Unsupervised Learning

Dr Xuhui Fan

Based on the slides from
Yipeng Zhou, Steve Cassidy, Jia Wu, Unnikrishnan P.C. , Pulkit Sharma

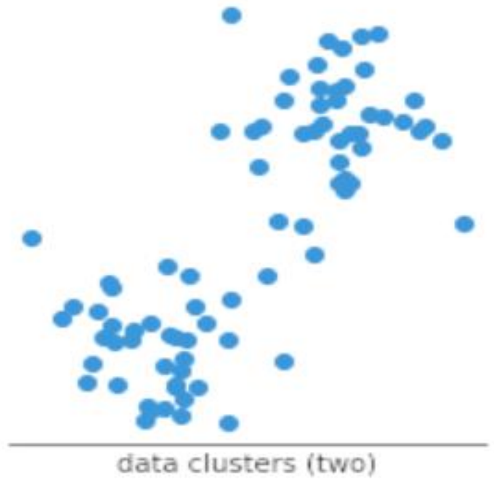
Agenda

- Background
- Clustering
- Data Normalisation
- KMeans Clustering
  - Example to provide a step-by-step guide to the algorithm
- Hierarchical Clustering
  - Example to provide a step-by-step guide to the algorithm
- Summarised Key Points

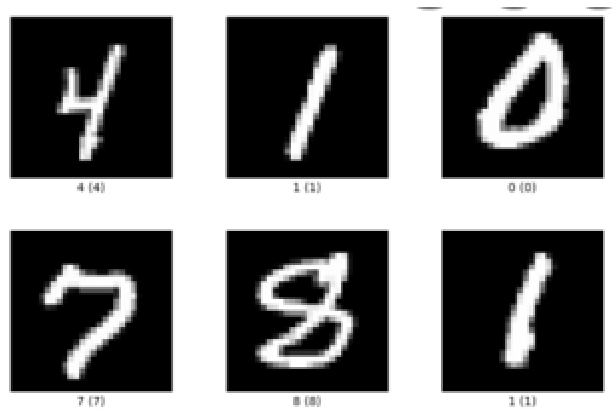
Background

- Unsupervised learning: find hidden structure in **<u>unlabelled data</u>**.
- Examples:
  - **<u>Clustering</u>**: has as a goal to partition the data into groups;

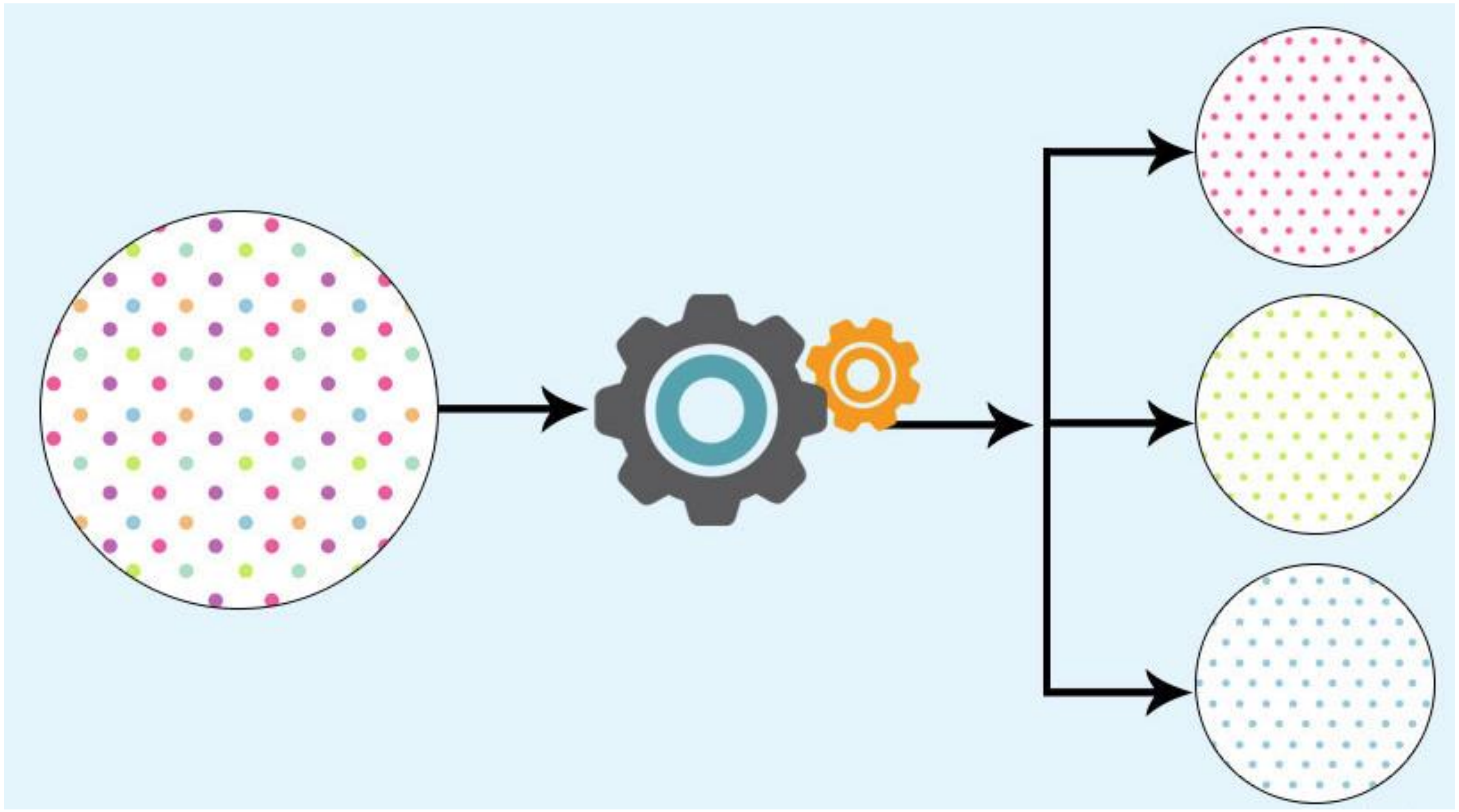
Hard to label for grouping



Easy to label for classification



Clustering



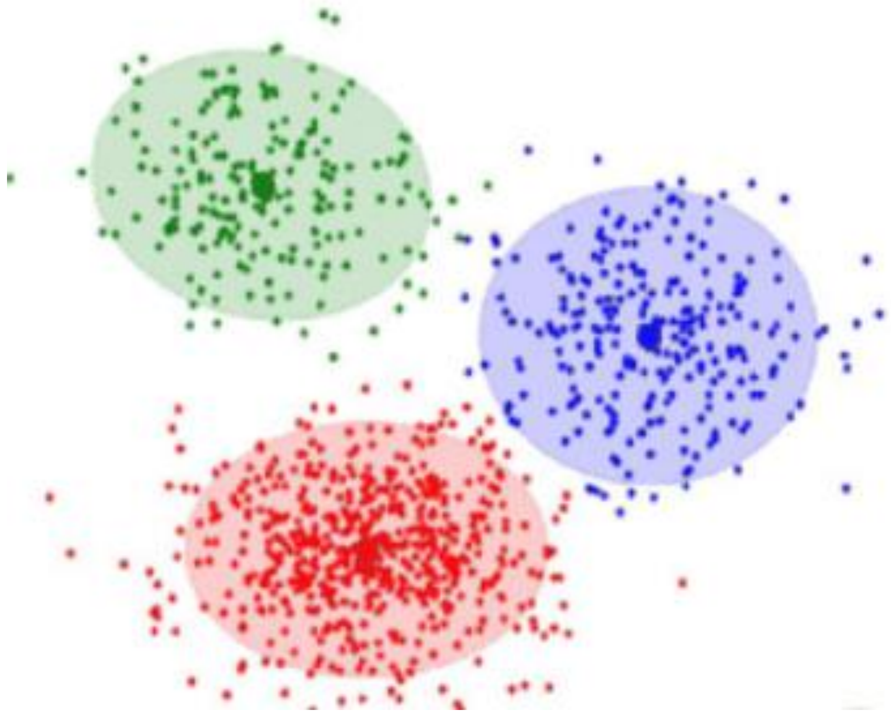
Clustering

- Clustering is a process of grouping similar objects together; i.e., to partition unlabeled examples into disjoint subsets of clusters, such that:
  - Samples **within a cluster are similar** (i.e., high intraclass similarity).
  - Samples **in different clusters are different** (i.e., low interclass similarity).

Clustering Examples

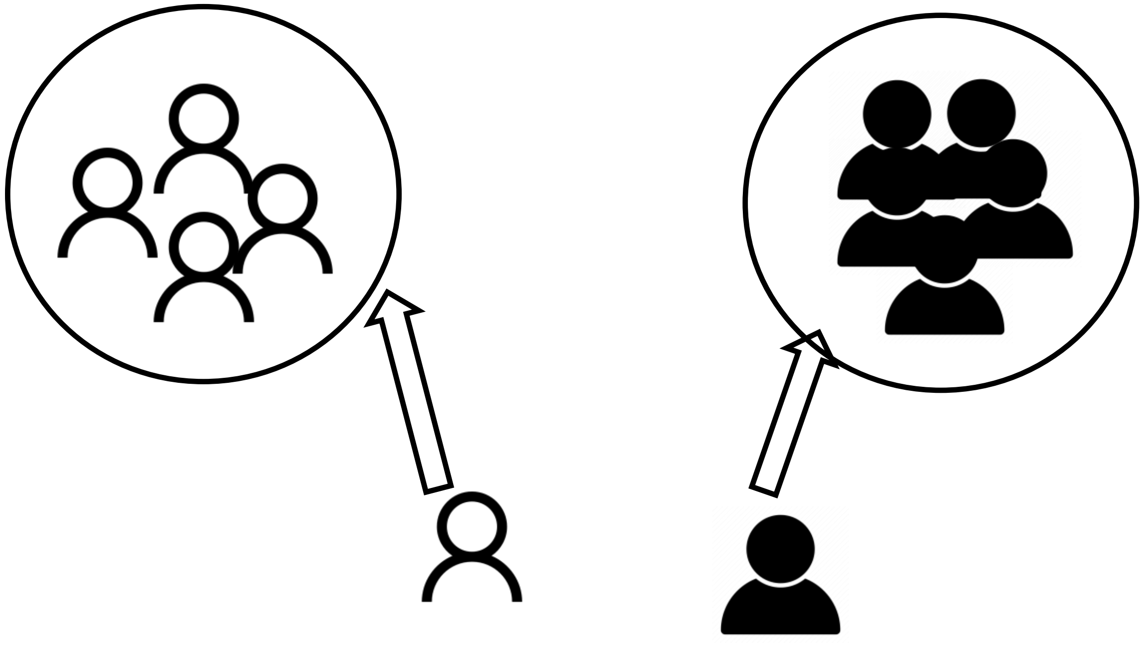
- Examples:
  - Outlier detection: find unusual events (e.g., a malfunction), that distinguish part of the data from the rest according to certain criteria.

