Shraddha Bhalerao - 1217285576

Report: Project Part 2: Unsupervised Learning (K-means)

Clustering is a type of Unsupervised Machine learning i.e. it needs no training data, it performs the computation on the actual dataset. This should be apparent from the fact that in K Means, we are just trying to group similar data points into clusters, there is no prediction involved. K-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster.

Basic K-Means Algorithm:

```
Given: n samples, a number k.
Begin
    initialize μ1, μ2, …, μk(randomly
    selected)
          do classify n samples according to
                       nearest μi
             recompute μi
          until no change in μi
    return μ1, μ2, …, μk
  End
```

K-Means algorithm works as follows, assuming we have inputs $x_1, x_2, x_3, \ldots, x_n$ and value of K

- **Step 0 – Extract/Load data set.**

  Given data is in .mat file. Extracting that data to numpy array.

```
#Extract Data from the matlab data file
Numpyfile = scipy.io.loadmat('AllSamples.mat')
SampleData = Numpyfile['AllSamples']
# print(SampleData.shape)
```

- **Step 1 - Pick K random points as cluster centres called centroids.**

  Two different Strategies are used while selecting centroids:

  1. **Selecting all the centroids randomly:** We randomly pick K cluster centres(centroids). Let's assume these are *c*1, *c*2..., *ck*, and we can say that;

     C = {c1, c2..., ck}

     C is the list of all centroids.

```
#select random centroid equal to the Number of Clusters we have
def selectCentroidRandom(NumberOfClusters,centroidList):
    centroidList = []
    pointSample = SampleData
    for c in range(0, NumberOfClusters):
        c = np.random.choice(pointSample.shape[0], 1, replace=False)
        centroidList.append(pointSample[c])
        pointSample = np.delete(pointSample, c, 0)

    centroidList = np.vstack( centroidList )
    # print('Centroids Selected are : ' , centroidList)
    return centroidList
```

  2. **Choosing the point furthest from the previous centres:** pick the first centre randomly; for the i-th centre (i>1), choose a sample (among all possible samples) such that the average distance of this chosen one to all previous (i-1) centres is maximal.

```
# select centroids as per the farthest point strategy equal to the Number of Clusters we have
# first centroid is selected randomly using  following equation
# c = np.random.choice(SampleData.shape[0], 1, replace=False)
# the other centroids are selcted to be fatrthest points from the above point and other centroids.

def SelectCentroidStrategyTwo(NumberOfClusters,centroidList,c):
    centroidList = []
    pointSample = SampleData
    centroidList = pointSample[c]
    pointSample = np.delete(pointSample, c, 0)

    for c in range(1, NumberOfClusters):
        distList = [dist(pointSample[j], centroidList) for j in range(len(pointSample))] #this is an numpy array
        if (NumberOfClusters > 2):
            avgDistArr = [np.mean(distList[i]) for i in range(len(distList))]
            maxDistIndex = np.argmax(avgDistArr)
        else:
            maxDistIndex = np.argmax(distList)
        centroidList = np.vstack([centroidList, SampleData[maxDistIndex]])
        pointSample = np.delete(pointSample, maxDistIndex, 0)

    plt.scatter(centroidList[:,0], centroidList[:,1], s = 180, c = 'k' , marker = '*')
    plt.title('Initial Centroid Chosen for Number of Clusters k = %i' %(NumberOfClusters))
    # print('Centroids Selected are : ' , centroidList)
    return centroidList
```

