HW 1

January 19, 2024

Pupipat Singkhorn

1 Homework 1 Clustering and Regression

1.1 Metrics

Model A	Predicted dog	Predicted cat
Actual dog	30	20
Actual cat	10	40

1.1.1 T1.

What is the accuracy of Model A?

[ ]:
```
tn = 30
fn = 10
fp = 20
tp = 40

accuracy = ( tp+tn ) / ( tp+tn+fp+fn )

print(f'Accuracy = {accuracy}')
```

```
Accuracy = 0.7
```

1.1.2 T2.

Consider cats as 'class 1' (positive) and dogs as 'class 0' (negative), calculate the precision, recall, and F1.

Model A	Predicted dog (-)	Predicted cat (+)
Actual dog	TN	FP
Actual cat	FN	TP

[ ]:
```
accuracy = ( tp+tn ) / ( tp+tn+fp+fn )
print(f'Accuracy = {accuracy}')
```

```
Accuracy = 0.7
```

[ ]:
```
tn = 30
fn = 10
fp = 20
tp = 40

precision = tp / (tp+fp)
recall = tp / (tp+fn)
F1 = 2*tp / ( 2*tp + fp + fn )

print(f'Precision = {precision}')
print(f'Recall = {recall}')
print(f'F1 = {F1}')
```

```
Precision = 0.6666666666666666
Recall = 0.8
F1 = 0.7272727272727273
```

1.1.3 T3.

Consider class cat as 'class 0' and class dog as 'class 1', calculate the precision, recall, and F1.

Model A	Predicted dog (+)	Predicted cat (-)
Actual dog	TP	FN
Actual cat	FP	TN

[ ]:
```
accuracy = ( tp+tn ) / ( tp+tn+fp+fn )

print(f'Accuracy = {accuracy}')
```

```
Accuracy = 0.7
```

[ ]:
```
tp = 30
tn = 40
fp = 10
fn = 20

precision = tp / (tp+fp)
recall = tp / (tp+fn)
F1 = 2*tp / ( 2*tp + fp + fn )

print(f'Precision = {precision}')
print(f'Recall = {recall}')
print(f'F1 = {F1}')
```

```
Precision = 0.75
Recall = 0.6
F1 = 0.6666666666666666
```

1.1.4 T4.

Now consider a lopsided population where there are 80% cats.
What is the accuracy of Model A?
Using dog as the positive class, what is the precision, recall, and F1?
Explain how and why these numbers change (or does not change) from the previous questions.

Model A	Predicted dog (+)	Predicted cat (-)
Actual dog	TP	FN
Actual cat	FP	TN

Model A	Predicted dog	Predicted cat
Actual dog	30	20
Actual cat	10	40

$Total\ Population = x$
$Actual\ dog = 0.2x$ (20% dogs)
$Actual\ cat = 0.8x$ (80% cats)

$Actual\ dog\ Predicted\ dog(TP) = 0.2x * \frac{30}{(30+20)} = 0.12x$
$Actual\ dog\ Predicted\ cat(FN) = 0.2x * \frac{20}{(30+20)} = 0.08x$
$Actual\ cat\ Predicted\ dog(FP) = 0.8x * \frac{10}{(10+40)} = 0.16x$
$Actual\ cat\ Predicted\ cat(TN) = 0.8x * \frac{40}{(10+40)} = 0.64x$

Model A	Predicted dog	Predicted cat
Actual dog	0.12x	0.08x
Actual cat	0.16x	0.64x

$$Accuracy = \frac{tp + tn}{tp + tn + fp + fn} = \frac{0.12x + 0.64x}{0.12x + 0.64x + 0.16x + 0.08x} = 0.76$$

Accuracy increase

$$precision = \frac{tp}{tp + fp} = \frac{0.12x}{0.12x + 0.16x} = 0.4285$$

precision decrease

$$recall = \frac{tp}{tp + fn} = \frac{0.12x}{0.12x + 0.08x} = 0.6$$

recall doesn't change

$$F1 = \frac{2}{1/recall + 1/precision} = \frac{2}{1/0.6 + 1/0.4285} = 0.5$$

F1 decrease

1.1.5 OT1.

Consider the equations for accuracy and F1. When will accuracy be equal, greater, or less than F1?

$$Accuracy = \frac{tp + tn}{tp + tn + fp + fn} = \frac{1}{1 + \frac{fp+fn}{tp+\mathbf{tn}}}$$

$$F1 = \frac{2tp}{2tp + fp + fn} = \frac{1}{1 + \frac{fp+fn}{tp+\mathbf{tp}}}$$

$$\therefore Accuracy = F1 \quad ; \; tn = tp$$
$$\therefore Accuracy > F1 \quad ; \; tn > tp$$
$$\therefore Accuracy < F1 \quad ; \; tn < tp$$

1.2 Hello Clustering

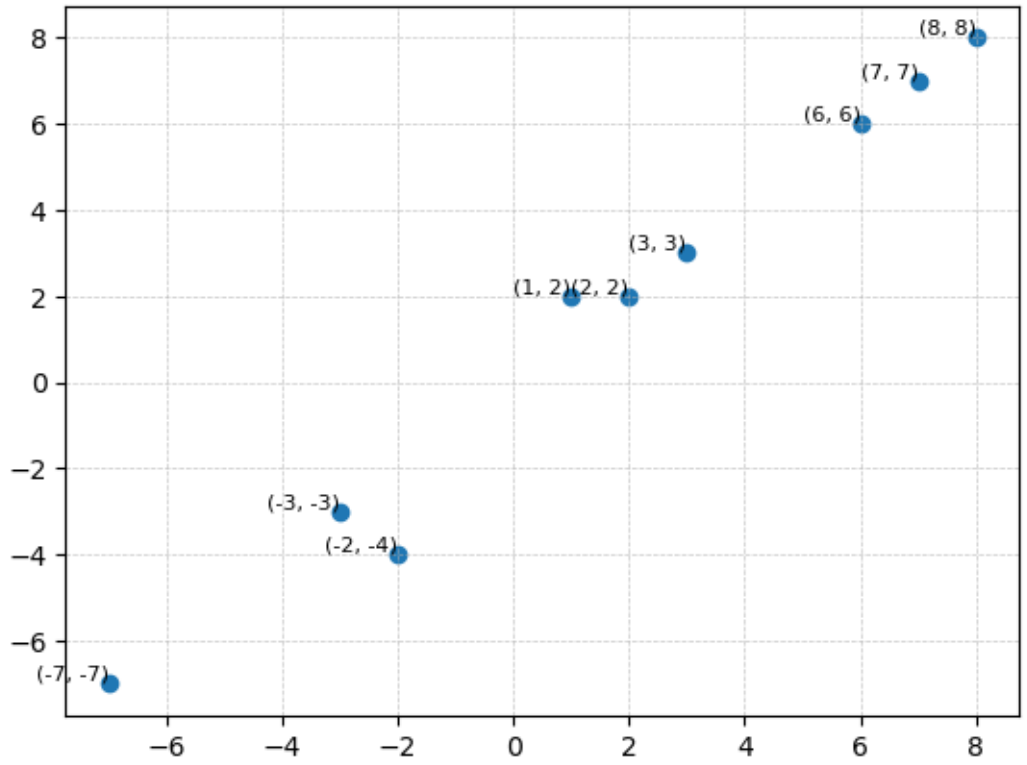
Recall from lecture that K-means has two main steps: the points assignment step, and the mean update step. After the initialization of the centroids, we assign each data point to a centroid. Then, each centroids are updated by re-estimating the means. Concretely, if we are given N data points, x1, x2, ..., xN , and we would like to form K clusters. We do the following; 1. Initialization: Pick K random data points as K centroid locations c1, c2, ..., cK. 2. Assign: For each data point k, find the closest centroid. Assign that data point to the centroid. The distance used is typically Euclidean distance. 3. Update: For each centroid, calculate the mean from the data points assigned to it. 4. Repeat: repeat step 2 and 3 until the centroids stop changing (convergence).

