20210922 Clustering

Wednesday, September 22, 2021 4:36 PM

- Submit Project preference by tomorrow
- Couple pitches couldn't make it to pitch day
- HW3 date moved by a week
- HWs are not necessarily getting harder, but they do cover different things
- There are different functions you can use for different use cases
- Unsupervised clustering:
  - Want like items together, unlike items separate

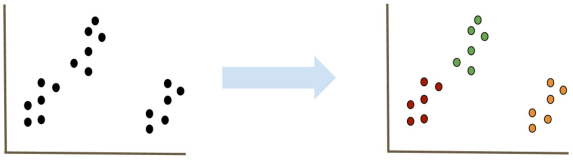
Clustering

Boston University CS 506 - Lance Galletti

What is a Clustering

A clustering is a grouping / assignment of objects (data points) such that objects in the same group / cluster are:

- similar to one another
- dissimilar to objects in other groups



Applications

- Outlier detection / anomaly detection
  - Data Cleaning / Processing
  - Credit card fraud, spam filter etc.
- Filling Gaps in your data
  - Using the same marketing strategy for similar people
  - Infer probable values for gaps in the data (similar users could have similar hobbies, likes / dislikes etc.)

- Don't want to have model fail just because you are missing a data point

The Clustering Problem

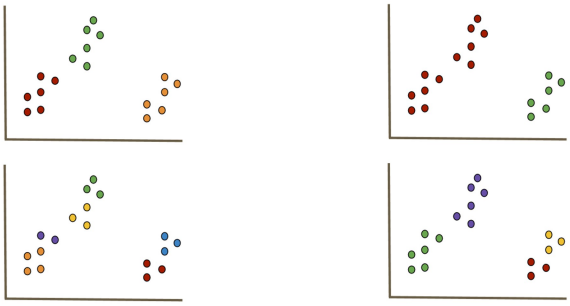
Given a collection of data points

Find a clustering such that:
- **Similar** data points are in the **same cluster**
- **Dissimilar** data points are in **different clusters**

Questions:
- What does **similar** mean?
- How do we find a **clustering**?
- How do we know if we have found a **good clustering**?

- Bit more formally, will keep refining how we are defining this
- How to go from dataset -> set of assignments

Clusters can be Ambiguous



- Can be a really hard problem
  - Ambiguous
  - All these clusters could potentially be valid
- There are different ways to approach this problem

Types of Clusterings

Partitional
Each object belongs to exactly one cluster

Hierarchical
A set of nested clusters organized in a tree

Density-Based
Defined based on the local density of points

Soft Clustering
Each point is assigned to every cluster with a certain probability

- Hierarchical:
  - One application: collected DNA of a bunch of different species
    - What species are most similar, when do they merge with the other group
    - What are two different species
- Soft clustering
  - Probabilistic clustering
  - You may a collection of weights of two different species
    - There may be overlap
      - Points equally probable in two species
      - Points that lean one way or the other

Partitional Clustering

Partitional Clustering

Given **n** data points and a number **k** of clusters: partition the **n** data points into **k** clusters.

Suppose we are given all possible ways of distributing these **n** data points into these **k** buckets / clusters. How would we find the best such partition?

Recall our goal: **similar** items should belong to the **same cluster** & **dissimilar** items should belong to **different clusters**.

A good partition is one where the total dissimilarity of points within each cluster is small.

- Up front we are saying how many clusters we want (k)
  - We will discuss how to pick this number
- If someone gave you all possible combinations of how these datapoints fit in k buckets
  - How to find the best such partition
  - Formulate mathematically what the goal is
  - Compute a global value for how well you are reaching this problem statement
  - You could take the best value or the optimum value for this goal
  - If the total dissimilarity is small, you have reached your goal

Example

Clearly the clustering on the left has smaller intra-cluster distances than the one on the right. That is:

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

Is a smaller quantity

- Both is k=2. Red or green
- The left one is better (but the right one is a valid partition)
  - What is meant by valid partition?
- Math forumula:
  - Inner sum: Take sum of all pairs of points of cluster, take the sum of their distances
  - Outer sum: Sum the above over all possible clusters k
- The graph on the left will have a smaller quantity than the graph on the right

Example

Given a distance function **d**, we can find points (not necessarily part of our dataset) for each cluster called **centroids** that are at the center of each cluster.

