PROJECT 2: UNSUPERVISED LEARNING (K-MEANS AND KMEANS++)

27th October 2019

OVERVIEW

1. Project Background and Description

In this project we were required to implement the k-means algorithm and apply this implementation on the given dataset, which contains 2-D points. We were required to implement two different strategies for choosing the initial clusters.

Strategy 1: **randomly pick the initial centers from the given samples.**

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

We had to test our implementation on the given data, with **the number k of clusters ranging from 2-10**. **Plot the objective function value vs. the number of clusters k**. Under each strategy, **plot the objective function twice**, each start from a different initialization.

2. Data Summary

The data given to us was Microsoft Access Table namely, "AllSamples.mat". To access this file in python we used the following code:

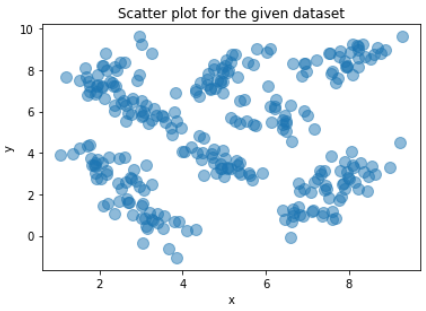
```
NumpyFile = scipy.io.loadmat("AllSamples.mat")
```

We used the NumPy library to load the data in the NumPy array format. The data can be visualized as follows:

	0	1
0	2.05925	7.20599
1	8.87578	8.96092
2	8.00706	2.77532
3	5.01729	3.76312
4	6.39056	5.17956
5	1.9548	7.78422
6	4.80754	3.03465
7	1.34837	3.9638
8	3.04102	-0.361385

The initial distribution of the given dataset can be visualized as given on the right. The x axis is the x coordinate or as stated above the elements in the first column of the table given above. Similarly, the y coordinates are the second column in the table stated above.

```
#turn on the interactive mode
plt.ion()
plt.scatter(NumpyFile['AllSamples'][:,0], NumpyFile['AllSamples'][:,1], alpha=0.5, s=100)
plt.title('Scatter plot for the given dataset')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



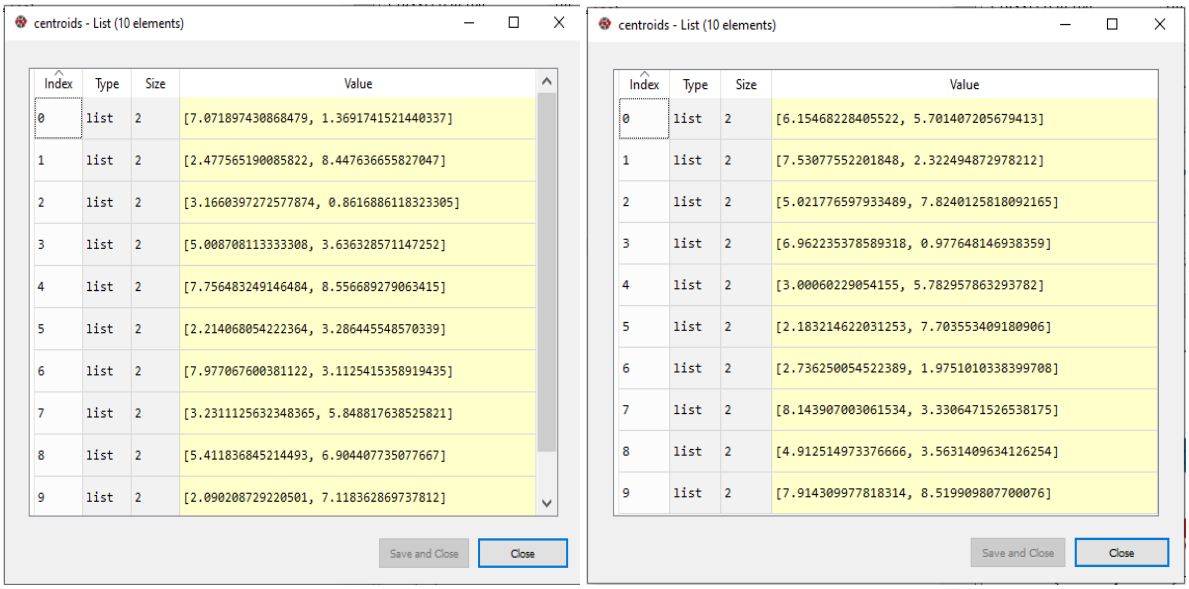
3. Strategy 1

According to the first strategy, we must **randomly pick the initial centers from the given samples.** To achieve this, I tried two basic strategies using the libraries NumPy and random.

First using the random I achieved it by using the following command:

