CSCI-580 Project Spam & Ham Classifier

Objective: In this project, you will implement a Naïve Bayes Classifier to classify emails into spam or ham classes.

You will be provided with four files: *training_ham.txt, training_spam.txt, testing_ham.txt* and *testing_spam.txt.* Your program must read the names of these files from the command line. In addition, your program will read an integer, k, from the command line. This is the command line arguments order:

./run *training_ham.txt training_spam.txt testing_ham.txt testing_spam.txt* 2

Here, "run" is an executable, and "2" is k, the integer that you will use for Laplace smoothing.

Specification Requirements:

1. Read in *training_ham.txt* and put all the words into *map<string, int>* (I will call this map *wordsHam*), where key is a word, and integer represents the count of the word in this file with *ham* emails. While reading this file, count the total number of words in this file, and count the total number of messages in this file (total lines).
2. Do the same with *training_spam.txt* file (I will call the map *wordsSpam*).
3. Calculate a *priori class probabilities:*

$$P(ham) = \frac{totalMsgHam + k}{totalMsgBoth + \ kn}$$

$$P(spam) = \frac{totalMsgSpam + k}{totalMsgBoth + \ kn}$$

   where n = 2 (since there are two classes, namely ham and spam), and k is the last command line argument; totalMsgBoth is the total number of messages (lines) in both files *training_ham.txt* and *training_spam.txt.*

4. For each word in the dictionary (each word occurring in either of the files), calculate the conditional word likelihood probabilities:

$$P(w_i|ham) = \frac{countHam + k}{totalWordsHam + \ kn}$$

$$P(w_i|spam) = \frac{countSpam + k}{totalWordsSpam + \ kn}$$

   Here, *n* is the size of the dictionary (total distinct words in both *ham* and *spam*), k is the same as before, *countHam* and *countSpam* are total of occurrences of the given word in *ham* and *spam* respectively, and *totalWordsHam* and *totalWordsSpam* are total words in *ham* and *spam* respectively.
   Consider using *map<string, pair<double, double> >* to store these probabilities, where the key is a given word, and *pair<double, double>* has P(w|ham) as *first* and P(w|spam) as *second* members of pair.

5. Read in *testing_ham.txt* and *testing_spam.txt* and for each email (line), calculate the unnormalized posterior probabilities of the given message being *ham* and *spam:*

$$\begin{aligned} P(ham|msg) &= P(ham|w_1, w_2, ..., w_n) \\ &= P(w_1, w_2, ..., w_n|ham)P(ham) \\ &= P(ham)P(w_1|ham)P(w_2|ham)\dots P(w_n|ham) \end{aligned}$$

$$\begin{aligned} P(spam|msg) &= P(spam|w_1, w_2, ..., w_n) \\ &= P(w_1, w_2, ..., w_n|spam)P(spam) \\ &= P(spam)P(w_1|spam)P(w_2|spam)\dots P(w_n|spam) \end{aligned}$$

In case, a word is not in the dictionary (has not occurred in *training_ham.txt* or *training_spam.txt*), choose a small probability for the conditional P(w|ham) or P(w|spam):

$$P(w_i|ham) = \frac{1}{totalWordsHam + \ totalWordsSpam}$$

$$P(w_i|spam) = \frac{1}{totalWordsHam + \ totalWordsSpam}$$

To avoid underflow, instead of the product of probabilities, use the sum of their logs:

$$\log P(ham|msg) = logP(ham) + logP(w_1|ham) + logP(w_2|ham) + \ldots + logP(w_n|ham)$$
$$\log P(spam|msg) = logP(spam) + logP(w_1|spam) + logP(w_2|spam) + \ldots + logP(w_n|spam)$$

6. Classify a message as *ham* if log(*P*(*ham*|*msg*)) > log(*P*(*spam*|*msg*)), and as *spam,* otherwise.
7. Calculate *Specificity, Sensitivity* and *Accuracy* of the classifier:

$$\boldsymbol{Specificity} = \frac{True\ Negative}{True\ Negative + False\ Positive} = \frac{True\ Negative}{Total\ Negative\ in\ a\ Population}$$

$$\boldsymbol{Sensitivity} = \frac{True\ Positive}{True\ Positive + False\ Negative} = \frac{True\ Positive}{Total\ Positive\ in\ a\ Population}$$

$$\boldsymbol{Accuracy} = \frac{True\ Positive + True\ Negative}{Total\ size\ of\ data}$$

8. Output the required results using *cout.* First, for each line in *testing_ham.txt*, output on a separate line “ham” or “spam” word according to your classification, followed by the

log($P(ham|msg)$) and log($P(spam|msg)$) values. Second, repeat similar output for each message in *testing_spam.txt* file. Finally, the last line of the output will contain the values of *Specificity, Sensitivity* and *Accuracy* of the classifier.

Input Format:

Each of the input files above have lines inside of them, where each line represents a separate email.

Output Format:

ham -40.0978 -51.307
spam -62.5378 -56.8432
0.69 0.927 0.74

<word><space><logHam><space><logSpam><endl>
<specificity><space><sensitivity><space><accuracy><endl>

Additional Specification Requirements:

- Break your program into smaller tasks and write a separate function for each task. Your *main()* function should just declare necessary data structures (or classes) and call functions to accomplish the goal of this assignment.
- You may write a C++ class to complete your program, but please put the class's definition and description into *main.cpp* file.
- Follow the best coding practice rules such as pass-by-reference large data structures or big objects as parameters to the functions to save time/space of copying large objects; and include comments to clarify your code.

Submission:

1) Submit *main.cpp* to turnin.

2) In addition to submitting files to turnin, you need to submit main.cpp to Blackboard. Make sure to include your name inside *main.cpp* file.

Failure to submit main.cpp file to Blackboard will result in 0pts for your project.