Narsupalli Mohana Krishna

HW1: Movie Review Classification

Username: rook_in_defence

Rank:53 , Accuracy: 0.83

Introduction:

This Assignment is mainly comprised of three parts:

1. Preprocessing Data 2. Converting to Vectors 3. Building k-Nearest Neighbour model

Preprocessing:

To pre-process the data, popular python libraries like numpy and pandas are used. To reduce overload of pre-processing multiple times, I decided to pre-process the data once so I could easily load and use these files for building my model. After loading Train File using pandas, first thing is to check for noisy data, so I checked for missing Text in Review column or missing labels in sentiment column of Train Dataset. Also checked for columns and their types, count of records. If there would be any such missing data, we need to drop those particular records.

To further proceed, I used nltk library functions such as tokenize for tokenizing, isalpha() to get only strings ,PorterStemmer for stemming and then removed stopwords using nltk function.Then I converted all the strings to lowercase using lower function.So I iterated through whole dataset and after processing each record, I replaced processed record with old record in same index of dataframe.Then for testing, I used train_test_split to split my data into 80% and 20% for training data and testing data respectively. Then my attributes train_X,train_y,test_X,test_y are ready for testing on given original Training Dataset. I also used Data_Cleaning Notebook especially for this preprocessing and the output **is processed_train.txt** for preprocessed train file and **processed_test.txt** for preprocessed test file and I used these two files for further modelling.

Conversion to Vectors:

Now that each each record is processed, we need to convert each record to corresponding Vector. We have options of one-hot-encoding,Tf-Idf Vectorization. I felt, one hot encoding is just representation of words to matrix, and so I have chosen Tf-Idf. In this, higher the value of Tf-Idf of a word, it is considered a prominent. So I have converted both the train and test files to vectors using Tf-Idf Vectorizer and it is also normalized.

Building Model:

After the vectors are ready, the idea is to find distance between one test vector and all other train vectors to get similar scores of each of test vector record. Then using indexing and argsort(), we sort and get top k indices.It means these index records in train file are very close to that particular record in test vector. Since these records are similar, we need to retrieve corresponding sentiment of these records for weighing or voting. So a new list consisting of sentiment values of these corresponding records are loaded.

Now we have both similarity weights and corresponding sentiment value. I felt rather than simply voting for majority sentiments, we can make use of weights. So, I multiplied each of similarity weight with corresponding sentiment and finally summed up them. The result is negative sentiment if the previous sum is negative, else the sentiment is positive and this final result is stored in an array and converted to dataframe and then downloaded as answer file for submission.

Approach and Associated Parameters:

As discussed above, I have used cosine similarity for getting similarity scores of all the closest from train file and weighed them using corresponding sentiment.

Tf-idf over one-hot: I have chosen Tf-Idf,since not every word is given equal importance. The word with more frequency is given less importance, and the word with less frequency is given more importance. Also when checked the accuracy, one-hot encoding is giving much poor results when compared to Tf-Idf vectorizer.

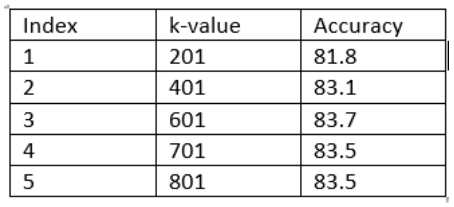
More importantly, I need to talk about **hyper-parameter tuning** for my k-Nearest Neighbour model. Here when using my Tf-Idf Vectorizer, I also used few parameters for tuning.

I checked that few of stop words are still persisting in my train and test file and so I used parameter stopwords='english' to assure stop words get removed. As such, I was getting accuracy score around 78% and then I used **max_df** and **min_df** parameters of TfIdfVectorizer. I played with few values of max_df and min_df and found that accuracy is higher when min_df=0.0015, and when max_df=0.5.

In the next parameter is the value of k. The best value of k is generally chosen around square root of number of samples. So accordingly square root of 15000 is close to125 and so I played with k value around that range and just to check with other possible values of k, I have taken k value up to 700, since both actual train and test files are very large. I found more accurate results around k value of 601.

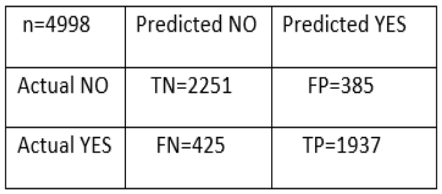
For k-value of 201, I got accuracy of 81.8 and when k-value is 401, it showed accuracy of 83.1.But when tested with k-value of 601, I got accuracy of 83.7 and with k- value of 701, accuracy started dropping and it showed 83.5 accuracy and almost remained the same for k-value of 801.

k-Value Table



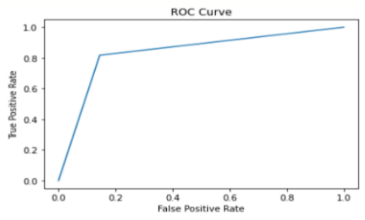
Index	k-value	Accuracy
1	201	81.8
2	401	83.1
3	601	83.7
4	701	83.5
5	801	83.5

Confusion Matrix



n=4998	Predicted NO	Predicted YES
Actual NO	TN=2251	FP=385
Actual YES	FN=425	TP=1937

ROC Curve



The above diagrams shows test result, when I tested original given train dataset by splitting in ratio 33.3 i.e one third is allocated for test set.So n value is 4998 and Confusion Matrix,ROC Curve are for those 4998 records. Then when I performed above process, with Tf-Idf vectorizer for above said values and k value, it showed up accuracy up to 83.7 and so I have chosen these methods and parameters for my actual given test set and performed with afore mentioned process.Also,the auc value is above 80 and so I felt these are good enough metrics for my model.So, I used k-value of 601 in my actual submission k-Nearest Model.

References:

Sklearn documentation: https://scikit-learn.org/stable/

NLTK Library: https://www.nltk.org/

Numpy and Pandas Libraries and Functions, CosineSimilarity function from sklearn