Points not in any cluster are suspicious outliers



Clustering Examples

- Customer segmentation for recommendation of online shopping, online video, etc.

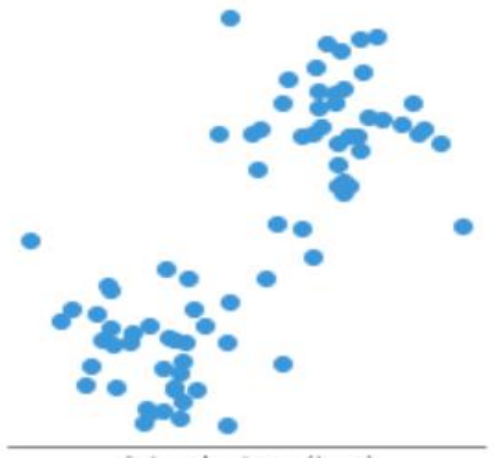


Measuring Similarity

- In order to find clusters in a data set we need to have some way of telling how similar two observations are to each other.
- The first step in understanding clustering is understanding how to measure similarity.

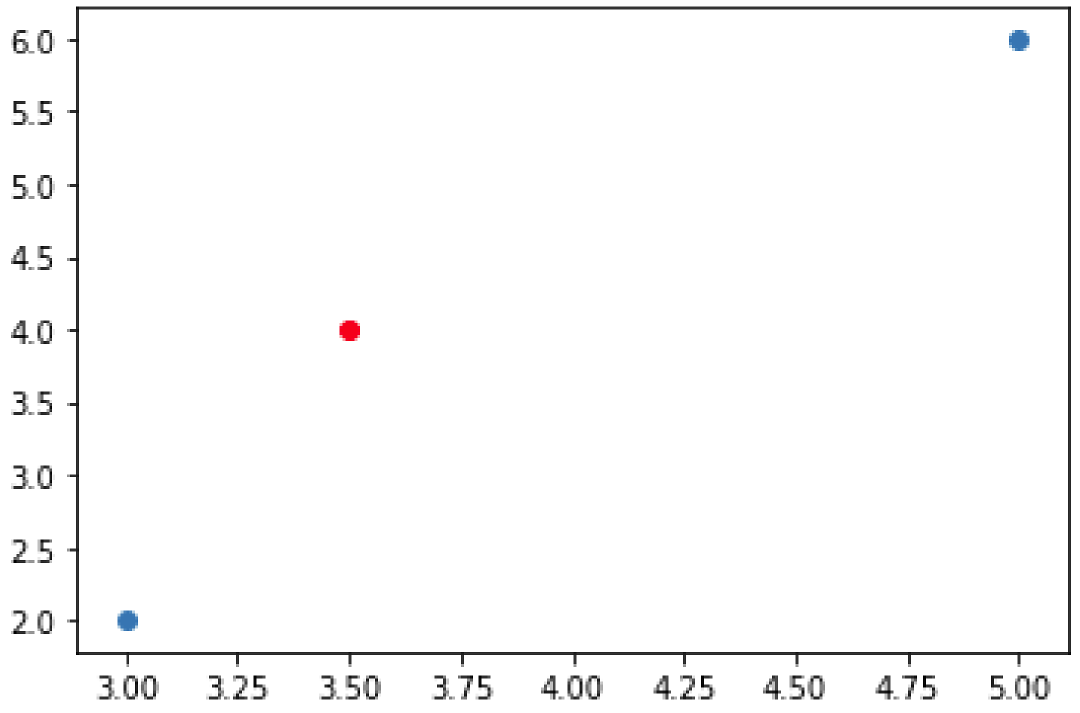
Similarity and Distance

- The key to understanding similarity metrics is to think about measuring distance in the real world.
- When we graph a two dimensional data set, we get a layout of points on the plane. From this it is easy to see when two points are similar - they are close to each other on the graph.


data clusters (two)

Similarity and Distance

- In the example below, we'd say that the middle (red) point (3.5,4) is closer or more similar to the first point (3,2) than (5,6).

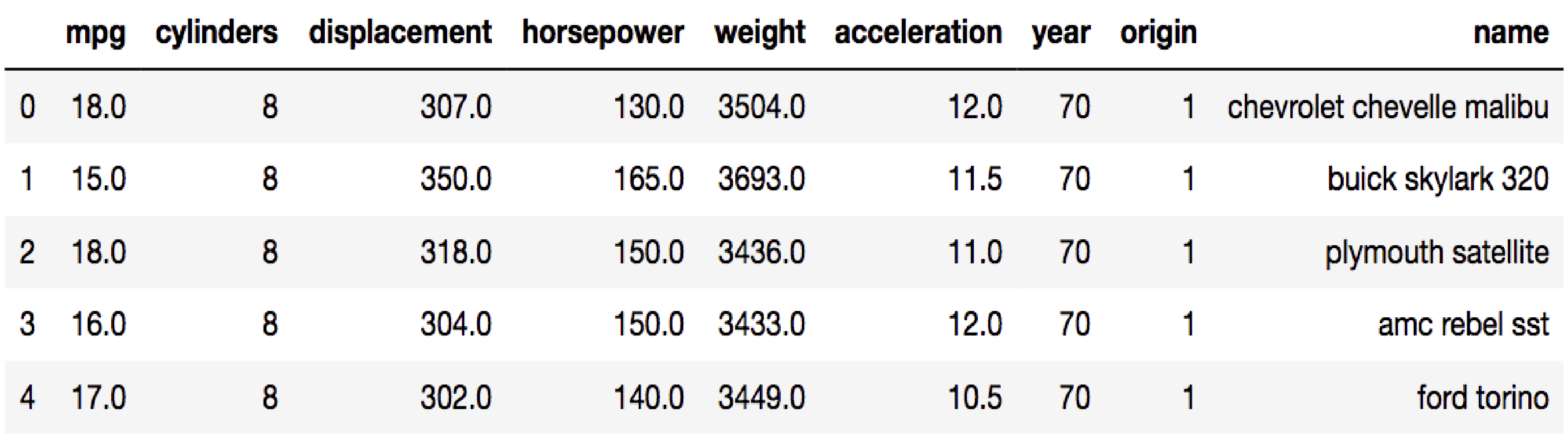


Similarity and Distance

- In two dimensions, we can measure the distance between the points (x_1, y_1) and (x_2, y_2) using Euclidean distance:
  - $d(a, b) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$
- This can be generalised into more dimensions like this:
  - $d(a, b) = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2}$
    - where a_i is the ith feature for point a. It's easy to picture this in two or three dimensions, when you have more dimensions, the maths is the same but you can't picture it.

Data Normalisation

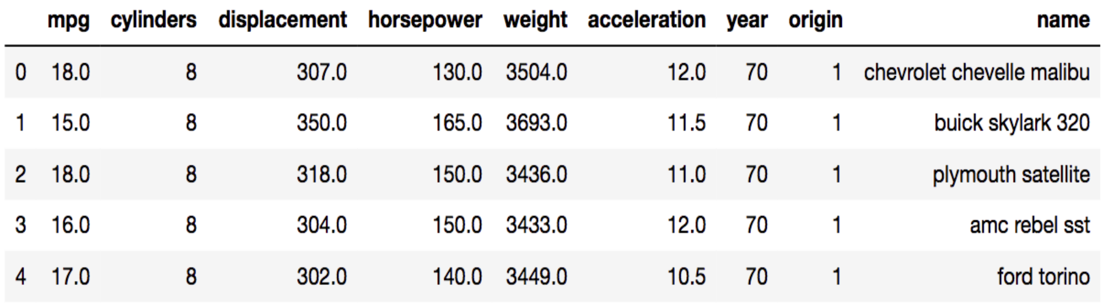
- Let's look at some data for describing cars and their MPG ratings:



	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
0	18.0	8	307.0	130.0	3504.0	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693.0	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150.0	3436.0	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150.0	3433.0	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140.0	3449.0	10.5	70	1	ford torino

- We could ask which is more similar to the **A** “chevrolet chevelle malibu” - the **B** “buick skylark 320” or **C** the "plymouth satellite"?

