

K Means Clustering and Intro to EM

- Alternating Optimization,
- Distance Metrics

Gates

Clustering: Grouping - Categorizing

- 1) Clustering is “unsupervised”. This means that the data is not labeled or categorized.
- 2) Clustering is used to “discover” if groups/categories exist and if so – what they are.
- 3) Clustering can be used to determine if the data in a dataset fits into any type of groups/categories/classes.
- 4) If categories can be identified, this information then be used to **classify** or **predict** other vectors.
- 5) Clustering **reduces** dimensionality.
- 6) Clustering can be used to create **labels**.
- 7) Clustering is a special case of alternating optimization/EM via Lloyd’s Algorithm

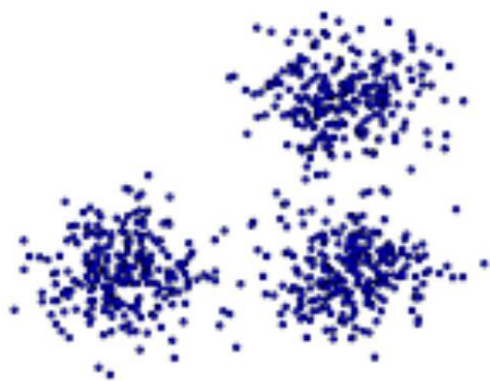
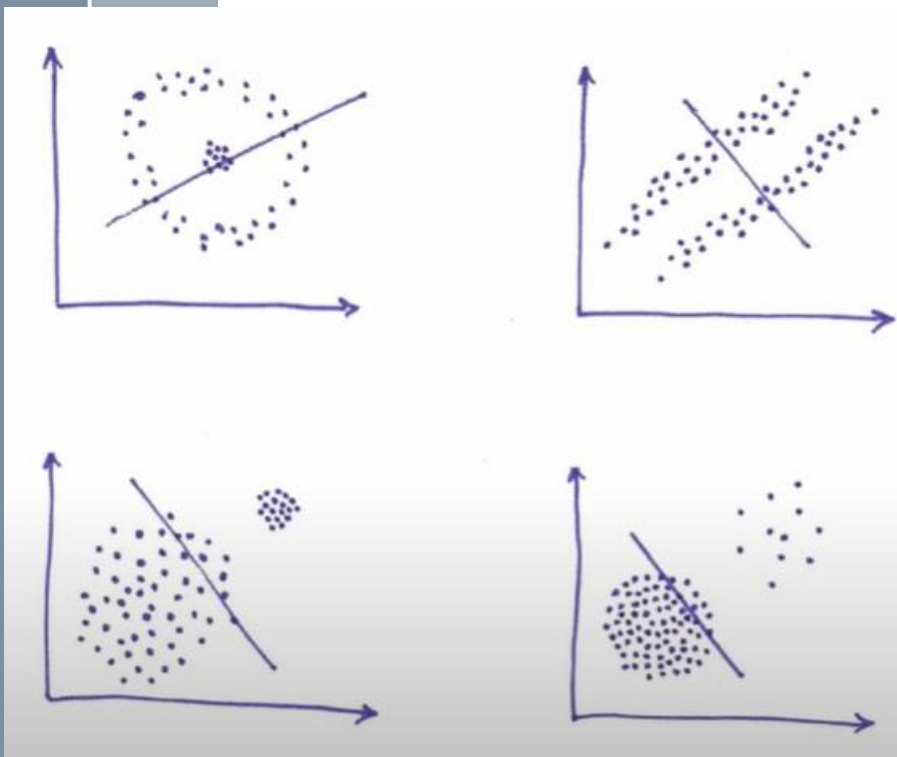
What is a VECTOR (in a dataset)?

π

Height	Weight	Age
84	250	17
72	200	16
70	210	15
86	278	18
74	190	15
80	245	16
79	267	19
71	187	15
69	211	14
82	289	17

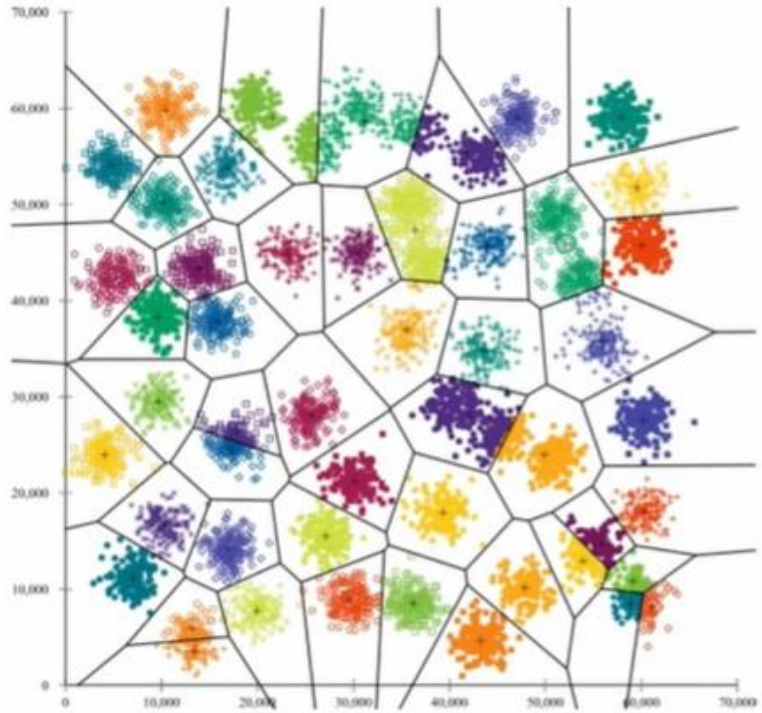
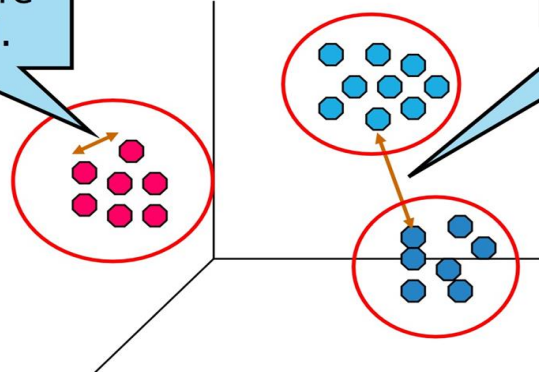
Height	Weight	Age	GPA	TestScore
84	250	17	3.8	994
72	200	16	3.5	876
70	210	15	3.6	769
86	278	18	3.9	901
74	190	15	3.4	899
80	245	16	3.9	955
79	267	19	4	850
71	187	15	3.6	900
69	211	14	3.7	700
82	289	17	3.9	941

	bring	chocolate	coffee	day	...	like	mountain	needed	taking
choc	0	3	2	1	...	0	0	0	0
choc	0	3	2	0	...	0	0	0	0
choc	0	3	1	0	...	1	0	0	0
choc	0	5	3	1	...	0	0	0	0
choc	0	9	4	2	...	0	0	0	0
choc	0	5	3	1	...	0	0	0	0
hike	1	1	0	0	...	0	0	0	1
hike	0	0	0	0	...	0	0	1	0
hike	0	1	0	0	...	0	0	0	0
hike	0	0	0	0	...	0	2	0	0
hike	0	0	0	0	...	0	4	0	0
hike	0	0	0	0	...	0	2	1	0



Intracluster distances are minimized.

Intercluster distances are maximized.



Common Clustering Categories

- ***Partitional Clustering (such as k – means)***: divides data objects into **nonoverlapping groups**. No object can be a member of more than one cluster, and every cluster must have at least one object.
- ***Hierarchical Clustering*** : determines cluster assignments by **building a hierarchy**. This is implemented by either a **bottom-up or a top-down** approach. These methods produce a tree-based hierarchy of points called a **dendrogram**. - *Agglomerative or divisive*
- ***Density Based Clustering*** : determines cluster assignments based on **the density of data points in a region**. This approach doesn't require the user to specify the number of clusters. Instead, there is a distance-based parameter that acts as a tunable threshold. Example : **DBSCAN** (Density-Based Spatial Clustering of Applications with Noise)

Applications of clustering

- 1) There are an enormous number of applications for clustering, as clustering is a method for discovering similarity (and differences) between data vectors/rows.
- 2) One can cluster books, articles, or documents by topics.
- 3) One can cluster customers by common attributes – such as purchase similarities, location similarities, expenditure similarities, etc.
- 4) One can cluster social data by attributes such as common interests, common career areas, etc.
- 5) One can cluster radiation data collected from objects in space to determine if they are stars, planets, galaxies, etc.
- 6) One can cluster music into categories – think about this for a second – how do you think Pandora does this?

Clustering can also be used for outlier detection. This is especially true for visual EDA (exploratory data analysis).

π

Example: Market Segmentation

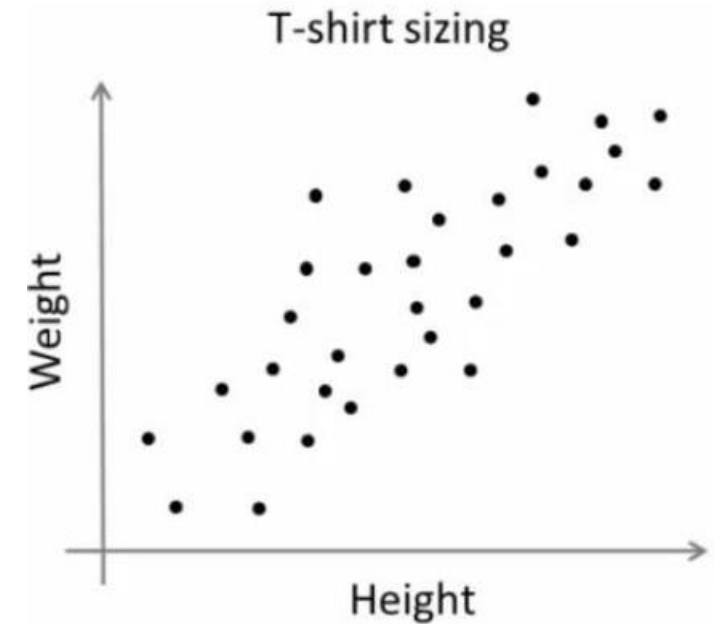
Suppose your company makes t-shirts.

T-shirts cannot come in an infinite number of sizes ☺

You, as the owner or manager, will determine how many sizes you want to manufacture (this is your k).

Then, you can collect data (a sample from your buyer population) that includes weight and height. [You will do this likely by gender as well.]

You will then **cluster** this data using a method like k means to determine the centroid (average height and weight in this case) of each of your clusters.



Example 2: College Admissions

Suppose it is your job to analyze and then suggest methods to improve college admissions.

You have all the admissions data from the past 10 years. Your data is labeled as Admit, Decline, and Waitlist.

You can apply clustering to the data (after you remove the labels) to answer many questions. Here, your first k is 3, but you can also experiment with other k values. Why?

Before performing k means clustering, assume that you assure only quantitate data and you remove and save the labels.

Questions you can explore:

- 1) Do your labels match the cluster centroids (using $k = 3$)?
- 2) Which features appear significant?
- 3) What is the average GPA of students in each cluster? What are other average scores in each cluster?
- 4) etc.

Common Clustering Categories

- ***Partitional Clustering (such as k – means)***: divides data objects into **nonoverlapping groups**. No object can be a member of more than one cluster, and every cluster must have at least one object.
- ***Hierarchical Clustering*** : determines cluster assignments by **building a hierarchy**. This is implemented by either a **bottom-up or a top-down** approach. These methods produce a tree-based hierarchy of points called a **dendrogram**. - *Agglomerative or divisive*
- ***Density Based Clustering*** : determines cluster assignments based on **the density of data points in a region**. This approach doesn't require the user to specify the number of clusters. Instead, there is a distance-based parameter that acts as a tunable threshold. Example : **DBSCAN** (Density-Based Spatial Clustering of Applications with Noise)

Clustering Uses a Measure of Distance or Similarity

π

Distance by Hand...Which is “closer” – rows 1 and 2 OR 1 and 3?

	cholesterol	weight	Height
1	251	267	70
2	105	103	62
3	156	193	72
4	7000	100	63
5	198	210	70
6	189	189	64

```
> (dm <- dist(HeartDF2_num, method = "manhattan"))
```

```
      1      2      3      4      5      6      7      8      9      :
2     318
3     171  151
```

Using Manhattan:

Dist(1, 2):

[251 267 70]

[105 103 62]

$$|(251-105)| + |(267-103)| + |(70-62)| = 318$$

Dist(1, 3)

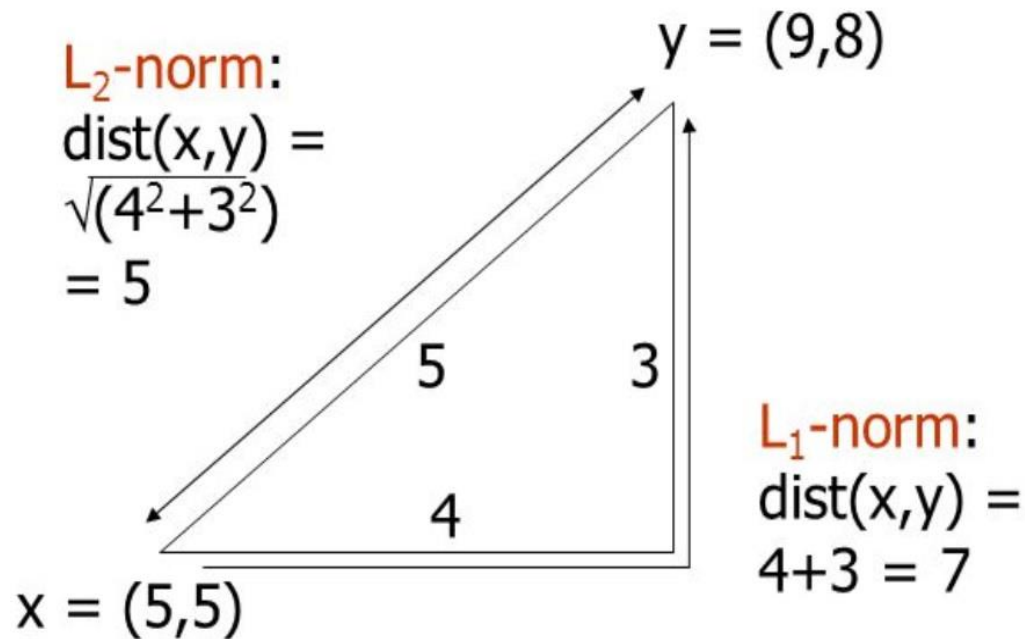
[251 267 70]

[156 193 72]

$$\text{abs}(251-156) + \text{abs}(267-193) + \text{abs}(70-72) = 171$$

Some Euclidean Distances

- L_2 norm : $d(x,y)$ = square root of the sum of the squares of the differences between x and y in each dimension.
 - The most common notion of “distance.”
- L_1 norm : sum of the differences in each dimension.
 - *Manhattan distance* = distance if you had to travel along coordinates only.

