```
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
```

Warning: you are connected to a GPU runtime, but not utilizing the GPU. Change to a standard runtime

```
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/"
```

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

```
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)
```

```
      #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(2)
      if (int(label) == 4):
        X_test.append(2)
      if (int(label) == 7):
        X_test.append(3)
      if (int(label) == 8):
        X_test.append(3)
      if (int(label) == 9):
        X_test.append(4)
      if (int(label) == 10):
        X_test.append(4)
    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(2)
```

Warning: you are connected to a GPU runtime, but not utilizing the GPU. Change to a standard runtime

```
        y_test.append(3)
      if (int(label) == 8):
        y_test.append(3)
      if (int(label) == 9):
        y_test.append(4)
      if (int(label) == 10):
        y_test.append(4)
```

```
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
```

Warning: you are connected to a GPU runtime, but not utilizing the GPU. Change to a standard runtime

```
      # 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
```

```
    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
```

Warning: you are connected to a GPU runtime, but not utilizing the GPU. Change to a standard runtime

```

    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)
```

```
10


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


 1 bestAverageK = 0
 2 highestMean = 0
 3 def mean(array,column,folds=5):
 4   sum = 0
 5   for i in range(1,folds+1):
 6     sum = sum + float(array[i][column])
 7   return round(sum / folds,3)
 8
 9 def formatMean(array):
10   global highestMean,bestAverageK
11   m = 0
12   formattedMean = ["Mean"]
13   for i in range(1,20):
14     m = mean(resultTable,i)
15
16     formattedMean.append(m)
17     if highestMean < m:
18       highestMean = m
19       bestAverageK = i
```

Warning: you are connected to a GPU runtime, but not utilizing the GPU. Change to a standard runtime

```
22
23 kf = KFold(n_splits=5, random_state=None, shuffle=False)
24 for j in range(kf.get_n_splits(X_train_shuffled)):
25   for train_index, test_index in kf.split(X_train_shuffled):
26       split_input_train_dataset , split_label_train_dataset = X_train_shuffled[train_index
27       split_input_test_dataset , split_label_test_dataset = X_train_shuffled[test_index],X
28
29   resultRow = [j+1]
30   for i in range(1 , 20):
31       knn = KNeighborsClassifier(n_neighbors=i)
32       knn.fit(split_input_train_dataset , split_label_train_dataset)
33       resultRow.append(str(round(accuracy_score(split_label_test_dataset,knn.predict(split
34   resultTable.append(resultRow)
35
36 #Calculate average for all value of K
37
38 resultTable.append(formatMean(resultTable))
39 table = tabulate(resultTable, headers=['Folds', "1","2","3","4","5","6","7","8","9","10","
40 print("Classification complete printing results")
41 print(table)
42 print(f"Highest mean is {highestMean} so the best K using 5 fold cross validation is {best
43 #bestAverageK =
44 #print(np.array(resultTable))
```

```
44 #print(np.array(resultTable))
```

```
Classification complete printing results
| Folds   |     1 |     2 |     3 |     4 |     5 |     6 |     7 |     8 |     9 |    1
|---------+-------+-------+-------+-------+-------+-------+-------+-------+-------+-----
|         |       |       |       |       |       |       |       |       |       |
| 1       | 0.288 | 0.287 | 0.296 | 0.303 | 0.298 | 0.295 | 0.309 | 0.307 | 0.304 | 0.29
| 2       | 0.288 | 0.287 | 0.296 | 0.303 | 0.298 | 0.295 | 0.309 | 0.307 | 0.304 | 0.29
| 3       | 0.288 | 0.287 | 0.296 | 0.303 | 0.298 | 0.295 | 0.309 | 0.307 | 0.304 | 0.29
| 4       | 0.288 | 0.287 | 0.296 | 0.303 | 0.298 | 0.295 | 0.309 | 0.307 | 0.304 | 0.29
| 5       | 0.288 | 0.287 | 0.296 | 0.303 | 0.298 | 0.295 | 0.309 | 0.307 | 0.304 | 0.29
| Mean    | 0.288 | 0.287 | 0.296 | 0.303 | 0.298 | 0.295 | 0.309 | 0.307 | 0.304 | 0.29
Highest mean is 0.311 so the best K using 5 fold cross validation is 16
```

```
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 16
Test dataset accuracy for best K 16 is 0.27864
              precision    recall  f1-score   support
```

Warning: you are connected to a GPU runtime, but not utilizing the GPU. Change to a standard runtime

```
      Rating 4       0.29      0.29      0.29      7343

      accuracy                           0.28     25000
     macro avg       0.26      0.26      0.25     25000
  weighted avg       0.26      0.28      0.26     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 0.30056
              precision    recall  f1-score   support

      Rating 1       0.36      0.29      0.32      7324
      Rating 2       0.24      0.10      0.14      5176
      Rating 3       0.25      0.21      0.23      5157
      Rating 4       0.30      0.52      0.38      7343
```

```
        accuracy                           0.30     25000
       macro avg       0.29      0.28      0.27     25000
    weighted avg       0.29      0.30      0.28     25000
```

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

```
Classifying using SVM
Test dataset accuracy is 0.3068
              precision    recall  f1-score   support

    Rating 1       0.32      0.41      0.36      7324
    Rating 2       0.22      0.02      0.04      5176
    Rating 3       0.32      0.05      0.09      5157
    Rating 4       0.30      0.58      0.40      7343

    accuracy                           0.31     25000
   macro avg       0.29      0.27      0.22     25000
weighted avg       0.29      0.31      0.25     25000
```

Warning: you are connected to a GPU runtime, but not utilizing the GPU. Change to a standard runtime