Given the following data points in x-y coordinates (2 dimensional)

x	y
1	2
3	3
2	2
8	8
6	6
7	7
-3	-3
-2	-4
-7	-7

[ ]:
```
import matplotlib.pyplot as plt
import numpy as np

data_x = np.array([1, 3, 2, 8, 6, 7, -3, -2, -7])
data_y = np.array([2, 3, 2, 8, 6, 7, -3, -4, -7])

# Scatter plot with detailed grid lines
```

```
plt.scatter(data_x, data_y)

# Customize grid lines
plt.grid(True, linestyle='--', linewidth=0.5, alpha=0.7)

# Add labels for each point
for i, txt in enumerate(zip(data_x, data_y)):
    plt.text(txt[0], txt[1], f'({txt[0]}, {txt[1]})', fontsize=8, ha='right',␣
 ↪va='bottom')

# Show the plot
plt.show()
```

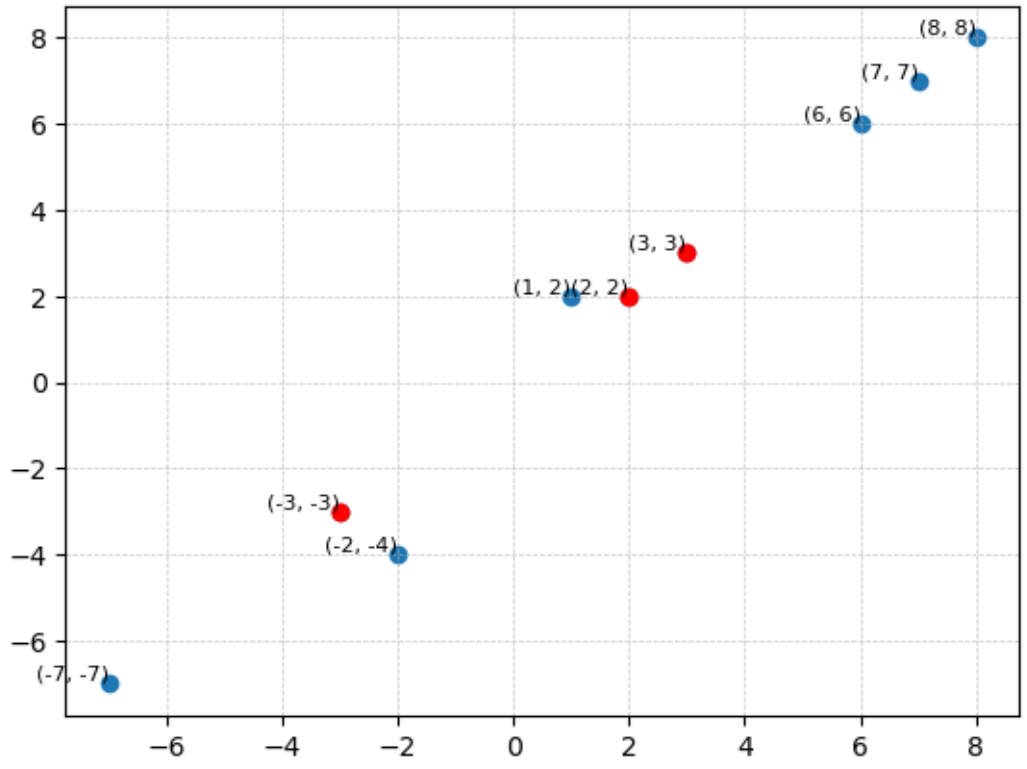


1.2.1 T5.

If the starting points are (3,3), (2,2), and (-3,-3). Describe each assign and update step. What are the points assigned? What are the updated centroids? You may do this calculation by hand or write a program to do it.

[ ]:
```
import matplotlib.pyplot as plt
import numpy as np
```

```
data_x = np.array([1, 3, 2, 8, 6, 7, -3, -2, -7])
data_y = np.array([2, 3, 2, 8, 6, 7, -3, -4, -7])

# Scatter plot with detailed grid lines
plt.scatter(data_x, data_y)

# Customize grid lines
plt.grid(True, linestyle='--', linewidth=0.5, alpha=0.7)

# Add labels for each point
for i, txt in enumerate(zip(data_x, data_y)):
    plt.text(txt[0], txt[1], f'({txt[0]}, {txt[1]})', fontsize=8, ha='right',␣
 ↪va='bottom')

# Add a red point at coordinates
plt.plot([3, 2, -3], [3, 2, -3], 'ro')

# Show the plot
plt.show()
```



K-mean clustering

1. Randomly init k centroids by picking from data points
2. Assign each data points to centroids
3. Update centroids for each cluster
4. Repeat 2-3 until centroids does not change

