```python
import numpy as np
import matplotlib.pyplot as plt
import os, os.path
import re
from collections import Counter
from random import seed
from random import randrange
from tabulate import tabulate
from sklearn.neighbors import KNeighborsClassifier
from sklearn.utils import shuffle
from sklearn.metrics import accuracy_score
from sklearn.naive_bayes import MultinomialNB
from sklearn import svm
from scipy.sparse import csr_matrix
from sklearn.model_selection import KFold
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import classification_report
```

```python
import pickle
import time

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import TruncatedSVD
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import Normalizer
from sklearn.neighbors import KNeighborsClassifier

```

```python
from google.colab import drive
drive.mount('/content/drive')
# !rm -r /content/drive/My\ Drive/Pattern\ Recognition/Project/dataset_large
# !apt-get install zip unzip
# !unzip /content/drive/My\ Drive/Pattern\ Recognition/Project/dataset_large.zip -d /conte
# # *****      IF YOU WANT TO IMPORT DATASET FROM LOCAL MACHINE KINDLY PASS THE FOLDER LOC
datasetPath = "/content/drive/My Drive/Pattern Recognition/Project/dataset_large/"
#datasetPath = "/content/drive/My Drive/Pattern Recognition/Project/dataset_large/"
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mour
```

```python
X_train = []
y_train = []
X_test = []
y_test = []
for files in os.listdir(datasetPath):
  for f in os.listdir(datasetPath+files):
    d = open(datasetPath+files+"/"+f, "r")
    data = str(d.read())
    label = re.search("[\d]*_([\d]*)",f).group(1)
    if files == "train":
      #print(f)
      X_train.append(data)
      if (int(label) == 1):
        y_train.append(1)
      if (int(label) == 2):
        y_train.append(1)
      if (int(label) == 3):
        y_train.append(1)
      if (int(label) == 4):
        y_train.append(1)
      if (int(label) == 7):
        y_train.append(2)
      if (int(label) == 8):
        y_train.append(2)
      if (int(label) == 9):
        y_train.append(2)
      if (int(label) == 10):
        y_train.append(2)


    if files == "test":
      #print(f)
      X_test.append(data)
      if (int(label) == 1):
        y_test.append(1)
      if (int(label) == 2):
        y_test.append(1)
      if (int(label) == 3):
        y_test.append(1)
      if (int(label) == 4):
        y_test.append(1)
      if (int(label) == 7):
        y_test.append(2)
      if (int(label) == 8):
        y_test.append(2)
      if (int(label) == 9):
        y_test.append(2)
      if (int(label) == 10):
        y_test.append(2)
```

```
X_train_raw = np.array(X_train)
y_train  = np.array(y_train)
X_test_raw = np.array(X_test)
y_test = np.array(y_test)
target_names = []
labels = np.unique(y_train)
for label in labels:
  target_names.append('Rating '+str(label))
print(X_train_raw.shape)
print(y_train.shape)
print(X_test_raw.shape)
print(y_test.shape)
```

```
(25000,)
(25000,)
(25000,)
(25000,)
```

```
import nltk
nltk.download('wordnet')
from nltk.stem import WordNetLemmatizer

def clean_dataset(X):
  documents = []
  stemmer = WordNetLemmatizer()
  for sen in range(0, len(X)):

      # Remove all the special characters
      document = re.sub(r'\W', ' ', str(X[sen]))

      # remove all single characters
      document = re.sub(r'\s+[a-zA-Z]\s+', ' ', document)

      # Remove single characters from the start
      document = re.sub(r'\^[a-zA-Z]\s+', ' ', document)

      # Substituting multiple spaces with single space
      document = re.sub(r'\s+', ' ', document, flags=re.I)

      # Removing prefixed 'b'
      document = re.sub(r'^b\s+', '', document)

      # Converting to Lowercase
      document = document.lower()

      # Lemmatization
      document = document.split()

      document = [stemmer.lemmatize(word) for word in document]
      document = ' '.join(document)

      documents.append(document)
  return documents
X_train_raw = clean_dataset(X_train_raw)
X_test_raw = clean_dataset(X_test_raw)
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

```
# Tfidf vectorizer:
```

```
vectorizer = TfidfVectorizer(max_df=0.7, max_features=500,
                             min_df=5, stop_words='english',
                             use_idf=True)

# Build the tfidf vectorizer from the training data ("fit"), and apply it
# ("transform").
X_train_tfidf = vectorizer.fit_transform(X_train_raw)

print("  Actual number of tfidf features: %d" % X_train_tfidf.get_shape()[1])

print("\nPerforming dimensionality reduction using LSA")
t0 = time.time()

# Project the tfidf vectors onto the first N principal components.
# Though this is significantly fewer features than the original tfidf vector,
# they are stronger features, and the accuracy is higher.
svd = TruncatedSVD(100)
lsa = make_pipeline(svd, Normalizer(copy=False))

# Run SVD on the training data, then project the training data.
X_train_lsa = lsa.fit_transform(X_train_tfidf)

print("  done in %.3fsec" % (time.time() - t0))

# explained_variance = svd.explained_variance_ratio_.sum()
# print("  Explained variance of the SVD step: {}%".format(int(explained_variance * 100)))


# Now apply the transformations to the test data as well.
X_test_tfidf = vectorizer.transform(X_test_raw)
X_test_lsa = lsa.transform(X_test_tfidf)
```

