Programming Assignment – 3

Documentation

Sentiment Classification with Naive Bayes

Team_5

NehaReddy Yenugu

Saketh Kallepalli

Ashritha Gugire

Date: 4/8/2024

Overview of the assignment:

This Assignment aims to perform sentiment analysis on tweets on airline reviews using Naive Bayes. The dataset consists of airline reviews categorized into two classes: positive and negative, with each of 4182 reviews for both train and test set.

We aim to build a complete system from scratch to categorize airline reviews as positive or negative. Through meticulous data preprocessing, vocabulary creation, and feature extraction, the system trains on the training set and evaluates its efficacy on the test set.

Software Requirements:

Platform used of execution: Jupyter Notebook(Python)

Required Python libraries: NumPy, NLTK, BeautifulSoup.

Specifically, below described is entire assignment in 3 steps:

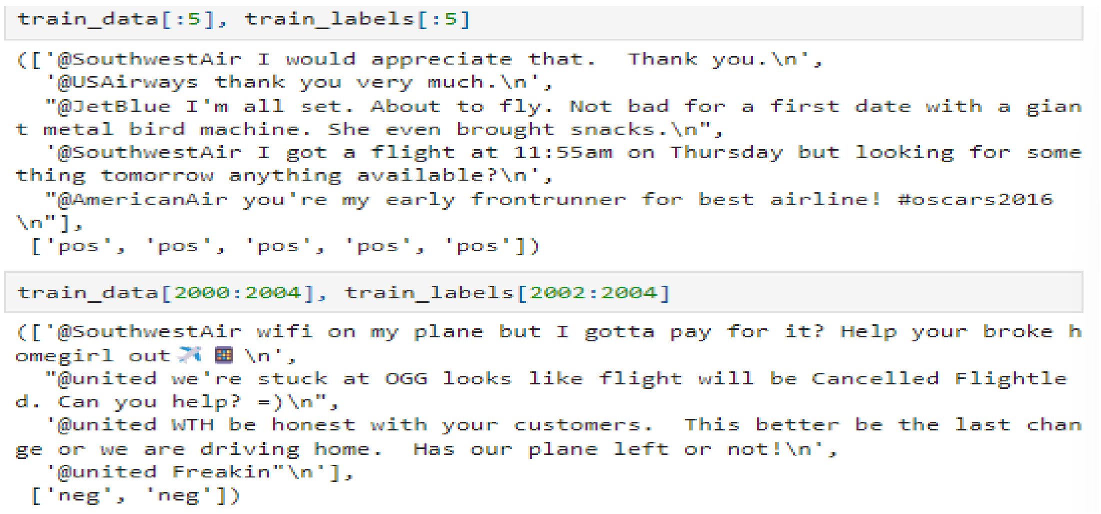
- Train a naïve Bayes model on a sentiment analysis task
- Testing the model
- Computing ratios of Positive and Negative words.

Structure of our project:

1. **Importing and Organizing Data:**

- Load Data: We utilized 'os' and 'glob' modules to navigate through the zip file (tweet) and load text data from respective directories.
- In here the dataset was segregated into training and testing subsets as per the folder name, such that it is suitable for next step (preprocessing).

Input:



```
train_data[:5], train_labels[:5]
```

```
(['@SouthwestAir I would appreciate that.  Thank you.\n',
  '@USAirways thank you very much.\n',
  "@JetBlue I'm all set. About to fly. Not bad for a first date with a gian
t metal bird machine. She even brought snacks.\n",
  '@SouthwestAir I got a flight at 11:55am on Thursday but looking for some
thing tomorrow anything available?\n',
  "@AmericanAir you're my early frontrunner for best airline! #oscars2016
\n"],
 ['pos', 'pos', 'pos', 'pos', 'pos'])
```

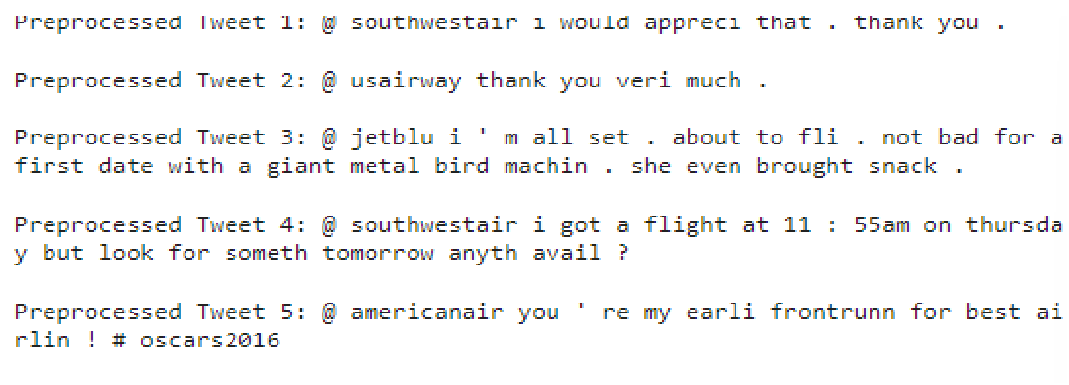
```
train_data[2000:2004], train_labels[2002:2004]
```

```
(['@SouthwestAir wifi on my plane but I gotta pay for it? Help your broke h
omegirl out ✈️ 🌃 \n',
  "@united we're stuck at OGG looks like flight will be Cancelled Flightle
d. Can you help? =)\n",
  '@united WTH be honest with your customers.  This better be the last chan
ge or we are driving home.  Has our plane left or not!\n',
  '@united Freakin"\n'],
 ['neg', 'neg'])
```

