1 Clustering

Given points in some space — often a high-dimensional space — group the points into a small number of *clusters*, each cluster consisting of points that are "near" in some sense. Some applications:

1. Many years ago, during a cholera outbreak in London, a physician plotted the location of cases on a map, getting a plot that looked like Fig. 1. Properly visualized, the data indicated that cases clustered around certain intersections, where there were polluted wells, not only exposing the cause of cholera, but indicating what to do about the problem. Alas, not all data mining is this easy, often because the clusters are in so many dimensions that visualization is very hard.

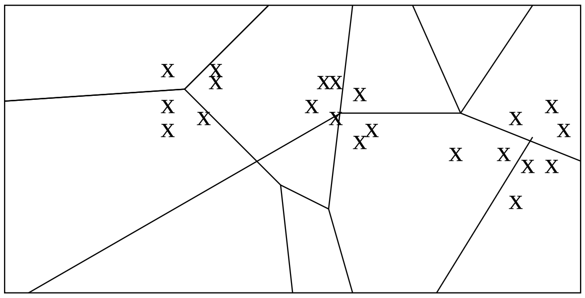

Figure 1: Clusters of cholera cases indicated where the polluted wells were

2. *Skycat* clustered 2×10^9 sky objects into stars, galaxies, quasars, etc. Each object was a point in a space of 7 dimensions, with each dimension representing radiation in one band of the spectrum. The Sloan Sky Survey is a more ambitious attempt to catalog and cluster the entire visible universe.

3. Documents may be thought of as points in a high-dimensional space, where each dimension corresponds to one possible word. The position of a document in a dimension is the number of times the word occurs in the document (or just 1 if it occurs, 0 if not). Clusters of documents in this space often correspond to groups of documents on the same topic.

1.1 Distance Measures

To discuss whether a set of points is close enough to be considered a cluster, we need a *distance measure* $D(x, y)$ that tells how far points x and y are. The usual axioms for a distance measure D are:

1. $D(x, x) = 0$. A point is distance 0 from itself.
2. $D(x, y) = D(y, x)$. Distance is symmetric.
3. $D(x, y) \leq D(x, z) + D(z, y)$. The *triangle inequality.*

Often, our points may be thought to live in a k-dimensional Euclidean space, and the distance between any two points, say $x = [x_1, x_2, \ldots, x_k]$ and $y = [y_1, y_2, \ldots, y_k]$ is given in one of the usual manners:

1. Common distance ("L_2 norm"): $\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$.
2. *Manhattan distance* ("L_1 norm"): $\sum_{i=1}^{k} |x_i - y_i|$.
3. Max of dimensions ("L_∞ norm"): $\max_{i=1}^{k} |x_i - y_i|$.

When there is no Euclidean space in which to place the points, clustering becomes more difficult. Here are some examples where a distance measure without a Euclidean space makes sense.

Example 1.1: We can think of Web pages as points in the roughly 10^8-dimensional space where each dimension corresponds to one word. However, computation of distances would be prohibitive in a space this large. A better approach is to base the distance $D(x, y)$ on the dot product of vectors corresponding to x and y, since we then have to deal only with the words actually present in both x and y.
We need to compute the lengths of the vectors involved, which is the square root of the sum of the squares of the numbers of occurrences of each word. The sum of the product of the occurrences of each word in each document is divided by each of the lengths to get a normalized dot product. We subtract this quantity from 1 to get the distance between x and y.
For instance, suppose there were only 4 words of interest, and the vectors $x = [2, 0, 3, 1]$ and $y = [5, 3, 2, 0]$ represented the numbers of occurrences of these words in the two documents. The dot product $x.y$ is $2 \times 5 + 0 \times 3 + 3 \times 2 + 1 \times 0 = 16$, the length of the first vector is $\sqrt{2^2 + 0^2 + 3^2 + 1^2} = \sqrt{14}$, and the length of the second is $\sqrt{5^2 + 3^2 + 2^2 + 0^2} = \sqrt{38}$. Thus,

$$D(x, y) = 1 - \frac{16}{\sqrt{14}\sqrt{38}} = 0.304$$

As another example, suppose document x has word-vector $[a_1, a_2, \ldots]$, and y is two copies of x; i.e., $y = [2a_1, 2a_2, \ldots]$. Then

$$D(x, y) = 1 - \frac{\sum_i 2a_i^2}{\sqrt{\sum_i a_i^2}\sqrt{\sum_i (2a_i)^2}} = 0$$

That is, x and y are essentially the same document; surely they are on the same topic, and deserve to be clustered together. □

Example 1.2: Character strings, such as DNA sequences may be similar even though there are some insertions and deletions as well as changes in some characters. For instance, *abcde* and *bcdxye* are rather similar, even though they don't have any positions in common, and don't even have the same length. Thus, instead of trying to construct a Euclidean space with one dimension for each position, we can define the distance function $D(x, y) = |x| + |y| - 2|LCS(x, y)|$, where LCS stands for the longest common subsequence of x and y. In our example, $LCS(abcde, bcdxye)$ is $bcde$, of length 4, so $D(abcde, bcdxye) = 5 + 6 - 2 \times 4 = 3$; i.e., the strings are fairly close. □

1.2 The Curse of Dimensionality

An unintuitive consequence of working in a high-dimensional space is that almost all pairs of points are about as far away as average.

