Contact

Name: Anya Rajan

Email: arajan2@ucsc.edu

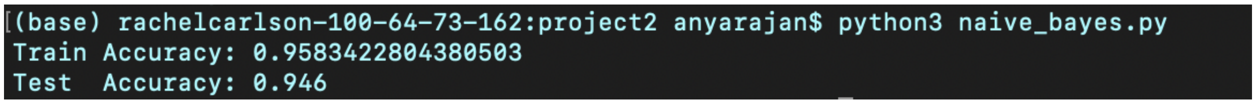
Instructions to Run

1. Open terminal
2. cd into the directory that holds the file naive_bayes.py
3. Once in the correct directory, run the following command:
    - python3 naive_bayes.py

Expected Output

The expected output was between 90-95% accuracy.

My Output



```
[(base) rachelcarlson-100-64-73-162:project2 anyarajan$ python3 naive_bayes.py
 Train Accuracy: 0.9583422804380503
 Test  Accuracy: 0.946
```

Understanding of Why the Program Works

My task was to fill in the code for the "fit" and "predict" functions. After the "fit" function runs, the following had to be true:

1. self.num_train_hams is set to the number of ham emails given.
2. self.num_train_spams is set to the number of spam emails given.
3. self.word_counts_spam is a dictionary where word_counts_spam[word] is the number of spam emails which contain this word.

4. self.word_counts_ham is a dictionary where word_counts_ham[word] is the number of ham emails which contain this word.

To set num_train_hams to the number of ham emails given, I found the length of the train_hams list passed into the function with len(). I did the same, respectively, for num_train_spams. To populate the self.word_counts_ham dictionary, I used an outer for loop to loop over the ham emails given. Inside this loop, I made a set that holds all the words in the email with word_set(train_hams[i]), the counter of the loop being "i". Then, I began an inner loop that loops over the words in this set. Inside this for loop, I set the element x (counter of the loop) of word_counts_ham equal to self.word_counts_ham.get(x, 0) + 1 to keep track of the word and how many times it has been seen in the email. I did the same, respectively, for spam.

My next task was to create the "predict" function. In this, I had to predict whether or not the email passed to the function is spam or ham. I first found the probability of spam emails in the total set by dividing the total number of spam by the sum of total spam and total ham. I also found the probability of ham emails this way. After this, I created the set of words that are in the email passed in with self.word_set(filename). I then looped over the set in a for loop. Inside this for loop, I added to the variable word_probs_spam, which I initialized to zero at the beginning of this function. I added the log of the *number of spam emails with the word* (plus one to prevent zero probabilities) divided by the *number of total spam emails* (plus two to prevent zero probabilities) to the word_probs_spam variable.

To find the final probability of the email being spam, I added word_probs_spam with the log of the total probability of getting a spam email (calculated at the beginning of this function). Normally, you would multiply all of these quantities together, but since we are using the log function, we have to add them. I did the same process for ham emails. After calculating both final probabilities, I checked if the probability of spam was higher than ham. If it was, the program returns the spam label. Otherwise, it returns the ham label.

* Using log accounts for underflow problems since we are multiplying many small numbers.

NOTE: For calculating the final probabilities, we are essentially using the LTP, but we do not need to worry about the denominators for both spam and ham because they are the same. Therefore, we are just calculating and comparing the numerators.