problemresult

June 12, 2024

1 Algorithm Execution and Results

The algorithm executes both K-median and K-means++ for each dataset 10 times. It then reports the best result from these runs. The termination of the algorithm is controlled by a tolerance level and a maximum number of iterations, although it typically completes well before reaching these thresholds.

After processing each dataset, cluster plots are generated and displayed within this notebook for visual analysis.

1.1 System Information

- **Chip:** Apple M2
- **Total Number of Cores:** 8 (4 performance and 4 efficiency)
- **Memory:** 8 GB

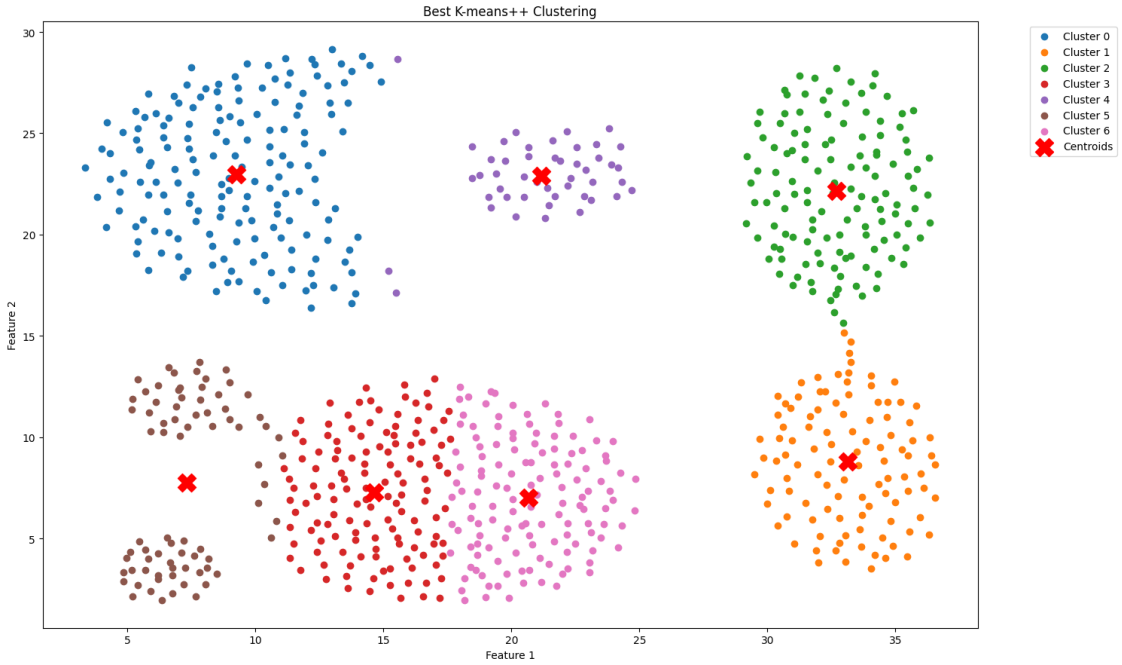
The runtime of the algorithm on this machine is approximately 3 seconds.

