Implementing K-Means Clustering

Mayeesha Humaira
dept. Computer Science and Engineering
Ahsanullah University of Science and Technology
Dhaka, Bangladesh
160204008@aust.edu
section - A1

***Abstract*—The main aim of this experiment is to implement K-Means Clustering algorithm to classify data points into two classes.**

***Index Terms*—centroid; Euclidean distance; Cluster;**

I. Introduction

k-means clustering is a simple algorithm. Firstly, number of clusters denoted by K must be determined and then pick centroids or centers of these clusters. Initial centroids can be randomly picked from the given data. The the K means algorithm can be performed in three steps given below until it converges.
Iterate until stable or no object move group:

1) Determine the centroid coordinate
2) Determine the distance of each object to the centroids using Euclidean distance. as sown in Eq. 1.

$$distance = \sqrt{(x_i - cenX_j)^2 + (y_i - cenY_j)^2} \quad (1)$$

where x_i an y_i are data provided and $cenX_j$ and $cenY_j$ are coordinates of the centroid.

3) Group the object based on minimum distance

II. Experimental Design / Methodology

One datasets named data_k_mean.txt was provided. Here 3000 data points were given. The following task were performed on this dataset.

1) **Task 1:**
Take input from the given source data file and plot all the points.
Solution:
Data points were plotted using matplotlib as shown in Fig 1.

2) **Task 2:**
Perform the k-means clustering algorithm applying Euclidean distance as a distance measure on the given dataset with k=2.
Solution:
K was a integer variable that was taken as input from user. Then using seed = 10 k number of centroids were picked from the given data. After that Euclidean distance was computed for every data points from the k centroids.

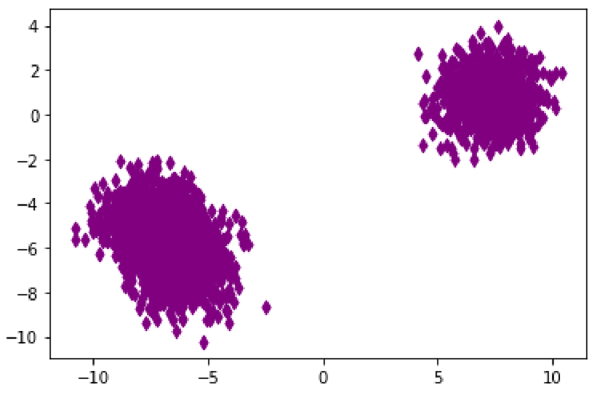

Fig. 1. Graphical representation of training data using matplotlib.

Afterwards, data points were grouped according to minimum distance. Then again k centroids were computed using mean of data in k groups. Using the new centroid Euclidean distance were computed again to see if any data moves from one cluster to another. If any data moves then the whole process is repeated otherwise the algorithm converges and the process stops.

3) **Task 3:**
Color the corresponding points on the clusters with different colors. Try to generalize your code as much as possible. Take random points from the given data as initial centroids.
Solution:
After performing the k-means clustering algorithm the given data points formed two clusters as shown in Fig. 2.

III. Result Analysis

If the initial seed is changed cluster also changes. Fig. 3 shows k=2 clusters using seed=100.

IV. Conclusion

K-means clustering is a simple clustering algorithms. The goal of k-means is to group data points into distinct non-overlapping subgroups. However, it also does not learn the number of clusters from the data and requires it to be predefined.

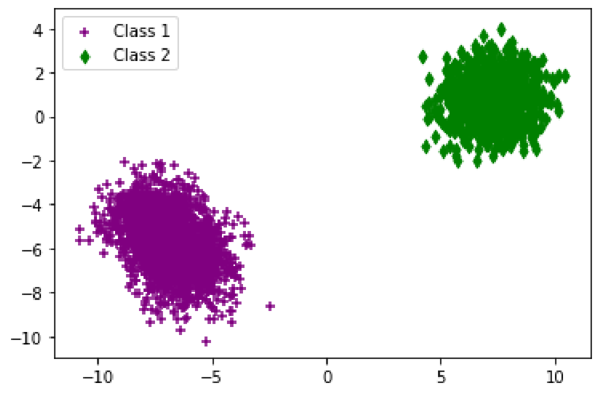

Fig. 2. Classified data points into k=2 clusters having initial seed=10.

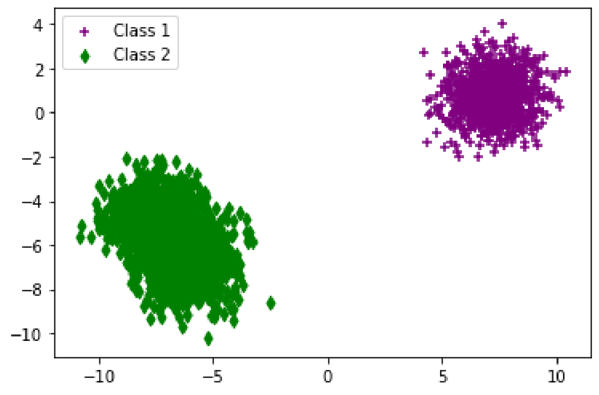

Fig. 3. Classified data points into k=2 clusters having initial seed=100.

V. Algorithm Implementation / Code

```
#Reading train data
import io
import pandas as pd
# reading the training data
df = pd.read_csv('data_k_mean.txt', sep = ' ',
    header = None)
df
data = df.to_numpy()
print(data)

import matplotlib.pyplot as plt
# plotting all data points
plt.scatter(df[0], df[1], c = 'purple', marker = 'd'
    )
plt.show()

# taking the input(k)
k = int(input("Enter the value of k : "))

# Performing k-men clustering
import numpy as np
# random centroids for first iteration
np.random.seed(seed=10)
random_numbers = np.random.randint(low=0, high=len(
    data), size=(k,))
centroids = [data[random_numbers[i]] for i in range(
    k)]
print(centroids)

distance = [] #to store the distance from point to
    classes
index_clusters = [-1 for i in range(len(data))] #to
    store class corresponding to index
count = 0 #to count the iteration number
clusters = {} #to store class numbers as keys and
    data points as values
# max 500 iterations
for x in range(500):
    count = x
    # flag to keep track whether change occurs or
    not
    flag = 0
    for y in range(k):
        clusters[y] = []
    # iterate through each data points
    for i in range(len(data)):
        distance = []

        for j in range(k):
            dist = np.sqrt(pow(abs((data[i][0] -
    centroids[j][0])), 2) + pow(abs((data[i][1] -
    centroids[j][1])), 2))
            distance.append(dist)
            #print(distance)
        index = distance.index(min(distance))
        #print(index)
        # check whether the change occurs or not
        if index_clusters[i] != index:
            flag = 1
            index_clusters[i] = index
        clusters[index].append(data[i])
    # if change occurs
    if flag == 0:
        break
    # calculating new centroids
    centroids = [np.mean(np.asarray(clusters[z]),
    axis=0) for z in range(k)]

#variables to plot class 1 and class 2 data points
x1 = np.asarray(clusters[0])[:, 0]
y1 = np.asarray(clusters[0])[:, 1]
x2 = np.asarray(clusters[1])[:, 0]
y2 = np.asarray(clusters[1])[:, 1]

# plotting classified data points of two classes
    with different colored marker
plt.scatter(x1, y1, c = 'purple', marker = '+',
    label = 'Class 1')
plt.scatter(x2, y2, c = 'green', marker = 'd', label
     = 'Class 2')
plt.legend(loc = 'best')
plt.show()
```