Task 1

In [1]:
```
import matplotlib.pyplot as plt
from IPython.display import display, Markdown
import numpy as np
import pandas as pd
from sklearn.metrics.pairwise import pairwise_distances
from sklearn.metrics import accuracy_score
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
class KMeansClustering:

    def __init__(self, k, stopping_criterion="no_change") -> None:
        self.k = k
        self.stopping_criterion = stopping_criterion
        self.centroids = None
        self._sse_score = None
        self._last_sse_score = float('inf')
        self._iterations = 0

    def euclidean_distance(self, data_point, centroids):
        return np.sqrt(np.sum((centroids - data_point)**2, axis=1))

    def __sum_of_squared_errors_calc(self, centroids, data, y):
        sum_of_errors = 0.0
        for idx, d in enumerate(data):
            sum_of_errors += np.sum((centroids[y[idx]] - d) ** 2)

        return sum_of_errors

    def get_sum_of_squared_error(self):
        return self._sse_score
    def get_iterations_to_converge(self):
        return self._iterations

    def fit(self, X, max_iterations=200):
        self.centroids = np.random.uniform(
            low=np.amin(X, axis=0),
            high=np.amax(X, axis=0),
            size=(self.k, X.shape[1]))

        y = []
        for _ in range(max_iterations):
            y = []
            for data_point in X:

                distances = self.euclidean_distance(
                    data_point=data_point,
                    centroids=self.centroids)
                # print(distances.shape)
                cluster_num = np.argmin(distances)
                y.append(cluster_num)
            y = np.asarray(y)

            cluster_indices = []

            for idx in range(self.k):
                cluster_indices.append(np.argwhere(y == idx))

            cluster_centers = []
```

```
            for i, indices in enumerate(cluster_indices):
                if len(indices) == 0:
                    cluster_centers.append(self.centroids[i])
                else:
                    cluster_centers.append(np.mean(X[indices], axis=0)[0])

            if self.stopping_criterion == "no_change" and np.max(self.centroids - np.arr
                break
            elif self.stopping_criterion == "increase_sse":
                current_sse = self.__sum_of_squared_errors_calc(X, np.array(cluster_cent
                if current_sse > self._last_sse_score:
                    break
                self._last_sse_score = current_sse
            else:
                self.centroids = np.array(cluster_centers)
            self._iterations += 1

        # Calculate the final SSE after performing K-means
        self._sse_score = self.__sum_of_squared_errors_calc(X, self.centroids, y)

        return y
```

In [3]:
```
data = np.array(pd.read_csv('datasets/kmeans_data/data.csv', header=None))
labels = np.ravel(pd.read_csv('datasets/kmeans_data/label.csv', header=None))
print('Data: ', data.shape)
print('Labels: ', labels.shape)
```

```
Data:  (10000, 784)
Labels:  (10000,)
```

In [4]:
```
unique_labels = np.unique(labels)
no_of_clusters = unique_labels.size
MAX_ITERATIONS = 100
```

In [5]:
```
euclidean_kmeans_m = KMeansClustering(k=no_of_clusters)
euclidean_kmeans_m_labels = euclidean_kmeans_m.fit(X=data, max_iterations=MAX_ITERATIONS
```

