8 More About Clustering

We continue our discussion of large-scale clustering algorithms, covering:

1. Fastmap, and other ways to create a Euclidean space from an arbitrary distance measure.

2. The GRGPF algorithm for clustering without a Euclidean space.

3. The CURE algorithm for clustering odd-shaped clusters in a Euclidean space.

8.1 Simple Approaches to Building a Euclidean Space From a Distance Measure

Any n points can be placed in $(n-1)$-dimensional space, with distances preserved exactly.

Example 8.1: Figure 19 shows how we can place three points a, b, and c, with only a distance measure D given, in 2-dimensional space. Start by placing a and b distance $D(a,b)$ apart. Draw a circle of radius $D(a,c)$ around a and a circle of radius $D(b,c)$ around b. By the triangle inequality, which D must obey, these circles intersect. Pick one of the two points of intersection as c. $\square$

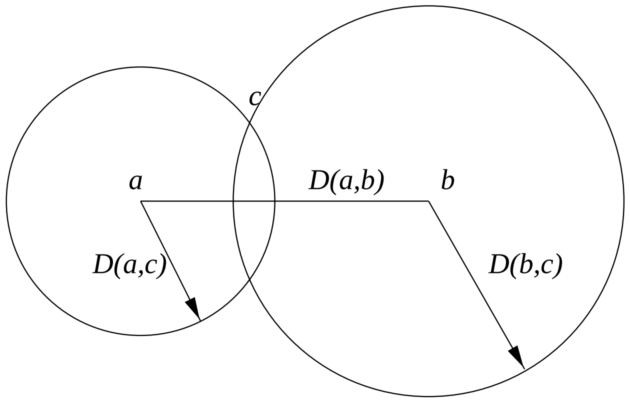

Figure 19: Placing three points in two dimensions

However, we usually have far too many points to try to place them in one fewer dimension. A more brute-force approach to placing n points in a k-dimensional space, where $k << n$, is called *multidimensional scaling.*

1. Start with the n points placed in k-dim space at random.

2. Take as the "error" the *energy* of a system of springs, each of length $D(x,y)$, that we imagine are strung between each pair of points x and y. The energy in a spring is the square of the difference between its actual length and $D(x,y)$.

3. Visit each point in turn, and try to place it in a position that minimizes the total energy of the springs. Since moving points around can affect the optimal position of other points, we must visit points repeatedly, until no more improvements can be made. At this point, we are in a local optimum, which may or may not be the global optimum.

Example 8.2: Suppose we have three points in a 3-4-5 "right triangle," as suggested in Fig. 20. If we try to place these points in one dimension, there must be some stretching and shrinking of springs. The optimum configuration, also shown in Fig. 20, is when the springs of length 3 and 4 are compressed to 7/3 and 10/3, while the spring of length 5 is stretched to 17/3. In this configuration, the total energy of the system is $(3-\frac{7}{3})^2+(4-\frac{10}{3})^2+(5-\frac{17}{3})^2=4/3$. $\square$

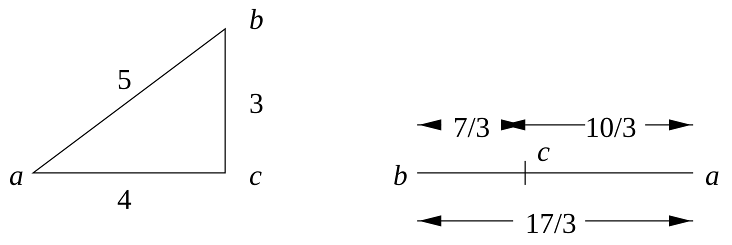

Figure 20: Optimal placement of three points in one dimension

8.2 Fastmap

Problem: multidimensional scaling requires at least $O(n^2)$ time to handle n points, since we must compute all distances at least once — perhaps more than once. *Fastmap* is a method for creating k pseudo-axes that can serve to place n points in a k-dim space in $O(nk)$ time.

In outline, Fastmap picks k pairs of points (a_i, b_i), each of which pairs serves as the "ends" of one of the k axes of the k-dim space. Using the law of cosines, we can calculate the "projection" x of any point c onto the line ab, using only the distances between points, not any assumed coordinates of these points in a plane. The diagram is shown in Fig. 21, and the formula is

$$x = \frac{D^2(a,c) + D^2(a,b) - D^2(b,c)}{2D(a,b)}$$

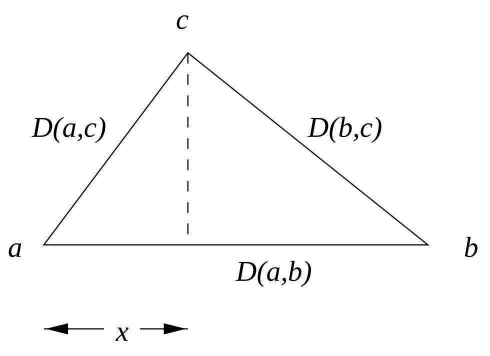

Figure 21: Computing the projection of point c onto line ab

Having picked a pair of points (a, b) as an axis, part of the distance between any two points c and d is accounted for by the projections of c and d onto line ab, and the remainder of the distance is in other dimensions. If the projections of c and d are x and y, respectively, then in the future (as we select other axes), the distance $D_{current}(c, d)$ should be related to the given distance function D by

$$D^2_{current}(c,d) = D^2(c,d) - (x-y)^2$$

The explanation is suggested by Fig. 22.

