CSE 343: Machine Learning

Assignment 4: Report

Aditya Aggarwal 2022028

SECTION B

(a) KMeans Algorithm

Calculating Euclidean Distance

$$D = \sqrt{\left(x_2 - x_1\right)^2 + \left(y_2 - y_1\right)^2}$$

```
def euclidean_distance(p1, p2):
    return np.sqrt(np.sum((np.array(p1) - np.array(p2)) ** 2))
```
0.0s

Initialization

Convert the list of sample points and the centroids to NumPy arrays. Create a list of labels with a size matching the number of samples to a NumPy array with all entries initialized to zero.

```
# Initialization
data = np.array(data)
centroids = np.array(centroids)
labels = np.zeros(len(data), dtype= int)
```

Assignment

Calculate the distances from each data point to every centroid and assign the label of the centroid with the minimum Euclidean distance to the particular sample.

```
# Assignment
for idx, point in enumerate(data):
    distances = [euclidean_distance(point, centroid) for centroid in centroids]
    labels[idx] = np.argmin(distances)
```

Update

Update the centroids by calculating the mean of points in each cluster, or retain the old centroid if the cluster has no points.

```
# Update
new_centroids = np.zeros_like(centroids)
for cluster in range(k):
    points = data[labels == cluster]
    if len(points) != 0:
        new_centroids[cluster] = np.mean(points, axis=0)
    else:
        new_centroids[cluster] = centroids[cluster]
```

Check for Convergence

If all centroids have moved less than the specified threshold, terminate the loop as the algorithm has converged. Otherwise, update the centroids.

```
# Convergence Check
if np.all(np.abs(new_centroids - centroids) < theshold):
    print(f"Algorithm converged at iteration: {i+1}.")
    break

centroids = new_centroids
```

(b) Centroids and Clusters Obtained

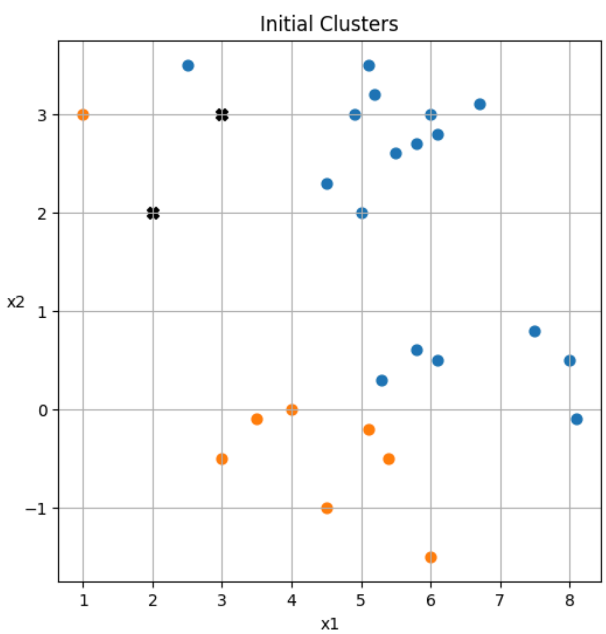
- The initial and final values of the centroids are as follows:

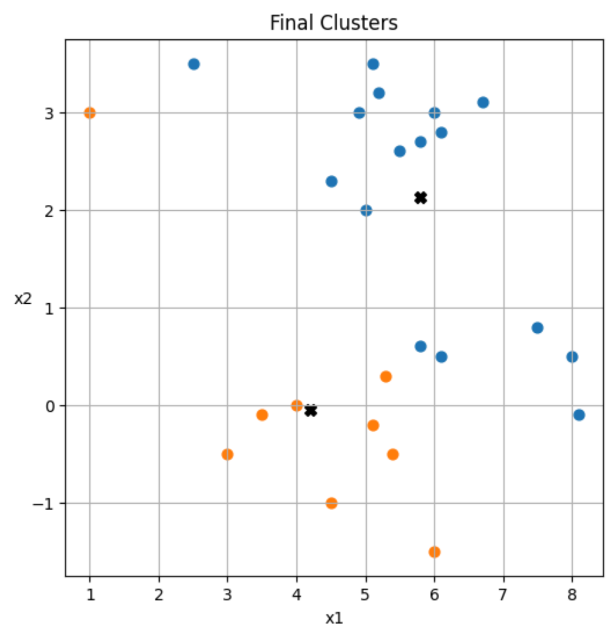
```
Initial values of the centroids are:
u1: (3.00, 3.00)
u2: (2.00, 2.00)
Final values of the centroids are:
u1: (5.80, 2.12)
u2: (4.20, -0.06)
```

