Author - Dipak Meher

Sentiment-Analysis-on-Customer-Reviews-on-Baby-Products

Task:

For the purposes of this assignment we have to use a **logistic regression** classifier to predict the sentiment for 18506 reviews for baby products provided in the test file (test.dat). Positive sentiment is represented by a review rating of +1 and Negative Sentiment is represented by a review rating of -1.

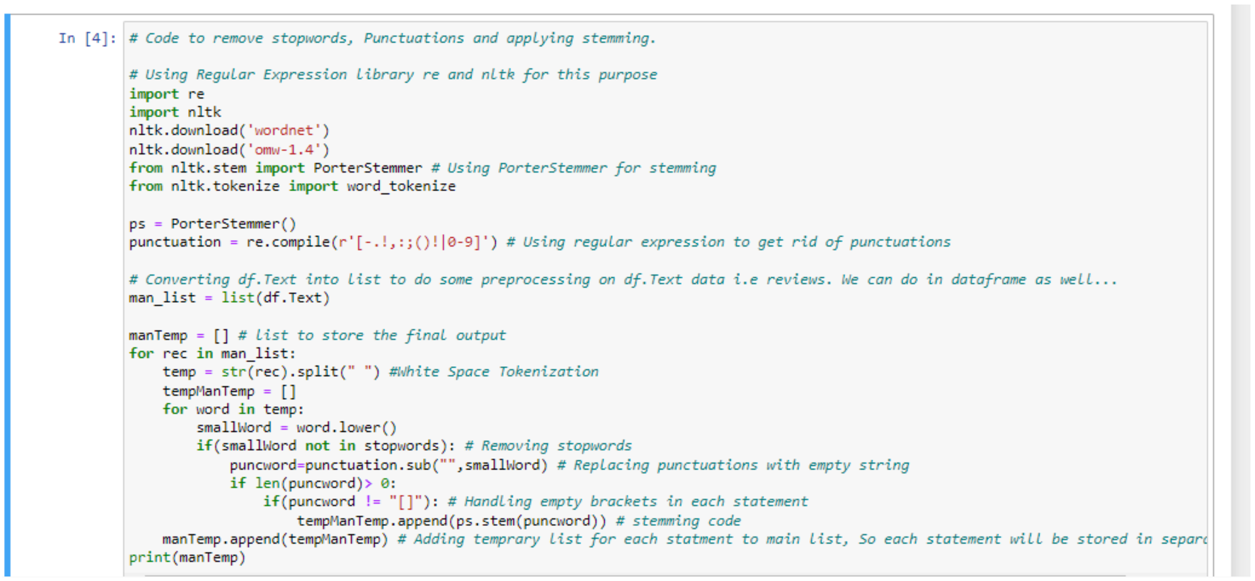
Methodology:

1. Since our data is the reviews on baby products, it is in a string format. We have to preprocess it in order to keep meaningful and impactful content and **remove the irrelevant words** which have minimal impact on the predictions like **Stop word, Punctuations, HTML Tags, etc...** Because we are performing a sentiment analysis here and it is designed to extract information about the sentiment expressed in a text, so any irrelevant information should be removed.

   [Stopwords: Stopwords are words that are commonly used in a language but do not carry much meaning and thus, they do not contribute much to the meaning of the text and can even interfere with the learning of the model.]

   We have used **nltk and re library** to implement the stopword, punctuations from the training data.
2. Then, we have used **Stemming** to reduce words to their base form because it helps to ensure that words are expressed in a consistent manner, which makes it easier for the sentiment analysis algorithm to extract the relevant sentiment information. For example: The root word 'Feed' has been described in many forms in the reviews like feedings, feeds, feed, etc. so stemming reduce these words into "feed" and maintain consistency in data.

   It also help in normalizing the text and thus, maintain consistency in the reviews. We've used **PorterStemmer from nltk** to implement stemming for our reviews.
3. Then, we found that the reviews still contains the irrelevant data in the form of empty brackets like []. We have taken appropriate measures to remove that because such data does not add any value to our sentiment analysis prediction and can create confusion for our model.
   This is the snap of code snippet for 1, 2 and 3 points where we have implemented removal punctuation and stopwords, and applied stemming.



```
In [4]: # Code to remove stopwords, Punctuations and applying stemming.

        # Using Regular Expression library re and nltk for this purpose
        import re
        import nltk
        nltk.download('wordnet')
        nltk.download('omw-1.4')
        from nltk.stem import PorterStemmer # Using PorterStemmer for stemming
        from nltk.tokenize import word_tokenize

        ps = PorterStemmer()
        punctuation = re.compile(r'[-.!,:;()!|0-9]') # Using regular expression to get rid of punctuations

        # Converting df.Text into list to do some preprocessing on df.Text data i.e reviews. We can do in dataframe as well...
        man_list = list(df.Text)

        manTemp = [] # list to store the final output
        for rec in man_list:
            temp = str(rec).split(" ") #White Space Tokenization
            tempManTemp = []
            for word in temp:
                smallWord = word.lower()
                if(smallWord not in stopwords): # Removing stopwords
                    puncword=punctuation.sub("",smallWord) # Replacing punctuations with empty string
                    if len(puncword)> 0:
                        if(puncword != "[]"): # Handling empty brackets in each statement
                            tempManTemp.append(ps.stem(puncword)) # stemming code
            manTemp.append(tempManTemp) # Adding temprary list for each statment to main list, So each statement will be stored in separa
        print(manTemp)
```

4. Sentiment analysis algorithms typically work with numerical data, so text needs to be converted into numerical representations, and this can be done using n-gram or term frequency-inverse document frequency (TF-IDF), in order to be analyzed.

