Implementing K-Means Clustering

Marufa Kamal
dept. Computer Science and Engineering
Ahsanullah University of Science and Technology
Dhaka, Bangladesh
160204073@aust.edu

***Abstract*—K-means is one of the most popular clustering problems which is quite simple and efficient. In this experiment we aim to implement the k-means clustering algorithm on a given training dataset which will determine the clusters based on the k value. This is an unsupervised algorithm which creates clusters with the values having similarity in them.**

***Index Terms*—K-means, Unsupervised Algorithm, Clusters, Euclidean distance, Machine Learning .**

I. Introduction

K-means clustering algorithm is an unsupervised machine learning algorithm used to form clusters from given data points based on the similarity between them. Cluster segmentation can be helpful in images, documents, etc, depending on the individual features of the data. The main idea is to find out the centroids between the clusters formed. But before that, the clusters are formed based on the similarity in between the data i.e., each data is categorized to its closest mean/centroid, and mean coordinates are updated, which are the averages of the data categorized in that mean so far. Data within the same cluster have minimum distance in between them whereas different clusters should have the maximum amount of distance. We assign a point to a cluster based on the nearest centroid value. The number of clusters formed depends on the value of K. We keep updating the centroids of the clusters until there is no change left to be done on the data points. That is when the algorithm stops. The following figure shows the process of this algorithm.

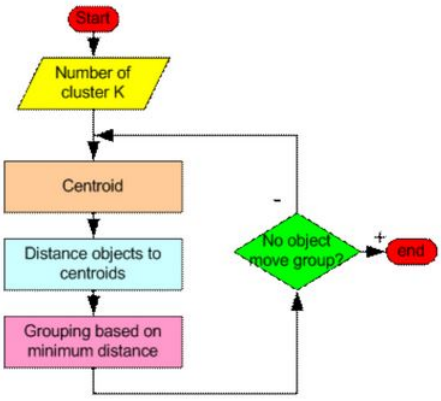

Fig. 1. k-Means Clustering flowchart

II. Experimental Design / Methodology

Let us now look at the different steps taken to solve conduct this experiment.

A. Plotting The Given Sample Points

There are 3000 points given in the text file which I am going to use to form the cluster and implement the algorithm. Using the python package '**matplotlib**' I have plotted all the points from the text file which contained x and y values for a point. The following image shows all the points are plotted in the graph and marked with a red color.

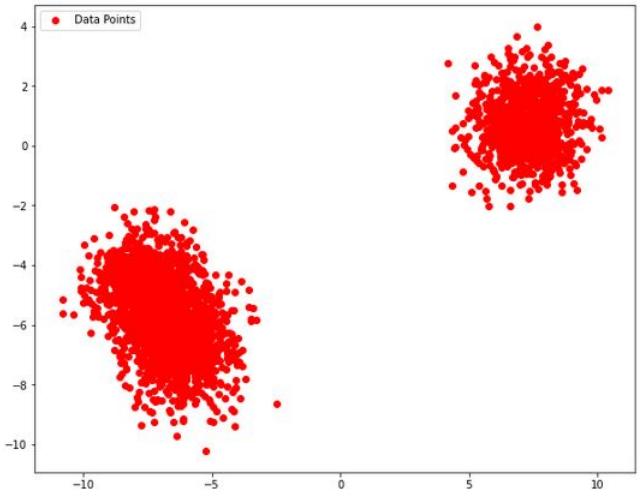

Fig. 2. Plotting the given sample points

B. Performing K-means Clustering algorithm

At first, we take the initial K value which helps us to determine the number of clusters. If k=2 there will be 2 clusters formed with the data. We pass this value and take 2 random mean values initially from the data point. Otherwise, the mean is calculated which the average of the total number of sample points in each cluster. The Euclidean distance is calculated between the mean values and every data point in the dataset. There are many other formulas to calculate distance such as Minkowski distance, Manhattan distance, etc. But here we chose Euclidean distance:-

$$\text{Euclidean distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Next, based on this distance value each point will be assigned to the cluster with the minimum distance. These steps will continue until the clusters stop updating the values i.e., there is no change.

C. Plotting The Clusters

For each data point we store the clusters in a 2D array and plot these points using '**matplotlib**' using red and blue color. If the number of k increased there would be more clusters depending on the value of k. And these clusters would be marked with other colors.