Data Normalisation



	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
0	18.0	8	307.0	130.0	3504.0	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693.0	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150.0	3436.0	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150.0	3433.0	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140.0	3449.0	10.5	70	1	ford torino

- Results:
  $d(A,B) = 196.9879$
  $d(A,C) = 71.7356$
- Normalising Variables:
  - in this case, the units of each variable are quite different
    - cylinders varies from 3 to 8
    - displacement from 68 to 455
    - weight from 1613 to 5140
  - this means that a difference in weight would dominate the other variables
  - solution is to normalise the data

Normalising Variables

- Normalise to make the mean and standard deviation the same for every variable
- For each column:
  - subtract the mean
  - divide by the standard deviation
- Result is data with mean of 0, std of 1.0
- Without normalisation:

  $d(A,B) = 196.9879$

  $d(A,C) = 71.7356$

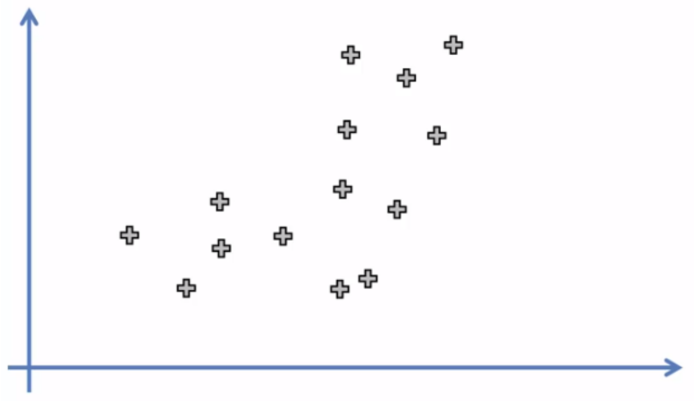
  With normalisation:

  $d(A,B) = 1.1071$

  $d(A,C) = 0.6472$

Example

- Given $x_1, x_2, \ldots, x_n$ points, find centroids to minimize the total distance between each point and its closest centroid.
- If k=1, $\min \Sigma_{i=1}^{n} \parallel x_i - \mu_x \parallel^2 + \parallel y_i - \mu_y \parallel^2$
- If k=2, we have two centroids: (μ_x^1, μ_y^1) , (μ_x^2, μ_y^2)
- The objective $\min \Sigma_{i=1}^{n} \parallel x_i - \mu_x^{(i)} \parallel^2 + \parallel y_i - \mu_y^{(i)} \parallel^2$
- $(\mu_x^{(i)}, \mu_y^{(i)})$ is the one closest to i

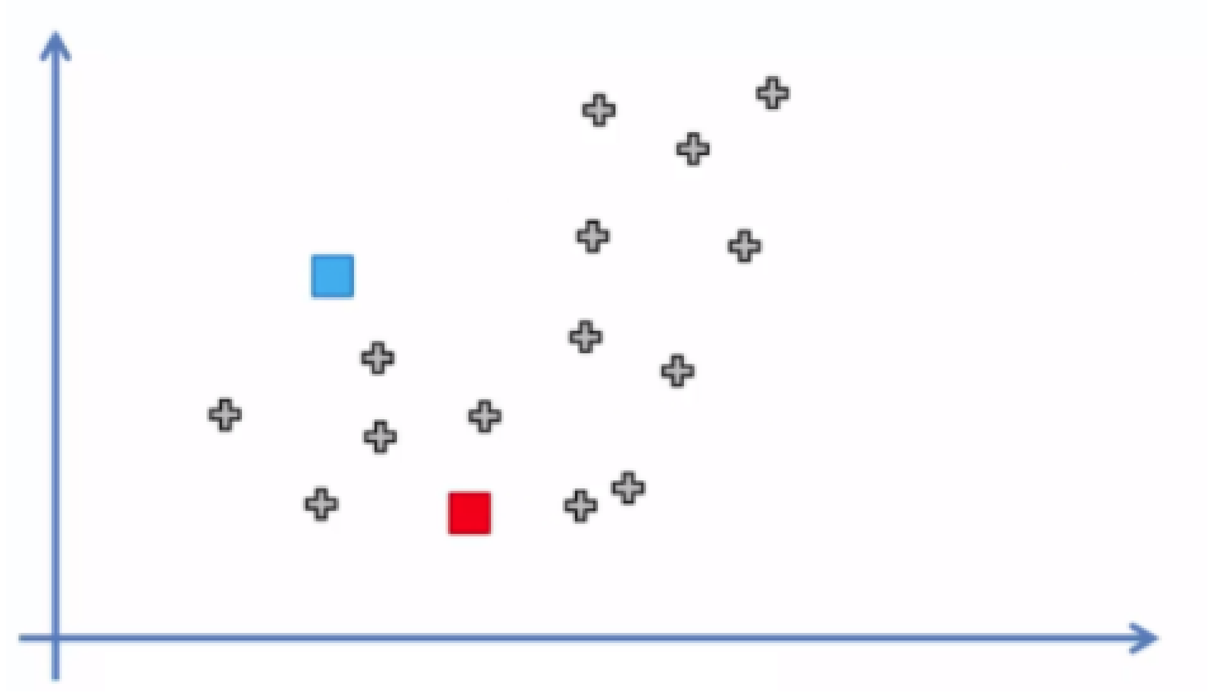


KMeans Clustering

- The basic algorithm is:
  1) start with K randomly placed cluster centroids;
  2) for each data point, assign it to the closest cluster centroid;
  3) re-estimate the cluster centroids for each cluster based on the data points that belong to it;
  4) if the centroid estimate has not changed significantly, terminate, otherwise repeat from step 2).

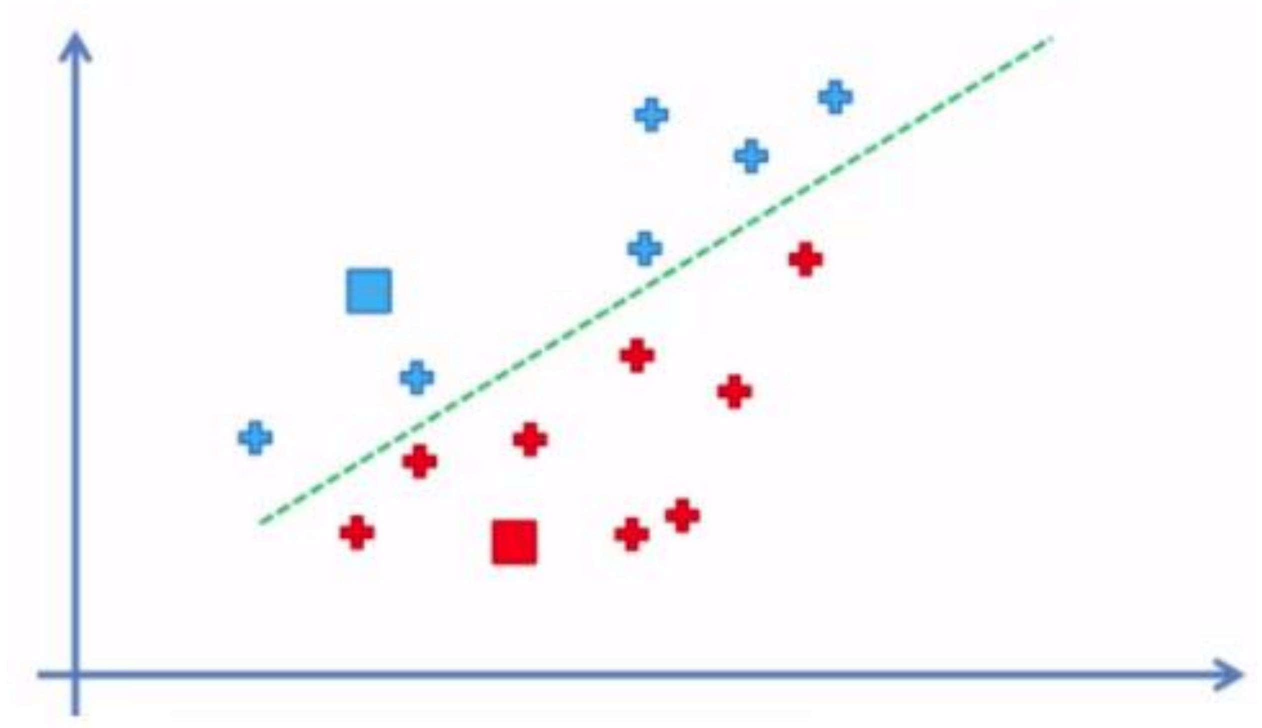
Step 1)

- Start with K randomly placed cluster centroids;
  - Initialization: Randomly we choose following two centroids (*k*=2) for two clusters.



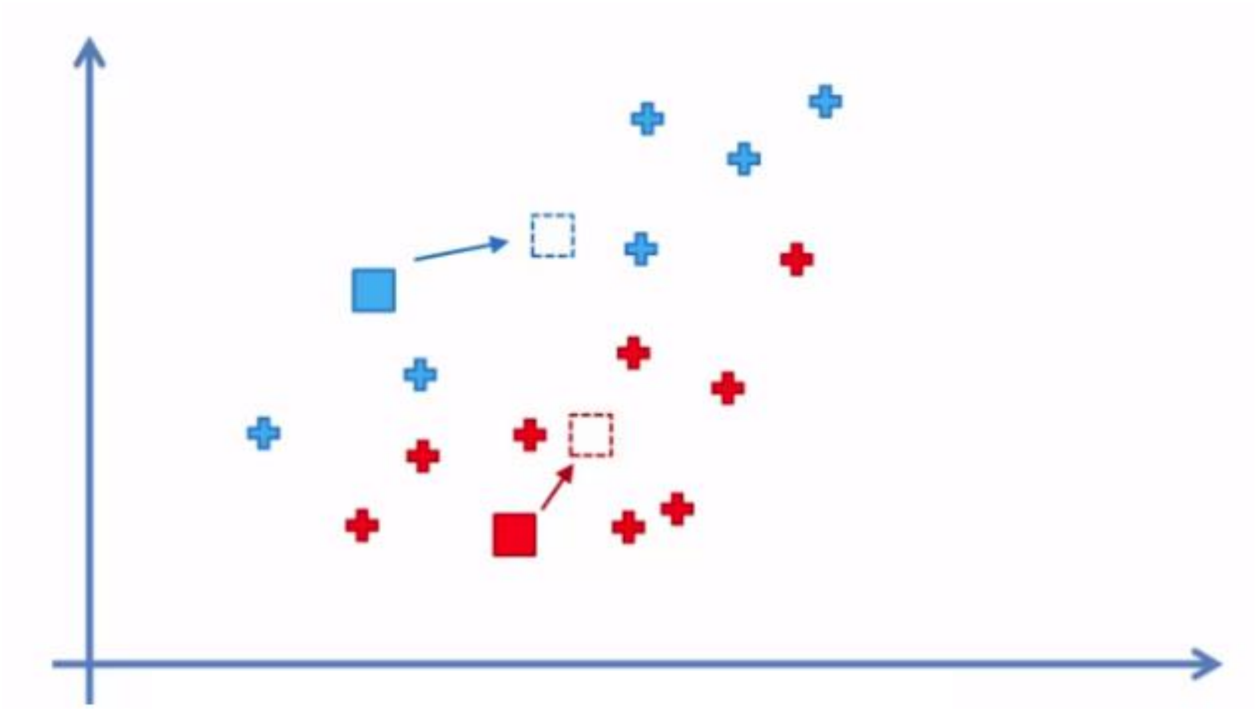
Step 2)

