HW1 Report

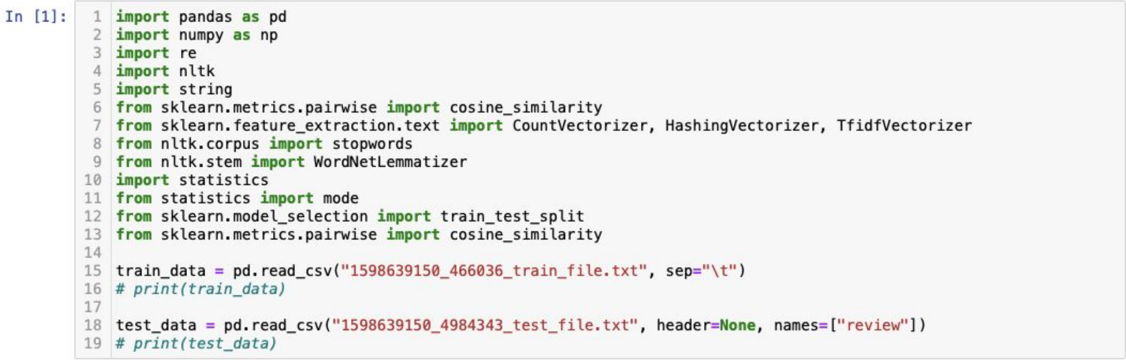
Miner2 username: hkhurana
Mason userID: hkhurana (G01229319)
Best Public Score: 0.83

There were two major steps to completing this assignment:

1. Experimenting locally to find the best value of k
2. Predicting sentiments for the testing dataset using the selected value of k

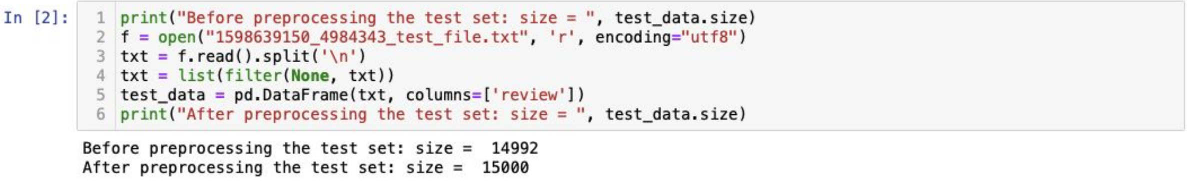
Step 1: Experimenting locally to find the best value of k

First I imported all the essential libraries I would be needing for the program. Then I read the train dataset and test dataset into train_data and test_data.



```
In [1]:
import pandas as pd
import numpy as np
import re
import nltk
import string
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import CountVectorizer, HashingVectorizer, TfidfVectorizer
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import statistics
from statistics import mode
from sklearn.model_selection import train_test_split
from sklearn.metrics.pairwise import cosine_similarity

train_data = pd.read_csv("1598639150_466036_train_file.txt", sep="\t")
# print(train_data)

test_data = pd.read_csv("1598639150_4984343_test_file.txt", header=None, names=["review"])
# print(test_data)
```

However, on reading the test set, I noticed that the size of the test dataset is 14992 which was supposed to be 15000. This is due to some bad spaces in the dataset, so I wrote a small script in order to combat this issue.



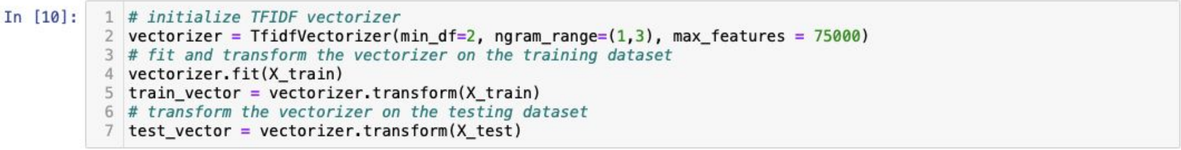
```
In [2]:
print("Before preprocessing the test set: size = ", test_data.size)
f = open("1598639150_4984343_test_file.txt", 'r', encoding="utf8")
txt = f.read().split('\n')
txt = list(filter(None, txt))
test_data = pd.DataFrame(txt, columns=['review'])
print("After preprocessing the test set: size = ", test_data.size)

Before preprocessing the test set: size =  14992
After preprocessing the test set: size =  15000
```

Then I did some checking for null values which turned out to be zero. Post this, I performed dataset cleaning which included converting the dataset to lower case, keeping only alphabetic characters, removing stop words and lemmatization of words.

