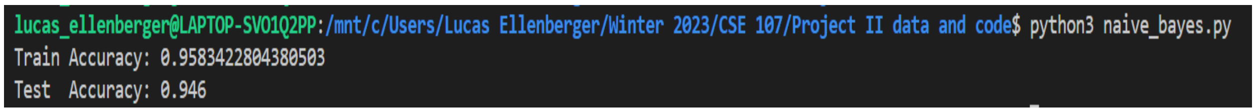
Lucas Ellenberger
CruzID: lmellenb
Email: lmellenb@ucsc.edu

I completed project 2 in python.

To run my code, use:
python3 naive_bayes.py

Again, I had to run my code in my WSL terminal because it was the only place able to install numpy. But, the code will run as expected on any machine or environment with numpy installed.

naive_bayes.py:



```
lucas_ellenberger@LAPTOP-SVO1Q2PP:/mnt/c/Users/Lucas Ellenberger/Winter 2023/CSE 107/Project II data and code$ python3 naive_bayes.py
Train Accuracy: 0.9583422804380503
Test  Accuracy: 0.946
```

Understanding of the program:

The program relies heavily on a naive version of Bayes' Theorem to calculate the probability that an email should be classified as spam or ham. The program is considered naive because the program considers the event of each word as independent events. However, in the actual world, words are used to form coherent sentences, so these words may not be independent events from each other. The program is fed training data with emails already classified as spam and ham emails. The program goes through every word in each email, excluding the header, and gets the set of words used in the email. The program counts the total number of times each word appears in a spam email, a ham email, or any email. Based on the training data, each word has two unique probabilities, the chance to be in a spam email and the chance to be in a ham email. We use Laplace smoothing to ensure there aren't any words with a probability of zero, which would majorly harm our calculations.

After calculating the various probabilities from the training data, the program is able to start predicting whether new, unseen emails are spam or ham. As test emails are fed to the program, the program uses Bayes' Theorem to calculate the probability that an email is spam or ham given the set of words used in the email. The program compares the probabilities of being a spam or ham email, and whichever has a higher probability, the program classifies the email as such. Since many of these probabilities are quite low, we take the log of each probability to prevent integer underflow, since computers struggle multiplying many small numbers together, which is required to calculate the probabilities of all of these unique events.

The program relies on a sufficiently large number of emails to train the algorithm. Without enough information, the program will be more likely to misclassify test emails. As you can see in the image above, my program has a 94.6% chance to classify new (test) emails correctly after being trained with enough emails.