CS506 LECTURE 6 (2/16/20)

- Project Deliverable 0 = create "improved" version of the project
  - ↳ go over w/ client
  - ↳ discuss Q's
  - ↳ ID and list limitations/risks in achieving project goals.
  - – revisit throughout project
  - – call them out as soon as you see it

★ Fork repo → go through PR process (see Git workshop)

Unsupervised Learning continued...

└ **Hierarchical Clustering** → goal: hierarchy from each individual point to all the points

└ agglomerative:

overall computationally expensive

1) every point in its own cluster
2) at each step, merge two closest clusters
3) stop when every point is in same cluster

└ Divisive) reverse

1) Everything in the SAME cluster
2) split until everything is in its own cluster

comp. expensive

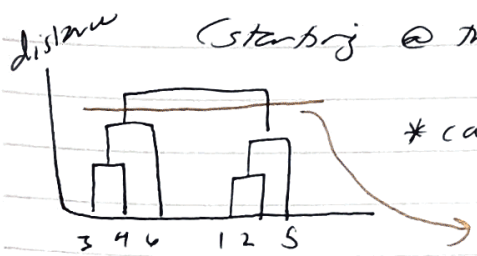
★ Agglomerative

1) each point is its own cluster
2) compute distance between all pairs of clusters → need to define calculating distances b/w clusters
3) merge two closest
4) repeat ~~3~~ 2 & 3 until everything in same cluster

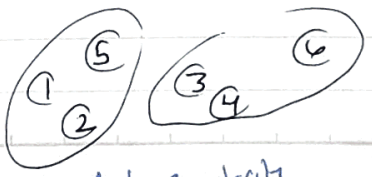
↳ calculating n^2 times (n^2 work = computationally expensive)

★ * at each step, record which clusters merged to produce DENDROGRAM

↳ read from bottom to top (starting @ the leaves)



* can "cut" @ any threshold & it produces clusters



- expected higher order hierarchy
  - ↳ more clearly understand distinctness b/w diff. clusters
- WHERE does it start becoming a different topic

ex. species detection using similarity detection → threshold = threshold for similarity rec of same/separate species

[can we implement this?
└ How can we compute distance between clusters
* distance between the centroids
* distance between closest points
~~* compute pairwise dist~~
* sum all the pairwise distances

$d(p_1, p_2) \rightarrow$ distance b/w points $\quad D(C_1, C_2) \rightarrow$ distance b/w clusters

Single-Link Distance $O(n^2)$

- minimum of all pairwise distances between a point from C_1 to C_2

$$D_{SL}(C_1, C_2) = \min \{ d(p_1, p_2) \mid p_1 \in C_1, p_2 \in C_2 \}$$

depends on our choice of d (p doesn't have to be 2)

pro
↳ (k means CANNOT do this)
└ can diff. large clusters

limitation = sensitive to noise points → because they executing intrinsic of single link def (the minimum)

tends to elongate clusters → distance completely determined by closest (can't really tell what's behind it)

COMPLETE-Link Distance

- max of all pairwise distances possible

pro: less susceptible to noise, more balanced
limitations = sensitive to outliers
splits up large clusters (all tend to have same diameter)

Average-Link Distance

- avg. of all pairwise differences

$$D_{AL}(C_1, C_2) = \frac{1}{|C_1|\,|C_2|} \sum_{\underbrace{p_1 \in C_1,\, p_2 \in C_2}_{\substack{p_1 \text{ is in } C_1 \text{ and} \\ p_2 \text{ is in } C_2}}} d(p_1, p_2)$$

pro: less susceptible to noise/outliers

limitation =

Centroid Distance

- compute distance between centroids

$$D_c(C_1, C_2) = d(\mu_1, \mu_2)$$

Ward's Distance

- difference between spread/variance of points in merged cluster and unmerged cluster

$$D_{WD}(C_1, C_2) = \sum_{p \in C_{12}} d(p, \underset{\substack{\uparrow \\ \text{centroid of 12 cluster}}}{\mu_{12}}) - \overbrace{\underbrace{\sum_{p_1 \in C_1} d(p_1, \mu_1)}_{\substack{\text{spread} \\ \text{of} \\ \text{cluster 1}}} - \underbrace{\sum_{p_2 \in C_2} d(p_2, \mu_2)}_{\substack{\text{spread of} \\ \text{cluster 2}}}}^{\text{compare to values BEFORE merge}}$$

☆ pick cluster w/ smallest variance

→ See example ~~[illegible]~~ walkthrough (~ 210 min)

- maintain distance matrix

- Finding the threshold requires exploration/tuning
  - in General, good to expose a hierarchy in the data

[↳ How to compare outputs of one clustering method to another?

Density Based Clustering

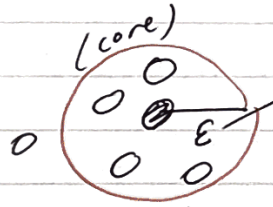
- goal: cluster points that are densely packed together

Need to: define density

* minimize variance
* low mean distance of points in a cluster from centroid

* amount of points per unit distance (like a window sliding along plane
  ↳ fix a specific volume

ex.

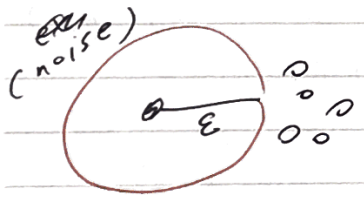


→ defines neighborhood of this point

min_pts = 3
↓
a parameter ... something to tune

→ need to ID if you're @ core of dense region or @ boundary
↳ see what points in the neighborhood are also dense (core vs. boundary points)

densest



points inside < 3 → NOT dense

Core point: ε-neighborhood contains ≥ min_pts

Border point: in the ε-neighborhood of a core point and you're not dense

Noise point: neither core nor border point

*can label entire dataset as core/border/noise → create clusters by connecting core points

~~DBScan~~ DBScan Algorithm

ε and min_pts given:

1) Find ε neighborhood of each point
2) Label pt as CORE if ≥ min_pts → iterate over EVERY point in dataset
3) label points in neighborhood that aren't CORE as BORDER
4) label points as ~~[illegible]~~ NOISE if they are neither CORE nor BORDER
5) For each CORE point, assign to same cluster all CORE points in its neighborhood
6) Assign border points to nearby clusters

pros 1) ID clusters of diff. shapes/sizes
2) resistant to noise

limit = prefer clusters of the same densities → need to set min & ε upfront
↳ if ε too big, only detects very dense clusters ↳ global def of density
↳ fail to ID clusters of varying densities

- see DEMO - (unthr in) → code DBScan algorithm → uploaded to github repo
+ NEXT lecture (2/17)

~~strategy~~:
* put each core point into a cluster

* create Class DBC (self, dataset, min_pts, epsilon) * using top down approach

- get_epsilon_neighborhood → use distance function from sim.py → * need to keep track of INDEXES
↳ list of points

P is index of points in the dataset

* like Dijkstra

- explore_Neighborhood
↳ every core point gives same assignment value (logged in assignments list) → implementing this
↳ need another loop over the P Neighborhood
⟹ need to add points to the queue while we explore it

* if not a core point
↳ give special assignment

.pop() and returns
↳ removes first element

= NextP

note assignment is the cluster label