# Report and Numerical study on K-Means Algorithm

**Aim-** To understand the theoretical and practical concepts of K-Means Algorithms and perfom a use case.

**Abstract- Clustering** is a type of Unsupervised learning. This is very often used when you don’t have labeled data. K-Means Clustering is one of the popular clustering algorithm. The goal of this algorithm is to find groups(clusters) in the given data.
K-Means is a very simple algorithm which clusters the data into K number of clusters.

**Dataset-** We have used a simple unlabelled data that contains **3000 rows and 2 columns**

**Algorithm-** The algorithm is stated below :-

**Step 1**
We randomly pick K cluster centers(centroids). Let’s assume these are c1,c2,…,ck, and we can say that;

C=c1,c2,…,ck
C is the set of all centroids.

**Step 2**
In this step we assign each input value to closest center. This is done by calculating Euclidean(L2) distance between the point and the each centroid.

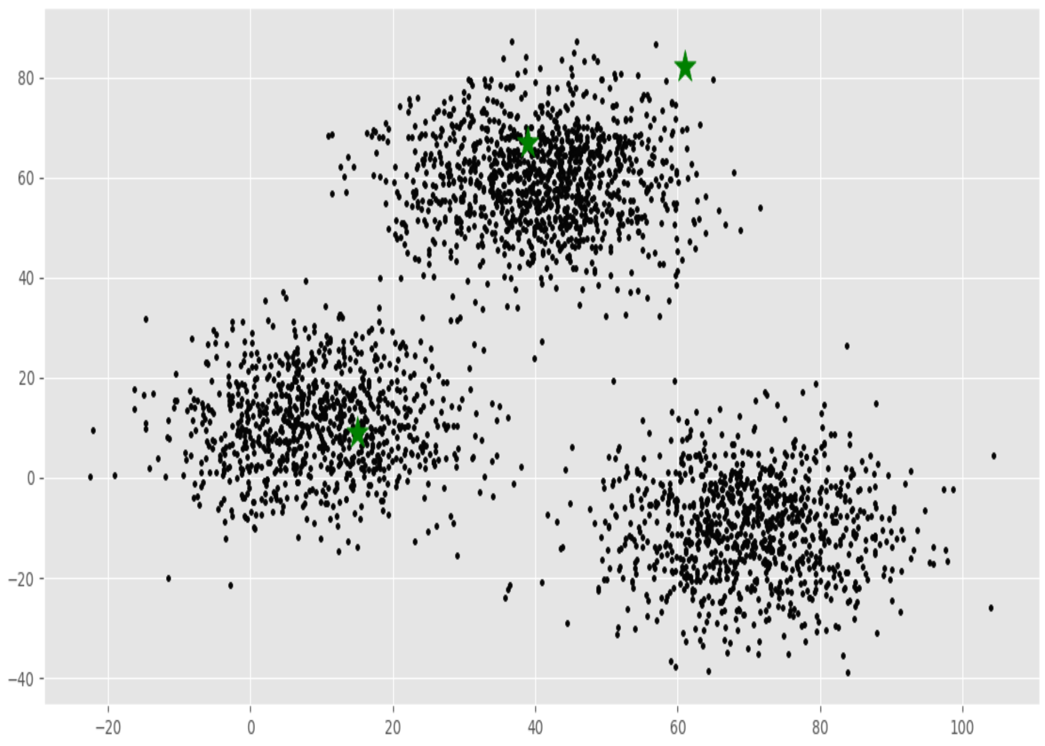
argminci∈Cdist(ci,x)2
Where dist(.) is the Euclidean distance.

