Negar Nejatishahidin
G01207447

# HW3 (Part 1) - Iris Clustering

The main goal of this project is to implement the KMeans algorithm. So I will first talk about the K-mean first.

## Kmeans clustering:

Kmeans algorithm is an iterative algorithm that tries to partition the dataset into *K* predefined distinct non-overlapping subgroups (clusters) where each data point belongs to only one group. It tries to make the inter-cluster data points as similar as possible while also keeping the clusters as different (far) as possible. It assigns data points to a cluster such that the sum of the squared distance between the data points and the cluster's centroid (arithmetic mean of all the data points that belong to that cluster) is at the minimum. The less variation we have within clusters, the more homogeneous (similar) the data points are within the same cluster.

The way kmeans algorithm works is as follows:

- Specify number of clusters *K*.
- Initialize centroids randomly selecting *K* data points for the centroids without replacement.
- Keep iterating until there is no change to the centroids. It means that assignment of data points to clusters isn't changing.

## Kmean++:

In some cases, if the initialization of clusters is not appropriate, K-Means can result in arbitrarily bad clusters. This is where K-Means++ helps. It specifies a procedure to initialize the cluster centers before moving forward with the standard k-means clustering algorithm.

Using the K-Means++ algorithm, we optimize the step where we randomly pick the cluster centroid. We are more likely to find a solution that is competitive to the optimal K-Means solution while using the K-Means++ initialization.

The steps to initialize the centroids using K-Means++ are:

1. The first cluster is chosen uniformly at random from the data points that we want to cluster. This is similar to what we do in K-Means, but instead of randomly picking all the centroids, we just pick one centroid here
2. Next, we compute the distance $D(x)$ of each data point ($x$) from the cluster center that has already been chosen
3. Then, choose the new cluster center from the data points with the probability of x being proportional to $(D(x))2$
4. We then repeat steps 2 and 3 until k clusters have been chosen

Lets talk about my own Implementation :

1) First select the centroids with the Kmean++ algorithm which I described previously.

```
def Random_centroid(self):
    """
    k centroid  with the KMean++ algorithm.
    """
    centroid_id = np.random.choice(self.X.shape[0], 1)
    self.centroid[0] = self.X[centroid_id]
    for i in range(1, self.k):
        dist_temp =  euclidean_distances(self.X, self.centroid[0:i] )
        t = np.argmax(np.amin(dist_temp, axis = 1))
        self.centroid[i] = self.X[t]
```

2) build the def to calculate the distance of each data points to the all centroids with the euclidean distances. The datapoint is in the cluster of centroid which is the nearest (the least euclidean distances) .

3) I have also calculate the centroid after each iteration in a way that the new centered is the mean of all the points in the cluster.

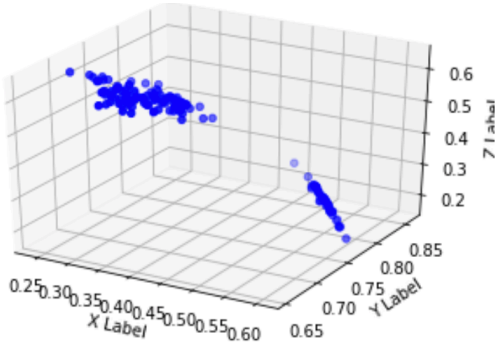
4) then, I have also calculate the error (sum of the distances). It is useful for both Bisecting KMeans and the Part 2. It also useful to check if the algorithm works or not.

5) The final part is the Main part. This algorithm is to first calculate the initial centroids,  then calculate the distance and cluster of each post. After that recalculate the centroids.

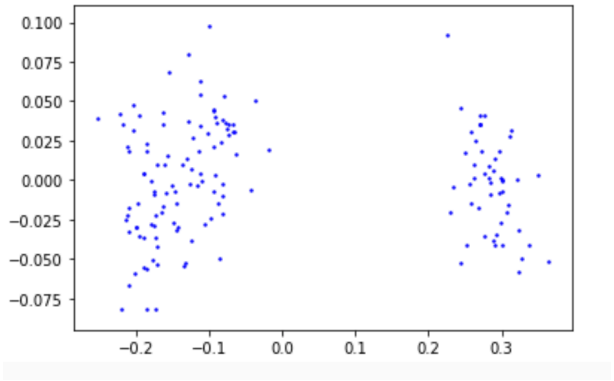
# Iris Clustering

## Dataset :

The dataset is Irish dataset. It consists of 150 recored with 4 features each. The rows being the samples and the columns being: Sepal Length, Sepal Width, Petal Length and Petal Width.
The visualization of the Dataset with the 3 features out of the 4 features :




We can also use PCA to reduce the dimension into 2 and then see the visualization in 2D :




The Main_Kmean.ipython file is implemented the Kmean clustering over this dataset. I have normalized the data before the clustering, so it will increase the accuracy greatly.

