Implementing K-Means Clustering

Anika Tanzim
Dept. Computer Science and Engineering
Ahsanullah University of Science and Technology
Dhaka, Bangladesh
160204072@aust.edu

***Abstract*—The objective of this experiment is to implement K-Means Clustering algorithm. K-means algorithm is an iterative algorithm that tries to partition the dataset into K pre-defined distinct non-overlapping clusters where each data point belongs to only one group. It 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 k-means clustering.**

***Index Terms*—K-Means Clustering, Euclidean Distance, Unsupervised Machine Learning**

I. Introduction

K-Means Clustering is an iterative unsupervised learning algorithm, which groups the unlabeled dataset into different clusters. K defines the number of pre-defined clusters that need to be created. For example, if K=2, there will be two clusters, and for K=3, there will be three clusters. It is a centroid-based algorithm, where each cluster is connected with a centroid or center. The main purpose of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.This algorithm typically processes two tasks. At first,it determines the best value for K center points or centroids by an iterative process.Then it assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.

To calculate the distance here we are going to use Euclidean Distance. The Euclidean distance between two points in Euclidean space is the length of a line segment between the two points.

Euclidean Distance:

Let point X have Cartesian coordinates (x1,x2) and let point Y have coordinates (y1,y2). Then the distance between X and Y is given by:

$$d(X,Y) = \sqrt{(y1-x1)^2 \ + \ (y2-x2)^2}$$

K-means runs this formula to compute the distance between the data points and the k number of means. The minimum distance indicates that the data should be in the cluster from which the data is closer. The process will get terminated when all the data will no more changes their clusters in the next iteration.

II. Experimental Design / Methodology

A. *Plotting the sample points:*

A train dataset is given. Each line in the dataset represents an item, and it contains numerical values (one for each feature). No points are classified. Using Matplotlib, all the points from dataset are plotted using green star marker in figure 1.

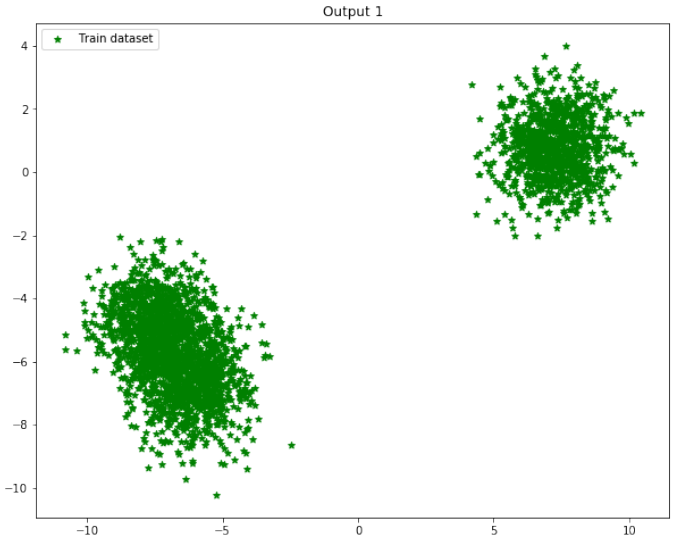

Fig. 1. Task 1 Output: Sample points plotted

B. *Implementing K-means algorithm and Classifying data points:*

Now, we perform the k-means clustering algorithm applying Euclidean distance as a distance measure on the given dataset by taking k as user input.

1) **Determining the initial centroid coordinates:** In the beginning after taking K as user input and we assume the centroid or center of these clusters. We can take any random objects as the initial centroids. So, we choose k inital data points randoomly as centroids from the given dataset.
2) **Claculating the distance:** We will be using the euclidean distance as a metric of similarity for our data set.

$$d(X,Y) = \sqrt{(y1-x1)^2 + (y2-x2)^2}$$

K-means runs this formula to compute the distance between the data points and the k number of means. The data will go to that cluster which centroid is closer to the data.

3) **Updating Centroids:** After assigning the data to the group, we have to count that how many sample are in each group/cluster. Then we will calculate the new centroids for the next iteration. We need to find the average value for its feature, for all the items in the mean/cluster. We can do this by adding all the values and then dividing by the number of items.

$$mean = \frac{\Sigma x}{n}$$

Here, n is the number of data in a cluster and x is the data points in a cluster.

4) **Classifying items:** After updating the centroids, again we have to calculate the distance and go through the whole algorithm. The iteration we terminates when the the predicted cluster for each data points does not change in the next iteration. The k-centroids will not change much after some iterations.The clustering may vary as the initial centroid are chosen randomly.

C. *Plotting the corresponding points on the clusters:*

After clustering successfully, now we are plotting the corresponding points on the clusters with different colors and markers. for this, we have initialized the different markers and different colors for k number of clusters.