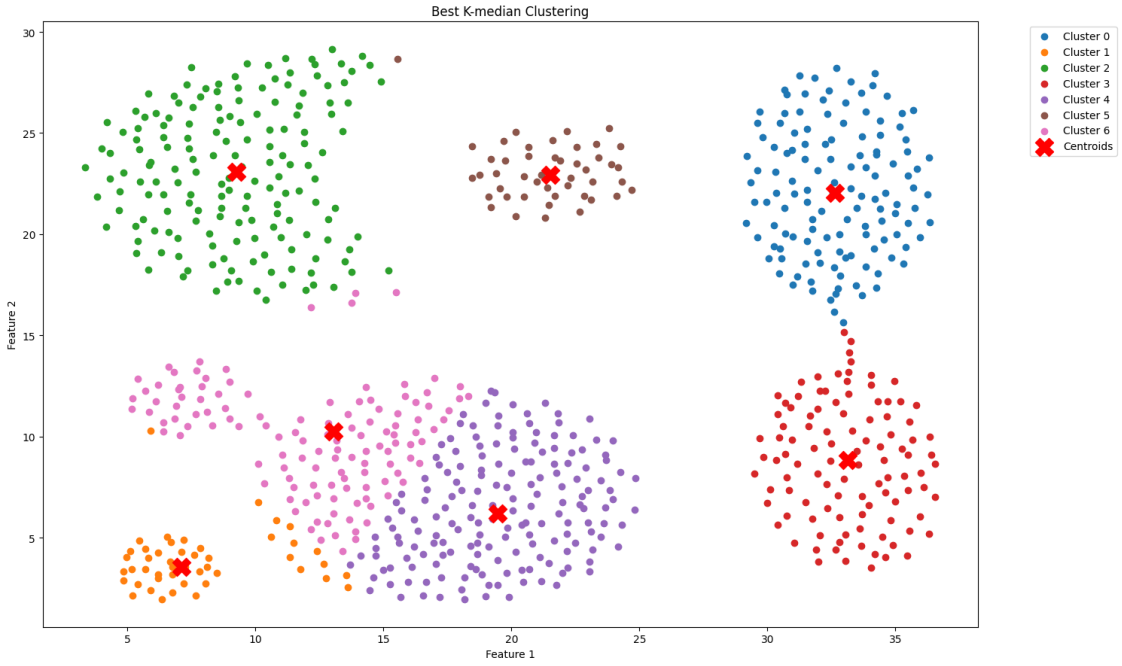
[1]:
```
from main import *
```

[2]:
```
# Load the data
data_set = load_data("Aggregation.data")

# Determine the number of clusters (assuming the last column is the label)
label_number = len(set(data_set.iloc[:, -1]))

# Extract true labels
true_labels = data_set.iloc[:, -1]

# Run clustering multiple times for K-means
best_clustered_data_kmeans, best_centroids_kmeans, best_purity_kmeans,␣
 ↪iteration_number = run_multiple_times(data_set.iloc[:, :-1], true_labels,␣
 ↪label_number)

# Run clustering multiple times for K-median
best_clustered_data_kmedian, best_centroids_kmedian, best_purity_kmedian,␣
 ↪iteration_number2= run_multiple_times(data_set.iloc[:, :-1], true_labels,␣
 ↪label_number, use_kmedian=True)
```

```
# Print the best results
print(f"Aggregation Best K-means Purity: {best_purity_kmeans:.4f}  iteration:␣
 ↪{iteration_number}")
print(f"Aggregation Best K-median Purity: {best_purity_kmedian:.4f} iteration:␣
 ↪{iteration_number2}")

# Plot the best results for K-means
plot_clusters(best_clustered_data_kmeans, best_centroids_kmeans, label_number,␣
 ↪'Best K-means++ Clustering')

# Plot the best results for K-median
plot_clusters(best_clustered_data_kmedian, best_centroids_kmedian,␣
 ↪label_number, 'Best K-median Clustering')
```

```
Aggregation Best K-means Purity: 0.9391  iteration: 10
Aggregation Best K-median Purity: 0.9327 iteration: 13
```

