Implementing K-Means Clustering

Tahira Salwa Rabbi Nishat
Computer Science and Engineering
Ahsanullah University of Science and Technology
Dhaka, Bangladesh

***Abstract*—K means clustering is a unsupervised learning problem. We were given a set of data points. We randomly chose two points and took them as initial centroid. Then for every data point we were given, we measured their Euclidian distance from the two centroids, assigned the data to cluster according to their lowest distance from any centroid. Then took the mean of each cluster and calculated the distance from each given data point to the mean. We did this until all assigned categories don't change any more.**

***Index Terms*—K-means clustering, unsupervised classification, unsupervised k means, clustering problems.**

I. Introduction

Clustering is the process of dividing the entire data into groups (also known as clusters) based on the patterns in the data. All the data points in a cluster should be similar to each other.And The data points from different clusters should be as different as possible.

K-Means clustering is an unsupervised clustering method that tries to minimize the distance of the points in a cluster with their centroid. The main objective of the K-Means algorithm is to minimize the sum of distances between the points and their respective cluster centroid.

The distance is measured using the Euclidean formula of measuring distance which is:

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

Here, (x_1, y_1) is a point of the train data and (x_2, y_2) is a point from the test data. Then the distance is sorted and ranked based on the lowest distances.

II. Experimental Design / Methodology

We were given a data set containing 300 data points. The main task was to plot the points, apply the K-Means algorithm and assign them into clusters.

The steps I have followed for this experience are:

1) First, I have loaded the train file and test file using read_csv() and plotted them using matplotlib.pyplot.
2) Then I took user input of number clusters k and selected k random points from the data as centroids.
3) Then I calculated the distance from centroids to each data points and assigned all the points to the closest cluster centroid.
4) Recomputed the centroids of newly formed clusters. these are the mean of the clusters.
5) Then I repeated steps 3 and 4 until all assigned categories don't change any more.

III. Result Analysis

The K-Means clustering algorithm assumes that the data points that stays closer to each other will belong to the same category and thus categorizes them. From the data points we were given, all of them looked visually distinguishable from each other.

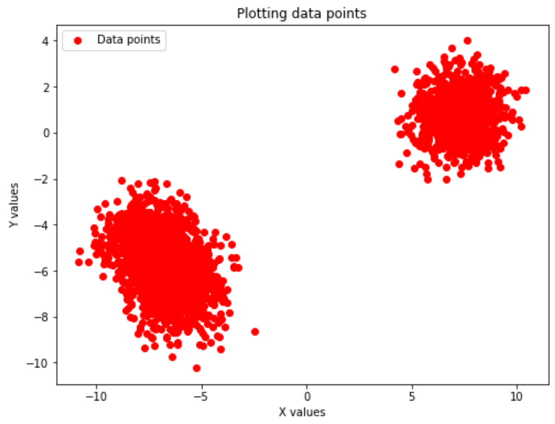

Fig. 1. Plotting datapoints

So, the K-means clustering algorithm was able to categorize them almost perfectly.

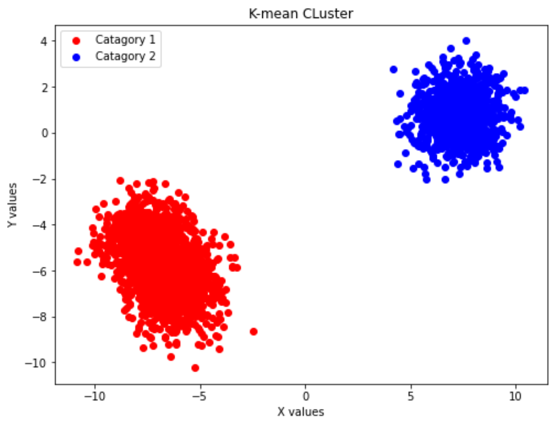

Fig. 2. Plotting categorized datapoints

The inter-cluster distance was quite high where the intra-cluster distance was relatively very low. So, the algorithm was able to categorize them nicely.

