Clustering

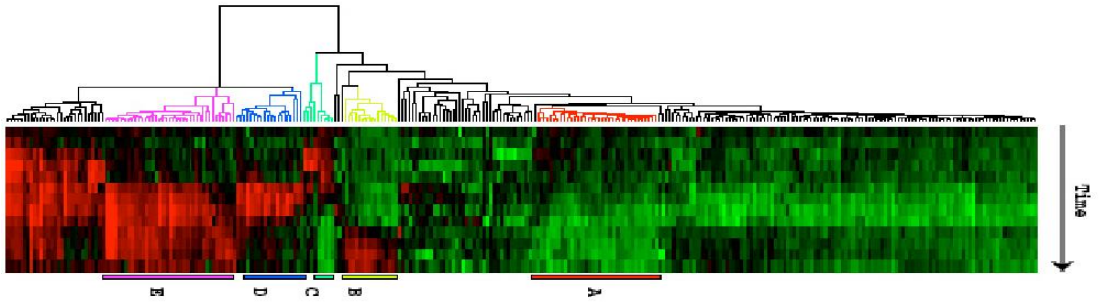
13-01-2025

Clustering

- Unsupervised learning
  - When your data doesn't have labels

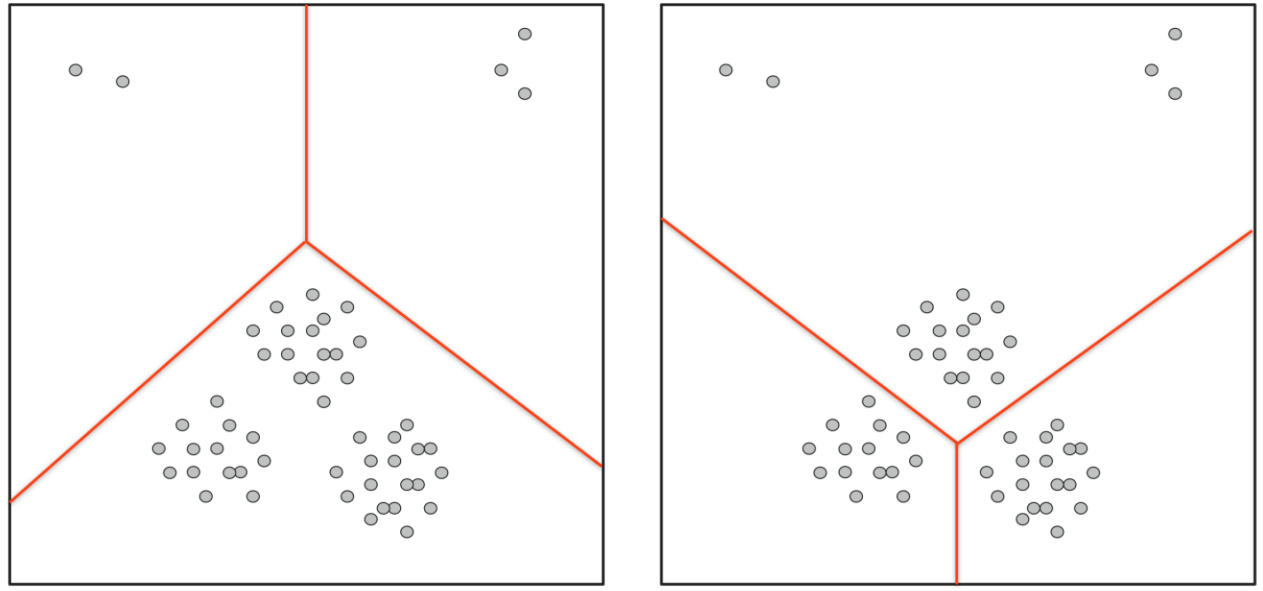
[Image segmentation via clustering, James Hayes]

- Useful for
  - Detecting patterns e.g. in image data, customer shopping results, anomalies…
  - For optimizing, e.g. distributing data across various machines, cleaning up search results, facility allocation for city planning…
  - when you "don't know" what is it exactly that we are looking for



Clustering: basic idea

- Grouping objects into small number of meaningful groups
  - How to define similarity / distance between objects?
  - What is meaningful?
  - How many groups?
  - Typically there is no supervision



[Picture from Ackerman talk]

Clustering framework: distance function

- In the object representation we need an appropriate distance function
  - Lp norms for vectors
  - Jaccard distance for sets
  - Edit distance for sequences
  - Divergences for probability distributions…
- Typically, nice to have the metric properties
  - $d(x,x) = 0,\ d(x,y) \geq 0$
  - $d(x,y) = d(y,x)$
  - $d(x,y) + d(y,z) \geq d(x,z)$
- Also nice if it is easy to calculate "average"

$$\min_x \sum_i d(p_i, x)$$

$$l_p(x,y) = \left(\sum_i |x_i - y_i|^p\right)^{1/p}$$

$$JD(A,B) = 1 - \frac{|A \cap B|}{|A \cup B|}$$

$$\in [0,1]$$

#edits in going from one Seq to another

Distance function: lp norms

- L2 norm/Euclidean distance

- L1 norm
- L-infinity norm
- Easy to calculate averages
- Also related is cosine distance

$$d_{cos}(x, y) = 1 - \frac{x^t y}{|x|_2 |y|_2} = 1 - \cos(\theta_{x,y})$$

Next step: objective function

- Specifying number of clusters
  - K-means / K-median
- Specifying cluster separation / quality
  - e.g. radius of cluster, Dunn’s index,..
- Graph based measures
- Working w/o an objective function
  - Hierarchical clustering schemes

K-center

Cost of a point = distance to the closest cluster center

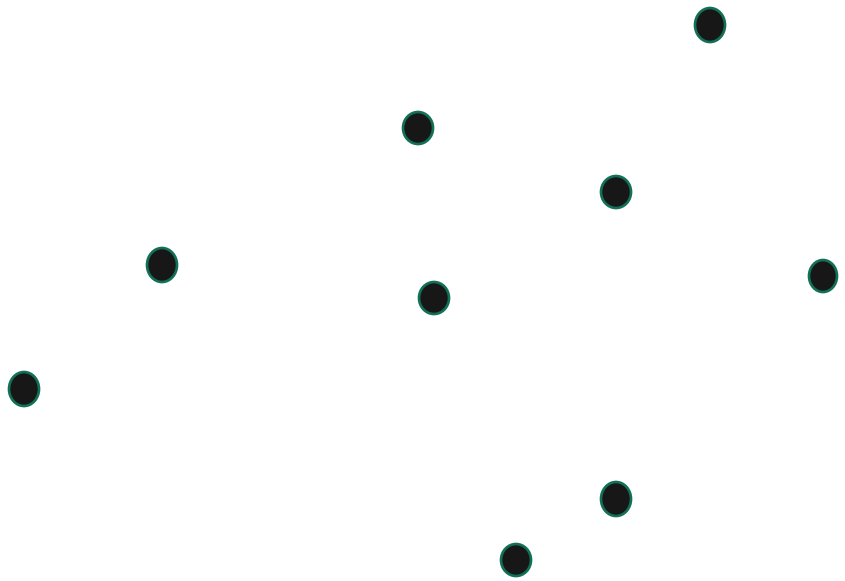
Cost of a single cluster = maximum of point costs

Cost of a clustering = maximum of cluster costs, say D

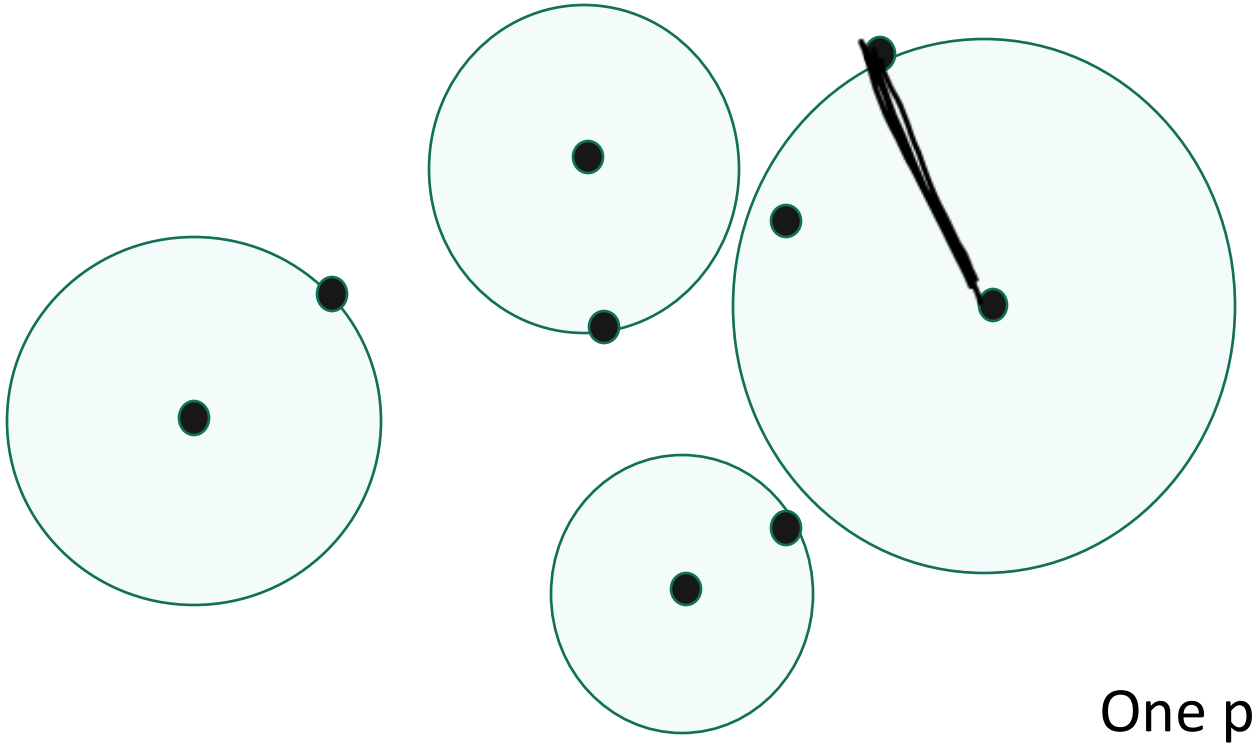
Find k-centers so that the above cost is minimized

Geometrically, we are trying to cover the point set by k balls of radius D and trying to minimize D