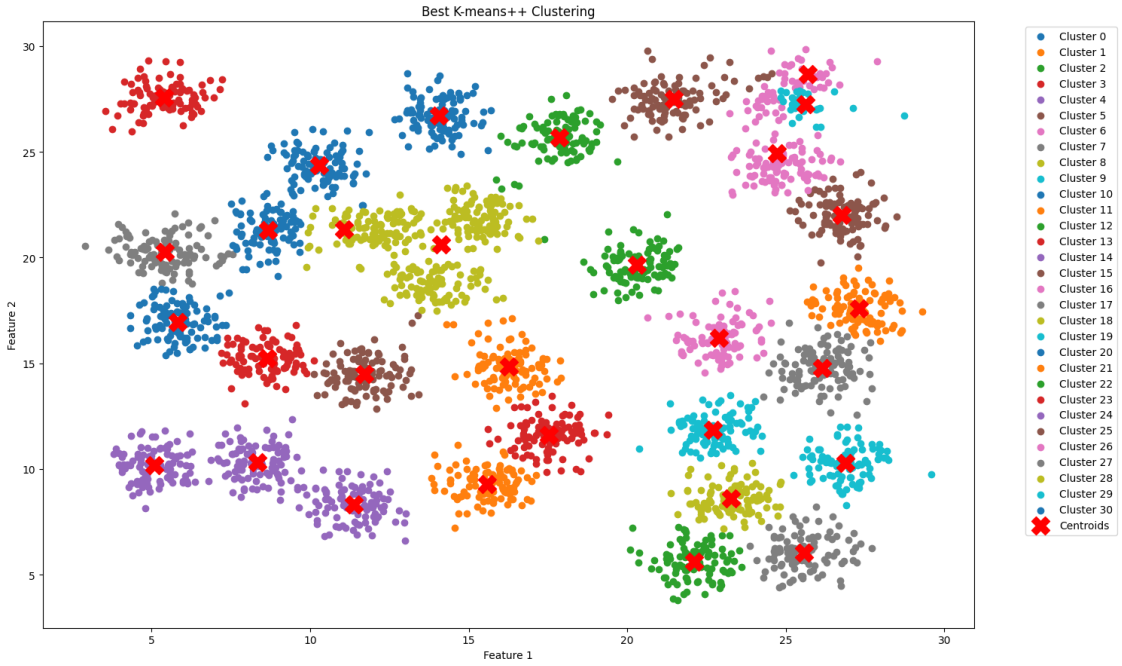


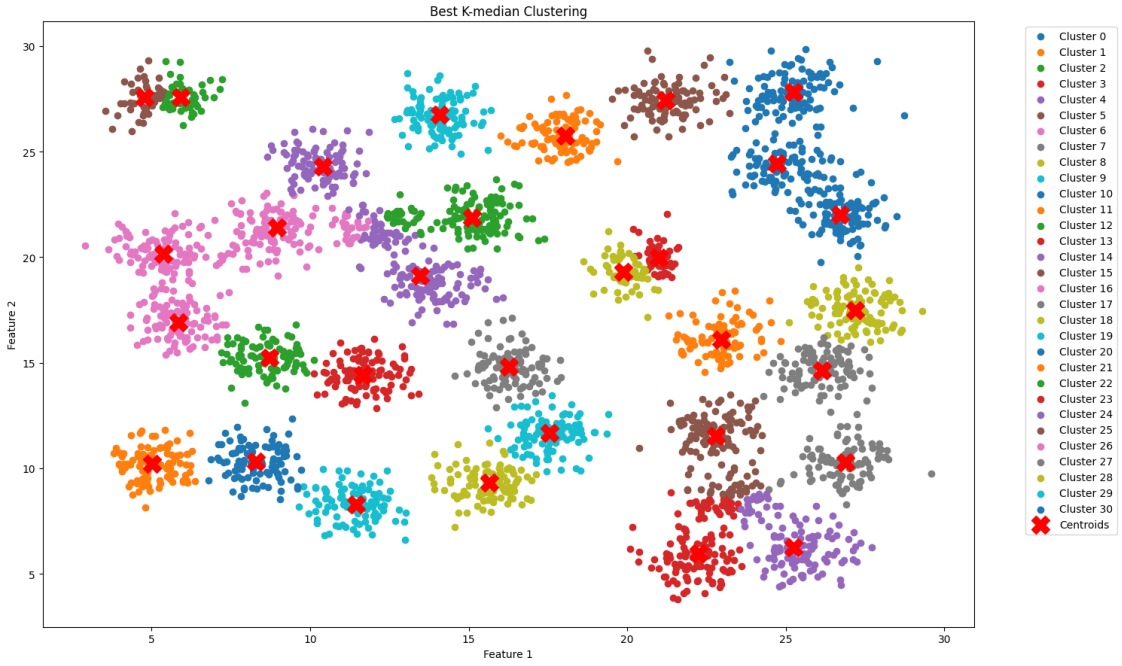


[3]:
```
# Load the data
data_set = load_data("D31.data")

# Determine the number of clusters (assuming the last column is the label)
label_number = len(set(data_set.iloc[:, -1]))

# Extract true labels
true_labels = data_set.iloc[:, -1]

# Run clustering multiple times for K-means
best_clustered_data_kmeans, best_centroids_kmeans, best_purity_kmeans,␣
 ↪iteration_number = run_multiple_times(data_set.iloc[:, :-1], true_labels,␣
 ↪label_number)

# Run clustering multiple times for K-median
best_clustered_data_kmedian, best_centroids_kmedian, best_purity_kmedian,␣
 ↪iteration_number2 = run_multiple_times(data_set.iloc[:, :-1], true_labels,␣
 ↪label_number, use_kmedian=True)

# Print the best results
print(f"D31 Best K-means Purity: {best_purity_kmeans:.4f} iteration:␣
 ↪{iteration_number}")
print(f"D31 Best K-median Purity: {best_purity_kmedian:.4f} iteration:␣
 ↪{iteration_number2}")
```

```
# Plot the best results for K-means
plot_clusters(best_clustered_data_kmeans, best_centroids_kmeans, label_number,␣
 ↪'Best K-means++ Clustering')

# Plot the best results for K-median
plot_clusters(best_clustered_data_kmedian, best_centroids_kmedian,␣
 ↪label_number, 'Best K-median Clustering')
```

```
D31 Best K-means Purity: 0.9129 iteration: 6
D31 Best K-median Purity: 0.9165 iteration: 4
```