Here, then, is the outline of the Fastmap algorithm. It computes for each point c, k projections, which we shall refer to as $c^{(1)}, c^{(2)}, \ldots, c^{(k)}$ onto the k axes, which are determined by pairs of points (a_1, b_1), $(a_2, b_2), \ldots, (a_k, b_k)$. For $i = 1, 2, \ldots, k$ do the following:

1. Using the current distance $D_{current}$, Pick a_i and b_i, as follows:
   (a) Pick a random point c.
   (b) Pick a_i to be the point as far as possible from c, using distance $D_{current}$.
   (c) Pick b_i to be the point as far as possible from a_i.

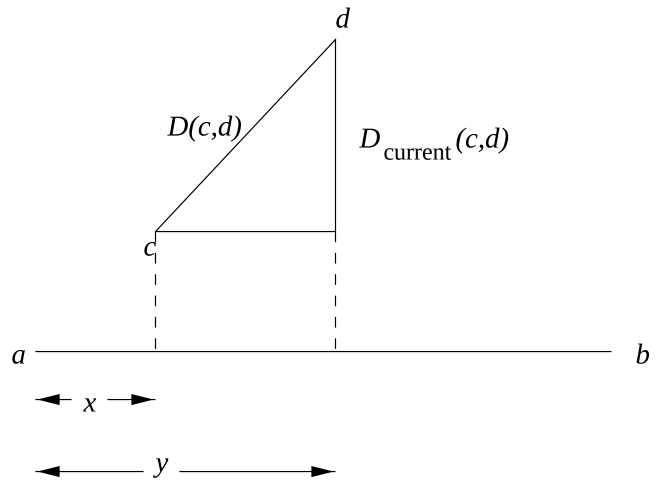

Figure 22: Subtracting the distance along axis ab to get the "current" distance between points c and d in other dimensions

2. For each point x, compute $x^{(i)}$, using the law-of-cosines formula described above.
3. Change the definition of $D_{current}$ to subtract the distance in the ith dimension as well as previous dimensions. That is
$$D_{current}(x, y) = \sqrt{D^2(x, y) - \sum_{j \le i}(x^{(j)} - y^{(j)})^2}$$
Note that no computation is involved in this step; we just use this formula when we need to find the current distance between specific pairs of points in steps (1) and (2).

8.3 Ways to Use Fastmap in Clustering Algorithms

1. Map the data to k dimensions in $O(nk)$ time. Cluster the points using any method that works for Euclidean spaces. Hope the resulting clusters are good (unlikely, according to GRGPF).
2. Improvement: Map to k dimensions using Fastmap, but use the Euclidean space only to estimate the *clustroids*: points that are closest on average to the other members of their cluster (like a centroid in a Euclidean space, but it has to be a particular point of the cluster, not a location that has no point). If there is a lot of data, we can use a sample for this stage. Then, assign points to clusters based on their true distance (using the distance measure, not the Euclidean space) to the clustroids.

8.4 Hierarchical Clustering

A general technique that, if we are not careful, will take $O(n^2)$ time to cluster n points, is *hierarchical* clustering.

1. Start with each point in a cluster by itself.
2. Repeatedly select two clusters to merge. In general, we want to pick the two clusters that are closest, but there are various ways we could measure "closeness." Some possibilities:
   (a) Distance between their centroids (or if the space is not Euclidean, between their clustroids).
   (b) Minimum distance between nodes in the clusters.
   (c) Maximum distance between nodes in the clusters.
   (d) Average distance between nodes of the clusters.
3. End the merger process when we have "few enough" clusters. Possibilities:

(a) Use a k-means approach — merge until only k clusters remain.

(b) Stop merging clusters when the only clusters that can result from merging fail to meet some criterion of compactness, e.g., the average distance of nodes to their clustroid or centroid is too high.