[ ]:
```
import numpy as np
import math
import matplotlib.pyplot as plt
```

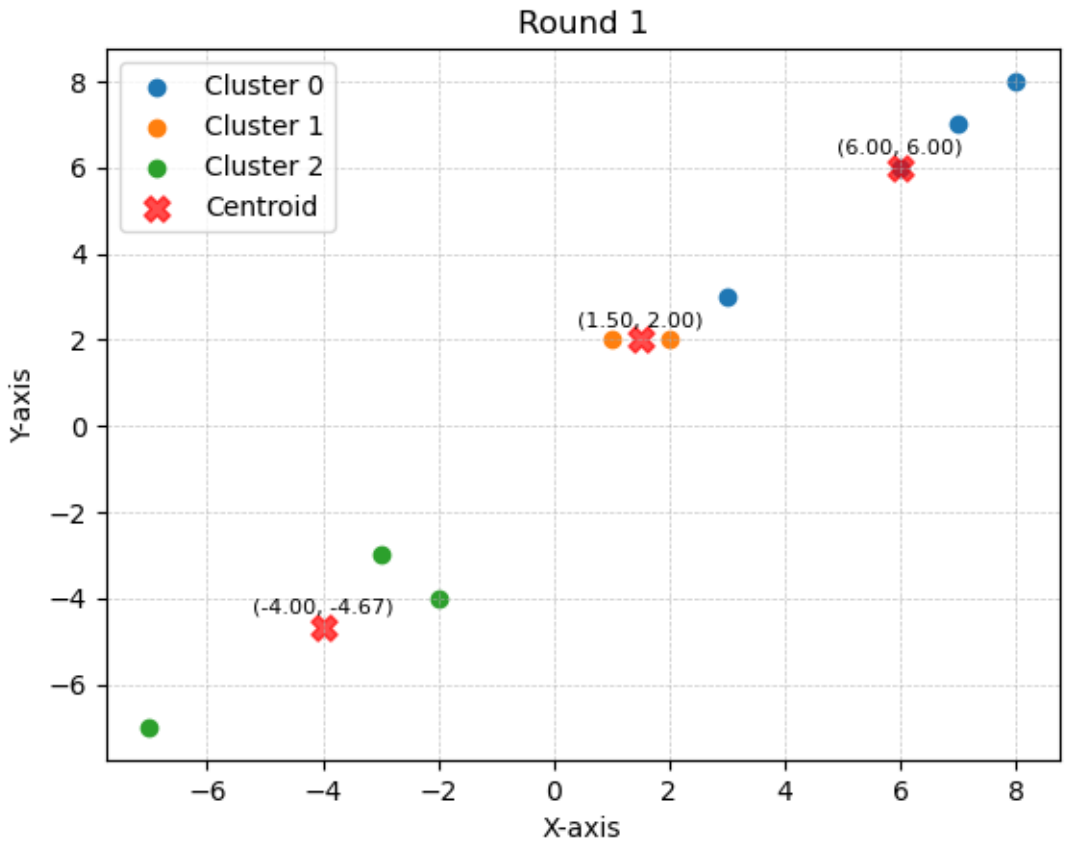
[ ]:
```
# function
def distance(point1, point2): # Euclidean Distance
    x1, y1 = point1
    x2, y2 = point2
    return math.sqrt((x2 - x1)**2 + (y2 - y1)**2) #float

def assign(data_points, cluster_centroid, cluster_points):
    for point in data_points: # assign each point to cluster
        distances = [distance(point, centroid) for centroid in cluster_centroid.
 ↪values()]
        min_distance_index = np.argmin(distances)
        cluster_points[min_distance_index].append(point)

def update_centroids(cluster_centroid, cluster_points, centroid_isUpdating):
    old_centroids = list(cluster_centroid.values())

    # calculate new_centroids
    new_centroids = []
    for idx, points in cluster_points.items():
        if points:  # Check if points list is not empty
            new_centroid_x, new_centroid_y = np.mean(points, axis=0)
            new_centroid = (new_centroid_x, new_centroid_y)
            new_centroids.append(new_centroid)
        else:
            new_centroids.append(cluster_centroid[idx]) # Use the existing␣
 ↪centroid if no points in the cluster

    # update new centroid
    for idx, centroid in enumerate(new_centroids):
        cluster_centroid[idx] = centroid

    if new_centroids == old_centroids:
        centroid_isUpdating = False
    return centroid_isUpdating #bool

def plot_clusters(data_points, cluster_centroid, cluster_points, title):
    for idx, points in cluster_points.items():
        cluster_x, cluster_y = zip(*points)
```

```
        plt.scatter(cluster_x, cluster_y, label=f'Cluster {idx}')

    # Annotate centroids with coordinates
    for idx, (x, y) in cluster_centroid.items():
        plt.annotate(f'({x:.2f}, {y:.2f})', (x, y), textcoords="offset points",␣
↪xytext=(0,5), ha='center', fontsize=8)

    centroid_x, centroid_y = zip(*cluster_centroid.values())
    plt.scatter(centroid_x, centroid_y, color='red', marker='X',s=90, alpha=0.
↪7, label='Centroid')

    plt.grid(True, linestyle='--', linewidth=0.5, alpha=0.7)
    plt.xlabel('X-axis')
    plt.ylabel('Y-axis')
    plt.title(title)
    plt.legend()
    plt.show()
```