**Step 3**
In this step, we find the new centroid by taking the average of all the points assigned to that cluster.

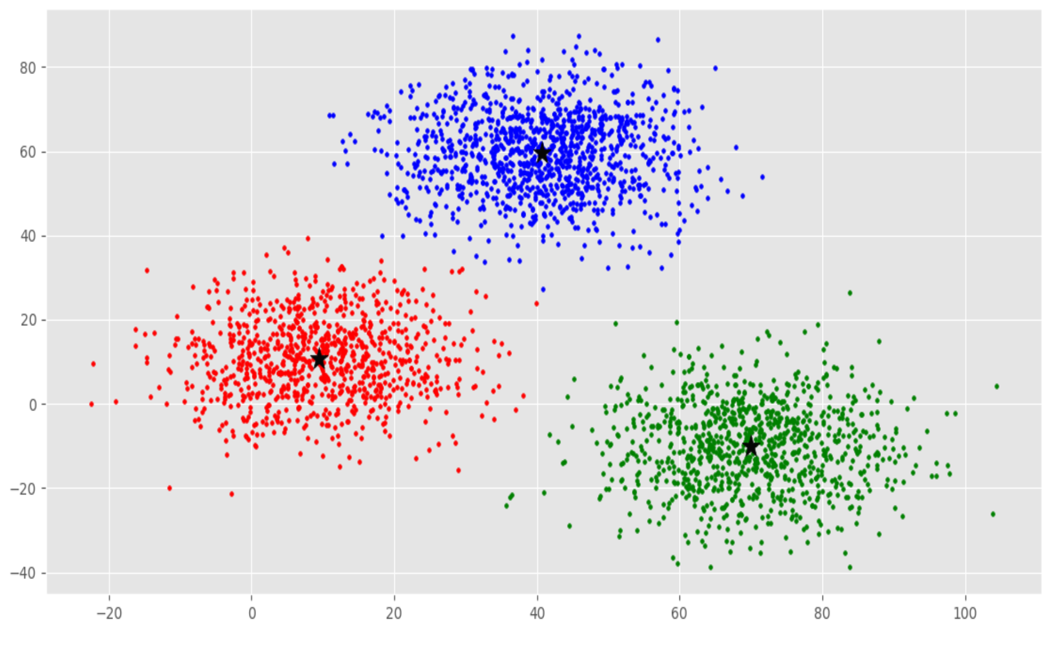
ci=1|Si|∑xi∈Sixi
Si is the set of all points assigned to the ith cluster.

**Step 4**
In this step, we repeat step 2 and 3 until none of the cluster assignments change. That means until our clusters remain stable, we repeat the algorithm.

