**Branch: - Computer Science and Engineering Class: - III Year**

**Subject: - Big Data Analytics Lab Sem: - VI**

**Student Manual**

**PRACTICAL NO. 3**

**Aim:** Write a program to implement K-Means algorithm. Use an appropriate data set for clustering.

**Software Requirement**: Jupyter

**Theory:**

K-Means Clustering is an unsupervised learning algorithm that is used to solve the clustering

problems in machine learning or data science. In this topic, we will learn what is K-means

clustering algorithm, how the algorithm works, along with the Python implementation of kmeans clustering.

**What is K-Means Algorithm?**

K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabelled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.

It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabelled dataset on its own without the need for any training.

It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

The algorithm takes the unlabelled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

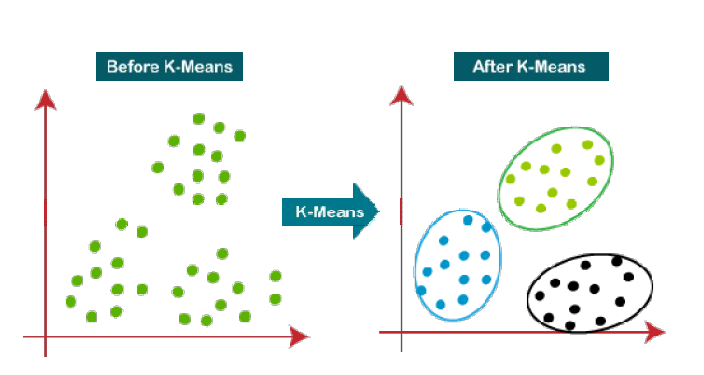
The k-means clustering algorithm mainly performs two tasks:

o Determines the best value for K center points or centroids by an iterative process.

o Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.

Hence each cluster has data points with some commonalities, and it is away from other clusters.

The below diagram explains the working of the K-means Clustering Algorithm:



**How does the K-Means Algorithm Work?**

The working of the K-Means algorithm is explained in the below steps:

Step-1: Select the number K to decide the number of clusters.

Step-2: Select random K points or centroids. (It can be other from the input dataset).

Step-3: Assign each data point to their closest centroid, which will form the predefined K clusters.

Step-4: Calculate the variance and place a new centroid of each cluster.

Step-5: Repeat the third steps, which means reassign each datapoint to the new closest centroid

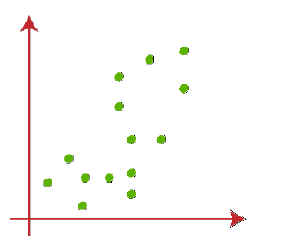
of each cluster.

Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.

Step-7: The model is ready.

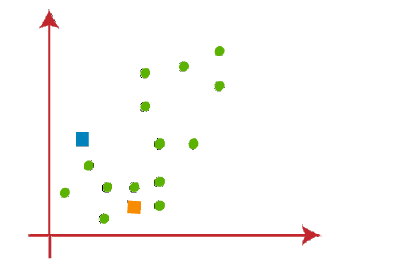
Let's understand the above steps by considering the visual plots:

Suppose we have two variables M1 and M2. The x-y axis scatter plot of these two variables is given below:



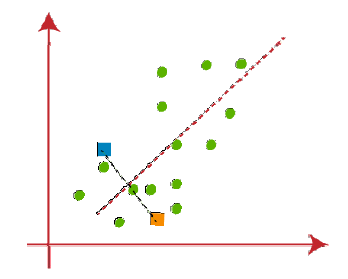
* Let's take number k of clusters, i.e., K=2, to identify the dataset and to put them into different clusters. It means here we will try to group these datasets into two different clusters.
* We need to choose some random k points or centroid to form the cluster. This point can be either the points from the dataset or any other point. So, here we are selecting the below two points as k points, which are not the part of our dataset.