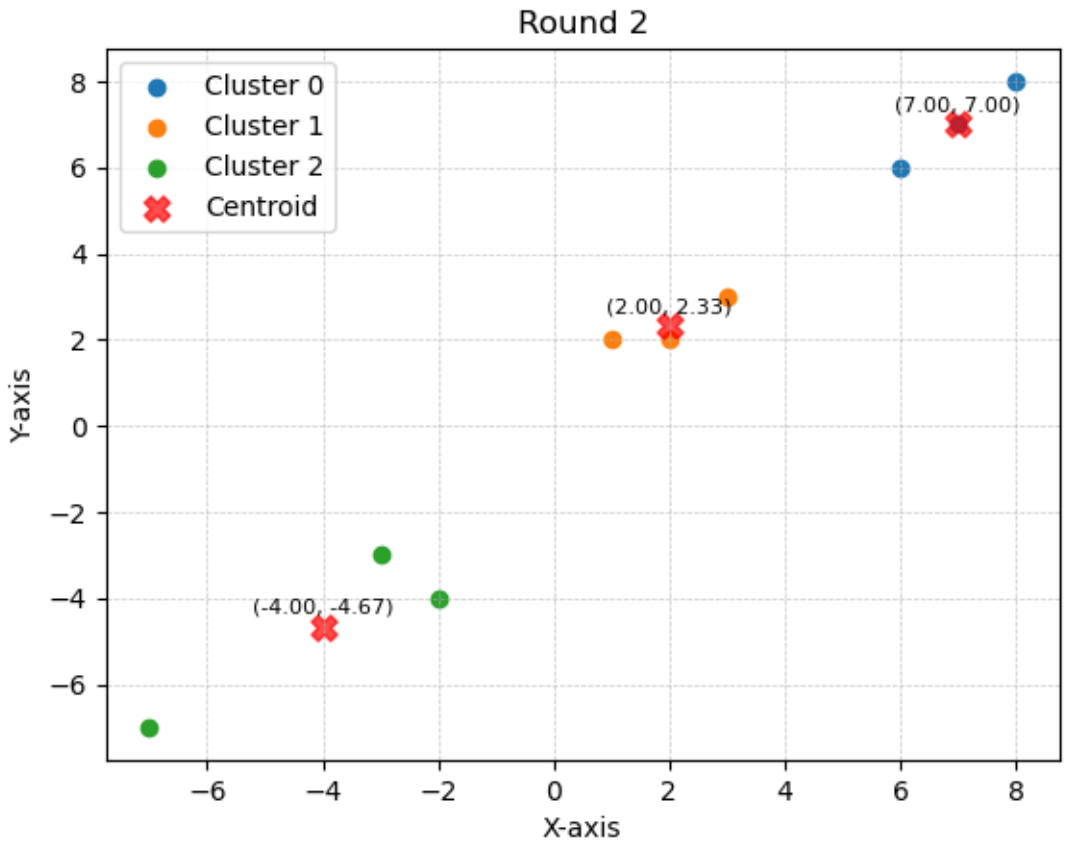
[ ]:
```
# Main function
data_x = np.array([1, 3, 2, 8, 6, 7, -3, -2, -7])
data_y = np.array([2, 3, 2, 8, 6, 7, -3, -4, -7])
data_points = list(zip(data_x, data_y))  # list of tuple [(x1, y1), (x2, y2), ..
↪.]

# Initialize k centroids
init_centroids = [(3,3), (2,2), (-3,-3)]
    # keyboard input
# k = int(input('k = '))
# init_centroids = []
# for i in range(k):
#     l = input('x,y = ').split()
#     x = int(l[0])
#     y = int(l[1])
#     init_centroids.append((x,y))

# Initialize cluster
cluster_centroid = dict() # {0: (x, y), 1: ...}
cluster_points = dict() # {0: [(),()], 1: ...}
for idx, point in enumerate(init_centroids):
    cluster_centroid[idx] = point
    cluster_points[idx] = []
print(f'Initial cluster_centroid {cluster_centroid}')
print(f'Initial cluster_points {cluster_points}')

centroid_isUpdating = True
round = 1
while centroid_isUpdating:
```

```
    print(f'Round {round}')
    # Clear cluster_points before updating
    cluster_points = {idx: [] for idx in cluster_points}

    # Assign each data point to clusters
    assign(data_points, cluster_centroid, cluster_points)

    # Update centroids
    centroid_isUpdating = update_centroids(cluster_centroid, cluster_points,␣
↪centroid_isUpdating)
    print(f'cluster_centroid {cluster_centroid}')
    print(f'cluster_points {cluster_points}')

    # Plot clusters for each round
    plot_clusters(data_points, cluster_centroid, cluster_points, title=f'Round␣
↪{round}')
    round += 1

print(f'Final cluster_centroids: {cluster_centroid}')
```

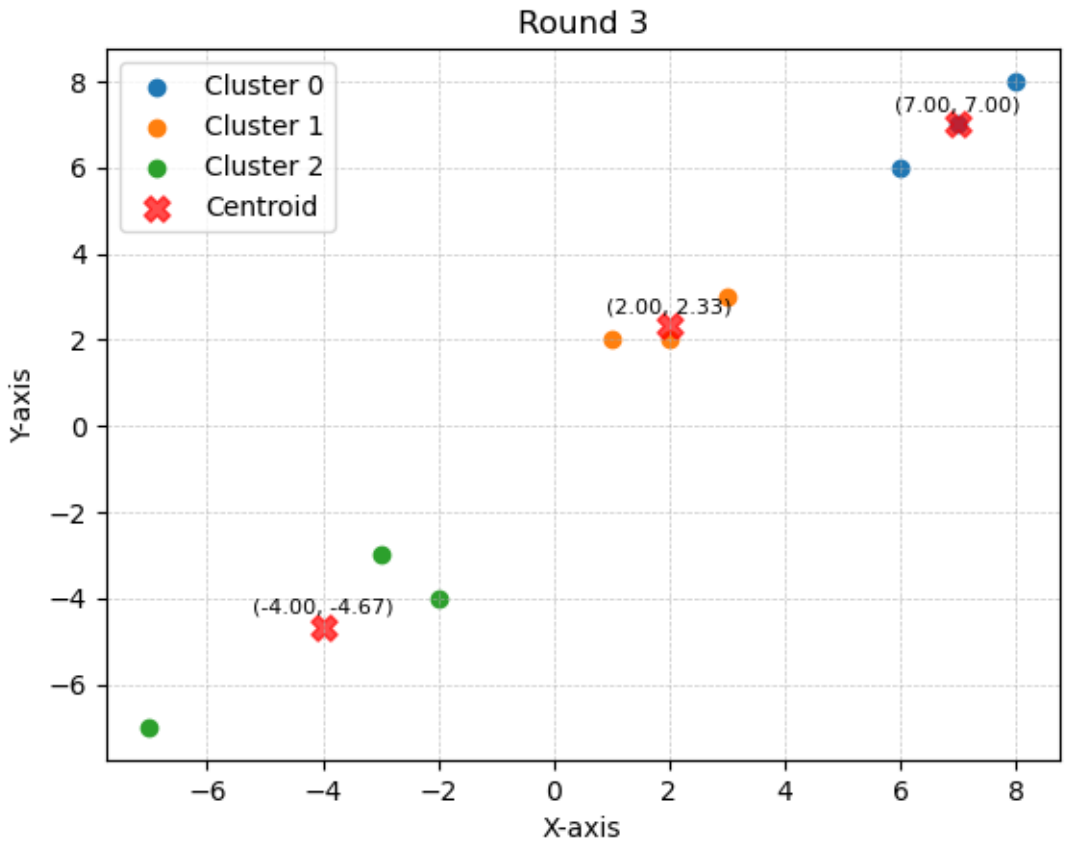
```
Initial cluster_centroid {0: (3, 3), 1: (2, 2), 2: (-3, -3)}
Initial cluster_points {0: [], 1: [], 2: []}
Round 1
cluster_centroid {0: (6.0, 6.0), 1: (1.5, 2.0), 2: (-4.0, -4.666666666666667)}
cluster_points {0: [(3, 3), (8, 8), (6, 6), (7, 7)], 1: [(1, 2), (2, 2)], 2:
[(-3, -3), (-2, -4), (-7, -7)]}
```




```
Round 2
cluster_centroid {0: (7.0, 7.0), 1: (2.0, 2.3333333333333335), 2: (-4.0,
-4.666666666666667)}
cluster_points {0: [(8, 8), (6, 6), (7, 7)], 1: [(1, 2), (3, 3), (2, 2)], 2:
[(-3, -3), (-2, -4), (-7, -7)]}
```




```
Round 3
cluster_centroid {0: (7.0, 7.0), 1: (2.0, 2.3333333333333335), 2: (-4.0,
-4.666666666666667)}
cluster_points {0: [(8, 8), (6, 6), (7, 7)], 1: [(1, 2), (3, 3), (2, 2)], 2:
[(-3, -3), (-2, -4), (-7, -7)]}
```




```
Final cluster_centroids: {0: (7.0, 7.0), 1: (2.0, 2.3333333333333335), 2: (-4.0,
-4.666666666666667)}
```

