,5) has distance $\sqrt{2}$, and no other pair of unclustered points is that close. Note that in Example 7.2, we did make this combination eventually, but not until we had combined another pair of points. In general, it is possible that this rule will result in an entirely different clustering from that obtained using the distance-of-centroids rule.

2. Take the distance between two clusters to be the average distance of all pairs of points, one from each cluster.

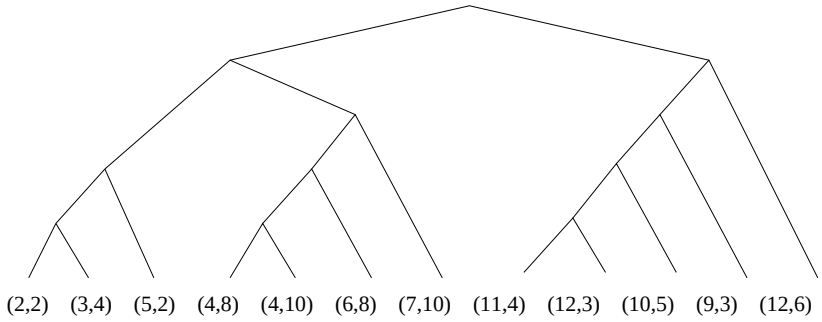

Figure 7.6: Tree showing the complete grouping of the points of Fig. 7.2

3. The *radius* of a cluster is the maximum distance between all the points and the centroid. Combine the two clusters whose resulting cluster has the lowest radius. A slight modification is to combine the clusters whose result has the lowest average distance between a point and the centroid. Another modification is to use the sum of the squares of the distances between the points and the centroid. In some algorithms, we shall find these variant definitions of "radius" referred to as "the radius."

4. The *diameter* of a cluster is the maximum distance between any two points of the cluster. Note that the radius and diameter of a cluster are not related directly, as they are in a circle, but there is a tendency for them to be proportional. We may choose to merge those clusters whose resulting cluster has the smallest diameter. Variants of this rule, analogous to the rule for radius, are possible.

Example 7.4: Let us consider the cluster consisting of the five points at the right of Fig. 7.2. The centroid of these five points is (10.8, 4.2). There is a tie for the two furthest points from the centroid: (9,3) and (12,6), both at distance $\sqrt{4.68} = 2.16$. Thus, the radius is 2.16. For the diameter, we find the two points in the cluster having the greatest distance. These are again (9,3) and (12,6). Their distance is $\sqrt{18} = 4.24$, so that is the diameter. Notice that the diameter is not exactly twice the radius, although it is close in this case. The reason is that the centroid is not on the line between (9,3) and (12,6). □

We also have options in determining when to stop the merging process. We already mentioned "stop when we have k clusters" for some predetermined k. Here are some other options.

1. Stop if the diameter of the cluster that results from the best merger exceeds a threshold. We can also base this rule on the radius, or on any of the variants of the radius mentioned above.

2. Stop if the *density* of the cluster that results from the best merger is below some threshold. The density can be defined in many different ways. Roughly, it should be the number of cluster points per unit volume of the cluster. That ratio can be estimated by the number of points divided by some power of the diameter or radius of the cluster. The correct power could be the number of dimensions of the space. Sometimes, 1 or 2 is chosen as the power, regardless of the number of dimensions.

3. Stop when there is evidence that the next pair of clusters to be combined yields a bad cluster. For example, we could track the average diameter of all the current clusters. As long as we are combining points that truly belong in a cluster, this average will rise gradually. However, if we combine two clusters that really don't deserve to be combined, then the average diameter will take a sudden jump.

Example 7.5: Let us reconsider Fig. 7.2. It has three natural clusters. We computed the diameter of the largest – the five points at the right – in Example 7.4; it is 4.24. The diameter of the 3-node cluster at the lower left is 3, the distance between (2,2) and (5,2). The diameter of the 4-node cluster at the upper left is $\sqrt{13} = 3.61$. The average diameter, 3.62, was reached starting from 0 after nine mergers, so the rise is evidently slow: about 0.4 per merger.

