

## 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  $z$ .

```

1 import numpy as np
2
3 # Data points
4 x = np.array([1, 1])
5 y = np.array([0, 1])
6 z = np.array([0, 0])
7
8 data_points = np.array([x, y, z])
9
10 # Initial centers
11 centers = np.array([x, y])
12
13
14 def k_means(data_points, centers, n_clusters, max_iterations=100,
15             tol=1e-4):
16     for _ in range(max_iterations):
17         # Assign each data point to the closest centroid
18         labels = np.argmin(np.linalg.norm(data_points[:, np.newaxis]
19         ] - centers, axis=2), axis=1)
20
21         # Update centroids to be the mean of the data points
22         # assigned to them
23         new_centers = np.zeros((n_clusters, data_points.shape[1]))
24
25         # End if centroids no longer change
26         for i in range(n_clusters):
27             new_centers[i] = data_points[labels == i].mean(axis=0)
28             if np.linalg.norm(new_centers - centers) < tol:
29                 break
30         centers = new_centers
31     return centers, labels
32
33 centers, labels = k_means(data_points, centers, n_clusters=2)
34 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  $c_2 = \begin{bmatrix} \text{blank1} \\ \text{blank2} \end{bmatrix}$ .

Answer: blank1 = 3.5, blank2 = 0.5.

```

1 import numpy as np

```

```

2
3 # Data points
4 x1 = np.array([0, 0])
5 x2 = np.array([0, 1])
6 x3 = np.array([1, 1])
7 x4 = np.array([1, 0])
8 x5 = np.array([3, 0])
9 x6 = np.array([3, 1])
10 x7 = np.array([4, 0])
11 x8 = np.array([4, 1])
12
13 data_points = np.array([x1, x2, x3, x4, x5, x6, x7, x8])
14
15 # Initial centers
16 c1_init = np.array([0, 0])
17 c2_init = np.array([3, 0])
18
19 centers = np.array([c1_init, c2_init])
20
21 def k_means(data_points, centers, n_clusters, max_iterations=100,
22             tol=1e-4):
23     for _ in range(max_iterations):
24         # Assign each data point to the closest centroid
25         labels = np.argmin(np.linalg.norm(data_points[:, np.newaxis]
26         ] - centers, axis=2), axis=1)
27
28         # Update centroids to be the mean of the data points
29         # assigned to them
30         new_centers = np.zeros((n_clusters, data_points.shape[1]))
31
32         # End if centroids no longer change
33         for i in range(n_clusters):
34             new_centers[i] = data_points[labels == i].mean(axis=0)
35         if np.linalg.norm(new_centers - centers) < tol:
36             break
37         centers = new_centers
38     return centers, labels
39
40 centers, labels = k_means(data_points, centers, n_clusters=2)
41 print("Converged centers:", centers)

```

## 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 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.

```
1 # Import necessary libraries
2
3 import random as rd
4 import numpy as np # linear algebra
5 from matplotlib import pyplot as plt
6
7 # Generate data
8
9 # Set three centers, the model should predict similar results
10
11 center_1 = np.array([2,2])
12 center_2 = np.array([4,4])
13 center_3 = np.array([6,1])
14
15 # Generate random data and center it to the three centers
16
17 data_1 = np.random.randn(200, 2) + center_1
18 data_2 = np.random.randn(200,2) + center_2
19 data_3 = np.random.randn(200,2) + center_3
20 data = np.concatenate((data_1, data_2, data_3), axis = 0)
21
22 # initialize cluster centers
23 k = 3
24 centers = data[np.random.choice(len(data), k, replace=False)]
25
26 def k_means(data_points, centers, n_clusters, max_iterations=100,
27             tol=1e-4):
28     for _ in range(max_iterations):
29         # Assign each data point to the closest centroid
30         labels = np.argmax(np.linalg.norm(data_points[:, np.newaxis]
31                                           ] - centers, axis=2), axis=1)
```