1.2.2 T6.

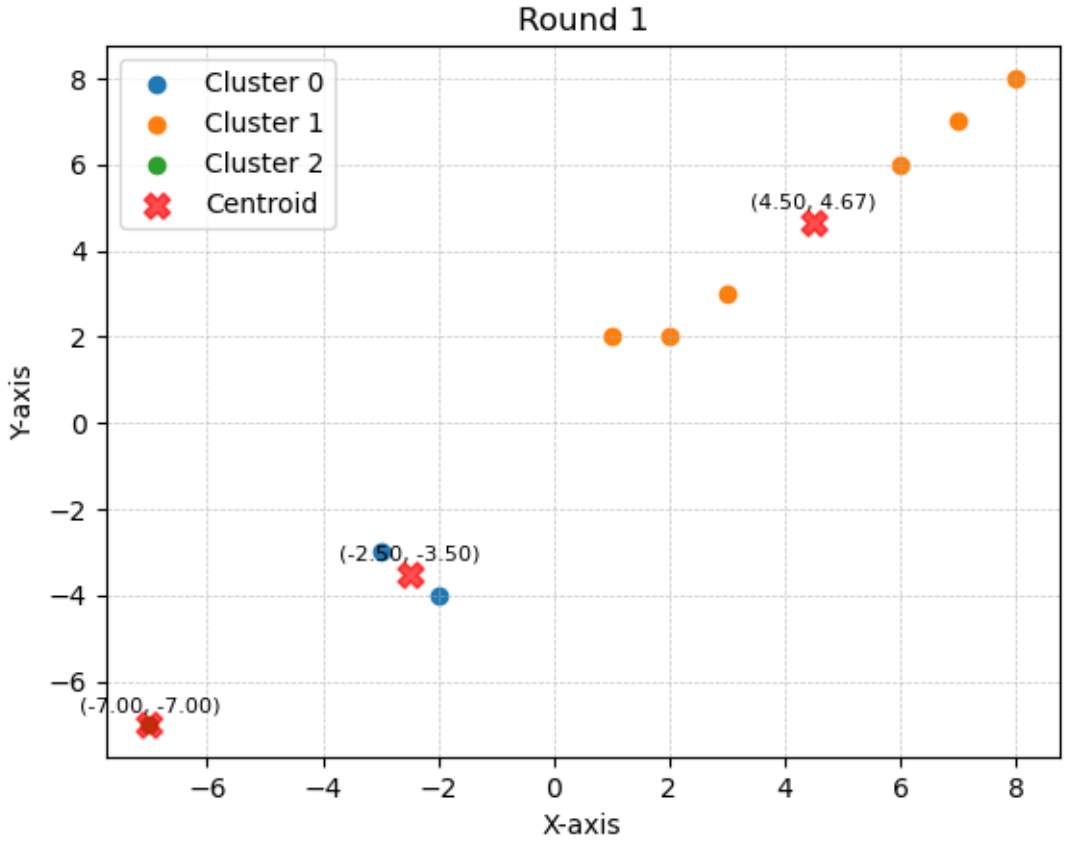
If the starting points are (-3,-3), (2,2), and (-7,-7), what happens?

[ ]:
```
import numpy as np
import math
import matplotlib.pyplot as plt

# function
def distance(point1, point2): # Euclidean Distance
    x1, y1 = point1
    x2, y2 = point2
    return math.sqrt((x2 - x1)**2 + (y2 - y1)**2) #float

def assign(data_points, cluster_centroid, cluster_points):
    for point in data_points: # assign each point to cluster
        distances = [distance(point, centroid) for centroid in cluster_centroid.
 ↪values()]
```