- Practically, calculating distances of all pairs of points is not possible (not efficient)
- So define a centroid
  - Center of mass (see black point for green cluster in both graphs)

Example

Q: When **d** is Euclidean, what is the **centroid** (also called **center of mass**) of **m** points $\{x_1, \ldots, x_m\}$?

A: The mean / average of the points

- With Euclidean distances, the center will be the average of the points
- You can choose any distance you want
- For now, you can assume the distance is Euclidean

Example

Turns out when **d** is Euclidean:

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

- mu is the center of mass for cluster k?
- Get familiar with this math
- Weighted sum by the size of the cluster
- Optimizing for either side optimizes for the other at the same time

K-means

Given $\mathbf{X} = \{x_1, \dots, x_n\}$ our dataset and **k**

Find **k** points $\{\mu_1, \dots, \mu_k\}$ that minimize the **cost function**:

$$\sum_{i}^{k} \sum_{x \in C_i} \|x - \mu_i\|_2^2$$

When **k=1** and **k=n** this is easy. Why?

When $\mathbf{x_i}$ lives in more than 2 dimensions, this is a very difficult (**NP-hard**) problem

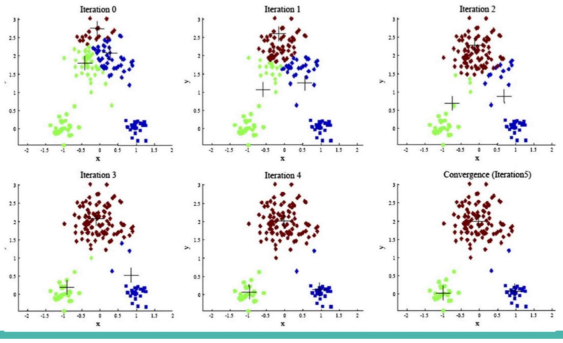
- Euclidean distance between x and ui
- k=1 everything is in the same cluster
- k=n everything has its own cluster
- For the purpose of this lecture, everything is in 2D
  - Higher dimensions becomes a lot harder

K-means - Lloyd's Algorithm

1. Randomly pick **k** centers $\{\mu_1, \dots, \mu_k\}$
2. Assign each point in the dataset to its closest center
3. Compute the new centers as the means of each cluster
4. Repeat 2 & 3 until convergence

- If equally distance between two clusters, can randomly assign the point to which one it belongs to
- k centers were random
  - So once you assign to clusters, the new mean of the cluster will be the new center
- Keep repeating

K-means - Lloyd's Algorithm



- The set of the points in each cluster changes through each iteration until converged
- Keep moving the center
- This will be part of lab, so you need to be comfortable with implementing it

K-means - Lloyd's Algorithm

Will this algorithm always converge?

Proof (by contradiction): Suppose it does not converge. Then, either:

1. The minimum of the cost function is only reached in the limit (i.e. after an infinite number of iterations).
   Impossible because we are iterating over a finite set of partitions

1. The algorithm gets stuck in a cycle / loop
   Impossible since this would require having a clustering that has a lower cost than itself and we know:
   - If old ≠ new clustering then the cost has improved
   - If old = new clustering then the cost is unchanged

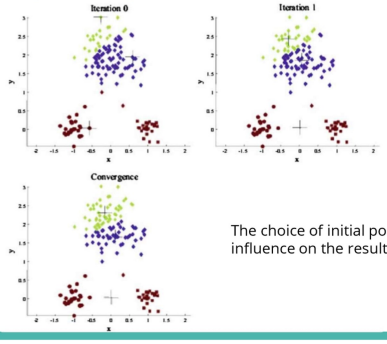
Conclusion: Lloyd's Algorithm always converges!

- Semi-formal
- Finite number of values, so only a finite number of possible cluster assignments, so not possible to generate an infinite amount of values
- Every iteration lowers the cost
- ?What if two different sets of cluster assignments that are equally good?

K-means - Lloyd's Algorithm

Will this always converge to the optimal solution?

