Problem 3

The parsers are very different for the two datasets (text vs images) but you are allowed to use a library/package to do so. These being very very popular research datasets, it should be easy to find appropriate parsers. You can try to normalize each column/feature separately with with one of the following ideas. Do not normalize labels. When normalizing a column, make sure to normalize its values across all datapoints (train, test, validation, etc) for consistency

Typical options for feature values (normalization optional):

- 20NG text row normalization TF(term,doc) / DL (doc). For text is critical to maintain a sparse format due to large number of columns; make sure any value transformation retains the 0 values.
- MNIST : since these images are black and white (and some gray) the pixel values are already in a pre-formatted range [0-255]. They may not require normalization, but perhaps its easier to get the values to have 0 mean instead of 128 mean. Depending on what similarity/distance measure you use, computation of similarity might be easy but the size of the similarity matrix might present a challenge.
- Shift-and-scale normalization: subtract the minimum, then divide by new maximum. Now all values are between 0-1
- Zero mean, unit variance : subtract the mean, divide by the appropriate value to get variance=1

Options for distance/similarity. You are encouraged to use your own implementation to compute the pairwise similarity/distance matrix; but we will accept a library available in Matlab/Java/Python/R.

- cosine or simple dot product (required)
- euclidian distance (required)
- editing distance (required)- use a threshold of tolerance on numerical feature values to asess "the same"
- jaccard similarity(optional)
- Manhattan distance(optional)

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import time
import collections
from sklearn.feature_extraction.text import TfidfVectorizer,CountVectorizer
import mnist
from sklearn.datasets import fetch_20newsgroups
newsgroups_train = fetch_20newsgroups(subset='train')
newsgroups_test = fetch_20newsgroups(subset = 'test')
```

Selecting the vectorizer

In [2]:
```
tfidf_vectorizer = TfidfVectorizer()
count_vectorizer = CountVectorizer()
train_tfidf_matrix = tfidf_vectorizer.fit_transform(newsgroups_train.data)
train_tf_matrix = count_vectorizer.fit_transform(newsgroups_train.data)
test_tfidf_matrix = tfidf_vectorizer.transform(newsgroups_test.data)
test_tf_matrix = count_vectorizer.transform(newsgroups_test.data)
```

In [3]:
```
print(train_tf_matrix.shape, test_tf_matrix.shape)
```

```
(11314, 130107) (7532, 130107)
```

In [4]:
```
print(train_tfidf_matrix.shape, test_tfidf_matrix.shape)
```

```
(11314, 130107) (7532, 130107)
```

Shape of the matrix

In [5]:
```
print("The Shape of tf sparse matrix:",train_tf_matrix.shape)
print("The Shape of tf idf sparse matrix:",train_tfidf_matrix.shape)
print("The Shape of tf sparse matrix:",test_tf_matrix.shape)
print("The Shape of tf idf sparse matrix:",test_tfidf_matrix.shape)
```

```
The Shape of tf sparse matrix: (11314, 130107)
The Shape of tf idf sparse matrix: (11314, 130107)
The Shape of tf sparse matrix: (7532, 130107)
The Shape of tf idf sparse matrix: (7532, 130107)
```

Each row represent the documents and the colunms are word indices.

In tf matrix the word frequency per documents is stored in matrix.

In tfidf matrix the tfidf (tf * idf) value of each word per document is stored in the matrix.

In [6]:
```
#lets look at some raw data of the sparse matrix
print(train_tf_matrix[0:2,0:10].todense())
```

```
[[0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]]
```

In [7]:
```
#lets look at some raw data of the sparse matrix
print(train_tfidf_matrix[50:51,0:100].todense())
```

```
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0.]]
```

Sparsity is a major challenge while dealing with text data

Cosine Similarity for Train Data

In [8]:
```
#importing the relevant library
from sklearn.metrics.pairwise import cosine_similarity
```

In [9]:
```
#start time
start = time.time()

#compute the cosine similarity of each doucment with one another
ng_tf_cs = cosine_similarity(train_tf_matrix,train_tf_matrix)

#end time
end = time.time()

#total time
print("\nThe total Running time for computing cosine simlarity of tf matrix
```