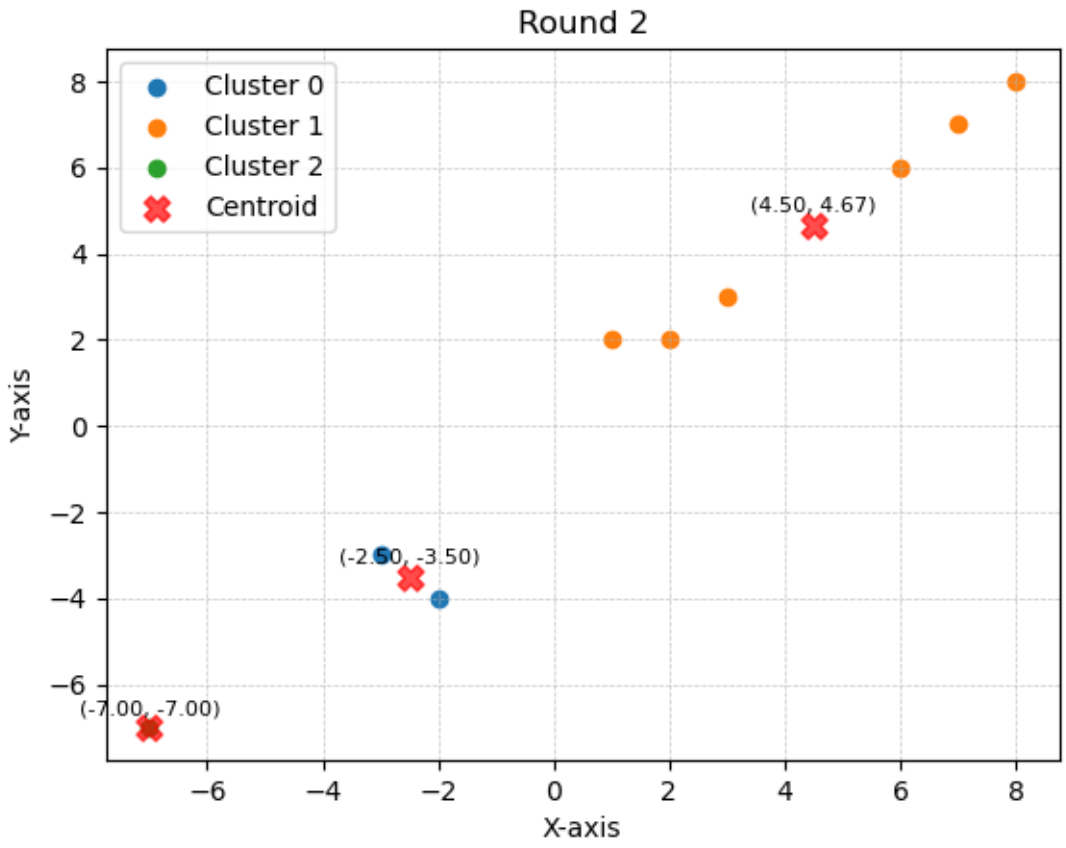
```
        min_distance_index = np.argmin(distances)
        cluster_points[min_distance_index].append(point)

def update_centroids(cluster_centroid, cluster_points, centroid_isUpdating):
    old_centroids = list(cluster_centroid.values())

    # calculate new_centroids
    new_centroids = []
    for idx, points in cluster_points.items():
        if points:  # Check if points list is not empty
            new_centroid_x, new_centroid_y = np.mean(points, axis=0)
            new_centroid = (new_centroid_x, new_centroid_y)
            new_centroids.append(new_centroid)
        else:
            new_centroids.append(cluster_centroid[idx]) # Use the existing␣
 ↪centroid if no points in the cluster

    # update new centroid
    for idx, centroid in enumerate(new_centroids):
        cluster_centroid[idx] = centroid

    if new_centroids == old_centroids:
        centroid_isUpdating = False
    return centroid_isUpdating #bool

def plot_clusters(data_points, cluster_centroid, cluster_points, title):
    for idx, points in cluster_points.items():
        cluster_x, cluster_y = zip(*points)
        plt.scatter(cluster_x, cluster_y, label=f'Cluster {idx}')

    # Annotate centroids with coordinates
    for idx, (x, y) in cluster_centroid.items():
        plt.annotate(f'({x:.2f}, {y:.2f})', (x, y), textcoords="offset points",␣
 ↪xytext=(0,5), ha='center', fontsize=8)

    centroid_x, centroid_y = zip(*cluster_centroid.values())
    plt.scatter(centroid_x, centroid_y, color='red', marker='X',s=90, alpha=0.
 ↪7, label='Centroid')

    plt.grid(True, linestyle='--', linewidth=0.5, alpha=0.7)
    plt.xlabel('X-axis')
    plt.ylabel('Y-axis')
    plt.title(title)
    plt.legend()
    plt.show()

# Main function
```

```
data_x = np.array([1, 3, 2, 8, 6, 7, -3, -2, -7])
data_y = np.array([2, 3, 2, 8, 6, 7, -3, -4, -7])
data_points = list(zip(data_x, data_y))  # list of tuple [(x1, y1), (x2, y2), ..
 ↪.]

# Initialize k centroids
init_centroids = [(-3,-3), (2,2), (-7,-7)]
    # keyboard input
# k = int(input('k = '))
# init_centroids = []
# for i in range(k):
#     l = input('x,y = ').split()
#     x = int(l[0])
#     y = int(l[1])
#     init_centroids.append((x,y))

# Initialize cluster
cluster_centroid = dict() # {0: (x, y), 1: ...}
cluster_points = dict() # {0: [(),()], 1: ...}
for idx, point in enumerate(init_centroids):
    cluster_centroid[idx] = point
    cluster_points[idx] = []
print(f'Initial cluster_centroid {cluster_centroid}')
print(f'Initial cluster_points {cluster_points}')

centroid_isUpdating = True
round = 1
while centroid_isUpdating:
    print(f'Round {round}')
    # Clear cluster_points before updating
    cluster_points = {idx: [] for idx in cluster_points}

    # Assign each data point to clusters
    assign(data_points, cluster_centroid, cluster_points)

    # Update centroids
    centroid_isUpdating = update_centroids(cluster_centroid, cluster_points,␣
 ↪centroid_isUpdating)
    print(f'cluster_centroid {cluster_centroid}')
    print(f'cluster_points {cluster_points}')

    # Plot clusters for each round
    plot_clusters(data_points, cluster_centroid, cluster_points, title=f'Round␣
 ↪{round}')
    round += 1

print(f'Final cluster_centroids: {cluster_centroid}')
```

```
Initial cluster_centroid {0: (-3, -3), 1: (2, 2), 2: (-7, -7)}
Initial cluster_points {0: [], 1: [], 2: []}
Round 1
cluster_centroid {0: (-2.5, -3.5), 1: (4.5, 4.666666666666667), 2: (-7.0, -7.0)}
cluster_points {0: [(-3, -3), (-2, -4)], 1: [(1, 2), (3, 3), (2, 2), (8, 8), (6,
6), (7, 7)], 2: [(-7, -7)]}
```




```
Round 2
cluster_centroid {0: (-2.5, -3.5), 1: (4.5, 4.666666666666667), 2: (-7.0, -7.0)}
cluster_points {0: [(-3, -3), (-2, -4)], 1: [(1, 2), (3, 3), (2, 2), (8, 8), (6,
6), (7, 7)], 2: [(-7, -7)]}
```




```
Final cluster_centroids: {0: (-2.5, -3.5), 1: (4.5, 4.666666666666667), 2:
(-7.0, -7.0)}
```

1.2.3 T7.

Between the two starting set of points in the previous two questions,
which one do you think is better?
How would you measure the 'goodness' quality of a set of starting points?

Answer

Which one do you think is better?

Set of points in T5. question (3,3), (2,2), (-3,-3) because this set has the same amount of data in each cluster, and upon visual inspection, there is no group straddling, unlike T6 question.

How would you measure the 'goodness' quality of a set of starting points?

The 'goodness' quality of a set of starting points can be measured using various metrics such as Inertia (Within-Cluster Sum of Squares), Silhouette Score, Davies-Bouldin Index, and Calinski-Harabasz Index (Variance Ratio Criterion), or it can be assessed visually.

1.2.4 OT2.

What would be the best K for this question? Describe your reasoning.

Answer: $K = 4$, From visual observation, it is evident that when K = 4, the grouping is more appropriate. The centroids of all points are close to the data within their respective clusters, unlike K = 3, where one cluster located at the bottom left corner of the image has a centroid positioned in the middle, serving as a representative between the three data points.

