CS 584- Data Mining and Applications
Kaustubh Prashant Karanjkar
Identifier: **Kaustubh**
#G01314326

HW1: Movie Reviews Classification

On Miner:
Rank: 23
Accuracy: 85%

AIM: Implement the Nearest Neighbor classification algorithm.

Approach:

1. Import the text data into Data Frame and perform pre-processing:

I used existing libraries to clean the text at the pre-processing stage. The data files for the train and test were in .txt format. To clean up the data in those files, I used the *pandas* package to convert them to a dataframe and separated the +1 and -1 into a new column called "Rating."
The next step is to clean the data, which I accomplished with the help of the *nltk* and *re* libraries. *stopwords*, *word_tokenize*, and *WordNetLemmatizer* were all imported from the nltk library. With the help of *word_tokenize* library I was able to separate phrases from the input file into words, the *stopwords* library helped me in removing the English words that doesn't contribute much meaning to a sentence, and *WordNetLemmatizer* helped me in grouping together the various inflected versions of a word to analyze as a single item. Instead of using stemming, I used lemmatization, which does morphological examination of the words.
The re library to build the following regular expression to remove all symbols and digits other than a-z or A-Z.

```
txt=re.sub('[^a-zA-Z]', ' ', test['Review'][i])
```

The above expression will help us in getting rid of all the undesirable punctuations, symbols, and digits like ***". @! # > percent $ & * () -, / 1 2 3 4 5 6 7 8 9"*** that won't help with sentiment analysis.

Next, I have used set data structure to remove all the redundant words from the data, this plays a significant role in dimensionality reduction.

2. Apply Train_Test_Split, TF-IDF and Cosine Similarity method

After cleaning the data set, next step is to train the model with the training data set. I divided the training data set into train data and test data for training the model and calculating the maximum accuracy using the *train_test_split* method imported from *sklearn.model_selection*

```
X_train, X_test, y_train, y_test =
train_test_split(trainCleanData,trainRating, random_state=1,
test_size=0.2)
```

The *trainCleanData* data set is divided into 80% training data (*X_train* and *y_train*) and 20% test data (*X_test and y_test*) using the above method. I have used *random_state* attribute to avoid random splitting of the data. `test_size=0.2,` means I have kept 20% of my data set as test data.
Next, I used *TfidfVectorizer* from the *sklearn.feature_extraction.text* module to quantify the words in the training and test data sets. The term frequency-inverse document frequency (TF-IDF) is a statistical measure that determines how essential or relevant a word is to a document in a set of documents. TF informs us about the frequency with which a term appears in a document, while IDF informs us about the relative rarity of a term in the collection of documents. We can get our final TF-IDF value by multiplying these numbers together. The higher the TF-IDF score, the more important or relevant the term is; the lower the TF-IDF value, the less relevant the term is.

```
X_trainTF = tfidf.fit_transform(X_train)
X_testTF = tfidf.transform(X_test)
```

Then, to scale and standardize the data, I used the fit_transform() and transform() methods on the training and test data sets, respectively.

To compute the similarity between *X_trainTF* and *X_testTF,* I used *cosine_similarity* method which I imported from *sklearn.metrics.pairwise* library. It is the dot product of both the documents.

```
cosSimilarX = cosine_similarity(X_testTF, X_trainTF)
```

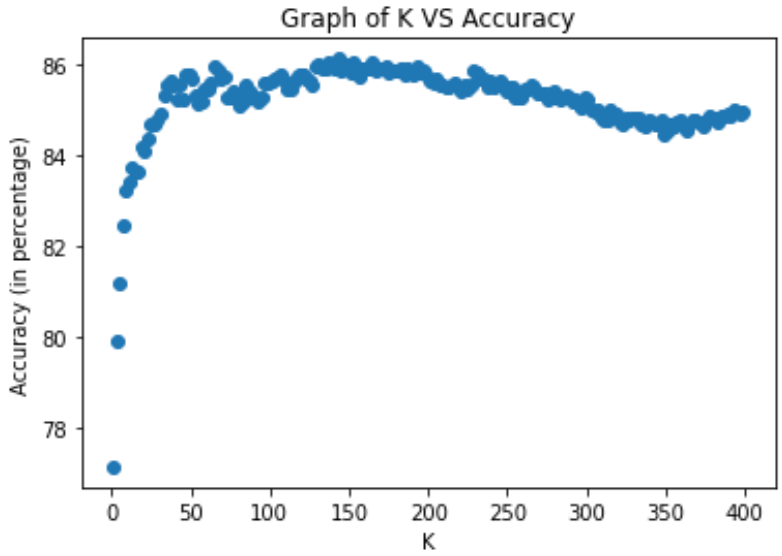
We can calculate by, **cos(*X_textTF,X_trainTF*) = (*X_textTF* • *X_trainTF*) / || *X_textTF* || || *X_trainTF* ||** , where • indicates vector dot product and || || is the length of vector *X_trainTF* and *X_testTF.*

3. KNN Classifier Implementation and Observation

The KNN Classifier Algorithm is implemented using the similarity matrix generated after applying cosine similarity. I used a for loop to go through each row of the similarity matrix (obtained after applying Cosine Similarity method). Then I used the *np.argsort()* method to sort the similarity score in descending order of the first K values, where K is the number of neighbors closest to the data point. Then, to retain a count of '+1' and '-1,' I established two counter variables named *positiveCounter* and *negativeCounter*, and afterwards compared those two variables. The sentiments list will be appended to the one with the greater count.

To calculate the accuracy of the algorithm, I used *metrics.accuracy_score* imported from *sklearn.* Moreover, to find what value of K has the best accuracy of prediction, I inserted the *NearestNeighborClassifier(k, cos_similarity, rating)* method in the for loop with the range from 0 to 400.

For **K = 143**, I got the maximum accuracy of **86.133%**



From the graph above, I can see that accuracy increases until K=143, after which there is a flat curve.