K-means - Lloyd's Algorithm



The choice of initial points has a large influence on the resulting clustering

- Counter example
  - Convergence was not the best option in this situation
- It always converges, but not always to the optimal solution

K-means - Initialization

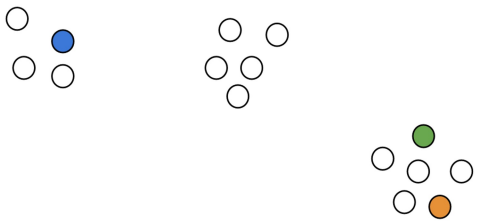
One solution: Run Lloyd's algorithm multiple times and choose the result with the lowest cost.

This can still lead to bad results because of randomness.

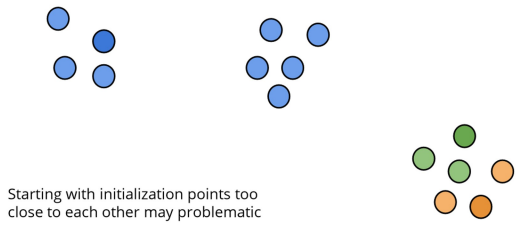
Another solution: Try different initialization methods

- Could try this a large number of times
  - Because of randomness, this may never find the true optimum
  - Computationally expensive
- So, let's try a different initialization method

K-means - Random



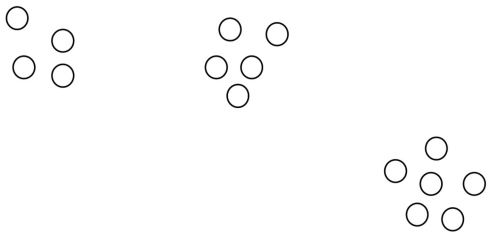
K-means - Random



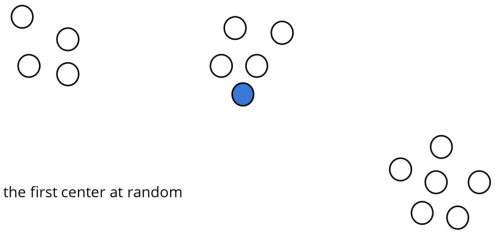
Starting with initialization points too close to each other may problematic

- Intuition: Maybe points too close to each other is problematic
- So lets try generated random that are far apart
- Start with one random point, get the furthest point from that, and the next the farthest from the both
- This does not deal well with outliers

K-means - Farthest First Traversal

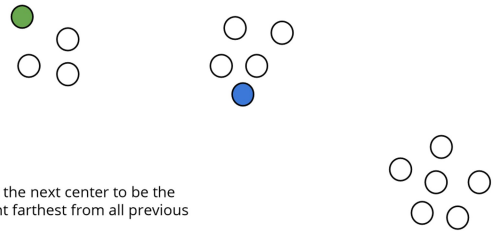


K-means - Farthest First Traversal



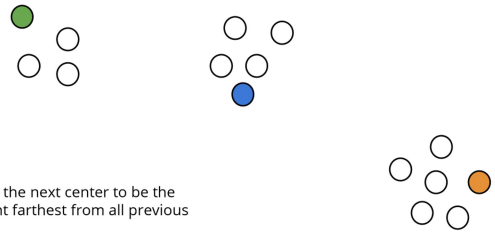
Pick the first center at random

K-means - Farthest First Traversal



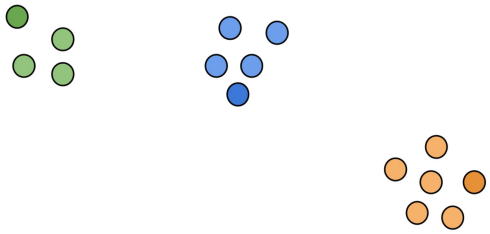
Pick the next center to be the point farthest from all previous