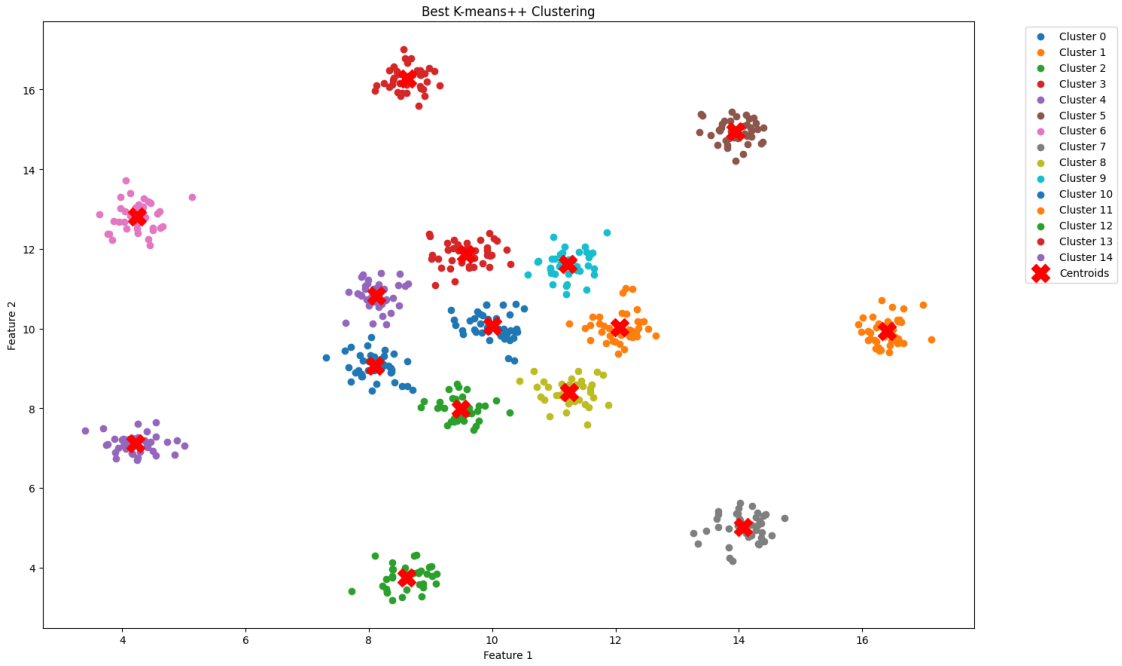


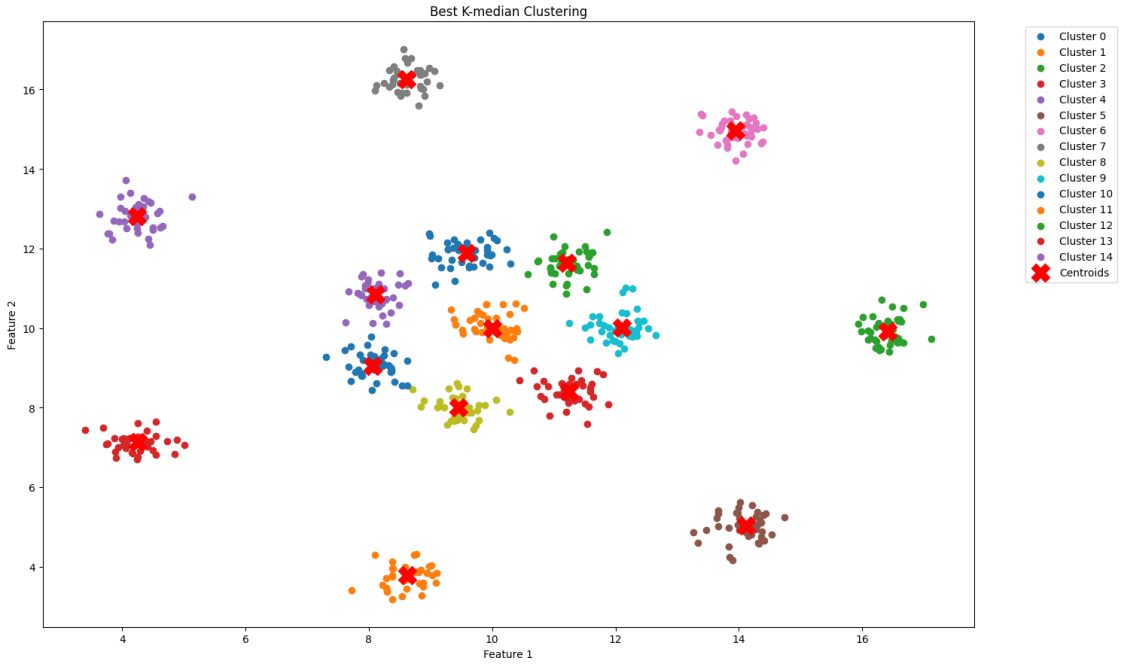


[4]:
```
# Load the data
data_set = load_data("R15.data")

# Determine the number of clusters (assuming the last column is the label)
label_number = len(set(data_set.iloc[:, -1]))

# Extract true labels
true_labels = data_set.iloc[:, -1]

# Run clustering multiple times for K-means
best_clustered_data_kmeans, best_centroids_kmeans, best_purity_kmeans,␣
 ↪iteration_number = run_multiple_times(data_set.iloc[:, :-1], true_labels,␣
 ↪label_number)

# Run clustering multiple times for K-median
best_clustered_data_kmedian, best_centroids_kmedian, best_purity_kmedian,␣
 ↪iteration_number2 = run_multiple_times(data_set.iloc[:, :-1], true_labels,␣
 ↪label_number, use_kmedian=True)

# Print the best results
print(f"R15 Best K-means Purity: {best_purity_kmeans:.4f} iteration:␣
 ↪{iteration_number}")
print(f"R15 Best K-median Purity: {best_purity_kmedian:.4f} iteration:␣
 ↪{iteration_number2}")
```

```
plot_clusters(best_clustered_data_kmeans, best_centroids_kmeans, label_number,␣
 ↪'Best K-means++ Clustering')

# Plot the best results for K-median
plot_clusters(best_clustered_data_kmedian, best_centroids_kmedian,␣
 ↪label_number, 'Best K-median Clustering')
```

```
R15 Best K-means Purity: 0.9967 iteration: 1
R15 Best K-median Purity: 0.9950 iteration: 1
```