```

30
31     # Update centroids to be the mean of the data points
    assigned to them
32     new_centers = np.zeros((n_clusters, data_points.shape[1]))
33
34     # End if centroids no longer change
35     for i in range(n_clusters):
36         new_centers[i] = data_points[labels == i].mean(axis=0)
37     if np.linalg.norm(new_centers - centers) < tol:
38         break
39     centers = new_centers
40     return centers, labels
41
42 centers, labels = k_means(data, centers, n_clusters=k)
43 print("Converged centers:", centers)
44 plt.title('Clustering Results')
45 plt.scatter(data[:, 0], data[:, 1], c=labels, cmap='viridis', alpha
    =0.5)
46 plt.scatter(centers[:, 0], centers[:, 1], marker='*', s=200, c='k')
47 plt.show()

```

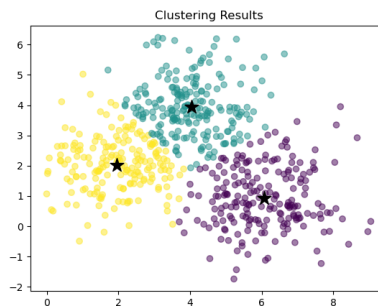


Figure 1: K=3

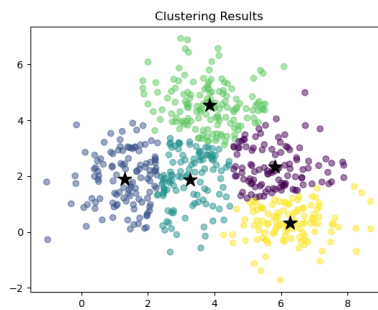


Figure 2: K=5

## 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?

```
1 from sklearn.datasets import load_iris
2 from sklearn.cluster import KMeans
3 from sklearn.metrics import accuracy_score
4 import numpy as np
5
6 # load the iris dataset
7 iris = load_iris()
8
9 # get the data and the true labels
10 data = iris.data
11 y_true = iris.target
12
13 # initialize the KMeans centers with K=3
14 k = 3
15 centers = data[np.random.choice(len(data), k, replace=False)]
16
17 def k_means(data_points, centers, n_clusters, max_iterations=1000,
18             tol=1e-6):
19     for _ in range(max_iterations):
20         # Assign each data point to the closest centroid
21         labels = np.argmin(np.linalg.norm(data_points[:, np.newaxis]
22     ] - centers, axis=2), axis=1)
23
24         # Update centroids to be the mean of the data points
25         # assigned to them
26         new_centers = np.zeros((n_clusters, data_points.shape[1]))
27
28         # End if centroids no longer change
29         for i in range(n_clusters):
30             new_centers[i] = data_points[labels == i].mean(axis=0)
31             if np.linalg.norm(new_centers - centers) < tol:
32                 break
33         centers = new_centers
34     return centers, labels
35
36 centers, y_pred = k_means(data, centers, n_clusters=k)
37
38 # create a mask that selects elements where the value is 0, 1, 2
39 mask_0 = (y_pred == 0)
40 mask_1 = (y_pred == 1)
41 mask_2 = (y_pred == 2)
42
43 y_pred0 = y_pred.copy()
44 y_pred0[mask_0] = 0
45 y_pred0[mask_1] = 1
46 y_pred0[mask_2] = 2
47
48 y_pred1 = y_pred.copy()
49 y_pred1[mask_0] = 0
50 y_pred1[mask_1] = 2
51 y_pred1[mask_2] = 1
```

```

48
49 y_pred2 = y_pred.copy()
50 y_pred2[mask_0] = 1
51 y_pred2[mask_1] = 0
52 y_pred2[mask_2] = 2
53
54
55 y_pred3 = y_pred.copy()
56 y_pred3[mask_0] = 1
57 y_pred3[mask_1] = 2
58 y_pred3[mask_2] = 0
59
60 y_pred4 = y_pred.copy()
61 y_pred4[mask_0] = 2
62 y_pred4[mask_1] = 0
63 y_pred4[mask_2] = 1
64
65 y_pred5 = y_pred.copy()
66 y_pred5[mask_0] = 2
67 y_pred5[mask_1] = 1
68 y_pred5[mask_2] = 0
69
70 # calculate the accuracy of the clustering
71 accuracy = 0.0
72 for pred in [y_pred0, y_pred1, y_pred2, y_pred3, y_pred4, y_pred5]:
73     accuracy = max([accuracy_score(y_true, pred), accuracy])
74
75 print("Accuracy of clustering: {:.2f}".format(accuracy))

```