```
  Actual number of tfidf features: 500

Performing dimensionality reduction using LSA
  done in 2.135sec
```

```
#scaling vector between 0 - 1
scaler = MinMaxScaler()
scaler.fit(X_train_lsa)
X_train_lsa_scaled =scaler.transform(X_train_lsa)

scaler = MinMaxScaler()
scaler.fit(X_test_lsa)
X_test_lsa_scaled =scaler.transform(X_test_lsa)
```

```
erageK = 0
tMean = 0
an(array,column,folds=5):
= 0
i in range(1,folds+1):
```

```
m = sum + float(array[i][column])
rn round(sum / folds,3)

rmatMean(array):
al highestMean,bestAverageK
0
attedMean = ["Mean"]
i in range(1,20):
= mean(resultTable,i)

rmattedMean.append(m)
 highestMean < m:
nighestMean = m
bestAverageK = i
rn formattedMean
Table = [[]]

Fold(n_splits=5, random_state=None, shuffle=False)
in range(kf.get_n_splits(X_train_tfidf)):
train_index, test_index in kf.split(X_train_tfidf):
split_input_train_dataset , split_label_train_dataset = X_train_tfidf[train_index],y_train|
split_input_test_dataset , split_label_test_dataset = X_train_tfidf[test_index],y_train[tes

ltRow = [j+1]
i in range(1 , 20):
knn = KNeighborsClassifier(n_neighbors=i)
knn.fit(split_input_train_dataset , split_label_train_dataset)
resultRow.append(str(round(accuracy_score(split_label_test_dataset,knn.predict(split_input_
ltTable.append(resultRow)

late average for all value of K

Table.append(formatMean(resultTable))
= tabulate(resultTable, headers=['Folds', "1","2","3","4","5","6","7","8","9","10","11","1
"Classification complete printing results")
table)
f"Highest mean is {highestMean} so the best K using 5 fold cross validation is {bestAverage
```

```
Classification complete printing results
| Folds    |     1 |    2 |     3 |     4 |     5 |    6 |    7 |     8 |     9 |    10
|---------+-------+------+-------+-------+-------+------+------+-------+-------+--------
|          |       |      |       |       |       |      |      |       |       |
| 1        | 0.637 | 0.63 | 0.661 | 0.654 | 0.674 | 0.67 | 0.69 | 0.675 | 0.696 | 0.691
| 2        | 0.637 | 0.63 | 0.661 | 0.654 | 0.674 | 0.67 | 0.69 | 0.675 | 0.696 | 0.691
| 3        | 0.637 | 0.63 | 0.661 | 0.654 | 0.674 | 0.67 | 0.69 | 0.675 | 0.696 | 0.691
| 4        | 0.637 | 0.63 | 0.661 | 0.654 | 0.674 | 0.67 | 0.69 | 0.675 | 0.696 | 0.691
| 5        | 0.637 | 0.63 | 0.661 | 0.654 | 0.674 | 0.67 | 0.69 | 0.675 | 0.696 | 0.691
| Mean     | 0.637 | 0.63 | 0.661 | 0.654 | 0.674 | 0.67 | 0.69 | 0.675 | 0.696 | 0.691
Highest mean is 0.712 so the best K using 5 fold cross validation is 19
```

```python
#Using best k to classify test database
print(f"Classifying using KNN test dataset using best K {bestAverageK}")
clf = KNeighborsClassifier(n_neighbors=bestAverageK)
clf.fit(X_train_lsa , y_train)
knn_accuracy = accuracy_score(y_test, clf.predict(X_test_lsa))
print(f"Test dataset accuracy for best K {bestAverageK} is {knn_accuracy} ")
print(classification_report(y_test, clf.predict(X_test_lsa), target_names=target_names))
```

```
Classifying using KNN test dataset using best K 19
Test dataset accuracy for best K 19 is 0.734
              precision    recall  f1-score   support

    Rating 1       0.72      0.76      0.74     12500
    Rating 2       0.75      0.71      0.73     12500

    accuracy                           0.73     25000
   macro avg       0.73      0.73      0.73     25000
weighted avg       0.73      0.73      0.73     25000
```

```python
print(f"Classifying using multinomialNB")
clf = MultinomialNB()
clf.fit(X_train_lsa_scaled, y_train)
nb_accuracy = accuracy_score(y_test, clf.predict(X_test_lsa_scaled))
print(f"Test dataset accuracy is {nb_accuracy} ")
print(classification_report(y_test, clf.predict(X_test_lsa_scaled), target_names=target_na
```

```
Classifying using multinomialNB
Test dataset accuracy is 0.79676
              precision    recall  f1-score   support

    Rating 1       0.82      0.76      0.79     12500
    Rating 2       0.78      0.83      0.80     12500

    accuracy                           0.80     25000
   macro avg       0.80      0.80      0.80     25000
weighted avg       0.80      0.80      0.80     25000
```

```python
print(f"Classifying using SVM")
clf = svm.SVC()
clf.fit(X_train_lsa_scaled , y_train)
svm_accuracy = accuracy_score(y_test, clf.predict(X_test_lsa_scaled))
print(f"Test dataset accuracy is {svm_accuracy} ")
print(classification_report(y_test, clf.predict(X_test_lsa_scaled), target_names=target_na
```

```
Classifying using SVM
Test dataset accuracy is 0.82336
              precision    recall  f1-score   support
```

```
    Rating 1       0.85      0.78      0.82     12500
    Rating 2       0.80      0.86      0.83     12500

    accuracy                           0.82     25000
   macro avg       0.83      0.82      0.82     25000
weighted avg       0.83      0.82      0.82     25000
```