2/10

Lloyds always converges, but not always at the optimal solution
- centers might be too close
  - L solve by k-means++
    - pick random centers, but each point has pr of being selected proportionate to dist from center (outliers more likely)
    - L $Pr(x \text{ picked}) = (D(x))^2 / \left|\sum_{x_i \in X} D(x_i))^2\right|$ = each point gets (dist to center)^2 # 'entries'

how to choose k:
1. iterate through k to find point of diminishing return
   - means gotta do many attempts (not always possible)
   - *sometimes, lower cost ≠ best use

clustering wants: k-means only focuses on
- similar dp's in same
- dissimilar dp's in different

= small inner cluster width/distance (a), larger distance between clusters (b)

= $(b-a)/\max(a,b)$: $\underbrace{\text{close to 1 when } a \ll \text{ or } b \gg}_{\text{want}}$

Silhouette Score

for each dp i: a_i = avg dist to neighbor in cluster

b_i = smallest mean dist to every point in another cluster

2/11

tags = [$string_1$, $string_2$, $string_3$, ...]

model = SentenceTransformer(<model>) ← LLM

feature_vectors = model.encode(tags) ← {for each tag, converts into array where each index is similarity score to a word in dict

clusters = KMeans(n_clusters=3, n_init=20).fit(feature_vecs).predict(feature_vecs)

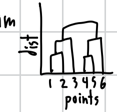
runs Kmeans with 3 centers, 20 times — equal to .fit_predict(feature_vectors)

for KMeans: the only 'randomness' comes from initial center selection
- L same centers = same result
- must use mean distance func to correctly cluster
  - L diff func will do something, but not whats intended

2/12

Heirarchal Clustering: clustering in tiers/threshholds

L outputs dendrogram



agglomerative - start w/ each point is own cluster
- compute dist between clusters
- merge 2 closest clusters
- repeat until ∀ in the same cluster

divisive - start with ∀ in 1 cluster
- at each step, split
- repeat until each point is its own cluster

↓

recursively apply kmeans with k=2

threshold distance isnt very practical

L hard to know what it means

more-so to view data that cant be plotted bc dp's in very high dimensions

*$d(x,y)$ - distance between: points

$D(a,b)$ - clusters

μ_n = mean/avg dp pos in C_n

ways to calc $D(C_1, C_2)$

1. single-length dist: min dist between any points in C_1, C_2
   - L sensitive to noise/in between dp's
   - L creates elongated clustering

2. complete length dist: max dist between any points in C_1, C_2
   - L opposite to SL: handles noise well
   - L sensitive to outliers

3. average-link dist: $\frac{1}{|C_1| \cdot |C_2|} \cdot \sum d(p_1, p_2)$

4. centroid dist: similar to kmeans
   - L pos of C is average position of $\forall p \in C$

5. Ward's Dist: $\sum_{p \in C_{12}} d(p, \mu_{12}) - \sum_{p \in C_1} d(p_1, \mu_1) - \sum_{p \in C_2} d(p_2, \mu_2)$

   centroid

   ↓ merging C_1 and C_2

(spread around mean in merged) - (spread around mean in C_1) - (spread around mean in C_2)

how to:

1. make matrix of all points vs themselves
2. compute distance between all points
3. group together 2 closest points
4. redefine matrix, include combined points as clusters
5. redefine dist to ∀ other points
6. add closest dp to cluster
7. repeat 4-7 until ∀ in 1 cluster

2/12

Density Based

density: 1. take radius of length ε around point
└ if # points within region > threshold: dense ✓

core point: ε-neighborhood contains ≥ <min_points>
border point: in ε-neighborhood of core point
noise point: neither core or border

DFS method: 1. iterate through dataset
2. if dp is core:
- iterate through neighborhood
- if neighbor is core: repeat ↑
  └ else: neighbor is border, add to cluster
- continue iterating through neighborhood
- once no more undiscovered nodes in neighborhood, repeat step 1 with next undiscovered dp in dataset

since *density definition* is defined, wont pick up on differently dense areas = mark as outliers
└ thus makes clusters of the same density