- The algorithm converges at the 3rd iteration

```
Algorithm converged at iteration: 3.
```

- Initial and Final Clusters





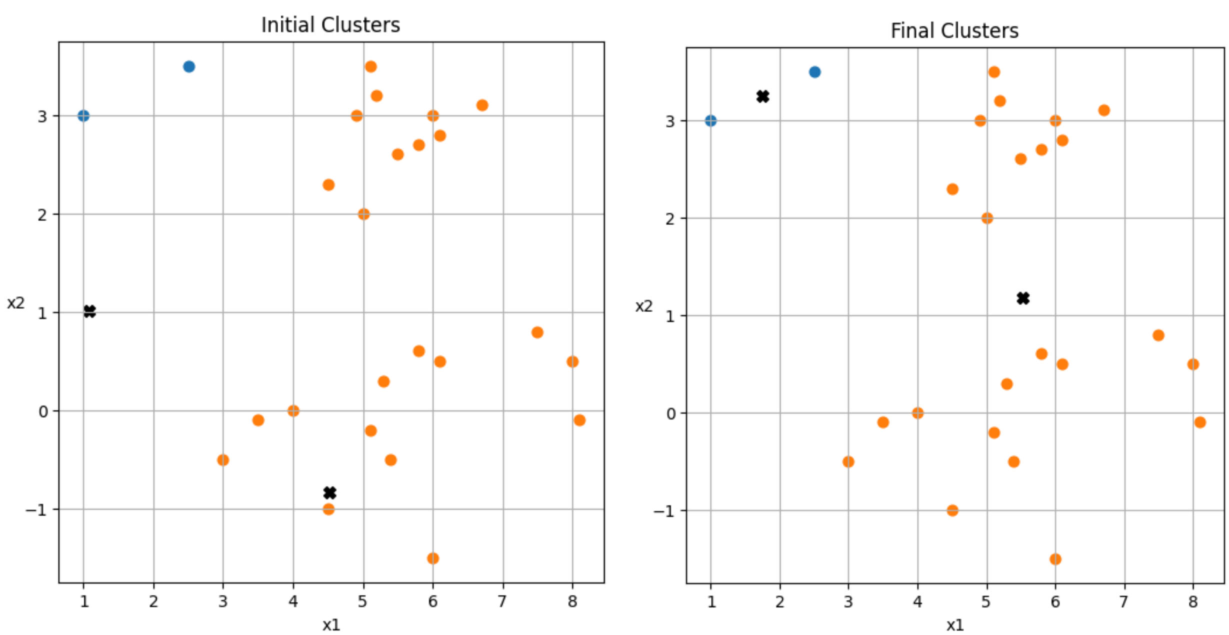
(c) Random Initialization of Centroids

- **Seed = 9**

1st Random Initialization

```
Algorithm converged at iteration: 2.
Initial values of the centroids are:
u1: (1.07, 1.01)
u2: (4.52, -0.83)
Final values of the centroids are:
u1: (1.75, 3.25)
u2: (5.53, 1.17)
```