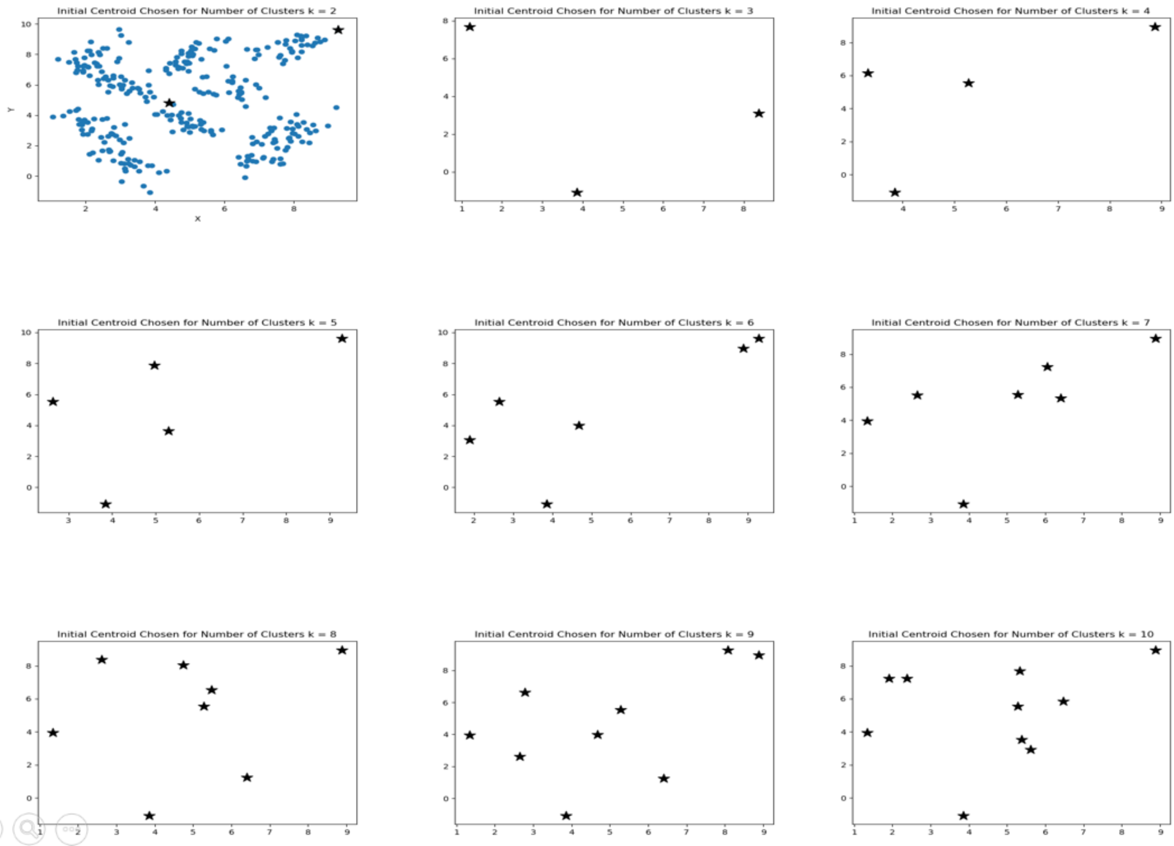

Figure 1 Initial Selection of Centroids as per Farthest Point Strategy

- **Step 2 - Assign each X_i to nearest cluster by calculating its distance to each centroid.**

  In this step we assign each input value to closest centre. This is done by calculating Euclidean(L2) distance between the point and each centroid.

$$\arg\min_{C_i \epsilon C} dist(C_i, X)$$

  Where dist(.)is the Euclidean distance.

```
# Euclidean Distance Calculator
def dist(dataPoint, Centroids, ax=1):
    return np.linalg.norm(dataPoint - Centroids, axis=ax)
```

```
while(ChangeInCentroidValue != 0):
    for i in range(len(SampleData)):
        distances = dist(SampleData[i], centroidList)
        cluster = np.argmin(distances)
        clusters[i] = cluster
```

- **Step 3 - Find new cluster centre by taking the average of the assigned points.**

  In this step, we find the new centroid by taking the average of all the points assigned to that cluster.

$$C_i = \frac{1}{|S_i|} \sum_{X_i \epsilon S_i} X_i$$

  S_i is the set of all points assigned to the i^{th} cluster.

```
# seperating sample data points into their respective clusters
# and Calculating new centroids over these Clusters
    for i in range(NumberOfClusters):
        samples = [SampleData[j] for j in range(len(SampleData)) if clusters[j] == i]
        samples = np.vstack(samples)
        centroidList[i] = np.mean(samples, axis=0)

# Calculating Change in centroid values
    ChangeInCentroidValue = dist(centroidList, CentroidPrev, None)
```

- **Step 4 - Repeat Step 2 and 3 until none of the cluster assignments change.**

  In this step, we repeat step 2 and 3 until none of the cluster assignments change. That means until our clusters remain stable, we repeat the algorithm.

Results:

We ran algorithm using two strategies explained above and for **Number of Clusters from 2 to 10.**

Clusters obtained **with different initializations using strategy 1 and 2** are as follows:

Here only one try from each strategy is shown. Other plots are submitted with code zip folder.