```
The total Running time for computing cosine simlarity of tf matrix in sec
onds: 12.987414121627808
```

In [10]:
```
#start time
start = time.time()

#compute the cosine similarity of each doucment with one another
ng_tfidf_cs = cosine_similarity(train_tfidf_matrix,train_tfidf_matrix)

#end time
end = time.time()

#total time
print("\nThe total Running time  for computing cosine simlarity of tfidf ma
```

```
The total Running time  for computing cosine simlarity of tfidf matrix in
seconds: 13.994904041290283
```

Cosine Similarity for Test Data

In [11]:
```
#start time
start = time.time()

#compute the cosine similarity of each doucment with one another
ng_test_tf_cs = cosine_similarity(test_tf_matrix,train_tf_matrix)

#end time
end = time.time()

#total time
print("\nThe total Running time for computing cosine simlarity of tf matrix
```

```
The total Running time for computing cosine simlarity of tf matrix in sec
onds: 8.332358837127686
```

In [12]:
```
#start time
start = time.time()

#compute the cosine similarity of each doucment with one another
ng_test_tfidf_cs = cosine_similarity(test_tfidf_matrix,train_tfidf_matrix)

#end time
end = time.time()

#total time
print("\nThe total Running time for computing cosine simlarity of tf matrix
```

```
The total Running time for computing cosine simlarity of tf matrix in sec
onds: 8.42072606086731
```

Euclidean Distance for Similarity

In [13]:
```
#importing the relevant library
from sklearn.metrics.pairwise import euclidean_distances
```

In [14]:
```
#start time
start = time.time()

#compute the cosine similarity of each doucment with one another
ng_tf_ed = euclidean_distances(train_tf_matrix)

#end time
end = time.time()

#total time
print("\nThe total Running time for computing similarity by euclidean dista
```

```
The total Running time for computing similarity by euclidean distance of
tf matrix in seconds: 14.707864046096802
```

In [15]:
```
#start time
start = time.time()

#compute the cosine similarity of each doucment with one another
ng_tfidf_ed = euclidean_distances(train_tfidf_matrix)

#end time
end = time.time()

#total time
print("\nThe total Running time for similarity by euclidean distance of tfi
```

```
The total Running time for similarity by euclidean distance of tfidf matr
ix in seconds: 14.818179845809937
```

Euclidean distances for Test Data

In [16]:
```
#start time
start = time.time()

#compute the cosine similarity of each doucment with one another
ng_test_tf_ed = euclidean_distances(test_tf_matrix,train_tf_matrix)

#end time
end = time.time()

#total time
print("\nThe total Running time for computing similarity by euclidean dista
```

```
The total Running time for computing similarity by euclidean distance of
tf matrix in seconds: 9.172903299331665
```

In [17]:

```
#start time
start = time.time()

#compute the cosine similarity of each doucment with one another
ng_test_tfidf_ed = euclidean_distances(test_tfidf_matrix,train_tfidf_matrix

#end time
end = time.time()

#total time
print("\nThe total Running time for similarity by euclidean distance of tfi
```

```
The total Running time for similarity by euclidean distance of tfidf matr
ix in seconds: 9.743610143661499
```

Edit Distance

In [29]:

```
def levenshtein_distance(s1, s2):
    if len(s1) < len(s2):
        return levenshtein_distance(s2, s1)

    if len(s2) == 0:
        return len(s1)

    previous_row = range(len(s2) + 1)
    for i, c1 in enumerate(s1):
        current_row = [i + 1]
        for j, c2 in enumerate(s2):
            insertions = previous_row[j + 1] + 1
            deletions = current_row[j] + 1
            substitutions = previous_row[j] + (c1 != c2)
            current_row.append(min(insertions, deletions, substitutions))
        previous_row = current_row

    return previous_row[-1]
```

In [ ]:

```
from sklearn.metrics import pairwise_distances
from sklearn.feature_extraction.text import CountVectorizer

# Vectorizing the text data into a sparse matrix
vectorizer = CountVectorizer()
vectors = vectorizer.fit_transform(newsgroups_train.data).toarray()