K-means - Farthest First Traversal

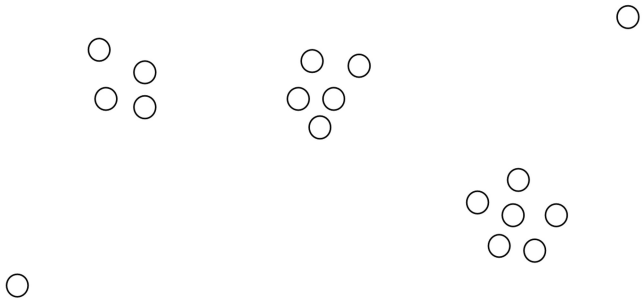


Pick the next center to be the point farthest from all previous

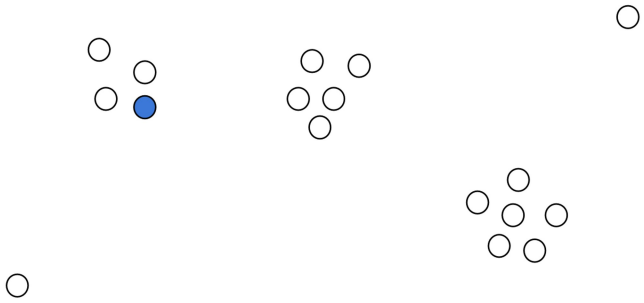
K-means - Farthest First Traversal



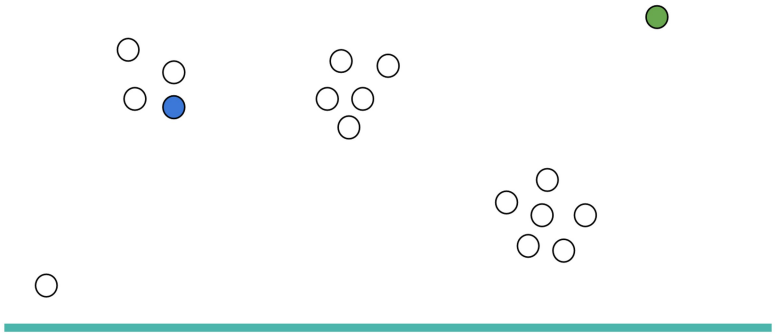
K-means - FFT and outliers



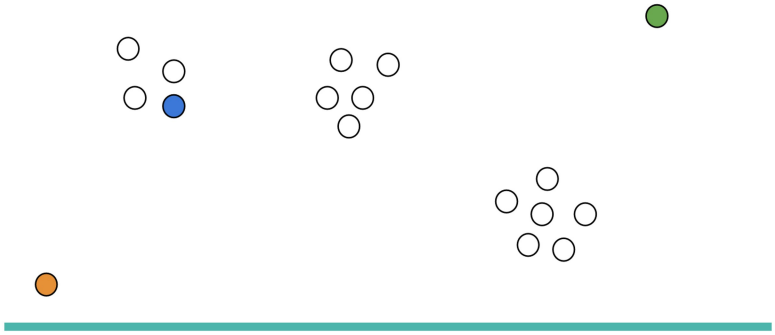
K-means - FFT and outliers



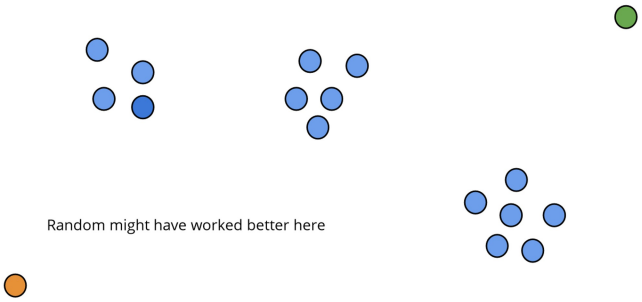
K-means - FFT and outliers



K-means - FFT and outliers



K-means - FFT and outliers



- This isn't good either
- In this case, we may have been better off with random points