- For each data point, assign it to the closest cluster centroid;



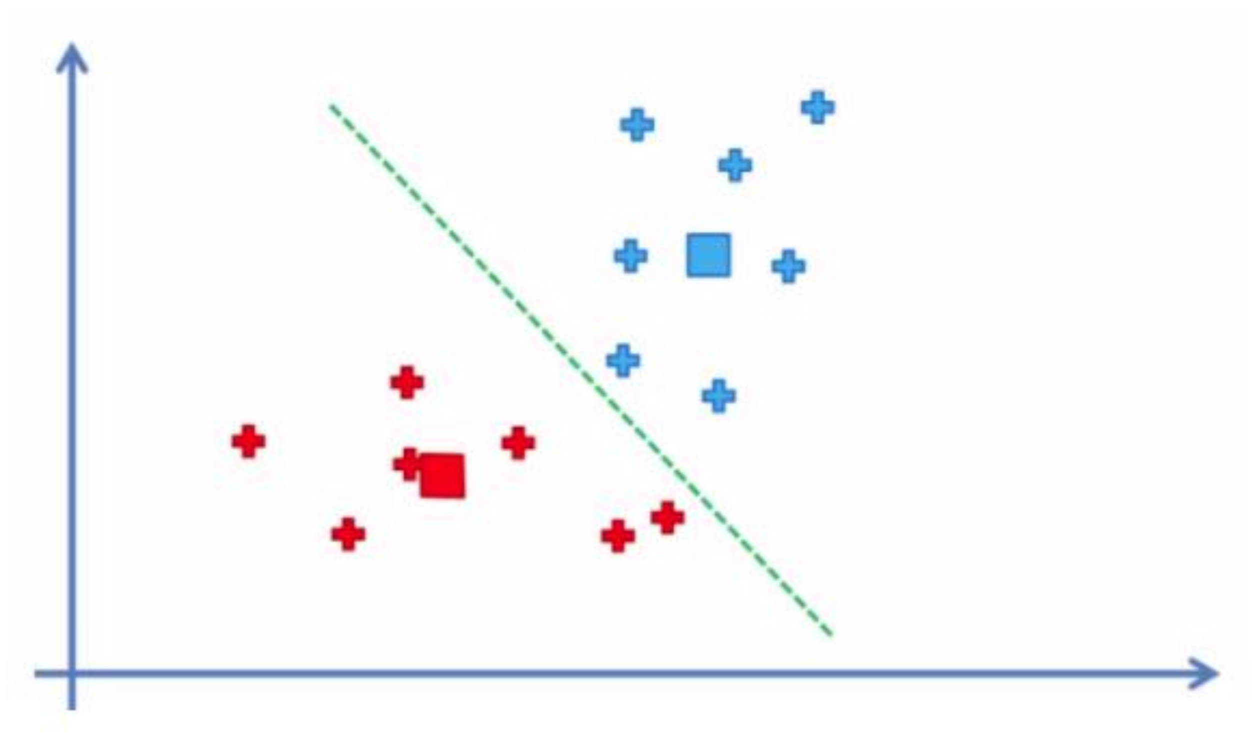
Step 3)

- Re-estimate the cluster centroids for each cluster based on the data points that belong to it;



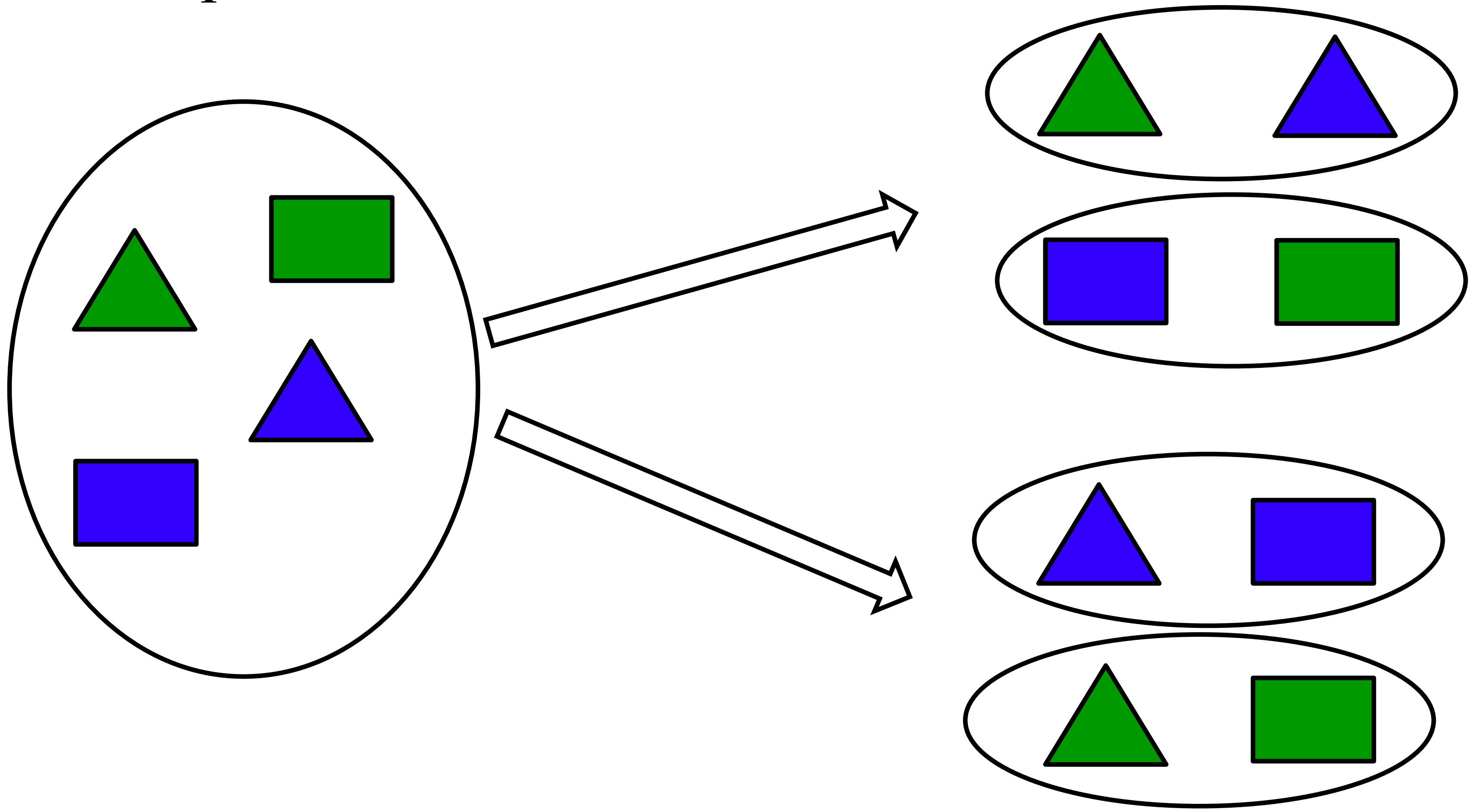
Step 4)

- If the centroid estimate has not changed significantly, stop, otherwise repeat from step 2).



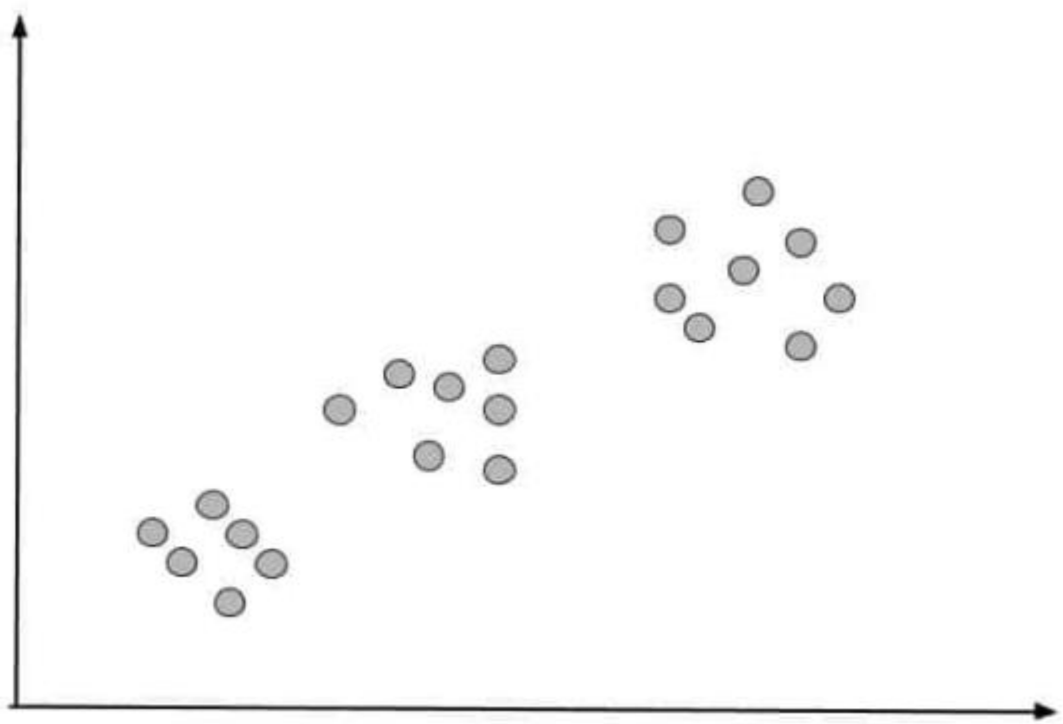
Influence of Different Measures

We can define distances with different features to recognise different patterns.