Next, I split the training set into 80-20 ratios (80% for training, 20% for testing) using the holdout method to assess the accuracy of our model.

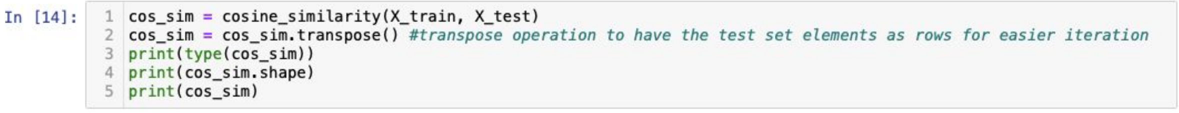
Next, feature selection was done using TFIDF vectorization. The vectorizer had many hyperparameters such as min_df, ngram_range and max_features. On playing around and experimenting with these values, I chose the following values:

- min_df = 2 (to choose words that have occurred at least twice)
- ngram_range = (1,3) (to consider 1 to 3 strings in a row)
- max_features = 75000 (to consider the top 75000 features ordered by term frequency across the corpus)



```
In [10]:
# initialize TFIDF vectorizer
vectorizer = TfidfVectorizer(min_df=2, ngram_range=(1,3), max_features = 75000)
# fit and transform the vectorizer on the training dataset
vectorizer.fit(X_train)
train_vector = vectorizer.transform(X_train)
# transform the vectorizer on the testing dataset
test_vector = vectorizer.transform(X_test)
```

On finding the train and test TFIDF vectors, the next step was to start implementing the KNN algorithm. The first step in the implementation was to find the cosine similarity vector between X_train and X_test. I had many options to measure similarity - cosine similarity, Euclidean distance, etc. but cosine similarity works best for text documents, so I went ahead with it.



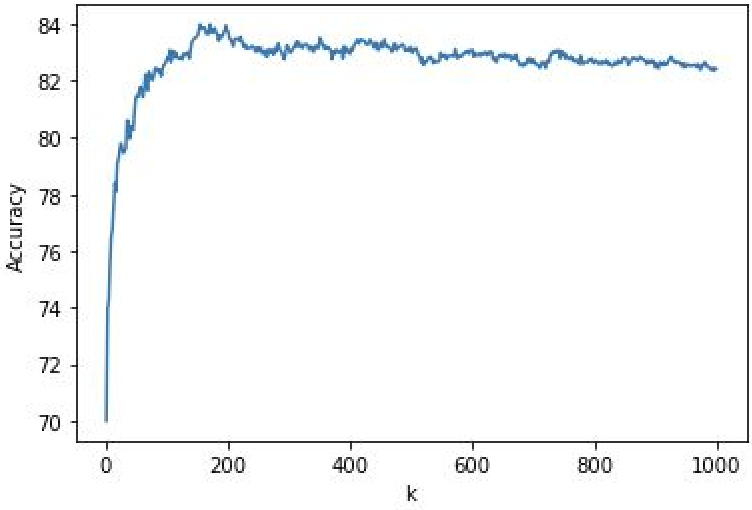
```
In [14]:
cos_sim = cosine_similarity(X_train, X_test)
cos_sim = cos_sim.transpose() #transpose operation to have the test set elements as rows for easier iteration
print(type(cos_sim))
print(cos_sim.shape)
print(cos_sim)
```

Next, I conducted an experiment to find the best value of k. So I ran a loop from k = 1 to 1000 (selecting odd numbers only) and calculated accuracy for each value of k. After iterating through all values of k and finding their accuracies, the following graph was obtained.

The steps performed in the loop are as follows:

- Find k nearest neighbours
- Predict class for test set based on majority voting algorithm

This generated a list of 'k's and accuracies which I used to plot the graph below.



Looking at the graph, it is clear that a peak in accuracy was seen somewhere between 150-250. So I chose this range, and randomly experimented with values and chose k = 241 which worked best for my accuracy on Miner (gave me an accuracy of 83%). According to my local experiment, the max value of k was 155 but it doesn't necessarily mean it'll be the best value for the actual testing dataset.

Step 2: Predicting sentiments for the testing dataset using the selected value of k

After selecting the value of k, the next step was to repeat the algorithm but on the testing dataset. All the steps were the same until cleaning of the datasets. Post the cleaning process, I performed TFIDF vectorization of the training and testing datasets. Then the cleaned and vectorized training and testing datasets were assigned to X_train, X_test and y_train appropriately. After the assignments, I repeated the same old KNN algorithm - calculated cosine similarity, found k nearest neighbours and predicted sentiments using majority voting algorithm. The predicted values were then written to a text file which was then uploaded to Miner.