   Since, in question it's been suggested to experiment with bag of words and n-gram representations, with and without TF-IDF weighting, we have also played a bit with that to get model with the highest accuracy. We've implemented 3 models and chosen the one based on their accuracy. The explanation to each model is given in the below points.

5. Train Test Data Split: we've split the training data into train and test data using sklearn library. The ratio of training and testing data is 80:20.

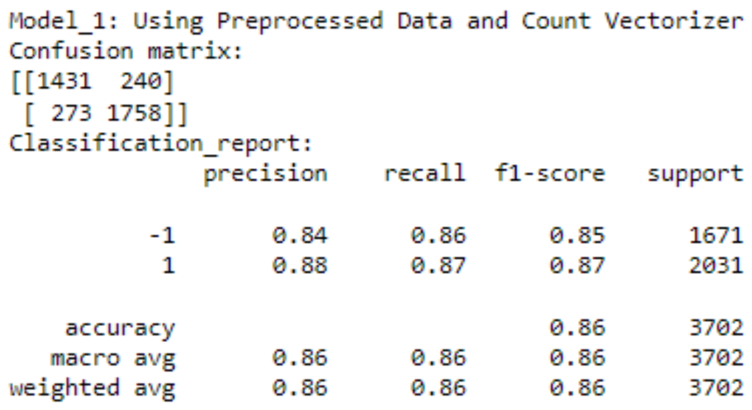
6. Model_1: Model_1 is implemented using CountVectorizer to convert the train and test data into numerical representation. In this case, it converts the text data into vector of terms. It is used to extract features from text by counting the number of times each word appears in the text. This provides a numerical representation of the text that can be used as input to logistic regression model.

   We convert the data into numerical representation using CountVectorizer of sklearn library and then fit the test data using this vectorizer. fit_transform() is used on the training data so that we can scale the training data and also learn the scaling parameters of that data. Here, the model built by us will learn the mean and variance of the features of the training set. These learned parameters are then used to scale our test data. We get train_matrix as a output of fit_transform() method.

   We use transform() method for test data because using the transform method we can use the same mean and variance as it is calculated from our training data to transform our test data. Thus, the parameters learned by our model using the training data will help us to transform our test data. We get test_matrix as a output of a transform() method.

   The fit method is calculating the mean and variance of each of the features present in our data. The transform method is transforming all the features using the respective mean and variance.

Then we use Logistic Regression of sklearn for classification in this assignment with a parameter max_iter = 10000. We train our model on train_matrix and then test onto the test_matrix. We get the accuracy of 86% with this model with the confusion matrix and classification report as follows:

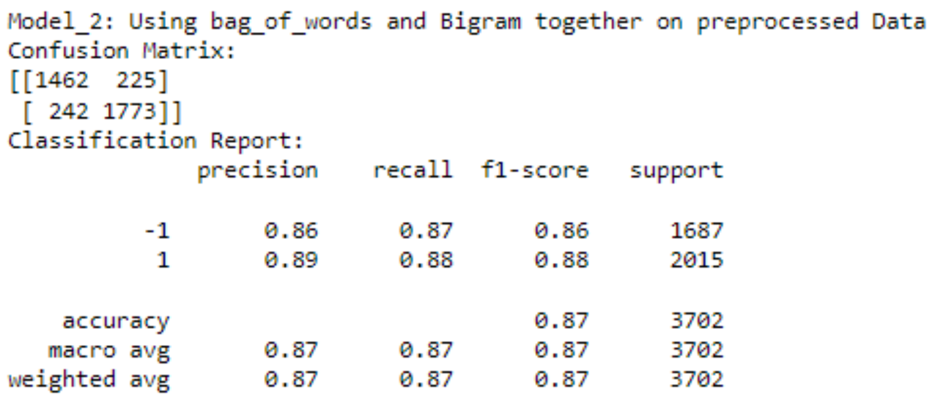


```
Model_1: Using Preprocessed Data and Count Vectorizer
Confusion matrix:
[[1431  240]
 [ 273 1758]]
Classification_report:
              precision    recall  f1-score   support

          -1       0.84      0.86      0.85      1671
           1       0.88      0.87      0.87      2031

    accuracy                           0.86      3702
   macro avg       0.86      0.86      0.86      3702
weighted avg       0.86      0.86      0.86      3702
```

