EE2211 Tutorial 11

Question 1: The K-means clustering method uses the target labels for calculating the distances from the cluster centroids for clustering.

a) True
b) False

Ans: b) because target labels are not available in clustering.

Question 2: The fuzzy C-means algorithm groups the data items such that an item can exist in multiple clusters.
a) True
b) False

Ans: a).

Question 3: How can you prevent a clustering algorithm from getting stuck in bad local optima?

a) Set the same seed value for each run
b) Use the bottom ranked samples for initialization
c) Use the top ranked samples for initialization
d) All the above
e) None of the above

Ans: e).

Question 4: Consider the following data points: $\boldsymbol{x} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, $\boldsymbol{y} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ and $\boldsymbol{z} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$. The k-means algorithm is initialized with centers at $\boldsymbol{x}$ and $\boldsymbol{y}$. Upon convergence, the two centres will be at

a) $\boldsymbol{x}$ and $\boldsymbol{z}$
b) $\boldsymbol{x}$ and $\boldsymbol{y}$
c) $\boldsymbol{y}$ and the midpoint of $\boldsymbol{y}$ and $\boldsymbol{z}$
d) $\boldsymbol{z}$ and the midpoint of $\boldsymbol{x}$ and $\boldsymbol{y}$
e) None of the above

Ans: e). The converged centers should be $\boldsymbol{x}$ and the midpoint of $\boldsymbol{y}$ and $\boldsymbol{z}$.

Matlab codes

```
X = [1,1; 0 1; 0 0];
[idx,C] = kmeans(X,2,'start',X([1,2],:))
```

Question 5: Consider the following 8 data points: $\boldsymbol{x}_1 = \begin{bmatrix}0\\0\end{bmatrix}$, $\boldsymbol{x}_2 = \begin{bmatrix}0\\1\end{bmatrix}$, $\boldsymbol{x}_3 = \begin{bmatrix}1\\1\end{bmatrix}$, $\boldsymbol{x}_4 = \begin{bmatrix}1\\0\end{bmatrix}$, $\boldsymbol{x}_5 = \begin{bmatrix}3\\0\end{bmatrix}$, $\boldsymbol{x}_6 = \begin{bmatrix}3\\1\end{bmatrix}$, $\boldsymbol{x}_7 = \begin{bmatrix}4\\0\end{bmatrix}$ and $\boldsymbol{x}_8 = \begin{bmatrix}4\\1\end{bmatrix}$. The k-means algorithm is initialized with centers at $\begin{bmatrix}0\\0\end{bmatrix}$ and $\begin{bmatrix}3\\0\end{bmatrix}$. The first center after convergence is $\boldsymbol{c}_1 = \begin{bmatrix}0.5\\0.5\end{bmatrix}$. The second centre after convergence is $\boldsymbol{c}_2 = \begin{bmatrix}blank1\\blank2\end{bmatrix}$ (up to 1 decimal place)

Answer: *blank1 = 3.5, blank2 = 0.5*

Matlab codes

```
X = [0,0; 0,1; 1,1; 1,0; 3,0; 3,1; 4,0; 4,1];
[idx,C] = kmeans(X,2,'start',X([1,5],:))
```

(K-means Implementation on 2D data)

Question 6:

Generate three clusters of data using the following codes.

```
## Import necessary libraries
import random as rd
import numpy as np # linear algebra
from matplotlib import pyplot as plt
## Generate data
## Set three centers, the model should predict similar results
center_1 = np.array([2,2])
center_2 = np.array([4,4])
center_3 = np.array([6,1])
## Generate random data and center it to the three centers
data_1 = np.random.randn(200, 2) + center_1
data_2 = np.random.randn(200,2) + center_2
data_3 = np.random.randn(200,2) + center_3
data = np.concatenate((data_1, data_2, data_3), axis = 0)
plt.scatter(data[:,0], data[:,1], s=7)
```

(i) Implement the Naïve K-means (the basic/standard algorithm shown in lecture) clustering algorithm to find the 3 cluster centroids. Classify the data based on the three centroids found and illustrate the results using a plot (e.g., mark the 3 clusters of data points using different colours).