8.5 The GRGPF Algorithm

This algorithm assumes there is a distance measure D, but no Euclidean space. It also assumes that there is too much data to fit in main memory. The data structure it uses to store clusters is like an R-tree. Nodes of the tree are disk blocks, and we store different things at leaf and interior nodes:

- In leaf blocks, we store *cluster features* that summarize a cluster in a manner similar to BFR. However, since there is no Euclidean space, the "features are somewhat different, as follows:
  (a) The number of points in the cluster, N.
  (b) The *clustroid*: that point in the cluster that minimizes the *rowsum*, i.e., the sum of the squares of the distances to the other points of the cluster.
    - If C is a cluster, $\hat{C}$ will denote its clustroid.
    - Thus, the rowsum of the clustroid is $\sum_{X \ in \ C} D(\hat{C}, X)$.
    - Notice that the rowsum of the clustroid is analogous to the statistic $SUMSQ$ that was used in BFR. However, $SUMSQ$ is relative to the origin of the Euclidean space, while GRGPF assumes no such space. The rowsum can be used to compute a statistic, the *radius* of the cluster that is analogous to the standard deviation of a cluster in BFR. The formula is $radius = \sqrt{rowsum/N}$.
  (c) The p points in the cluster that are closest to the clustroid and their rowsums, for some chosen constant p.
  (d) The p points in the cluster that are farthest from the clustroid.
- In interior nodes, we keep samples of the clustroids of the clusters represented by the descendants of this tree node. An effort is made to keep the clusters in each subtree close. As in an R-tree, the interior nodes thus inform about the approximate region in which clusters at their descendants are found. When we need to insert a point into some cluster, we start at the root and proceed down the tree, choosing only those paths along which a reasonably close cluster might be found, judging from the samples at each interior node.

8.6 Maintenance of Cluster Features by GRGPF

There is an initialization phase on a main-memory full of data, where the first estimate of clusters is made, and the clusters themselves are arranged in a hierarchy corresponding to the "R-tree." The need for this hierarchy was mentioned in item 8.5 above.

We then examine all the other points and try to insert them into the existing clusters. The choice of clusters is reconsidered if there are too many to fit the representations in main memory, or if a cluster gets too big (too high a radius). There are many details about what happens when new points are added. Here are some of the key points:

(a) A new point X is placed in the cluster C such that $D(\hat{C}, X)$ is a minimum. Use the samples at the interior nodes of the tree to avoid searching the entire tree and all the clusters.

(b) If point X is added to cluster C, we add to each of the $2p+1$ rowsums maintained for that cluster (rowsums of $\hat{C}$ and the p closest and p furthest points) the square of the distance from that point to X. We also estimate the rowsum of X as $Nr^2 + ND(\hat{C}, X)$, where r is the radius of the cluster. The validity of this formula, as an approximation is, based on the property of the "curse of dimensionality" we discussed in Section 7.2. That is, for each of the N points Y in the cluster, the distance between X and Y can be measured by going to the clustroid (the term $D(\hat{C}, X)$), and then from the clustroid to Y (which is Y's contribution to r). If we assume that the lines (in a hypothetical Euclidean space) from $\hat{C}$ to X and Y are likely to be almost perpendicular, then the formula is justified by the Pythagorean theorem.

(c) We must consider the possibility that after adding X, cluster C now hasa different clustroid, which could be X or one of the p points closest to $\hat{C}$. If, after accounting for X, one of these points has a lower rowsum, it becomes the clustroid. obviously, this process cannot be maintained indefinitely, since eventually the clustroid will migrate outside the p closest points. Thus, there may need to be periodic recomputation of the cluster features for a cluster, which is possible — at the cost of disk I/O's — because all points in the cluster are stored on disk, even if they are not in the main-memory tree that stores the cluster features and samples.

8.7 Splitting and Merging Clusters in GRGPF

Sometimes, the radius of the clustroid exceeds a given threshold, and the algorithm decides to split the cluster into two. This process requires bringing the entire cluster into main memory and performing some split algorithm on just these points in memory.

An additional consequence is that the number of clusters in one leaf node increases by 1. If the node (disk block) overflows, then it must be split, as in a B-tree. That, in turn, may cause nodes up the tree to be split, and in the worst case, there is no room in main memory to hold the entire tree any more.

In that case, the solution is to raise the threshold that the radius of a cluster may have, and consider merging clusters throughout the tree. Suppose we consider merging clusters C_i and C_j. We wish to avoid having to bring all of the points of these clusters into main memory, so we guess at the clustroid and the rowsums as follows:

(a) Assume that the clustroid of the combined cluster will be one of the points furthest from the clustroid of either C_i or C_j. Remember that these points are all kept in the tree with their cluster features.