K-center



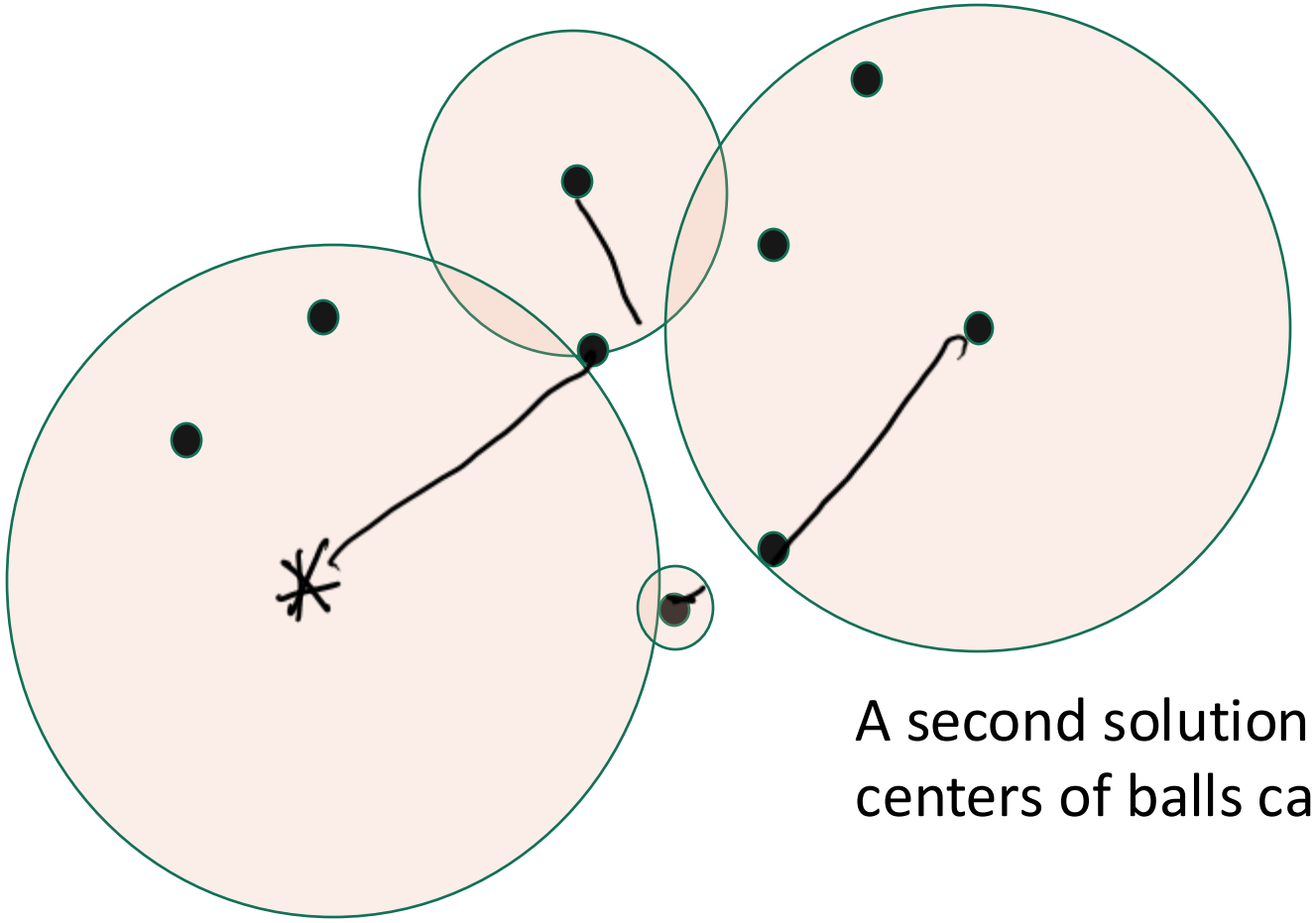
K-center



One possible solution– here centers are points in the input themselves

Value of solution is given by which point, i.e. who is the “unhappiest”?

K-center



max radii

A second solution – notice that the centers of balls can be arbitrary

You can use upto k balls

Do you think this is optimal?

K-center – few observations

- We need to “cover” the input with at most k balls
  - Algorithm gets to choose the centers of the balls
  - All points must be covered by the balls
  - Solution value is given by the largest radii among the balls used

- If we are allowed k balls, is it possible that we get a better solution by using $k-1$ balls only, rather than k?

Few more observations

- NP hard !!
- Easy 2-approximation
- What does 2-approximation mean?
  - If the optimal solution using at most k balls needs a max-radius OPT, then our algorithm will have a max-radius $2 \times OPT$

2 - Approx

- Choose 1st center arbitrarily.
- For each remaining center,
  - choose point farthest from chosen centers.
  - till k centers are chosen.

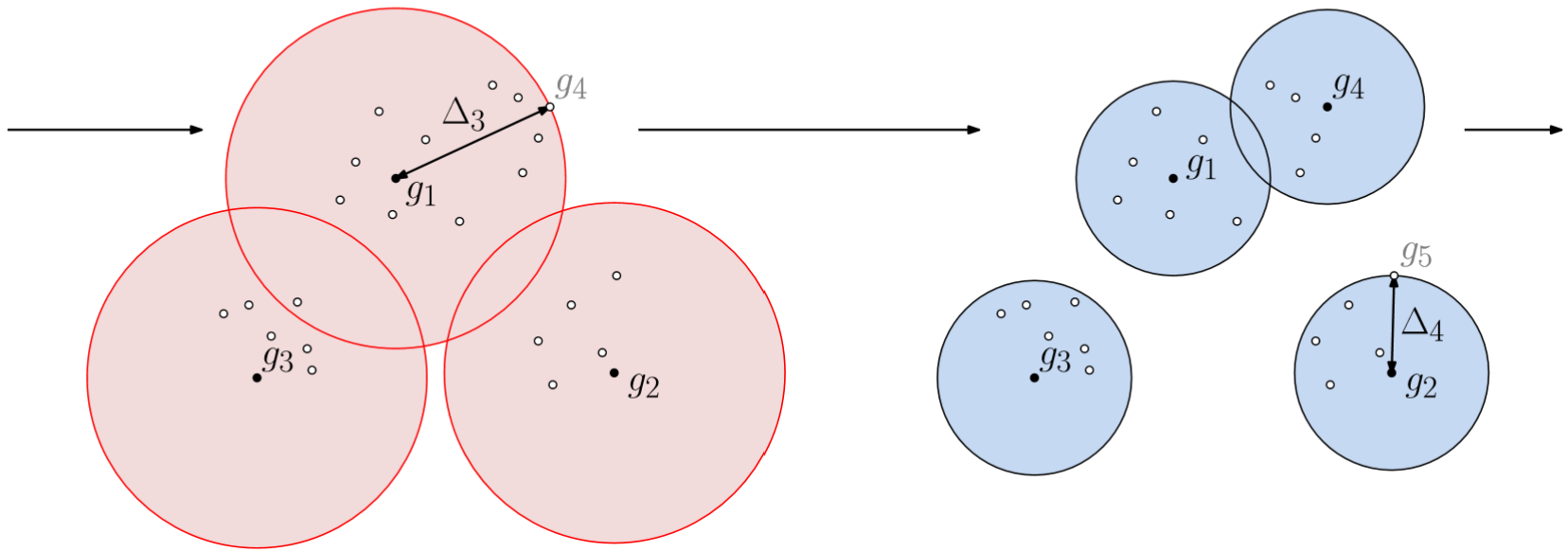
Runtime ?

Why 2-approx ? $D_{algo} \leq 2 D_{opt}$

Let $g_1 \cdots g_k$ = k center chosen in order

g_{k+1} = farthest point from $\{g_1 \cdots g_k\}$

$G_i = \{g_1 \cdots g_i\}$.



$k = 3$

$k = 4$

G_i = first i centers $= \{g_1 \cdots g_i\}$

$\Delta_i = \max_x d(x, G_i)$ i.e. farthest from G_i

Claim 1: $\Delta_{i+1} \leq \Delta_i$ Which & why?

Note: Δ_k = cost of the greedy solution.

$\Delta_k = \max_x d(x, G_k)$.

$d(x, G_i) = \min_{g \in G_i} d(x, g)$

G_i = first i centers; $g_{k+1} = \arg\max_x d(x, G_k)$.

$\Delta_i = \max_x d(x, G_i)$ i.e. farthest from G_i

Claim 1: $\Delta_{i+1} \leq \Delta_i$

As $G_{i+1} \supseteq G_i$!

$\forall x,\ d(x, G_{i+1}) \leq d(x, G_i)$

G_i = first i centers; $g_{k+1} = \arg\max_x d(x, G_k)$.

$\Delta_i = \max_x d(x, G_i)$ i.e. farthest from G_i

Claim 1: $\Delta_{i+1} \leq \Delta_i$

Claim 2: Every pair of centers in G_i is seperated by $\geq \Delta_i$, $i \leq k$

$$\Delta_{i+1} = \underset{x}{\operatorname{argmax}}\, d(x, G_i) \leq \underset{x}{\operatorname{argmax}}\, d(x, G_{i-1})$$

$$= \Delta_i$$

Claim 2: For any two centers g_i & g_j $j < i$.

$$d(g_i, g_j) \geq d(g_i, G_{i-1}) = \Delta_i$$

Putting these together, for any i, j with $\Delta_1 \geq \Delta_2 \geq \cdots \geq \Delta_{i+1} \geq \Delta_k$

$$d(g_i, g_j) \geq \Delta_{\min(i,j)} \geq \Delta_k$$

$$d(g_{k+1}, g_i) \geq d(g_{k+1}, G_k) = \Delta_k$$

Claim 1 $\Delta_{i+1} \leq \Delta_i$

$$\Delta_{i+1} = d(g_{i+1}, G_i) \leq d(g_{i+1}, G_{i-1})$$

$$\leq d(g_i, G_{i-1})$$ [Since g_i was chosen by algo]

$$= \Delta_i$$

Claim 2 :

Note that for any two i, j, $j < i$

$$d(g_i, g_j) \geq d(g_i, G_{i-1}) = \Delta_i$$

By putting these together, for any i

- consider G_{i+1}, and $g_x, g_y \in G_{i+1}$, $x < y$.

$$d(g_y, g_x) \geq d(g_y, G_{y-1}) = \Delta_y \geq \Delta_{i+1}$$

$g_x \in G_{y-1}$

Hence, when we consider G_{k+1} all pairs of points in G_{k+1} are separated by Δ_k

Claim 3: For any set of k centers O,

$$\Delta(O) \geq \frac{\Delta(G)}{2} = \frac{\delta_k}{2}$$

Showing this is enough to claim 2-approx ?

Pf: Induction!

Claim 3: For any set of k centers O,

$$\Delta(O) \geq \frac{\Delta(G)}{2}$$

Pf: every x lies within $\Delta(O)$ of O.

$\Rightarrow$ every $x \in G$ " " " "

$|G_{k+1}| = k+1$ $\Rightarrow$ $\exists$ two $x, y \in G_{k+1}$ that have same centers in O.

Say x, y both mapped to c.

$\Rightarrow \max\big(d(x,c), d(y,c)\big) \leq \Delta(G)$

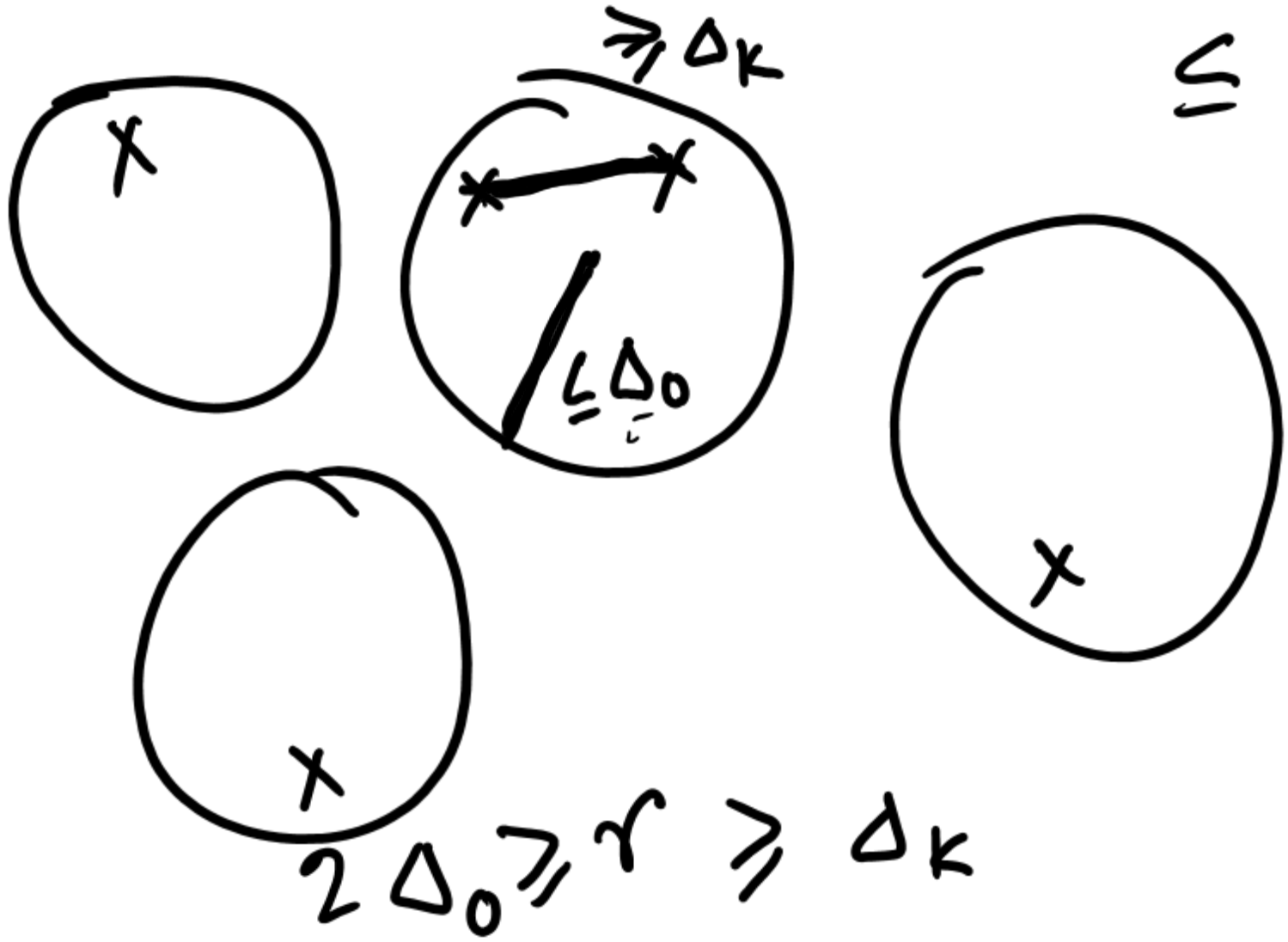
Since $x, y \in G_{k+1}$, $d(x,y) \geq \Delta_k = \Delta(G)$.