I have used my Kmean clustering algorithm with K= 3 and 30 times for the iteration. Since the first centroid is random, I did this task for 10 times and finally use the one with the least error.

## Run Time :

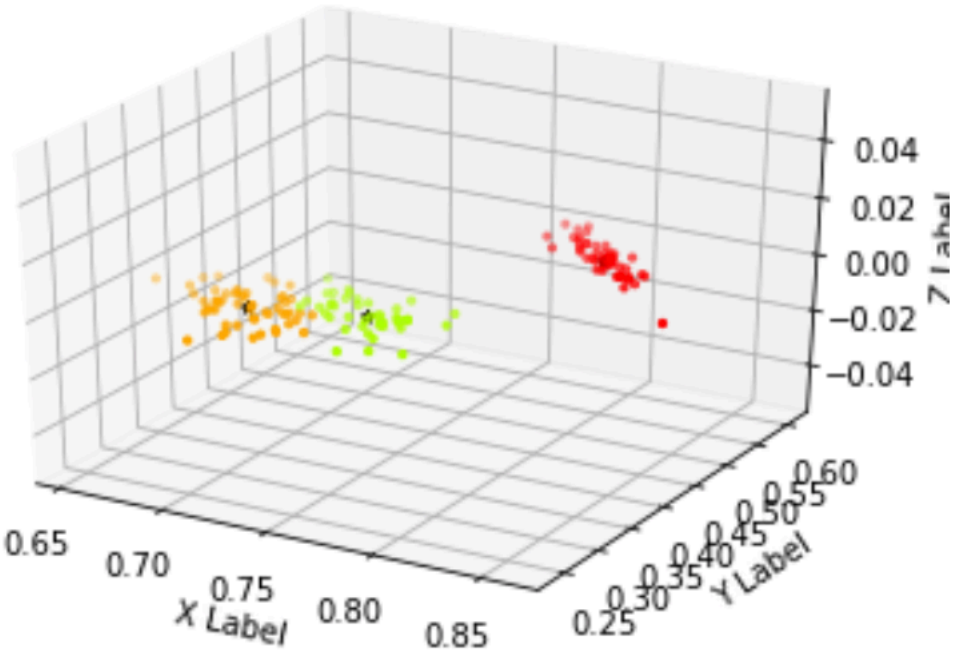
The whole KMeans algorithem for this dataset takes 0.1877439022064209 seconds to run.

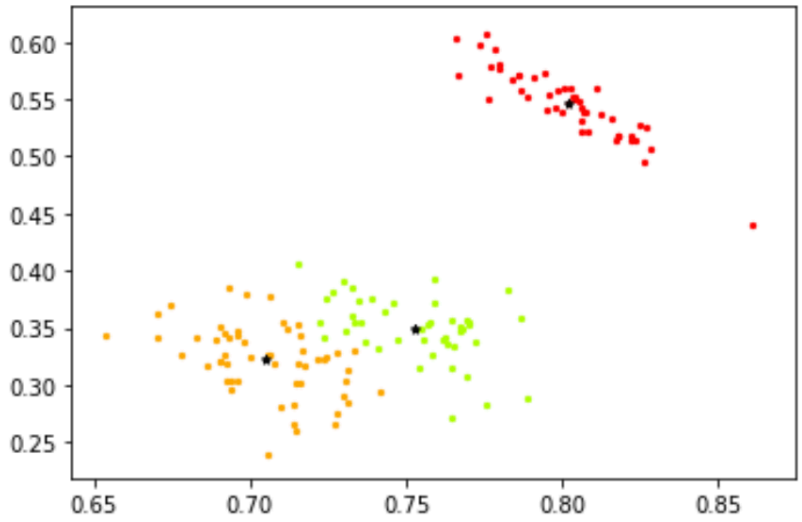
## accuracy:

The accuracy is 95% on data.

