9 Hierarchical Clustering

This marks the beginning of the *clustering* section. The basic idea is to take a set X of items and somehow partition X into subsets, so each subset has similar items. Obviously, it would be great if we could be more specific, but that would end up omitting some particular form of clustering.

Clustering is an extremely *broad and ill-defined* topic. There are **many many** variants; we will not try to cover all of them. Instead we will focus on three broad classes of algorithms, and try to touch on some new developments.

Shouldn't we be spending *more* emphasis on clustering? Perhaps not. Of all of the studies that have emerged, one theme has rung consistently:

When data is easily cluster-able, most clustering algorithms work quickly and well.
When is not easily cluster-able, then no algorithm will find good clusters.

9.1 Hard Clustering Formulation

Lets specify our clustering goal a bit more. Start with a set $X \subset \mathcal{M}$ (say $\mathcal{M} = \mathbb{R}^d$), and a metric $\mathbf{d} : \mathcal{M} \times \mathcal{M} \to \mathbb{R}_+$.

A *cluster* S is a subset of X. A *clustering* is a partition $\mathcal{C}(X) = \{S_1, S_2, \ldots, S_k\}$ where

(C1) each $S_i \subset X$,

(C2) each pair $S_i \cap S_j = \emptyset$ (this is relaxed in *soft* clustering), and

(C3) $\bigcup_{i=1}^{k} S_i = P$.

Then the goal is two-fold:

width: For each $S \in \mathcal{C}(X)$ for all $x, x' \in S$ then $\mathbf{d}(x, x')$ is small.

split: For each $S_i, S_j \in \mathcal{C}(X)$, for all (most) $x_i \in S_i$ and $x_j \in S_j$ (and $i \neq j$) then $\mathbf{d}(x_i, x_j)$ is large.

Overall, the goal is for "split" / "width" to be large.

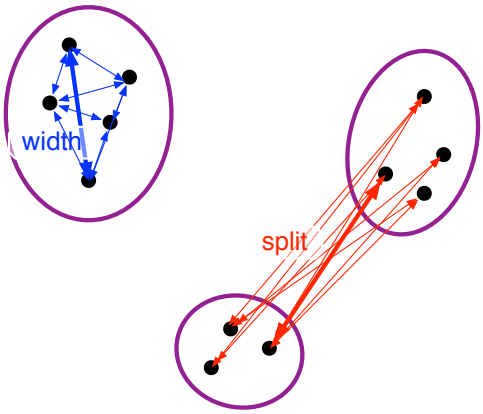


Figure 9.1: Example of clustering $\mathcal{C}(X) = \{S_1, S_2, S_3\}$.

9.2 Hierarchical/Agglomerative Clustering

The first type of clustering we study is *hierarchical*. The basic algorithm is:

If two points (clusters) are close (or close enough), put them in the same cluster.

A bit more formally, we can write this as an algorithm.

Algorithm 9.2.1 Hierarchical Clustering

Each $x_i \in X$ is a separate cluster S_i.
while Two clusters are *close enough* **do**
  Find the *closest* two clusters S_i, S_j
  Merge S_i and S_2 into a single cluster

It remains to define *close enough* and *closest*. These both basically require a distance between clusters. There are several options:

- distance between *center* of clusters. What does *center* mean?
  - the mean (average) point.
  - the center-point (like a high-dimensional median)
  - center of the MEB (minimum enclosing ball)
  - some (random) representative point
- distance between closest pair of points
- distance between furthest pair of points
- average distance between all pairs of points in different clusters
- radius of minimum enclosing ball of joined cluster
- smallest average distance between points in a cluster and center in the other

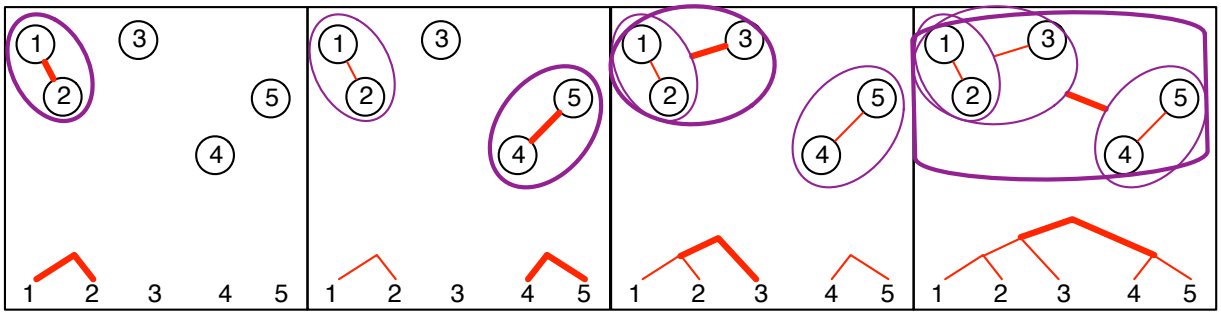
Among all clusters, there are often ties. Often break ties arbitrarily, but these choices can dramatically change the resulting clusters.

Finally we need to define *close enough*; this indicates some sort of threshold. There are again several options:

- If information is known about the data, perhaps some absolute value τ can be given.
- If the $\{$ diameter, or radius of MEB, or average distance from center point$\}$ is beneath some threshold. This fixes the scale of a cluster, which could be good or bad.
- If the *density* is beneath some threshold; where *density* is # points/volume or # points / radius^d
- If the joined (both clusters) density increases too much from single cluster density.
  Variations of this are called the "elbow" technique.
- When the number of clusters is k (for some magical value k).