In [6]:
```
np.unique(euclidean_kmeans_m_labels)
```

Out[6]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=int64)
```

In [7]:
```
cosine_distances = pairwise_distances(data, metric='cosine')
cosine_kmeans_m = KMeansClustering(k=no_of_clusters)
cosine_kmeans_m_labels = cosine_kmeans_m.fit(cosine_distances, max_iterations=MAX_ITERAT
```

In [8]:
```
jaccard_distances = pairwise_distances(data, metric='hamming')
jaccard_kmeans_m = KMeansClustering(k=no_of_clusters)
jaccard_kmeans_m_labels = jaccard_kmeans_m.fit(X=jaccard_distances, max_iterations=MAX_I
```

Q1. Run K-means clustering with Euclidean, Cosine and Jarcard similarity. Specify K= the number of categorical values of y (the number of classifications). Compare the SSEs of Euclidean-K-means, Cosine-K-means, Jarcard-K-means. Which method is better?

In [9]:
```
sse_euclidean_m = euclidean_kmeans_m.get_sum_of_squared_error()
sse_euclidean_m
```

Out[9]:
```
40230403.18542118
```

In [10]:
```
sse_cosine_m = cosine_kmeans_m.get_sum_of_squared_error()
sse_cosine_m
```

Out[10]:
```
6112.655685768307
```

In [11]:
```
see_jaccard_m = jaccard_kmeans_m.get_sum_of_squared_error()
see_jaccard_m
```

Out[11]:
```
1390.2873536906732
```

In [12]:
```
text1 = f"**SSE** of Euclidean K-means = {sse_euclidean_m}<br>"
text2 = f"**SSE** of Cosine K-means = {sse_cosine_m}<br>"
text3 = f"**SSE** of Jarcard K-means = {see_jaccard_m}<br>"
display(Markdown(f"{text1}{text2}{text3}"))
```

SSE of Euclidean K-means = 40230403.18542118

SSE of Cosine K-means = 6112.655685768307

SSE of Jarcard K-means = 1390.2873536906732

Looking at the values above I can see Euclidean K-means having the highest SSE, followed by Cosine K-means with the Jarcard K-means havign the lowest SSE.

Q2. Compare the accuracies of Euclidean-K-means Cosine-K-means, Jarcard-K-means. First, label each cluster using the majority vote label of the data points in that cluster. Later, compute the predictive accuracy of Euclidean-K-means, Cosine-K-means, Jarcard-K-means. Which metric is better? (10 points)

In [13]:
```
def label_clusters(labels, true_labels):
    unique_labels = np.unique(true_labels)
    cluster_labels = np.zeros(len(labels), dtype=np.int)
    for cluster in range(no_of_clusters):
        cluster_indices = np.where(labels == cluster)[0]
        cluster_true_labels = true_labels[cluster_indices]
        majority_label = np.argmax([np.sum(cluster_true_labels == label) for label in un
        cluster_labels[cluster_indices] = majority_label
    return cluster_labels

# Label clusters using majority vote
cluster_labels_euclidean = label_clusters(euclidean_kmeans_m_labels, labels)
cluster_labels_cosine = label_clusters(cosine_kmeans_m_labels, labels)
cluster_labels_jaccard = label_clusters(jaccard_kmeans_m_labels, labels)

# Compute predictive accuracy
accuracy_euclidean = accuracy_score(labels, cluster_labels_euclidean)
accuracy_cosine = accuracy_score(labels, cluster_labels_cosine)
accuracy_jaccard = accuracy_score(labels, cluster_labels_jaccard)
```

In [14]:
```
acc_text1 = f"**Accuracy** of Euclidean-K-means =  {accuracy_euclidean * 100}%<br>"
acc_text2 = f"**Accuracy** of Cosine-K-means = {accuracy_cosine * 100}%<br>"
acc_text3 = f"**Accuracy** of Jaccard-K-means = {accuracy_jaccard * 100}%<br>"
display(Markdown(f"{acc_text1}{acc_text2}{acc_text3}"))
```

Accuracy of Euclidean-K-means = 59.38%

Accuracy of Cosine-K-means = 33.839999999999996%

Accuracy of Jaccard-K-means = 26.63%

Based on the accuracy computation of majority vote, Euclidean accuracy seems to perform better.

Q3: Set up the same stop criteria: “when there is no change in centroid position OR when the SSE value increases in the next iteration OR when the maximum preset value (e.g., 500, you can set the preset value by yourself) of iteration is complete”, for Euclidean-K-means, Cosine-K□means, Jarcard-K-means. Which method requires more iterations and times to converge? (10 points)

In [15]:
```
euclidean_iterations = euclidean_kmeans_m.get_iterations_to_converge()
cosine_iterations = cosine_kmeans_m.get_iterations_to_converge()
jarcard_iterations = jaccard_kmeans_m.get_iterations_to_converge()

