ASHUTOSH KUMAR SINGH
19CS30008

Question 10

(a) In this part, we apply the K-Means algorithm with k = 4, 8 and 12.

(b) Document-cluster association for k = 4, 8 and 12 :

```
Number of clusters, k = 4
Clusters obtained:
Cluster 1: ['Swimming']
Cluster 2: ['European Central Bank', 'Financial technology', 'International Monetary Fund']
Cluster 3: ['Data Science', 'Artificial intelligence']
Cluster 4: ['Linear algebra', 'Basketball', 'Cricket']

Number of clusters, k = 8
Clusters obtained:
Cluster 1: ['Basketball']
Cluster 2: ['European Central Bank']
Cluster 3: ['Cricket']
Cluster 4: ['Financial technology']
Cluster 5: ['Linear algebra']
Cluster 6: ['International Monetary Fund']
Cluster 7: ['Data Science', 'Artificial intelligence']
Cluster 8: ['Swimming']

Number of clusters, k = 12
Clusters obtained:
Cluster 1: ['Swimming', 'Cricket']
Cluster 2: ['Basketball']
Cluster 3: ['European Central Bank']
Cluster 4: ['Data Science', 'Artificial intelligence']
Cluster 5: ['International Monetary Fund']
Cluster 6: ['Linear algebra']
Cluster 7: []
Cluster 8: ['Financial technology']
Cluster 9: []
Cluster 10: []
Cluster 11: []
Cluster 12: []
```

(c) In the given data, we have just 9 documents, so k = 12 is obviously meaningless, because many clusters will remain empty, and most clusters will have only one document, so it will not make much sense what the clusters represent.
k = 8 also is less meaningful because grouping 9 documents into 8 clusters will more or less give one document per cluster, and hence the purpose of clustering will not be fulfilled.
In general, k should be well less than N. So here, **k = 4 comes out to be the best choice**. Using k = 4, we also obtain good clustering results as we can see that similar documents have been grouped together in this case. If we look at the documents, then

we can see that the documents belong to mostly three categories - Sports, Finance and AI/ML, and the clusters in k = 4 come out to be very similar to this grouping. Hence, k = 4 is the best choice.

Also, an important point to note is that while deriving these results, the cluster representatives have been initialized from the data set itself, else with so less training data, random initialization produces many empty clusters and the results are worse.

The code for this problem is attached below.

[14]:
```
import random
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
import wikipedia

MAX_NUM_ITERATIONS = 100
CONVERGENCE_LIMIT = 1e-6
```

[27]:
```
class KMeans():
    # initialize the KMeans object
    def __init__(self, x_train, y_train, num_clusters=3, init_type='choose'):
        self.data = x_train
        self.targets = y_train
        self.num_clusters = num_clusters
        self.sample_size = x_train.shape[0]
        self.feature_size = x_train.shape[1]

        if init_type == 'choose':
            self.centers = np.copy(self.data[np.random.choice(self.sample_size,␣
↪self.num_clusters, replace=(False if self.num_clusters <= self.sample_size␣
↪else True))])
        else:       # init_type == 'random'
            self.centers = np.random.uniform(size=(self.num_clusters, self.
↪feature_size))

        self.prev_centers = np.copy(self.centers)
        self.cluster_labels = np.zeros(self.sample_size, dtype=int)

    # function to get the norm of 2 vectorized feature vectors
    def diff_norm(self, p, q):
        return np.linalg.norm(p - q, ord=2, axis=1)

    # function to assign clusters to data points based on minimum norm
    def assign_clusters(self):
        for i in range(self.sample_size):
            norms = self.diff_norm(self.data[i], self.centers)
            self.cluster_labels[i] = np.argmin(norms)

    # function to update the centers (cluster representatives)
    def update_centers(self):
        self.prev_centers = np.copy(self.centers)
        for curr_cluster in range(self.num_clusters):
            curr_group = self.data[self.cluster_labels == curr_cluster]
            if len(curr_group) != 0:
                self.centers[curr_cluster] = np.mean(curr_group, axis = 0)
            else:
                self.centers[curr_cluster] = np.zeros(self.feature_size)
```

```
    # function to calculate the J_clust value
    def calculate_loss(self):
        return np.mean(np.square(self.diff_norm(self.data, self.centers[self.
 ↪cluster_labels])))

    # function to train the K-Means algorithm
    def train(self, details=True):
        for i in range(MAX_NUM_ITERATIONS):
            self.assign_clusters()
            self.update_centers()
            loss = self.calculate_loss()
            if details:
                print(f"Iteration {i} Loss: {loss}")
                print("----------------------------")
            converged = True
            for curr_cluster in range(self.num_clusters):
                if np.linalg.norm(self.prev_centers[curr_cluster] - self.
 ↪centers[curr_cluster], ord=2) > CONVERGENCE_LIMIT:
                    converged = False
            if converged:
                break
```

[ ]:
```
titles = [
    'Linear algebra',
    'Data Science',
    'Artificial intelligence',
    'European Central Bank',
    'Financial technology',
    'International Monetary Fund',
    'Basketball',
    'Swimming',
    'Cricket'
]

# function to load the decument data
def load_data():
    articles = [wikipedia.page(title, preload=True).content for title in titles]
    vectorizer = TfidfVectorizer(stop_words={'english'})
    x_train = vectorizer.fit_transform(articles).toarray()
    y_train = np.arange(len(titles))

    return (x_train, y_train), vectorizer

# main function to perform all required tasks
def main():
```

```
    random.seed(60)
    np.random.seed(60)
    (x_train, y_train), vectorizer = load_data()
    k = [4, 8, 12]
    loss = []
    for num_clusters in k:
        print("Number of clusters, k = {}".format(num_clusters))
        kmeans = KMeans(x_train, y_train, num_clusters, 'choose')
        kmeans.train(details=False)
        loss.append(kmeans.calculate_loss())
        clusters = [[] for i in range(num_clusters)]
        for i, title in enumerate(titles):
            index = kmeans.cluster_labels[i]
            clusters[index].append(title)
        print("Clusters obtained:")
        for i, cluster in enumerate(clusters):
            print("Cluster {}: {}".format(i + 1, cluster))
        print()

if __name__ == '__main__':
    main()
```