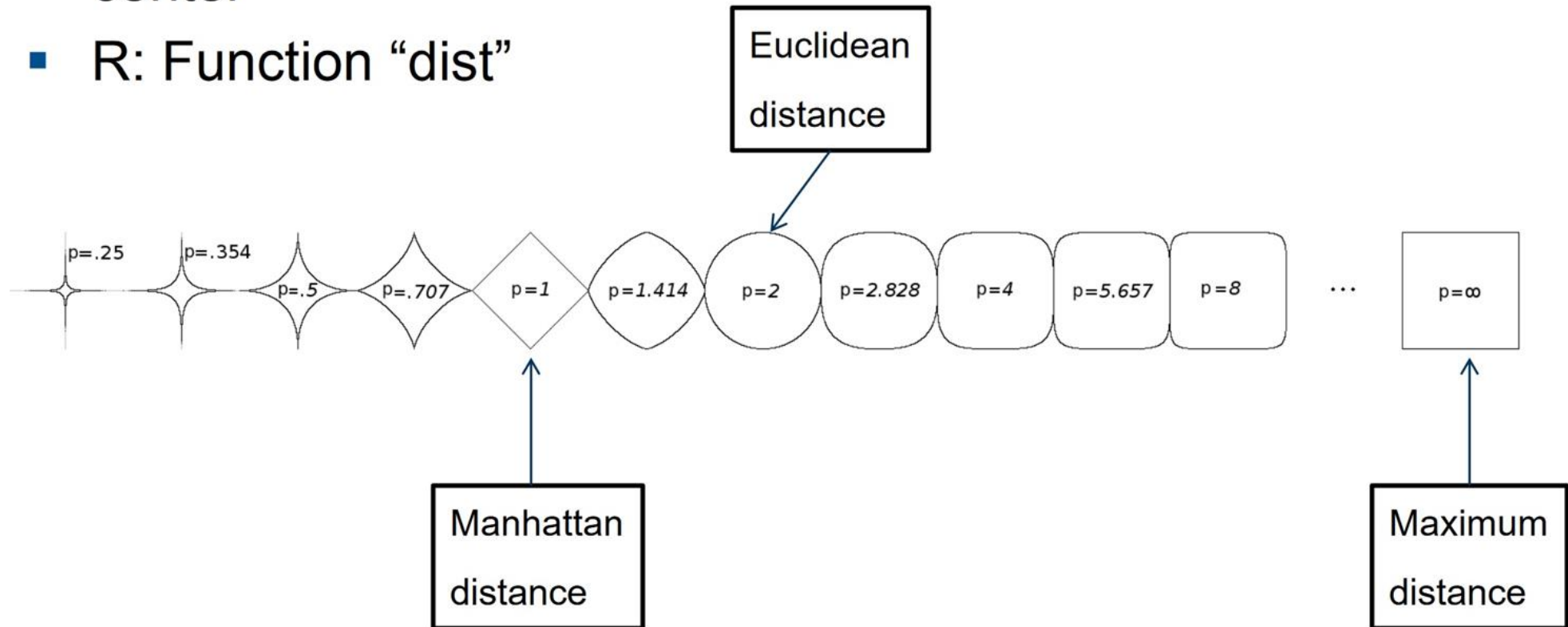


Note: The L_3 norm, cubes the differences and so on.

The L_1 norm is often called the **Manhattan distance** as it is based on the idea of “block distance” rather than point to point direct distance.

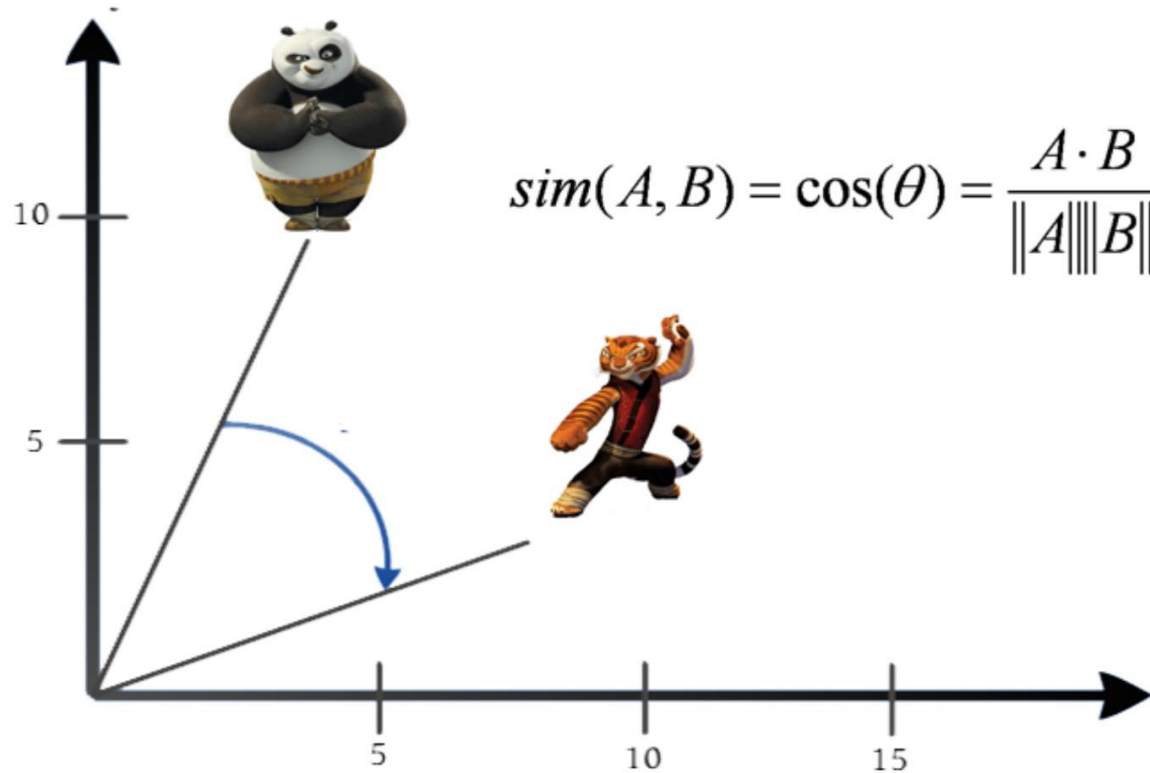
Intuition for Minkowski Distance

- p : Index of Minkowski Distance
- Points on the line have equal Minkowski Distance from center
- R : Function “dist”

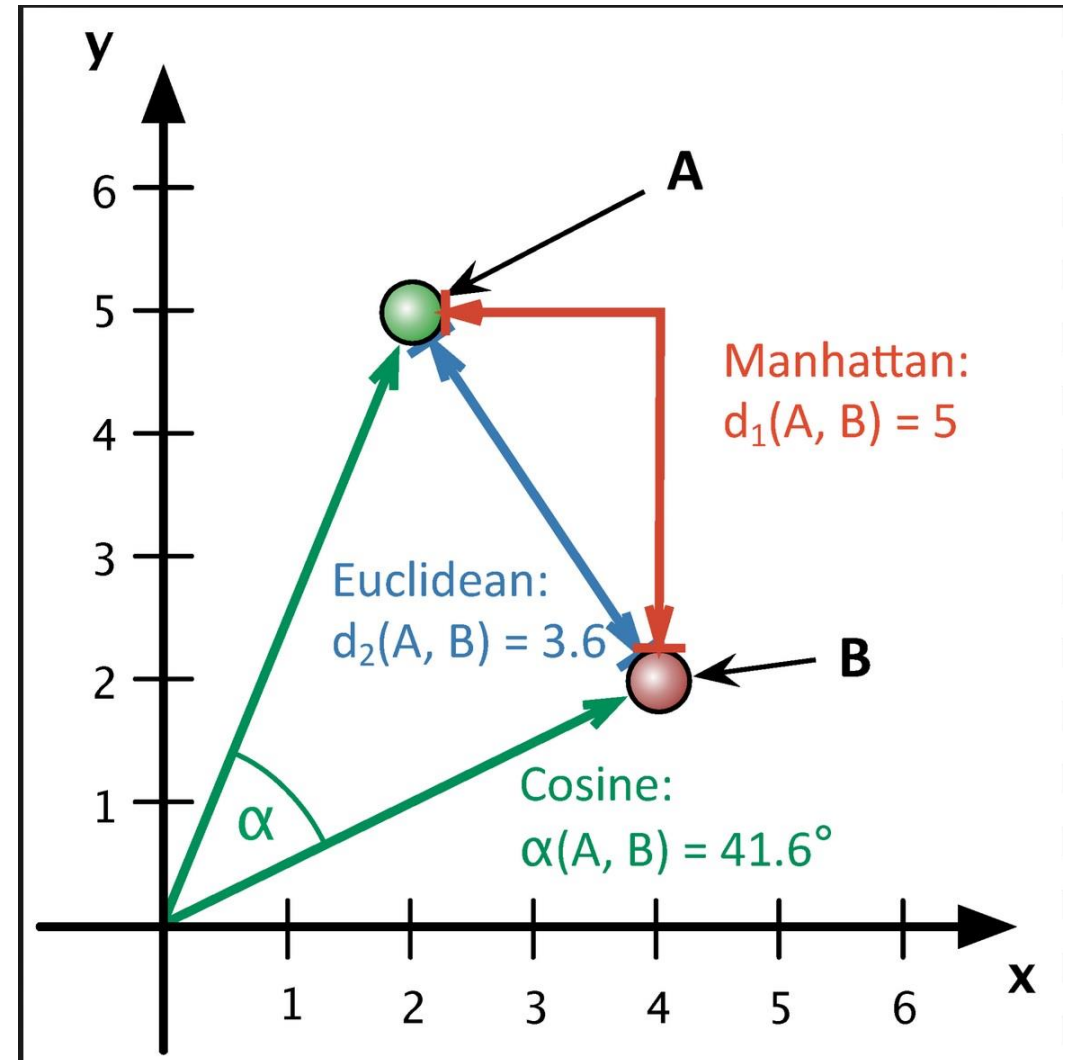


Cosine Similarity

<http://dataaspirant.com/2015/04/11/five-most-popular-similarity-measures-implementation-in-python/>



Cosine Similarity is non-Euclidean.



Cosine Similarity – by hand – in Python

```
import numpy as np

def cosine_sim(x, y):
    x = np.array(x)
    y = np.array(y)

    if len(x) != len(y) :
        return None

    dot_product = np.dot(x, y.T)

    # Magnitudes of x and y
    magnitude_x = np.sqrt(np.sum(x**2))
    magnitude_y = np.sqrt(np.sum(y**2))

    # Cosine Sim and angle as degrees
    cos_sim = dot_product / (magnitude_x *
magnitude_y)
    angle=np.arccos(cos_sim)
    angle=np.degrees(angle)

    return cos_sim, angle

CS, Cos_angle=cosine_sim([[251,267,70]], [[105,103,62]])

print(CS)
print(Cos_angle)
```

```
from sklearn.metrics.pairwise import cosine_similarity
import pandas as pd

DF = pd.read_csv("C:/Users/profa/Desktop/UCB/ML
CSCI 5622/Data/HeartRisk_JustNums.csv")

print(cosine_similarity(DF, DF))
```

Cosine similarity

Good for high D data.