$\Rightarrow \Delta(G) \leq d(x,y) \leq d(x,c) + d(y,c)$

$\Delta(G) = \Delta_k$

$\leq \Delta(G) + \Delta(G)$ (Claim 2)

$= 2\Delta(G)$. !


$\geq \Delta_k$
$\leq 2\Delta_0$
$\leq \Delta_0$
$2\Delta_0 \geq r \geq \Delta_k$

$\Delta_k \leq 2\,\Delta_0$

Δ_k ↓ algo soln

Δ_0 = OPT

algo soln ≤ 2 OPT

Can we do better?

- We cannot get a $2 - \epsilon$ approximation, for any constant $\epsilon > 0$, unless $P = NP$!

- If the distance function does not obey triangle inequality, then we cannot get any algorithm for any arbitrary constant factor approximation, unless $P = NP$
  - For specific distance functions, for specific datasets, we might get good heuristics

K-means

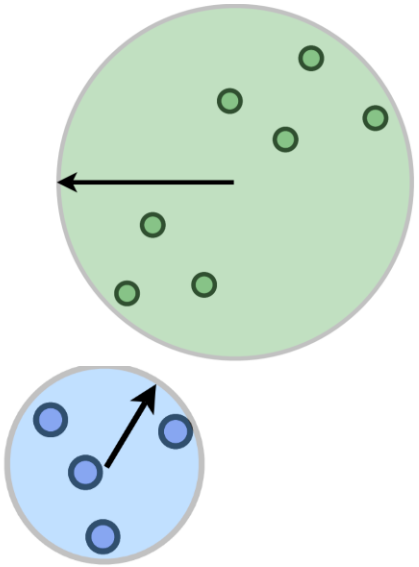
- Distance function is typically L2
- C = $\{c_1, c_2, \ldots, c_k\}$, cost(C)= $\sum_x \min_{c_x} d(x, c_x)^2$
- Find C to optimize the above cost
  - Leads to a natural partitioning of the data
- Large amount of work, both from theory & data mining community
  - Great example of divergence between theory and practice and how that prompted new research directions for both

square is important

d( ) ≡ Euclidean distance

k-means objective: alternate view

- Define “best” k-clustering of the data by
  - minimizing the variance of each cluster
    - The mean $c_i = \frac{1}{|C_i|}\sum_{x \in C_i} x$ is the “expected” location of a point
    - Hence variance of $C_i = \sum_{x \in C_i} ||x - c_i||^2 \frac{1}{n_i}$



[slide from Sergei V. & Suresh V.]

Connection with max likelihood estimation

- Take the Gaussian pdf. Consider the likelihood of the set of points if the center is μ and co-variance matrix is $\sigma^2 I$

$$p(x_1, x_2, \ldots x_k \,|\mu, \sigma) \propto \prod \exp(-|x_i - \mu|^2/\sigma^2)$$

$\propto \exp\left(\sum |x_i - \mu|^2 / \sigma^2\right)$

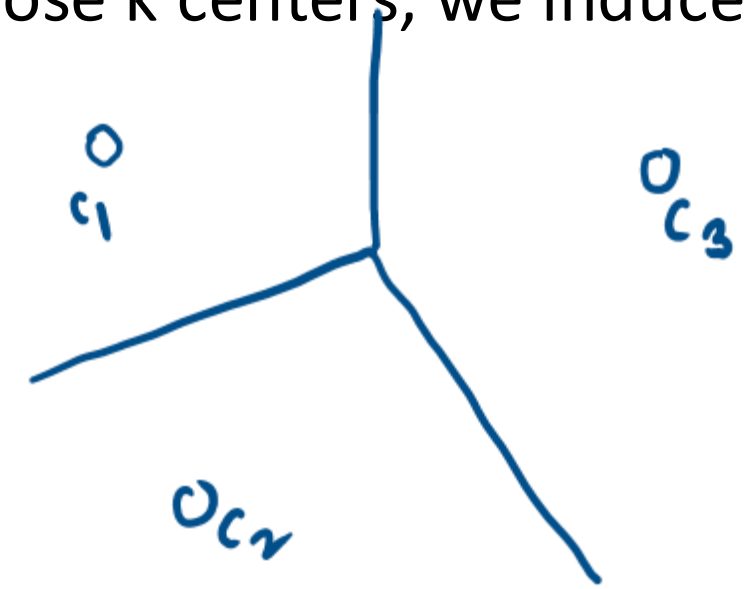
- Taking the log of the probability likelihood

$$\log(p(x_1, \ldots x_k \,|\mu, \sigma) \propto \sum_i |x_i - \mu|^2 + function(\sigma, dimension)$$

k-means cost of cluster assigned to μ.

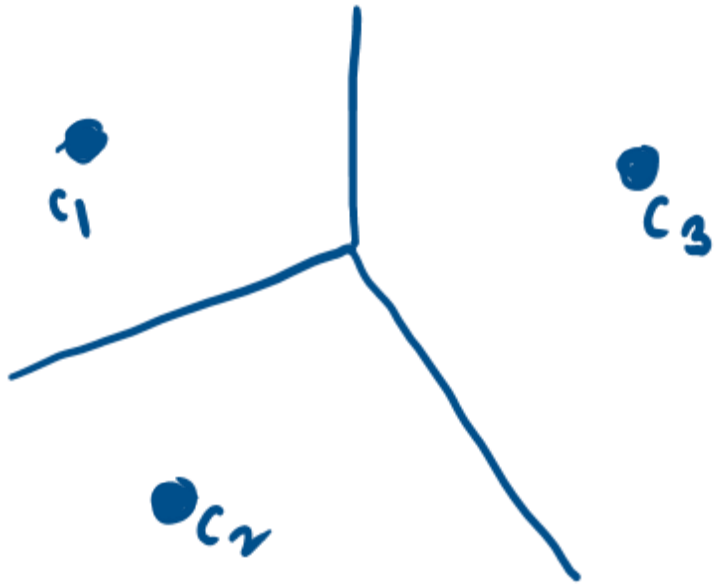
Center vs partitioning

- When we choose k centers, we induce a partitioning of the space



- This is also called Voronoi partitioning. How many possible partitions?

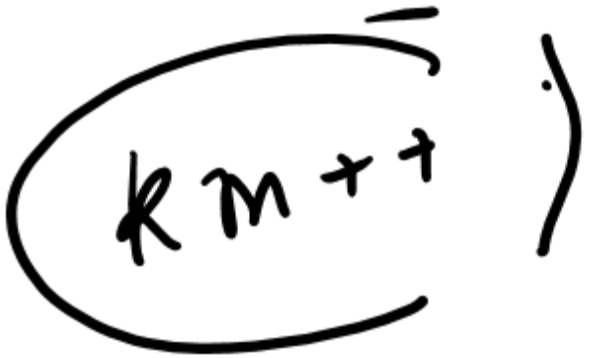
Center vs partitioning



- OTOH, if we consider a Voronoi partition, that suggests an “optimal” center for that partitioning. Why?

The canonical algorithm: Lloyd's algorithm

- Iterative algorithm
- Iterate
  - Find current centers of partitions
  - Assign points to nearest centers
  - Recalculate centers



Typically, this algo is referred to as "k-means"!

Lloyd's algorithm

- Iterative algorithm
- Iterate
  - Find current centers of partitions
  - Assign points to nearest centers
  - Recalculate centers

- Stopping criteria
  - when no (or small #) points change cluster
  - when cluster centers don't shift much
  - All of these are heuristics

Lloyd's algorithm: analysis

- k centers, N points, d dimensions
- Time taken to calculate new cluster assignments : O(kNd)
- Time taken to calculate new centers : O(Nd)
- Number of iterations?

Lloyd's algorithm: convergence?

- For any current clustering, consider the objective function

$$\text{cost(C)} = \sum_x \min_{c_x} d(x, c_x)^2$$

c_x = center closest to x

The algorithm's state is then completely captured by the current partitioning into the k clustering

Question to ponder: Is it possible for the cost to fluctuate up and down over the running of the algorithm, e.g. it goes from 1000 to 100 and then back to 1000?

If not, why not?

Lloyd's algorithm: convergence?

- For any current clustering, consider the objective function

$$\text{cost(C)} = \sum_x \min_{c_x} d(x, c_x)^2$$

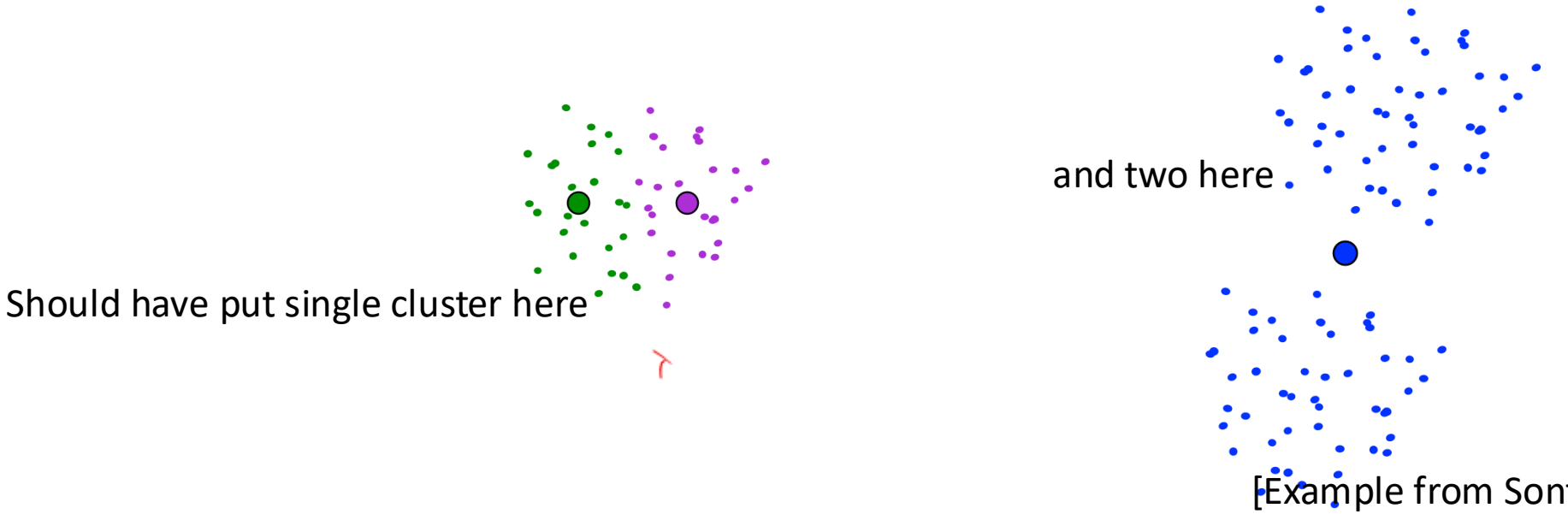
c_x = center closest to x

- At every step of the algorithm, this potentially decreases. Why?

Convergence?