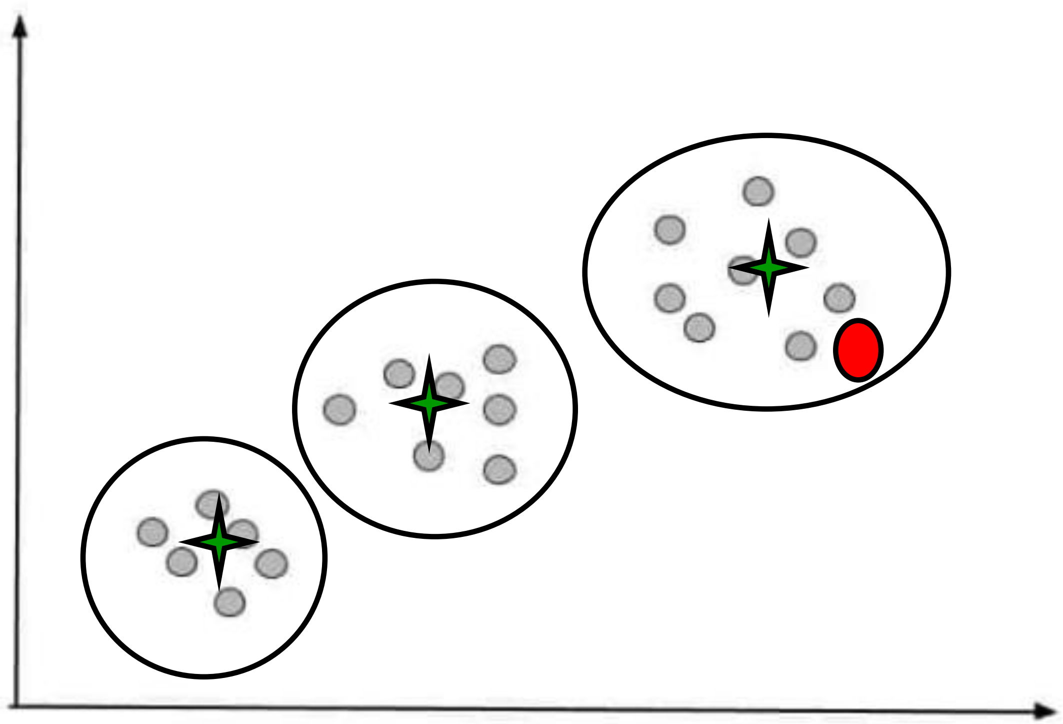
Overfitting v.s. Underfitting

- How many clusters should we select?
- Can we decide # of clusters automatically?
- The clustering error always decreases with # of clusters.
- Overfitting/underfitting occurs if # of clusters is over large/small



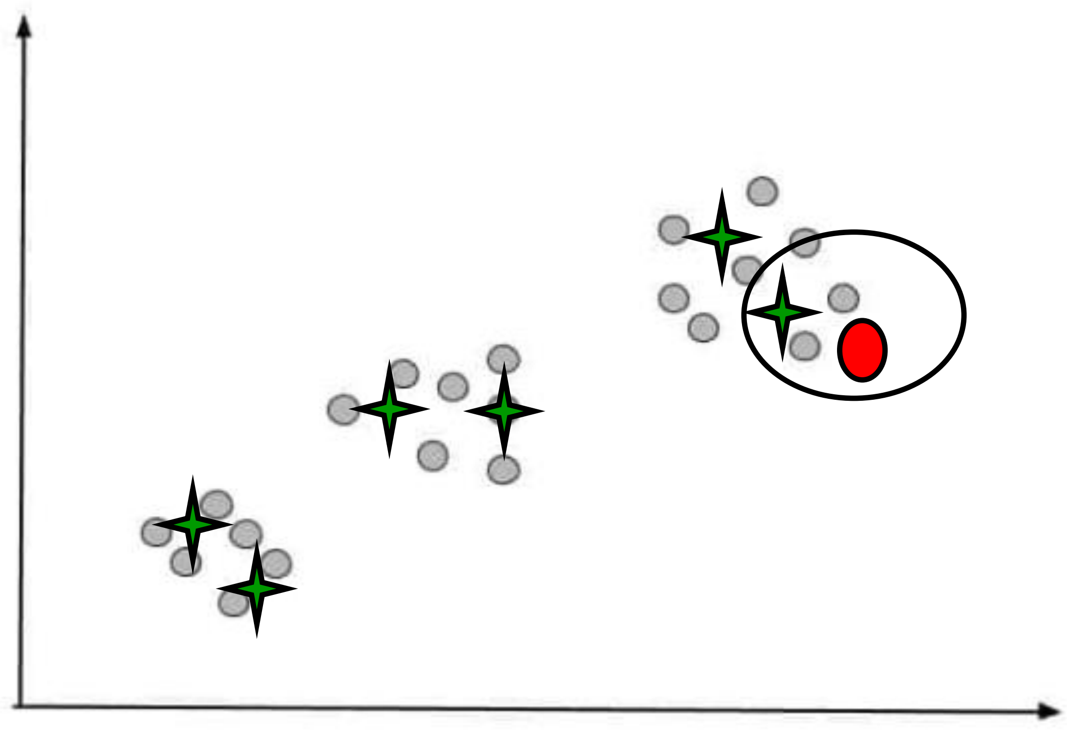
Overfitting

- Extreme case: 100 points, 100 clusters
- Take recommendation as an example. Clustering similar users so that we can make recommendation based on similar behaviour in the same cluster.



Overfitting

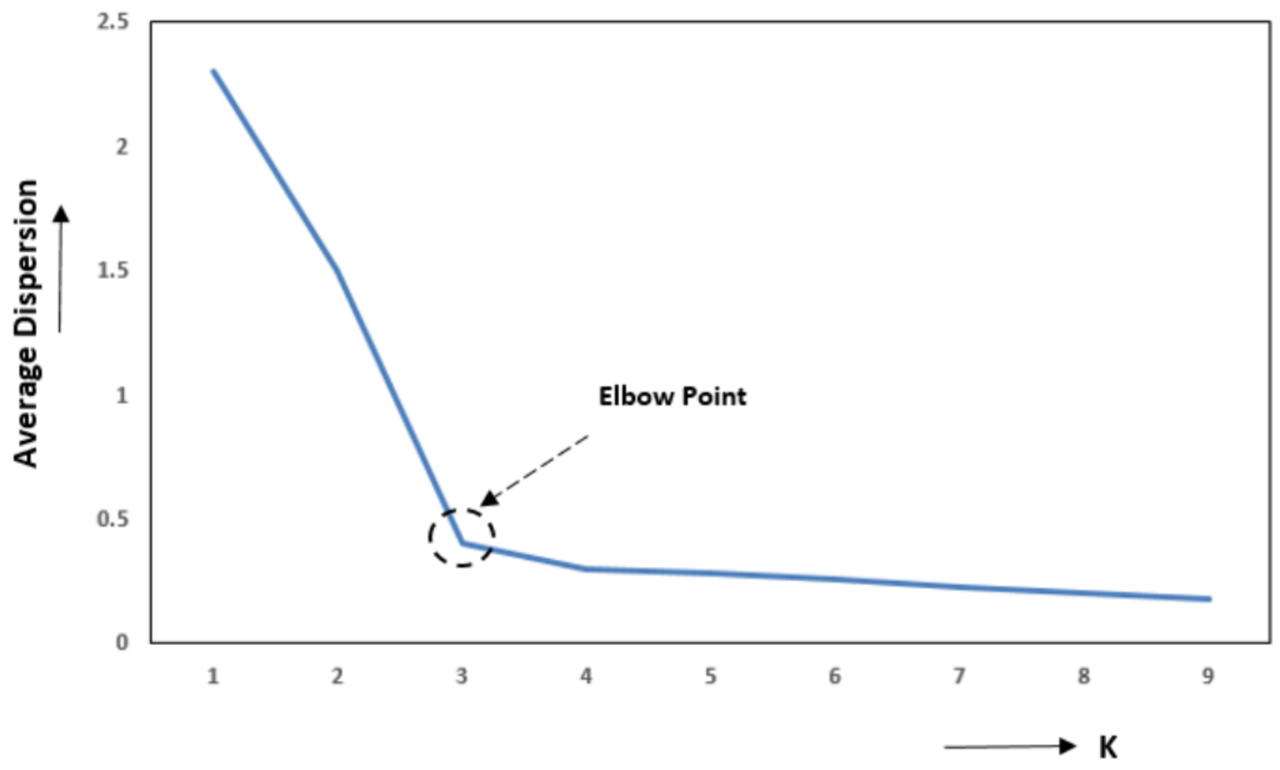
- Overfitting occurs if we cluster users into 6 clusters.
- It is not as accurate as that with 3 cluster to predict the behaviour of the red point.