- 1) Imagine that each row or data vector is a numerical vector.
- 2) Next, no matter what dimension you are using, the origin is (0, 0, ...0)
So in 2D the origin is (0,0) in 3D its (0,0,0) and so on.
- 3) Next, Any two vector points in any D space create an angle between them.
- 4) The COS of any angle is defined as the normalized dot product:

$$(V1 \cdot V2) / |V1| |V2|$$

Example: Suppose vector 1 (V1) is [1, 3] and suppose vector 2 (V2) is [2,2]

Then, the dot product $V1 \cdot V2 = (1*2) + (3*2) = 8$

Next, $|V1| = \text{sqrt}(1^2 + 3^2) = \text{sqrt}(10)$

$|V2| = \text{sqrt}(2^2 + 2^2) = \text{sqrt}(8)$

So $\text{COS}() = 8 / (\text{sqrt}(8)*\text{sqrt}(10)) = .894$

To solve for the angle, find the arccos (.894) = 26.6 degrees

Non-Euclidean distances

- ◆ *Jaccard distance* for sets = 1 minus ratio of sizes of intersection and union.
- ◆ *Cosine distance* = angle between vectors from the origin to the points in question.
- ◆ *Edit distance* = number of inserts and deletes to change one string into another.

Some fun math: the axioms of a distance measure

1) For any measure D to be a “distance measure” it must meet the following properties:

(a) Given any two vectors x and y , $D(x,y) \geq 0$. In other words, distance cannot be negative.

(b) $D(x, y) = 0$ iff $x = y$. In other words, the only way the distance between two vectors is 0 is if they are the same vector.

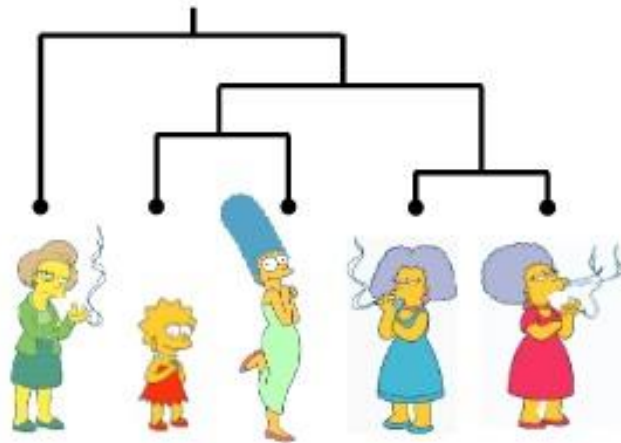
(c) $D(x,y) = D(y,x)$. Direction does not matter.

(d) $D(x, y) \leq D(x,z) + D(z,y)$. This is called the Triangle Inequality.

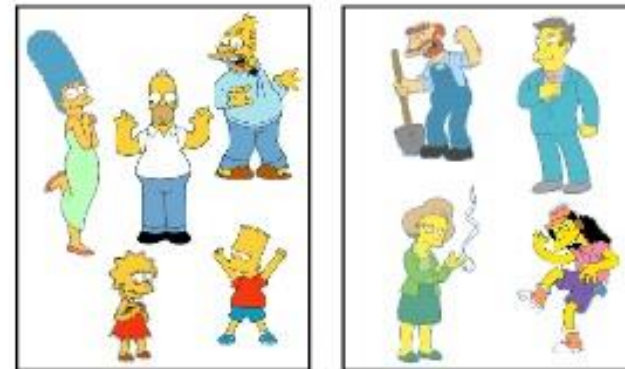
Partition vs. Hierarchical

- **Partitional algorithms:** Construct various partitions and then evaluate them by some criterion (we will see an example called BIRCH)
- **Hierarchical algorithms:** Create a hierarchical decomposition of the set of objects using some criterion

Hierarchical



Partitional

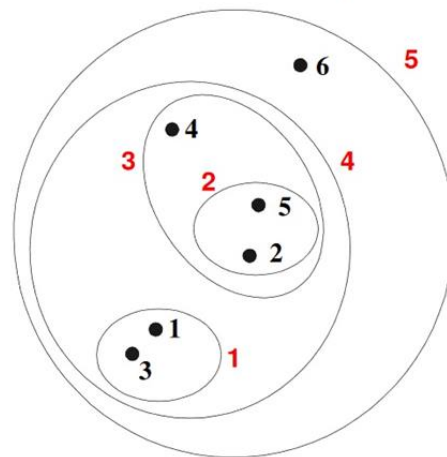
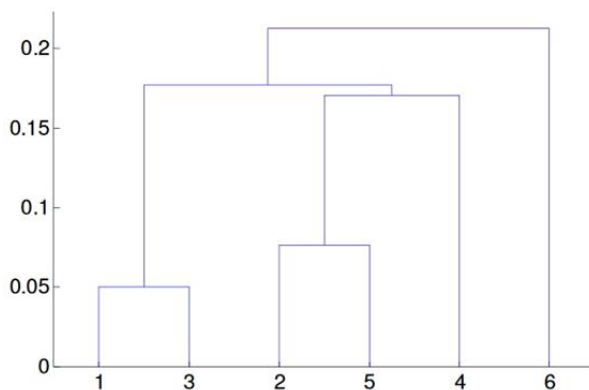


Hierarchical Clustering

- Two main types of hierarchical clustering
 - **Agglomerative:**
 - Start with the points as individual clusters
 - At each step, merge the closest pair of clusters until only one cluster (or k clusters) left
 - **Divisive:**
 - Start with one, all-inclusive cluster
 - At each step, split a cluster until each cluster contains a point (or there are k clusters)
- Traditional hierarchical algorithms use a **similarity** or **distance matrix**
 - Merge or split one cluster at a time

Hierarchical Clustering

- Produces a set of nested clusters organized as a hierarchical tree
- Can be visualized as a dendrogram
 - A tree like diagram that records the sequences of merges or splits



- 1) It is not necessary to choose or preselect the number of clusters.
- 2) Hierarchical clusterings may correspond well to taxonomies – such as the animal kingdom.

AGNES and DIANA

Hierarchical clustering can be divided into two main types:

Agglomerative clustering: Commonly referred to as **AGNES** (AGglomerative NESTing) works in a **bottom-up** manner. Each observation (vector or row) is initially considered as a **single-element** cluster.

At each step of the algorithm, the two clusters that are the most similar are combined into a new bigger cluster. This procedure is iterated until all points are a member of just one single big cluster

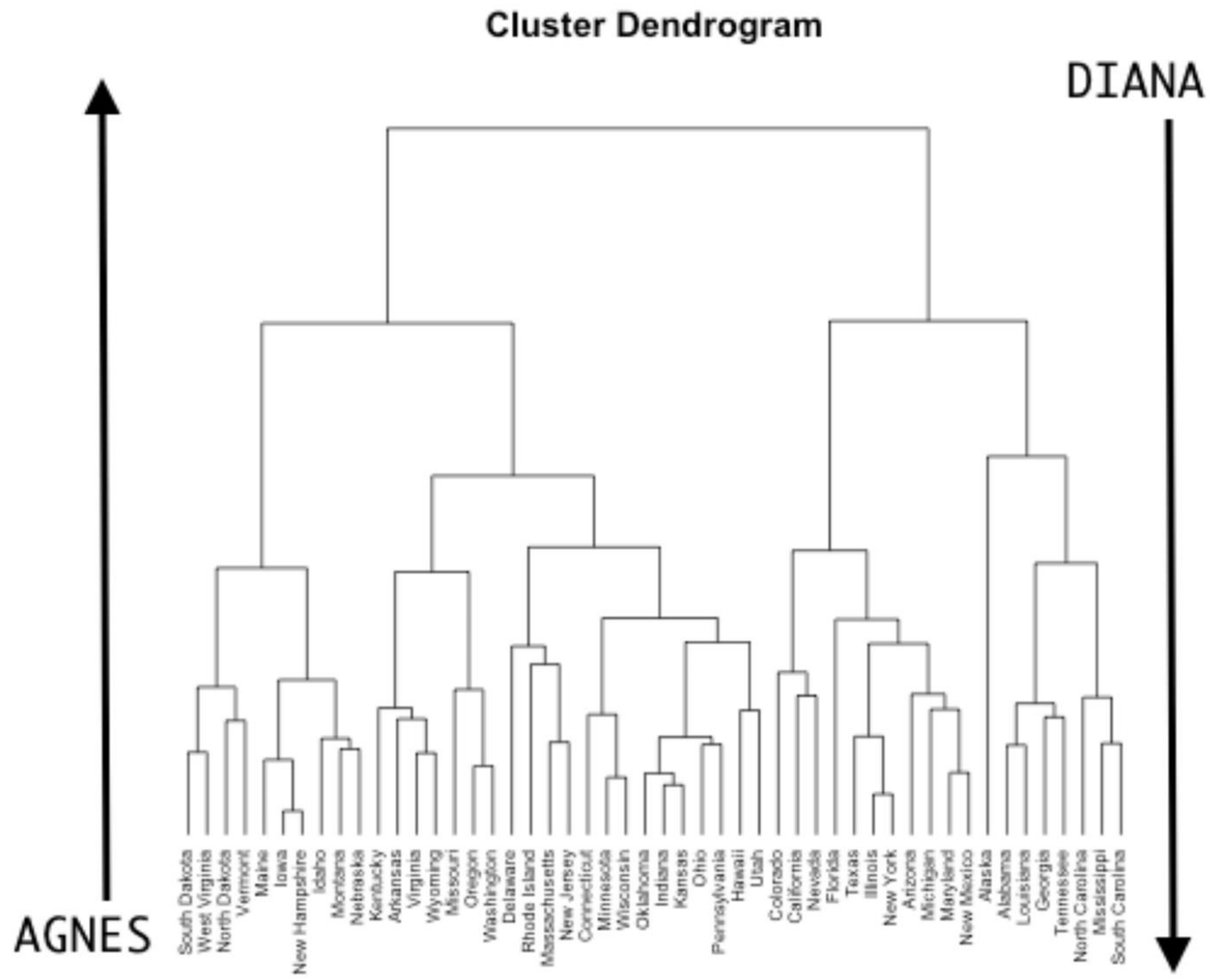
Divisive hierarchical clustering: Commonly referred to as **DIANA** (Divise ANALysis) works in a **top-down** manner.

Begins with the root - all observations (all rows) are in a single cluster. At each step of the algorithm, the current cluster is **split into two clusters** that are considered most heterogeneous. The process is iterated until all observations are in their own cluster.

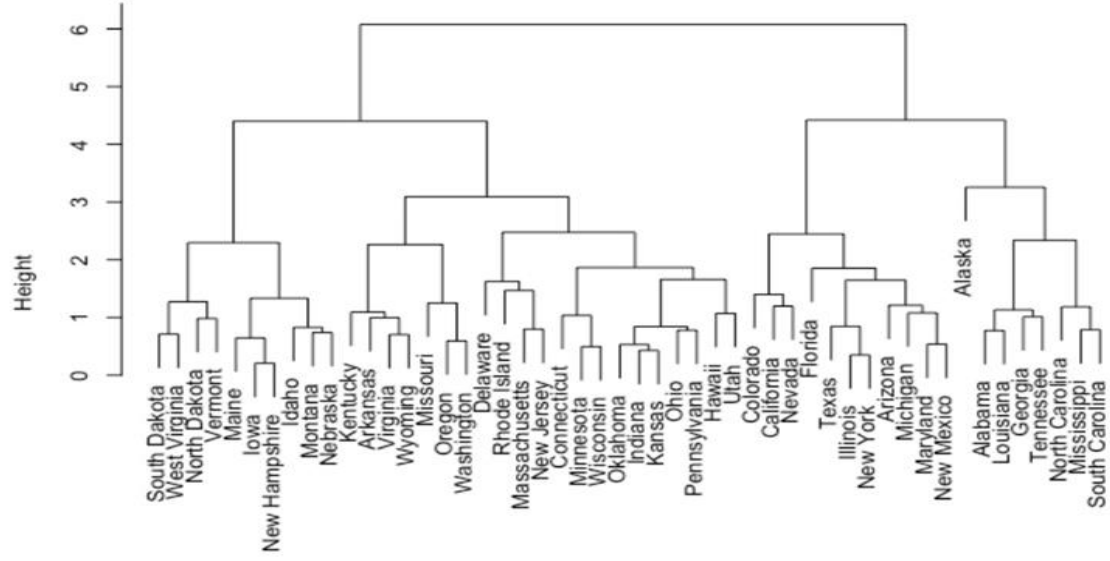
Common Clustering Methods (FYI only)

- **Maximum or complete linkage clustering:** Computes all **pairwise dissimilarities** between the elements in cluster 1 and the elements in cluster 2, and considers the **largest** value of these dissimilarities as the distance between the two clusters. It tends to produce more compact clusters.
- **Minimum or single linkage clustering:** Computes all **pairwise dissimilarities** between the elements in cluster 1 and the elements in cluster 2, and considers the **smallest** of these dissimilarities as a linkage criterion. It tends to produce long, “loose” clusters.
- **Mean or average linkage clustering:** Computes all **pairwise dissimilarities** between the elements in cluster 1 and the elements in cluster 2, and considers the **average** of these dissimilarities as the distance between the two clusters. Can vary in the compactness of the clusters it creates.
- **Centroid linkage clustering:** Computes the dissimilarity between the centroid for cluster 1 (a mean vector of length p , one element for each variable) and the centroid for cluster 2.
- **Ward’s minimum variance method:** **Minimizes** the total within-cluster variance. At each step the pair of clusters with the smallest between-cluster distance are merged. Tends to produce more compact clusters.