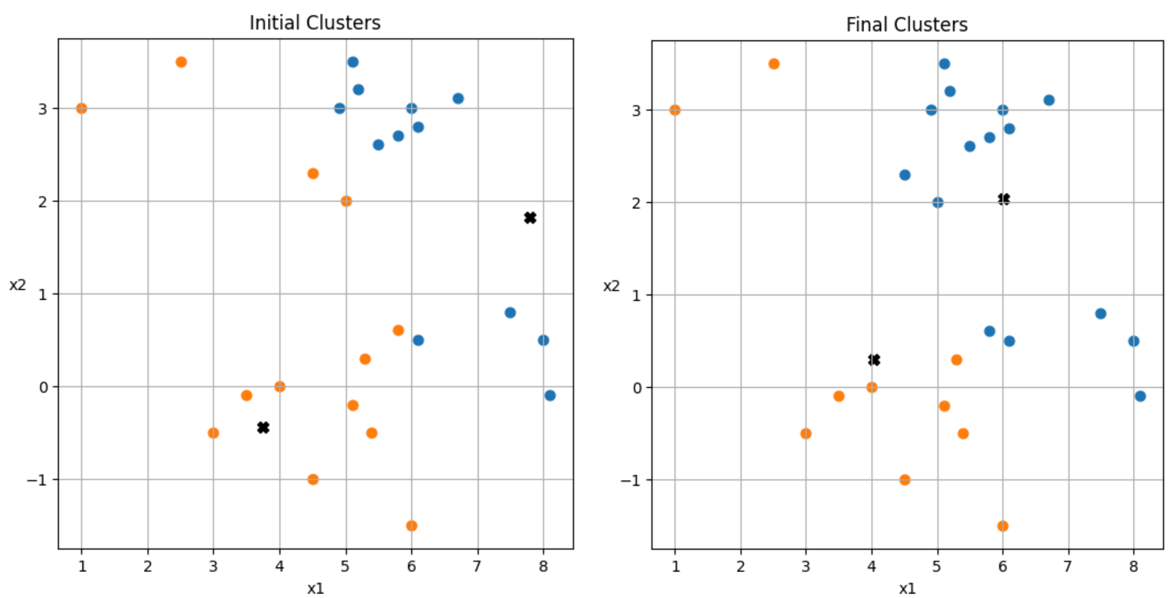
Initial and Final Clusters



2nd Random Initialization

```
Algorithm converged at iteration: 4.
Initial values of the centroids are:
u1: (7.79, 1.82)
u2: (3.75, -0.44)
Final values of the centroids are:
u1: (6.02, 2.03)
u2: (4.03, 0.30)
```

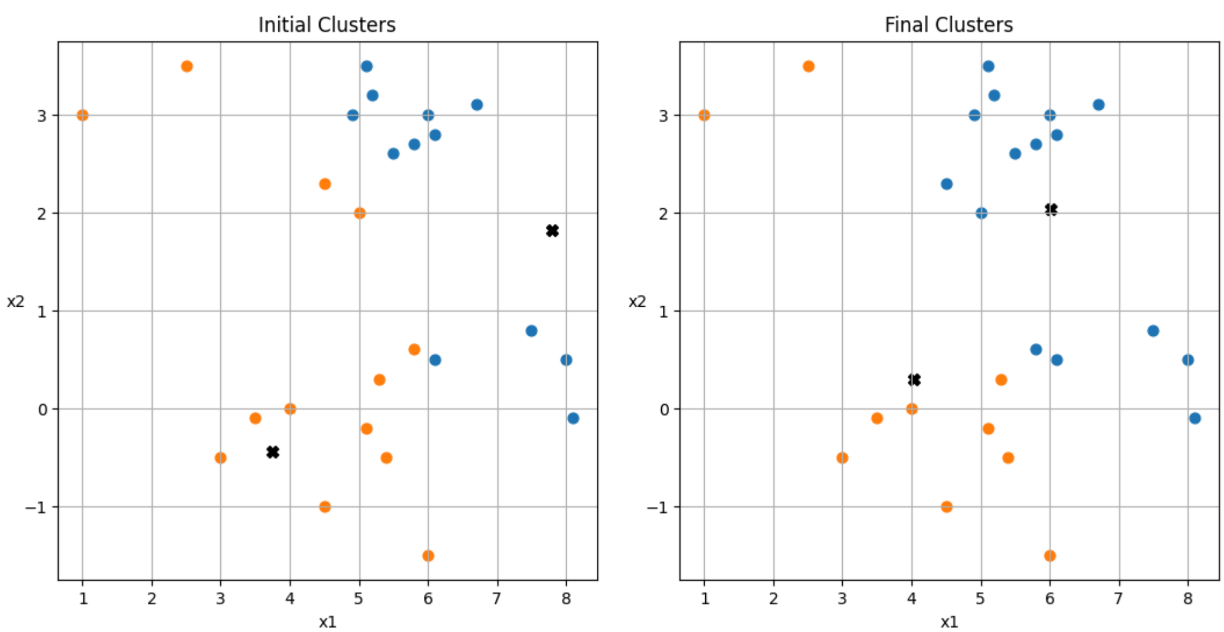
Initial and Final Clusters



3rdRandom Initialization

```
Algorithm converged at iteration: 4.
Initial values of the centroids are:
u1: (7.79, 1.82)
u2: (3.75, -0.44)
Final values of the centroids are:
u1: (6.02, 2.03)
u2: (4.03, 0.30)
```