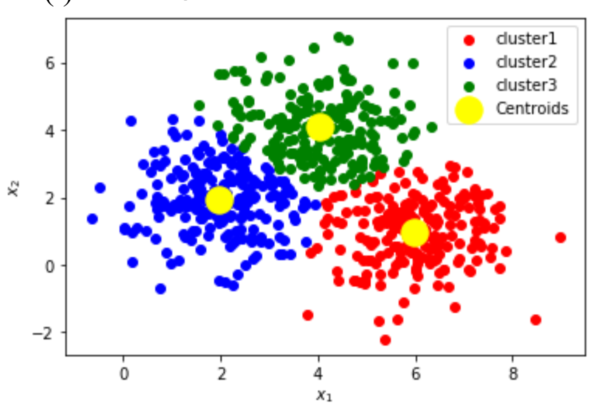
(ii) Change the number of clusters K to 5 and classify the data points again with a plot illustration.

Answer:

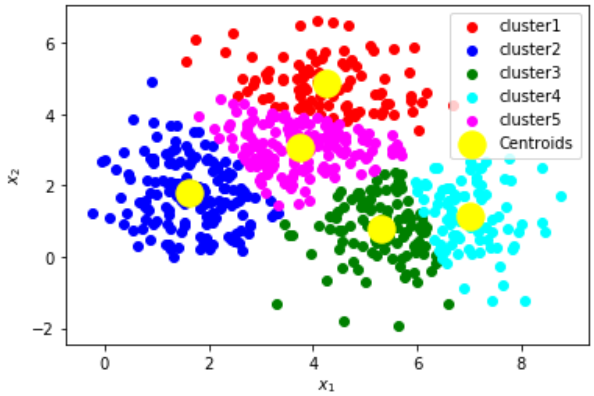
```
##Ref: https://www.kaggle.com/andyxie/k-means-clustering-implementation-in-python
##Ref: https://medium.com/machine-learning-algorithms-from-scratch/k-means-clustering-from-
##scratch-in-python-1675d38eee42
X = data
m=X.shape[0] #number of training examples
d=X.shape[1] #number of features. Here d=2
n_iter=100
##
K=3 # number of clusters
##
##Step 1: Initialize the centroids randomly from the data points:
Centroids=np.array([]).reshape(d,0)
for i in range(K):
   rand=rd.randint(0,m-1) # randomly pick a number from 0 to m-1
   Centroids=np.c_[Centroids,X[rand]] #concatenation along the second axis
Output={}
##Repeat step 2 till n_iter/convergence is achieved.
for i in range(n_iter):
```

```
    #Step 2.a: For each training example compute the euclidian distance from the centroid and assign
the cluster based on the minimal distance
    EuclidianDistance=np.array([]).reshape(m,0)
    for k in range(K):
        # Compute the distance between the kth centroid and every data point
        tempDist=np.sum((X-Centroids[:,k])**2,axis=1)
        # stack the K sets of Euclid distance in K columns
        EuclidianDistance=np.c_[EuclidianDistance,tempDist]
    # Center indicator: locate the column (argmin) that has the minimum distance
    C=np.argmin(EuclidianDistance,axis=1)+1
    #Step 2.b: We need to regroup the data points based on the cluster index C and store in the
Output dictionary and also compute the mean of separated clusters and assign it as new centroids.
Y is a temporary dictionary which stores the solution for one particular iteration.
    Y={}
    for k in range(K):
        # each Y[k]: array([], shape=(2, 0), dtype=float64)
        Y[k+1]=np.array([]).reshape(d,0)
    for i in range(m):
        # Indicate and collect data X according to the Center indicator
        Y[C[i]]=np.c_[Y[C[i]],X[i]] #np.shape(Y[k])=(2, number of points nearest to kth center)
    for k in range(K):
        Y[k+1]=Y[k+1].T # transpose the row-wise data to column-wise

    # Compute new centroids
    for k in range(K):
        Centroids[:,k]=np.mean(Y[k+1],axis=0)

Output=Y
## Plot data
plt.scatter(X[:,0],X[:,1],c='black',label='unclustered data')
plt.xlabel('$x_1$')
plt.ylabel('$x_2$')
plt.legend()
plt.title('Plot of data points')
plt.show()
## plot clusters
color=['red','blue','green','cyan','magenta']
labels=['cluster1','cluster2','cluster3','cluster4','cluster5']
for k in range(K):
    plt.scatter(Output[k+1][:,0],Output[k+1][:,1],c=color[k],label=labels[k])
plt.scatter(Centroids[0,:],Centroids[1,:],s=300,c='yellow',label='Centroids')
plt.xlabel('$x_1$')
plt.ylabel('$x_2$')
plt.legend()
plt.show()
```