2. **Preprocessing:**

- In here the imported tweet are then stored in a list
  Firstly initialize (Preprocessed_tweets = [ ])

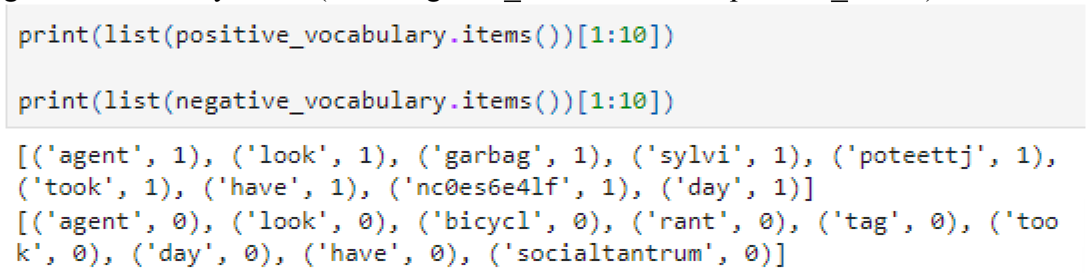
- Cleaning Data: Then by applying text processing techniques, we removed HTMl tags, used emoji module to convert emojis to text.
- Tokenization and Stemming: As said we have replaced all unnecessary punctuations using ‘WordPunctTokenizer’ also separating punctuations from words.
- Then applied Stemming using ‘PorterStemmer’ to reduce words to their root forms. (in here we did use stemming to reduce by using Stem variable).
- Output of preprocessed_tweets



```
Preprocessed Tweet 1: @ southwestair i would appreci that . thank you .

Preprocessed Tweet 2: @ usairway thank you veri much .

Preprocessed Tweet 3: @ jetblu i ' m all set . about to fli . not bad for a
first date with a giant metal bird machin . she even brought snack .

Preprocessed Tweet 4: @ southwestair i got a flight at 11 : 55am on thursda
y but look for someth tomorrow anyth avail ?

Preprocessed Tweet 5: @ americanair you ' re my earli frontrunn for best ai
rlin ! # oscars2016
```

3. **Creating Vocabulary and Bag of Words:**

- Without including any test data, we have created a vocabulary through the above list of preprocessed_tweets, we used a set method to store unique positive and negative words separately.
- In here we did loop through each processed tweet and then updates word sets based on good and bad keywords (0 for negative_words and 1 for positive_words).



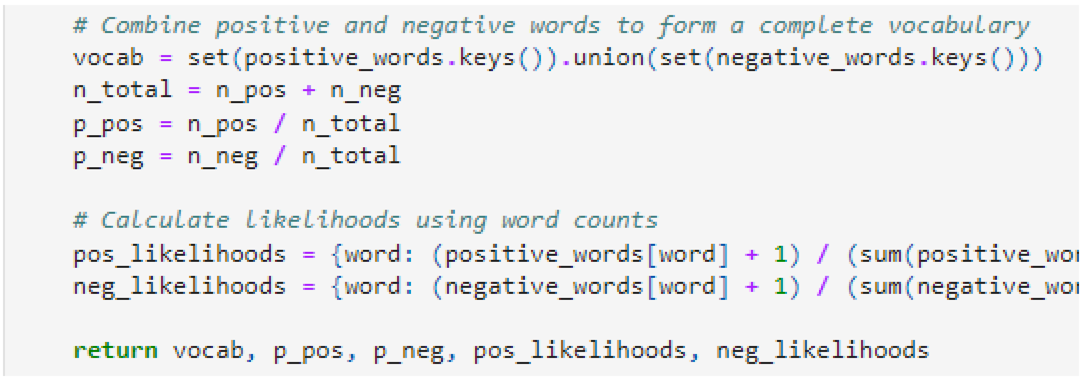
```
print(list(positive_vocabulary.items())[1:10])

print(list(negative_vocabulary.items())[1:10])
```

```
[('agent', 1), ('look', 1), ('garbag', 1), ('sylvi', 1), ('poteettj', 1),
('took', 1), ('have', 1), ('nc0es6e4lf', 1), ('day', 1)]
[('agent', 0), ('look', 0), ('bicycl', 0), ('rant', 0), ('tag', 0), ('too
k', 0), ('day', 0), ('have', 0), ('socialtantrum', 0)]
```

- For each review updates the count for positive and negative occurrences based on review label, and increments the total count.
- It then constructs the vocabulary from all unique words across both classes.