If we are forced to merge two of these natural clusters, the best we can do is merge the two at the left. The diameter of this cluster is $\sqrt{89} = 9.43$; that is the distance between the two points (2,2) and (7,10). Now, the average of the diameters is $(9.43 + 4.24)/2 = 6.84$. This average has jumped almost as much in one step as in all nine previous steps. That comparison indicates that the last merger was inadvisable, and we should roll it back and stop. □

7.2.4 Hierarchical Clustering in Non-Euclidean Spaces

When the space is non-Euclidean, we need to use some distance measure that is computed from points, such as Jaccard, cosine, or edit distance. That is, we cannot base distances on "location" of points. The algorithm of Section 7.2.1 requires distances between points to be computed, but presumably we have a way to compute those distances. A problem arises when we need to represent a cluster, because we cannot replace a collection of points by their centroid.

Example 7.6: The problem arises for any of the non-Euclidean distances we have discussed, but to be concrete, suppose we are using edit distance, and we decide to merge the strings `abcd` and `aecdb`. These have edit distance 3 and might well be merged. However, there is no string that represents their average, or that could be thought of as lying naturally between them. We could take one of the strings that we might pass through when transforming one string to the other by single insertions or deletions, such as `aebcd`, but there are many such options. Moreover, when clusters are formed from more than two strings, the notion of "on the path between" stops making sense. □

Given that we cannot combine points in a cluster when the space is non-Euclidean, our only choice is to pick one of the points of the cluster itself to represent the cluster. Ideally, this point is close to all the points of the cluster, so it in some sense lies in the "center." We call the representative point the *clustroid*. We can select the clustroid in various ways, each designed to, in some sense, minimize the distances between the clustroid and the other points in the cluster. Common choices include selecting as the clustroid the point that minimizes:

1. The sum of the distances to the other points in the cluster.
2. The maximum distance to another point in the cluster.
3. The sum of the squares of the distances to the other points in the cluster.

Example 7.7: Suppose we are using edit distance, and a cluster consists of the four points `abcd`, `aecdb`, `abecb`, and `ecdab`. Their distances are found in the following table:

	`ecdab`	`abecb`	`aecdb`
`abcd`	5	3	3
`aecdb`	2	2	
`abecb`	4		

If we apply the three criteria for being the centroid to each of the four points of the cluster, we find:

Point	Sum	Max	Sum-Sq
`abcd`	11	5	43
`aecdb`	7	3	17
`abecb`	9	4	29
`ecdab`	11	5	45

We can see from these measurements that whichever of the three criteria we choose, `aecdb` will be selected as the clustroid. In general, different criteria could yield different clustroids. □

The options for measuring the distance between clusters that were outlined in Section 7.2.3 can be applied in a non-Euclidean setting, provided we use the clustroid in place of the centroid. For example, we can merge the two clusters whose clustroids are closest. We could also use the average or minimum distance between all pairs of points from the clusters.

Other suggested criteria involved measuring the density of a cluster, based on the radius or diameter. Both these notions make sense in the non-Euclidean environment. The diameter is still the maximum distance between any two points in the cluster. The radius can be defined using the clustroid in place of the centroid. Moreover, it makes sense to use the same sort of evaluation for the radius as we used to select the clustroid in the first place. For example, if we take the clustroid to be the point with the smallest sum of squares of distances to the other nodes, then define the radius to be that sum of squares (or its square root).

Finally, Section 7.2.3 also discussed criteria for stopping the merging of clusters. None of these criteria made direct use of the centroid, except through the notion of radius, and we have already observed that "radius" makes good sense in non-Euclidean spaces. Thus, there is no substantial change in the options for stopping criteria when we move from Euclidean to non-Euclidean spaces.

7.2.5 Exercises for Section 7.2

Exercise 7.2.1: Perform a hierarchical clustering of the one-dimensional set of points 1, 4, 9, 16, 25, 36, 49, 64, 81, assuming clusters are represented by

their centroid (average), and at each step the clusters with the closest centroids are merged.

