CLUSTERING - unsupervised learning

└ similar points clustered together

- each data point assigned to a specific group where they are ① similar to each other, ② dissimilar to other groups

Goal: define a process that can give us clustering ↑ to optimize this
- what does similar mean? - How do we find clustering?
- How do we know if it's good clustering?

difficulties: ambiguous

Types of Clustering
- Partitional: each object belongs to exactly 1 cluster (k clusters)
- Hierarchical: set of nested clusters organized as tree (cut the tree @ any point → new cluster)
- Density-Based: defined based on local density of points → (need to define density)
- Soft Clustering: each point assigned to clusters w/ a probability

→ Clustering in real world → detecting fraudulent purchases } filtering out outliers
→ filling in NaN values — marked as spam (filter) }

Partitional Clustering

need to specify k upfront = limitation

- given n data points and a k # of clusters
  - partition n data points into k clusters
- given all the ways to distribute n into k buckets
  - which is the best?

if you sum all dissimilarities (ex. using pairwise) ↑ intra-cluster distances should get a small #
↳ good partition = total dissimilarity is small

$$\text{Cost Function} = \sum_{k=1}^{K} \sum_{x_i, x_j \in C_k} d(x_i, x_j)$$

* want to minimize this "cost"

K = sum of all clusters

$x_i, x_j \in C_k$ = for each cluster C_k take distance between x_i, x_j

$\in$ "elements belonging to set"

→ given function distance d can find centroid (center of mass)
↳ when Euclidean
→ mean/average = centroid

Turns out when d is Euclidean → just look @ all distances to the centroid → no need to look @ all pairwise distances

$$\sum_{k}^{K} \sum_{x_i, x_j \in C_k} d(x_i, x_j)^2 = \sum_{k}^{K} |C_k| \sum_{x_i \in C_k} d(x_i, \mu_k)^2$$

- in a given cluster
- $|C_k|$ = scalar = size of C_k = # of points in C_k
- μ_k = center of the cluster
- * optimizing pairwise differences is equivalent to optimizing distance to center

★ **K-means** Given $X = \{x_1 \dots x_n\}$

- find k points $\{\mu_1 \dots \mu_k\}$ that minimize the cost function = sum over all clusters over all points of the d of point to centroid

$$\sum_{i}^{k} \sum_{x \in C_i} \|x - \mu_i\|_2^2 = \sum_{i=1}^{k} \sum_{x \in C_i} L_2(x, \mu_i)^*$$

- minimize distance to centroid
- because it's equivalent to minimizing dissimilarity
- L_2 norm?
- L_2 = euclidean distance
- recall: $p=2$ means euclidean distance
- centroid of C_i → mean (because euclidean)
- ★ Kmeans here

↳ when $k=1$ and $k=n$ → this is easy. Why?

- $k=1$ → one cluster — no other options
- $k=n$ → each point has its own little cluster

limitation ↳ when x_i lives in more than 2 dim (> 2 features), this is VERY difficult (NP-hard)

Trying to optimize → algorithm

K-means → Lloyd's Algorithm

1) Randomly pick k centers $(\mu_1, \dots \mu_k)$
2) assign each point to its closest center → calculate distance to all centers & choose smallest
3) compute the new centers as means of each cluster → computing mean of points in a cluster generate new center
4) repeat 2 & 3 until convergence
   - when the mean = center (need ALL means to stop moving)
   - ↳ the points assigned are the same that you get when computing new one! ↳ center doesn't move

Will this algorithm always converge? Proof by contradiction

↳ only a finite # of partitions → only a finite # of ways we can pick to set

1) min. cost function is only reached in the limit (infinite # of iterations) = IMPOSSIBLE, we have finite # points

[Does it always converge?

Proof by contradiction → 1) Suppose it does NOT converge: then...

* minimum of cost function only reached in limit (infinite iterations)

⇒ IMPOSSIBLE (finite # of points) → finite n, and k
∴ finite # of double sums

OR

* algorithm gets stuck in the loop/cycle

⇒ IMPOSSIBLE (@ each step, we lower the cost
if old ≠ new, cost has improved
if old = new, cost hasn't changed

* cannot have a cost that is "smaller than itself"

[But does it converge to the OPTIMAL solution?

↳ NOT always

Solution = (ex.) Run Lloyd's algorithm multiple times ✓ choose result w/ lowest cost

(Limitation: bad results because of randomness

Solution = (ex.) Random Initialization (different initialization methods)

① Farthest First Traversal

- pick the next center to be farthest from all previous ~~pts~~

1) pick a random first one
2) ↺

limitation = outliers in dataset cause issues → choose outliers as centers and then all the actual data classified as outliers

* Need to combine "Random" & "Farthest First Traversal"

② K-Means ++

- Initialize w/ a combo of the methods
1) start w/ random center
2) Let D(x) be distance between (x) and centers selected so far (x) = possible candidate for a center (within a cluster)
choose next center w/ prob. proportional to $D(x)^a$ → the distance
when a=0 (random)
~~weighted probability~~
- x selected based on weighted probability as a function of where that point's distance to what's picked so far
↳ points @ greater distance = higher probability
when a=∞ (farthest first traversal)
a=2 (k-means ++)
- squared distance

* trying to get farthest first traversal probabilistically

[How do you pick points w/ probability proportional to the distance?

→ Suppose given a random number generator (black box)
– generates uniform random number between 0 and any N

↳ How do you use this to select points w/ prob. proportional to $D(x)^a$ $a = 2$

$a = 2$
which of x, y, z do we pick?
(w/ prob. proportional to the distance)

* $D(x)^2 = 3^2 = 9$
$D(y)^2 = 2^2 = 4$
$D(z)^2 = 1^2 = 1$

* write out the distances (note: don't have to be int distances)

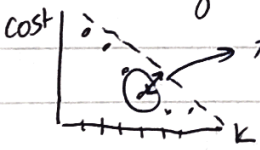
* assign x to 9 buckets in random # generator → generate a # between 0 and N
y to 4 buckets → it necessarily falls into one of the buckets!!
z to 1 bucket
↓ the prob. of falling into x bucket is HIGHER

$N = D(x)^2 + D(y)^2 + D(z)^2 = 14$

* if black box can only generate numbers (0,1) → divide by N

Limitations + prefers clusters of similar size
not good w/ clusters of different densities
tries to find globular shapes (fits in a circle) → a big ball in N-D space

• How to choose the right k? (# of centers)
1) Iterate through different values of k (elbow method) ← computationally expensive



find something in the elbow of the curve
(maximize distance to inter reference

2) Use empirical / domain-specific knowledge of the data
– Is there a known approx. distribution? (k-means good for → spherical gaussians)

k-Means Variations

1) k-medians (L_1 norm / manhattan distance)
2) k-medioids (use any distance function + centers in the dataset) → ALWAYS
3) Weighted k-means (each point has different weight when computing means)