π

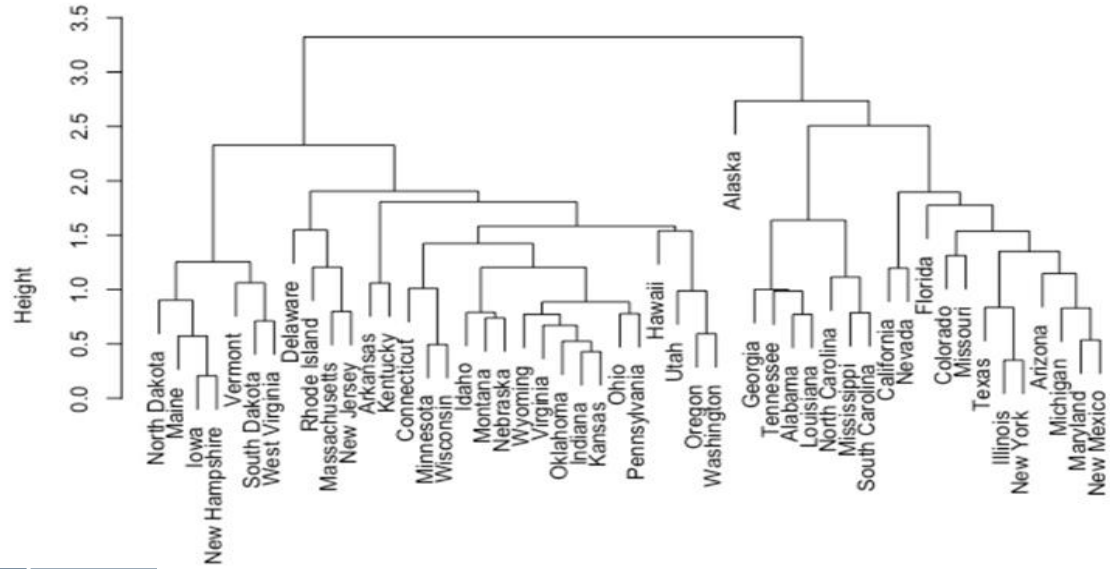


Complete Linkage

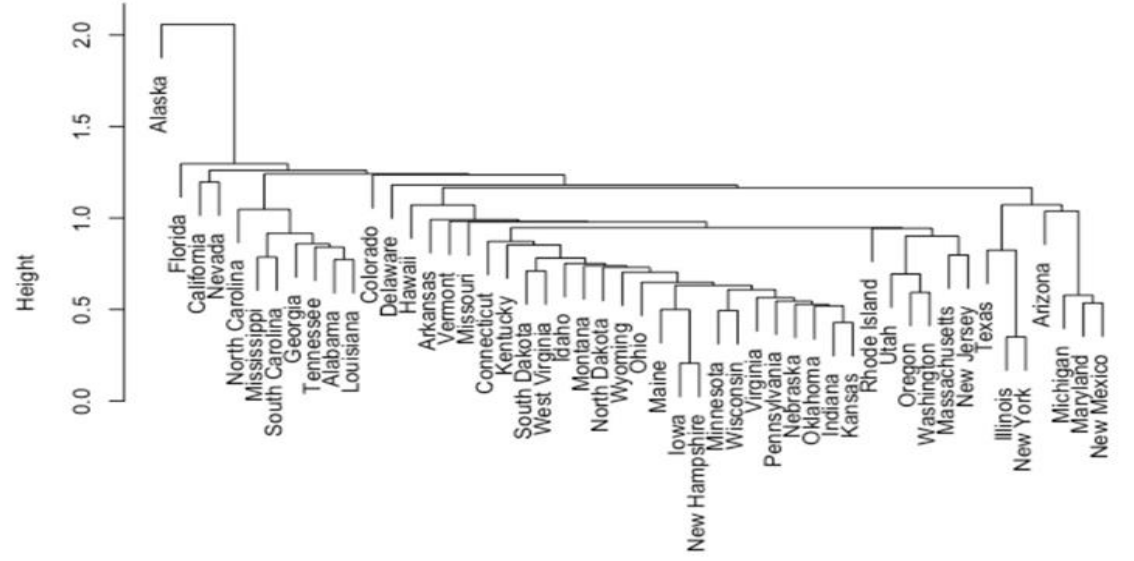


d
hclust (*, "complete")

Average Linkage

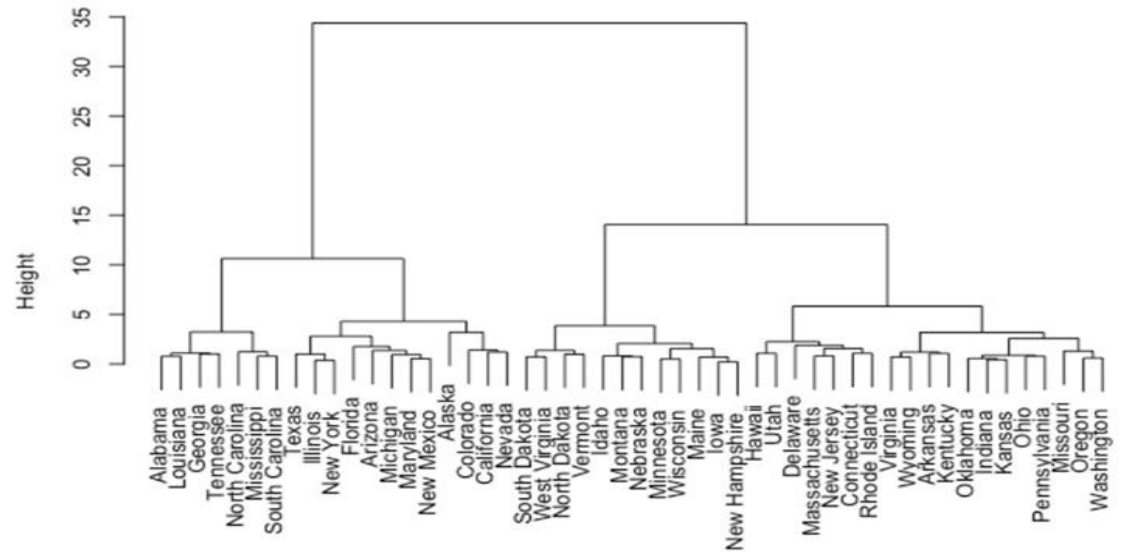


Single Linkage



d
hclust (*, "single")

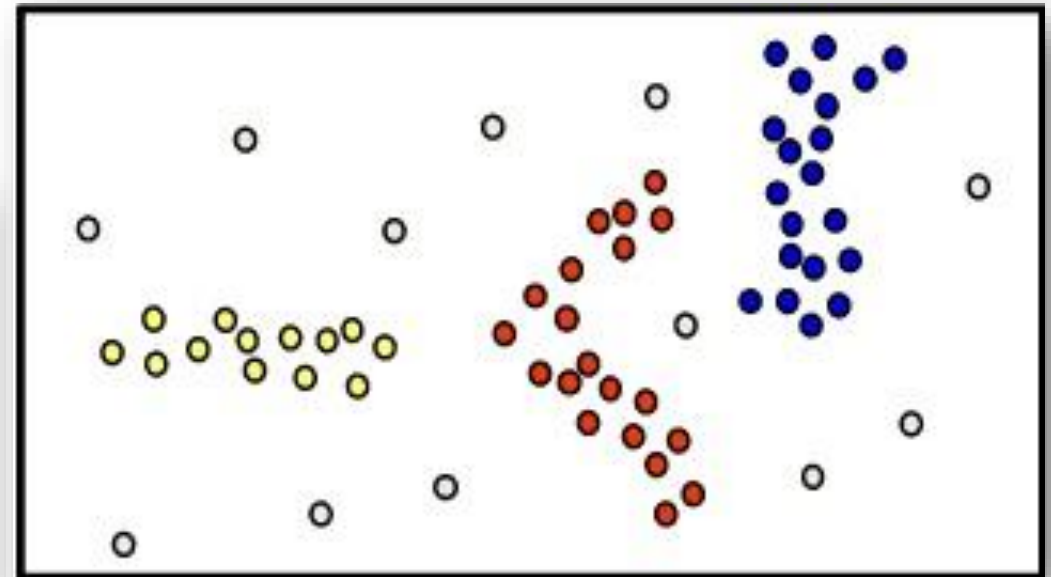
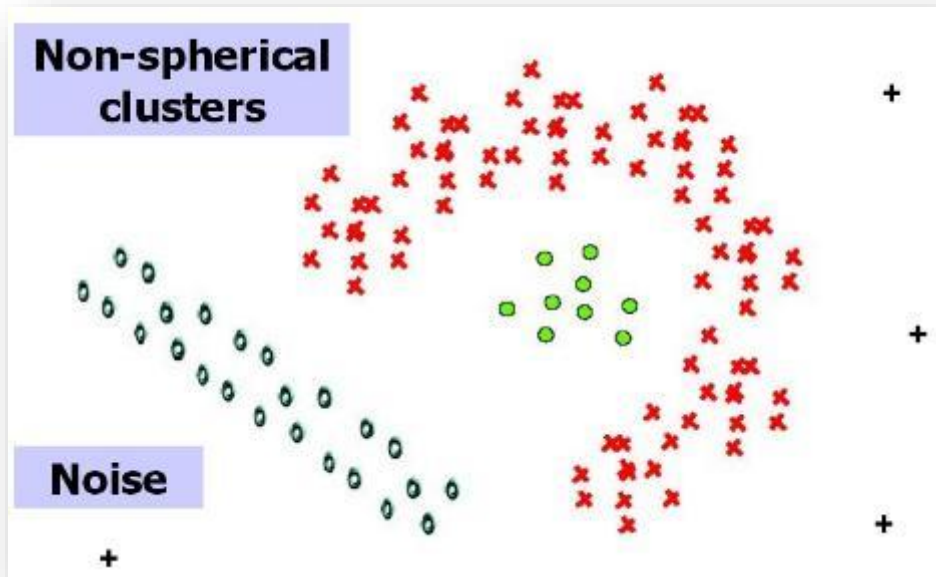
Ward's Method



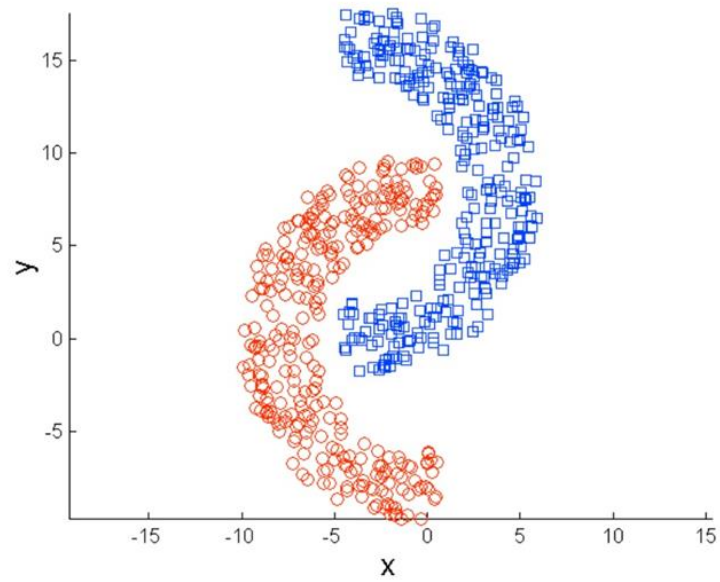
Types of Clusters: Density-Based

Density-based

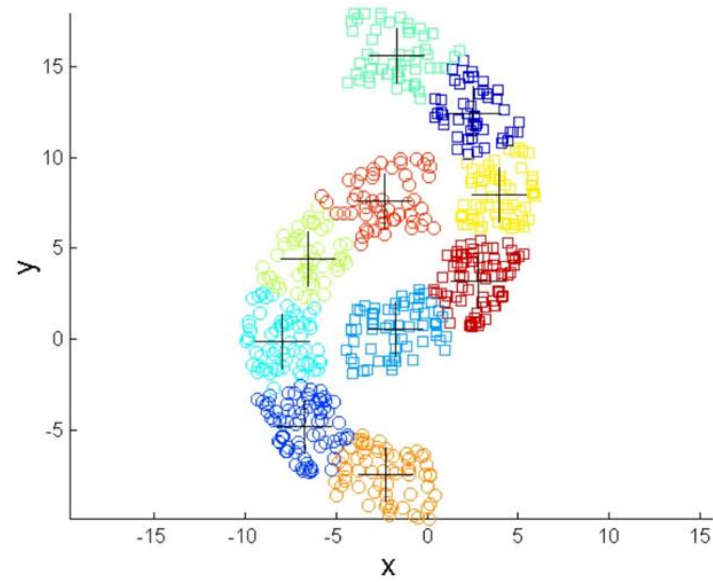
- A cluster is a dense region of points, which is separated by low-density regions, from other regions of high density.
- Used when the **clusters are irregular or intertwined**, and when noise and outliers are present.