text_0 = f"Note : Max iterations have been set to 500 and the change in centroid positio
text_1 = f"Iterations to converge for Euclidean-K-means = **{euclidean_iterations}** <br
text_2 = f"Iterations to converge for Cosine-K-means = **{cosine_iterations}** <br>"
text_3 = f"Iterations to converge for Jaccard-K-means = **{jarcard_iterations}** <br>"

display(Markdown(f"{text_0}{text_1}{text_2}{text_3}"))
```

Note : Max iterations have been set to 500 and the change in centroid position is less than 1e-3.
Iterations to converge for Euclidean-K-means = **100**
Iterations to converge for Cosine-K-means = **24**
Iterations to converge for Jaccard-K-means = **12**

From the results we Euclidean K-means take more iterations than Cosine and Jarcard K-means to converge.

Q4: Compare the SSEs of Euclidean-K-means Cosine-K-means, Jarcard-K-means with respect to the following three terminating conditions: (10 points)

- when there is no change in centroid position
- when the SSE value increases in the next iteration
- when the maximum preset value (e.g., 100) of iteration is complete

(a) Check for SSE for Euclidean K-means, Cosine K-means and Jacard K-means when there is no change in centroid position

In [16]:
```
euclidean_kmeans_1 = KMeansClustering(k=no_of_clusters, stopping_criterion="no_change")
euclidean_kmeans_1_predicted_labels = euclidean_kmeans_1.fit(data, max_iterations=MAX_IT
sse_euclidean_kmeans_1 = euclidean_kmeans_1.get_sum_of_squared_error()
print('SSE of Euclidean K-means when there is no change in centroid position =', sse_euc
```

```
SSE of Euclidean K-means when there is no change in centroid position = 53177875.8046999
1
```

In [17]:
```
cosine_kmeans_1 = KMeansClustering(k=no_of_clusters, stopping_criterion="no_change")
cosine_kmeans_1_predicted_labels = cosine_kmeans_1.fit(cosine_distances, max_iterations=
sse_cosine_kmeans_1 = cosine_kmeans_1.get_sum_of_squared_error()
print('SSE of Cosine K-means when there is no change in centroid position =', sse_cosine
```

```
SSE of Cosine K-means when there is no change in centroid position = 3002.5699963699267
```

In [18]:
```
jaccard_kmeans_1 = KMeansClustering(k=no_of_clusters, stopping_criterion="no_change")
jaccard_kmeans_1_predicted_labels = jaccard_kmeans_1.fit(jaccard_distances, max_iteratio
sse_jaccard_kmeans_1 = jaccard_kmeans_1.get_sum_of_squared_error()
print('SSE of Jaccard K-means when there is no change in centroid position =', sse_jacca
```

```
SSE of Jaccard K-means when there is no change in centroid position = 1466.230909814285
```

(b) Check for SSE for Euclidean K-means, Cosine K-means and Jacard K-means when the SSE value increases in the next iteration

In [19]:
```
euclidean_kmeans_2 = KMeansClustering(k=no_of_clusters, stopping_criterion="increase_sse
euclidean_kmeans_2_predicted_labels = euclidean_kmeans_2.fit(data, max_iterations=MAX_IT
sse_euclidean_kmeans_2 = euclidean_kmeans_2.get_sum_of_squared_error()
print('SSE of Euclidean K-means when the SSE value increases in the next iteration =',
      sse_euclidean_kmeans_2)
```