[ ]:
```
import numpy as np
import math
import matplotlib.pyplot as plt

# function
def distance(point1, point2): # Euclidean Distance
    x1, y1 = point1
    x2, y2 = point2
    return math.sqrt((x2 - x1)**2 + (y2 - y1)**2) #float

def assign(data_points, cluster_centroid, cluster_points):
    for point in data_points: # assign each point to cluster
        distances = [distance(point, centroid) for centroid in cluster_centroid.
 ↪values()]
        min_distance_index = np.argmin(distances)
        cluster_points[min_distance_index].append(point)

def update_centroids(cluster_centroid, cluster_points, centroid_isUpdating):
    old_centroids = list(cluster_centroid.values())

    # calculate new_centroids
    new_centroids = []
    for idx, points in cluster_points.items():
        if points:  # Check if points list is not empty
            new_centroid_x, new_centroid_y = np.mean(points, axis=0)
            new_centroid = (new_centroid_x, new_centroid_y)
            new_centroids.append(new_centroid)
        else:
            new_centroids.append(cluster_centroid[idx]) # Use the existing␣
 ↪centroid if no points in the cluster

    # update new centroid
    for idx, centroid in enumerate(new_centroids):
        cluster_centroid[idx] = centroid

    if new_centroids == old_centroids:
        centroid_isUpdating = False
    return centroid_isUpdating #bool
```

```
def plot_clusters(data_points, cluster_centroid, cluster_points, title):
    for idx, points in cluster_points.items():
        cluster_x, cluster_y = zip(*points)
        plt.scatter(cluster_x, cluster_y, label=f'Cluster {idx}')

    # Annotate centroids with coordinates
    for idx, (x, y) in cluster_centroid.items():
        plt.annotate(f'({x:.2f}, {y:.2f})', (x, y), textcoords="offset points",␣
 ↪xytext=(0,5), ha='center', fontsize=8)

    centroid_x, centroid_y = zip(*cluster_centroid.values())
    plt.scatter(centroid_x, centroid_y, color='red', marker='X',s=90, alpha=0.
 ↪7, label='Centroid')

    plt.grid(True, linestyle='--', linewidth=0.5, alpha=0.7)
    plt.xlabel('X-axis')
    plt.ylabel('Y-axis')
    plt.title(title)
    plt.legend()
    plt.show()

# Main function
data_x = np.array([1, 3, 2, 8, 6, 7, -3, -2, -7])
data_y = np.array([2, 3, 2, 8, 6, 7, -3, -4, -7])
data_points = list(zip(data_x, data_y))  # list of tuple [(x1, y1), (x2, y2), ..
 ↪.]

# Initialize k centroids
init_centroids = [(-3,-3), (2,2), (-7,-7), (0,0)]
    # keyboard input
# k = int(input('k = '))
# init_centroids = []
# for i in range(k):
#     l = input('x,y = ').split()
#     x = int(l[0])
#     y = int(l[1])
#     init_centroids.append((x,y))

# Initialize cluster
cluster_centroid = dict() # {0: (x, y), 1: ...}
cluster_points = dict() # {0: [(),()], 1: ...}
for idx, point in enumerate(init_centroids):
    cluster_centroid[idx] = point
    cluster_points[idx] = []
print(f'Initial cluster_centroid {cluster_centroid}')
print(f'Initial cluster_points {cluster_points}')
```

```
centroid_isUpdating = True
round = 1
while centroid_isUpdating:
    print(f'Round {round}')
    # Clear cluster_points before updating
    cluster_points = {idx: [] for idx in cluster_points}

    # Assign each data point to clusters
    assign(data_points, cluster_centroid, cluster_points)

    # Update centroids
    centroid_isUpdating = update_centroids(cluster_centroid, cluster_points,␣
 ↪centroid_isUpdating)
    print(f'cluster_centroid {cluster_centroid}')
    print(f'cluster_points {cluster_points}')

    # # Plot clusters for each round
    # plot_clusters(data_points, cluster_centroid, cluster_points,␣
 ↪title=f'Round {round}')
    # round += 1

print(f'Final cluster_centroids: {cluster_centroid}')

plot_clusters(data_points, cluster_centroid, cluster_points, title='Final␣
 ↪Clusters')
```

```
Initial cluster_centroid {0: (-3, -3), 1: (2, 2), 2: (-7, -7), 3: (0, 0)}
Initial cluster_points {0: [], 1: [], 2: [], 3: []}
Round 1
cluster_centroid {0: (-2.5, -3.5), 1: (4.5, 4.666666666666667), 2: (-7.0, -7.0),
3: (0, 0)}
cluster_points {0: [(-3, -3), (-2, -4)], 1: [(1, 2), (3, 3), (2, 2), (8, 8), (6,
6), (7, 7)], 2: [(-7, -7)], 3: []}
Round 1
cluster_centroid {0: (-2.5, -3.5), 1: (6.0, 6.0), 2: (-7.0, -7.0), 3: (1.5,
2.0)}
cluster_points {0: [(-3, -3), (-2, -4)], 1: [(3, 3), (8, 8), (6, 6), (7, 7)], 2:
[(-7, -7)], 3: [(1, 2), (2, 2)]}
Round 1
cluster_centroid {0: (-2.5, -3.5), 1: (7.0, 7.0), 2: (-7.0, -7.0), 3: (2.0,
2.3333333333333335)}
cluster_points {0: [(-3, -3), (-2, -4)], 1: [(8, 8), (6, 6), (7, 7)], 2: [(-7,
-7)], 3: [(1, 2), (3, 3), (2, 2)]}
Round 1
cluster_centroid {0: (-2.5, -3.5), 1: (7.0, 7.0), 2: (-7.0, -7.0), 3: (2.0,
2.3333333333333335)}
cluster_points {0: [(-3, -3), (-2, -4)], 1: [(8, 8), (6, 6), (7, 7)], 2: [(-7,
```

```
-7)], 3: [(1, 2), (3, 3), (2, 2)]}
Final cluster_centroids: {0: (-2.5, -3.5), 1: (7.0, 7.0), 2: (-7.0, -7.0), 3:
(2.0, 2.3333333333333335)}
```