9.2.1 Hierarchy

We can keep track of the order clusters are merged and build a *hierarchy* of clusterings. This can be useful structure beyond an opaque single partition of the data. This also means we do not need to (initially) choose when to stop. We can keep merging until there is only 1 cluster remaining. Then we can choose to "chop off the top of the tree" later.



The most famous example of this are phylogenic trees.

9.2.2 Efficiency

Lets use as an example where the distance between clusters is the distance between the centroids, and we stop when there is one cluster. For n points, this takes about n^3 time steps.

In each we make one merge, so there are $n-1 = O(n)$ rounds. In each round with m clusters, we may need to check $\binom{m}{2}$ pairs, to find the closest pair. When $m < n/2$ (which if half of all rounds), then this takes about $n^2/4 = O(n^2)$ time each round. It then takes only at most n time to recompute the centroid. So the total cost is about $(n-1) \cdot n^2/4 = O(n^3)$.

We can reduce this to $O(n^2 \log n)$. We maintain a *priority queue* of the $\binom{m}{2} = O(n^2)$ distances. Each update takes the closest distance in $O(\log n)$ time, but then affects $O(n)$ distances in the queue. These take $O(n \log n)$ time total to update. Centroids can also be updated in $O(1)$ time (by storing the count and center point - they can be merged by taking the weighted average). So each round takes $O(n \log n)$ time, and the total process takes $O(n^2 \log n)$ time.

But this is still pretty slow (as we will see compared to other techniques).

9.3 k-Center Clustering

Today we get a preview of *assignment based clustering*. In this method, each cluster is represented by a single point, to which all other points in the cluster are "assigned." In some sense the hierarchical versions using the centroid fall under this category.

In general these are defined for a set X, and distance $\mathbf{d} : X \times X \to \mathbb{R}_+$, and the output is a set $C = \{c_1, c_2, \ldots, c_k\}$. This implicitly defines a set of clusters where $\phi_C(x) = \arg\min_{c \in C} \mathbf{d}(x, c)$. Then the *$k$-center cluster problem* is to find the set C of k clusters (often, but always as a subset of X) to

$$\text{minimize } \max_{x \in X} \mathbf{d}(\phi_C(x), x).$$

So we want every point assigned to the closest center, and want to minimize the *longest* distance of any such assignment.

There are other variants:

- the *k-means clustering problem*: minimize $\sum_{x \in X} \mathbf{d}(\phi_C(x), x)^2$
  This will be the topic of next lecture
- the *k-median clustering problem*: minimize $\sum_{x \in X} \mathbf{d}(\phi_C(x), x)$

Unfortunately, the k-center clustering problem is NP-hard to solve exactly. In fact, it is NP-hard to find a clustering within a factor 2 of the optimal cost!

Luckily, there is an algorithm that achieves this factor 2 approximation, it is quite fast, and it works very well in practice.

9.4 Gonzalez Algorithm

He we discuss a simple greedy strategy for the k-center clustering problem. It is usually attributed to Gonzalez (1985), but it may likely be much older. The lesson is:

Be greedy, and avoid your neighbors!

Algorithm 9.4.1 Gonzalez Greedy Algorithm for k-Center Clustering

Choose $c_1 \in X$ arbitrarily. Let $C_1 = \{c_1\}$.
(*In general let* $C_i = \{c_1, \ldots, c_i\}$.)
for $i = 2$ to k **do**
  Set $c_i = \arg\max_{x \in X} \mathbf{d}(x, \phi_{C_{i-1}}(x))$.

As Algorithm 9.4.1 describes, the algorithm is to always pick the point in x that is furthest from the current set of centers, and let it also be a center.

In the worst case, this is a 2-approximation to the optimal clustering for the k-center clustering problem. But is often much better in practice.

It only takes time about $k^2 n = O(k^2 n)$. There are k rounds, and each round can be done in about nk time. We maintain the set $\phi_{C_i}(x)$ for each x. When a new c_i is found, and added to the set of centers, all n assignments $\phi_{C_i}(x)$ can be updated in linear $O(n)$ time, by checking each distance $\mathbf{d}(x, \phi_{C_{i-1}}(x))$ against $\mathbf{d}(x, c_i)$ and switching the assignment if the later is smaller. Then the minimum can be found in the next round on a linear scan (or on the same linear scan).

Algorithm 9.4.2 Detailed Gonzalez Greedy Algorithm for k-Center Clustering

Choose $c_1 \in X$ arbitrarily, and set $\phi[j] = 1$ for all $j \in [n]$
for $i = 2$ to k **do**
  $M = 0, \quad c_i = x_1$
  for $j = 1$ to n **do**
    if $\mathbf{d}(x_j, c_{\phi[j]}) > M$ **then**
      $M = \mathbf{d}(x_j, c_{\phi[j]}), \quad c_i = x_j$
  for $j = 1$ to n **do**
    if $\mathbf{d}(x_j, c_{\phi[j]}) > \mathbf{d}(x_j, c_i)$ **then**
      $\phi[j] = i$

This works for any metric. However, it biases the choice of centers to be on the "edges" of the dataset. There are heuristic to try to recenter afterwards, but usually not worth it—just use another algorithm instead.