Exercise 7.2.2: How would the clustering of Example 7.2 change if we used for the distance between two clusters:

(a) The minimum of the distances between any two points, one from each cluster.

(b) The average of the distances between pairs of points, one from each of the two clusters.

Exercise 7.2.3: Repeat the clustering of Example 7.2 if we choose to merge the two clusters whose resulting cluster has:

(a) The smallest radius.

(b) The smallest diameter.

Exercise 7.2.4: Compute the density of each of the three clusters in Fig. 7.2, if "density" is defined to be the number of points divided by

(a) The square of the radius.

(b) The diameter (not squared).

What are the densities, according to (a) and (b), of the clusters that result from the merger of any two of these three clusters. Does the difference in densities suggest the clusters should or should not be merged?

Exercise 7.2.5: We can select clustroids for clusters, even if the space is Euclidean. Consider the three natural clusters in Fig. 7.2, and compute the clustroids of each, assuming the criterion for selecting the clustroid is the point with the minimum sum of distances to the other point in the cluster.

! **Exercise 7.2.6:** Consider the space of strings with edit distance as the distance measure. Give an example of a set of strings such that if we choose the clustroid by minimizing the sum of the distances to the other points we get one point as the clustroid, but if we choose the clustroid by minimizing the maximum distance to the other points, another point becomes the clustroid.

7.3 K-means Algorithms

In this section we begin the study of point-assignment algorithms. The best known family of clustering algorithms of this type is called k-means. They assume a Euclidean space, and they also assume the number of clusters, k, is known in advance. It is, however, possible to deduce k by trial and error. After an introduction to the family of k-means algorithms, we shall focus on a particular algorithm, called BFR after its authors, that enables us to execute k-means on data that is too large to fit in main memory.

7.3.1 K-Means Basics

A k-means algorithm is outlined in Fig. 7.7. There are several ways to select the initial k points that represent the clusters, and we shall discuss them in Section 7.3.2. The heart of the algorithm is the for-loop, in which we consider each point other than the k selected points and assign it to the closest cluster, where "closest" means closest to the centroid of the cluster. Note that the centroid of a cluster can migrate as points are assigned to it. However, since only points near the cluster are likely to be assigned, the centroid tends not to move too much.

```
Initially choose k points that are likely to be in
    different clusters;
Make these points the centroids of their clusters;
FOR each remaining point p DO
    find the centroid to which p is closest;
    Add p to the cluster of that centroid;
    Adjust the centroid of that cluster to account for p;
END;
```

Figure 7.7: Outline of k-means algorithms

An optional step at the end is to fix the centroids of the clusters and to reassign each point, including the k initial points, to the k clusters. Usually, a point p will be assigned to the same cluster in which it was placed on the first pass. However, there are cases where the centroid of p's original cluster moved quite far from p after p was placed there, and p is assigned to a different cluster on the second pass. In fact, even some of the original k points could wind up being reassigned. As these examples are unusual, we shall not dwell on the subject.

7.3.2 Initializing Clusters for K-Means

We want to pick points that have a good chance of lying in different clusters. There are two approaches.

1. Pick points that are as far away from one another as possible.
2. Cluster a sample of the data, perhaps hierarchically, so there are k clusters. Pick a point from each cluster, perhaps that point closest to the centroid of the cluster.

The second approach requires little elaboration. For the first approach, there are variations. One good choice is:

```
Pick the first point at random;
```

```
WHILE there are fewer than k points DO
    Add the point whose minimum distance from the selected
      points is as large as possible;
END;
```

Example 7.8: Let us consider the twelve points of Fig. 7.2, which we reproduce here as Fig. 7.8. In the worst case, our initial choice of a point is near the center, say (6,8). The furthest point from (6,8) is (12,3), so that point is chosen next.