Limitation of k-means



Original Points



K-means Clusters

DBSCAN: Density-Based Clustering

- DBSCAN is a Density-Based Clustering algorithm
- Reminder: In density based clustering we partition points into dense regions separated by not-so-dense regions.
- Important Questions:
 - How do we measure density?
 - What is a dense region?
- DBSCAN:
 - Density at point p : number of points within a circle of radius Eps
 - Dense Region: A circle of radius Eps that contains at least $MinPts$ points

Visual Example of k - means

- › 1) Choose k
- › 2) Randomly choose k centroids
- › 3) Place all points into one of the k clusters based on distance.
- › 4) Update the centroids
- › 5) Re-assign points to closest cluster

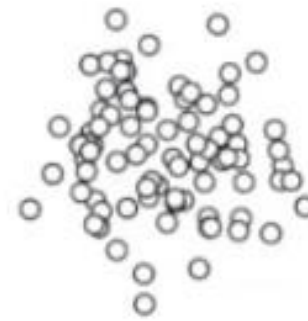
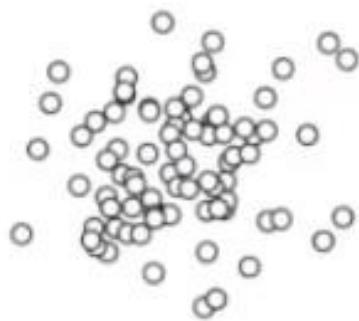
Repeat steps 4 and 5 for N iterations