7. Model_2 is built using bag_of_words and bigram together on the preprocessed data.

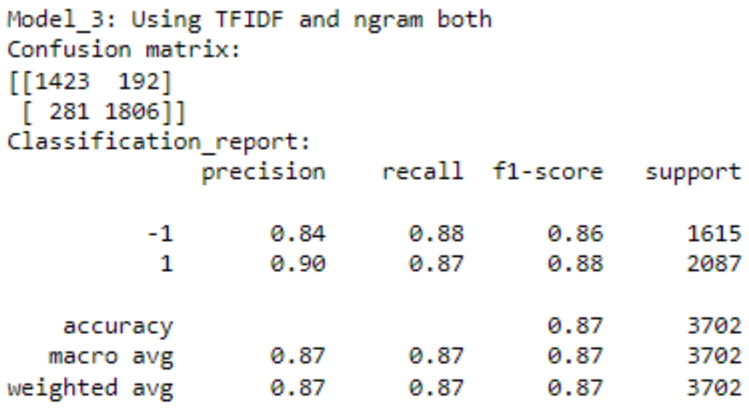
   The basic idea behind the Bag of Words model is to convert a collection of text documents into a numerical representation, where each document is represented by a vector of word frequencies. The drawback of using bag_of_words is, it does not capture the relationship between the data. For example, for the sentence “I love my Father”, it will consider each word count separately and does not take the relationship between these words like love and Father into consideration. Hence, there is chance of losing important information and if we use this information for training then we can end up building less efficient model.

   Thus, we are using Bi-gram as well with bag_of_words, which captures the relationships between words in a reviews by considering two words in order. Using bag of words and bi-gram helps improving the models accuracy. We use Count Vectorizer of sklearn library for implementing bag_of_words and bi-gram. We train the logistic regression classifier with this data and test it on test_matrix. We get the improved accuracy of 87% with the confusion matrix and classification report as follows:



```
Model_2: Using bag_of_words and Bigram together on preprocessed Data
Confusion Matrix:
[[1462  225]
 [ 242 1773]]
Classification Report:
              precision    recall  f1-score   support

          -1       0.86      0.87      0.86      1687
           1       0.89      0.88      0.88      2015

    accuracy                           0.87      3702
   macro avg       0.87      0.87      0.87      3702
weighted avg       0.87      0.87      0.87      3702
```

8. Model_3 is implemented using TF-IDF and n_gram both. We use TfidfVectorizer of sklearn for this and transform the train and test data. We then train the model using train_matrix and make the prediction on our model_3 with a test_matrix. We get accuracy of 87% with the confusion matrix and classification report as follows:

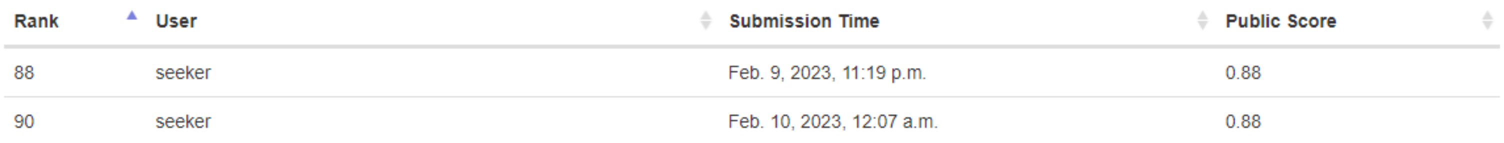


```
Model_3: Using TFIDF and ngram both
Confusion matrix:
[[1423  192]
 [ 281 1806]]
Classification_report:
              precision    recall  f1-score   support

          -1       0.84      0.88      0.86      1615
           1       0.90      0.87      0.88      2087

    accuracy                           0.87      3702
   macro avg       0.87      0.87      0.87      3702
weighted avg       0.87      0.87      0.87      3702
```

9. Since we are getting 87% accuracy with Model_2 and Model_3, we are going ahead with Model_2 for this assignment which is implemented using bag_of_words and bi_gram together.
10. To find the accuracy till now, we have split the train and test data from the train_file.dat and did the necessary computation. Since we have chosen the model now, we will train this model on the entire 18506 record of training data now.
11. Then, we preprocess the data given in test_file.dat. We pass this data through same data preprocessing steps like removing stopwords, punctuations, and applying stemming.

    One thing to note here is, if we use read_csv() function to read the test_file.dat then it will fetch the less number of rows since it will ignore the empty rows. If you continue with your predictions with the reduced rows and upload the prediction file in miner then you will get an error saying "Prediction file has incorrect number of entries". To avoid this, we have read the file using open(test_file, 'r', encoding='utf-8') and then converted it into DataFrame.
12. Then, we've convert this preprocessed test data into numerical matix using transform function and predict the sentiment for the reviews given in the test_file.dat.

    Note: Avoid using fit_tranform function for test_data transformation in this step because it will throw error saying "Feature Mismatch while prediction of test data been done".
13. Then, we have stored the predictions for our test data into the HW1_Meher.dat file and uploaded on Miner. We got the accuracy of 88% i.e. +1 improvement than the accuracy computed on the local machine.



Rank	User	Submission Time	Public Score
88	seeker	Feb. 9, 2023, 11:19 p.m.	0.88
90	seeker	Feb. 10, 2023, 12:07 a.m.	0.88

14. In this way, we have implemented the sentiment analysis using Logistic Regression.