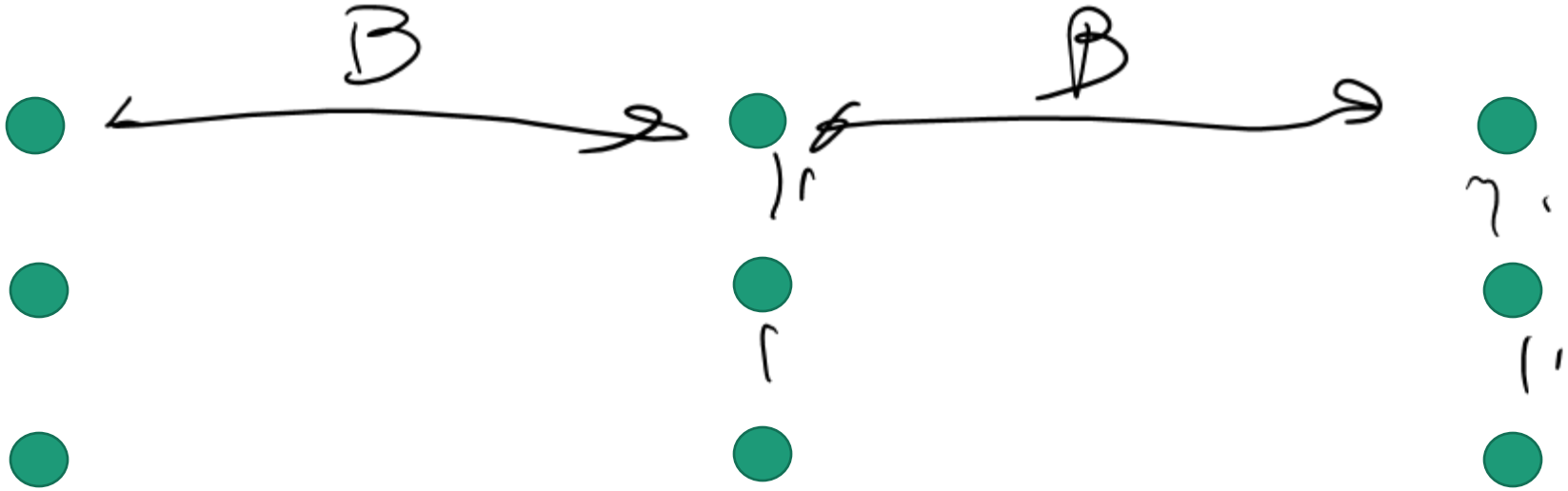
- It is known that in some datasets, Lloyd's algorithm can take exponential $(2^{\sqrt{n}})$ number of steps
  - These tend to be unrealistic

- Bigger problem is where it converges to--- depends on initialization



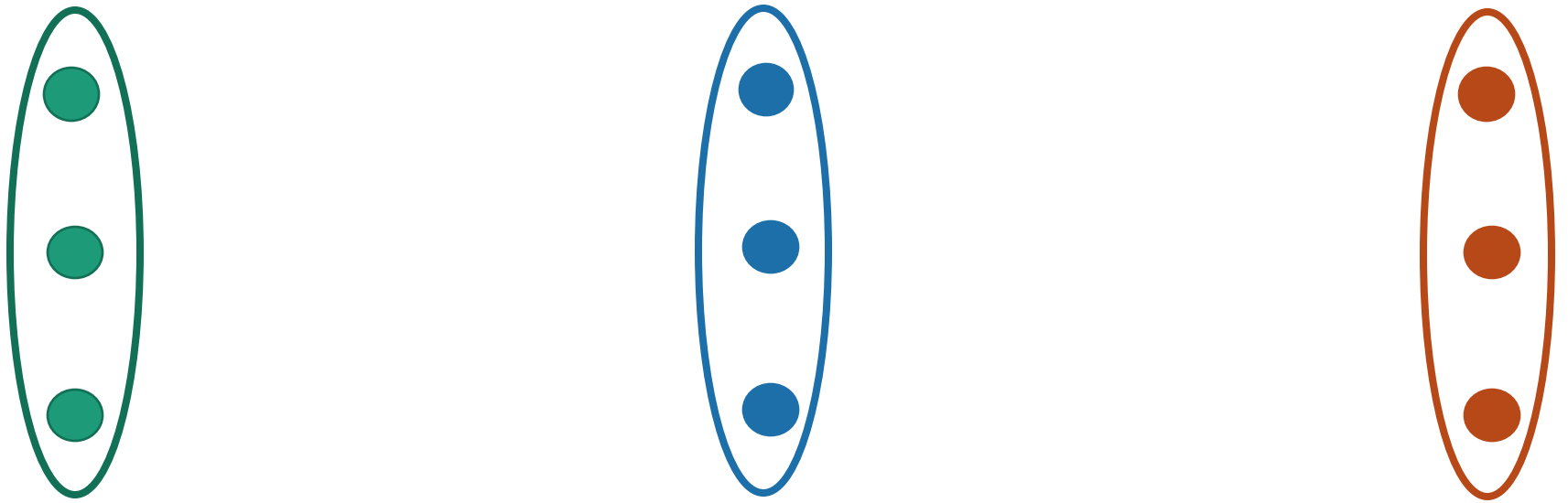
Example

- What could happen here?

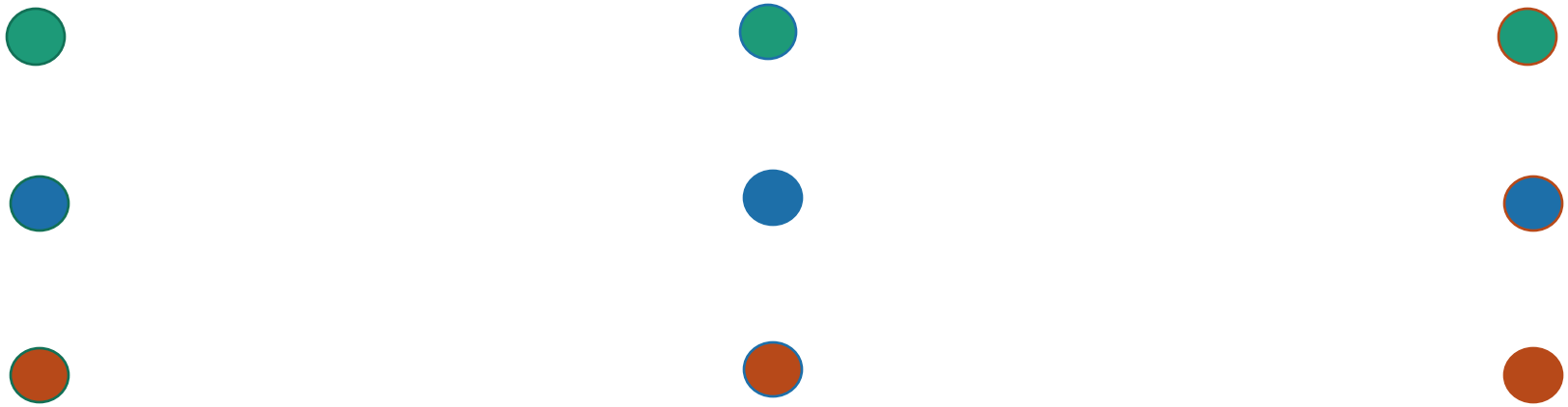


[Example from Sontag]

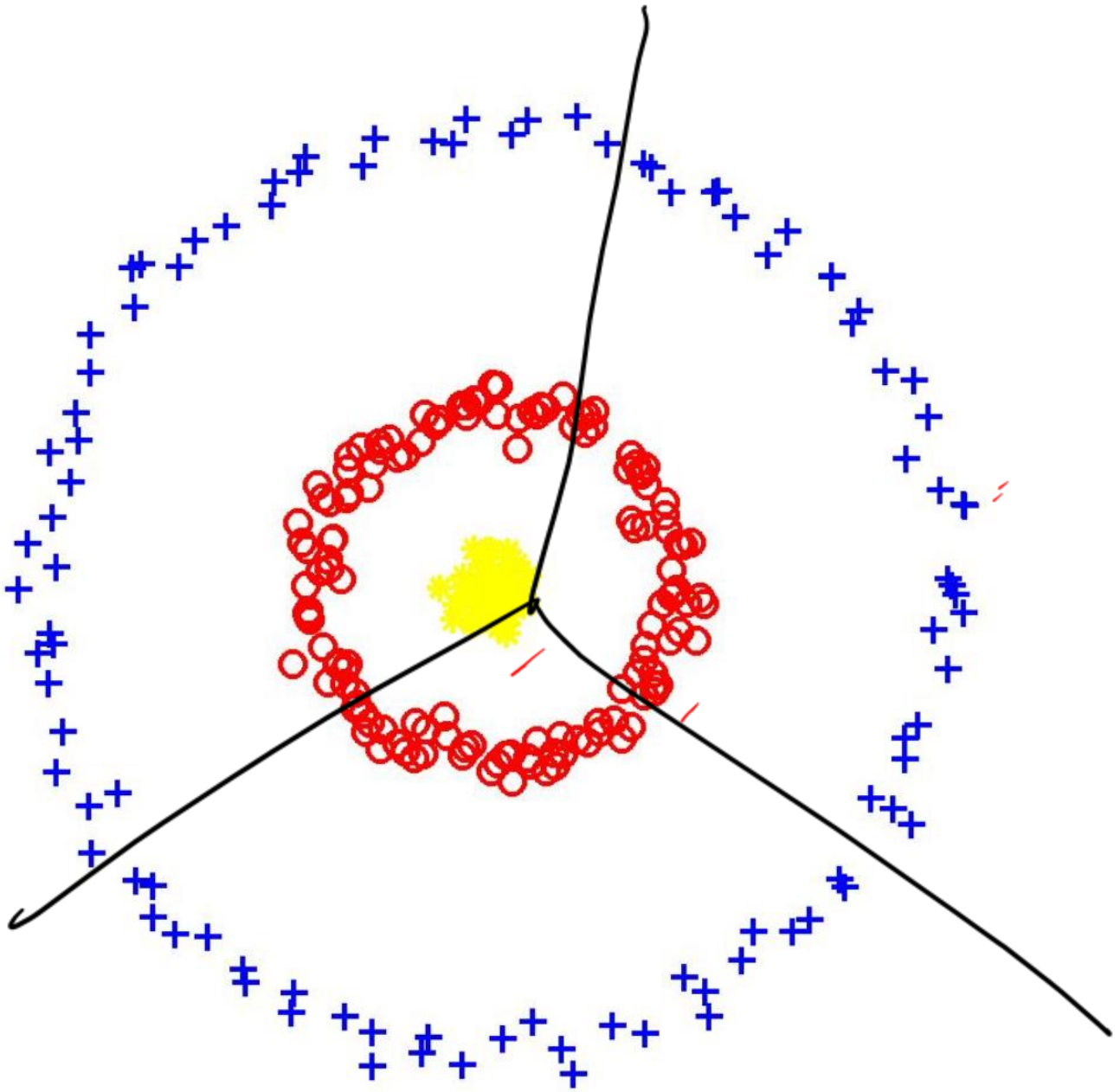
Example: good center choice



Example: bad center choice

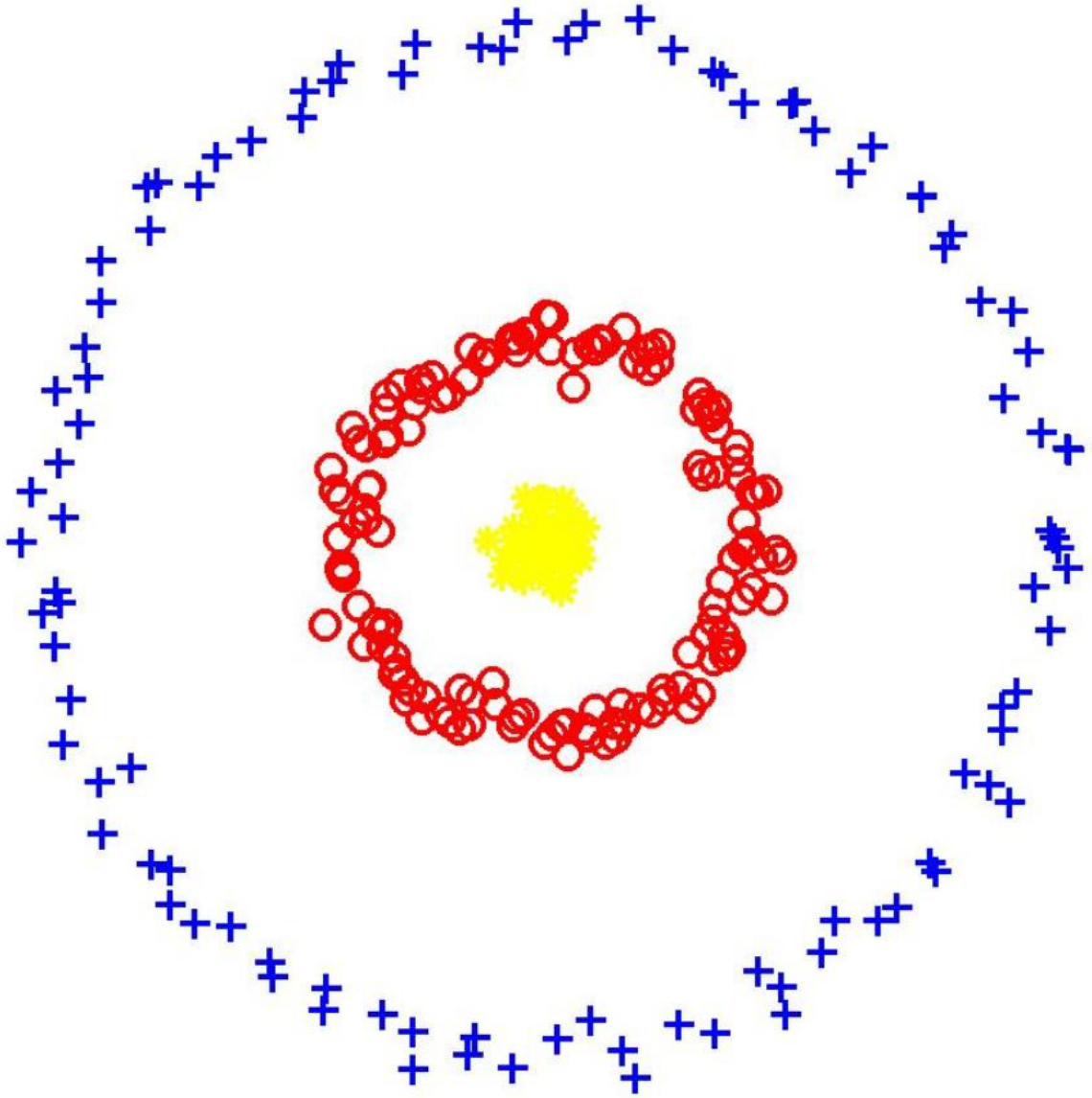


Here?



