Implementing K-Means Clustering

Rashedun Nobi Chowdhury
ID: 160204039
Group: A2
Email: 160204039@aust.edu

***Abstract*—Of different sectors in computer science, machine learning is one of the most significant. In recent years application of machine learning has taken off. Two types of machine learning techniques exist, namely, supervised techniques and unsupervised techniques. Unsupervised training is usually performed using an unlabeled dataset to find patterns that may not be understandable at a first glance. In this experiment, I implement K Means Clustering which is an unsupervised machine learning model. From a given unlabeled dataset the model divides them into K clusters based on the similarities. I have also performed some basic experiment on the model.**

***Index Terms*—Machine Learning, Unsupervised Learning, K Means Clustering**

I. Introduction

K means clustering is one of the simplest unsupervised machine learning techniques to implement. In supervised learning we use a labeled dataset to train the model. After training the model is able to predict new data and classify them based on the knowledge base. In unsupervised learning however, the dataset is not labeled. The model is used to find patterns on the dataset and put similar data points in the same cluster. In K Means Clustering, the model starts with K random centroids. Let us consider K=2, then we start with 2 random centroids. Data points closest to a specific point creates a cluster. The centroids are updated at each iteration based on the clusters they form. For the given dataset, the points look like 1.

To calculate the distance between the I am using euclidean distance. Euclidean distance can be calculated as,

$$distance = \sqrt{(x_{train}[0] - x_{test}[0])^2 + (x_{train}[1] - x_{test}[1])^2} \quad (1)$$

It should be mentioned that I have worked with K=2 clusters. The rest of the report is organized as follows, section 2 contains the experimental design. In section 3 I briefly describe the results. In section 4 I conclude the report with some discussions on the advantages and disadvantages of the algorithm. Finally in section 5, I attach a snapshot of my implemented code.

II. Experimental Design / Methodology

For this experiment, I have implemented the algorithm in python. I used **MatPlotLib** to plot the data points.

Algorithm 1 K Means Clustering

1) Read the dataset
2) Take K random points as cluster
3) For each data, calculate distance using 1
4) Assign the data to a cluster from which it is the closest
5) Calculate the centroid again by taking the mean of all the points in the cluster
6) Repeat 2-5 until no points move to a different cluster
7) Plot the data using appropriate markers for each cluster

III. Result Analysis

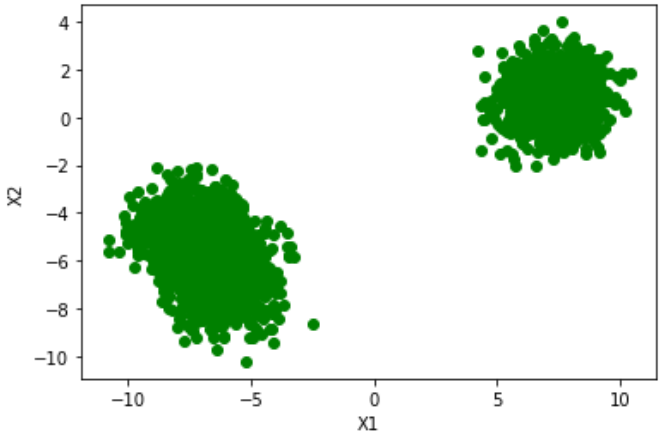

Fig. 1. Distribution of Data Points

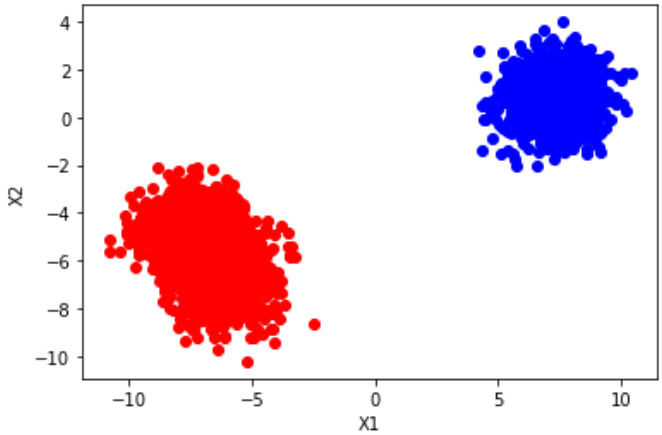

Fig. 2. Data Points divided in 2 clusters

In figure 2, the data points are divided into two clusters with the help of K Means Clustering algorithm.

IV. Conclusion

K means clustering is really easy to implement. The algorithm also requires low computational resources. However it may struggle to converge if there are some data points that are almost in the middle and often switch clusters thus creating ambiguity for the centroids.

V. Algorithm Implementation / Code

A snapshot of my implementation of the algorithm in python is given below,

```
centroid_1 = dataset.iloc[random.randint(0,3000-1),:].values
centroid_2 = dataset.iloc[random.randint(0,3000-1),:].values


change = True
cluster_1 = []
cluster_2 = []
while(change):
    change = False

    cluster_1 = []
    cluster_2 = []
    cluster_1.append(centroid_1)
    cluster_2.append(centroid_2)

    previous_centroid_1 = centroid_1
    previous_centroid_2 = centroid_2

    for x in range(len(plot_points)):
        distance_1 = math.sqrt((plot_points[x][0]-centroid_1[0])**2 + (plot_points[x][1]-centro
        distance_2 = math.sqrt((plot_points[x][0]-centroid_2[0])**2 + (plot_points[x][1]-centro
        if(distance_1<distance_2):
            cluster_1.append(plot_points[x])
        else:
            cluster_2.append(plot_points[x])

    centroid_1 = np.mean(np.array(cluster_1),axis = 0)
    centroid_2 = np.mean(np.array(cluster_2),axis = 0)


    if(np.array_equal(centroid_1, previous_centroid_1) and np.array_equal(centroid_2, previous
        change = False
    else:
        change = True
```