Example 1.3: Suppose we throw points at random into a k-dimensional unit cube. If $k = 2$, we expect that the points will spread out in the plane, with some very nearby points and some pairs at almost the maximum possible distance, $\sqrt{2}$.
However, suppose k is large, say 100, or 100,000. Regardless of which norm we use, L_2, L_1, or L_∞, we know that $D(x, y) \geq \max_i |x_i - y_i|$, if $x = [x_1, x_2, \ldots]$ and $y = [y_1, y_2, \ldots]$. For large k, it is very likely that there will be some dimension i such that x_i and y_i are almost as different as possible, even if x and y are very close in other dimensions. Thus, $D(x, y)$ is going to be very close to 1. □

Another interesting consequence of high-dimensionality is that all vectors, such as $x = [x_1, x_2, \ldots]$ and $y = [y_1, y_2, \ldots]$ are almost orthogonal. The reason is that if we project x and y onto any of the $\binom{k}{2}$ planes formed by two of the k axes, there is going to be one in which the projected vectors are almost orthogonal.

1.3 Approaches to Clustering

At a high level, we can divide clustering algorithms into two broad classes:

1. *Centroid* approaches. We guess the centroids or central point in each cluster, and assign points to the cluster of their nearest centroid.

2. *Hierarchical* approaches. We begin assuming that each point is a cluster by itself. We repeatedly merge nearby clusters, using some measure of how close two clusters are (e.g., distance between their centroids), or how good a cluster the resulting group would be (e.g., the average distance of points in the cluster from the resulting centroid).

1.4 Outline

We shall consider the following algorithms for clustering; they differ in whether or not they assume a Euclidean distance, and in whether they use a centroid or hierarchical approach.

1. BFR: Centroid based; assumes Euclidean measure, with clusters formed by a Gaussian process in each dimension around the centroid.

2. Fastmap: Not really a clustering algorithm, but a way to construct a low-dimensional Euclidean space from an arbitrary distance measure.

3. GRGPF: Centroid-based, but uses only a distance measure, not a Euclidean space.

4. CURE: Hierarchical and Euclidean, this algorithm deals with odd-shaped clusters.

Note that all the algorithms we consider are oriented toward clustering large amounts of data, in particular, data so large it does not fit in main memory. Thus, we are especially concerned with the number of passes through the data that must be made, preferably one pass.

1.5 The k-Means Algorithm

This algorithm is a popular main-memory algorithm, on which the BFR algorithm is based. k-means picks k cluster centroids and assigns points to the clusters by picking the closest centroid to the point in question. As points are assigned to clusters, the centroid of the cluster may migrate.

Example 1.4: For a very simple example of five points in two dimensions, observe Fig. 2. Suppose we assign the points 1, 2, 3, 4, and 5 in that order, with $k = 2$. Then the points 1 and 2 are assigned to the two clusters, and become their centroids for the moment.

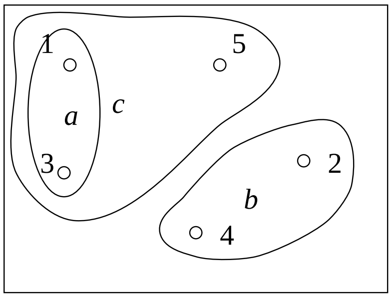

Figure 2: An example of the k-means algorithm

When we consider point 3, suppose it is closer to 1, so 3 joins the cluster of 1, whose centroid moves to the point indicated as a. Suppose that when we assign 4, we find that 4 is closer to 2 than to a, so 4 joins 2 in its cluster, whose center thus moves to b. Finally, 5 is closer to a than b, so it joins the cluster $\{1, 3\}$, whose centroid moves to c. $\square$

- We can initialize the k centroids by picking points sufficiently far away from any other centroid, until we have k.

- As computation progresses, we can decide to split one cluster and merge two, to keep the total at k. A test for whether to do so might be to ask whether so doing reduces the average distance from points to their centroids.

- Having located the centroids of the k clusters, we can reassign all points, since some points that were assigned early may actually wind up closer to another centroid, as the centroids move about.
- If we are not sure of k, we can try different values of k until we find the smallest k such that increasing k does not much decrease the average distance of points to their centroids. Example 1.5 illustrates this point.