IV. Conclusion

Our model works wonderfully for the given data points. It was able to categorize every point nicely. The K means classifier is an unsupervised classifying algorithm. It uses some really basic but strong theory for classifying. It was also easy to implement. But problem will arise if the data points are not easily cluster-able. If the inter-cluster distances get lower and intra-class distances get bigger, using K-means algorithm may not be the most efficient way then. If the data points don't get stabilized after a certain iteration, we may need to put a threshold value to minimize the iteration cost.

V. Algorithm Implementation / Code

```
# -*- coding: utf-8 -*-
"""160204070_assignment5.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1
    uEaa3B_kTk_rnrwpFqIv9PMQJ-tqh7L7
"""

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from random import randint

from google.colab import files
upload = files.upload()

df = pd.read_csv('data_k_mean.txt', sep=" ", header=
    None, dtype='float')

data = df.values

data_length = len(data)
print(data_length)

fig, ax = plt.subplots(1, figsize = (8, 6))
plt.scatter(data[:, 0], data[:, 1], color = 'red',
    marker = 'o',label="Data points")
plt.title('Plotting data points')
plt.xlabel("X values")
plt.ylabel("Y values")
plt.legend(loc="best")

k = input("enter the value of k:")

rn1 = randint(0, data_length - 1)
rn2 = randint(0, data_length - 1)
print(rn1)
print(rn2)

centorid_1 = np.zeros(2)
centorid_2 = np.zeros(2)
centorid_1 = data[rn1, :]
centorid_2 = data[rn2 ,:]
print(centorid_1)
print(centorid_2)
print(type(centroid))

classifier = np.zeros(data_length)
for j in range(0, 1000):
    cls1, cls2 = 0, 0
    cls1_nmbr, cls2_nmbr = 0, 0
    for i in range(0, data_length):
        dis1 = np.sqrt(np.power((data[i][0] -
    centorid_1[0]), 2) + np.power((data[i][1] -
    centorid_1[1]), 2))
        dis2 = np.sqrt(np.power((data[i][0] -
    centorid_2[0]), 2) + np.power((data[i][1] -
    centorid_2[1]), 2))
        if dis1 <= dis2:
            cls1 += 1
        else:
            cls2 += 1

    firstclass = np.zeros((2, cls1))
    secondclass = np.zeros((2, cls2))

    count = 0

    for i in range(0, data_length):

        dis1 = np.sqrt(np.power((data[i][0] -
    centorid_1[0]), 2) + np.power((data[i][1] -
    centorid_1[1]), 2))
        dis2 = np.sqrt(np.power((data[i][0] -
    centorid_2[0]), 2) + np.power((data[i][1] -
    centorid_2[1]), 2))
        if dis1 <= dis2:
            firstclass[0, cls1_nmbr] = data[i][0]
            firstclass[1, cls1_nmbr] = data[i][1]
            if classifier[i] != 1:
                count += 1
            classifier[i] = 1
            cls1_nmbr += 1
        else:
            secondclass[0][cls2_nmbr] = data[i][0]
            secondclass[1][cls2_nmbr] = data[i][1]
            if classifier[i] != 2:
                count += 1
            classifier[i] = 2
            cls2_nmbr += 1
    if count == 0:
        print("Iterration: " ,j)
        break

    centorid_1[0] = np.mean(firstclass[0, :])
    centorid_1[1] = np.mean(firstclass[1, :])
    centorid_2[0] = np.mean(secondclass[0, :])
    centorid_2[1] = np.mean(secondclass[1, :])

fig, ax = plt.subplots(1, figsize = (8, 6))
plt.scatter(firstclass[0], firstclass[1], color='Red
    ', label="Catagory 1")
plt.scatter(secondclass[0], secondclass[1], color='
    Blue', label="Catagory 2")
plt.title('K-mean CLuster')
plt.xlabel("X values")
plt.ylabel("Y values")
plt.legend(loc="best")
plt.show()
```