Overfitting [Advanced]

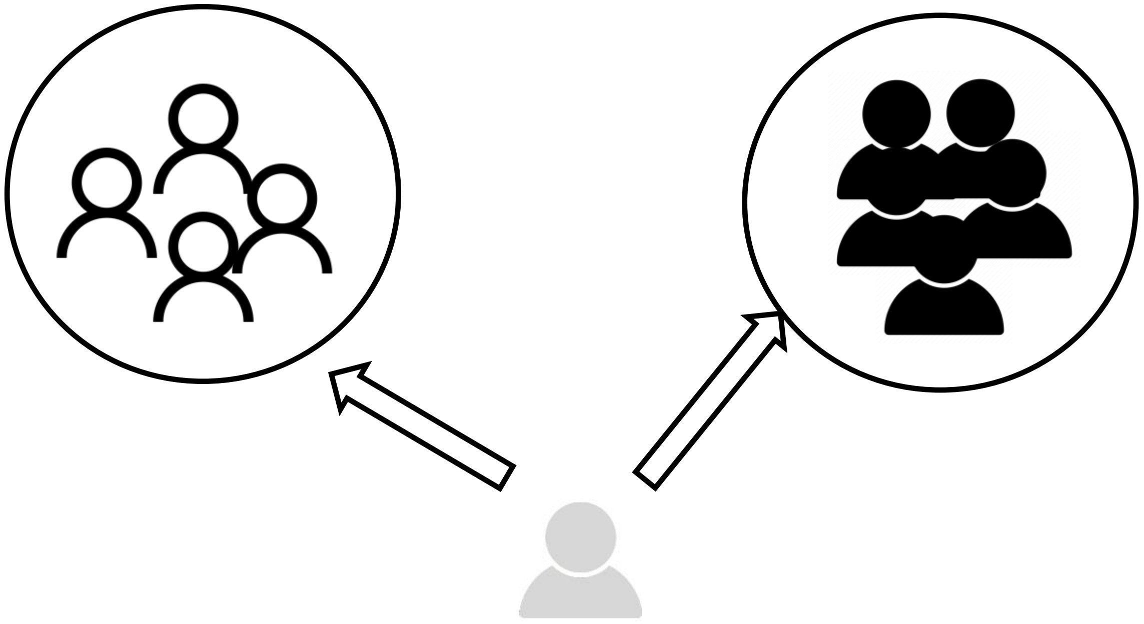
- Remedy method: Elbow method

Elbow Method for selection of optimal "K" clusters



KMeans Variants [Advanced]

- The problem can be more complicated for many real applications.
- For example, the soft clustering calling for variants of KMeans.



Hierarchical Clustering

- KMeans is good if you know how many clusters to expect.
- If you don't know how many clusters there might be, **Hierarchical Clustering** may be more useful.
  - Bottom up clustering starts with individuals and groups the most similar together
  - Top down clustering splits the data into two groups, then four etc

Agglomerative Clustering

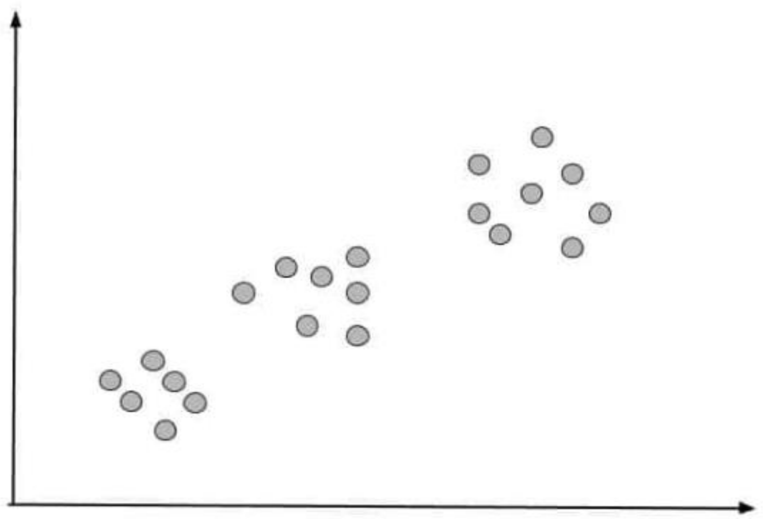
- Bottom up (*agglomerative*) clustering is easiest to understand.
- Use the distance metric to compute an $N \times N$ *proximity matrix* containing the similarity between all pairs of points.
- Select the closest pair of points and merge them into our first cluster of two.

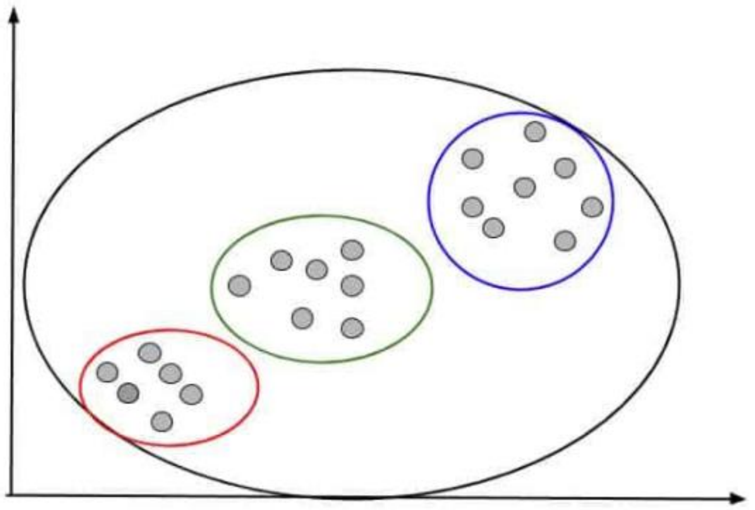
Agglomerative Clustering

- Next we merge the next closest.
  - This could be another two single points or
  - it could be one point and the just created cluster of two
- To measure the distance from the cluster we need a function that can compare clusters
- This continues until there is just one cluster.

Hierarchical Clustering

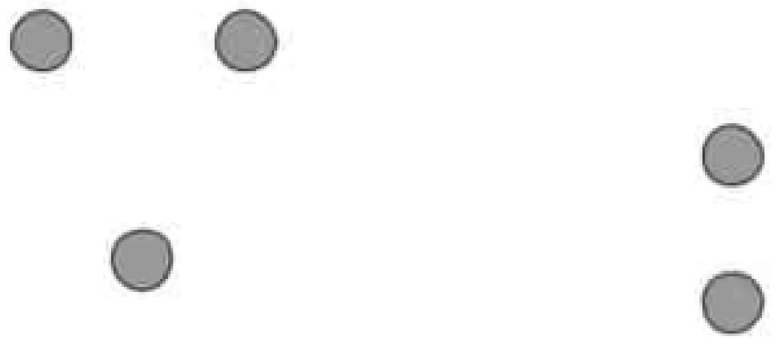
- Suppose we have following data points, and we have to separate them into different clusters.
- Hierarchical clustering cluster the data points based on its similarity. Hierarchical clustering continues clustering until one single cluster left.
- As you can see in this image. Hierarchical clustering combines all three smaller clusters into one final cluster.





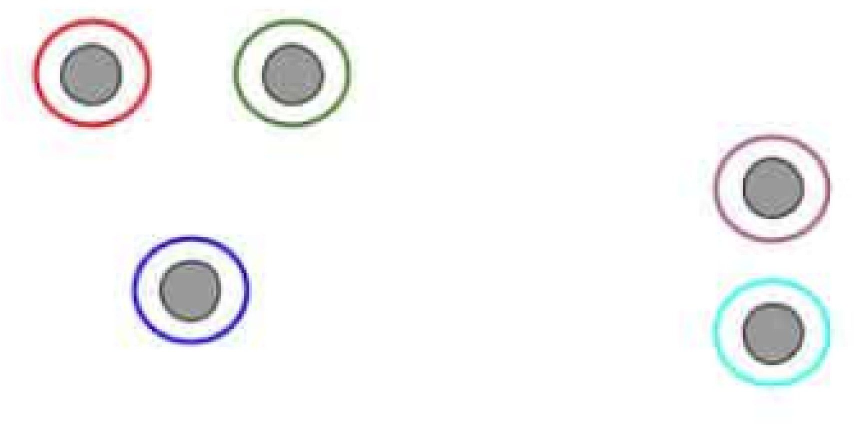
Example

- Agglomerative Hierarchical Clustering uses a **bottom-up approach** to form clusters. That means it starts from single data points. Then it clusters the closer data points into one cluster. The same process repeats until it gets one single cluster.
- Suppose we have following data points.



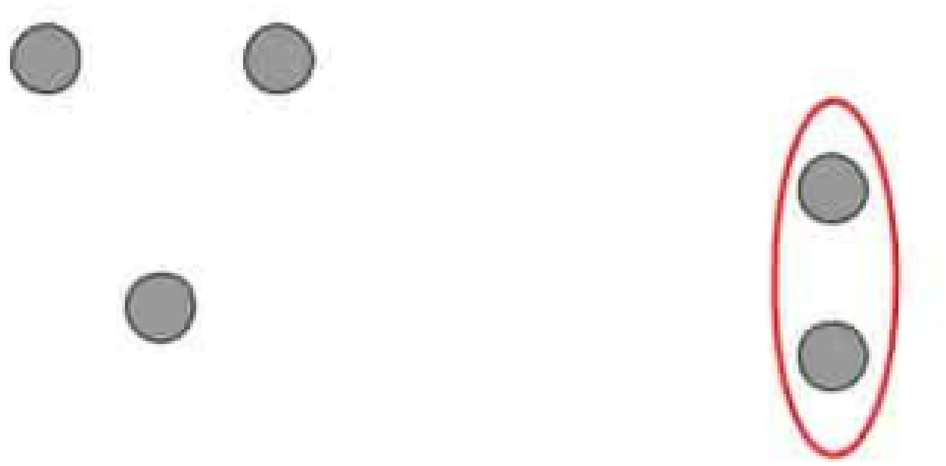
Example (continued)

- Make each data point a single cluster. Suppose that forms 5 clusters.