```
SSE of Euclidean K-means when the SSE value increases in the next iteration = 127476952.
23166637
```

In [20]:
```
cosine_kmeans_2 = KMeansClustering(k=no_of_clusters, stopping_criterion="increase_sse")
cosine_kmeans_2_predicted_labels = cosine_kmeans_2.fit(cosine_distances, max_iterations=
sse_cosine_kmeans_2 = cosine_kmeans_2.get_sum_of_squared_error()
print('SSE of Cosine K-means when the SSE value increases in the next iteration =',
      sse_cosine_kmeans_2)
```

```
SSE of Cosine K-means when the SSE value increases in the next iteration = 12155.2830087
76443
```

In [21]:
```
jaccard_kmeans_2 = KMeansClustering(k=no_of_clusters, stopping_criterion="increase_sse")
jaccard_kmeans_2_predicted_labels = jaccard_kmeans_2.fit(jaccard_distances, max_iteratio
sse_jaccard_kmeans_2 = jaccard_kmeans_2.get_sum_of_squared_error()
print('SSE of Jaccard K-means when the SSE value increases in the next iteration =',
      sse_jaccard_kmeans_2)
```

```
SSE of Jaccard K-means when the SSE value increases in the next iteration = 2223.4063286
472156
```

(c) Check for SSE for Euclidean K-means, Cosine K-means and Jacard K-means when the maximum preset value (e.g., 100) of iteration is complete

In [22]:
```
euclidean_kmeans_3 = KMeansClustering(k=no_of_clusters, stopping_criterion="max_iteratio
euclidean_kmeans_3_predicted_labels = euclidean_kmeans_3.fit(data, max_iterations=MAX_IT
sse_euclidean_max_iteration = euclidean_kmeans_3.get_sum_of_squared_error()
print(f'SSE of Euclidean K-means when the maximum preset value {MAX_ITERATIONS} is compl
      sse_euclidean_max_iteration)
```

```
SSE of Euclidean K-means when the maximum preset value 100 is complete = 44918706.507018
97
```

In [23]:
```
cosine_kmeans_3 = KMeansClustering(k=no_of_clusters, stopping_criterion="max_iterations"
cosine_kmeans_3_predicted_labels = cosine_kmeans_3.fit(cosine_distances, max_iterations=
sse_cosine_max_iteration = cosine_kmeans_3.get_sum_of_squared_error()
print(f'SSE of Cosine K-means when the maximum preset value {MAX_ITERATIONS} is complete
      sse_cosine_max_iteration)
```

```
SSE of Cosine K-means when the maximum preset value 100 is complete = 2954.7077619641896
```

In [24]:
```
jaccard_kmeans_3 = KMeansClustering(k=no_of_clusters, stopping_criterion="max_iterations
jaccard_kmeans_3_predicted_labels = jaccard_kmeans_3.fit(jaccard_distances, max_iteratio
sse_jarcard_max_iteration = jaccard_kmeans_3.get_sum_of_squared_error()
print(f'SSE of Jaccard K-means when the maximum preset value {MAX_ITERATIONS} is complet
      sse_jarcard_max_iteration)
```

```
SSE of Jaccard K-means when the maximum preset value 100 is complete = 1575.995552778456
4
```

In [25]:
```
table = f"""
| Algorithm | No Change in Centroid Position | SSE Value Increases in Next Iteration | M
|-----------|--------------------------------|---------------------------------------|
| Euclidean  | {sse_euclidean_kmeans_1}              | {sse_euclidean_kmeans_2}
| Jaccard    |{sse_jaccard_kmeans_1}          |{sse_jaccard_kmeans_2}                  |
| Cosine     |{sse_cosine_kmeans_1}           |{sse_cosine_kmeans_2}                   |
"""
display(Markdown(table))
```

Algorithm	No Change in Centroid Position	SSE Value Increases in Next Iteration	Maximum Preset Value of Iterations
Euclidean	53177875.80469991	127476952.23166637	44918706.50701897
Jaccard	1466.230909814285	2223.4063286472156	1575.9955527784564
Cosine	3002.5699963699267	12155.283008776443	2954.7077619641896