K-means++

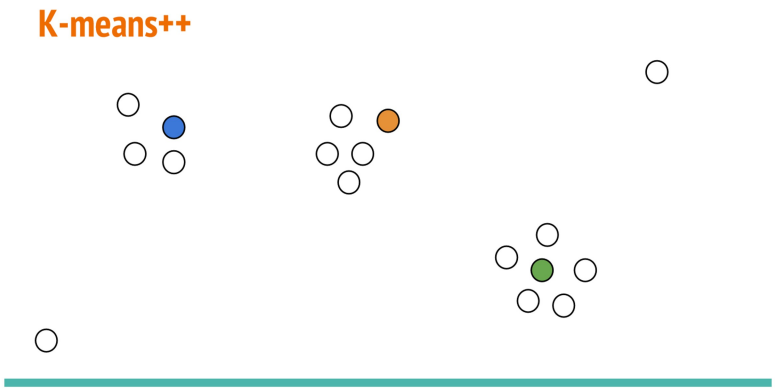
Initialize with a combination of the two methods:

1. Start with a random center
2. Let **D(x)** be the distance between **x** and the centers selected so far. Choose the next center with probability proportional to $\mathbf{D(x)^a}$

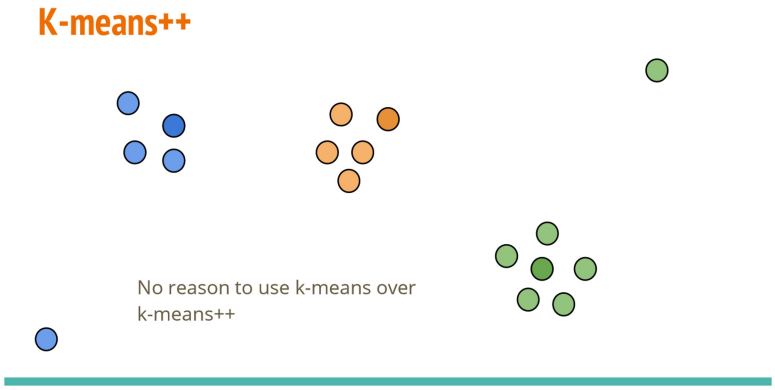
When:

- **a = 0** : random initialization (all points have equal probability)
- **a = ∞** : farthest first traversal
- **a = 2** : K-means++

- Same algorithm as Lloyd's, but the initialization of the starting centroids is different



- Picked points that are farther from each other, but with some randomness in selection



- K-means++ is clearly superior, do not use k-means

K-means++

Suppose we are given a black box that will generate a uniform random number between 0 and any **N**. How can we use this black box to select points with probability proportional to $\mathbf{D(x)^a}$?

-

K-means++

Suppose we are given a black box that will generate a uniform random number between 0 and any **N**. How can we use this black box to select points with probability proportional to $\mathbf{D(x)^a}$?

K-means++

Suppose we are given a black box that will generate a uniform random number between 0 and any **N**. How can we use this black box to select points with probability proportional to **D(x)²**?

Let's set **a = 2**

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

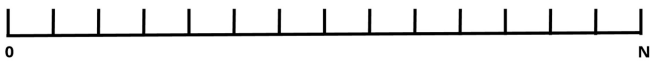
- How do we generate a probability distribution to do this?

K-means++

Suppose we are given a black box that will generate a uniform random number between 0 and any **N**. How can we use this black box to select points with probability proportional to **D(x)²**?

Let's set **a = 2**

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



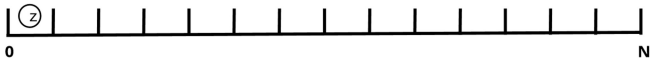
- Generate random number on this line
  - Populate this line based on the distances seen above

K-means++

Suppose we are given a black box that will generate a uniform random number between 0 and any **N**. How can we use this black box to select points with probability proportional to $\mathbf{D(x)^2}$?

Let's set **a = 2**

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

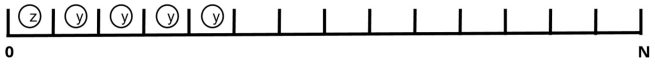


K-means++

Suppose we are given a black box that will generate a uniform random number between 0 and any **N**. How can we use this black box to select points with probability proportional to $\mathbf{D(x)^2}$?