[5]:
```
# Load the data
data_set = load_data("glass.data")

# Determine the number of clusters (assuming the last column is the label)
label_number = len(set(data_set.iloc[:, -1]))

# Extract true labels
true_labels = data_set.iloc[:, -1]

# Run clustering multiple times for K-means
best_clustered_data_kmeans, best_centroids_kmeans, best_purity_kmeans,␣
 ↪iteration_number = run_multiple_times(data_set.iloc[:, :-1], true_labels,␣
 ↪label_number)

# Run clustering multiple times for K-median
best_clustered_data_kmedian, best_centroids_kmedian, best_purity_kmedian,␣
 ↪iteration_number2 = run_multiple_times(data_set.iloc[:, :-1], true_labels,␣
 ↪label_number, use_kmedian=True)

# Print the best results
print(f"glass Best K-means Purity: {best_purity_kmeans:.4f} iteration:␣
 ↪{iteration_number}")
print(f"glass Best K-median Purity: {best_purity_kmedian:.4f} iteration:␣
 ↪{iteration_number2}")
```

```
glass Best K-means Purity: 0.5421 iteration: 3
```

```
glass Best K-median Purity: 0.5327 iteration: 3
```

[6]:
```
# Load the data
data_set = load_data("iris.data")

# Determine the number of clusters (assuming the last column is the label)
label_number = len(set(data_set.iloc[:, -1]))

# Extract true labels
true_labels = data_set.iloc[:, -1]

# Run clustering multiple times for K-means
best_clustered_data_kmeans, best_centroids_kmeans, best_purity_kmeans,␣
 ↪iteration_number = run_multiple_times(data_set.iloc[:, :-1], true_labels,␣
 ↪label_number)

# Run clustering multiple times for K-median
best_clustered_data_kmedian, best_centroids_kmedian, best_purity_kmedian,␣
 ↪iteration_number2 = run_multiple_times(data_set.iloc[:, :-1], true_labels,␣
 ↪label_number, use_kmedian=True)

# Print the best results
print(f"iris Best K-means Purity: {best_purity_kmeans:.4f} iteration:␣
 ↪{iteration_number}")
print(f"iris Best K-median Purity: {best_purity_kmedian:.4f} iteration:␣
 ↪{iteration_number2}")
```

```
iris Best K-means Purity: 0.8933 iteration: 1
iris Best K-median Purity: 0.9000 iteration: 1
```