(i) K=3



(ii) K=5



(K-means Classification of iris data, 4D input features)

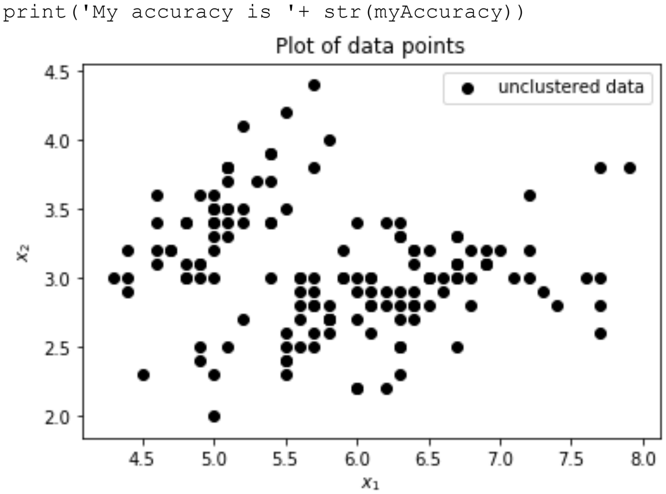
Question 7:

Load the iris data "`from sklearn.datasets import load_iris`". Assume that the class labels are not given. Use the Naïve K-means clustering algorithm to group all the data based on K=3. How accurate is the result of clustering comparing with the known labels?

Answer:

```
##Ref: https://www.kaggle.com/andyxie/k-means-clustering-implementation-in-python
##Ref: https://medium.com/machine-learning-algorithms-from-scratch/k-means-clustering-from-
##scratch-in-python-1675d38eee42
## Import libraries
import random
import numpy as np # linear algebra
from matplotlib import pyplot as plt
from sklearn.datasets import load_iris
iris_dataset = load_iris()
X = np.array(iris_dataset['data'])
y = np.array(iris_dataset['target'])
plt.scatter(X[:,0], X[:,1], s=7)
m=X.shape[0] #number of training examples
d=X.shape[1] #number of features. Here d=4
n_iter=100
##

K=3 # number of clusters
##

# set the random seed
random.seed(0)

#Step 1: Randomly initialize over all points in the dataset by sampling without replacement
Centroids=np.array([]).reshape(d,0)
ind = random.sample(range(X.shape[0]),K)
Centroids = X[ind,:].T

##Step 2: For each training example compute the euclidian distance from the centroid and assign
the cluster based on the minimal distance
Output={}

##Repeat step 2 till convergence is achieved.
for i in range(n_iter):
    #step 2.a
     EuclidianDistance=np.array([]).reshape(m,0)
     for k in range(K):
         tempDist=np.sum((X-Centroids[:,k])**2,axis=1)
         EuclidianDistance=np.c_[EuclidianDistance,tempDist]
     C=np.argmin(EuclidianDistance,axis=1)+1
    #step 2.b
     Y={}
```

```
    Idx1=[]
    Idx2=[]
    Idx3=[]
    for k in range(K):
       Y[k+1]=np.array([]).reshape(d,0)
    for i in range(m):
       # put X to each cluster vector Y
       Y[C[i]]=np.c_[Y[C[i]],X[i]]
       # collect indices of each clustered group
       if C[i]==1: Idx1 += [i]
       if C[i]==2: Idx2 += [i]
       if C[i]==3: Idx3 += [i]

    for k in range(K):
       Centroids[:,k]=np.mean(Y[k+1],axis=1).T

Output=Y

## Plot data
plt.scatter(X[:,0],X[:,1],c='black',label='unclustered data')
plt.xlabel('$x_1$')
plt.ylabel('$x_2$')
plt.legend()
plt.title('Plot of data points')
plt.show()

## Tutorial's way of computing classification accuracy
Class1_est = y[Idx1]
Class2_est = y[Idx2]
Class3_est = y[Idx3]
Cls1_correct = 0

# Find best permutation to compute clustering accuracy
Idx_List = [Idx1, Idx2, Idx3]
import itertools
list_perms = list(itertools.permutations([1, 2, 3]))
correct = np.zeros((len(list_perms),1))
i=0
for p in list_perms:
   for c in range(0,3):
      correct[i] += len(np.intersect1d(np.array(Idx_List[p[c]-1]), np.where(y==c)))
   i=i+1

myAccuracy = np.max(correct)/len(y)
print('My accuracy is '+ str(myAccuracy))
```



My accuracy is 0.8866666666666667