K-Means Clustering implementation

Md. Ashraful Haque
dept. Computer Science and Engineering
Ahsanullah University of Science and Technology
Dhaka, Bangladesh
160204101@aust.edu

***Abstract*—K-Means Clustering is one of the most famous Clustering algorithms which uses the unsupervised method to classify a unlabelled data set into n clusters.**

I. Introduction

The algorithm randomly initializes n centroids for n clusters. Then classify each data point according to the euclidean distances from the centroids. This algorithm runs until the k clusters have no change in their members.

II. Experimental Design / Methodology

step 1 Collecting the test data. In our case since it is a unsupervised learning technique, we have test data points without labels.

step 2

User select the number K which is the number of clusters the algorithm should produce.

step 3

An iteration goes through all the test data points (xi, yi). For each data point of the test data, the distances between all centroids and the test data are calculated.

step 4

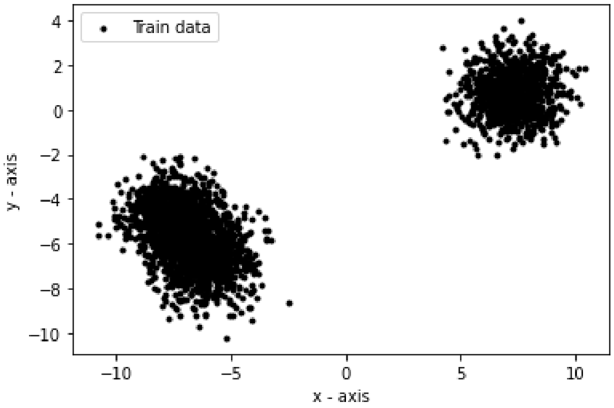
The class of the centroid which is the closest to the data point is selected to be the class of that data.

step 5

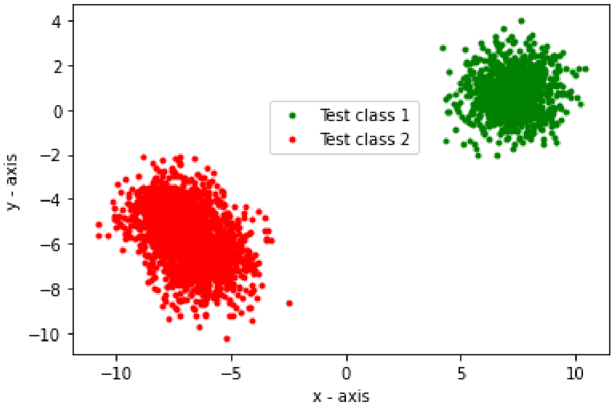
New Centroids are calculated after the classification is done in each iteration. Usually the mean of the class is taken to be the centroid after first iteration.

III. Result Analysis

Task 1 Ploting the input



Task 2 Classification results of the test set (k = 2)



IV. Conclusion

So, by these experiments we can come to the conclusion that by variations of the different kind, we can optimize the performance of the clustering algorithm.

V. Algorithm Implementation / Code

Task 1:

```
trainData = {}
entry = []

with open("data_k_mean.txt") as f:
    for line in f:
        entry = line.split(" ")

        if (entry[1][-1] == ’\n’):
            entry[1] =  entry[1][:-1]

        classInfo = 0

        trainData[(float(entry[0]),float(entry[1]))]
     = classInfo
trainData

import matplotlib.pyplot as plt

trainDatax = []
trainDatay = []

coorList = []

for coor  in trainData.keys():
    trainDatax.append(coor[0])
    trainDatay.append(coor[1])
    coorList.append(coor)


trainDatax

trainDatay

plt.scatter(trainDatax, trainDatay, color = ’k’,
    marker = ".", label = ’Train data’)


plt.xlabel(’x - axis’)
plt.ylabel(’y - axis’)

plt.legend(loc = ’upper left’)

plt.show()
```

Task 2 :

```
import math

def euclidianDistance(queryInstance, trainingSamples
    ):
    distance = math.sqrt(math.pow(queryInstance[0] -
     trainingSamples[0] , 2) + math.pow(
    queryInstance[1] - trainingSamples[1],2 ) )


    return distance

def classify(coorList,centroidList, k ):


    centroid1 , centroid2 = centroidList[0] ,
    centroidList[1]
    trainData = {}
    for eachCoordinate in coorList:
        if(euclidianDistance(centroid1 ,
    eachCoordinate) <= euclidianDistance(centroid2 ,
     eachCoordinate)):
            trainData[eachCoordinate] = 1
        else:
            trainData[eachCoordinate] = 2

    return trainData


x = {1:2 , 2 :3}
y = {1:5 , 2 :3}

shared_items = {k: x[k] for k in x if k in y and x[k
    ] == y[k]}
print(len(shared_items))

def getAVG(data):
    x_values = 0
    y_values = 0
    for i in data:
        x_values += i[0]
        y_values += i[1]

    mean = (x_values/len(data) , y_values/len(data))

    return mean

def getNewCentroids(trainData, k):

    class1 = []
    class2 = []

    centroidList = []

    for key , value in trainData.items():
        if(value == 1):
            class1.append(key)
        else:
            class2.append(key)

    #print(class1)
    #print(class2)

    centroid1 = getAVG(class1)
    centroid2 = getAVG(class2)


    return centroid1 , centroid2

k =  2
centroidList = []
centroidList = random.sample(range(len(coorList)), k
    )
centroidList

k =  2
centroidIndexList = []
centroidIndexList = random.sample(range(len(coorList
    )), k)

centroidList = []

for eachIndex in centroidIndexList:
    centroidList.append(coorList[eachIndex])

shared_items = {}

#trainData = newTrainData
while(len(shared_items) < int(len(coorList))):

    newTrainData = classify(coorList,centroidList,k)

    print(newTrainData)
    shared_items = {k: newTrainData[k] for k in
    newTrainData if k in trainData and newTrainData[
    k] == trainData[k]}
    print(len(shared_items))
    #print(len(coorList))


    trainData = newTrainData

    centroidList = []
    centroidList = getNewCentroids(trainData ,k )





#if(shared_items < len(coorList)):
```

Task 3:

```
testData1x = []
testData1y = []
testData2x = []
testData2y = []

for coor , classinfo in trainData.items():
    if(classinfo == 1):
        testData1x.append(coor[0]);
        testData1y.append(coor[1]);
    else:
        testData2x.append(coor[0]);
        testData2y.append(coor[1]);

print(testData1x , testData1y)
print(testData2x , testData2y)




plt.scatter(testData1x, testData1y, color = ’g’,
    marker = ".", label = ’Test class 1’)

plt.scatter(testData2x, testData2y, color = ’r’,
    marker = ".", label = ’Test class 2’)


plt.xlabel(’x - axis’)
plt.ylabel(’y - axis’)

#plt.legend(loc = ’center left’)
plt.legend(loc=’center left’, bbox_to_anchor=(.35,
    0.65))

plt.show()
```