# Creating an array to store the edit distances
edit_distances = np.zeros((len(vectors), len(vectors)))

# Computing the edit distance for each pair of documents
for i in range(len(vectors)):
    for j in range(i, len(vectors)):
        distance = levenshtein_distance(vectors[i], vectors[j])
        edit_distances[i, j] = distance
        edit_distances[j, i] = distance
```

Training Accuracy

In [18]:

```
#accuracy on training data using cosine similarity and tfidf vector
k = 5
sum = 0
for i in range(0,len(newsgroups_train.data)):
    similar_index =  np.argsort(ng_tfidf_cs[i])[:-(k+1):-1].tolist()
    l = newsgroups_train.target[similar_index].tolist()
    label = max(l,key=l.count)
    actual_label = newsgroups_train.target[i]
    if label == actual_label:
        sum += 1
print("training accuracy",sum/len(newsgroups_train.data))
```

```
training accuracy 0.9165635495845855
```

In [19]:

```
#accuracy on training data using cosine similarity and tfidf vector
k = 10
sum = 0
for i in range(0,len(newsgroups_train.data)):
    similar_index =  np.argsort(ng_tfidf_cs[i])[:-(k+1):-1].tolist()
    l = newsgroups_train.target[similar_index].tolist()
    label = max(l,key=l.count)
    actual_label = newsgroups_train.target[i]
    if label == actual_label:
        sum += 1
print("training accuracy",sum/len(newsgroups_train.data))
```

```
training accuracy 0.8680395969595192
```

In [20]:
```
#accuracy on training data using cosine similarity and tf vector
k = 5
sum = 0
for i in range(0,len(newsgroups_train.data)):
    similar_index =  np.argsort(ng_tf_cs[i])[:-(k+1):-1].tolist()
    l = newsgroups_train.target[similar_index].tolist()
    label = max(l,key=l.count)
    actual_label = newsgroups_train.target[i]
    if label == actual_label:
        sum += 1
print("training accuracy",sum/len(newsgroups_train.data))
```

```
training accuracy 0.8656531730599257
```

In [21]:
```
#accuracy on training data using euclidean distances and tfidf vector
k = 5
sum = 0
for i in range(0,len(newsgroups_train.data)):
    similar_index =  np.argsort(ng_tfidf_ed[i])[:k].tolist()
    l = newsgroups_train.target[similar_index].tolist()
    label = max(l,key=l.count)
    actual_label = newsgroups_train.target[i]
    if label == actual_label:
        sum += 1
print("training accuracy",sum/len(newsgroups_train.data))
```

```
training accuracy 0.9165635495845855
```

In [22]:
```
#accuracy on training data using euclidean distances and tf vector
k = 5
sum = 0
for i in range(0,len(newsgroups_train.data)):
    similar_index =  np.argsort(ng_tf_ed[i])[:k].tolist()
    l = newsgroups_train.target[similar_index].tolist()
    label = max(l,key=l.count)
    actual_label = newsgroups_train.target[i]
    if label == actual_label:
        sum += 1
print("training accuracy",sum/len(newsgroups_train.data))
```

```
training accuracy 0.8793530139649991
```

Test Accuracy

In [23]:
```
#accuracy on test data using cosine similarity and tfidf vector
k = 5
sum = 0
for i in range(0,len(newsgroups_test.data)):
    similar_index =  np.argsort(ng_test_tfidf_cs[i])[:-(k+1):-1].tolist()
    l = newsgroups_train.target[similar_index].tolist()
    label = max(l,key=l.count)
    actual_label = newsgroups_test.target[i]
    if label == actual_label:
        sum += 1
print("training accuracy",sum/len(newsgroups_test.data))
```

```
training accuracy 0.6755177907594264
```

In [24]:
```
#accuracy on test data using cosine similarity and tf vector
k = 5
sum = 0
for i in range(0,len(newsgroups_test.data)):
    similar_index =  np.argsort(ng_test_tf_cs[i])[:-(k+1):-1].tolist()
    l = newsgroups_train.target[similar_index].tolist()
    label = max(l,key=l.count)
    actual_label = newsgroups_test.target[i]
    if label == actual_label:
        sum += 1
print("training accuracy",sum/len(newsgroups_test.data))
```

