Problem 1

$$\min \sum_i \sum_k \pi_{ik} \cdot ||X_i - \mu_k||^2$$

A)

Prove that E step update on membership (π) achieves the minimum objective given the current centroids (μ)

The E-step in K-means clustering updates the cluster membership of each (π) given the centroids (μ). The objective of K-means is to minimize the within cluster sum of squares. The distance between each datapoint and it's assigned centroid. The E-step assigns the new clusters such that the sum of square is minimized for point X_i.

$\frac{\partial J}{\partial \pi_{ik}} = \sum_i \sum_k \pi_{ik} \cdot ||X_i - \mu_k||^2$

-> π_{ik} = {1 if k = $argmin_j ||X_i$ - $\mu_j||^2$, else 0}

This step ensures that within cluster sum of squares is minimized since each data point is assigned to the centroid closest to it. By minimizing the within cluster sum of squares, the E-step updates (π) to achieve minimum objective given current centroid.

B)

Prove that M step update on centroids (μ) achievess the minimum objective given the current memberships (π)

Proof:

O = $\sum_i \sum_k \pi_{ik} \cdot ||X_i - \mu_k||^2$

$\frac{\partial O}{\partial \mu_k}$ = $\frac{\partial(\sum_i \sum_k \pi_{ik} \cdot ||X_i - \mu_k||^2)}{\partial \mu_k}$

$\frac{\partial O}{\partial \mu_k}$ = $(-2 \cdot \sum_i \cdot ||X_i - \mu_k||^2)$

0 = $(-2) \cdot \sum_i \cdot \pi_{ik} \cdot (X_i - \mu_k)$

0 = $2 \cdot \sum_i (\pi_{ik} \cdot X_i) - (\pi_{ik} \cdot \mu_k)$

0 = $2 \sum_i (\pi_{ik} \cdot X_i) - 2 \sum_i (\pi_{ik} \cdot \mu_k)$

$2 \sum_i (\pi_{ik} \cdot \mu_k) = 2 \sum_i (\pi_{ik} \cdot X_i)$

μ_k = $\frac{2 \sum_i (\pi_{ik} \cdot X_i)}{2 \sum_i (\pi_{ik})}$

μ_k = $\frac{\sum_i (\pi_{ik} \cdot X_i)}{\sum_i (\pi_{ik})}$

C)

This is because Kmeans is not necessarily a convex funtion.

There can be many local minimums along the function curve.

When Kmean randomly initialize the mu or the pi, it is not guaranteed that it's initialized adjacent to the globabl minimum.

And since Kmeans can only go down the curve, it's not gauranteed that the curve will reach to the global minimum

It does not guarantee to converge to the global minimum of the objective function, which is the sum of squared distances between samples and their assigned centroid.

This is due to the non-convex nature of the objective function, meaning that it has multiple local minima and a single global minimum, and KMeans can only guarantee convergence to a local minimum, not necessarily the global one.

Problem 2

In [1]:
```
import numpy as np
from sklearn.metrics import euclidean_distances
from sklearn.metrics import pairwise_distances_argmin

def init_random_centroids(data, k):
    centroids = data.copy()
    np.random.shuffle(centroids)
    return centroids[:k]

def assign_cluster(data, centroids):
    distances = pairwise_distances_argmin(data, centroids, metric = 'euclidean')
    return distances

def update_centroids(data, centroids, cluster_assignment):
    new_centroids = np.zeros(centroids.shape)
    for i in range(centroids.shape[0]):
        data_points = data[cluster_assignment == i]
        new_centroids[i] = np.mean(data_points, axis=0)
    return new_centroids

def kmeans(data, k, max_iter = 100):
    centroids = init_random_centroids(data, k)
    for i in range(max_iter):
        cluster_assignment = assign_cluster(data, centroids)
        centroids = update_centroids(data, centroids, cluster_assignment)
    return centroids, cluster_assignment

def evaluate_purity(data, cluster_assignment, labels):
    n_clusters = len(np.unique(cluster_assignment))
    total_purity = 0
    for i in range(n_clusters):
        cluster = data[cluster_assignment == i]
        cluster_labels = labels[cluster_assignment == i]
        unique_labels, counts = np.unique(cluster_labels, return_counts = True)
        total_purity += np.max(counts)
    return total_purity / data.shape[0]

def loss(data, labels, centroids, cluster_assignment):
    distances = euclidean_distances(data, centroids[cluster_assignment])
    return np.sum(np.min(distances, axis = 1))

def run_kmeans(data, labels, k, max_iter = 100):
    centroids, cluster_assignment = kmeans(data, k, max_iter)
    purity = evaluate_purity(data, cluster_assignment, labels)
    loss_val = loss(data, labels, centroids, cluster_assignment)
    return centroids, cluster_assignment, purity, loss_val

def gini_index(cluster_assignment, labels):
    n_clusters = len(np.unique(cluster_assignment))
    gini_index = 0
    for i in range(n_clusters):
        cluster = cluster_assignment == i
        cluster_labels = labels[cluster]
        unique_labels, counts = np.unique(cluster_labels, return_counts = True)
        prob = counts / cluster_labels.shape[0]
        gini_index += 1 - np.sum(prob**2)
    return gini_index / n_clusters
```