4. **Navie Bayes Model:**

- From above total count it calculates the prior probabilities of positive and negative classes based on their relative frequencies.
- Finally, it computes the likelihood of each word in the vocabulary by applying laplace smoothing.
- The function returns the vocabulary, prior probabilities of each class, and the likelihoods of each word given the class, ready for use in classifying new reviews.



```
    # Combine positive and negative words to form a complete vocabulary
    vocab = set(positive_words.keys()).union(set(negative_words.keys()))
    n_total = n_pos + n_neg
    p_pos = n_pos / n_total
    p_neg = n_neg / n_total

    # Calculate likelihoods using word counts
    pos_likelihoods = {word: (positive_words[word] + 1) / (sum(positive_wo
    neg_likelihoods = {word: (negative_words[word] + 1) / (sum(negative_wo

    return vocab, p_pos, p_neg, pos_likelihoods, neg_likelihoods
```

5. **Classification and Prediction:**

- As given, trained on the dataset of text reviews. For each token in processed text, if token is found within the vocabulary during training, the function multiplies the current class probabilities by the likelihood of that token (positive or negative)
- Finally, after comparing the probabilities for each class it returns "Pos" if the probability of positive is greater than the probability of negative.

6. **Evaluation (Models Performance):**

- Used Test dataset to evaluate the performance of the Navie bayes model.
- In here we calculated Accuracy, Confusion matrix, Classification report.

Below is the summary how effectively is NB Classification model is:

- On average we got accuracy about 78.62%, which indicates the models' predictions match the true labels of the test dataset.
- Out of all instances we got Precision (Positive) as 0.51, indicating 51% were predicted as positive, and precision (Negative) as 0.90, indicating 9% were predicted as actually negative. So, which tells us that model is quite reliable in its negative predictions.
- With Recall and F-1 Score (for positive and negative) is 51% and 90%, indicating model is equally balanced in predicted positive and negative instances, while it was more effective at recognizing negative sentiments.
- Out confusion matrix:
- **Confusion Matrix:**
- **defaultdict(<class 'int'>, {('pos', 'pos'): 599, ('neg', 'pos'): 583, ('neg', 'neg'): 2689, ('p os', 'neg'): 311})**
- Where the number of positive instances correctly predicted (599 as positive), number of negative instances correctly predicted as negative (2689).
- On the other hand, the False Positive were 583 and False Negatives were 311.

Overall, the model's performance was good, because it shows high precision, recall, F1- score for the negative class. However, it was less effective for the positive class with over all 51%.

So, there can be improvements made in feature engineering and processing steps.

The model performance metrics and confusion matrix are saved in the file model_performance.txt

Algorithm Described Step wise:

So, in our project we used multiple components as one algorithm which is divided into several steps:

Here is a breakdown of how these steps form a complete algorithm.

1. Training Phase
2. Classification Phase
3. Evaluation Phase

1. Training Phase (train_nb function): This is where the model learns from the training data. It involves preprocessing the data, calculating the frequencies of words in positive and negative sentiments, and then using these frequencies to compute the likelihood of each word given a sentiment class. The model also learns the prior probabilities of each class based on the training data. This phase is crucial for setting up the model with the knowledge it needs to make predictions.

2. Classification Phase (classify_naive_bayes function): This phase involves preprocessing this text in the same way as the training data, and then using the learned information (word likelihoods and class priors) to calculate the probability of the text belonging to each class. The text is classified based on which probability is higher, essentially applying the Bayes theorem to make a prediction.

3. Evaluation Phase (calculate_confusion_matrix function): After predictions have been made, it's important to assess how well the model is performing. This function calculates a confusion matrix based on the predicted and actual labels of the test dataset, allowing for a detailed evaluation of the model's accuracy, precision, recall, and F1 scores.

Bonus Points: Answered

Impact of TF-IDF:

When TF-IDF representation is used instead of binary representation for Naive Bayes, the accuracy improved to 79%.

Precision, recall, and F1-score also improved for both positive and negative classes.

Regularization in Logistic Regression:

Regularizing Logistic Regression with L2 regularization improved the accuracy to 89%.

Both L1 and L2 regularization enhanced precision, recall, and F1-score for both positive and negative classes.

References:

[1] Natural Language Processing with Classification: https://github.com/y33-j3T/Coursera-Deep-Learning published in GitHub.

[2] Text Classification Algorithms:
https://github.com/kk7nc/Text_Classification/blob/master/README.rst published in GitHub.
[3]TF and Inverse Document Frequency https://www.analyticsvidhya.com/blog/2021/09/creating-a-movie-reviews-classifier-using-tf-idf-in-python/ by Pratheek Majumder, on 28 November 2023.