Here?

Center based clustering is not an appropriate notion here. We will come back to these

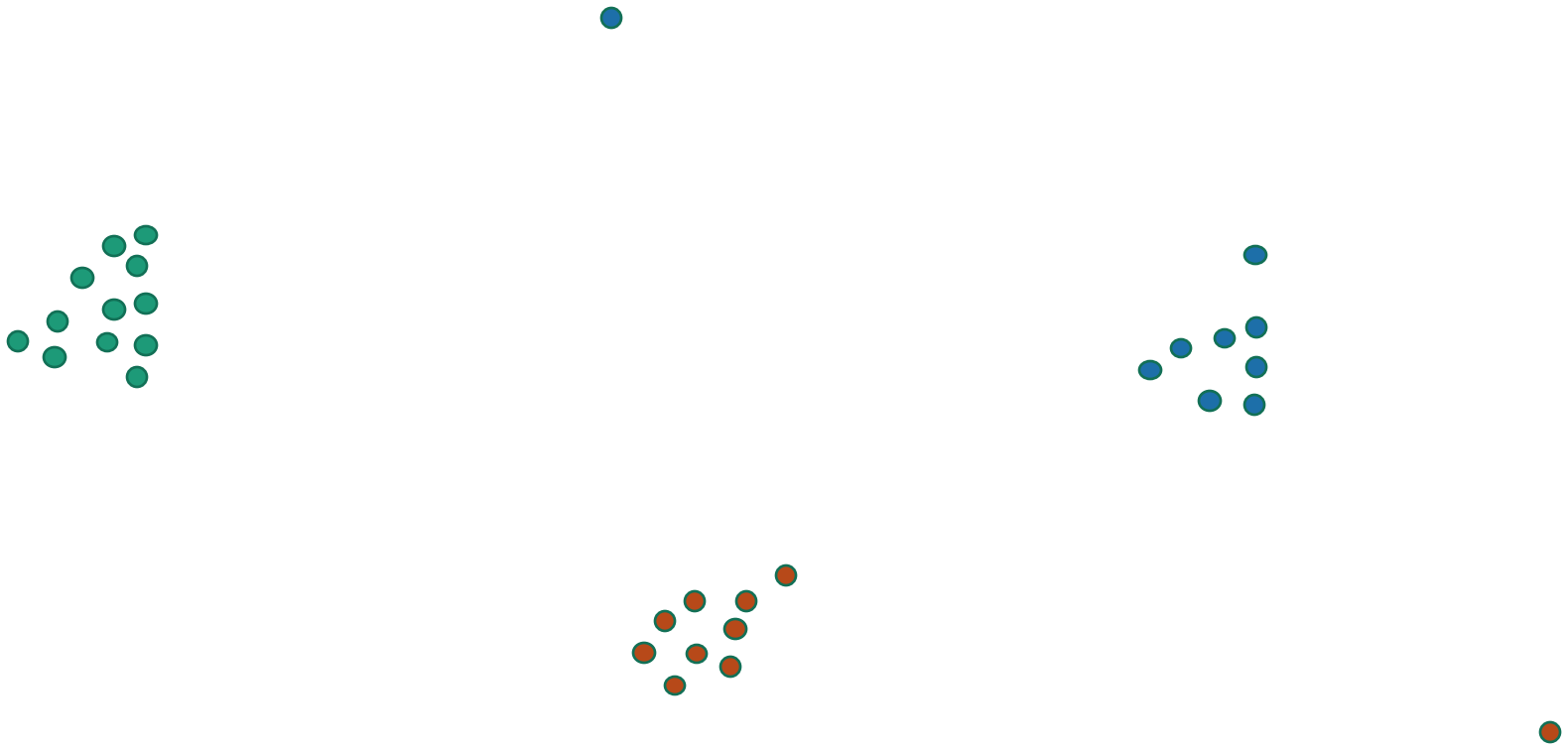


How to initialize

- What if we initialize using farthest points?
  - Choose the next center to be the farthest point from the existing set of centers

How to initialize

- Choosing the centers as furthest is very sensitive to outliers



How to initialize?

- What if we choose the next center uniformly at random?

How to initialize?

- What if we choose the next center uniformly at random?

Nice visualization

- https://www.naftaliharris.com/blog/visualizing-k-means-clustering/

Can we mix the different strategies?

- We want:
  - If there are number of points that are far away from already chosen centers, the next center should be more inclined towards them
  - If there is a “large cloud” of points anywhere, near or far, that should also bias the next center choice
  - Can we achieve both by assigning each point a “probability of being chosen”

kmeans++

- Define $D_i(x)$ as the distance of the point x from the current set of chosen centers.
- Choose point x with probability $\frac{D(x)^\alpha}{\sum D(y)^\alpha}$

kmeans++

- Define $D(x)$ as the distance of the point x from the current set of chosen centers.
- Choose point x with probability $\frac{D(x)^{\alpha}}{\sum D(y)^{\alpha}}$

- $\alpha = 0$: uniform at random
- $\alpha = \infty$: furthest point
- We will use $\alpha = 2$: kmeans++

Arthur, Vassilivitiskii, 06.
Original Lloyd's algo 1957

kmeans++

- Note: after you add another center, the distances $D(x)$ need to be updated i.e. everyone's probabilities change

- So center initialization algo:
  1. Choose first center arbitrarily
  2. While $|C| < k$:
     1. Update $D(x)$ for all points x
     2. Choose a point according to $\{ D(x)^2 / \sum_x D(x)^2 \}$ and add to C

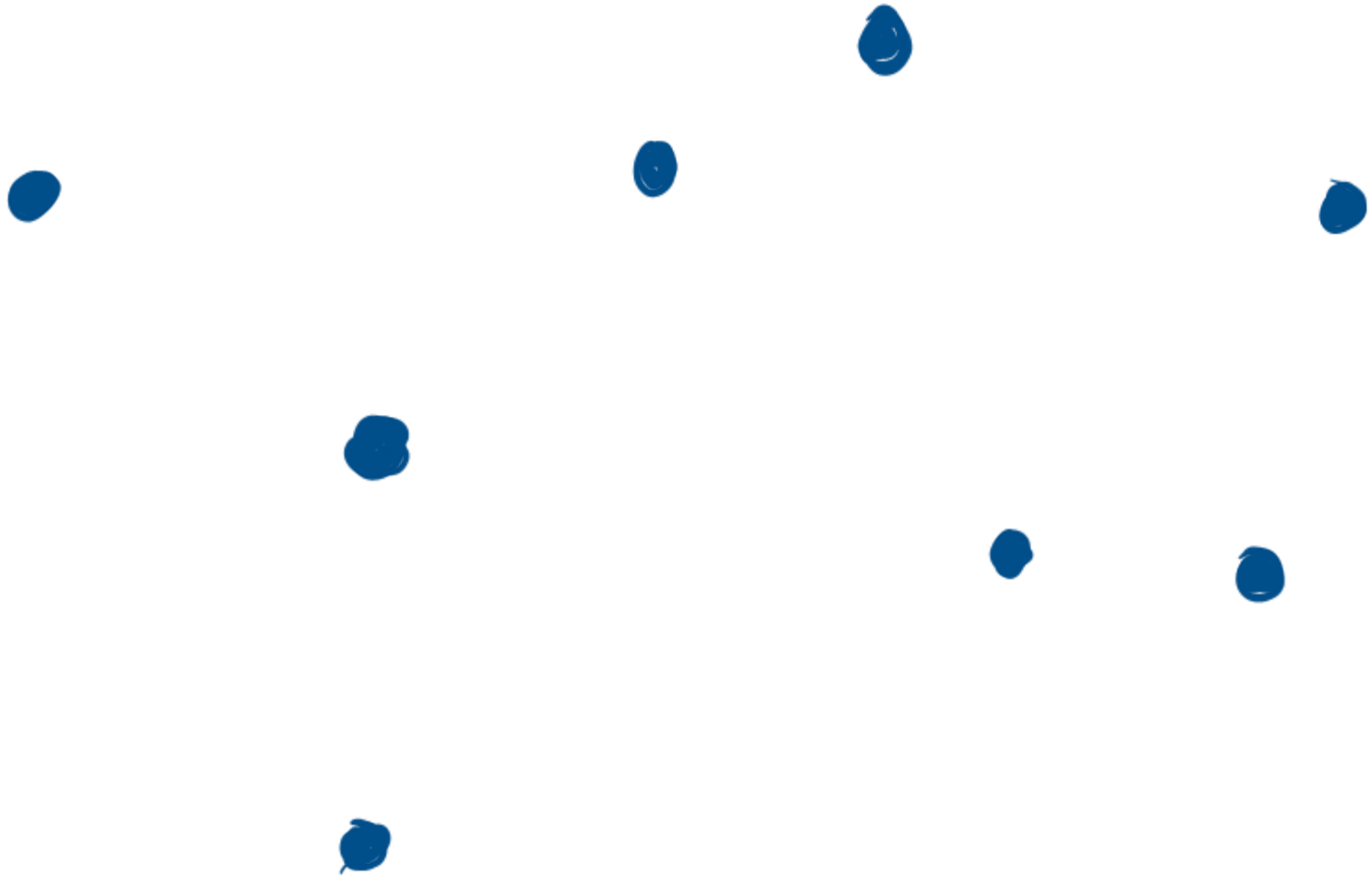
kmeans++

- Note: after you add another center, the distances $D(x)$ need to be updated i.e. everyone's probabilities change

- So center initialization algo:
  1. Choose first center arbitrarily
  2. While $|C| < k$:
     1. Update $D(x)$ for all points x
     2. Choose a point according to $D(x)^2 / \sum_x D(x)^2$ and add to C

- After this run couple of Lloyd's iterations, even one iteration is enough

Example



What is the benefit?

Theorem (AV07): The algorithm always returns a $\Theta(\log k)$ approximation.