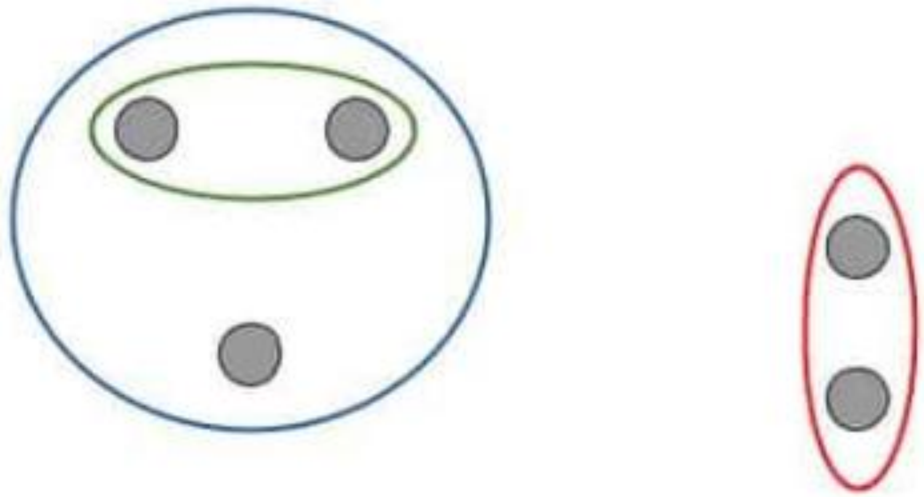
Example (continued)

- Take the 2 closet data points and make them one cluster. Now the total clusters become 4.



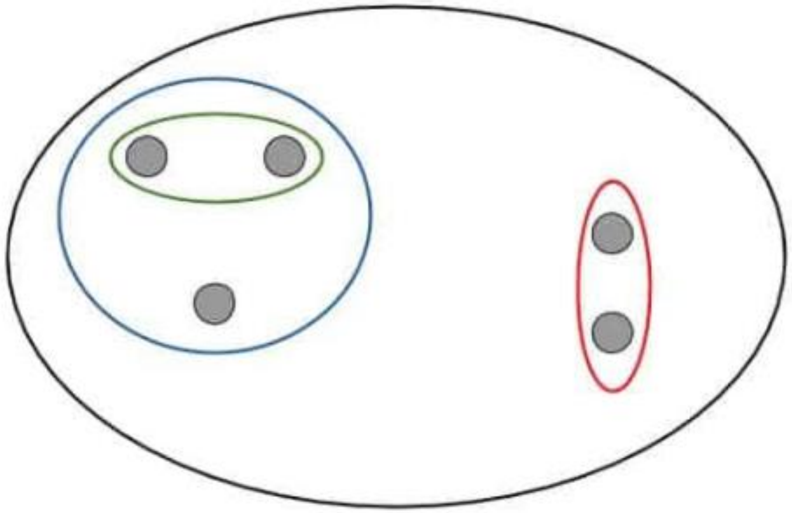
Example (continued)

- Take the 2 closest clusters and make them one cluster. Now the total clusters become 3.



Example (continued)

- Repeat until only one cluster is left. When only one huge cluster is left, the algorithms stops.

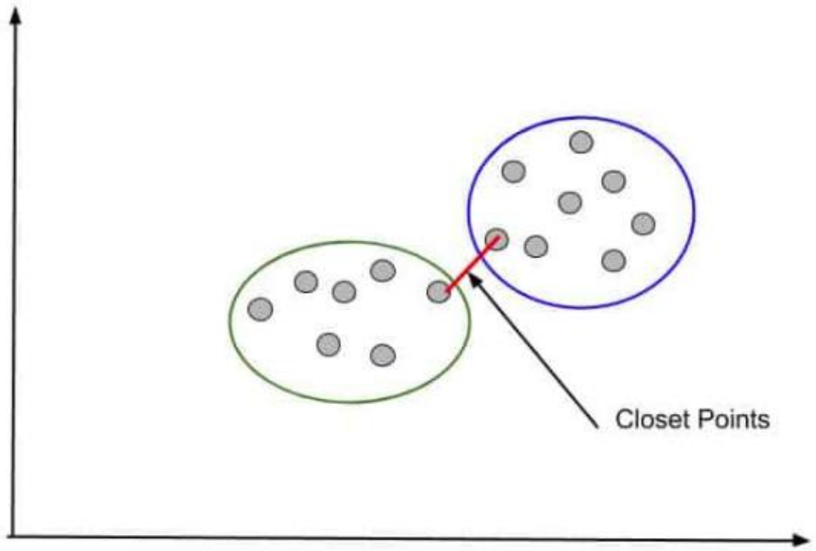


Distance between Two Clusters?

- You may be wondering how to choose the closet clusters?. Because how to find the distance between two clusters is different than finding distance of two data points.
- Right?
- So , let’s see.

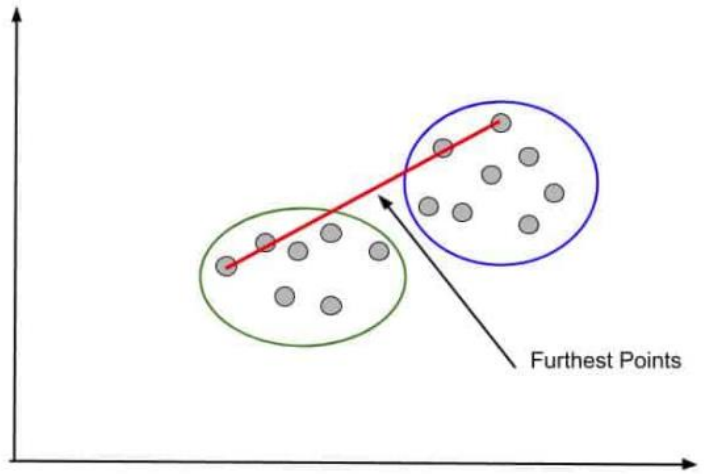
Distance between Two Clusters

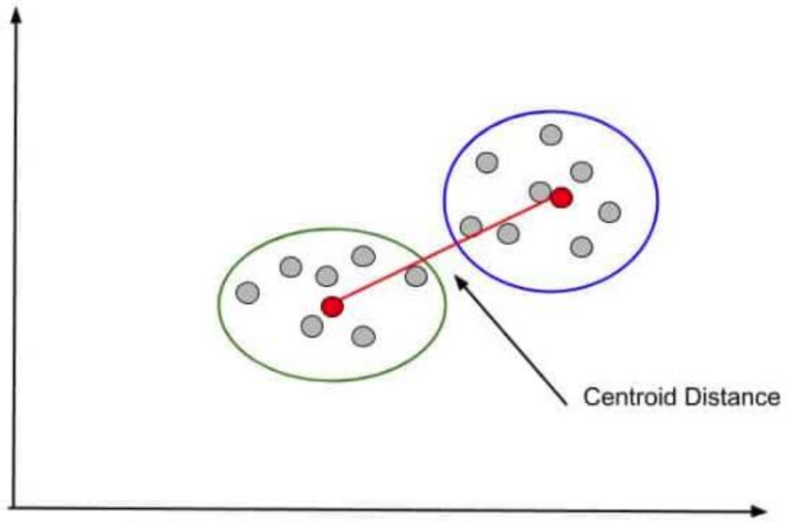
- For calculating the distance between two data points, we use **the Euclidean Distance Formula.**
- But, to calculate the distance between two clusters, we can use two methods.
- **1. Closest Points-** That means we take the distance of two closest points from two clusters. It is also known as **Single Linkage**. Something like that



Distance between Two Clusters

- **Complete-linkage**





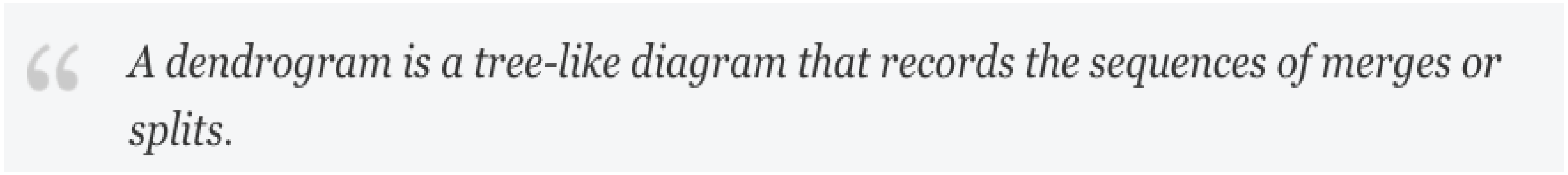
- **Average-linkage**
- **Distance between Centroids**
  - Another option is to find the centroid of clusters and then calculate the distance between two centroids. It is known as **Centroid-linkage**.

Distance between Two Clusters

- Choosing the method for distance calculation is an important part of Hierarchical Clustering. Because it affects performance.
- That’s why you should keep in mind while working on Hierarchical clustering that distances between clusters are crucial.

# of Clusters

- How should we Choose the Number of Clusters in Hierarchical Clustering?
  - Ready to finally answer this question that’s been hanging around since we started learning? To get the number of clusters for hierarchical clustering, we make use of an awesome concept called a **Dendrogram**.