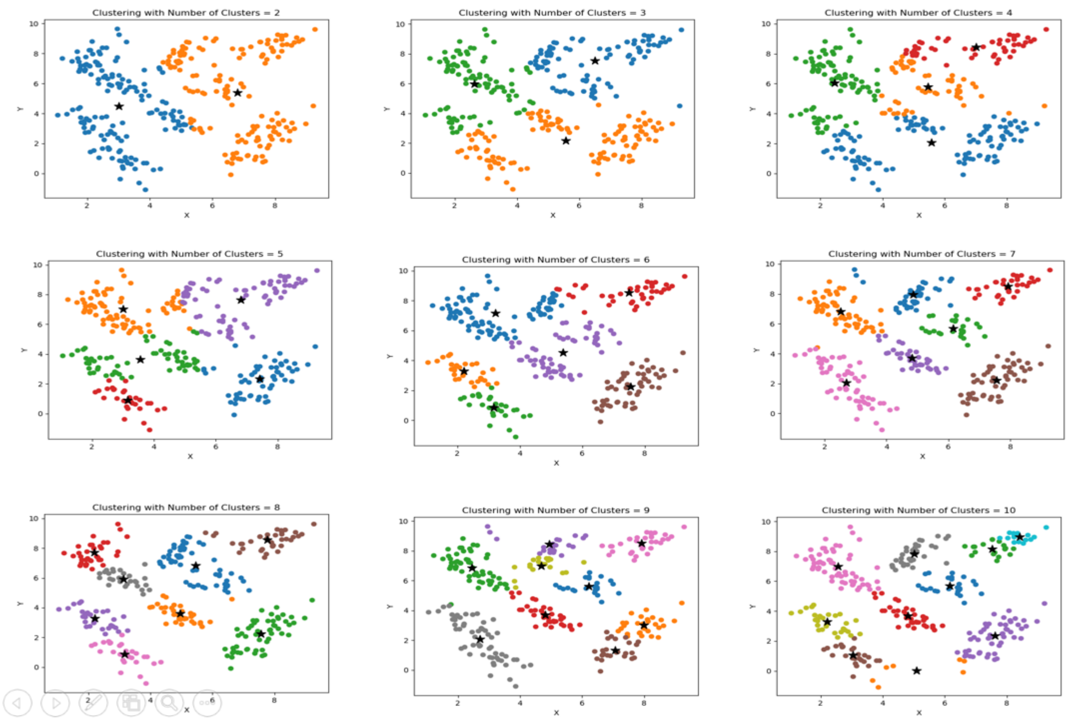

Figure 2 Clustering plots with k = 2 to 10 (from top left) Strategy 1 Try 1

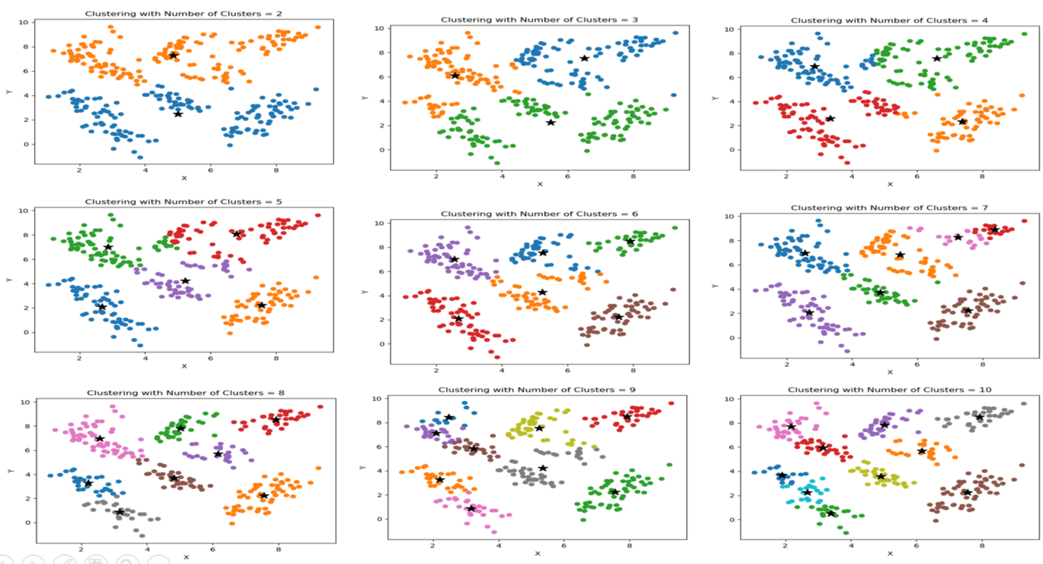

Figure 3 Clustering plots with k = 2 to 10 (from top left) Strategy 2 Try 1

Then We Calculated **Objective Function Value** and plotted it against the number of clusters.