Consider the below image:

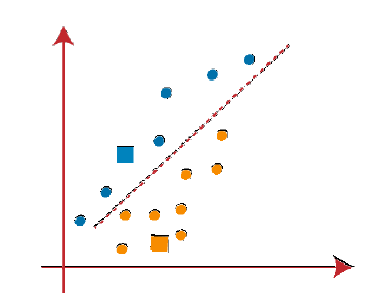


* Now we will assign each data point of the scatter plot to its closest K-point or centroid We will compute it by applying some mathematics that we have studied to calculate the distance between two points. So, we will draw a median between both the centroids.

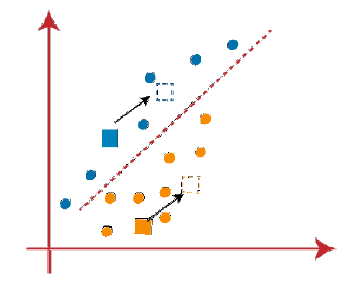
Consider the below image:



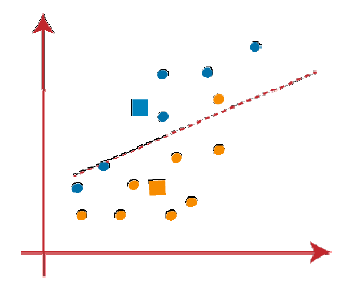
* From the above image, it is clear that points left side of the line is near to the K1 or blue centroid, and points to the right of the line are close to the yellow centroid. Let's color them as blue and yellow for clear visualization.



* As we need to find the closest cluster, so we will repeat the process by choosing a **new centroids.** To choose the new centroids, we will compute the center of gravity of these centroids, and will find new centroids as below:

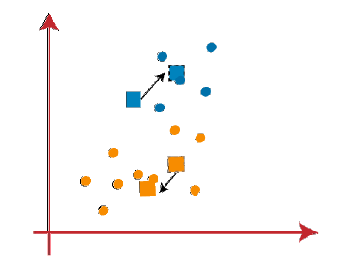


* Next, we will reassign each datapoint to the new centroid. For this, we will repeat the same process of finding a median line. The median will be like below image:

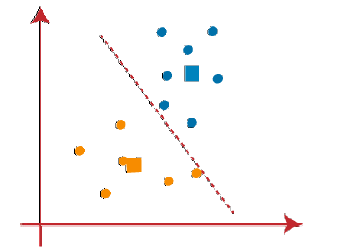


As reassignment has taken place, so we will again go to the step-4, which is finding new centroids or K-points.

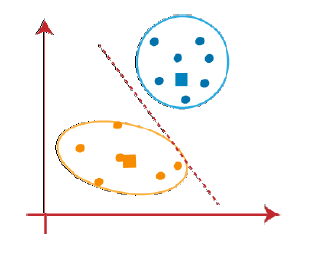
* We will repeat the process by finding the center of gravity of centroids, so the new centroids will be as shown in the below image:



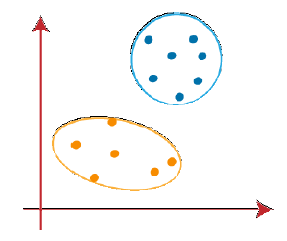
As we got the new centroids so again will draw the median line and reassign the data points. So, the image will be:



We can see in the above image; there are no dissimilar data points on either side of the line, which means our model is formed. Consider the below image:



As our model is ready, so we can now remove the assumed centroids, and the two final clusters will be as shown in the below image:



**Program:**

* Here, we will call some basic and important libraries to work.

|  |
| --- |
| import pandas as pd import numpy as np  import matplotlib.pyplot as plt |

* sklearn is one of the most important packages in machine learning and big data analytics & it provides the maximum number of functions and algorithms. To use **k- means clustering we need to call it from sklearn package**.