π

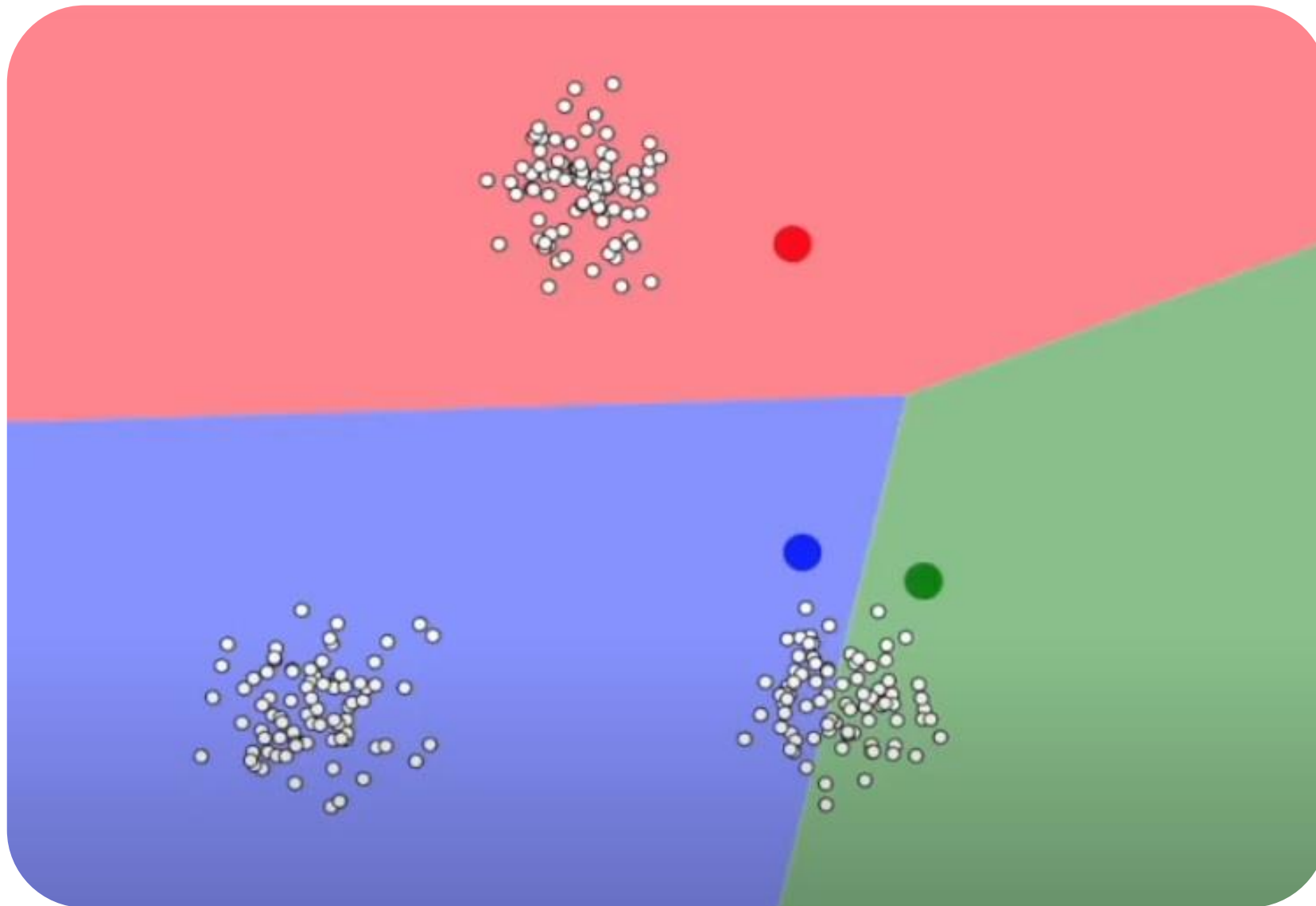


Choose k
 $= 3$



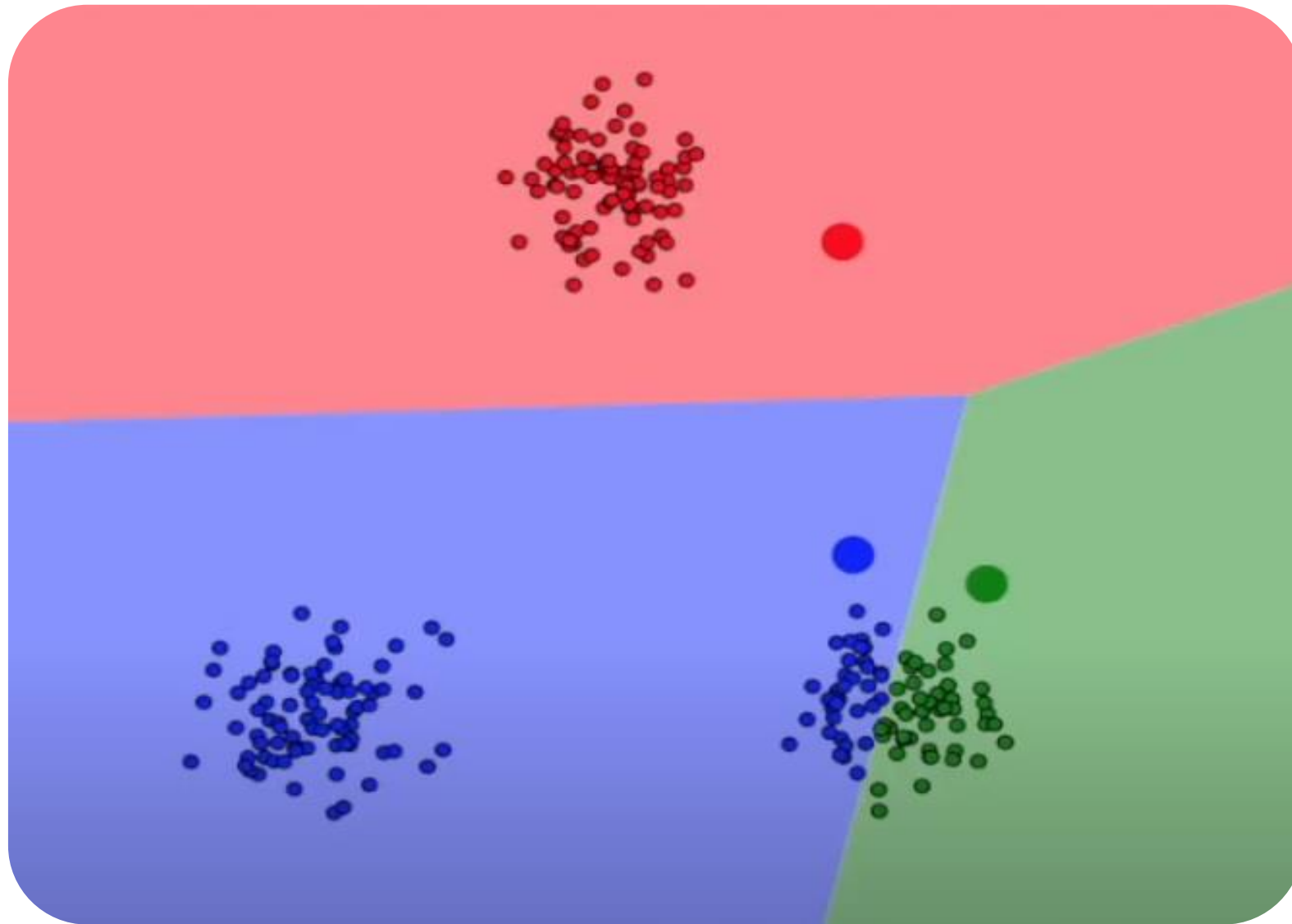
π

Randomly Choose Initial Centroids



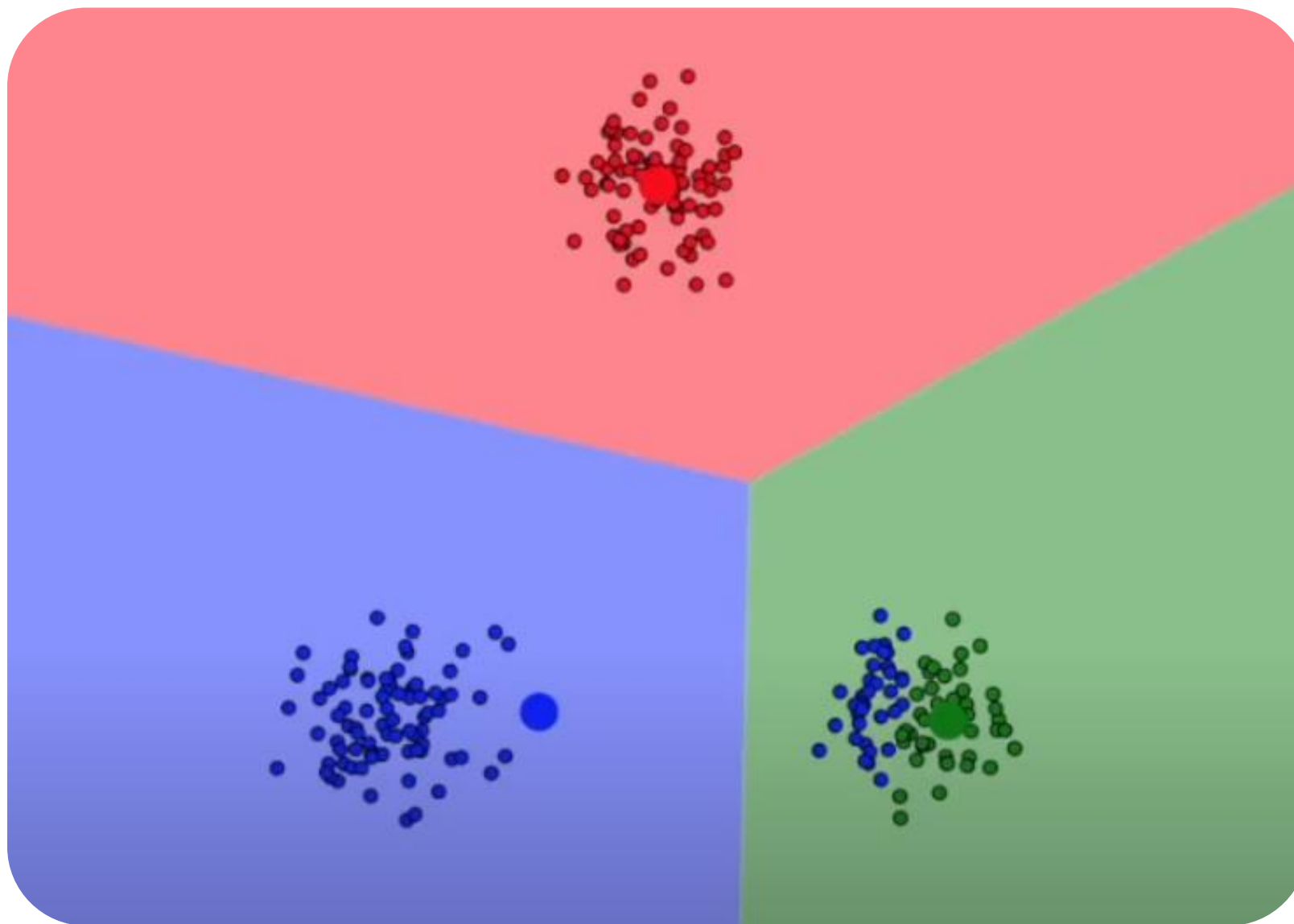
π

Place Points into Closest Cluster



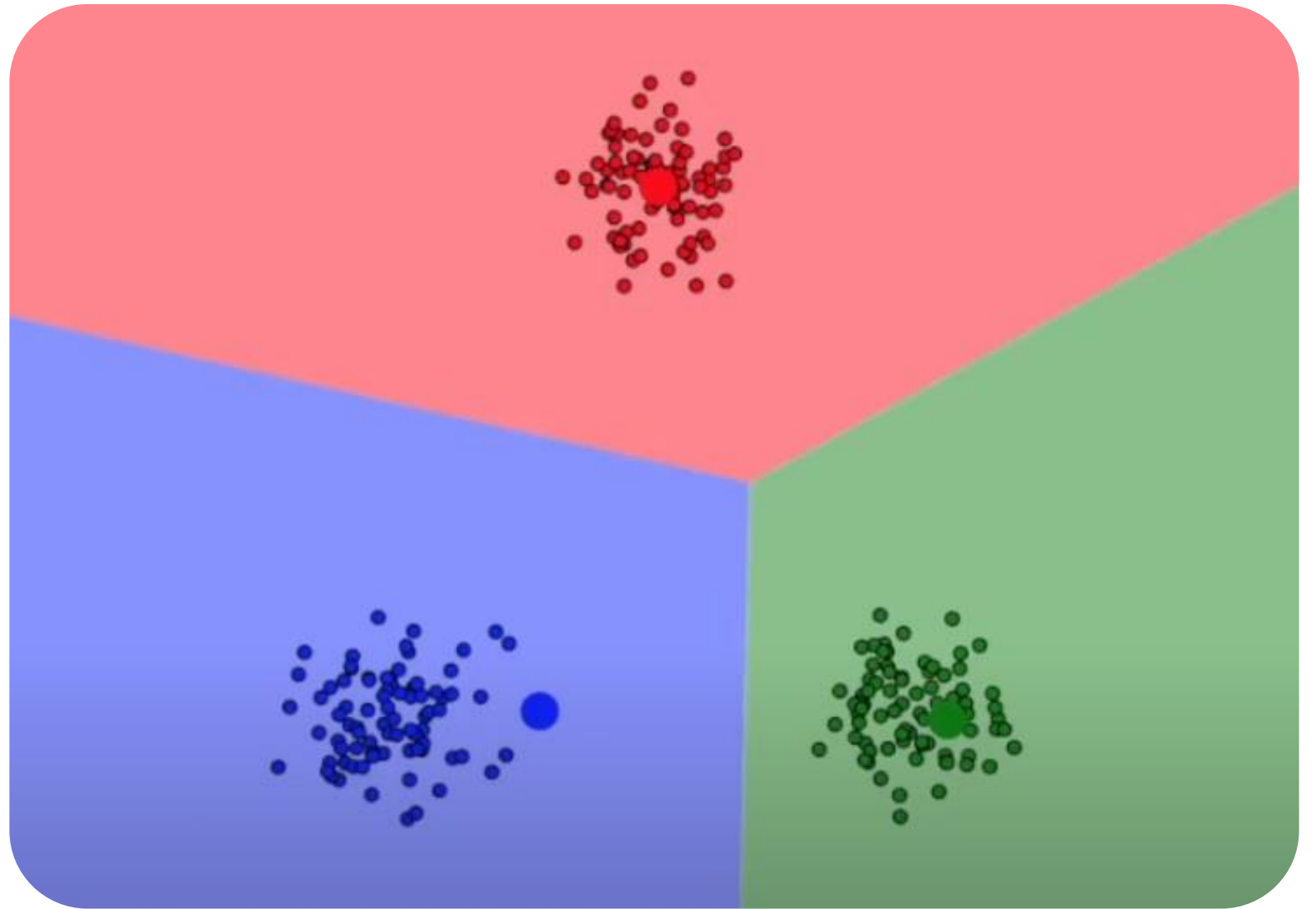
π

Update Centroids



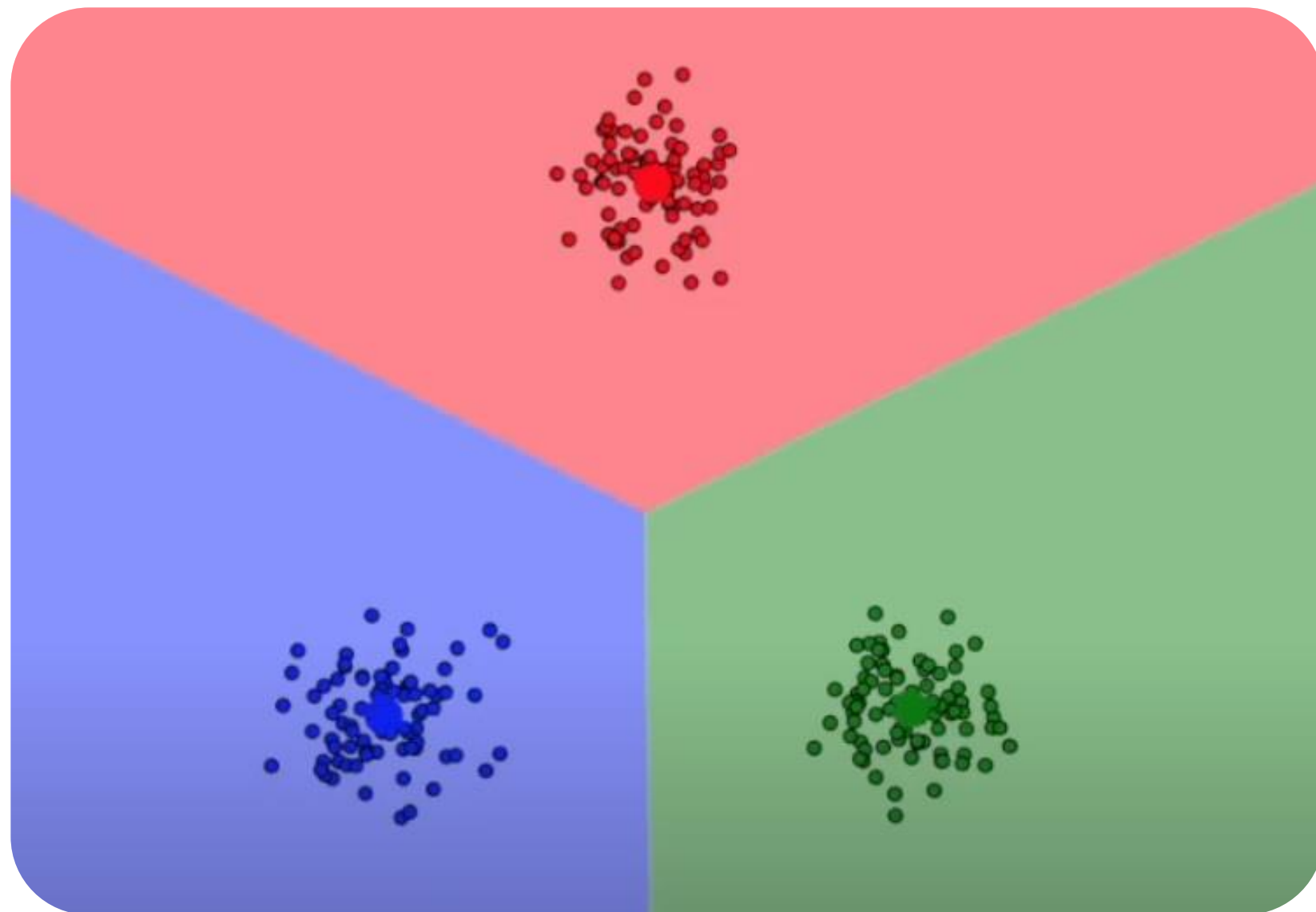
π

Re- calculate ALL distances from points to centroids and re-assign points to centroids as needed.

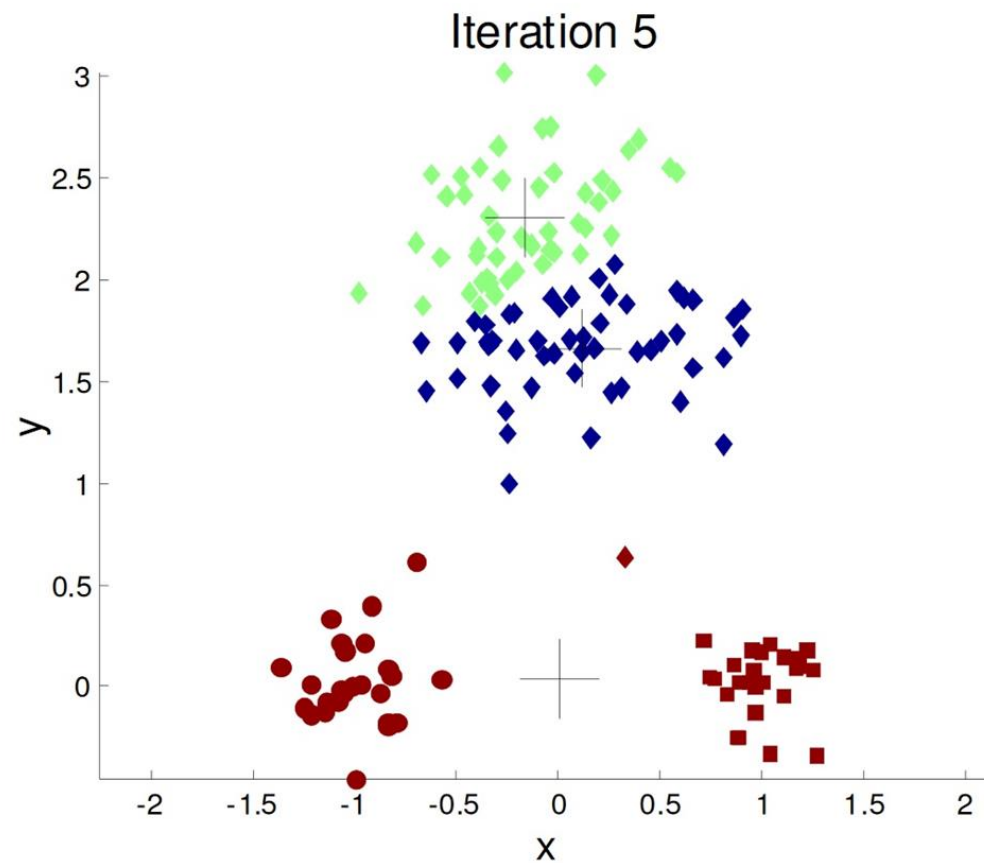


π

Recalculate Centroids

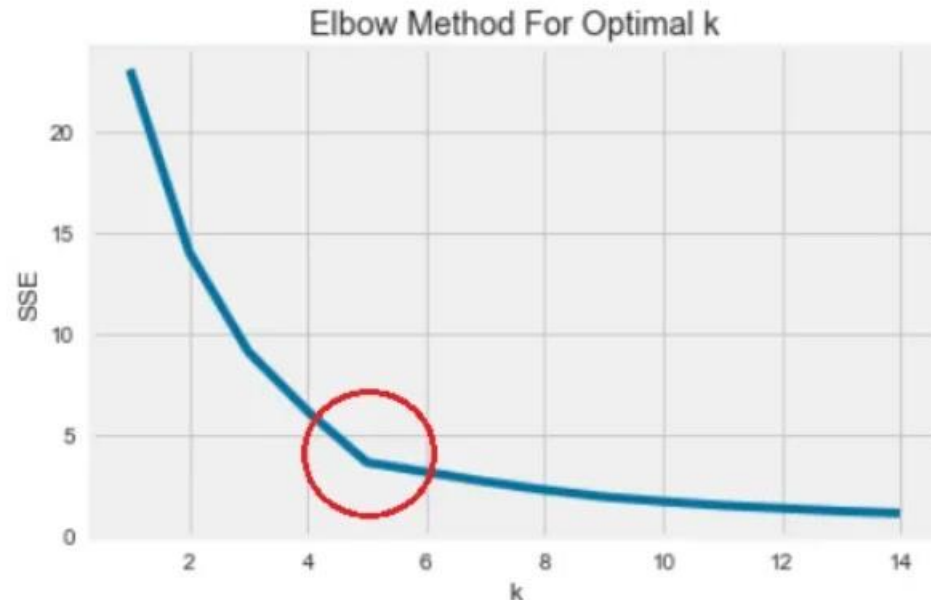


Importance of Choosing Initial Centroids



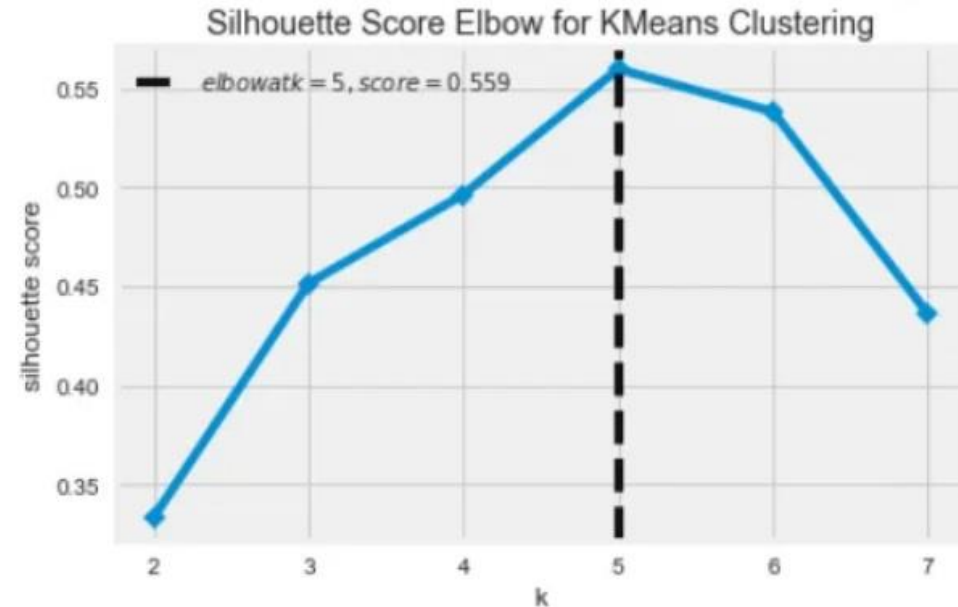
Choosing k: Elbow Method

- › **Elbow Method** : There is an “bending” point where the SSE (sum squared error) curve starts to **bend** known as the **elbow point**. The x-value of this point is thought to be a reasonable trade-off between error and number of clusters. The *elbowpoint* is the point where the rate of decrease of mean distance (from points to their centroid) will not change significantly with increase in number of clusters.