**Results-** Below are the results of the K-Means ALgorithm

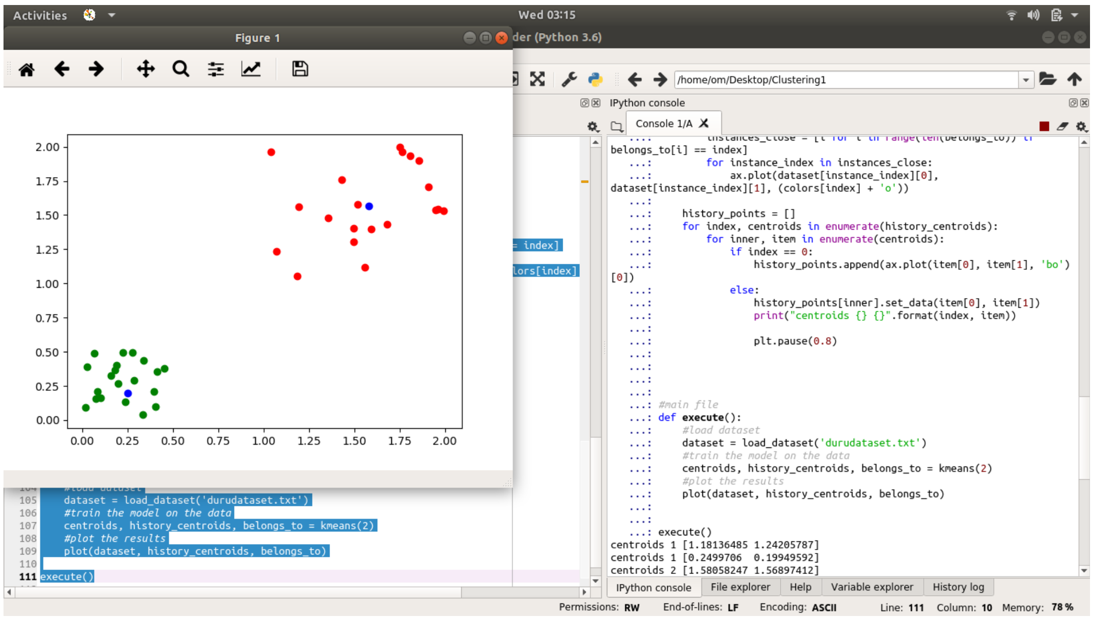


**(fig 1-Shows initial centroid)**



**(fig2 shows updated centroid)**

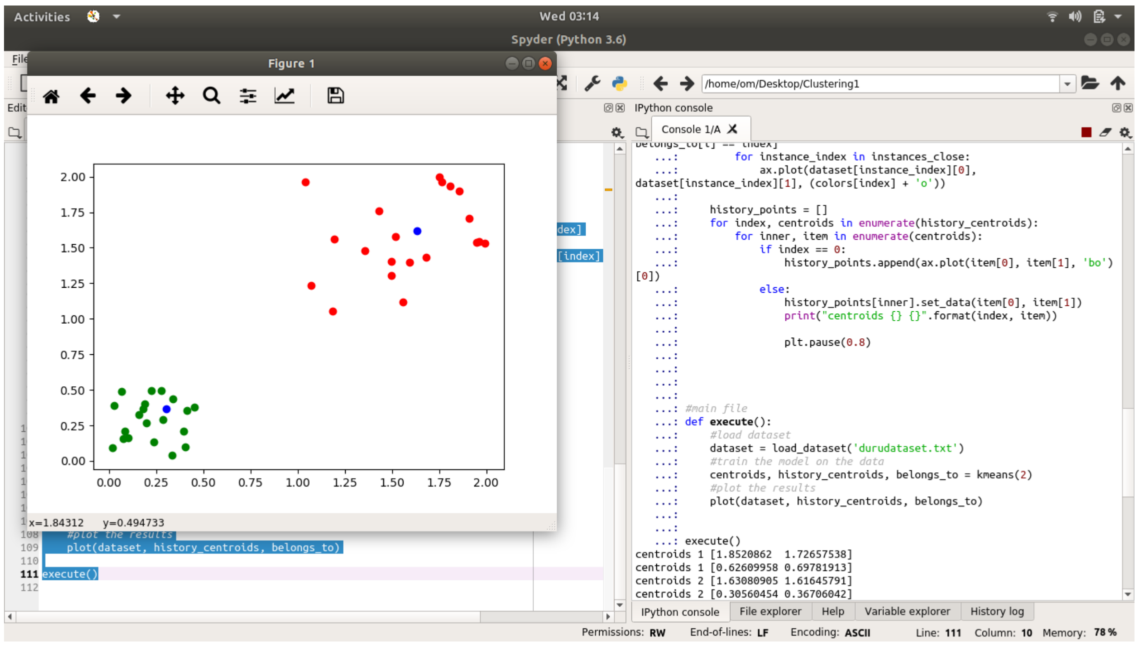
**Example-2** The Second use case.



```
    dataset = load_dataset('durudataset.txt')
    #train the model on the data
    centroids, history_centroids, belongs_to = kmeans(2)
    #plot the results
    plot(dataset, history_centroids, belongs_to)

execute()
```

```
belongs_to[i] == index]
   ...:             for instance_index in instances_close:
   ...:                 ax.plot(dataset[instance_index][0],
dataset[instance_index][1], (colors[index] + 'o'))
   ...:
   ...:         history_points = []
   ...:         for index, centroids in enumerate(history_centroids):
   ...:             for inner, item in enumerate(centroids):
   ...:                 if index == 0:
   ...:                     history_points.append(ax.plot(item[0], item[1], 'bo')
[0])
   ...:                 else:
   ...:                     history_points[inner].set_data(item[0], item[1])
   ...:                     print("centroids {} {}".format(index, item))
   ...:
   ...:                     plt.pause(0.8)
   ...:
   ...:
   ...:
   ...:
   ...: #main file
   ...: def execute():
   ...:     #load dataset
   ...:     dataset = load_dataset('durudataset.txt')
   ...:     #train the model on the data
   ...:     centroids, history_centroids, belongs_to = kmeans(2)
   ...:     #plot the results
   ...:     plot(dataset, history_centroids, belongs_to)
   ...:
   ...:
   ...: execute()
centroids 1 [1.18136485 1.24205787]
centroids 1 [0.2499706  0.19949592]
centroids 2 [1.58058247 1.56897412]
```



```
    #plot the results
    plot(dataset, history_centroids, belongs_to)

execute()
```

```
   ...:             for instance_index in instances_close:
   ...:                 ax.plot(dataset[instance_index][0],
dataset[instance_index][1], (colors[index] + 'o'))
   ...:
   ...:         history_points = []
   ...:         for index, centroids in enumerate(history_centroids):
   ...:             for inner, item in enumerate(centroids):
   ...:                 if index == 0:
   ...:                     history_points.append(ax.plot(item[0], item[1], 'bo')
[0])
   ...:                 else:
   ...:                     history_points[inner].set_data(item[0], item[1])
   ...:                     print("centroids {} {}".format(index, item))
   ...:
   ...:                     plt.pause(0.8)
   ...:
   ...:
   ...:
   ...:
   ...: #main file
   ...: def execute():
   ...:     #load dataset
   ...:     dataset = load_dataset('durudataset.txt')
   ...:     #train the model on the data
   ...:     centroids, history_centroids, belongs_to = kmeans(2)
   ...:     #plot the results
   ...:     plot(dataset, history_centroids, belongs_to)
   ...:
   ...:
   ...: execute()
centroids 1 [1.8520862  1.72657538]
centroids 1 [0.62609958 0.69781913]
centroids 2 [1.63080905 1.61645791]
centroids 2 [0.30560454 0.36706042]
```