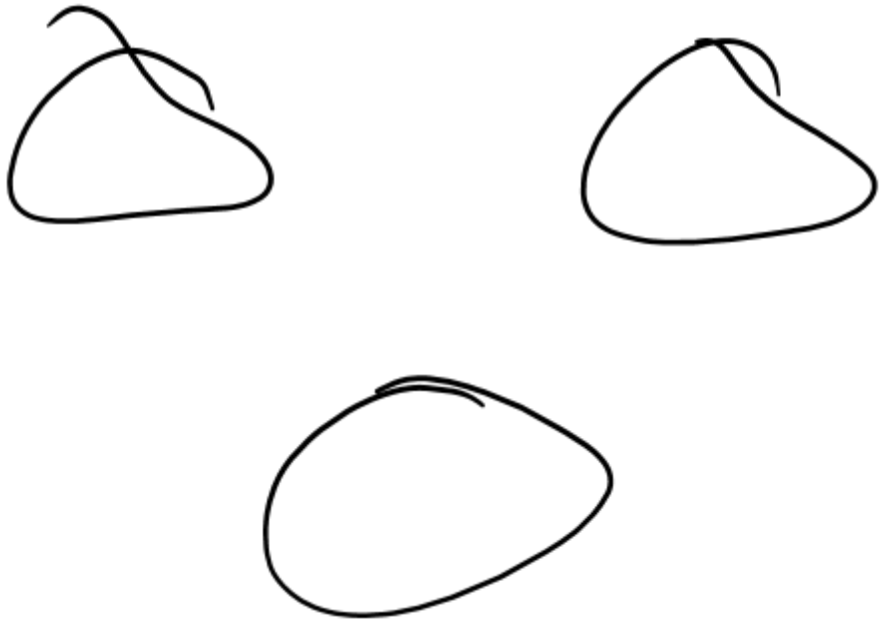
What is the benefit?

Theorem (AV07): The algorithm always returns a $\Theta(\log k)$ approximation.

Theorem (ORSS06): A slightly modified version of this algorithm attains O(1) approximation if the data is "*nicely clusterable*" with k clusters.

What is “nicely clusterable”?

- Dataset X is “nicely clusterable” with k clusters if going from k-1 to k clusters reduces the cost significantly



What is “nicely clusterable”?

- Dataset X is “nicely clusterable” with k clusters if going from k-1 to k clusters reduces the cost significantly

$cost^*(X, k)$ = optimal cost with k cluster.

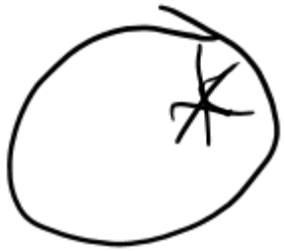
X is (k, ϵ) separated if

$$cost^*(X, k) \leq \epsilon^2 cost^*(X, k-1) \qquad \epsilon < 1$$

Why does kmeans++ work?

- Intuition:

- Consider a cluster C that OPT uses. If we select a point from there then that cluster is covered pretty well.



Why does kmeans++ work?

- Intuition:

- Consider a cluster C that OPT uses. If we select a point from there then that cluster is covered “pretty well”.
  - i.e. cost incurred by our center is with some constant * cost incurred by OPT

- If we select two points from the same OPT cluster, it must be because the contribution of other clusters to the OPT cost is small.

Why does kmeans++ work?

- Intuition:

- Consider a cluster C that OPT uses. If we select a point from there then that cluster is covered pretty well.
- If we select two points from the same OPT cluster, it must be because the contribution of other clusters to the OPT cost is small.

- If the data is already well separated, mostly the first event happens!

What is the benefit?

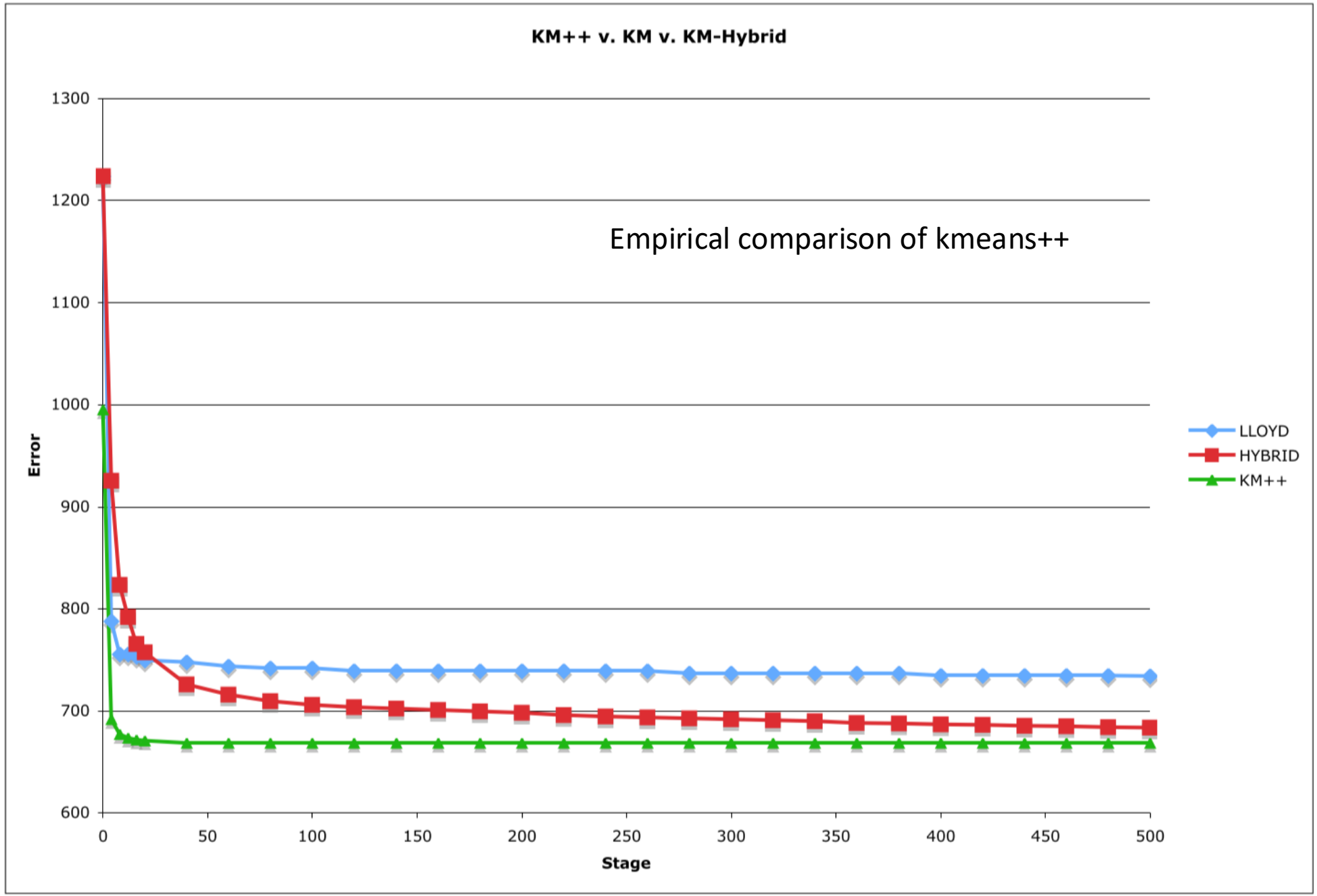
Theorem (AV07): The algorithm always returns a $\Theta(\log k)$ approximation.

Theorem (ORSS06): A slightly modified version of this algorithm attains O(1) approximation if the data is "*nicely clusterable*" with k clusters.

Runtime?

- Theoretical bound?
  - how much time to select the k-centers?

$y = k\text{-means cost}$



How to run kmeans on large dataset?

- Assume that we have a large data and access to multiple machines.

- Can we parallelize the $O(nkd \times \ iterations)$ runtime?

Parallelizing kmeans

- Partition the data
  - $X_1, X_2, \ldots\ldots, X_p$ be the partitions

Parallelizing kmeans

- Partition the data
  - $X_1, X_2, \ldots., X_p$ be the partitions

- In parallel compute a clustering for each X_i
  - $\{C_1^i, \ldots.. C_k^i\}$ be the clustering, define $w_j^i = |\ C_j^i\ |$

Parallelizing kmeans

- Partition the data
  - $X_1, X_2, \ldots\ldots, X_p$ be the partitions
- In parallel compute a clustering for each X_i
  - $\{C_1^i, \ldots\ldots C_k^i\}$ be the clustering, define $w_j^i = |\, C_j^i \,|$
- Consider the cluster centroids as points with weights w_j^i and then recluster them

How good is the solution?

- Suppose the algo in phase 1 gave a β approx. solution and the algo in phase 2 gave γ approx. solution to its (smaller) input

Theorem (GNMO00, AJM09): Overall we get a $4\gamma(1+\beta)+2\beta$ approx

How good is the solution?

- Suppose the algo in phase 1 gave a β approx. solution and the algo in phase 2 gave γ approx. solution to its (smaller) input

Theorem (GNMO00, AJM09): Overall we get a $4\gamma(1+\beta)+2\beta$ approx.

Setting $\beta = \gamma = O(\log k)$ would give a $O(\log^2 k)$ approx

Summary of kmeans

- Kmeans used to be one of the algorithms in which there was a vast gap between theory and practice

- Was used to drive new research in theory, which gave new algorithms useful in practice
  - Also work in *smoothed analysis* that goes beyond worst case

Shape of clusters

k-means / k-median is based on intuition that clusters are like "balls in high dim"

Easy to seperate colors using single linkage, not so using k-means.

k-median

$$C = \{c_1 \dots c_k\}$$

$$d(x, C) = \min_{c \in C} d(x, c)$$

$$\min_{C} \sum_{x \in X} d(x, C)$$

$d(x, y) = \|x - y\|_1$

– Lloyd's algo for this distance function for k-median problem

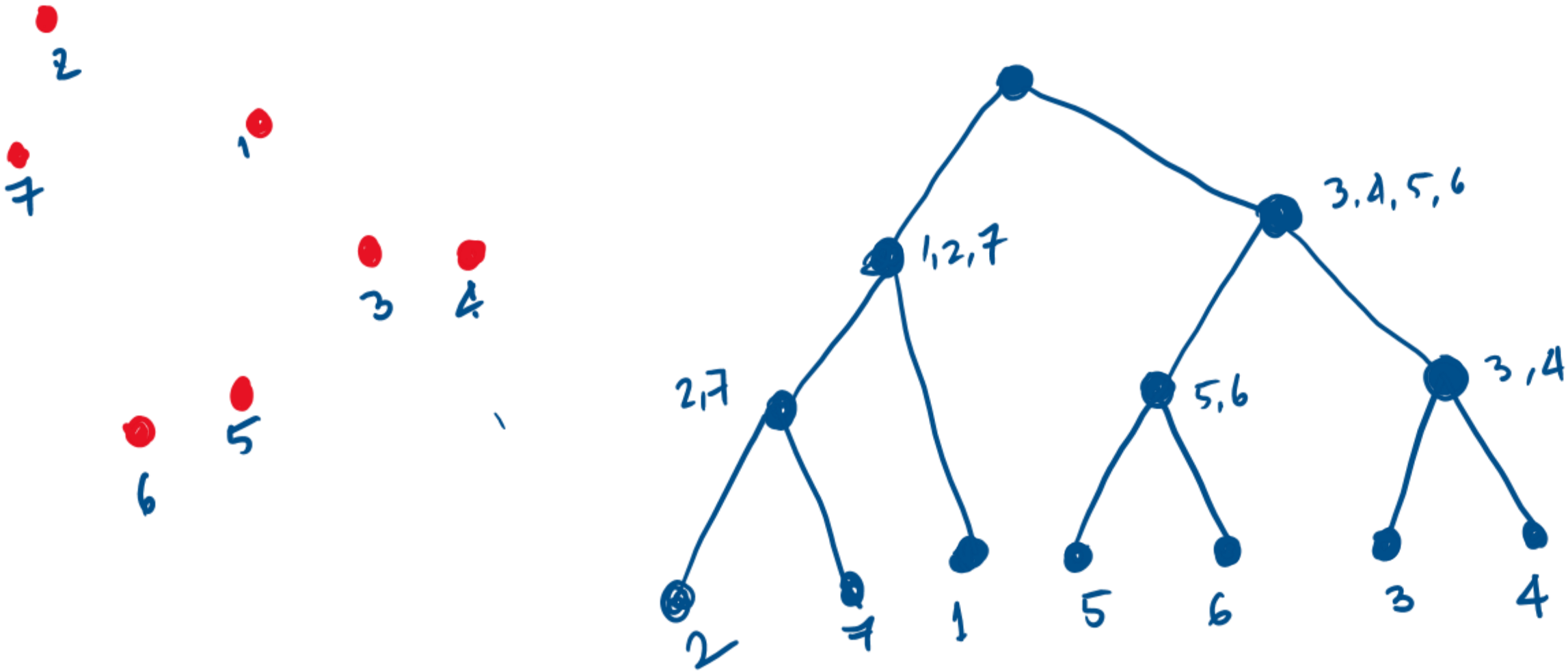
$$\min_z \sum_i \|x_i - z\|_1$$???