Initial and Final Clusters



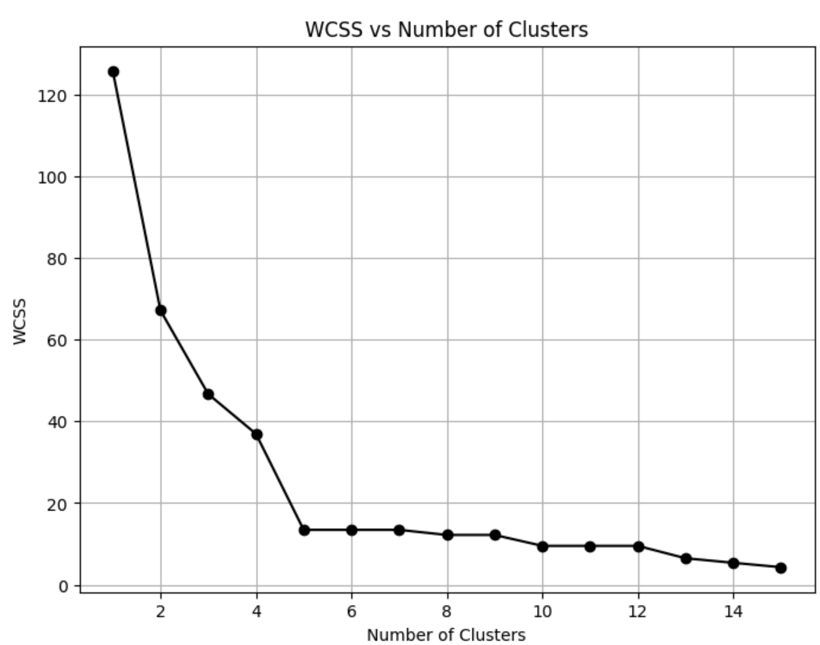
(d) Optimal Number of Clusters

Computing Within-Cluster Sum of Squares (WCSS) Loss:

$$WCSS = \sum_{i=1}^{k} \sum_{x_j \in C_i} \left\| x_j - a_i \right\|^2$$

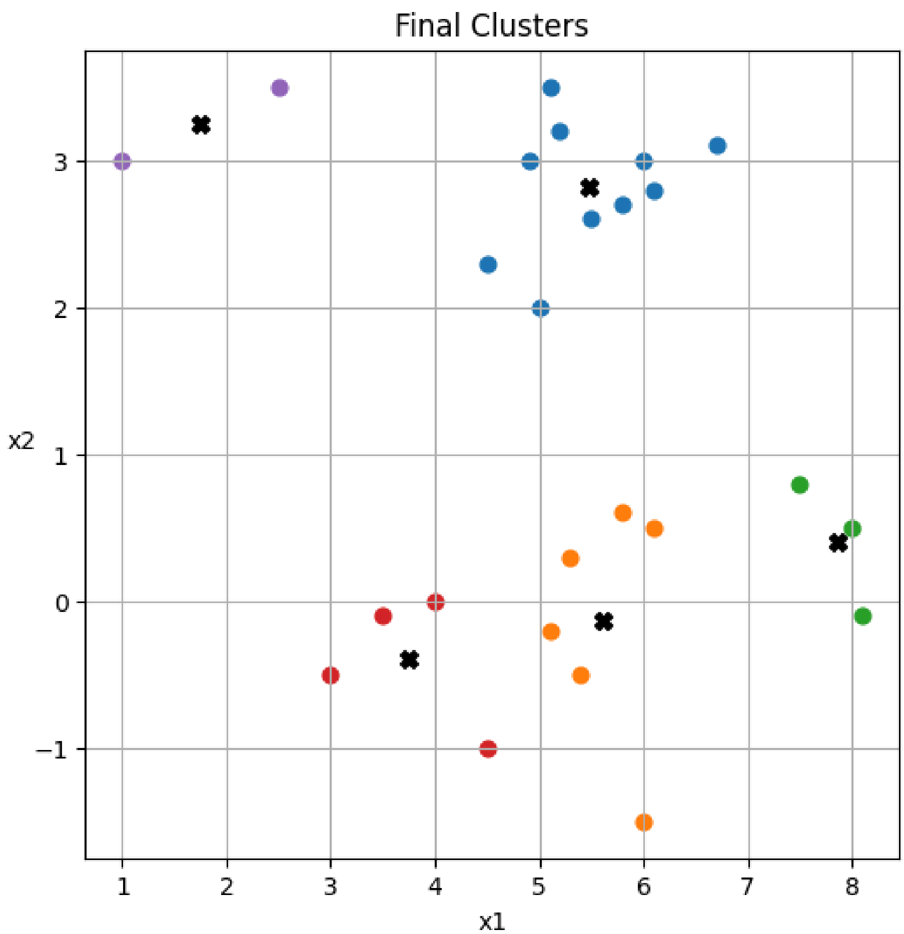
```
def get_wcss(data, centroids, labels):
    wcss = 0
    for i, point in enumerate(data):
        centroid = centroids[labels[i]]
        wcss += euclidean_distance(point, centroid) ** 2
    return wcss
```