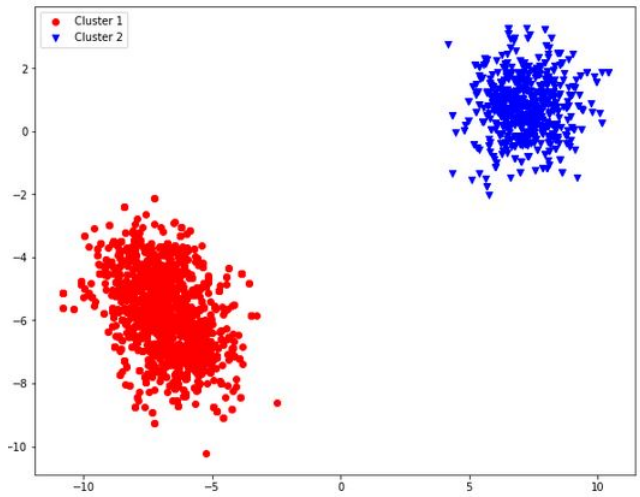

Fig. 3. Plotting The Clusters

III. Result Analysis

Since we have taken the value of k=2, 2 clusters have been formed marked with red and blue color. As we can see the data points in each cluster are very close to each other with the lowest distance possible. On the other hand the centroid between the clusters are kept as far as they can be kept. Describe your interpretation of the different results found in the experiment.

IV. Conclusion

In this experiment I implemented k-Means clustering algorithm and saw how it can be helpful when initializing the cluster centroids. It depends on the data to determine the optimum number of clusters. Multiple experiments and proper data analysis should be done in order to understand the optimum number of clusters.

V. Algorithm Implementation / Code

```
# -*- coding: utf-8 -*-
"""PR_assm5.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1
    RSontjvVSloUjwe1E-6GC2rbzraUzRRX
"""
```

```
#problem 1: plot all the points
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

train_dataset = np.loadtxt('/content/drive/MyDrive
    /4.2/pattern /lab5/data_k_mean.txt')
print(len(train_dataset))

x,y=train_dataset[:,0],train_dataset[:,1] # all
    values o column 0
#y=train_dataset[:,1]# all values o column 1

fig, ax = plt.subplots()
fig.set_figheight(8)
fig.set_figwidth(10)
ax.scatter(x,y,marker='o',color='r',label='Data
    Points')
legend = ax.legend(loc='upper left', shadow=False,
    labelspacing=0.5)
legend.get_frame().set_facecolor('None')
plt.show()
plt.savefig('TrainClass.png')

from math import sqrt
def euclidean_distance(x,y):
  euc_dis=sqrt(pow((x[0]-y[0]),2)+pow((x[1]-y[1]),2)
    )
  return euc_dis

import random
random.seed(1)

random.shuffle(train_dataset)
print(train_dataset)

print(train_dataset[0][0],train_dataset[0][1])

def calculate_centroid(cluster):
    sum_x=0
    sum_y=0
    for i in cluster:
        sum_x+=train_dataset[i][0]
        sum_y+=train_dataset[i][1]
    n=max(1,len(cluster))
    mean_x=(sum_x)/n
    mean_y=(sum_y)/n
    return mean_x,mean_y

def update_cluster(means,k):
    update=[]
    for i in range(k):
        x=[]
        update.append(x)
    for i in range(len(train_dataset)):
        compare=9999  #distance big value
        clus=-1#new cluster
        for j in range(k):
            d=euclidean_distance(means[j],
    train_dataset[i])
            if(d<compare):
              clus=j
              compare=d
        update[clus].append(i)
    return update

def check(a,b,k):
    for i in range(k):
        if(len(a[i])!=len(b[i])):
            return 0
        a[i].sort()
        b[i].sort()
        for j in range(len(a[i])):
```

```
            if(a[i][j]!=b[i][j]):
                return 0
    return 1
def k_means_clustering(k):
    val=[]
    for i in range(k):
        x=[]
        val.append(x)
    for i in range(k):
        val[i].append(i)
    while(1):
        centroid=[]
        for t in range(k):
            centroid.append(calculate_centroid(val[t
    ]))
        current_cluster=update_cluster(centroid,k)
        if(check(val,current_cluster,k)):
            break
        val=current_cluster.copy()
    return current_cluster

k=int(input('Enter a value for k:'))
final_cluster=k_means_clustering(k)
print("The final cluster")
for i in range(k):
    print(final_cluster[i])
x_train=[]
y_train=[]
for i in range(k):#size of the array
    x,y=[],[]
    x_train.append(x)
    y_train.append(y)
for i in range(k):
    for j in final_cluster[i]:
      x_train[i].append(train_dataset[j][0])
      y_train[i].append(train_dataset[j][1])

fig,ax=plt.subplots()
colors=['red','blue','green','yellow','orange','c','
    m','y','k']
marker=['o','v','1','^','s','p','+','D','X','P','2',
    'H','3','d','4']
for i in range(k):
    name="Cluster "+str(i+1)
    ax.scatter(x_train[i],y_train[i],marker=marker[i
    ],color=colors[i],label=name)
fig.set_figheight(8)
fig.set_figwidth(10)
legend.get_frame().set_facecolor('None')
ax.legend()
plt.show()
plt.savefig('4.png')
```