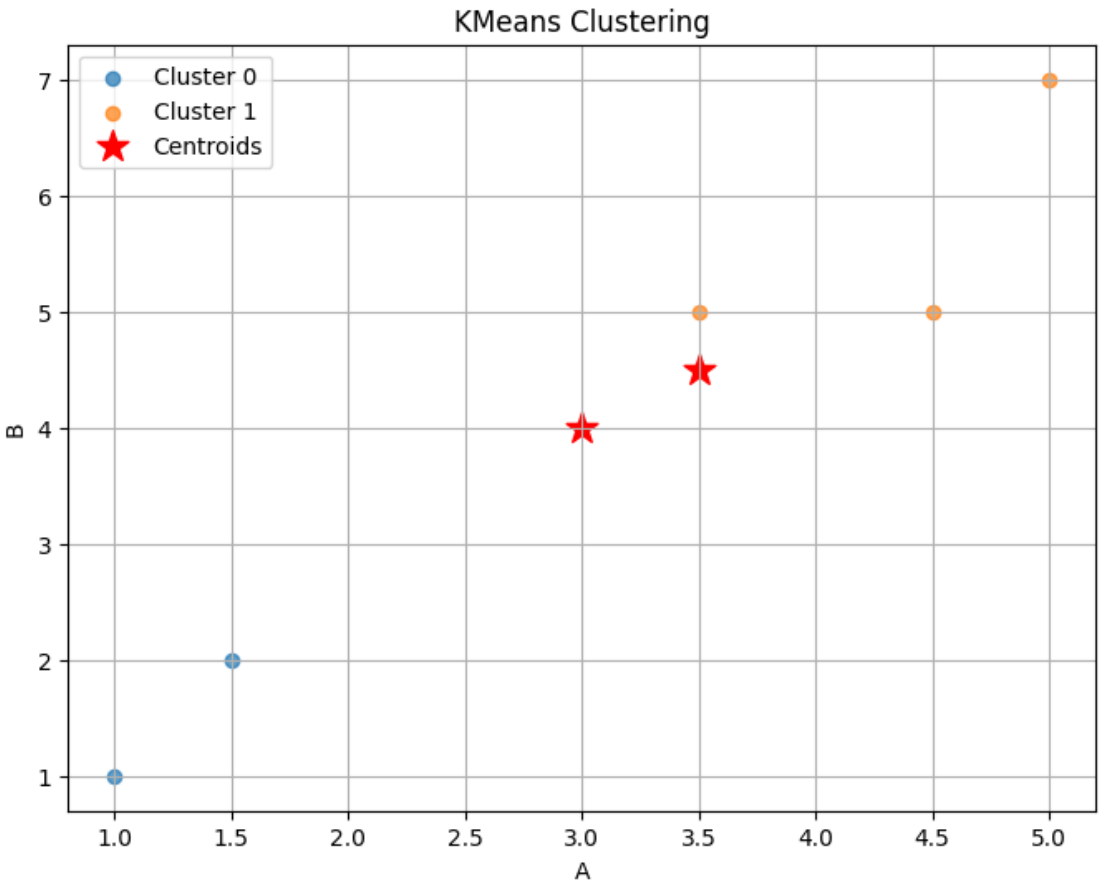
[ ]:
```
```

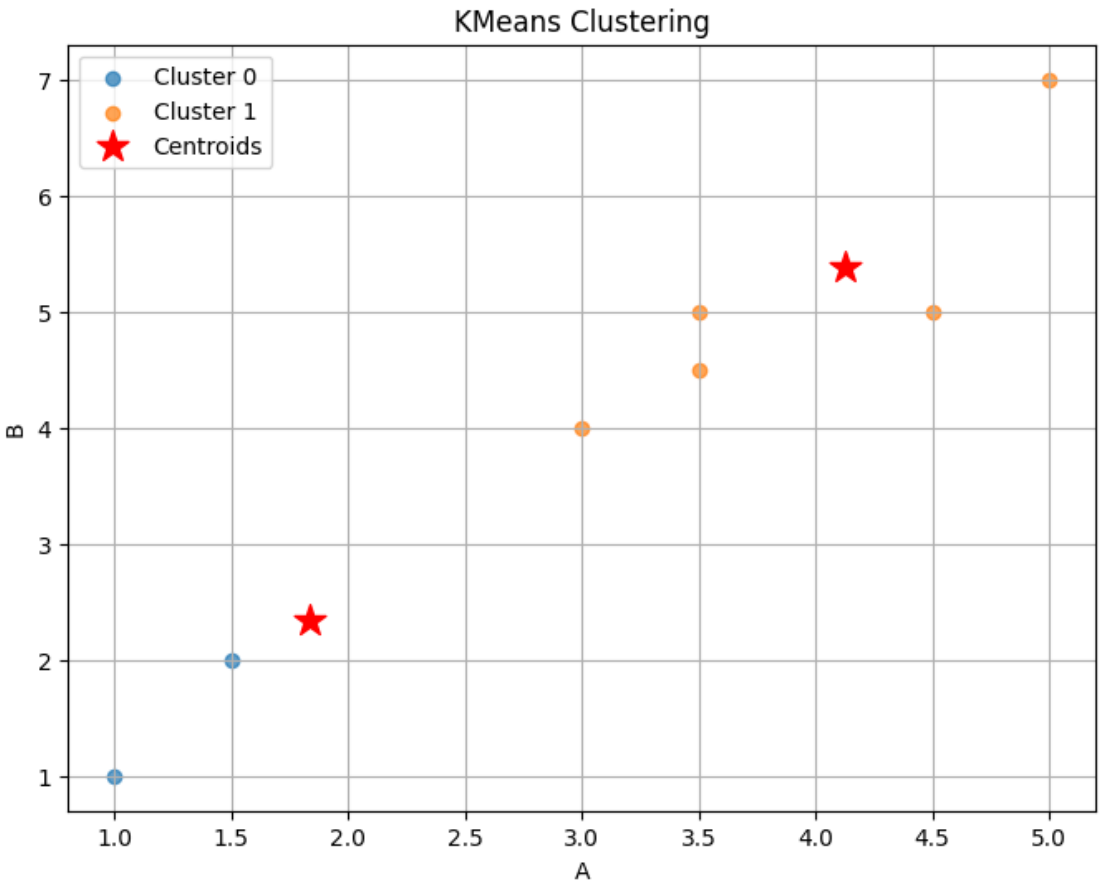
2 Home Work part 2

[7]:
```
import numpy as np
import matplotlib.pyplot as plt

def euclidean_distance(x1, x2):
    return np.sqrt(np.sum((x1 - x2) ** 2))

def kmeans(X, k=2, max_iters=100, plot_steps=False):
    n_samples, n_features = X.shape

    # Initialize centroids
    random_sample_idxs = np.random.choice(n_samples, k, replace=False)
    centroids = [X[idx] for idx in random_sample_idxs]
```

```
    # Optimization
    for _ in range(max_iters):
        # Update clusters
        clusters = create_clusters(X, centroids)
        if plot_steps:
            plot_clusters(X, clusters, np.array(centroids))

        # Update centroids
        centroids_old = centroids
        centroids = get_centroids(X, clusters)

        # Check convergence
        if is_converged(centroids_old, centroids):
            break

    return get_cluster_labels(clusters, n_samples), centroids

def create_clusters(X, centroids):
    clusters = [[] for _ in range(len(centroids))]
    for idx, sample in enumerate(X):
        centroid_idx = closest_centroid(sample, centroids)
        clusters[centroid_idx].append(idx)
    return clusters

def closest_centroid(sample, centroids):
    distances = [euclidean_distance(sample, centroid) for centroid in centroids]
    closest_idx = np.argmin(distances)
    return closest_idx

def get_centroids(X, clusters):
    centroids = np.zeros((len(clusters), X.shape[1]))
    for cluster_idx, cluster in enumerate(clusters):
