Student Name: Niketan Eshwar Moon
Student Id: a1790186
Course: Applied Natural Language Processing (Semester 1/ 2022)

ANLP Assignment 1
Report For Sentimental Analysis Using Naive Bayes Classifier

Importing the libraries

First I imported the libraries that were required to apply preprocessing techniques, normalization techniques, and for importing the dataset.
Different libraries used include
- re - regular expression
- numpy - for doing mathematical calculations
- Pandas - for reading the dataset
- Matplotlib.pyplot - for plotting and visualizing data
- Sklearn - for performing train test split, for finding confusion matrix
- warnings - to ignore the warnings
- nltk - Imported to use it for text preprocessing. For stop words removal, for tokenization, for lemmatization.

Importing the dataset

The dataset is an “imdb_master.csv” file which is imported using pandas.read_csv method. While importing the dataset, I ran into an encoding error and hence added another parameter encoding=”ISO-8859-1” to read the CSV properly without any error.

Diving deep into the data (Data Exploration)

After importing the dataset, the main focus is to explore the data. For looking how the dataframe looks I used head() function. This gave me insights into the dataframe. I also used info() method to find out the data type of each label and see if there is any missing data to be handled.
The dataset includes
- Unnamed: 0 - contains index which is irrelevant
- type - train and test. This tells data belongs to which set (training or test)
- review - Contains reviews - which is most important for figuring out the sentiment
- label - pos, neg and unsup. pos is positive sentiment, neg is negative sentiment, and unsup is no decision. So this label (unsup) can be ignored.
- File - filename - irrelevant for finding out the sentiment

From this we can say that label is a target variable and review is an independent variable that will be used to analyze the sentiment.
From info() method we can also say that there is no missing data that needs to be handled. The dataset has overall 100000 entries including train and test set and 5 columns.

For finding what the values for type and label were I used value_counts() method and found out that type has two categories train and test (already discussed above) and label has three categories - pos, neg and unsup (already discussed above).

1. A Description of the text-preprocessing techniques Used

Data Preprocessing

The first step to creating useful data out of the data frame is to get rid of the unwanted data.

- I removed unwanted columns **Unnamed: 0** and **file**.
- Removed all the row entries that had unsup as label which was 50000 entries in total.

Text Preprocessing

I wanted to apply all the text preprocessing techniques on both the training and test dataset. So first text preprocessing is applied on all the dataset and then the dataset is separated into training set and test set.

Data Cleaning

Removal of punctuation marks, numbers, removing html tags, urls and then converting the whole text to lower case.

a. Removing punctuation marks - Done using regular expression re library where all the punctuation marks were substituted with no space using re.sub() method. The rule used was **r’[^\w\s]’** which says match no instances of alphanumeric and spaces.
Also used pandas data frame apply() method to apply the functionality on each row of the review column.

b. Removing numbers - Used same re.sub() method with rule as **r'[\d+]'** which says to remove one or more occurrences of digits.

c. Removing html tags - The regular expression used was '<.*?>|&([a-z0-9]+|#[0-9]{1,6}|#x[0-9a-f]{1,6});' such that it also covers some entities like &nsbm which are not particularly inside the html tags.

d. Removing Urls - This was used to remove any occurrence of url from text to make good sense of the data.

e. Convert to lowercase - Used .lower() method to convert the text to lowercase.

Tokenization and stop words removal
Tokenization basically split the text into words separated by a space. I have used TokTokTokenizer a method from nltk library to tokenize the data inside. This tokenizer creates a list of words which is similar to text.split().
Once the list of separate words is created, I wanted to remove words that are stop words. Stop words are words like a, an, the, this, we. These are some common stop words. The goal is to remove such words so that the text data makes much more sense. As the stop words does not play role in finding out the sentiment as these are neutral words.
 But first I needed stop words data. For this I used nltk library to get stopwords and then removed duplicates from that list of stopwords. Now, I looped through tokens list and kept only those words that were not in the stopwords list. If you want to extend the stop words list you can directly append any word to that stopwords list and run the code again to tokenize and remove stopwords.