A) MNIST Data

In [2]:
```
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784', version = 1)
data = mnist.data / 255.0
labels = mnist.target.astype(int)
```

In [3]:
```
# For k = 5
centroids, cluster_assignment, purity, loss_val = run_kmeans(data, labels, k = 5, max_iter = 100)
print("Purity:", purity)

gini = gini_index(cluster_assignment, labels)
print("Gini index:", gini)
```

```
Purity: 0.4013285714285714
Gini index: 0.7386591248499118
```

In [4]:
```
# For k = 10
centroids, cluster_assignment, purity, loss_val = run_kmeans(data, labels, k = 10, max_iter = 100)
print("Purity:", purity)

gini = gini_index(cluster_assignment, labels)
print("Gini index:",gini)
```

```
Purity: 0.5987714285714286
Gini index: 0.46830837431349026
```

In [5]:
```
# For k = 20
centroids, cluster_assignment, purity, loss_val = run_kmeans(data, labels, k = 20, max_iter = 100)
print("Purity:", purity)

gini = gini_index(cluster_assignment, labels)
print("Gini index:",gini)
```

```
Purity: 0.7137857142857142
Gini index: 0.3635434133606016
```

B) Fashion Data

In [6]:
```
from sklearn.datasets import fetch_openml
fashion = fetch_openml('Fashion-MNIST', version = 1)
data = fashion.data / 255.0
labels = fashion.target.astype(int)
```

In [7]:
```
# For k = 5
centroids, cluster_assignment, purity, loss_val = run_kmeans(data, labels, k = 5, max_iter = 100)
print("Purity:", purity)

gini = gini_index(cluster_assignment, labels)
print("Gini index:", gini)
```

```
Purity: 0.3823285714285714
Gini index: 0.6973612601329064
```

In [8]:
```
# For k = 10
centroids, cluster_assignment, purity, loss_val = run_kmeans(data, labels, k = 10, max_iter = 100)
print("Purity:", purity)

gini = gini_index(cluster_assignment, labels)
print("Gini index:",gini)
```

```
Purity: 0.5541857142857143
Gini index: 0.47311027118322474
```

In [9]:
```
# For k = 20
centroids, cluster_assignment, purity, loss_val = run_kmeans(data, labels, k = 20, max_iter = 100)
print("Purity:", purity)

gini = gini_index(cluster_assignment, labels)
print("Gini index:",gini)
```

```
Purity: 0.6524142857142857
Gini index: 0.4290915488437658
```

C) 20NG DATA

In [10]:
```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_selection import SelectKBest, chi2

train_set = fetch_20newsgroups(subset = 'train')
train_data = train_set.data
train_label = train_set.target
test_set = fetch_20newsgroups(subset = 'test')
test_data = test_set.data
test_label = test_set.target

# Normalizing the data
vectorizer = TfidfVectorizer(stop_words = 'english')
train_data = vectorizer.fit_transform(train_data)
train_data = np.array(train_data.todense())
```

In [11]:
```
# For k = 5
centroids, cluster_assignment, purity, loss_val = run_kmeans(train_data[:100], train_label[:100], k = 5, max_iter = 1000)
print("Purity:", purity)

gini = gini_index(cluster_assignment, train_label[:100])
print("Gini index:", gini)
```

```
Purity: 0.2
Gini index: 0.871459137498951
```

In [12]:
```
# For k = 10
centroids, cluster_assignment, purity, loss_val = run_kmeans(train_data[:100], train_label[:100], k = 10, max_iter = 1000)
print("Purity:", purity)

gini = gini_index(cluster_assignment, train_label[:100])
print("Gini index:", gini)
```

```
Purity: 0.28
Gini index: 0.7840994687131051
```

In [13]:
```
# For k = 20
centroids, cluster_assignment, purity, loss_val = run_kmeans(train_data[:100], train_label[:100], k = 20, max_iter = 1000)
print("Purity:", purity)

gini = gini_index(cluster_assignment, train_label[:100])
print("Gini index:", gini)
```

```
Purity: 0.4
Gini index: 0.5981527777777778
```