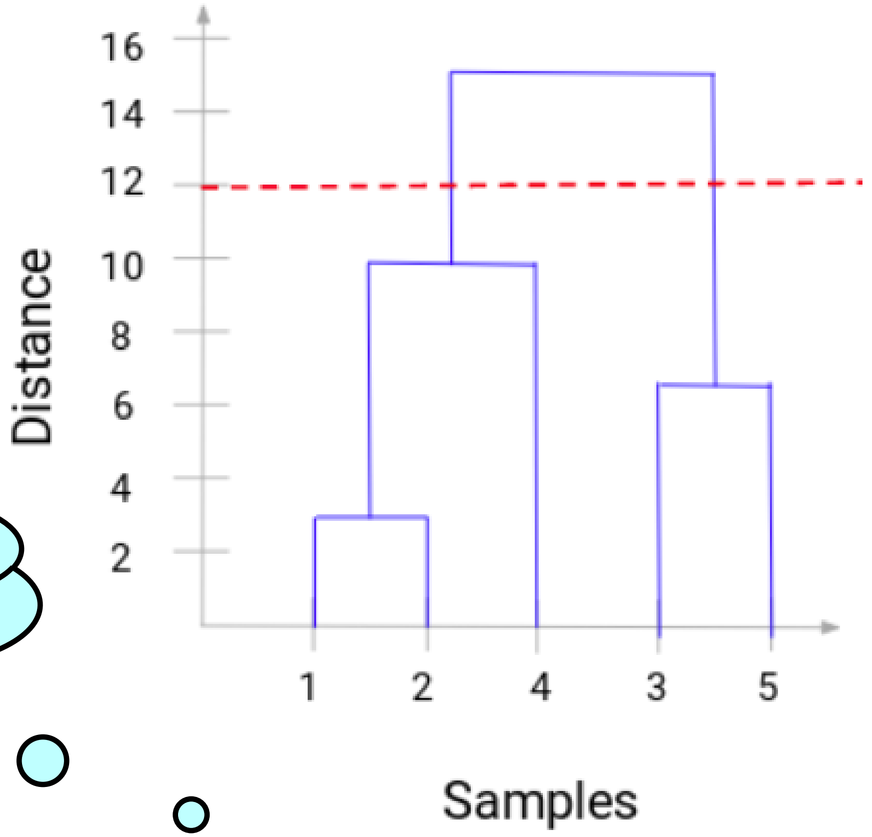

“ *A dendrogram is a tree-like diagram that records the sequences of merges or splits.*

Determining clusters

- Now, we can set a threshold distance and draw a horizontal line. Let's set this threshold as 12 and draw a horizontal line:

➢ The number of clusters will be the number of vertical lines which are being intersected by the line drawn using the threshold.

Pretty straightforward Right?



Summarised Key Points -I

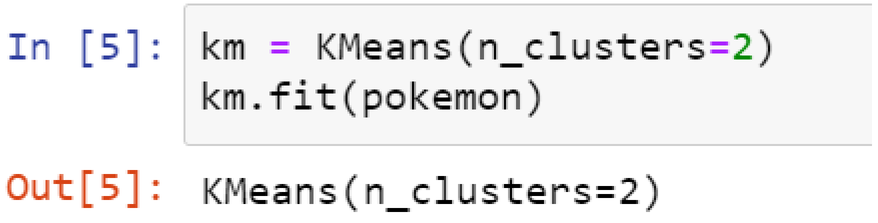
- **Advantages of KMeans**
  - Simple to implement algorithm;
  - Be computationally fast;
  - An instance can change cluster (move to another cluster) when the centroids are recomputed.
- **Advantages of Hierarchical Clustering**
  - Simple to implement algorithm;
  - Don's need to specify the # of clusters required for the algorithm;
  - Outputs a hierarchy, i.e., a structure that is more informative than the unstructured set of flat clusters returned by k-means.

Summarised Key Points -II

- **Disadvantages of KMeans**
  - Difficult to predict the number of clusters (*k*-Value);
  - Initial seeds have a strong impact on the final results.
- **Disadvantages of Hierarchical Clustering**
  - It is not possible to undo the previous step: once the instances have been assigned to a cluster, they can no longer be moved around.
  - Hierarchical clustering requires the computation and storage of an $N \times N$ *proximity matrix*. For very large datasets, this can be expensive and slow.

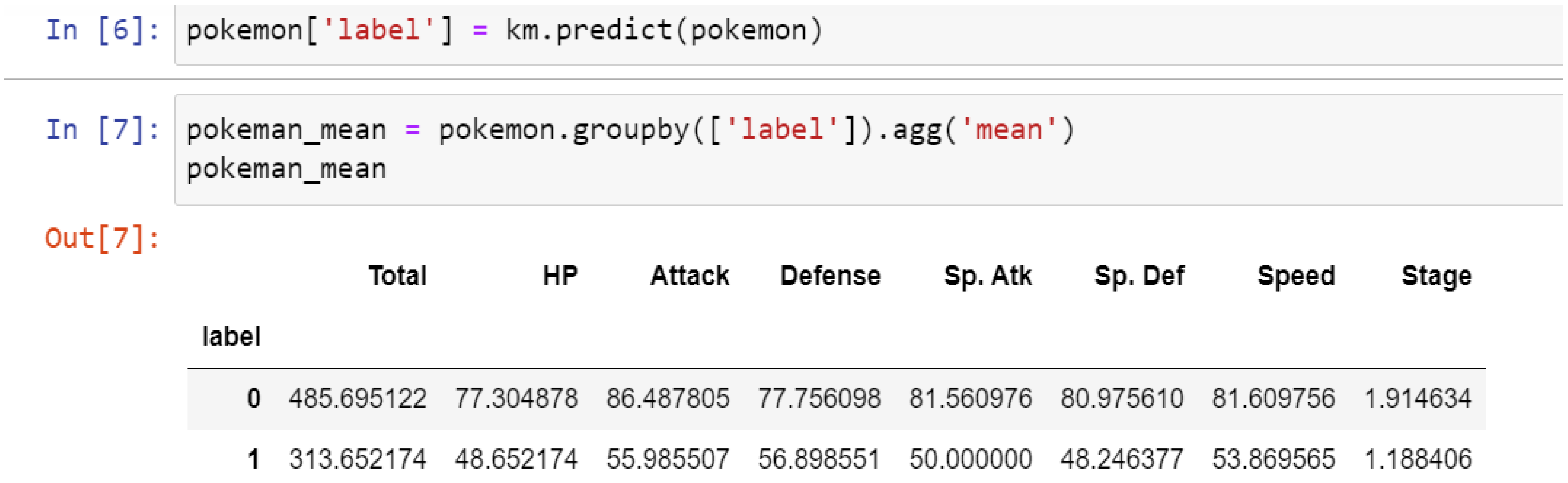
Sample Code for Clustering

- from sklearn.cluster import KMeans



```
In [5]: km = KMeans(n_clusters=2)
        km.fit(pokemon)

Out[5]: KMeans(n_clusters=2)
```



```
In [6]: pokemon['label'] = km.predict(pokemon)
```

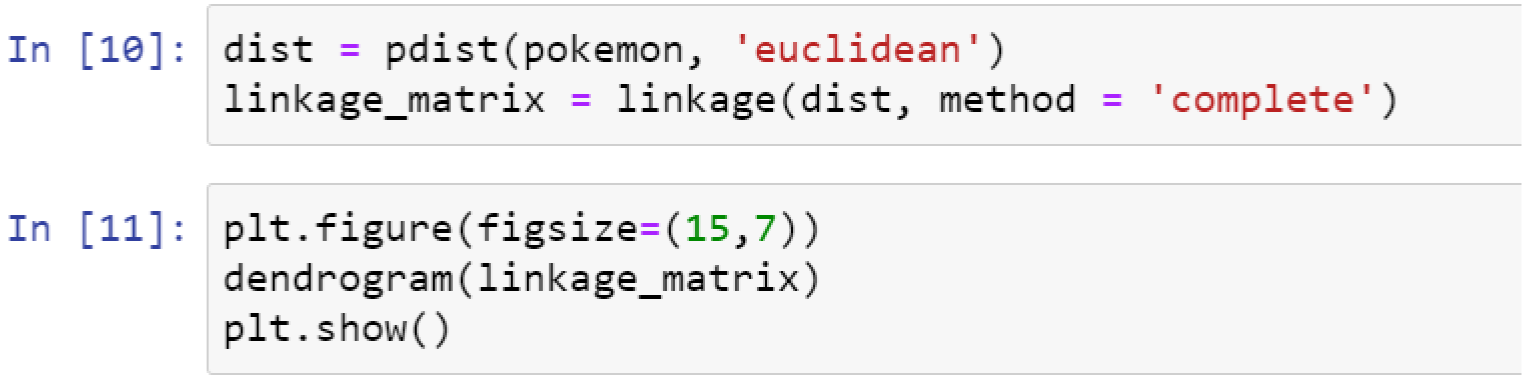
```
In [7]: pokeman_mean = pokemon.groupby(['label']).agg('mean')
        pokeman_mean

Out[7]:
```

	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Stage
label								
0	485.695122	77.304878	86.487805	77.756098	81.560976	80.975610	81.609756	1.914634
1	313.652174	48.652174	55.985507	56.898551	50.000000	48.246377	53.869565	1.188406

Sample Code for Clustering

- from scipy.cluster.hierarchy import linkage, dendrogram, cut_tree



```
In [10]: dist = pdist(pokemon, 'euclidean')
         linkage_matrix = linkage(dist, method = 'complete')
```

```
In [11]: plt.figure(figsize=(15,7))
         dendrogram(linkage_matrix)
         plt.show()
```