```
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
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import classification_report
```

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

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
```

```
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 LOCAT
datasetPath = "/content/drive/My Drive/Pattern Recognition/Project/dataset_large/"
```

```
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
```

```
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/

  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
  inflating: /content/drive/My Drive/Pattern Recognition/Project/dataset_large/train/
```

```
X_train = []
X_test = []
y_train = []
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):
        X_test.append(1)
      if (int(label) == 2):
        X_test.append(1)
      if (int(label) == 3):
        X_test.append(1)
      if (int(label) == 4):
        X_test.append(1)
      if (int(label) == 7):
        X_test.append(2)
      if (int(label) == 8):
        X_test.append(2)
      if (int(label) == 9):
        X_test.append(2)
      if (int(label) == 10):
        X_test.append(2)
    if files == "test":
      #print(f)
      y_train.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 = np.array(X_train)
X_test = np.array(y_test)
y_train = np.array(y_train)
y_test = np.array(y_test)
target_names = []
labels = np.unique(X_test)
for label in labels:
  target_names.append('Rating '+str(label))
print(X_train.shape)
print(X_test.shape)
print(y_train.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 = clean_dataset(X_train)
y_train = clean_dataset(y_train)
```

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

```
def transform(array):
    from sklearn.feature_extraction.text import CountVectorizer
    vectorizer = CountVectorizer(max_features=5000, min_df=5, max_df=0.7, stop_words=stopw
```

```
    vectorized = vectorizer.fit_transform(array)
    transformer = TfidfTransformer()
    transformed = transformer.fit_transform(vectorized)
    return np.array(transformed.toarray())

```

```
print("Creating Count Vectors for training data")
X_train = transform(X_train)
print(np.array(X_train).shape)
print("Creating Count Vectors for testing data")
y_train = transform(y_train)
print(np.array(y_train).shape)

```

```
Creating Count Vectors for training data
(25000, 5000)
Creating Count Vectors for testing data
(25000, 5000)
```

```
def pad_along_axis(array: np.ndarray, target_length: int, axis: int = 0):

    pad_size = target_length - array.shape[axis]

    if pad_size <= 0:
        return array

    npad = [(0, 0)] * array.ndim
    npad[axis] = (0, pad_size)

    return np.pad(array, pad_width=npad, mode='constant', constant_values=0)

X_train = pad_along_axis(X_train, y_train.shape[1], axis=1)
```

```
# convert to sparse matrix to dense matrix
print("Converting sparse matrix to dense matrix")
X_train_dense = csr_matrix(X_train)
y_train_dense = csr_matrix(y_train)
```

```
Converting sparse matrix to dense matrix
```

```

print("Shuffling test and training datasets")
X_train_shuffled,X_test_shuffled = shuffle(X_train_dense,X_test)
print(X_train_shuffled.shape)
print(X_test_shuffled.shape)
y_train_shuffled,y_test_shuffled = shuffle(y_train_dense,y_test)
print(y_train_shuffled.shape)
print(y_test_shuffled.shape)

```

```
    Shuffling test and training datasets
    (25000, 5000)
    (25000,)
    (25000, 5000)
    (25000,)
```

```
bestAverageK = 0
highestMean = 0
def mean(array,column,folds=5):
  sum = 0
  for i in range(1,folds+1):
    sum = sum + float(array[i][column])
  return round(sum / folds,3)

def formatMean(array):
  global highestMean,bestAverageK
  m = 0
  formattedMean = ["Mean"]
  for i in range(1,20):
    m = mean(resultTable,i)

    formattedMean.append(m)
    if highestMean < m:
      highestMean = m
      bestAverageK = i
  return formattedMean
resultTable = [[]]

kf = KFold(n_splits=5, random_state=None, shuffle=False)
for j in range(kf.get_n_splits(X_train_shuffled)):
  for train_index, test_index in kf.split(X_train_shuffled):
      split_input_train_dataset , split_label_train_dataset = X_train_shuffled[train_index
      split_input_test_dataset , split_label_test_dataset = X_train_shuffled[test_index],X

  resultRow = [j+1]
  for 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
  resultTable.append(resultRow)

