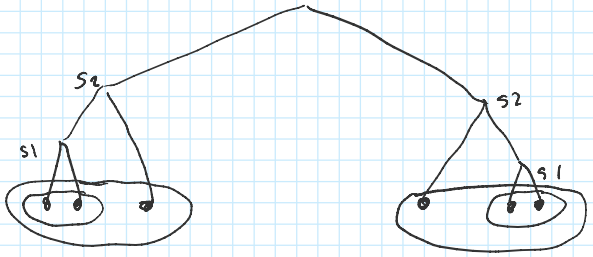
L8 Hierarchical Clustering

Sunday, February 16, 2020 7:14 AM

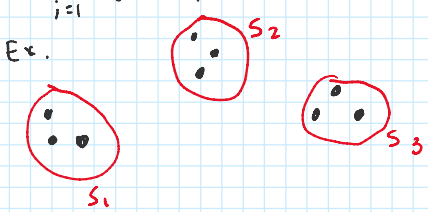
- starts by identifying similar points and grouping into clusters.
- similar clusters are then identified and considered for merging into larger clusters



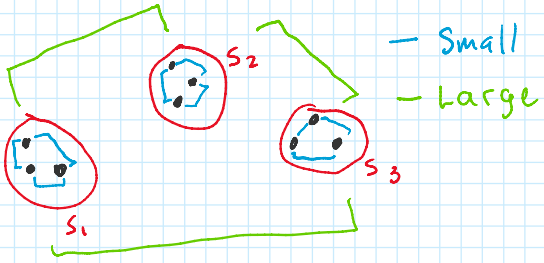
What is clustering??

- input is some set X with points in some space M
  Ex. $X \subset M$ (eg $M = \mathbb{R}^d$)
  data
- the other part of the input is a "distance" defined on the space.
  distance $d: M \times M \to \mathbb{R}$
- goal: $S = \{S_1, S_2, \dots, S_k\}$ is a set of clusters with properties
  - $S_i \subset X$ (a cluster is some sub-set of the data)
  - $S_i \cap S_j = \emptyset$ (hard clustering)
  - $\bigcup_{i=1}^{k} S_i = X$ (union of all clusters is the original data X)

Ex.



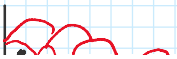
- Desirable properties?
  - we want the distance between clusters to be large.
  - we want the distance within clusters to be small.



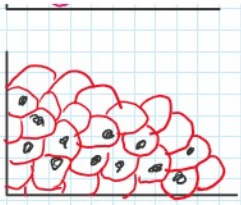
  - some clustering techniques only work with one distance in mind and hope that the other notion of distance is implicitly taken care of.

- Most Examples have $X \subset \mathbb{R}^d$ with $d = \|\cdot - \cdot\|$ (euclidean distance)
  - Recommended steps
    - S1. Plot the points
    - S2. Draw circles around the clusters you can identify visually.

- In many cases, the boundaries between points are not clear.
  - we can adapt our algorithms to identify small clusters by



- In many cases, the boundaries between points are not clear.
  - We can adapt our algorithms to identify small clusters by tiling the plain and estimating which points are close to one another.
  - But still, we should do a visual assessment if possible to determine if clustering is the best data mining approach.



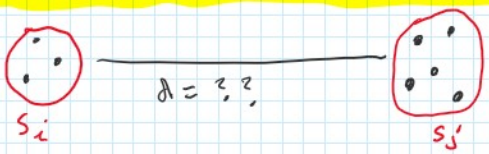
Hierarchical Agglomerative Clustering

- if two points (or clusters) are close enough...
  → Put them together in the same cluster.
- S1: Each $x_i \in X$ is put into its own, separate cluster.
  (i.e, if $|X| = n \rightarrow n$ clusters)
- S2: while two clusters S_i, S_j are close enough... (some threshold)
  - Find closest pair S_i, S_j
  - Merge S_i, S_j into a single cluster $S = S_i \cup S_j$

- **Distance Between Clusters S_i, S_j**

  $d: M \times M \rightarrow \mathbb{R}$ for $X \subset M$

  - At step 1 ($S_i = \{x_i\}$)
    ↳ then $d(S_i, S_j) = d(x_i, x_j)$
  - But at some point, sets S_i, S_j may contain multiple points x_i...
  - How can we define distance between multi-point clusters??