Let's set **a = 2**

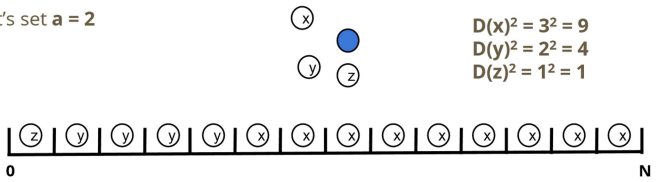
$D(x)^2 = 3^2 = 9$
$D(y)^2 = 2^2 = 4$
$D(z)^2 = 1^2 = 1$



K-means++

Suppose we are given a black box that will generate a uniform random number between 0 and any **N**. How can we use this black box to select points with probability proportional to **$D(x)^2$**?

Let's set **a = 2**

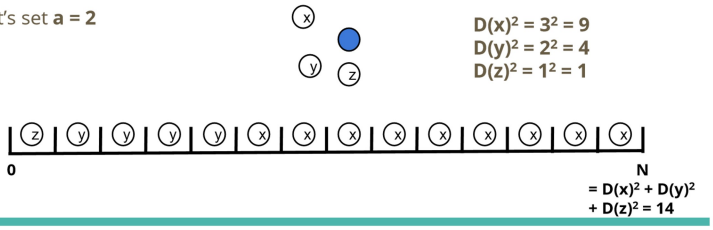


- Ranges assigned in proportion to the probability you wanted

K-means++

Suppose we are given a black box that will generate a uniform random number between 0 and any **N**. How can we use this black box to select points with probability proportional to **$D(x)^2$**?

Let's set **a = 2**

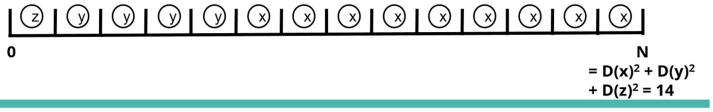


- Generate random integer between 1 and 14, and you will know whether you are x, y, z
- Method to use If you only have access to a basic random generator

K-means++

Suppose we are given a black box that will generate a uniform random number between 0 and any **N**. How can we use this black box to select points with probability proportional to **$D(x)^2$**?

Using the black box, we can generate a number between 0 and N to determine which point to pick next. It will be chosen with probability proportional to **$D(x)^2$**.

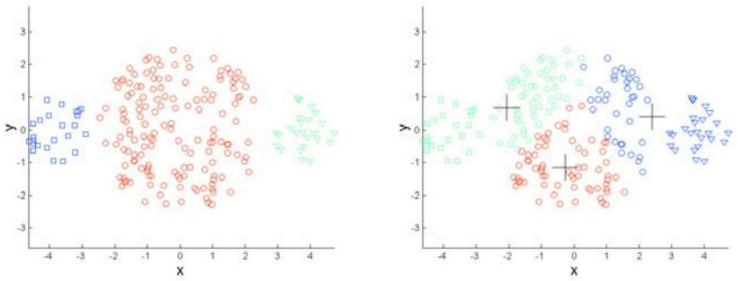


K-means++

What happens if the black box can only generate numbers between 0 and 1?

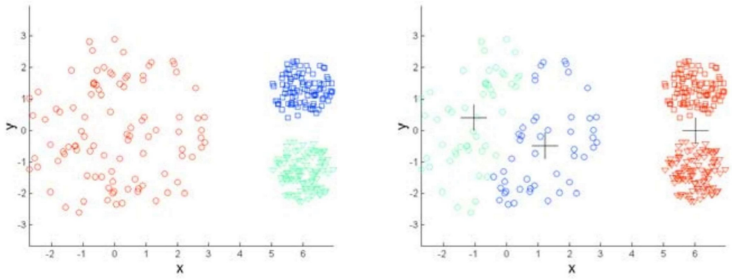
- Divide everything by N

K-means - Limitations



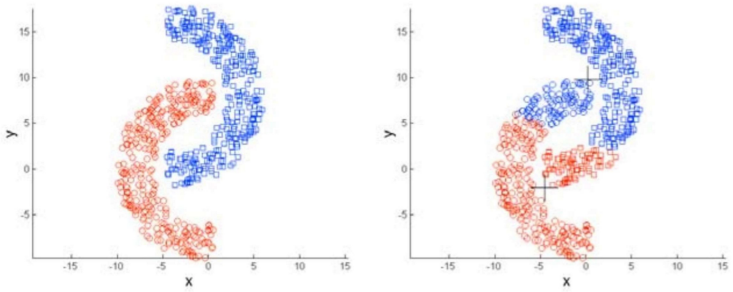
- Tends to prefer clusters:
  - Of similar sizes
  - globular clusters
- Tends to break up larger clusters

