8 Hierarchical Agglomerative 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 data is not easily cluster-able, then no algorithm will find good clusters.

8.1 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") or ("split" - "width") to be large.

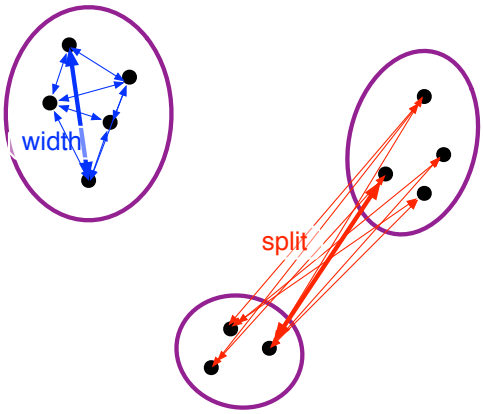


Figure 8.1: Example of clustering $\mathcal{C}(X) = \{S_1, S_2, S_3\}$.

8.2 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. Repeat.

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

Algorithm 8.2.1 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_j 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

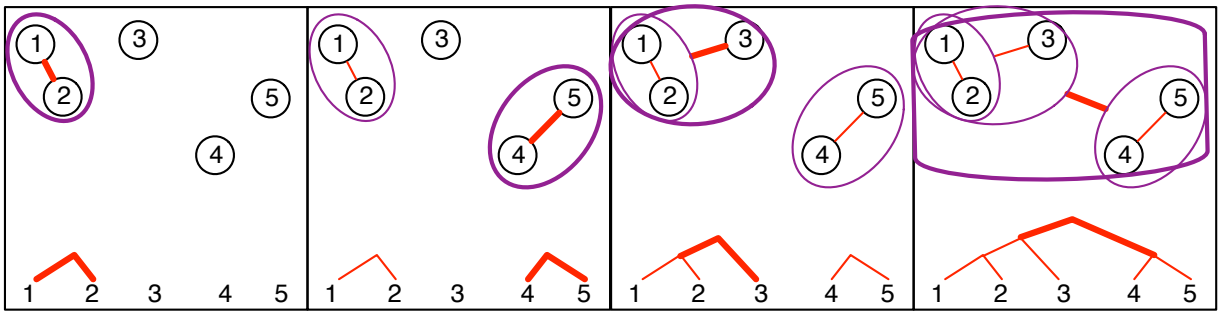
Among all clusters, there are often ties. Often break ties arbitrarily, but these choices can dramatically change the resulting clusters.

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).

8.2.1 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.

8.2.2 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).