#Calculate average for all value of K

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

```
Classification complete printing results
| Folds   |     1 |     2 |     3 |     4 |     5 |    6 |    7 |     8 |     9 |    10
|---------+-------+-------+-------+-------+-------+------+------+-------+-------+-------
|         |       |       |       |       |       |      |      |       |       |
| 1       | 0.526 | 0.527 | 0.535 | 0.525 | 0.539 | 0.54 | 0.54 | 0.531 | 0.542 | 0.537
| 2       | 0.526 | 0.527 | 0.535 | 0.525 | 0.539 | 0.54 | 0.54 | 0.531 | 0.542 | 0.537
| 3       | 0.526 | 0.527 | 0.535 | 0.525 | 0.539 | 0.54 | 0.54 | 0.531 | 0.542 | 0.537
| 4       | 0.526 | 0.527 | 0.535 | 0.525 | 0.539 | 0.54 | 0.54 | 0.531 | 0.542 | 0.537
| 5       | 0.526 | 0.527 | 0.535 | 0.525 | 0.539 | 0.54 | 0.54 | 0.531 | 0.542 | 0.537
| Mean    | 0.526 | 0.527 | 0.535 | 0.525 | 0.539 | 0.54 | 0.54 | 0.531 | 0.542 | 0.537
Highest mean is 0.55 so the best K using 5 fold cross validation is 15
```

```
1 #using lowest k to classify test database
2 print(f"Classifying using KNN test dataset using best K {bestAverageK}")
3 clf = KNeighborsClassifier(n_neighbors=bestAverageK)
4 clf.fit(X_train_shuffled , X_test_shuffled)
5 knn_accuracy = accuracy_score(y_test_shuffled, clf.predict(y_train_shuffled))
6 print(f"Test dataset accuracy for best K {bestAverageK} is {knn_accuracy} ")
7 print(classification_report(y_test_shuffled, clf.predict(y_train_shuffled), target_names=t
8
```

```
Classifying using KNN test dataset using best K 15
Test dataset accuracy for best K 15 is 0.51252
              precision    recall  f1-score   support

    Rating 1       0.51      0.49      0.50     12500
    Rating 2       0.51      0.53      0.52     12500

    accuracy                           0.51     25000
   macro avg       0.51      0.51      0.51     25000
weighted avg       0.51      0.51      0.51     25000
```

```
1 print(f"Classifying using multinomialNB")
2 clf = MultinomialNB()
3 clf.fit(X_train_shuffled , X_test_shuffled)
4 nb_accuracy = accuracy_score(y_test_shuffled, clf.predict(y_train_shuffled))
5 print(f"Test dataset accuracy is {nb_accuracy}")
6 print(classification_report(y_test_shuffled, clf.predict(y_train_shuffled), target_names=t
7
```

```
Classifying using multinomialNB
Test dataset accuracy is 0.5378
              precision    recall  f1-score   support

    Rating 1       0.56      0.37      0.45     12500
    Rating 2       0.53      0.70      0.60     12500

    accuracy                           0.54     25000
   macro avg       0.54      0.54      0.52     25000
weighted avg       0.54      0.54      0.52     25000
```

```
print(f"Classifying using SVM")
clf = svm.SVC()
clf.fit(X_train_shuffled , X_test_shuffled)
svm_accuracy = accuracy_score(y_test_shuffled, clf.predict(y_train_shuffled))
print(f"Test dataset accuracy is {svm_accuracy} ")
print(classification_report(y_test_shuffled, clf.predict(y_train_shuffled), target_names=t
```

```
Classifying using SVM
Test dataset accuracy is 0.52848
              precision    recall  f1-score   support

    Rating 1       0.53      0.53      0.53     12500
    Rating 2       0.53      0.53      0.53     12500

    accuracy                           0.53     25000
   macro avg       0.53      0.53      0.53     25000
weighted avg       0.53      0.53      0.53     25000
```