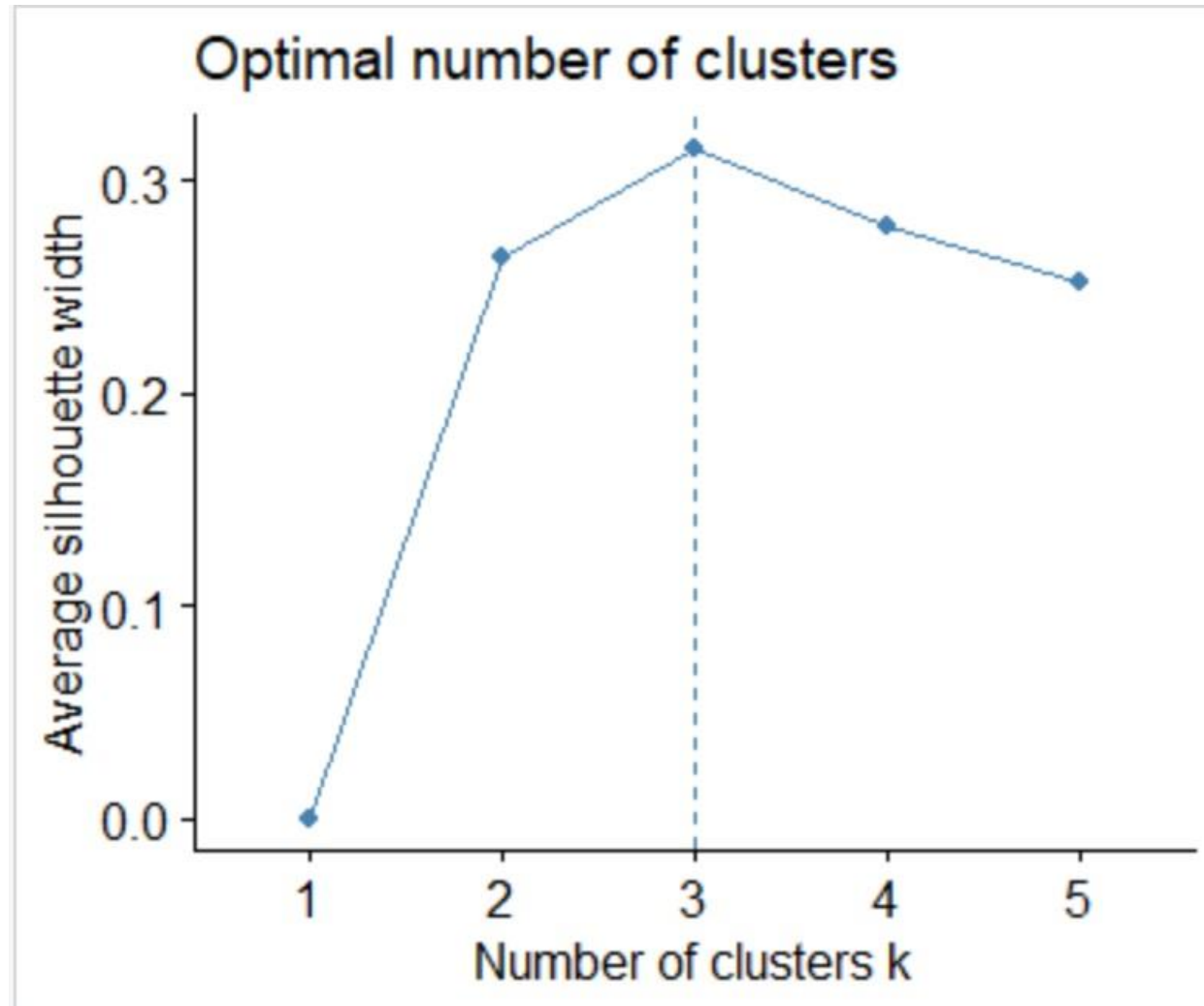
Silhouette Method

- › **Silhouette Coefficient** : is a measure of **cluster cohesion and separation**. It **quantifies** how well a data point fits into its assigned cluster based on two factors: **How close the data point is to other points in the cluster** and **how far away the data point is from points in other clusters**.
- › Silhouette coefficient values range between -1 and 1. Larger numbers indicate that samples are closer to their clusters than they are to other clusters.



<https://nzlul.medium.com/clustering-method-using-k-means-hierarchical-and-dbscan-using-python-5ca5721bbfc3>

```
library("factoextra")  
 $\pi$ (fviz_nbclust(MyDF, kmeans, method='silhouette', k.max=5))
```



Part 2

The Math and Code Behind K Means

The Code

By Hand – In Python

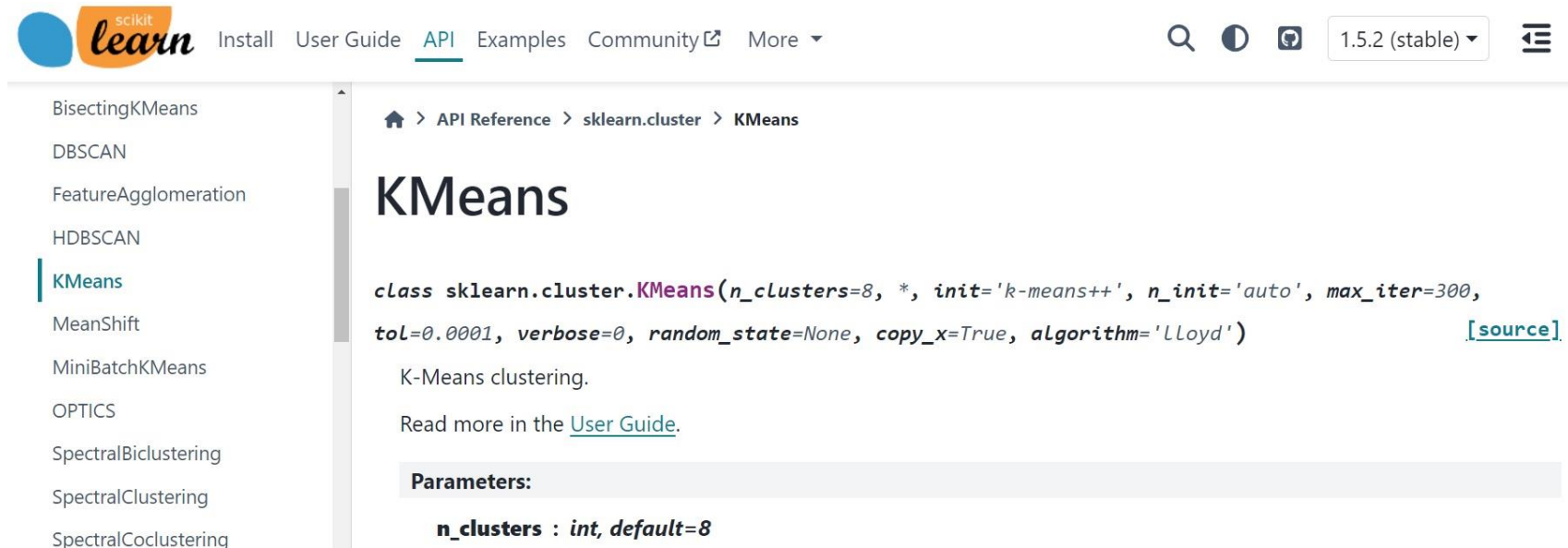
https://gatesboltonanalytics.com/?page_id=924

Using Sklearn in Python

https://gatesboltonanalytics.com/?page_id=262

and

<https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html>



The screenshot shows the scikit-learn website's API reference for the `KMeans` class. The page includes a navigation menu with links for 'Install', 'User Guide', 'API', 'Examples', 'Community', and 'More'. The 'API' link is selected. The breadcrumb trail is 'API Reference > sklearn.cluster > KMeans'. The main heading is 'KMeans'. Below the heading, the class signature is displayed: `class sklearn.cluster.KMeans(n_clusters=8, *, init='k-means++', n_init='auto', max_iter=300, tol=0.0001, verbose=0, random_state=None, copy_x=True, algorithm='Lloyd')` with a '[source]' link. A brief description follows: 'K-Means clustering.' and a link to 'Read more in the User Guide.' A 'Parameters:' section is visible, listing `n_clusters : int, default=8`.

K Means – the Math and EM

Suppose we have a dataset $\{x_1, x_2, \dots, x_n\}$

We choose k , the number of clusters.

Let μ_k be the centroid (mean) of cluster k .

Goal: Assignment points to clusters so that the sum of the square distances between points and their closest cluster is minimized.

For each datapoint x_n , let r_{ik} in $\{0,1\}$ be a **binary indicator function** such that if x_n is assigned to μ_k then $r_{nk} = 1$.

What is our Loss (or Objective) Function that we want to Optimize?

$$\mathcal{L} = \sum_{k=1}^K \sum_{i=1}^n r_{ik} \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2,$$

where $r_{ik} = 1$ if $\mathbf{x}_i \in S_k$ and 0 otherwise.

Our goal is to find values of $\boldsymbol{\mu}_k$ and r_{ik} that minimize \mathcal{L} .

Steps:

Initialization: Choose μ_k

1) E Step: Update r_{ik}

Minimize L wrt r_{ik} while keeping μ_k fixed.

To do this, we place points into their closest clusters.

2) M Step: Update μ_k

Update L wrt μ_k while keeping fixed r_{ik} .

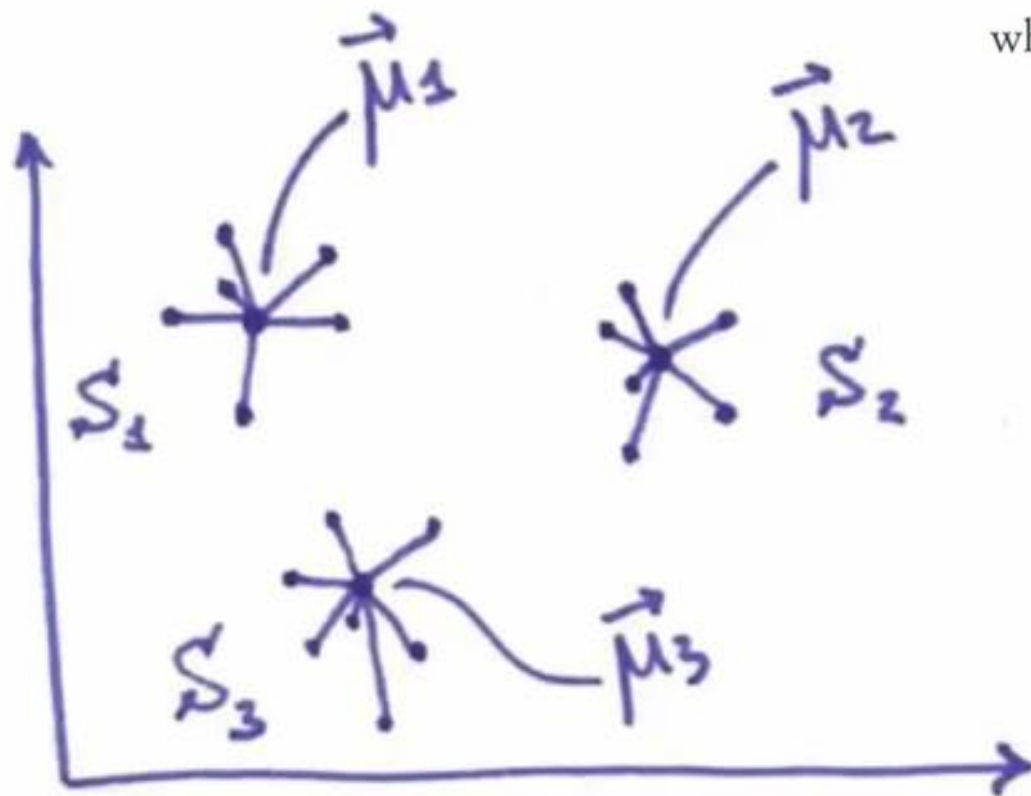
To do this, we recalculate each cluster mean, μ_k , based on the points in those clusters.

3) Repeat until convergence. Note that convergence may not be to a global min.

Make sure it makes sense loss-wise

$$\mathcal{L} = \sum_{k=1}^K \sum_{i=1}^n r_{ik} \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2,$$

where $r_{ik} = 1$ if $\mathbf{x}_i \in S_k$ and 0 otherwise.



The Math: The best r_{ik}

$$\mathcal{L} = \sum_{k=1}^K \sum_{i=1}^n r_{ik} \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2,$$

where $r_{ik} = 1$ if $\mathbf{x}_i \in S_k$ and 0 otherwise.

The E Step:

- › All data points are considered to be independent.
- › Therefore, we can choose r_{ik} to be 1 for whichever k gives the minimum value of $\|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2$. This just means that we can place all the points in their “closest” clusters.

The Math: The best mean

The M Step: Updating μ_k with r_{ik} fixed.

$$\mathcal{L} = \sum_{k=1}^K \sum_{i=1}^n r_{ik} \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2,$$

where $r_{ik} = 1$ if $\mathbf{x}_i \in S_k$ and 0 otherwise.