$z_1 = \text{median}(x_{i1} \cdots x_{n1})$

$z_d = \cdots \quad (x_{id} \cdots x_{nd})$

Hierarchical clustering

- If we do not know the target number of clusters, what can we do?
- Can we produce a family of clusterings, for each k?
- Two ways:
  - Top-down/divisive: keep on creating bi-partition e.g. applying 2-means
  - Bottom up/agglomerative: start from individual point and keep on aggregating


2
1
7
3
4
6
5
1,2,7
3,4,5,6
2,7
5,6
3,4
2
7
1
5
6
3
4

Agglomerative clustering

- Intuition: keep merging closest set of data points, building larger clusters out of smaller ones

Algo:

Keep a current list of clusters

Initially, each point in its own cluster

while #clusters > 1:

choose closest pair

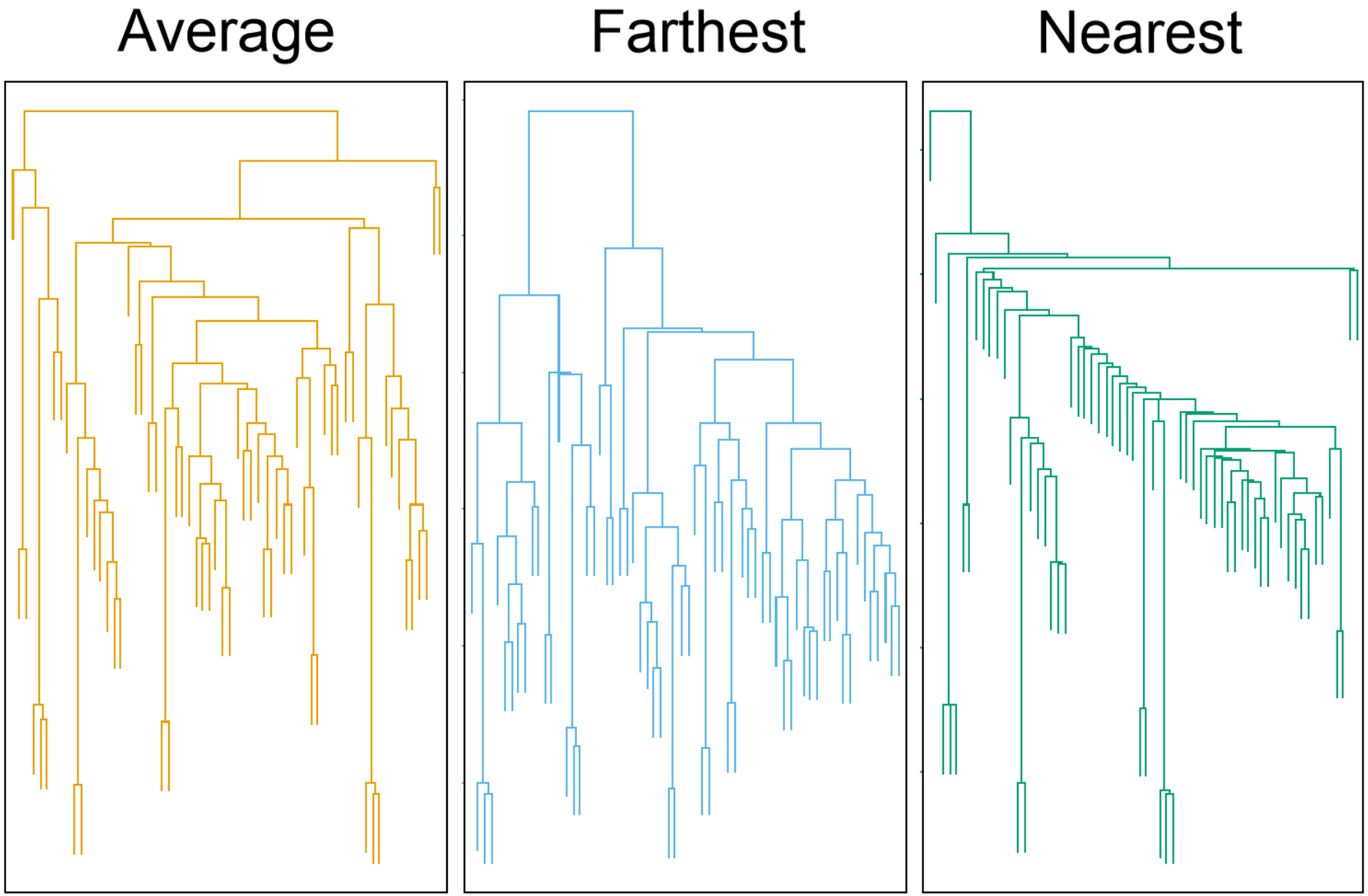
merge them,

update list by deleting these and adding new cluster

How to define closeness?

- Defining closeness for two clusters is not obvious

- Many ways
  - closest pair: single linkage clustering – $d_1(C, C') = \min_{x \in C, y \in C'} d(x, y)$
  - Furthest pair: complete link clustering -- $d_2(C, C') = \max_{x \in C, y \in C'} d(x, y)$
  - Average of all pairs : average link -- $d_3(C, C') = avg_{x \in C, y \in C'} d(x, y)$



Mouse tumor data from [Hastie *et al.*]

Questions

- Is there any principle?
  - Would be nice to have an objective function
- What is the complexity?

Questions

- Is there any principle?
  - Would be nice to have an objective function
- What is the complexity?
- Can we parallelize this?

Few properties

- All the three linkage algorithms have the following property --- there are no “inversions”
  - This means that the dissimilarity scores between merged clusers only increases as we run the algorithm

- The algorithms can work with any dissimilarity measure, not necessarily metric

Objective function?

Can we understand the single linkage algo in terms of an objective function?

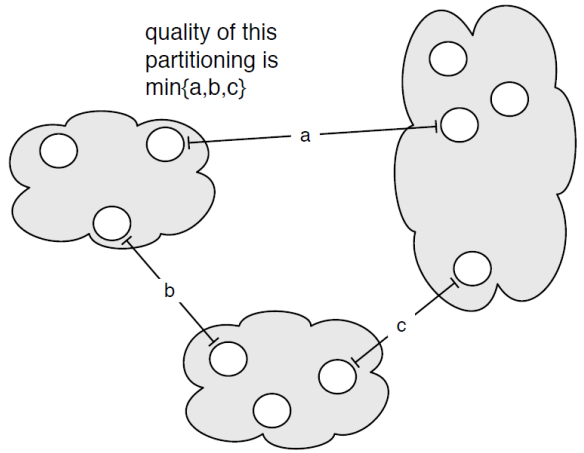
I.e. what kind of clusters is it trying to create?

Single Linkage Clustering: properties

- Suppose we want to create cluster $C_1, C_2, .. C_k$ that maximizes minimum inter-cluster distance

$$maximize\ cost_k(C)$$

$$cost_k(C) = \min_{c_i, c_j} \mathrm{d}(\mathrm{C_i}\ \mathrm{C_j})$$



Single Linkage Clustering: properties

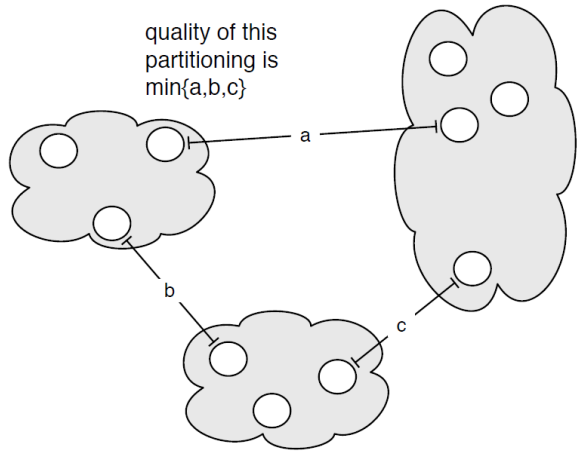
- Want to create cluster $C_1, C_2, .. C_k$ that maximizes minimum intercluster distance

$$maximize\ cost_k(C)$$

$$cost_k(C) = \min_{C_i, C_j} \mathrm{d}(\mathrm{C_i}\ \mathrm{C_j})$$

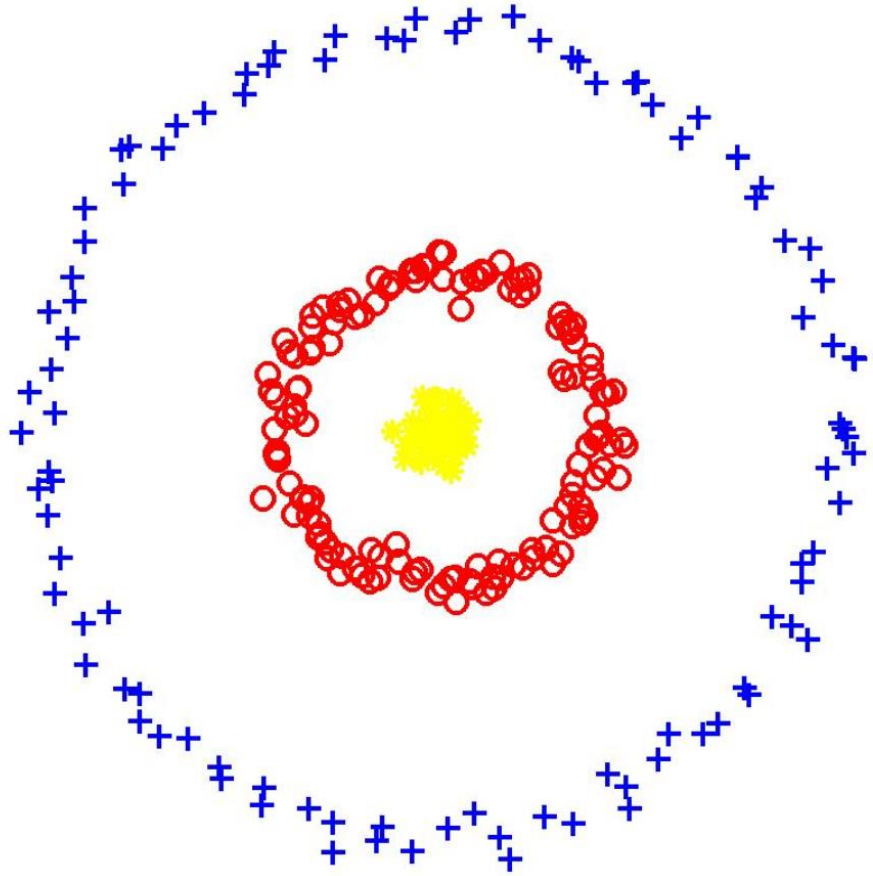
Algorithm -- to get k such clusters, remove last

k-1 links from single linkage dendogram



Single Linkage Clustering

- Is actually the same as Euclidean Minimum Spanning Tree!
- Can handle this specific example nicely, unlike center-based clustering



Linkage based Clustering

- Effective in low-dimensions.
- Not so for higher ($\Leftarrow$) curse of dimensionality.
- Typically needs $O(n^2)$ space.

The bad news

Suppose we take the k-center objective.

Q: If we use single/avg/complete linkage and stop at k-clusters, how badly do they do? And which does better than others

[all in worst case settings]

The bad news

Suppose we take the k-center objective.

Q: If we use single/avg/complete linkage and stop at k-clusters, how badly do they do? And which does better than others

A: Single L. can be factor k worse than OPT.
Avg/Complete can be $O(\log k)$ worse than OPT.

Bad example

Points are all in a line.

$d(x_i, x_{i+1}) = 1 - i\varepsilon$ for some $\varepsilon << \frac{1}{n^2}$

What does single linkage do?

What is optimal k-center?

Bad example

$d(x_i, x_{i+1}) = 1 - i\varepsilon$ for some $\varepsilon << \frac{1}{n^2}$

Points are all in a line.

Cost of OPT k-center $\leq n/k$.

Single linkage creates a cluster out of $n-k$ node $\Rightarrow$ cost $\approx n-k$.

Such chains are a real issue with single linkage.