K-means - Limitations



- Doesn't handle points of varying density well

K-means - Limitations

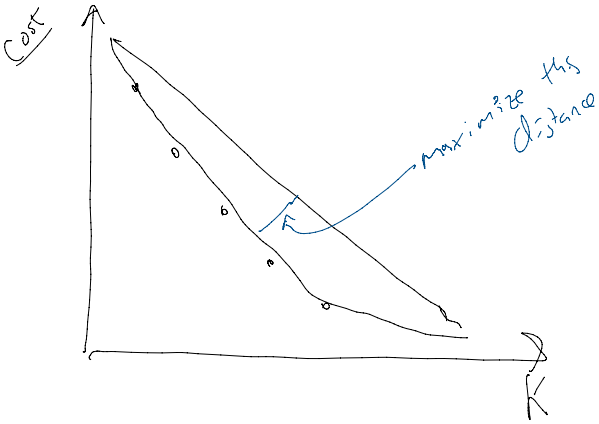


- Doesn't deal well with non-globular shapes
- Prefers globular shapes of all the same sizes

How to choose the right k?

1. Iterate through different values of k (elbow method)
2. Use empirical / domain-specific knowledge
   Example: Is there a known approximate distribution of the data? (K-means is good for spherical gaussians)

How to choose the right k:



- Really tricky
- Can use domain specific knowledge
  - As a data scientist, you must always be in contact with someone who has domain specific knowledge
    - You are expert in algorithms
  - These values don't make sense… these do makes sense
- From the math side: elbow method
- ?Silhouette scores?

K-means Variations

1. K-medians (uses the L_1 norm / manhattan distance)
2. K-medoids (any distance function + the centers must be in the dataset)
3. Weighted K-means (each point has a different weight when computing the mean)

- Certain values may have higher value
  - Assign them more weight in the calculation
  - Handling outliers, points that don't represent the data that well in general
- k-means:
  - If you remove outliers first, does this more look like k-means++?
    - Removing outliers would put you in a better spot, but you would still prefer k-means++ because you have a better chance of choosing points that are farther away from each other

File - C:\Users\Wolfs\PycharmProjects\cs506_20210922\random1.py

```
 1
 2 """
 3 Always start a new git branch whenever you do
   something new
 4 Also create virtual environments (optional)
 5
 6
 7
 8 You can have different dependencies for different
   projects without effecting everything, so if you need
   to use two
 9 versions of numpy
10 """
11
12 # matplotlib is the way you graph things nicely in
   python
13 import random
14 import matplotlib.pyplot as plt
15
16 # Do one instance, then can put it inside a for loop
17
18 # OUr distances
19 point_distances = {
20     "x": 9,
21     "y": 4,
22     "z": 1
23 }
24 # We want to pick x, y and z proportional to these
   distances
25 N = sum(point_distances.values())
26
27
28 def get_point_from_rand(result):
29     """
30
31     :param result:
32     :return:
33     """
34     threshold = 0
35     for p in point_distances:
36         threshold += point_distances[p]
```

File - C:\Users\Wolfs\PycharmProjects\cs506_20210922\random1.py

```
37          if result <= threshold:
38              return p
39
40 count = {
41     "x": 0,
42     "y": 0,
43     "z": 0
44 }
45
46 for i in range(1000):
47     res = random.randint(1, N)  # Rand int is
   inclusive
48     point = get_point_from_rand(res)
49     count[point] += 1
50 # In different programming languages, you don't want
   to iterate over keys some languages randomly pull out
   the keys
51 # plt.bar([key for key in point_distances], [count[key
   ] for key in point_distances] for key in  count, color
   ='green')
52 plt.bar([key for key in count], [count[key] for key in
    count], color='green')
53 plt.xticks([key for key in count], [key for key in
   count])
54 plt.show()
55 # Barchart
```