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 of the above
e) None of the above
Ans: e).

Question 4

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

a) x and z
b) x and y

c) y and the midpoint of y and z
d) z and the midpoint of x and y
e) None of the above

Ans: e). The converged centers should be x and the midpoint of y and z.

```python
import numpy as np

# Data points
x = np.array([1, 1])
y = np.array([0, 1])
z = np.array([0, 0])

data_points = np.array([x, y, z])

# Initial centers
centers = np.array([x, y])


def k_means(data_points, centers, n_clusters, max_iterations=100,
    tol=1e-4):
    for _ in range(max_iterations):
        # Assign each data point to the closest centroid
        labels = np.argmin(np.linalg.norm(data_points[:, np.newaxis
    ] - centers, axis=2), axis=1)

        # Update centroids to be the mean of the data points
    assigned to them
        new_centers = np.zeros((n_clusters, data_points.shape[1]))

         # End if centroids no longer change
        for i in range(n_clusters):
            new_centers[i] = data_points[labels == i].mean(axis=0)
        if np.linalg.norm(new_centers - centers) < tol:
            break
        centers = new_centers
    return centers, labels

centers, labels = k_means(data_points, centers, n_clusters=2)
print("Converged centers:", centers)
```

Question 5

Consider the following 8 data points: $x_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, $x_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, $x_3 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, $x_4 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, $x_5 = \begin{bmatrix} 3 \\ 0 \end{bmatrix}$, $x_6 = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$, $x_7 = \begin{bmatrix} 4 \\ 0 \end{bmatrix}$, and $x_8 = \begin{bmatrix} 4 \\ 1 \end{bmatrix}$. The k-means algorithm is initialized with centers at $c_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ and $c_2 = \begin{bmatrix} 3 \\ 0 \end{bmatrix}$. The first center after convergence is $c_1 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$. The second centre after convergence is $c_2 = \begin{bmatrix} blank1 \\ blank2 \end{bmatrix}$.

Answer: blank1 = 3.5, blank2 = 0.5.

```python
import numpy as np
```

```python

# Data points
x1 = np.array([0, 0])
x2 = np.array([0, 1])
x3 = np.array([1, 1])
x4 = np.array([1, 0])
x5 = np.array([3, 0])
x6 = np.array([3, 1])
x7 = np.array([4, 0])
x8 = np.array([4, 1])

data_points = np.array([x1, x2, x3, x4, x5, x6, x7, x8])

# Initial centers
c1_init = np.array([0, 0])
c2_init = np.array([3, 0])

centers = np.array([c1_init, c2_init])

def k_means(data_points, centers, n_clusters, max_iterations=100,
    tol=1e-4):
    for _ in range(max_iterations):
        # Assign each data point to the closest centroid
        labels = np.argmin(np.linalg.norm(data_points[:, np.newaxis
    ] - centers, axis=2), axis=1)

        # Update centroids to be the mean of the data points
    assigned to them
        new_centers = np.zeros((n_clusters, data_points.shape[1]))

         # End if centroids no longer change
        for i in range(n_clusters):
            new_centers[i] = data_points[labels == i].mean(axis=0)
        if np.linalg.norm(new_centers - centers) < tol:
            break
        centers = new_centers
    return centers, labels


centers, labels = k_means(data_points, centers, n_clusters=2)
print("Converged centers:", centers)
```

Question 6

Generate three clusters of data using the following codes.

```python
# 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 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.

```python
# 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)

# initialize cluster centers
k = 3
centers = data[np.random.choice(len(data), k, replace=False)]

def k_means(data_points, centers, n_clusters, max_iterations=100, tol=1e-4):
    for _ in range(max_iterations):
        # Assign each data point to the closest centroid
        labels = np.argmin(np.linalg.norm(data_points[:, np.newaxis] - centers, axis=2), axis=1)
```

```python

        # Update centroids to be the mean of the data points assigned to them
        new_centers = np.zeros((n_clusters, data_points.shape[1]))

         # End if centroids no longer change
        for i in range(n_clusters):
            new_centers[i] = data_points[labels == i].mean(axis=0)
        if np.linalg.norm(new_centers - centers) < tol:
            break
        centers = new_centers
    return centers, labels

centers, labels = k_means(data, centers, n_clusters=k)
print("Converged centers:", centers)
plt.title('Clustering Results')
plt.scatter(data[:, 0], data[:, 1], c=labels, cmap='viridis', alpha=0.5)
plt.scatter(centers[:, 0], centers[:, 1], marker='*', s=200, c='k')
plt.show()
```

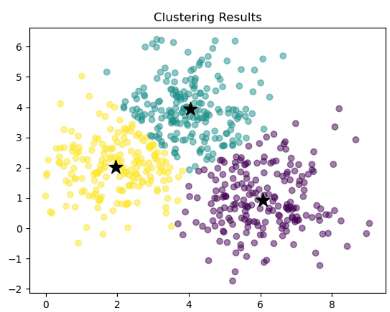

Figure 1: K=3

Figure 2: K=5

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?

```python
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
from sklearn.metrics import accuracy_score
import numpy as np

# load the iris dataset
iris = load_iris()

# get the data and the true labels
data = iris.data
y_true = iris.target

# initialize the KMeans centers with K=3
k = 3
centers = data[np.random.choice(len(data), k, replace=False)]

def k_means(data_points, centers, n_clusters, max_iterations=1000, tol=1e-6):
    for _ in range(max_iterations):
        # Assign each data point to the closest centroid
        labels = np.argmin(np.linalg.norm(data_points[:, np.newaxis] - centers, axis=2), axis=1)

        # Update centroids to be the mean of the data points assigned to them
        new_centers = np.zeros((n_clusters, data_points.shape[1]))

         # End if centroids no longer change
        for i in range(n_clusters):
            new_centers[i] = data_points[labels == i].mean(axis=0)
        if np.linalg.norm(new_centers - centers) < tol:
            break
        centers = new_centers
    return centers, labels

centers, y_pred = k_means(data, centers, n_clusters=k)
# create a mask that selects elements where the value is 0, 1, 2
mask_0 = (y_pred == 0)
mask_1 = (y_pred == 1)
mask_2 = (y_pred == 2)

y_pred0 = y_pred.copy()
y_pred0[mask_0] = 0
y_pred0[mask_1] = 1
y_pred0[mask_2] = 2

y_pred1 = y_pred.copy()
y_pred1[mask_0] = 0
y_pred1[mask_1] = 2
y_pred1[mask_2] = 1
```

```python

y_pred2 = y_pred.copy()
y_pred2[mask_0] = 1
y_pred2[mask_1] = 0
y_pred2[mask_2] = 2


y_pred3 = y_pred.copy()
y_pred3[mask_0] = 1
y_pred3[mask_1] = 2
y_pred3[mask_2] = 0

y_pred4 = y_pred.copy()
y_pred4[mask_0] = 2
y_pred4[mask_1] = 0
y_pred4[mask_2] = 1

y_pred5 = y_pred.copy()
y_pred5[mask_0] = 2
y_pred5[mask_1] = 1
y_pred5[mask_2] = 0

# calculate the accuracy of the clustering
accuracy = 0.0
for pred in [y_pred0, y_pred1, y_pred2, y_pred3, y_pred4, y_pred5]:
    accuracy  = max([accuracy_score(y_true, pred), accuracy])

print("Accuracy of clustering: {:.2f}".format(accuracy))
```