        if cluster:
            cluster_mean = np.mean(X[cluster], axis=0)
            centroids[cluster_idx] = cluster_mean
    return centroids

def is_converged(centroids_old, centroids):
    distances = [euclidean_distance(centroids_old[i], centroids[i]) for i in␣
 ↪range(len(centroids))]
    return sum(distances) == 0

def get_cluster_labels(clusters, n_samples):
    labels = np.zeros(n_samples)
    for cluster_idx, cluster in enumerate(clusters):
        for sample_idx in cluster:
```

```
            labels[sample_idx] = cluster_idx
    return labels.astype(int)

def plot_clusters(X, clusters, centroids):
    plt.figure(figsize=(8, 6))
    for i, cluster in enumerate(clusters):
        if cluster:
            plt.scatter(X[cluster][:, 0], X[cluster][:, 1], label=f'Cluster␣
 ↪{i}', alpha=0.7)
    plt.scatter(centroids[:, 0], centroids[:, 1], marker='*', s=200, c='red',␣
 ↪label='Centroids')
    plt.xlabel('A')
    plt.ylabel('B')
    plt.title('KMeans Clustering')
    plt.legend()
    plt.grid(True)
    plt.show()

# Define the data
data = {
    'A': [1.0, 1.5, 3.0, 5.0, 3.5, 4.5, 3.5],
    'B': [1.0, 2.0, 4.0, 7.0, 5.0, 5.0, 4.5]
}
X = np.array(list(zip(data['A'], data['B'])))

# Initialize and fit KMeans
cluster_labels, centroids = kmeans(X, k=2, max_iters=2, plot_steps=True)
```


KMeans Clustering
Cluster 0
Cluster 1
Centroids
B
A
1
2
3
4
5
6
7
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0


KMeans Clustering
Cluster 0
Cluster 1
Centroids
B
A
1
2
3
4
5
6
7
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0