When clustering the samples into k clusters/sets Di, with respective center/mean vectors μ1, μ2, … μk, the objective function is defined as

$$\sum_{i=1}^{k} \sum_{\mathbf{x} \in D_i} ||\mathbf{x} - \boldsymbol{\mu}_i||^2$$

Following are the plot we obtained under all the 4 conditions:

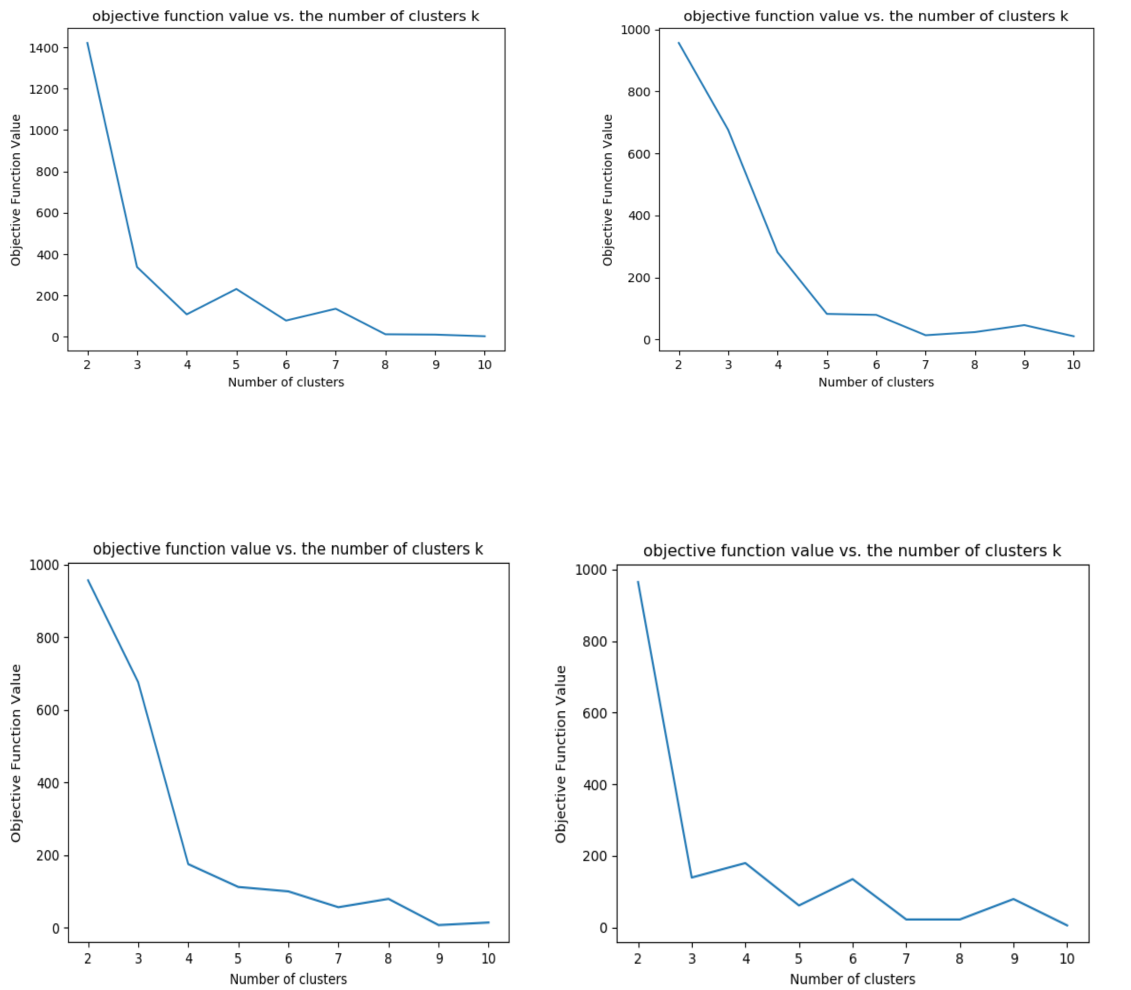

Figure 4 (A) Strategy 1 Try 1 (B) Strategy 1 Try 2 (C) Strategy 2 Try 1 (D) Strategy 2 Try 2 (From top Left)