## Clustering Visualization:

Here is the visualization of the results in 3D and 2D:






## Main Bisecting KMeans :

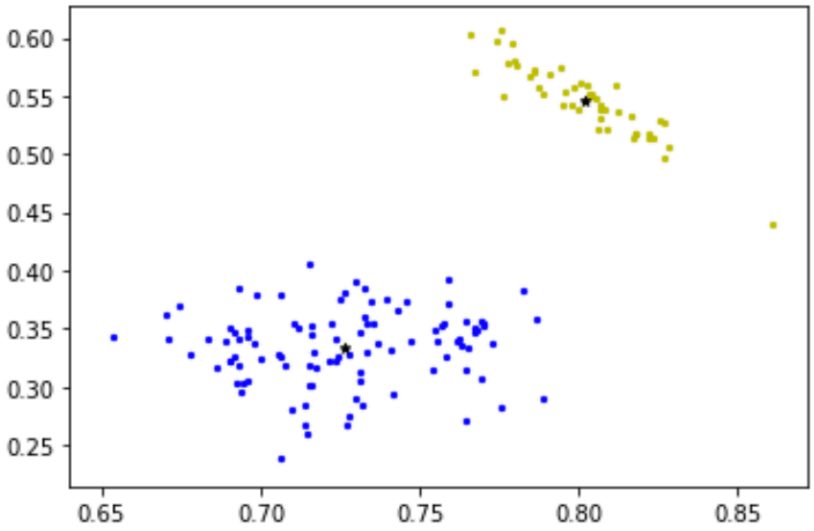
The algorithm is implemented with the use of previous Kmeans algorithm. The whole process is in 3 steps :

First: do Kmean clustering for 2 cluster instead of 3.
This part takes 0.01410984992980957 seconds
We can also visualize this two cluster :

```
from KMean import *

BKmean_Irish = KMeans(iris_np, 2,4)
BKmean_Irish.Random_centroid()
B2Error, B2label, Bcentroid = BKmean_Irish.Main(10)
```

The class with bigger error is the blue class.




Second: find the cluster with bigger error,

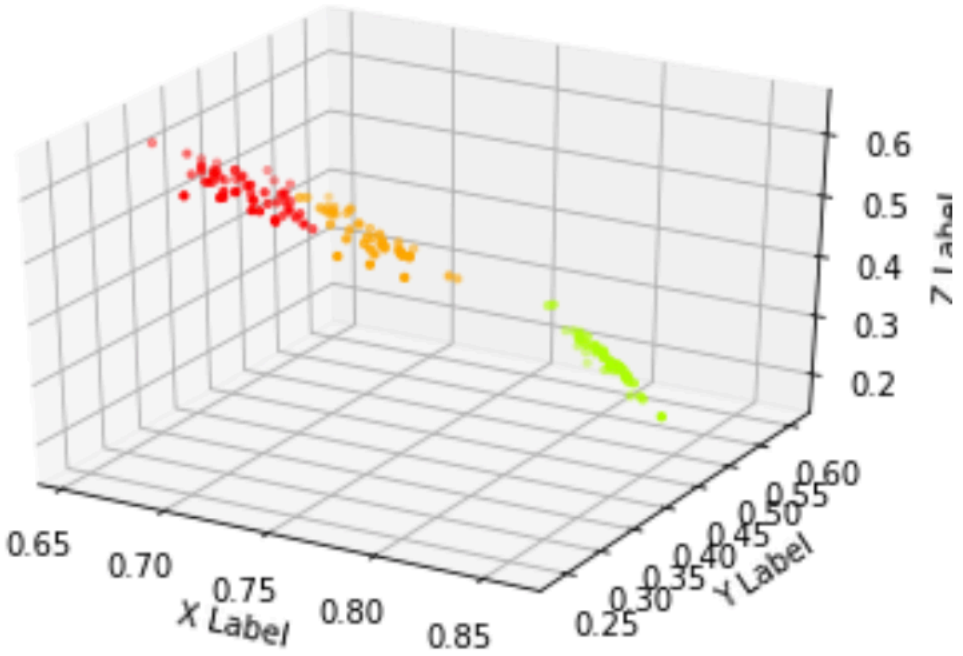
```
High error = np.argmax(B2Error)
```

Finally: do Kmean clustering for that cluster with k=2 .

This part takes 0.020317792892456055 seconds.

```
B3Kmean_Irish = KMeans(iris_np[B2label == i], 2, 4)
B3Kmean_Irish.Random_centroid()
B3Error, B3label, B3centroid = B3Kmean_Irish.Main(10)
```

The accuracy for this clustering algorithm is also 95% .


0.6
0.5
0.4
0.3
0.2
Z Label
0.65
0.70
0.75
0.80
0.85
X Label
0.25
0.30
0.35
0.40
0.45
0.50
0.55
0.60
Y Label