```
centroids = random.sample(list(NumpyFile['AllSamples']),k=k)
```

The centroids returned in two iterations were as follows for k = 10,



centroids - List (10 elements)

Index	Type	Size	Value
0	list	2	[7.071897430868479, 1.3691741521440337]
1	list	2	[2.477565190085822, 8.447636655827047]
2	list	2	[3.1660397272577874, 0.8616886118323305]
3	list	2	[5.008708113333308, 3.636328571147252]
4	list	2	[7.756483249146484, 8.556689279063415]
5	list	2	[2.214068054222364, 3.286445548570339]
6	list	2	[7.977067600381122, 3.1125415358919435]
7	list	2	[3.2311125632348365, 5.848817638525821]
8	list	2	[5.411836845214493, 6.904407735077667]
9	list	2	[2.090208729220501, 7.118362869737812]

Save and Close | Close

centroids - List (10 elements)

Index	Type	Size	Value
0	list	2	[6.15468228405522, 5.701407205679413]
1	list	2	[7.53077552201848, 2.322494872978212]
2	list	2	[5.021776597933489, 7.8240125818092165]
3	list	2	[6.962235378589318, 0.977648146938359]
4	list	2	[3.00060229054155, 5.782957863293782]
5	list	2	[2.183214622031253, 7.703553409180906]
6	list	2	[2.736250054522389, 1.9751010338399708]
7	list	2	[8.143907003061534, 3.3306471526538175]
8	list	2	[4.912514973376666, 3.5631409634126254]
9	list	2	[7.914309977818314, 8.519909807700076]

Save and Close | Close

To plot the graph, we used the library matplotlib. We have plotted a scatter plot for the dataset we had, and the color coding is done on the basis of following array:

```
#source: https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.scatter.html
colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd', '#8c564b', \
          '#e377c2', '#7f7f7f', '#bcbd22', '#17becf']
```

The classification was stored in a dictionary. The graphs as shown on the next page the plotting was done using the following code.

```
for classification in classifications:
    color = colors[classification]
    for points in classifications[classification]:
        plt.scatter(points[0],points[1],color=color,s=200)
```

To plot the centroid values as ‘x’ we used the following code:

```
for centroid in centroids:
    plt.scatter(centroid[0],centroid[1],marker='x',color='k',s=200)
```

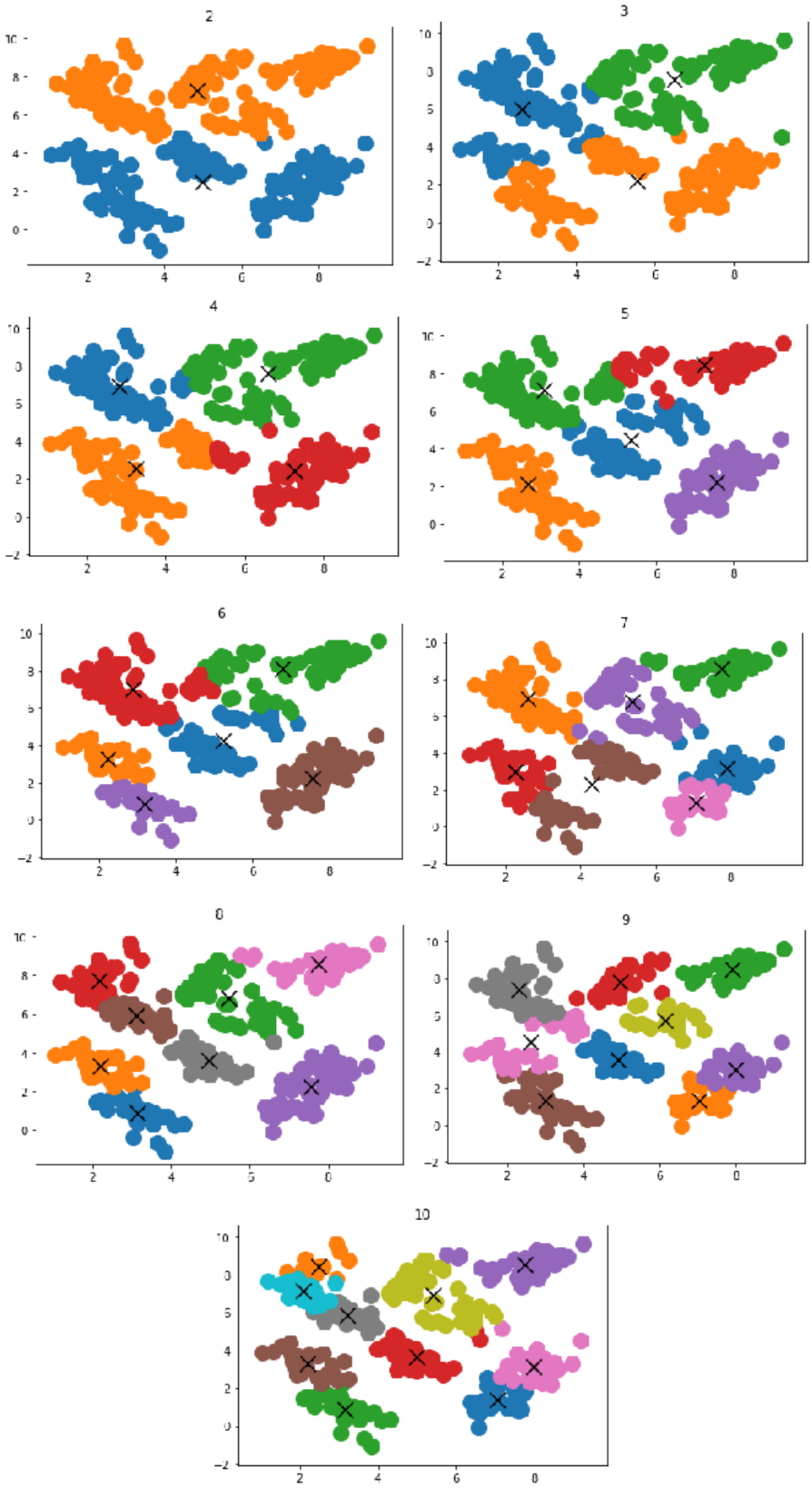


Figure 1 All the images above show the scatter plot with each classification being highlighted with different color and the centroid values given with 'x'

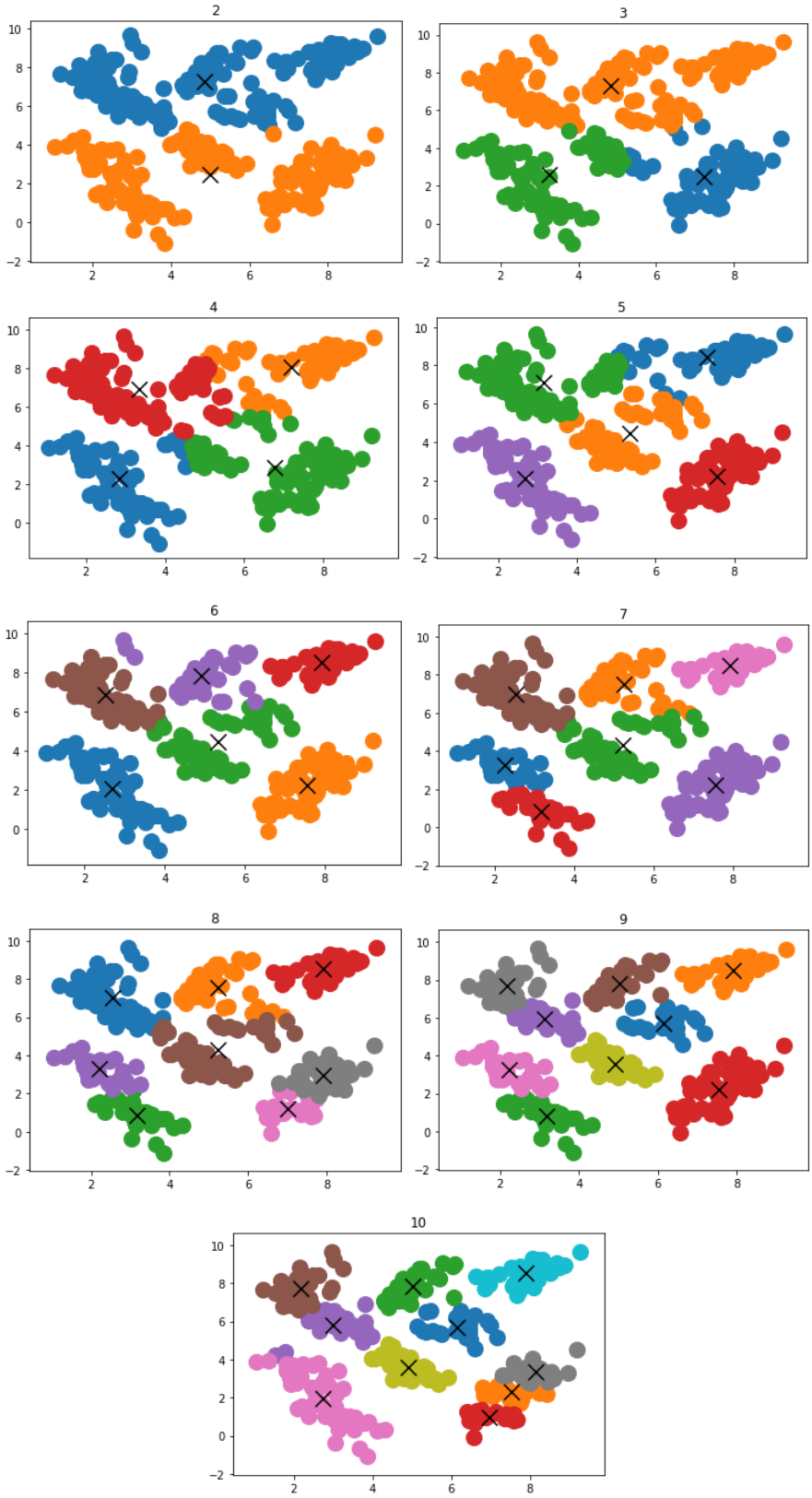


Figure 2 All the images above show the scatter plot with each classification being highlighted with different color and the centroid values given with 'x' for iteration 2 strategy 1

For the strategy 1 we had thought of generalizing this a little further but not as good as k means++ by taking the initial values in the range of sample space as follows:

```
minRangeX = min(NumpyFile['AllSamples'][:,0])
maxRangeX = max(NumpyFile['AllSamples'][:,0])
minRangeY = min(NumpyFile['AllSamples'][:,1])
maxRangeY = max(NumpyFile['AllSamples'][:,1])
```

The code for this procedure was as follows:

```
#initialize the centroid value with random values in the given range of values
centroids = (max(maxRangeX,maxRangeY)-min(minRangeX,minRangeY))*\
np.random.rand(k,2)+min(minRangeX,minRangeY)
```

4. Strategy 2

According to the strategy 2, we must pick the **first center randomly**; **for the i-th center (i>1), choose a sample (among all possible samples) such that the average distance of this chosen one to all previous (i-1) centers is maximal**.

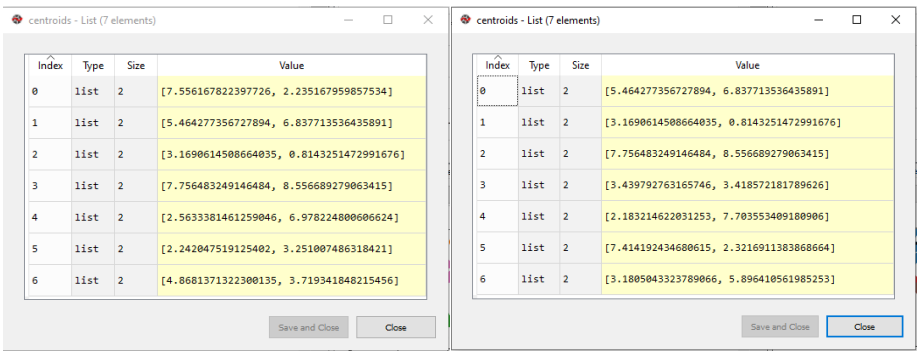
We achieved this, by writing the following function:

```
def calculateInitialCentroids(samples,k):
    centroids = []
    #pick the first center randomly
    #source: https://pynative.com/python-random-choice/
    centroids.append(np.ndarray.tolist(random.choice(samples)))
    #for the i-th center (i>1), choose a sample (among all possible samples)
    #such that the average distance of this chosen one to all previous (i-1)
    #centers is maximal.
    ## there needs to be k centroids therefore range 0 to k-1
    for i in range(0,k-1):
        averageDistanceFromAllCentroids = {}
        for value in samples:
            # credits to
            # https://stackoverflow.com/questions/1401712/how-can-the-euclidean-distance-be-calculated-with-numpy
            # for following line
            distanceFromCentroids = [np.linalg.norm(value-centroid) for centroid in centroids]
            averageDistanceFromAllCentroids[tuple(value)] = np.mean(distanceFromCentroids)
        centroids.append(list(max(averageDistanceFromAllCentroids, key=averageDistanceFromAllCentroids.get)))
    return centroids
```

The function call can be traced as follows:

```
#initialize the centroid value with random values in the given range of values
centroids = calculateInitialCentroids(NumpyFile['AllSamples'],k)
```

The initial centroid achieved for the two iterations are as follows:



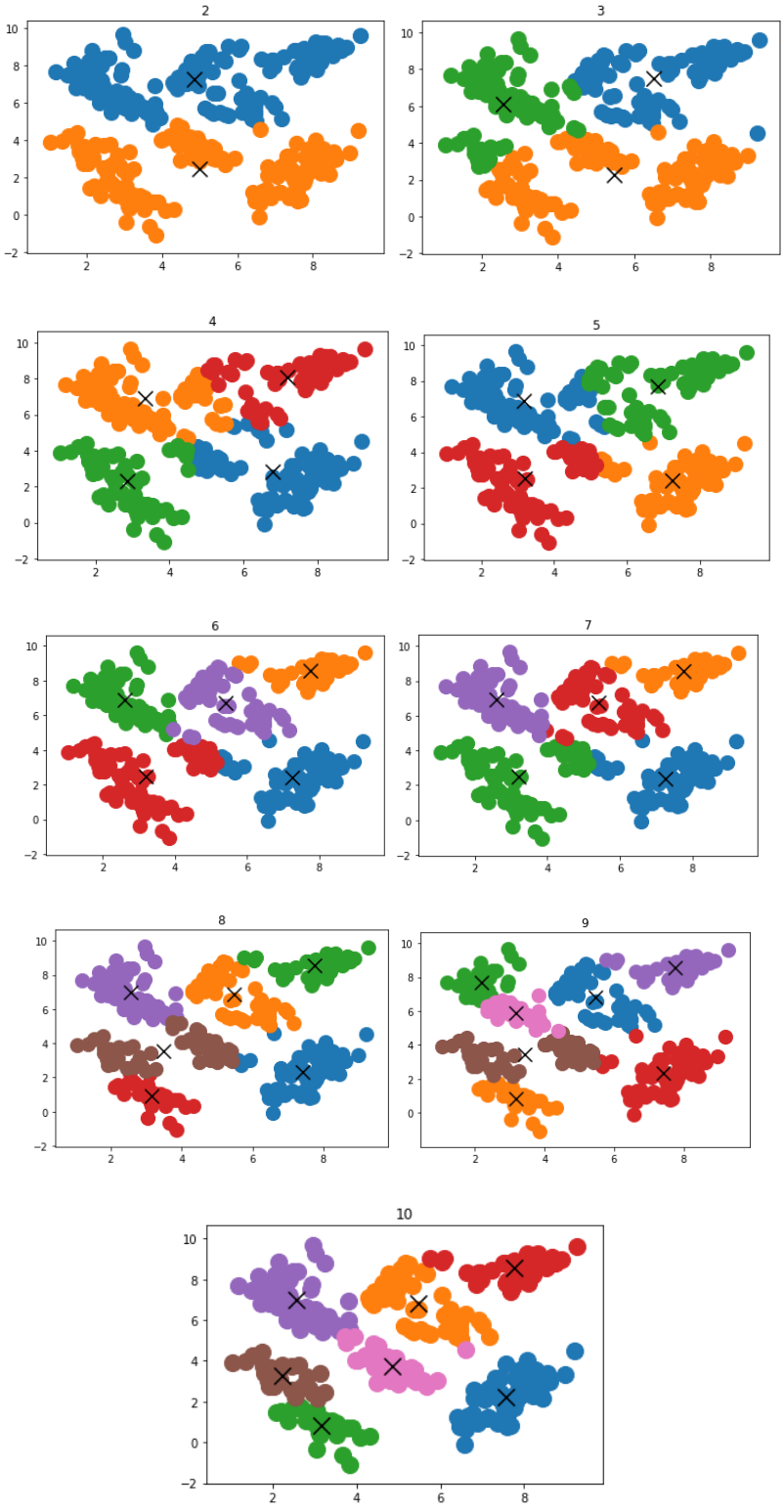


Figure 3 All the images above show the scatter plot with each classification being highlighted with different color and the centroid values given with 'x' for iteration 1 strategy 2

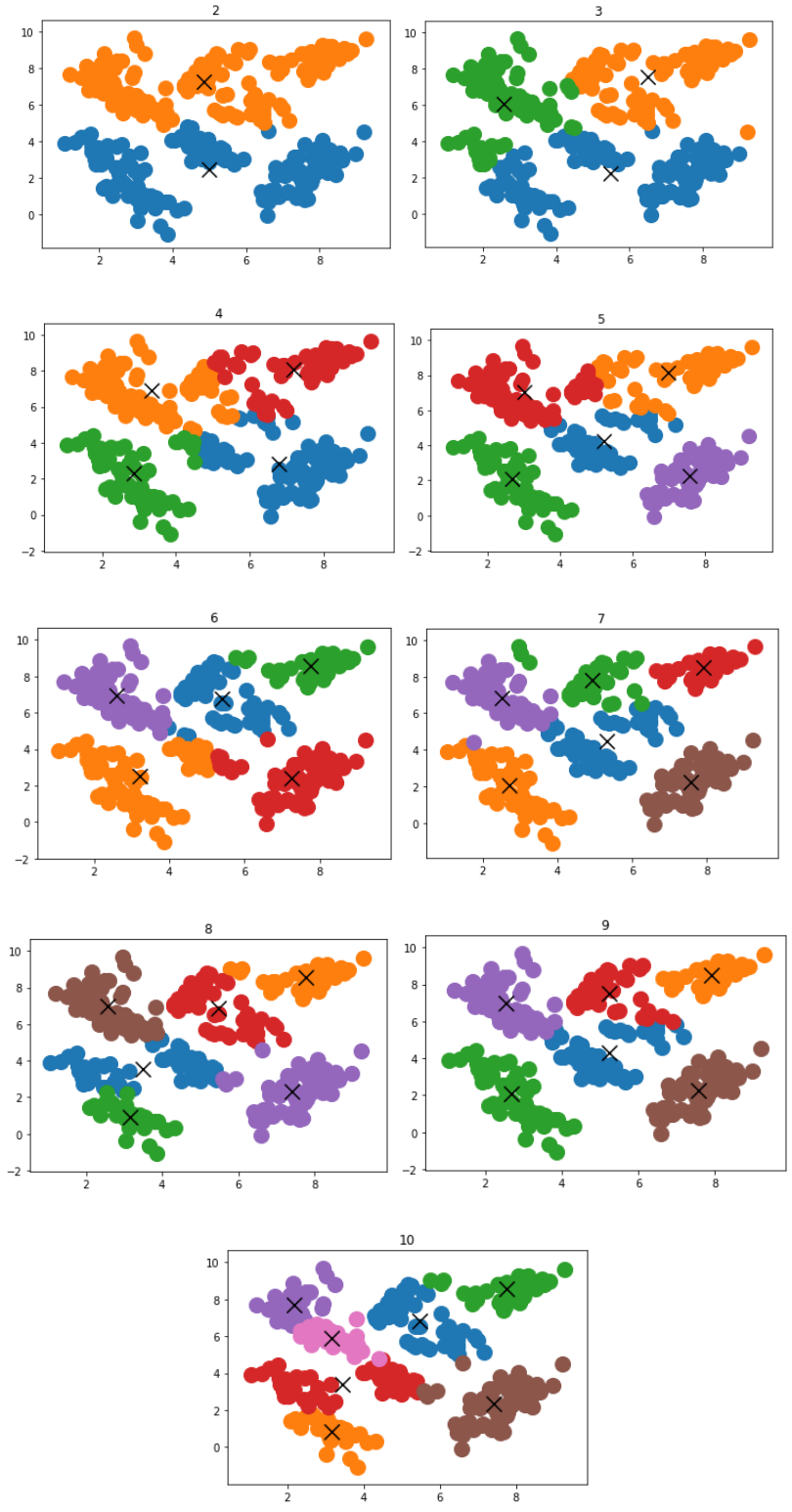


Figure 4 All the images above show the scatter plot with each classification being highlighted with different color and the centroid values given with 'x' for iteration 2 strategy 2

5. Objective function

Referring to the course notes: When clustering the samples into k clusters/sets Di, with respective center/mean vectors $\mu_1, \mu_2, \ldots \mu_k$, the objective function is defined as:

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

The calculation of the objective function was as follows(same for both strategy 1 and 2):