```
training accuracy 0.4305629314922995
```

In [25]:
```
#accuracy on test data using euclidean distances and tfidf vector
k = 5
sum = 0
for i in range(0,len(newsgroups_test.data)):
    similar_index =  np.argsort(ng_test_tfidf_ed[i])[:k].tolist()
    l = newsgroups_train.target[similar_index].tolist()
    label = max(l,key=l.count)
    actual_label = newsgroups_test.target[i]
    if label == actual_label:
        sum += 1
print("training accuracy",sum/len(newsgroups_test.data))
```

```
training accuracy 0.6755177907594264
```

In [26]:
```
#accuracy on test data using euclidean distances and tf vector
k = 5
sum = 0
for i in range(0,len(newsgroups_test.data)):
    similar_index =  np.argsort(ng_test_tf_ed[i])[:k].tolist()
    l = newsgroups_train.target[similar_index].tolist()
    label = max(l,key=l.count)
    actual_label = newsgroups_test.target[i]
    if label == actual_label:
        sum += 1
print("training accuracy",sum/len(newsgroups_test.data))
```

```
training accuracy 0.38382899628252787
```

In [27]:
```
#Print labels
similar_index = np.argsort(ng_test_tf_cs[123])[:5].tolist()
print(similar_index)
l = newsgroups_train.target[similar_index].tolist()
print(l)
for i in l :
    print(newsgroups_train.target_names[i])
label = max(l,key=l.count)
print(label)
actual_label = newsgroups_train.target[i]
print(actual_label)
```

```
[8665, 4772, 9080, 2931, 4495]
[2, 2, 2, 2, 2]
comp.os.ms-windows.misc
comp.os.ms-windows.misc
comp.os.ms-windows.misc
comp.os.ms-windows.misc
comp.os.ms-windows.misc
2
4
```

Problem 4

Train and test KNN classification (supervised)

Some datasets might come organized into train/test in which case we respect that. Other datasets come without this organization in which case we randomly ("random" here is very important, data must be shuffled) pick about 80% of data as training , 10% as validation (also used in training) and 10% as testing data (completely unavailable to training)

For each of the two datasets, now in matrix format and with pairwise similarity computed, train and test KNN classification. Report both training performance and testing performance. You are required to implement KNN yourself, but can youse suport libraries and data-structures.

In [*]:

```
import numpy as np
from collections import Counter
from sklearn.model_selection import train_test_split
from sklearn.datasets import fetch_openml

class KNN:
    def __init__(self, k=3):
        self.k = k

    def fit(self, X, y):
        self.X_train = X
        self.y_train = y

    def predict(self, X):
        predicted_labels = [self._predict(x) for x in X]
        return np.array(predicted_labels)

    def _predict(self, x):
        # Compute distances between x and all examples in the training set
        distances = [euclidean_distance(x, x_train) for x_train in self.X_t
        # Sort by distance and return indices of the first k neighbors
        k_indices = np.argsort(distances)[:self.k]
        # Extract the labels of the k nearest neighbor training samples
        k_nearest_labels = [self.y_train[i] for i in k_indices]
        # return the most common class label
        most_common = Counter(k_nearest_labels).most_common(1)
        return most_common[0][0]

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



# Load the dataset
mnist = fetch_openml('mnist_784', version=1)
images, labels = mndata.load_training()

# The MNIST dataset loaded via the mnist package is a list, convert it to n
images = np.array(images)
labels = np.array(labels)

# Rescale the images data to values between 0 and 1
images = images / 255.0

# Now your previous code should work
X_train, X_test, y_train, y_test = train_test_split(images, labels, test_si
knn = KNN(k=3)
knn.fit(X_train, y_train)
predictions = knn.predict(X_test)

# Print accuracy
print("KNN Test Accuracy: ", np.sum(predictions == y_test) / len(y_test))
```

```
/usr/local/lib/python3.10/site-packages/sklearn/datasets/_openml.py:968:
FutureWarning: The default value of `parser` will change from `'liac-arf
f'` to `'auto'` in 1.4. You can set `parser='auto'` to silence this warni
ng. Therefore, an `ImportError` will be raised from 1.4 if the dataset is
dense and pandas is not installed. Note that the pandas parser may return
different data types. See the Notes Section in fetch_openml's API doc for
details.
  warn(
```