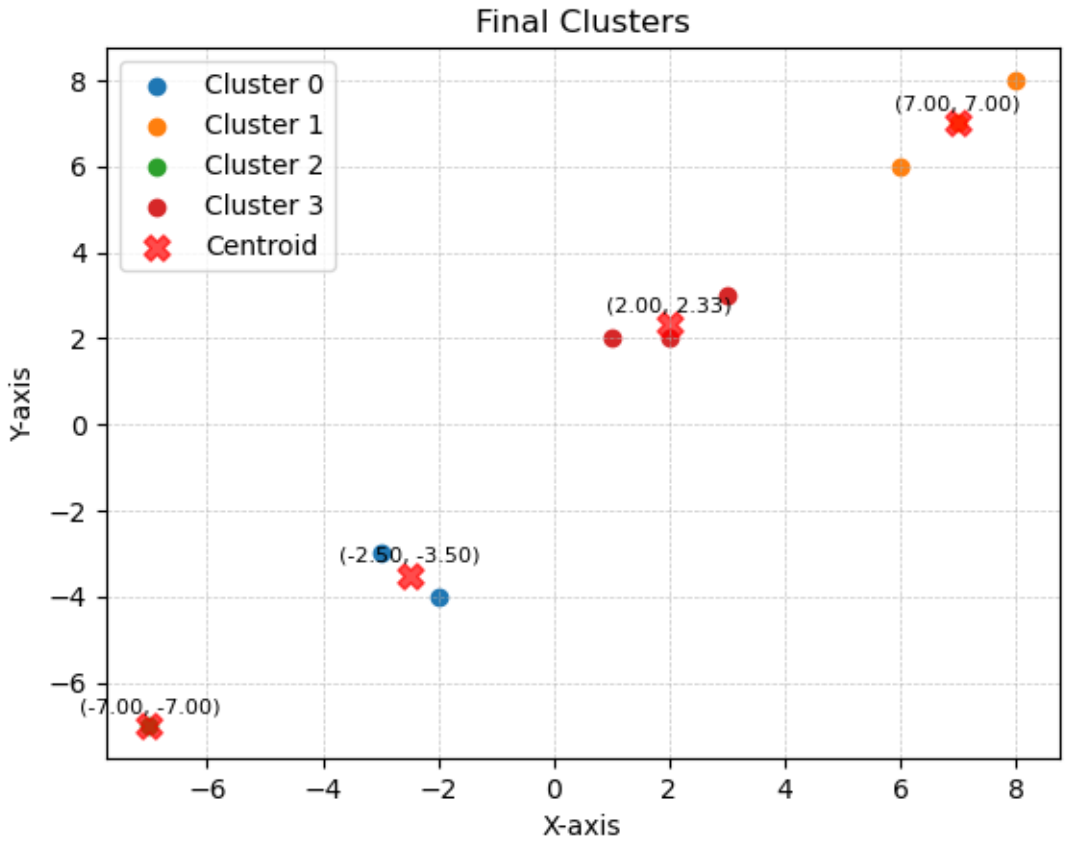


image for K = 4

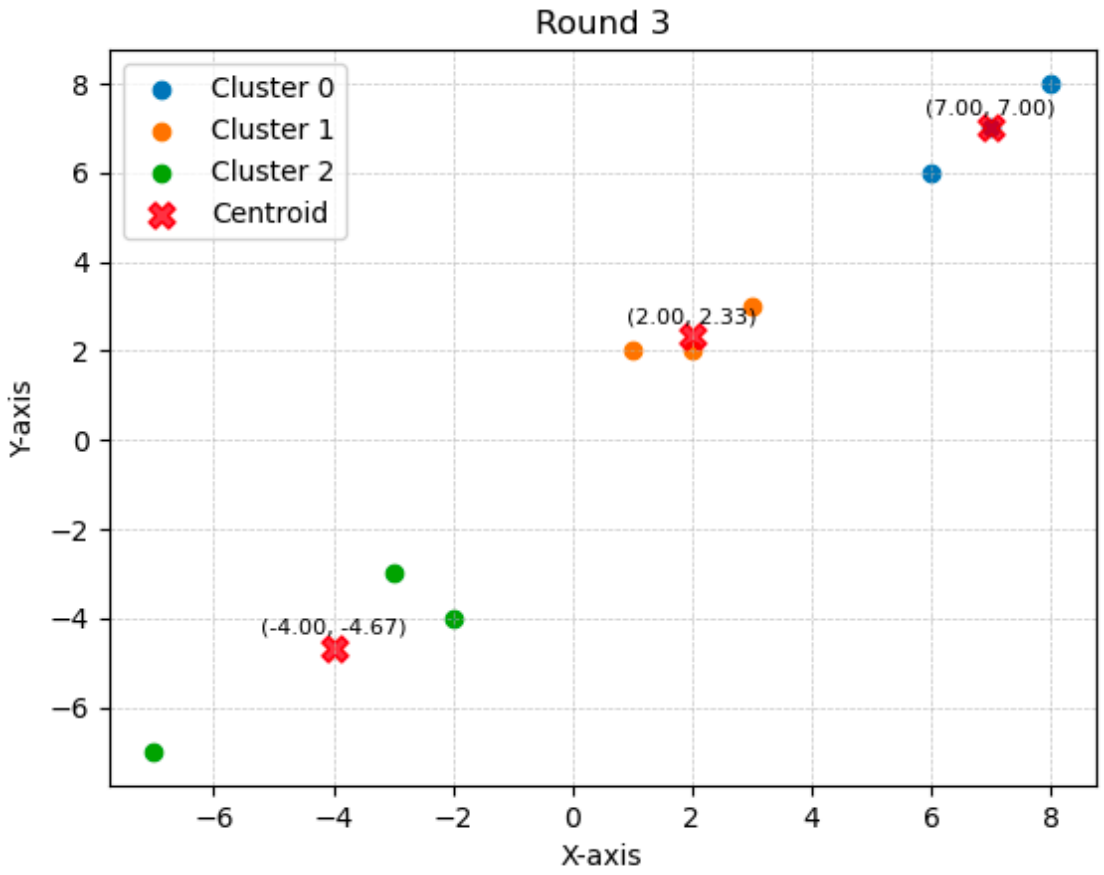


image for K = 3

1.3 Titanic : My heart will go on

In this part of the exercise we will work on the Titanic dataset provided by Kaggle. The Titanic dataset contains information of the passengers boarding the Titanic on its final voyage. We will work on predicting whether a given passenger will survive the trip. Let's launch Jupyter and start coding! We start by importing the data using Pandas

[ ]:
```
import pandas as pd
```

[ ]:
```
train_url = "http://s3.amazonaws.com/assets.datacamp.com/course/Kaggle/train.
↪csv"
train = pd.read_csv(train_url) #training set

test_url = "http://s3.amazonaws.com/assets.datacamp.com/course/Kaggle/test.csv"
test = pd.read_csv(test_url) #test set
```

[ ]:
```
train.head()
```

[ ]:
```
train.tail()
```

[ ]:
```
train.describe()
```

[ ]:
```
train.info()
```

1.3.1 T8.

What is the median age of the training set? You can easily modify the age in the dataframe by

[ ]:
```
train["Age"] = train["Age"].fillna(train["Age"].median())
```

Note that you need to modify the code above a bit to fill with mode() because mode() returns a series rather than a single value.