Lemmatization
The first problem was to choose which technique stemming or lemmatization. Upon further consideration, I used lemmatization because of the below reasons:
Stemming cuts the suffixes.
For Example:

Word	Stemming
Studies	Studi
Studying	Study

As you can see the same root word is converted to two different words which increased computation and adds garbage to the data.

Lets see what happens if I pass the same words to Lemmatization

Word	Lemmatization

Studies	study
Studying	study

As you can see the words were converted to their root forms which reduces the computation time on the data, removes duplicates or similar root words.
Hence I used lemmatization technique. For lemmatizing each word, I need to go through each word so I tokenized the data again.

These were the text preprocessing techniques I used.

Removing duplicates from dataset

There were scenarios were duplicates data can be found. I chose to remove the duplicate rows so that computation is reduced. For removing the duplicates, I used pandas dataframe drop_duplicates method.

2. A description of how to train the Naive Bayes classifier and evaluate the performance

Separating the training set and test set

Now, the full dataset is processed. Now its time to train model on this data. But before training the dataset, the dataset needs to be separated into training set and test so that model is trained only on training dataset and test dataset is used for predicting the model. For this I filtered full_dataset with type == “train” to get the training dataset and type==”test” to get the test dataset.
 Training data is 24902 rows and 3 columns and test dataset is 24798 rows and 3 columns.

Converting the label classes from pos and neg to 1 and 0 respectively

Now after separating the dataset, I converted the labels from pos to neg into 1 and 0 respectively for both training and test dataset. For changing the values np.where to search through the rows and substitute 0 for neg and 1 for pos

After converting the data, I again separated the data to pass to the Naive Bayes classifier into:
X - which will be a list of all the reviews from training set
y - which will be a list of all the labels in the training set
X_test - which will be a list of all the reviews from test set
y_test - which will be a list of all the labels for each review in the test set. This is the actual result.

Creating a Naive Bayes class

For this class the three things that were must was value of k (important for k smoothing), applying fit method and predict method.

fit Method

Step 1: The fit method first calculates the prior probabilities for each class. I created prior_probabilities method.

prior_probabilities method:

- This method takes in X and y.
- First it calculates the count of each class in the whole document.
- Later it counts the total length of documents which is len(X)
- Now it calculates the probability for each class by
  $P(class) \ = \ Count\ of\ class\ in\ the\ whole\ document\ /\ Total\ length\ of\ the\ documents$
- Now these probabilities are called prior probabilities and are saved.

Step 2: For calculating the likelihood for each word with each class, first we need to calculate the count of words for each class
For this I created **get_count_words_with_class** method.

get_count_words_with_class:

- This method loops through the data and calculates count of words in each class for each word
- This method creates a word dictionary
  Word_dict = {
  “word” : [occurence in class 0, occurrence in class 1]
  }

Step 3: The next step is to generate vocabulary. A vocabulary is a list of all the unique words in the data. I created **generate_vocabulary** method to generate a vocabulary.

generate_vocabulary:

- Takes in word dict created in method get_count_words_with_class
- Loops through the dictionary and save the unique word in the list called self.vocab

Step 4 : The next step required is to count the total words in each class. I created **get_total_words_each_class** method to get the count of all words in each class.

get_total_words_each_class:

- This loops through the word dict
- It takes the value of word which is a list and calculates the sum of positive class count and negative class count.
- The way its done is

```
for k, v in word_dict.items():
        pos_class_count = pos_class_count + v[1]
        neg_class_count = neg_class_count + v[0]
```

Step 5: Now we got all the unknowns that were required to calculate the likelihood of each word in different class.
I created **get_conditional_probabilities_with_class** method to calculate the likelihood of each word.
get_conditional_probabilities_with_class:

- It loops through the word dict that was created in Step 2.
- Now it calculates the probability of each word with both the positive and negative classes.
- Formula used:
  $$P(word|class) = \frac{count\ of\ word\ in\ the\ class + k}{total\ words\ in\ each\ class + (len(vocab) * k)}$$
  Here k is the smoothing criteria. This is used to avoid zero probability for a word.