**Code** :- Created on Sat May 19 02:17:42 2018

```
@author: om
"""

#importing all the neccessary libraries
from copy import deepcopy
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
plt.rcParams['figure.figsize'] = (16, 9)
plt.style.use('ggplot')

# Importing the dataset
data = pd.read_csv('xclara.csv')
print("Input Data and Shape")
print(data.shape)
data.head()

# Getting the values and plotting it
f1 = data['V1'].values
f2 = data['V2'].values
X = np.array(list(zip(f1, f2)))
plt.scatter(f1, f2, c='black', s=7)

# Euclidean Distance Caculator
#it will give us sum of squarred errors across all the data points
#If axis is an integer, it specifies the axis of x along which to compute the vector norms.
# If axis is a 2-tuple, it specifies the axes that hold 2-D matrices, and the matrix norms of
these matrices are computed. If axis is None then either a vector norm (when x is 1-D) or a
matrix norm (when x is 2-D) is returned.

def dist(a, b, ax=1):
    return np.linalg.norm(a - b, axis=ax)


#Step-1 Randomly picking up the centroids
# Number of clusters
k = 3
# X coordinates of random centroids
C_x = np.random.randint(0, np.max(X)-20, size=k)
# Y coordinates of random centroids
C_y = np.random.randint(0, np.max(X)-20, size=k)
#Desired dtype of the result
C = np.array(list(zip(C_x, C_y)), dtype=np.float32)
print("Initial Centroids")
print(C)
```

```
# Plotting along with the Centroids
plt.scatter(f1, f2, c='#050505', s=7)
plt.scatter(C_x, C_y, marker='*', s=300, c='g')

# To store the value of centroids when it updates
C_old = np.zeros(C.shape)
# Cluster Lables(0, 1, 2)
clusters = np.zeros(len(X))
# Error func. - Distance between new centroids and old centroids
error = dist(C, C_old, None)
# Loop will run till the error becomes zero
while error != 0:

    #Step-2 Assign each data point with the closest centroid
    # Assigning each value to its closest cluster
    for i in range(len(X)):
        distances = dist(X[i], C)
        cluster = np.argmin(distances) #returns minimum values along axis
        clusters[i] = cluster

    # Storing the old centroid values
    #deep copy concept
    C_old = deepcopy(C)

    #Step-3 Finding new centroids by taking average
    # Finding the new centroids by taking the average value
    for i in range(k):
        points = [X[j] for j in range(len(X)) if clusters[j] == i]
        C[i] = np.mean(points, axis=0)
    error = dist(C, C_old, None)

colors = ['r', 'g', 'b', 'y', 'c', 'm']
fig, ax = plt.subplots()
for i in range(k):
        points = np.array([X[j] for j in range(len(X)) if clusters[j] == i])
        ax.scatter(points[:, 0], points[:, 1], s=7, c=colors[i])
ax.scatter(C[:, 0], C[:, 1], marker='*', s=200, c='#050505')
```