Example 1.5: Consider the data suggested by Fig. 3. Clearly, $k = 3$ is the right number of clusters, but suppose we first try $k = 1$. Then all points are in one cluster, and the average distance to the centroid will be high.

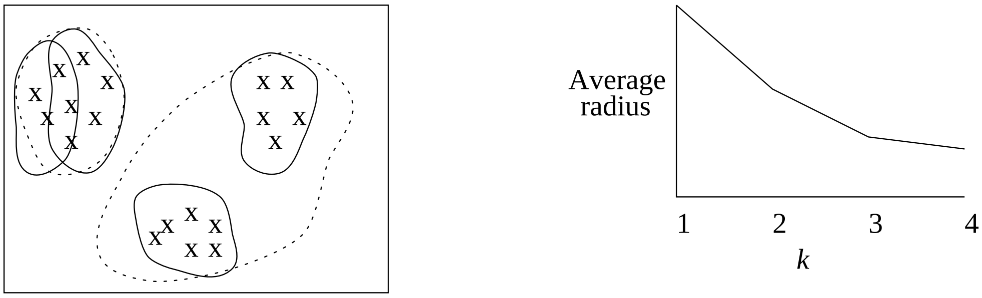

Figure 3: Discovering that $k = 3$ is the right number of clusters

Suppose we then try $k = 2$. One of the three clusters will be by itself and the other two will be forced into one cluster, as suggested by the dotted lines. The average distance of points to the centroid will thus shrink considerably.
If $k = 3$, then each of the apparent clusters should be a cluster by itself, and the average distance from points to their centroids shrinks again, as indicated by the graph in Fig. 3. However, if we increase k to 4, then one of the true clusters will be artificially partitioned into two nearby clusters, as suggested by the solid lines. The average distance to centroid will drop a bit, but not much. It is this failure to drop further that tips us off that $k = 3$ is right, even if the data is in so many dimensions that we cannot visualize the clusters. $\square$

1.6 The BFR Algorithm

Based on k-means, this algorithm reads its data once, consuming a main-memory-full at a time. The algorithm works best if the clusters are normally distributed around a central point, perhaps with a different standard deviation in each dimension. Figure 4 suggests what the data belonging to a typical cluster in two-dimensions might look like. A centroid, marked by +, has points scattered around, with the standard deviation σ in the horizontal dimension being twice what it is in the vertical dimension. About 70% of the points will lie within the 1σ ellipse; 95% will lie within 2σ, 99.9% within 3σ, and 99.9999% within 4σ.

1.7 Representation of Clusters in BFR

Having worked on Skycat, Usama Fayyad (the "F" in BFR) probably thought of clusters as "galaxies," as suggested in Fig. 5. A cluster consists of:

1. A central core, the *Discard set* (DS). This set of points is considered certain to belong to the cluster. All the points in this set are replaced by some simple statistics, described below. Note: although called "discarded" points, these points in truth have a significant effect throughout the running of the algorithm, since they determine collectively where the centroid is and what the standard deviation of the cluster is in each dimension.

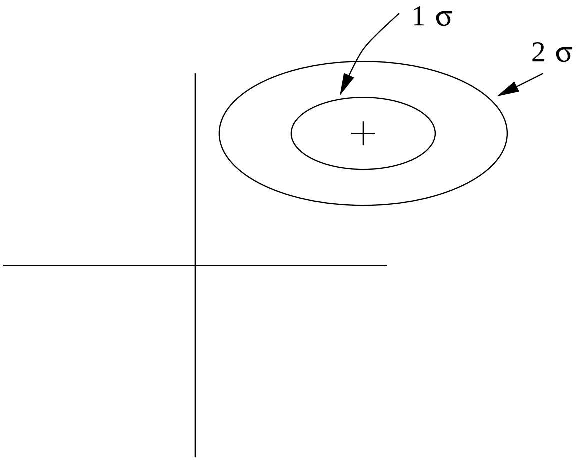

Figure 4: Points in a cluster are assumed to have been generated by taking a centroid for the cluster and adding to it in each dimension a normally distributed random variable with mean 0

2. Surrounding subgalaxies, collectively called the *Compression set* (CS). Each subcluster in the CS consists of a group of points that are sufficiently close to each other that they can be replaced by their statistics, just like the DS for a cluster is. However, they are sufficiently far away from any cluster's centroid, that we are not yet sure which cluster they belong to.

3. Individual stars that are not part of a galaxy or subgalaxy, the *Retained set* (RS). These points can neither be assigned to any cluster nor can they be grouped into a subcluster of the CS. They are stored in main memory, as individual points, along with the statistics of the DS and CS.