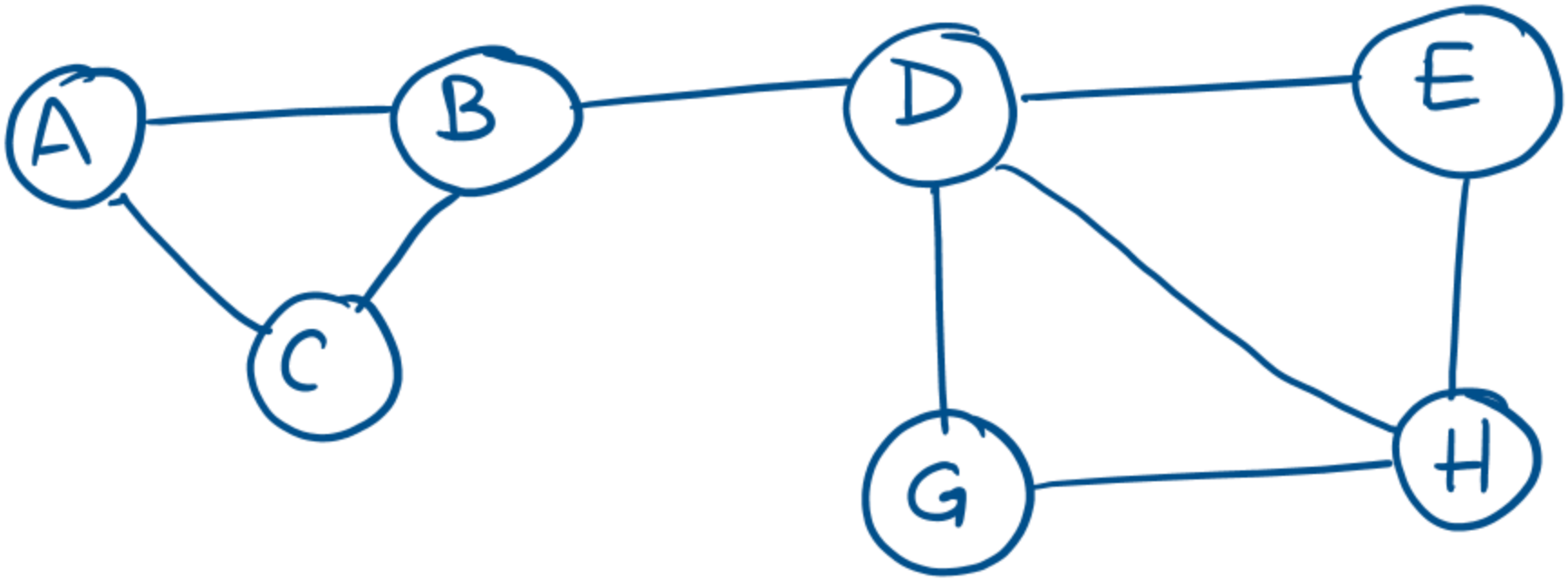
Data as graphs

Graphs: often ideal way of representing data that captures interactions.

E.g. Nodes = users., Edges ≡ who follows whom on FB.

Nodes = cities, Edges = roads / train lines / air routes connecting them

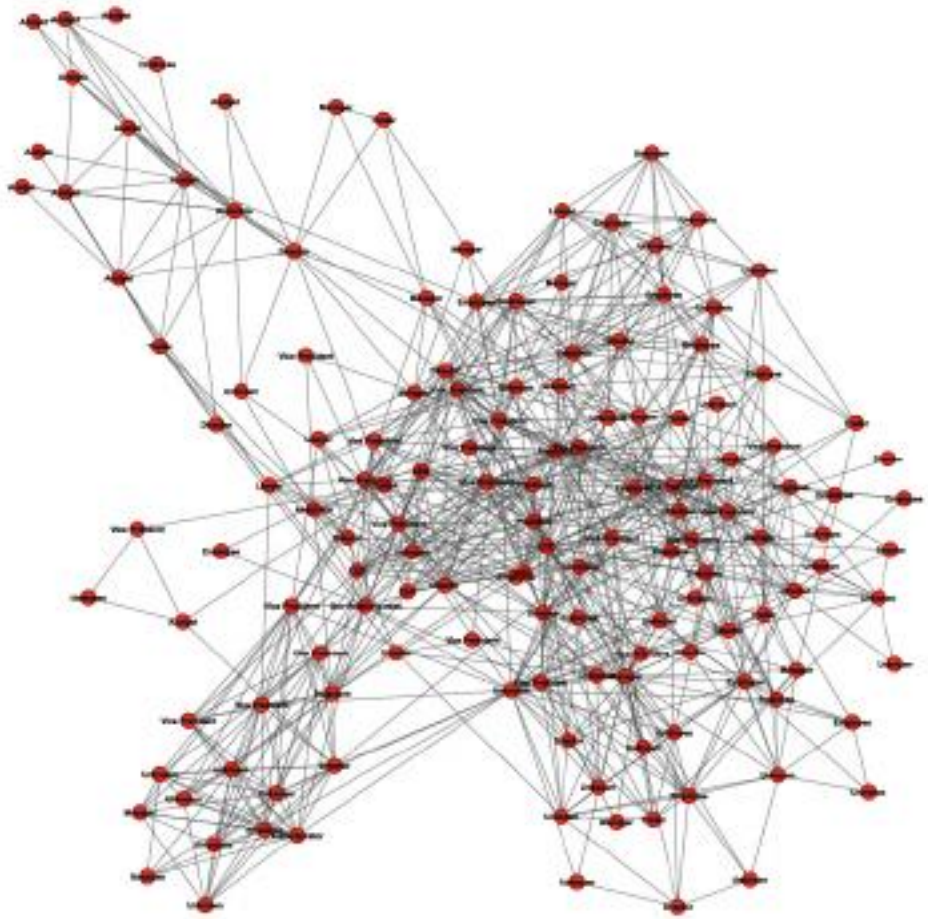
Nodes = genes/proteins Edges = interaction.



Undirected graph $\leftrightarrow$ symmetric relationships.

We could have directions, weights, timestamps.

Enron email network

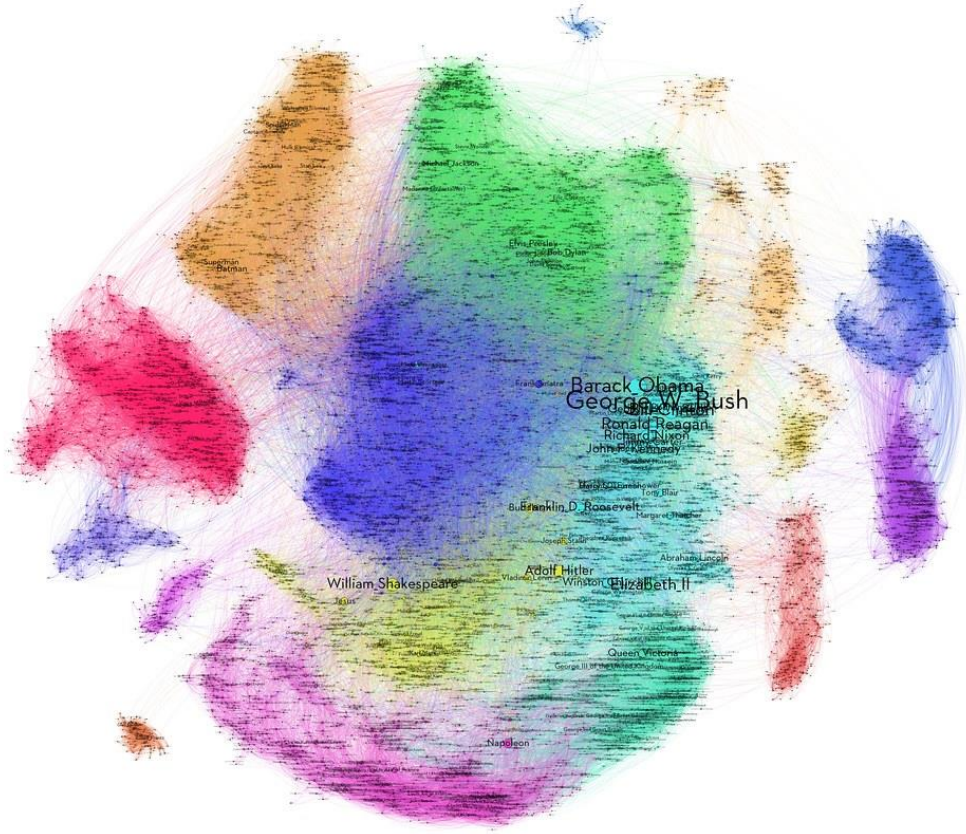


Nodes = employees of Enron

Edge = whether / how many email exchange.

[Image from Cambridge Intelligence]

Wikipedia people pages



https://www.hackdiary.com/2012/04/05/extracting-a-social-graph-from-wikipedia-people-pages/

Clustering

Clustering remains an important problem.

Applications: Finding out gene / protein groups,

- Communities, used in friend recommendations.
- VLSI layout, planning of infrastructure.
-

But…

We paid that clustering needs a distance measure?

What distance measure can we take here?!

Small world

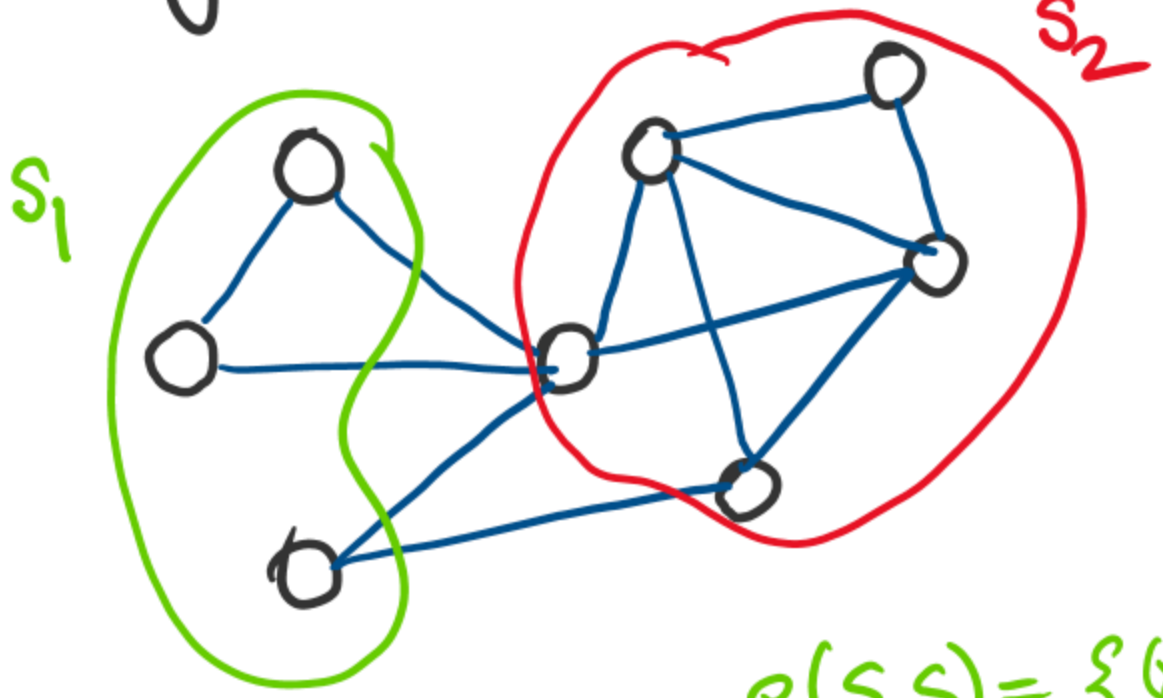
Most real networks are 'small world"
i.e. shortest path distance b/w most pairs
is very small, basically a small constant.

This is inspite of degree of most nodes being small!

Network clusters / communities

Few different definitions.

① Density based.



Density (S)

$= \frac{\#\text{edges b/w nodes both in } S}{|S|}$

$= \frac{|e(S,S)|}{|S|}$

$e(S,S) = \{(x,y) \in E,\ x \in S,\ y \in S\}$

Recent results in hierarchical clustering

- Parallelizing single linkage in small number of MapReduce rounds
  - Slight generalization of Boruvka's MST algorithm gives log(n) rounds
  - can be done in constant number of rounds

- Objective function based hierarchical clustering, how to approximate

Summary

- Clustering (i.e. unsupervised learning) is a rich area
  - Much scope for work in both theory and practice
  - New questions: streaming, online, notions of fairness and privacy
  - Structured data (e.g. ranking), non-Euclidean distances
- Distance functions and their properties
- K-means objective and algorithms – kmeans++, kmeans||
- Introduction to hierarchical clustering