HOMEWORK 1

NAME : ISHANA SHINDE
[redacted]
ACCURACY SCORE: 0.79
RANK: 41(At time of Upload)

README :

1. I used Google Collab Pro(to get more RAM) for this assignment.
2. Use !pip install on Collab to install the python libraries and packages.
3. Upload the training data file train_new.csv and open it as Train_data
4. Upload the test data file 1629732045_8755195_test_new.txt and open it as Test_data in read only mode.
5. Run the cells on Google Collab Pro.

Note: In Cell 3 during POS tagging and lemmatization it throws an error sometimes for that simply comment the !pip install pattern line at the top and run the cell again.

Introduction:

K-Nearest Neighbor is a simple and supervised machine learning model that can be used to solve classification and regression problems. In this assignment we are implementing a basic KNN algorithm to classify a dataset of 25,000 Movie reviews that was provided.

Approach:

With very basic knowledge about data mining I spent the initial days trying to learn about different libraries, what they do and how they are implemented.

I divided the homework into parts -

1. Cleaning data and Preprocessing.
2. K-Fold Cross Validation
3. Implementing KNN.

1. Cleaning Data and Preprocessing:

A. Reading data from the Train and Test files:

Once after opening the file Train_new.csv all the lines are read and stored into the train_lines list.
Each line is read from the train_lines list, converted into lower case and then split based on the (“\t”) parameter. The first element is stored in the Movie_Rating list and the second element is stored in the Train_Movie_Reviews list.

B. Removing Punctuations and Whitespaces:

The Movie Review is then passed to the clean_data where all the punctuations, special characters, html tags and whitespaces are removed. Also the "#EOF" at the end of each review is removed. For this I have used Regular Expressions.
re.sub(re.compile('<.*?>|[^\w+\']|[0-9]'), ' ', x)
re.sub(re.compile('[^\w\s+]'), '', x)

C. Creating Dataframe:

Once the cleaning is done I've created a dictionary that contains the Movie Ratings, Movie Reviews and Cleaned Data. I converted this dictionary to a Dataframe and also dropped missing values (if present in the dataset) and reset their indices.

D. Remove Stopwords:

Next I have removed the Stop Words from the cleaned data set. For that I used the NLTK library to import stopwords. Then I created a set - stop_words that stores all the stopwords as searching in a set is more efficient than searching a list. I applied the function that removes stop words on my Train_df['Cleaned data'].

E. Lemmatization and POS Tagging:

After Stopwords removal, I am applying lemmatization and POS tagging to the cleaned sentences. For this I am using WordNetLemmatizer from NLTK and tag from Pattern.I decided to use lemmatization as it retains the meaning of the word unlike stemming that just removes some of the characters in the end. Along with this I also used POS Tagging which basically returns a tuple with each word and its corresponding tag.

2. **Converting to Vectors**

For this I tried 2 vectorizers - Countvectorizer and TF-IDF Vectorizer. For CountVectorizer I got an accuracy of 0.6801(68%) and for TF-IDF I got an accuracy of 0.7923(79%). So I decided to go ahead with the TF-IDF Vectorizer. Here, it evaluates how relevant a word is to a review with respect to the entire review set. Two metrics here are multiplied: times the word appears in the reivew, and the inverse review frequency of the word across all reviews. This would ensure that words or phrases appearing fewer times would be given more weightage as they would most likely be relevant to their reviews. TF-IDF of each word was calculated, and instead of each word being assigned a count - from the bag of words model - each word was now associated with a "score", a floating point number depicting the relevance of the word to the review in which it was found, with respect to the entire dataset.

3. **Cross Validation**

I used the K-Fold cross validation to check for what KNN-K value, I get approximate 50% accuracy. This Validation I implemented before my own implementation of KNN so for this I used the sklearn KNeighborsClassifier. Here I tried for different values of k and approximately for k = 341, I got 0.8056 accuracy and with the same k value for my Knn implementation I got a score of 0.79 on miner.

4. **KNN Implementation**

For the KNN implementation I have calculated the cosine similarities by using the cosine_similarity function in sklearn.metrics.pairwise. Then I have defined a predict_neighbors function that calculates the majority among the nearest neighbors. It adds all the labels of the neighbors and then checks if the total is greater than 0. If yes then it returns "1" else it returns "-1". For the main KNN for every similar element in the cosine_similarites it will sort in descending order using argsort and then we pick the k neighbors. Then it returns the indices for maximum cosine values and that is passed to the predict_neighbors function. The result from the predict_neighbors function is then stored in the final_prediction variable. Then we run a check if the final_prediction is 1 then we add " +1 " to the final list of predictions else we add a " -1 ". This final list of predictions is written to an output file.

REDUCING RUN TIME :

Steps taken to reduce run time:

1. Removing punctuations and special characters.
2. Removing stop words.
3. Lemmatization.
4. Using Cosine similarity instead of euclidean or any other similarities.
5. Using TF-IDF over CountVectorizer.

CONCLUSION:

The final result of this assignment is a classifier with 79% accuracy. As the dataset was very large and with a large amount of features there is still more room for improvement in terms of feature extraction. There are other methods that can be looked into such as using Spacy instead of NLTK for preprocessing as well as using Doc2Vec instead of TF-IDF for feature extraction. Other methods like PCA for dimensionality reduction can also be looked into.