The statistics used to represent each cluster of the DS and each subcluster of the CS are:

1. The count of the number of points, N.

2. The vector of sums of the coordinates of the points in each dimension. The vector is called SUM, and the component in the ith dimension is SUM_i.

3. The vector of sums of squares of the coordinates of the points in each dimension, called $SUMSQ$. The component in dimension i is $SUMSQ_i$.

Note that these three pieces of information, totaling $2k+1$ numbers if there are k dimensions, are sufficient to compute important statistics of a cluster or subcluster, and they are more convenient to maintain as points are added to clusters than would, say, be the mean and variance in each dimension. For instance:

- The coordinate μ_i of the centroid of the cluster in dimension i is SUM_i/N.

- The variance in dimension i is

$$\frac{SUMSQ_i}{N} - \left(\frac{SUM_i}{N}\right)^2$$

  and the standard deviation σ_i is the square root of that.

1.8 Processing a Main-Memory-Full of Points in BFR

With the first load of main memory, BFR selects the k cluster centroids, using some chosen main-memory algorithm, e.g., pick a sample of points, optimize the clusters exactly, and chose their centroids as the initial centroids. The entire main-memory full of points is then processed like any subsequent memory-load of points, as follows:

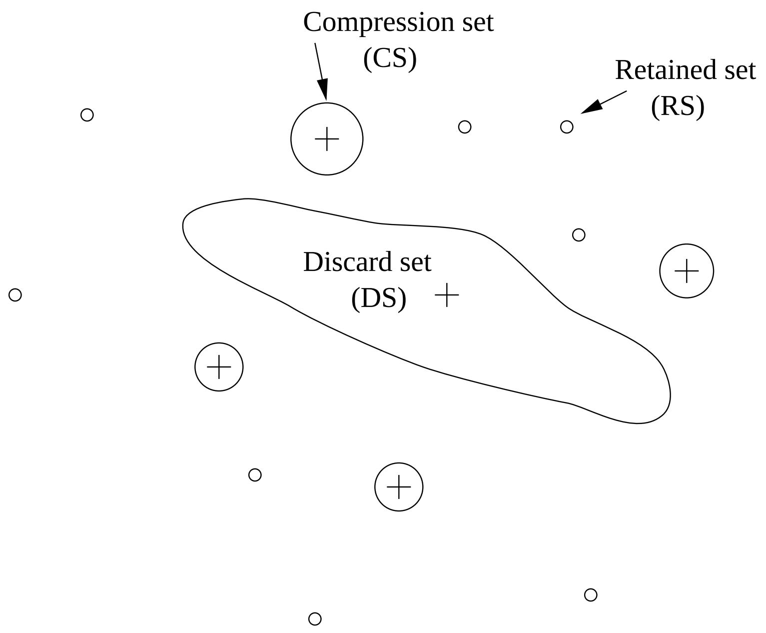

Figure 5: Examples of a cluster (DS), subclusters (CS), and individual points (RS)

1. Determine which points are sufficiently close to a current centroid that they may be taken into the DS and their statistics (N, SUM, $SUMSQ$) combined with the prior statistics of the cluster. BFR suggests two ways to determine whether a point is close enough to a centroid to be taken into the DS:

   (a) Take all points whose *Mahalanobis radius* is below a certain threshold, say four times the standard deviation of the cluster. The Mahalanobis radius is essentially the distance from the centroid, scaled in each dimension by σ_i, the standard deviation in that dimension. More precisely, if μ_i is the mean in dimension i, then the radius of point $y = [y_1, y_2, \ldots]$ is

   $$\sqrt{\sum_i \left(\frac{y_i - \mu_i}{\sigma_i}\right)^2}$$

   (b) Based on the number of points in the various clusters, ask whether it is likely (say at the 95% level) that the currently closest centroid will in the future move sufficiently far away from the point y, and another centroid will move sufficiently close to y that the latter is then closer to y than the former. If it is unlikely that the closest centroid for y will ever be different, then assign y to the DS, and place it in the cluster of the closest centroid.

2. Adjust the statistics N, SUM and $SUMSQ$ for each cluster to include the DS points just included.

3. In main memory, attempt to cluster the points that have not yet been placed in the DS, including points of the RS from previous rounds. If we find a cluster of points whose variance is below a chosen threshold, then we shall regard these points as a subcluster, replace them by their statistics, and consider them part of the CS. All other points are placed in the RS.

4. Consider merging a new subcluster with a previous subcluster of the CS. The test for whether it is desirable to do so is that the combined set of points will have a variance below a threshold. Note that the statistics kept for CS subclusters is sufficient to compute the variance of the combined set.

5. If we are at the last round, i.e., there is no more data, then we can assign subclusters in CS and points in RS to their nearest cluster, even though they will of necessity be fairly far away from any cluster centroid.