Implementing K-Means Clustering

Mayisha Farzana
dept.Computer Science and Engineering
Ahsanullah University of Science and Technology
Dhaka, Bangladesh
160204028@aust.edu

***Abstract*—This experiment aims to learn about the K-Means clustering algorithm, which is one of the most commonly used clustering algorithms. This is a method of learning that is not supervised. This suggests that the class label is undefined. However, in order to determine the case,we need to know the ground truth in order to test the algorithm's efficiency.**

***Index Terms*—Unsupervised learning; clustering;K-means clustering; python code**

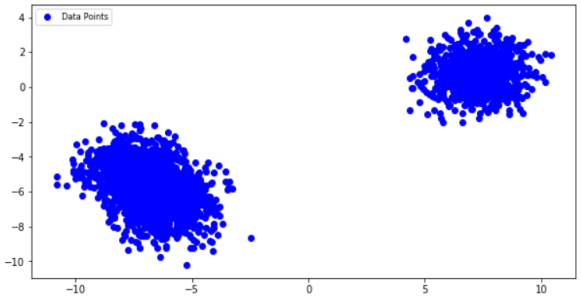
I. Introduction

K-means is one of the most basic unsupervised learning algorithms for solving the well-known problem of clustering. The procedure employs a basic and straightforward method for classifying a given data set using a predetermined number of clusters (assume k clusters). In the beginning, we determine the number of cluster K, and we assume the centroid or center of these clusters. We can take any random objects as the initial centroids, or the first K objects in sequence can also serve as the initial centroids. Since different locations produce different results, these centroids should be positioned with caution. As a result, the best option is to put them as far apart as possible. The next step is to connect each point in a data set with the centroid that is closest to it. If there are no pending points, the first stage is over, and an early grouping is completed. At this stage, we must re-calculate k new centroids to serve as barycenters for the clusters formed in the previous phase. Once we have created these k new centroids, we will need to rebind the same data set points to the nearest new centroid. A loop has been generated. As a result of this loop, we may notice that the k centroids change their location step by step until no more changes are done. In other words, centroids do not move anymore.

II. Experimental Design / Methodology

A.Choosing the initial centroids: Since in this experiment we choose to cluster the whole dataset in three cluster we have to choose three centroids. At first we choose it randomly but the centroid should be as far as possible so we fixed the range.

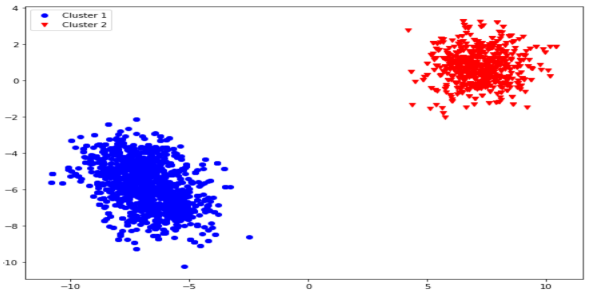
B. Calculating the distance matrix: Here in this experiment we calculate the Euclidian distance between the sample points and centroid. After that this distance matrix will be used to assign groups to each column of the distance matrix. In which row it will be determined by the minimum distance stored in the distance matrix. After assigning the group now we have to count that how many sample are in each group/cluster.



Then we will calculate the new centroids for the next iteration. For distance calculation, we will use euclidean distance.

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

C. Calculating new centroid: The new centroid will be all the data samples average which corresponding row has a 1 in the group matrix. Thus we will get new set of centroid. After that the whole procedure will continue until each group has the same number of samples.In our experiment, the centroid value will not change much after some iteration. It will change sometimes cause the initial centroid are chosen randomly. So it is possible to get slight different output but the difference won't be very high.



III. Result Analysis

From the test points, we will get different classes for each value. We will see that we found three sets of red, blue coded centroids and tuned them to perfection until we found the same set of matrices to represent the clusters.

IV. Conclusion

The K-means clustering algorithm is observed in this experiment. This is a commonly used and popular clustering strategy. If we want to calculate the cluster purity output in this algorithm, we need to know the ground truth.

V. Algorithm Implementation / Code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import random
random.seed(1)
train_dataset = np.loadtxt('data_k_mean.txt')
print(train_dataset)
length=len(train_dataset)
print(length)
f, ax = plt.subplots()
f.set_figheight(5)
f.set_figwidth(10)
ax.scatter(train_dataset[:,0],train_dataset[:,1],
    marker='o',color='b',label='Data Points')
legend = ax.legend(loc='upper left', shadow=False,
    fontsize='small',labelspacing=0.5)
legend.get_frame().set_facecolor('None')
plt.show()
random.shuffle(train_dataset)
print(train_dataset)
def dis(a,b):
    return np.sqrt(((a[0]-b[0])**2)+((a[1]-b[1])**2)
    )
def calculate_mean(cluster):
    x_sum=0
    y_sum=0
    for p in cluster:
        x_sum+=train_dataset[p][0]
        y_sum+=train_dataset[p][1]

    n=max(1,len(cluster))
    return (x_sum)/n,(y_sum)/n
#####
def get_new_cluster(means,k):
    new=[]
    for i in range(k):
        x=[]
        new.append(x)

    for tp in range(len(train_dataset)):
        mn=1000000000000#distance big value
        nc=-1#new cluster
        for i in range(k):
            d=dis(means[i],train_dataset[tp])
            if(d<mn):
                mn=d#new distance
                nc=i
        new[nc].append(tp)
    return new
def check(c_old,c_new,k):
    for i in range(k):
        if(len(c_old[i])!=len(c_new[i])):
            return False
    for i in range(k):
        c_old[i].sort()
        c_new[i].sort()
        for j in range(len(c_old[i])):
            if(c_old[i][j]!=c_new[i][j]):
                return False
    return True
def k_means(k):
    ar=[]
    for i in range(k):
        x=[]
        ar.append(x)

    for i in range(k):
        ar[i].append(i)
    prev=ar.copy()
    while(True):
        means=[]
        for c in range(k):
            means.append(calculate_mean(ar[c]))
        curr=get_new_cluster(means,k)
        if(check(ar,curr,k)):
            break
        ar=curr.copy()
    return curr
K=int(input('Enter a value for K:'))
final_cluster=k_means(K)

for i in range(K):
    print(final_cluster[i])
x_point=[]
y_point=[]
for i in range(K):#
    x=[]
    y=[]
    x_point.append(x)#
    y_point.append(y)
for i in range(K):
    for x in final_cluster[i]:
        x_point[i].append(train_dataset[x][0])
        y_point[i].append(train_dataset[x][1])
fig,ax=plt.subplots()
colors=['blue','red','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_point[i],y_point[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()
```