(b) To decide on the new clustroid, we need to estimate the rowsum for each point X in either cluster. Suppose for example that X is in C_i. Then we estimate:

$$Rowsum(X) = Rowsum_i(X) + N_j\left(D^2(X,\hat{C}_i) + D^2(\hat{C}_i,\hat{C}_j)\right) + Rowsum_j(\hat{C}_j)$$

Here, N_j is the number of points in C_j, and as always, hats indicate the clustroid of a cluster. The rationale for the above formula is that:

i. The first term represents the distances from X to all the nodes in its cluster.

ii. The middle term represents part of the paths from X to each point in C_j. We assume that the path starts out going from X to $\hat{C}_i$, the clustroid of C_i. From there, it goes to $\hat{C}_j$. Note that, by the "curse of dimensionality," we may assume that the lines from X to $\hat{C}_i$ and from $\hat{C}_i$ to $\hat{C}_j$ are "perpendicular" and combine them using the Pythagorean theorem.

iii. The final term represents the component of the paths from X to all the members of C_j that continues after $\hat{C}_j$ is reached from X. Again, it is the "curse of dimensionality" assumption that lets us assume all the lines from $\hat{C}_j$ to members of C_j are orthogonal to the line from $\hat{C}_i$ to $\hat{C}_j$.

(c) Having decided on the new clustroid, compute the rowsums for all points that we retain in the cluster features of the new cluster using the same formula as in (2).

8.8 CURE

We now return to clustering in a Euclidean space. The special problem solved by CURE is that when clusters are not neatly expressed as Gaussian noise around a central point (as BFR assume) many things can go wrong in a k-means approach.

Example 8.3: Figure 23 suggests two possible problems. In (a), even though $k = 4$ is the right number of clusters (solid circles represent the extent of the clusters), an algorithm that tries to minimize distances to a centroid might well cluster points as suggested by the dotted lines, with the large cluster broken into three parts, and the three small clusters combined into one. In (b), a long cluster could be interpreted as many round clusters, since that would minimize the average distance of points to the cluster centroids (Notice, however, that by using the Mahalanobis radius, as in Section 7.8, a cluster that is long is one dimension is recognized as "circular," and we would not expect BFR to make the mistake of Fig. 23(b). □

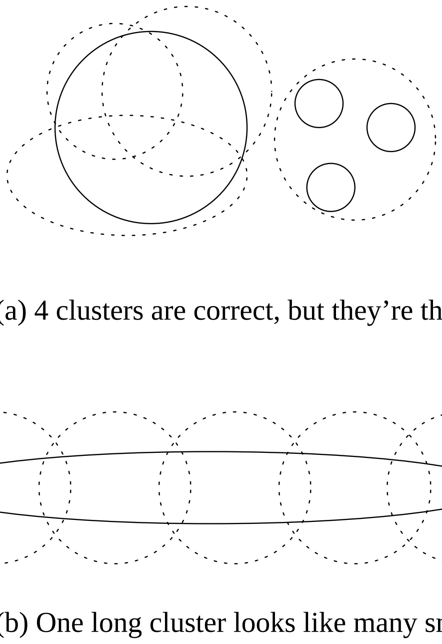
(a) 4 clusters are correct, but they're the wrong 4!
(b) One long cluster looks like many small ones!

Figure 23: Problems with centroid-based clustering

Here is an outline of the CURE algorithm:

1. Start with a main memory full of random points. Cluster these points using the hierarchical approach of Section 8.4. Note that hierarchical clustering tends to avoid the problems of Fig. 23, as long as the true clusters are dense with points throughout.

2. For each cluster, choose c "sample" points for some constant c. These points are picked to be as dispersed as possible, then moved slightly closer to the mean, as follows:

   (a) Pick the first sample point to be the point of the cluster farthest from the centroid.

   (b) Repeatedly pick additional sample points by choosing that point of the cluster whose minimum distance to an already chosen sample point is as great as possible.

   (c) When c sample points are chosen, move all the samples toward the centroid by some fractional distance, e.g., 20% of the way toward the centroid. As a result, the sample points need not be real points of the cluster, but that fact is unimportant. The net effect is that the samples are "typical" points, well dispersed around the cluster, no matter what the cluster's shape is.

3. Assign all points, including those involved in steps (1) and (2) to the nearest cluster, where "nearest" means shortest distance to some sample point.

Example 8.4: Figure 24 suggests the process of picking $c = 6$ sample points from an elongated cluster, and then moving htem 20% of the way toward the centroid. □

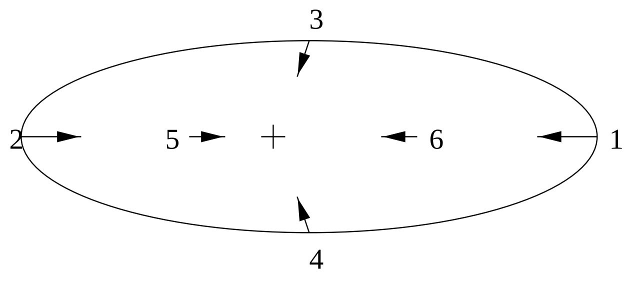

Figure 24: Selecting sample points in CURE