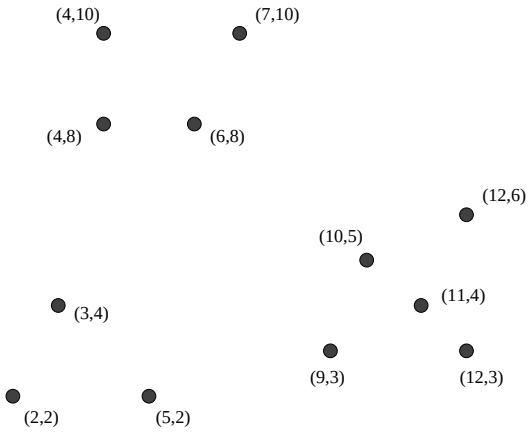

Figure 7.8: Repeat of Fig. 7.2

Among the remaining ten points, the one whose minimum distance to either (6,8) or (12,3) is a maximum is (2,2). That point has distance $\sqrt{52} = 7.21$ from (6,8) and distance $\sqrt{101} = 10.05$ to (12,3); thus its "score" is 7.21. You can check easily that any other point is less than distance 7.21 from at least one of (6,8) and (12,3). Our selection of three starting points is thus (6,8), (12,3), and (2,2). Notice that these three belong to different clusters.

Had we started with a different point, say (10,5), we would get a different set of three initial points. In this case, the starting points would be (10,5), (2,2), and (4,10). Again, these points belong to the three different clusters. □

7.3.3 Picking the Right Value of k

We may not know the correct value of k to use in a k-means clustering. However, if we can measure the quality of the clustering for various values of k, we can usually guess what the right value of k is. Recall the discussion in Section 7.2.3, especially Example 7.5, where we observed that if we take a measure of appropriateness for clusters, such as average radius or diameter, that value will grow slowly, as long as the number of clusters we assume remains at or above the true number of clusters. However, as soon as we try to form fewer

clusters than there really are, the measure will rise precipitously. The idea is expressed by the diagram of Fig. 7.9.

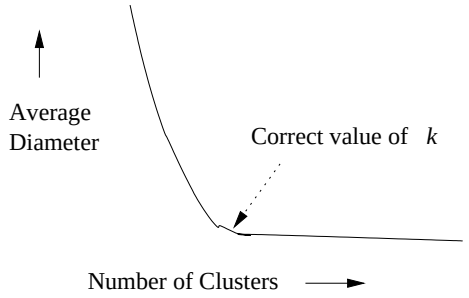

Figure 7.9: Average diameter or another measure of diffuseness rises quickly as soon as the number of clusters falls below the true number present in the data

If we have no idea what the correct value of k is, we can find a good value in a number of clustering operations that grows only logarithmically with the true number. Begin by running the k-means algorithm for $k = 1, 2, 4, 8, \ldots$. Eventually, you will find two values v and $2v$ between which there is very little decrease in the average diameter, or whatever measure of cluster cohesion you are using. We may conclude that the value of k that is justified by the data lies between $v/2$ and v. If you use a binary search (discussed below) in that range, you can find the best value for k in another $\log_2 v$ clustering operations, for a total of $2\log_2 v$ clusterings. Since the true value of k is at least $v/2$, we have used a number of clusterings that is logarithmic in k.

Since the notion of "not much change" is imprecise, we cannot say exactly how much change is too much. However, the binary search can be conducted as follows, assuming the notion of "not much change" is made precise by some formula. We know that there is too much change between $v/2$ and v, or else we would not have gone on to run a clustering for $2v$ clusters. Suppose at some point we have narrowed the range of k to between x and y. Let $z = (x + y)/2$. Run a clustering with z as the target number of clusters. If there is not too much change between z and y, then the true value of k lies between x and z. So recursively narrow that range to find the correct value of k. On the other hand, if there is too much change between z and y, then use binary search in the range between z and y instead.

7.3.4 The Algorithm of Bradley, Fayyad, and Reina

This algorithm, which we shall refer to as *BFR* after its authors, is a variant of k-means that is designed to cluster data in a high-dimensional Euclidean space. It makes a very strong assumption about the shape of clusters: they must be normally distributed about a centroid. The mean and standard deviation for a cluster may differ for different dimensions, but the dimensions must be