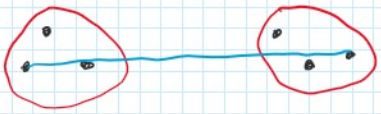
Approaches:

1. Single-Link Distance

$$d(S_i, S_j) = \min d(x_i', x_j') \quad \text{for} \begin{cases} x_i \in S_i \\ x_j \in S_j \end{cases}$$

- compares the pair-wise distance between all points in each cluster
- returns the minimum distance found between each comparison
- attempts to make distances between clusters large

2. Complete-Link Distance



$$d(S_i, S_j) = \max d(x_i, x_j) \quad \text{for} \begin{cases} x_i \in S_i \\ x_j \in S_j \end{cases}$$

- Compares the pair-wise distance between all points in each cluster
- returns the maximum distance found between each comparison
- attempts to make differences within clusters small

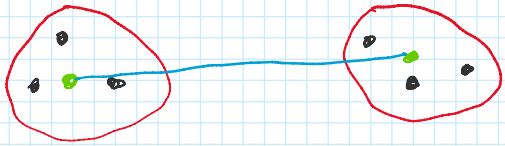
3. Mean-Link Distance

$$d(S_i, S_j) = d(\bar{S_i}, \bar{S_j})$$

where $\bar{S_i} = \frac{1}{|S_i|} \sum_{x \in S_i} x_i$

and similarly . . .

where $\bar{S_j} = \frac{1}{|S_j|} \sum_{x \in S_j} x_j$



- here, it is unnecessary to calculate the pair-wise distance for each point
- we only need to calculate the average point and then compute distances across averages as if they were single points.
- the centroid can be computed in a variety of ways to come up with new methods for distance.
- for instance, in some cases we may not know how to compute an "average" (ie, non-Euclidean spaces)
- in general, the centroid must be some "point".

4. Others

1. Single Ball Difference
   - Finds the minimum ball which encloses the points in $S_i \cup S_j$
   - The radius of the ball is returned as $d(S_i, S_j)$

2. Changes to calculation of "centroid"
   - for example, the centroid $= \sum_{x_i \in S_i} x_i$

Efficiency of HAC:

- most interesting when data is large: $n = |X|$ is large
- distance is not Euclidean in $\mathbb{R}^2$

let's think about the scenario where n is large and we are not in Euclidean space . . .

- then we can't pre-compute stuff by calculating an "average"
- we must make pair-wise comparisons as in the "Single-Link" and "Complete Link Methods"

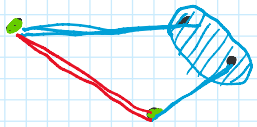
- Clustering a set of size n requires (n-1) rounds (Thus $O(n)$)
  - only one cluster is eliminated at each merge of S_i, S_j
  - within each merge, find the closest pair
    - for the first half $(n/2)$ merges, there are at least $(n/2)$ pairs
    - Thus we check $\binom{n/2}{2}$ comparisons which is roughly $O(n^2)$
    - We can improve to $O(n)$ checks by only updating the distances of the last merged cluster.

Ex:

$O(n^2)$

Every distance between every point is checked for the minimum.

$O(n)$



We don't check distance against every single point in the cluster.

Only the new blue lines need to be re-checked,

- Thus there are $O(n)$ rounds
- Each round we need to check $O(n)$ distances
- These distances can be maintained in a priority queue.
  - removing and updating the minimum distance in the priority queue.
  - this can be done in $\log(n)$ time

∴ The entire procedure can carefully be done in $O(n \cdot n \log(n))$ time.

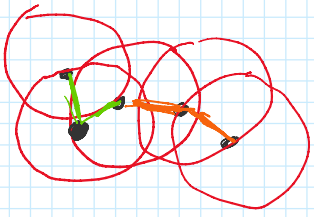
$= O(n^2 \log(n))$ computes of the distances.

- but each of these computations may be $O(n)$ or $O(n^2)$ because we need to compute the max and min of these pairs.
- mostly $O(n)$ if we only update the distances for the last merged cluster.

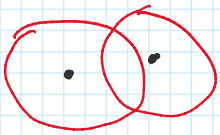
- $O(n^2 \log(n))$ is still very slow for large n
  - Other algorithms will have $O(nk)$ behavior, where k = the number of clusters.
    - These are <u>much</u>, <u>much</u> faster.
  - thus HAC is very slow, and some extensions have been developed to speed it up.
    - DB Scan (Density Based Clustering)

$$\begin{cases} X \subset \mathbb{R}^d \\ d = \|\cdot - \cdot\| \text{ (Euclidean)} \end{cases}$$

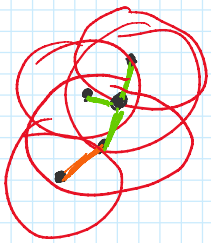
- this gives something <u>like single link</u> <u>but</u> has <u>no</u> <u>hierarchy</u> and <u>is</u> <u>faster.</u>
  - S1 - Define a ball with a fixed radius "r"
  - S2 - Draw the ball around each point and link the points who fall into the ball of other points
  - S3 - Define "degree" as the number of other points in some origin points ball.
  - S4 - Declare "core points" as those points whose degree is above some pre-defined threshold.
  - S5 - Extend the cluster to points within the ball's of non-core points



cluster



can decide to throw out or group by some way.



cluster