Name: Adarsha Poudel

Perm: 3347028

Email: adarsha@ucsb.edu

Architecture:

I have one struct named "Label" and one class called "NaiveBayesClassifier" with one parametrized constructor and four functions.

```
struct Label
```

This struct Label is used to represent the features/words of the document with their positive review count and negative review count. E.g. Label l ("love", 5, 6) means l stores the feature "love" which has 5 positive reviews and 6 negative review.

NaiveBayesClassifier: This class uses hash table named "entries" to store features and its positive/negative count. It then uses different functions and constructor to train and test data.

```
Constructor:    NaiveBayesClassifier(std::string);
```

This constructor is used to train data. It takes in the filename as a string. Then it opens the file, splits each line of document into words(features) and their label. After that, it uses "insert" function to insert the word and their label into the hash table.

```
void insert(std::string, int);
```

This function is called by the constructor to help train the data. It gets the feature as string and its class(review) as an integer (0 or 1). If the feature already exists, it just increments the count of review of that word. If it does not, it creates new struct object and inserts the feature into the hash table.

```
int hash(std::string);
```

This function returns the index of my hash table given the feature as string by moding the product of ascii value of the characters in that string by the size of hashtable.

```
double test(std::string);
```

This function takes the filename of testing data as string, opens that file, and tests the file. It uses "return Probability" function to get the probability of certain feature belonging to some class, i.e, P($word_i$| positive/negative review). It then manipulates the probabilities, compares them, and return the accuracy of our model as a double.

```
double returnProbability(std::string, int label);
```

This function is helper function for "test" which returns probability of a word belonging to certain label, i.e., P($word_i$| positive/negative review).

Preprocessing:

I represented each document as bag-of-words model where features are the words in the document d_i. Keeping in mind this model loses order-specific information, I tried to improve the bag of words model introducing bigram to my features. So, each feature represents adjacent two words without the space between them. For example, if the document is "I love you": "Ilove", "loveyou" are the features. Preprocessing is done in my constructor of NaiveBayesClassfier class.

Model Building:

For each features/bigrams word that I extracted in preprocessing, I stored the positive review count and negative review count of that feature. So, when testing, for each feature in the testing text, I calculated the P (word| +ve/-ve review) using this formula:

$$\text{P (word| +ve/-ve review)} = \frac{\text{Number of } +\frac{ve}{-ve}\text{review count of that word + alpha}}{Total\ words\ in + \frac{ve}{-ve}\ review + alpha * vocabulary}$$ where alpha is arbitrary number and vocabulary is number of unique words in the training text. Doing this I prevented the situation where the unseen feature produces 0 probability and affects the MAP. I tried different alphas; however, alpha = 1.0 gave me the best accuracy.

After calculating conditional probabilities of each feature given certain class (positive or negative review), I added logs of probabilities of each feature instead of multiplying each probability in one document to prevent overflow. I then compared the value of P(Word1, Word2, …, WordN | +ve review) and P(Word1, Word2, …, WordN | -ve review), and classified that document accordingly.

Challenges:

After implementing hash table on my own without any library, the huge problem for me was to determine a good hash table size so that hashtable perform the most efficient. So, I read up one paper, where they described that the size of hashtable should always be prime and be about 1.3* total keys you are inserting. Hence, I found out the total number of unique words in training text to be around 5000. So, I choose the prime 6577 to meet the criteria.

Implementing the simple naïve model using bag of words model only gave me 85 percent accuracy. To improve the accuracy, I incorporated bigram feature and was able to boost my accuracy to 0.88.

My next challenge was the overflow by multiplying many probabilities of features. I overcame this struggle using logarithmic property of addition: $\text{Log}(xy) = \log(x) + \log(y)$

Weaknesses:

Since I am using bags of words model, I lose order-specific information of the features, I also limit the context of my words in the document. While I did use bigram to value order-specific information of words, I think the mixture of trigram/bigram would have made an accurate model. I could not implement trigram, because it took longer than 10 minutes for me to execute my program. Now for the context-based information of the words, I could have used stop-word removal to remove words that does not add weight to the reviews.

Results:

I have defined important features as the words which has the most positive/negative review count so that it has more predictive power.

Notice each word here represents a bigram: two words without spaces.

```
1
0
0
0
1
1
27 seconds (training)
28 seconds (labeling)
0.980 (training)
0.886 (testing)
The top ten important positive review words are:
isa: Positive Review Count = 5534 P(isa|+ve review) = 0.003
youcan: Positive Review Count = 6167 P(youcan|+ve review) = 0.003
ofthe: Positive Review Count = 11892 P(ofthe|+ve review) = 0.006
thisis: Positive Review Count = 4476 P(thisis|+ve review) = 0.002
tothe: Positive Review Count = 4661 P(tothe|+ve review) = 0.002
thisgame: Positive Review Count = 10985 P(thisgame|+ve review) = 0.005
inthe: Positive Review Count = 6919 P(inthe|+ve review) = 0.003
thegame: Positive Review Count = 11699 P(thegame|+ve review) = 0.005
ifyou: Positive Review Count = 6640 P(ifyou|+ve review) = 0.003
its: Positive Review Count = 7634 P(its|+ve review) = 0.004

The top ten important negative review words are:
dont: Negative Review Count = 6750 P(dont|-ve review) = 0.002
youcan: Negative Review Count = 5466 P(youcan|-ve review) = 0.002
ofthe: Negative Review Count = 12818 P(ofthe|-ve review) = 0.005
onthe: Negative Review Count = 5543 P(onthe|-ve review) = 0.002
tothe: Negative Review Count = 5586 P(tothe|-ve review) = 0.002
thisgame: Negative Review Count = 15086 P(thisgame|-ve review) = 0.005
inthe: Negative Review Count = 8416 P(inthe|-ve review) = 0.003
thegame: Negative Review Count = 15952 P(thegame|-ve review) = 0.006
ifyou: Negative Review Count = 6971 P(ifyou|-ve review) = 0.003
its: Negative Review Count = 8228 P(its|-ve review) = 0.003
```