The fit method is complete. This is how we train the data with Naive Bayes classifier. Now for evaluation we use the test dataset and predict in which class each text is in. This is done using predict method

Predict method:

- This method takes in test dataset and iterates over each review.
- Now it splits each review to get the list of words and iterate over it.
- If the word is in the conditional probability of word dictionary that we generated in Step 5, we add the log of each probability to the particular class using np.log method.
- Once it iterates over all the data, now it first converts the log probabilities into actual probability using np.exp() method
- Then we predict the probability using formula
  $$P(class \mid words) = P(class) * P(words|class)$$
- Once the predicted probability is calculated for each class, we compare both the probabilities and predict the class which has higher probability.

This will give you all the predicted values for each review.
Performance Evaluation
 Performance evaluation is done using F1 measure. The higher the f1 score the better is our model. F1 score ranges from 0 to 1.

$$F1\ score = \frac{2 * Precision * Recall}{presion + recall}$$

Now the unknowns are precision and recall

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

For getting the True Positive, True Negative, False Positive and False Negative I used sklearn’s confusion matrix method.

Now once we get all the unknowns, I simply calculated the f1 score using **get_accuracy_and_f1_score** method which calculates the f1 score and accuracy score and returns it. The accuracy score is calculated using the following formula.

$$Accuracy\ Score\ = \frac{True\ Positive + True\ Negative}{True\ Positive + True\ Negative + False\ Positive + False\ Negative}$$

I also did the sanity check for correction of f1 score and accuracy score. For this I used sklearn.metrics f1_score and accuracy_score method to calculate the f1 score and accuracy score. Both the methods evaluated the same score.

3. Factors that affected the classification model
There are different factors that affected performance of the classification model

- **Applying K smoothing on the likelihood probabilities**
  We applied k smoothing while calculating the likelihood probability in the Step 5 of fit method.
  I analysed the effect of k smoothing on the model by simply substitution k=0 in the Naive Bayes class and compared results with k smoothing and without k smoothing.

K smoothing	F1 Score	Accuracy Score
With K smoothing	0.552688530048277	0.674933462375998
Without K smoothing	0.43788551334524695	0.6067828050649245

As you can clearly see, f1 score and accuracy score is reduced when k smoothing is not used. So decided to use k smoothing to calculate the likelihood as the probability may be zero if k smoothing is not used. This is because the count of word in a particular class may be zero and according to likelihood probability without k smoothing this results in zero probability.

- **Applying Lemmatization technique**

Lemmatization converted each word into its root form. But does it really affects the performance of the model. To check this, I analyzed scores when lemmatization is used and when lemmatization is not used.

Lemmatization	Shape of the dataset after dropping the duplicates
Applied	49700, 3
Not Applied	49701

This above analysis tells that without lemmatization (converting the word to the root form), there exist a duplicate row that adds garbage to the model.

Lemmatization	F1 Score	Accuracy Score
With Lemmatization	0.552688530048277	0.674933462375998
Without Lemmatization	0.5411923055353037	0.6701076656316787

As you can see, its a slight change in score but applying lemmatization technique definitely increases the performance of the model.

- Applying stop words removal technique
  Removing the stop words from the data is the correct way. This makes sense because it is always good to remove common english words that dont add meaning to the prediction. Since these are neutral words keeping them may increase the computation time and adds confusion to the model. To test this I trained my model without stop words removal and with stop words removal.

Stop Words Removal	Training Time	Prediction time	F1 Score	Accuracy Score
Applied	676 ms	2.48 s	0.552688530048277	0.674933462375998
Not Applied	997 ms	4.57 s	0.23180479596129575	0.5582258064516129

We can observe from the above table that it takes more time to train, almost double time to predict and it reduces the performance severely if we dont remove stop words from the data. This is why its recommended to use stop words removal to improve the performance of the model.