For example, if k=2, the clustering for the given datset will be,

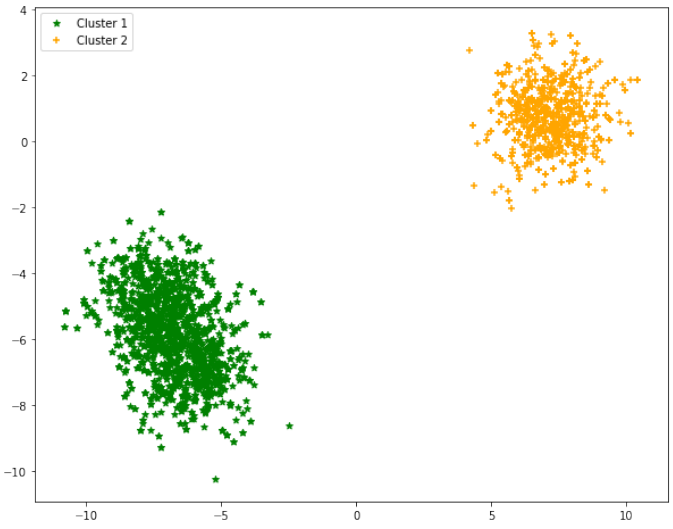

Fig. 2. Task 2 Output: Sample points Classified for k=2

Again, if k=3, the clustering for the given datset will be,

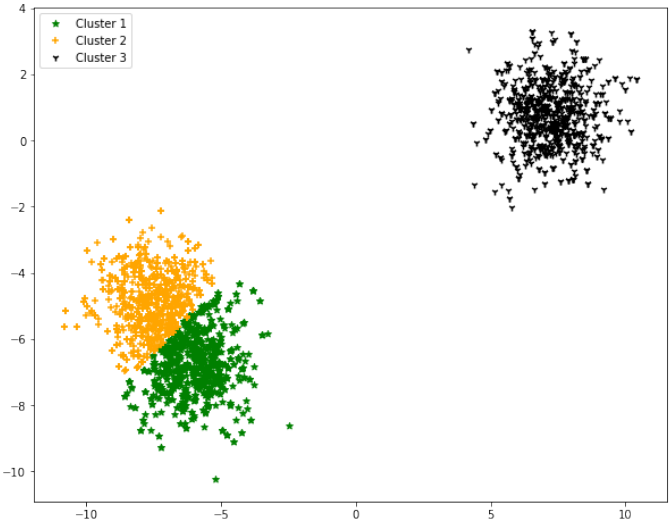

Fig. 3. Task 2 Output: Sample points Classified for k=3

III. Result Analysis

The above graphs show the scatter plot of the data colored by the cluster they belong to. We can see, it is good in capturing structure of the data if clusters have a spherical-like shape. It tries to construct a nice spherical shape around the centroid. kmeans does a poor job in clustering the data for complicated geometric data.

IV. Conclusion

Kmeans clustering is one of the most popular clustering algorithms. It can be applied when solving clustering tasks to get an idea of the structure of the dataset. The goal of kmeans is to group data points into distinct non-overlapping clusters. It gives a good result when the clusters have a kind of spherical shapes.

V. Algorithm Implementation / Code

```
#!/usr/bin/env python
# coding: utf-8

# In[97]:


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import random
random.seed(1)

dataset = np.loadtxt('data_k_mean.txt')

x=dataset[:,0]
y=dataset[:,1]

fig, ax = plt.subplots()
fig.set_figheight(8)
fig.set_figwidth(10)
plt.title("Output 1")
ax.scatter(x,y,marker='*',color='g',label='Train
    dataset')
ax.legend()
```

```
random.shuffle(dataset)

print(dataset)


# In[98]:


def distance(a,b):
    x= np.sqrt(((a[1]-b[1])**2)+((a[0]-b[0])**2))
    return x

def initMean():
    number_of_rows = dataset.shape[0]
    random_indices = np.random.choice(number_of_rows
    , size=1, replace=False)
    random_rows = dataset[random_indices, :]
    x,y =random_rows[0][0],random_rows[0][1]
    return x,y


def mean(cluster):
    sumX=0
    sumY=0
    for i in cluster:
        sumX = sumX + dataset[i][0]
        sumY = sumY + dataset[i][1]
    n=len(cluster)
    if n<1:
        n=1
    meanX=sumX/n
    meanY=sumY/n
    return meanX,meanY


# In[99]:


def cluster(means,k):
    newCluster=[]
    for i in range(k):
        x=[]
        newCluster.append(x)

    for point in range(len(dataset)):
        minDis=100000000000
        newc=-1
        for j in range(k):
            dis=distance(means[j],dataset[point])
            if(dis<minDis):
                minDis = dis
                newc = j
        newCluster[newc].append(point)
    return newCluster


# In[100]:


def update(oldCluster,newCluster,k):
    for i in range(k):
        if(len(oldCluster[i])!=len(newCluster[i])):
            return 0
    for i in range(k):
        for j in range(len(oldCluster[i])):
            if(oldCluster[i][j]!=newCluster[i][j]):
                return 0
    return 1


# In[101]:
```

```
def k_means(k):
    ar=[]

    for i in range(k):
        ar.append([i])

    flag =0
    while(True):
        means=[]

        for c in range(k):
            if flag==0:

                means.append(initMean())
                print(mean(ar[c]))
            else:
                means.append(mean(ar[c]))
        flag=1
        curr=cluster(means,k)
        u=update(ar,curr,k)
        if(u==1):
            break

        ar=curr.copy()

    return curr


# In[102]:


K=int(input('Enter a value for K:'))
output=k_means(K)


xPoint=[]
yPoint=[]
for i in range(K):
    x=[]
    y=[]
    xPoint.append(x)
    yPoint.append(y)
for i in range(K):
    for x in output[i]:
        xPoint[i].append(dataset[x][0])
        yPoint[i].append(dataset[x][1])

m=['*','+','1','s','v','p','+','D','X','P','2','H','
    3','d','4']
c=['green','orange','k','red','green','c','m','y','
    blue']

fig,ax=plt.subplots()
for i in range(K):
    ax.scatter(xPoint[i],yPoint[i],marker=m[i],color
    =c[i],label="Cluster "+str(i+1))
fig.set_figheight(8)
fig.set_figwidth(10)
ax.legend()
plt.show()
```