Take $d\mathcal{L}/dr_{ik} = 2 \sum_n r_{ik} (\mathbf{x}_i - \boldsymbol{\mu}_k)$

Set this to 0 and solve.

We get: $\boldsymbol{\mu}_k = \sum_n r_{ik} \mathbf{x}_i / \sum_n r_{ik}$

Here, the denominator is the number of points assigned to cluster k (because otherwise it is 0).

So in short, this is just the mean (average) of the points in each cluster for all k clusters.

In Summary

- 1) First, choose k .
- 2) Next, randomly choose points to represent each cluster center.

$$\mathcal{L} = \sum_k \sum_{i \in S_k} \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2 = \sum_{k=1}^K \sum_{i=1}^n r_{ik} \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2$$

Not analytically solvable. Not convex. Gradient descent can be messy.

Alternative approach (Lloyd's algorithm): iteratively optimize over r_{ik} and over $\boldsymbol{\mu}_k$.

- For fixed $\boldsymbol{\mu}_k$: assign each point \mathbf{x}_i to the nearest cluster center.

$$i \in S_k \text{ if } k = \arg \min_j \|\mathbf{x}_i - \boldsymbol{\mu}_j\|^2.$$



Assign each point to the nearest cluster

- For fixed r_{ik} :

$$\boldsymbol{\mu}_k = \frac{1}{|S_k|} \sum_{i \in S_k} \mathbf{x}_i.$$



Calculate the mean of each cluster
- mean of the points currently assigned to it.

Repeat Until Convergence

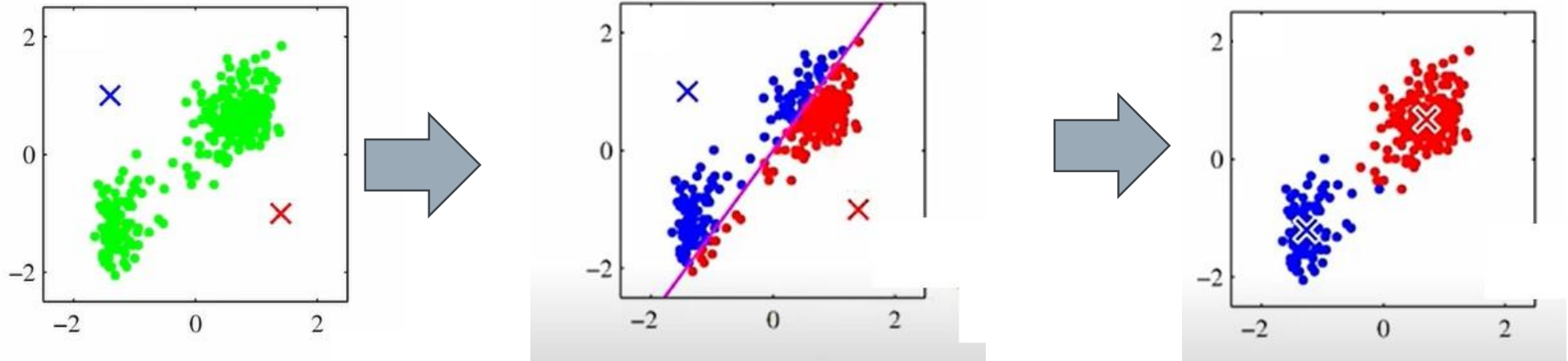
Here, we repeat the EM steps until we converge.

Convergence may not result in a global min.

In words, the two phases are:

E: Reassign points (this updates r_{ik})

M: Calculate the mean (this updates μ_k)



Bishop, *Pattern Recognition and Machine Learning*

Bishop, *Pattern Recognition and Machine Learning*

For K-means, The **Expectation(E)** step is where each data point is assigned to the “nearest” cluster and the **Maximization(M)** step is where the centroids are recomputed using the mean of the points in that centroid.

Notes

- 1) The K means algorithm is generally based on the use of the Euclidean distance and the measure of similarity.
- 2) This limits the type of data that can be considered as “*it would be inappropriate for cases where some or all of the variables represented categorical labels*” (Bishop, 2006)

Part 3:

Mild Introduction to Expectation
Maximization for Gaussian Mixture Models

Expectation Maximization (EM)

- › The Expectation-Maximization algorithm is a very influential and widely used machine learning algorithm.
- › **K-means is special variant of the EM algorithm** with the assumption that the clusters are spherical and we use only the mean to determine a cluster centroid (not a standard deviation as well).
- › EM is a method to find the **maximum likelihood** estimate of parameters, θ , in a **latent** variable model.
- › EM is made up of an “E” step and an “M” step. It starts with random values for the parameters, θ , and then alternates between:
 - **In the E step**, the algorithm computes the latent variables - expectation of the log-likelihood using the current parameter estimates.
 - **In the M step**, the algorithm determines the parameters that maximize the expected log-likelihood obtained in the E step, and corresponding model parameters are updated based on the estimated latent variables.

Example:

- › Suppose you are a hiker and are interested in the temperature and humidity distribution during both summer and winter in Colorado.
- › You ask a friend who lives in CO to gather N temperature and humidity datapoints so you can calculate the mean and variance. You are assuming a gaussian distribution.
- › BUT, (pretend with me here), your friend gives you the N datapoints *without* telling you whether they were gathered during summer or winter.
- › In this case, the season (summer or winter) is now a **latent variable**.
- › Next, at this moment, we are **missing two bits of information**. For any given datapoint, we do not know the season it was gathered in. In addition, we also do not know the mean and variance **for each season**.
- › **One more note:** If we knew which season (summer or winter) each datapoint came from, we could solve this fast, Just take the mean and variance for the points gathered in that season. **But we don't!**
- › Also, if we knew the mean and variance for each season, then we could place all datapoints into the correct gaussian (season) based on which it is closer to. **But we don't.**
- › In cases like these, we use EM.

EM Assumptions and a Random Start

In EM (and in kmeans) - we must know “k”.

In k means, k is the number of clusters (latent variables) we have. In EM, it is the number of distributions (latent) that we have.

In our example, we have $k = 2$, Summer and Winter.

- › Next, in EM, we start the algorithm by randomly choosing our parameters and then assigning all points to whichever they are “closest to”. Technically, we need to ML here – but we will come back to that.
- › In k-means, this was easy because our “parameters” are just the means of our centroids. We can initialize kmeans by randomly choosing our initial centroids (means).
- › However, suppose we have a Gaussian Mixture instead. This means we have two parameters: mean and variance for each Gaussian. We will still choose them randomly and we will still “assign” all the points to them based on a measure of similarity. However, assigning points in k means only requires a distance metric (like Euclidean distance). Assigning points to gaussians is a little bit more complicated (but not much!)
- › NOTE: Mahalanobis Distance is used to measure the “distance” between a datapoint and a gaussian distribution.

Given a probability distribution Q on \mathbb{R}^N , with mean $\vec{\mu} = (\mu_1, \mu_2, \mu_3, \dots, \mu_N)^\top$ and positive semi-definite covariance matrix S , the Mahalanobis distance of a point $\vec{x} = (x_1, x_2, x_3, \dots, x_N)^\top$ from Q is^[6]

$$d_M(\vec{x}, Q) = \sqrt{(\vec{x} - \vec{\mu})^\top S^{-1} (\vec{x} - \vec{\mu})}.$$

EM for Gaussian Mixture Models (GMMs)

- › Note that EM can be applied broadly. GMMs is one example that is useful in explaining how EM works.
- › To explain EM as applied to GMM, we will need to gain some understanding of GMM, of Maximum Likelihood (ML), and then of EM.

Gaussian Mixture Models

This is a Gaussian mixture distribution. It is a true probability and it sums to 1.

Gaussian mixture:

$$p(\mathbf{x}) = \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k).$$

Intuitively, we could use the same iterative approach as in the Lloyd's algorithm for K -means:

- 1) Assign each point to the "closest" gaussian.
- 2) Update the parameters of the gaussians based on their assigned points.

Here, we are looking at GMs in terms of discrete latent variables.

$p(z_k=1) = \pi_k$ where z defines a "cluster" – in this case – a Gaussian.

For example, if we have a mixture of 3 Gaussians, then a point \mathbf{x} can be a member of a gaussian with probability such that the sum of the probabilities is 1.

If $z_k = 1$,

Likelihood in GMM

Gaussian mixture:

$$p(\mathbf{x}) = \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k).$$

Log-likelihood:

$$\mathcal{L} = \sum_{i=1}^n \log \left[\sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_i \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right],$$

where $\mathcal{N}(\mathbf{x}_i \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \dots \exp \left(-\frac{1}{2}(\mathbf{x}_i - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1}(\mathbf{x}_i - \boldsymbol{\mu}_k) \right)$.

Set the derivative with respect to $\boldsymbol{\mu}_k$ to zero:

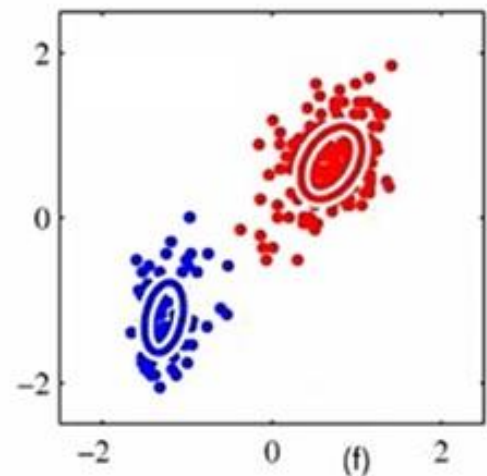
$$-\sum_{i=1}^n \underbrace{\frac{\pi_k \mathcal{N}(\mathbf{x}_i \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^K \pi_j \mathcal{N}(\mathbf{x}_i \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}}_{z_{ik}} \boldsymbol{\Sigma}_k (\mathbf{x}_i - \boldsymbol{\mu}_k) = 0.$$

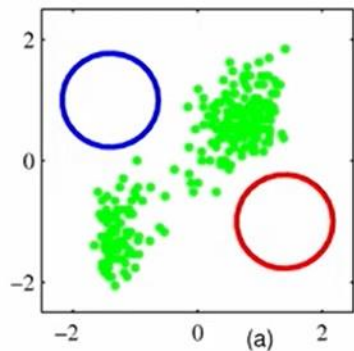
$$\sum_{i=1}^n z_{ik} (\mathbf{x}_i - \boldsymbol{\mu}_k) = 0.$$

$$\boldsymbol{\mu}_k = \frac{\sum z_{ik} \mathbf{x}_i}{\sum z_{ik}}.$$

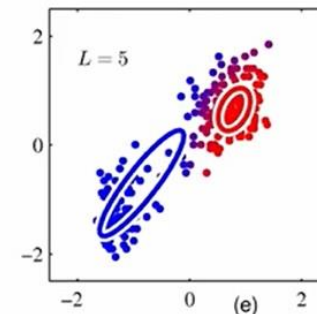
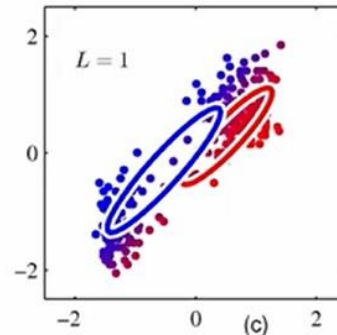
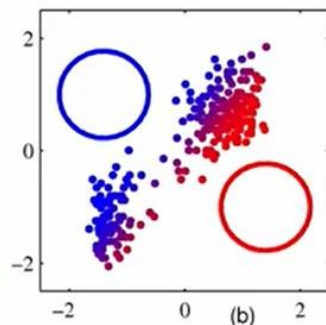
This is a *weighted* mean of all points.

Very similar derivation shows that $\boldsymbol{\Sigma}_k$ should be the weighted covariance matrix, and $\pi_k = \sum z_{ik}/n$.





Important:
Notice that the colors range. This is because each point is assigned to a Gaussian using probability (not just 0 or 1 like in kmeans).



Bishop, *Pattern Recognition and Machine Learning*

Bishop, *Pattern Recognition and Machine Learning*

Bishop, *Pattern Recognition and Machine Learning*

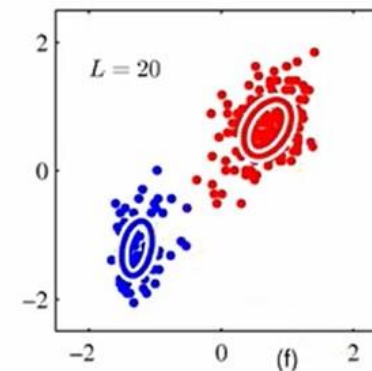
Bishop, *Pattern Recognition and Machine Learning*

Expectation-maximization algorithm iteratively alternates between updating μ_k , Σ_k , π_k and updating z_{ik} :

- **E-step:** compute the posterior probability z_{ik} for each point to be in each Gaussian component.
- **M-step:** update the parameters (μ_k , Σ_k , π_k) of each Gaussian using weighted averages.

EM is a very generic algorithm to optimize likelihood in probabilistic models with *latent variables*. (In GMMs, latent variables are true class memberships.) E-step computes posterior over latent variables, conditioned on the parameters of the model. M-step optimizes the parameters, conditioned on the latent variables.

Notice that the final gaussians have different means and variances.



References

1) Bishop's Book

<https://www.microsoft.com/en-us/research/uploads/prod/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf>

2)