```
def computeObjectiveFunc(data, centroids):
    sum = 0
    for centroidNumber in range(0,len(centroids)):
        for value in data[centroidNumber]:
            sum += np.square(value[0]-centroids[centroidNumber][0])\
            + np.square(value[1]-centroids[centroidNumber][1])
    return sum
```

Following is the graphs for the objective functions obtained vs the k value, the graphs were by writing the following code:

```
plt.plot(range(2,len(objectiveFunctionValue)+2),objectiveFunctionValue)
plt.scatter(range(2,len(objectiveFunctionValue)+2),objectiveFunctionValue,color='k')
plt.show()
```

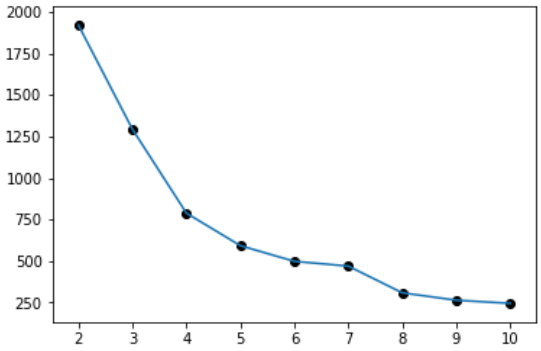


Figure 5 Strategy 1 Iteration1

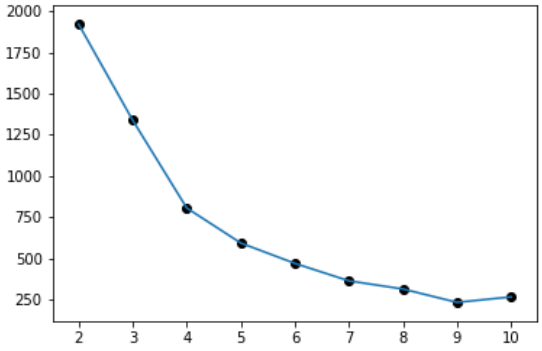


Figure 6 Strategy 1 Iteration2

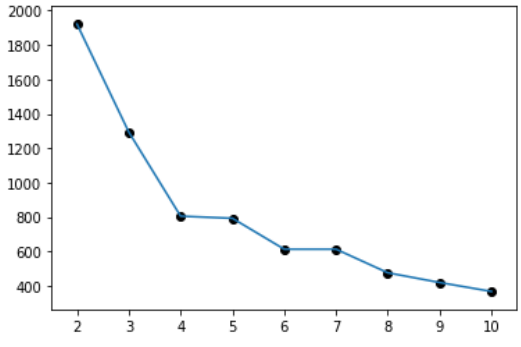


Figure 7 Strategy 2 Iteration 1

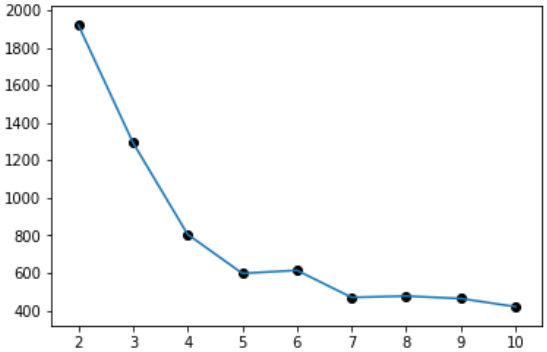


Figure 8 Strategy 2 Iteration 2

6. Resources:

1. https://canvas.asu.edu/courses/31489/assignments/686560?module_item_id=1905559
2. https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.scatter.html
3. https://pynative.com/python-random-choice/