|  |
| --- |
| from sklearn.cluster import KMeans |

* load the file into DataFrame object

|  |
| --- |
| df=pd.read\_csv('Filename.csv') |

* Head method shows us only the first 5 Rows

|  |
| --- |
| df.head() |

* To get multiple column data and save it to its own variable, here we use ‘X’ as name of variable

|  |
| --- |
| X = df[['column\_name1','column\_name2']] |

* To define the number of Cluster used following command
* Here the number of clusters will be 3. This number is given arbitrarily by us. We can choose any number to define the number of clusters and **kmean** is your defined model. To **train** our model, we use **fit() function like kmean.fit(argument),** here the **argument** in is our **data set** that need to be Clustered. After using the **fit()** function our model is ready.

|  |
| --- |
| *#Initialize the class object*  kmean=KMeans(n\_clusters=3) kmean.fit(X) |

* we can see our three centers by using the following command

|  |
| --- |
| kmean.cluster\_centers\_ |

* Inertia measures how well a dataset was clustered by K-Means. ***It is calculated by measuring the distance between each data point and its centroid, squaring this distance, and summing these squares across one cluster.*** A good model is one with low inertia

|  |
| --- |
| kmean.inertia\_ |

* In the Elbow method, we are actually varying the number of clusters ( K ) from 1 – 11. For each value of K, we are calculating ***WCSS ( Within-Cluster Sum of Square ).*** ***WCSS is the sum of squared distance between each point and the centroid in a cluster***.
* As the number of clusters increases, the WCSS value will start to decrease. WCSS value is largest when K = 1.
* **We calculated the WCSS value for each K value.**

|  |
| --- |
| WCSS = []  for i in range(1,11):  kmean= KMeans(n\_clusters=i)  kmean.fit(X)  WCSS.append(kmean.inertia\_) |

* To get the WCSS value for each K value use following command

|  |
| --- |
| WCSS |

* When we plot the **dataset** here **‘X’** is our **dataset**.

|  |
| --- |
| plt.plot(X) |

* When we plot the WCSS with the K value, the plot looks like an ***Elbow***.

|  |
| --- |
| plt.plot(range(1,11),WCSS,marker = 'x')  plt.title('The Elbow Method Graph') plt.xlabel('String1') plt.ylabel('String2') plt.show() |

* To draw a scatter plot we used ***scatter()*** function, scatter() function plots one dot for each observation. It needs two arrays of the same length, one for the values of the x-axis, and one for values on the y-axis

|  |
| --- |
| plt.scatter(X['column\_name1'],X['column\_name2']) |

* Now let’s train the model on the dataset with a number of clusters 5.

|  |
| --- |
| kmean= KMeans(n\_clusters=5)  kmean.fit(X) |

* Using K-Means, each point is assigned to a specific cluster. We can use model.predict() function ***to find the cluster number for each observation***. Here model is the name of object

|  |
| --- |
| cluser\_numbers = kmean.predict(X) |

* To print result of cluser\_nambers for all clusters

|  |
| --- |
| cluser\_numbers |

* To print result of cluser\_nambers for 1st clusters

|  |
| --- |
| cluser\_numbers==0 |

* cluster\_number give us different clusters corresponding to X. Now let’s plot all the clusters using matplotlib.

|  |
| --- |
| plt.scatter(X[cluser\_numbers==0]['C1'],X[cluser\_numbers==0]['C2'],c='red')  plt.scatter(X[cluser\_numbers==1]['C1'],X[cluser\_numbers==1]['C2'],c='blue')  plt.scatter(X[cluser\_numbers==2]['C1'],X[cluser\_numbers==2]['C2'],c='green')  plt.scatter(X[cluser\_numbers==3]['C1'],X[cluser\_numbers==3]['C2'],c='yellow')  plt.scatter(X[cluser\_numbers==4]['C1'],X[cluser\_numbers==4]['C2'],c='cyan')  plt.scatter(kmean.cluster\_centers\_[:,0],kmean.cluster\_centers\_[:,0],c='black') |

**Result:**