WCSS Loss vs Number of Clusters for **Seed 666**:

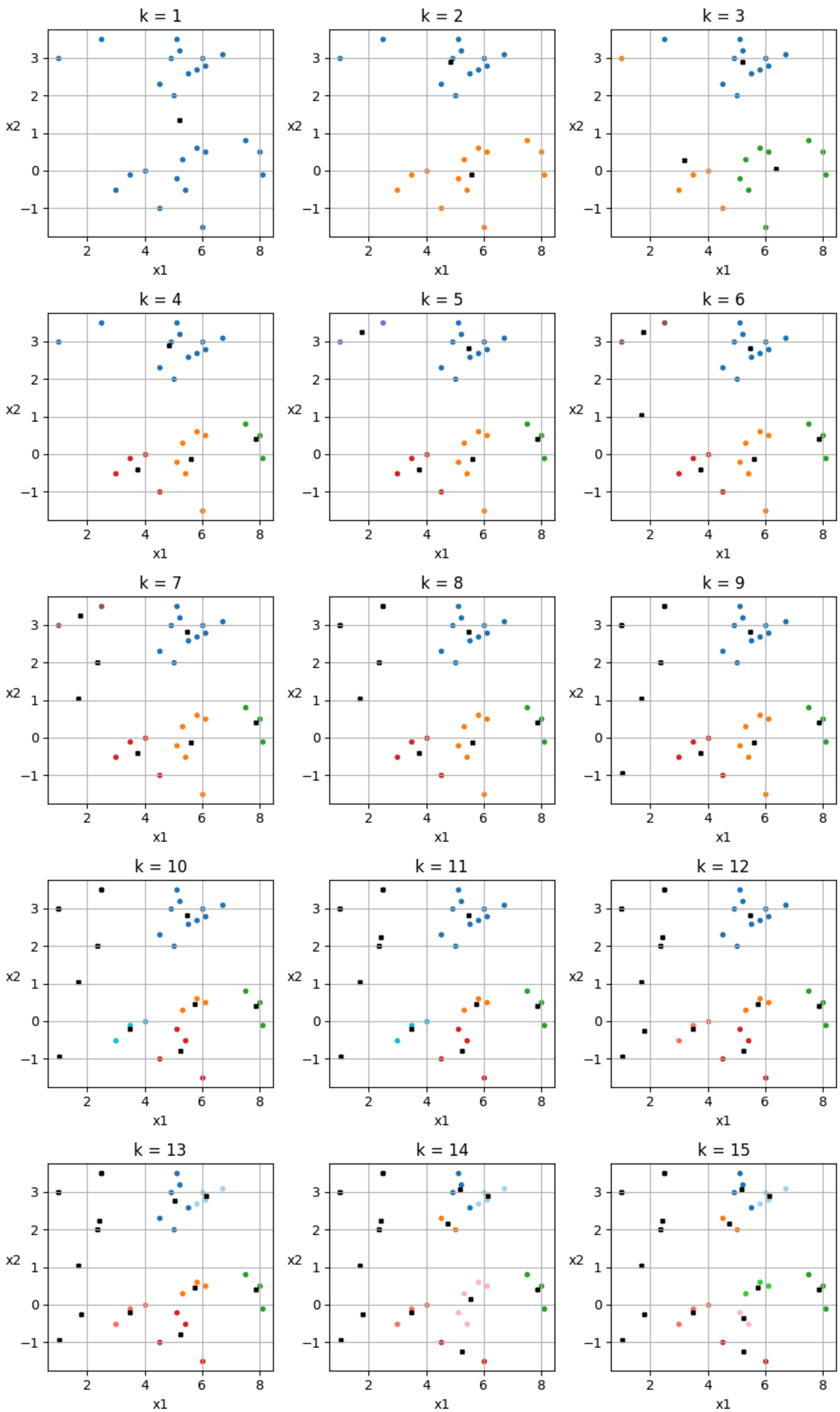


As we can see, there is a sharp decline in decrease in loss at k = 5. Thus, the optimal number of clusters, $M = 5$

For the number of clusters set to k = M = 5, the final clusters are as follows:



Below is the plot for the final clusters initialized with random centroids at **seed-666** with k clusters, $\forall k \in \{1, 2, \ldots, 15\}$


k = 1
k = 2
k = 3
k = 4
k = 5
k = 6
k = 7
k = 8
k = 9
k = 10
k = 11
k = 12
k = 13
k = 14
k = 15
x1
x2