Faculty of Science

Course: CSCI 2020U: System Development and Integration
Component: Assignment #1
Topic: Spam Detection

Collaboration Policy

You are permitted to work on this assignment in a team, and submit the results as a team. For this sort of assignment, with an open-ended component, the collaboration between multiple team members can be beneficial. Between groups, however, please limit the discussion to the level of general strategy (not code). Groups of **size 2 are recommended**. In any case, be sure that all members of the team fully understand all code, otherwise they will miss intended learning objectives, which may be a considerable disadvantage at other assessments.

Overview

You have become frustrated with all the advertisements in your inbox. You resolve to create a spam detector to filter out the spam. The spam detector will use a dataset of E-Mails (spam or otherwise) to train your program to recognize whether or not new E-Mails are spam. The program will use a unigram approach [1], where each word is counted and associated with whether or not the message is spam. Your program will calculate probabilities based on each word's frequency [2]. Luckily, you have not emptied your spam folder or inbox in quite a while, so you have many samples to use to train your system.

Primary Instructions

Before you begin, download the training and testing data, which contains a bunch of spam (unwanted messages, often advertisements) and ham (wanted emails, not spam) messages. The data is divided into two folders: **train** and **test**.

The train folder will be used to determine word frequencies, and the test folder will be used to evaluate your spam filter. Within each of these two folders, the spam messages are in a folder called spam, and the ham messages are in a folder called ham.

Training

In the training phase, the program will read all the files (each of which contains one E-Mail) in both the training/spam and training/ham folders. For each file, you will determine which words exist in each file. A good starting point for this code is the File I/O sample that we completed in the lectures. For simplicity, let's ignore case, and let's also ignore how many times a word appears in a single file, and just count how many files have the word.

To do this, we'll combine these words into two frequency maps: trainHamFreq and trainSpamFreq. The map trainHamFreq contains a map of words and the number of files containing that word in the ham folder. The map trainSpamFreq contains a map of words and the number of files containing that word in the spam folder.

The goal of this part of the process is to collect a set of $Pr(S|W_i)$ values, for each word W_i. $Pr(S|W_i)$ is the probability that a file is spam, given that it contains the word W_i. $Pr(W_i|S)$ is the probability that the word

W_i appears in a spam file. Pr(W_i|H) is the probability that the word W_i appears in a ham file.

$$Pr(S|W_i) = \frac{Pr(W_i|S)}{Pr(W_i|S) + Pr(W_i|H)}$$

$$Pr(W_i|S) = \frac{of\ spam\ files\ containing\ W_i}{of\ spam\ files}$$

$$Pr(W_i|H) = \frac{of\ ham\ files\ containing\ W_i}{of\ ham\ files}$$

***Note**: It is recommended that you put all of these probabilities, $Pr(S|W_i)$, into a map (e.g. TreeMap), indexed by the word, W_i, itself. Using a map for this purpose is efficient, and will result in shorter code since you will not need to search through a list of probabilities.*

Testing

Once we have built up our probability map, we'll examine two more folders: test/ham and test/spam. The emails in test/ham are not spam, and the emails in test/spam are spam. Do not use these files for training your program. We are now going to see how well your spam detector works.

Each file in both directories will be examined, word by word. The ham and spam probabilities for each word in a file will be used to compute a probability that the file is spam.

$$\eta = \sum_{i=1}^{N} \left[\ln\left(1 - Pr(S|W_i)\right) - \ln\left(Pr(S|W_i)\right) \right]$$

$$Pr(S|F) = \frac{1}{1+e^{\eta}}$$

***Note**: The reason for the logarithms and exponents is a complicated one. The short story is that if we just multiply probabilities together, we may end up with values too small to accurately represent with floating points for large files. Taking the logarithm normalizes the numbers somewhat.*

The probability that a file is spam, Pr(S|F), is determined by the formulae above. Some useful Mathematical operations and constants are given in the following table:

Function or Constant	**Description**
Math.E	The constant e
Math.log(x)	Calculates the natural logarithm of x: ln(x)
Math.pow(x,y)	Calculate x to the exponent y: x^y

The results of this step will be a List of TestFile objects. The TestFile class is given in listing 1.

Listing 1: The TestFile class

```
import java.text.DecimalFormat;

public class TestFile {
       private String filename;
       private double spamProbability;
       private String actualClass;

       public TestFile(String filename, double spamProbability, String actualClass) {
              this.filename = filename;
              this.spamProbability = spamProbability;
              this.actualClass = actualClass;
        }

       public String getFilename(){ return this.filename; }
       public double getSpamProbability(){ return this.spamProbability; }
       public String getSpamProbRounded(){
        DecimalFormat df = new DecimalFormat("0.00000");
        return df.format(this.spamProbability);
       }
       public String getActualClass(){return this.actualClass;} public
       void setFilename(String value) { this.filename = value; }
       public      void      setSpamProbability(double      val)      {
       this.spamProbability = val; }
       public void setActualClass(String value) { this.actualClass =
       value; }
}
```

Evaluation

When your application starts, it will immediately ask the user to choose a directory. An example of this code, for JavaFX, is given in listing 2. The user will select the folder which contains the train and test folders. Your application will then run the training and testing phases, as described above.

Listing 2: Sample code for using DirectoryChooser

```
DirectoryChooser directoryChooser = new DirectoryChooser();
directoryChooser.setInitialDirectory(new File("."));
File mainDirectory = directoryChooser.showDialog(primaryStage);
```

Once training and testing has completed, display your results in a TableView, including columns for the filename (which is unique), your spam detector's categorization, and the actual category (which is already known, based on the folder name). At the bottom of your application window, display some summary stats, including the percentage of correct guesses (accuracy), and the ratio of correct positives (spam) to spam guesses (correct or not) (precision). An example of the output is given in figure 1.

$$accuracy = \frac{numCorrectGuesses}{numGuesses} = \frac{numTruePositives + numTrueNegatives}{numFiles}$$

$$precision = \frac{numTruePositives}{numFalsePositives + numTruePositives}$$

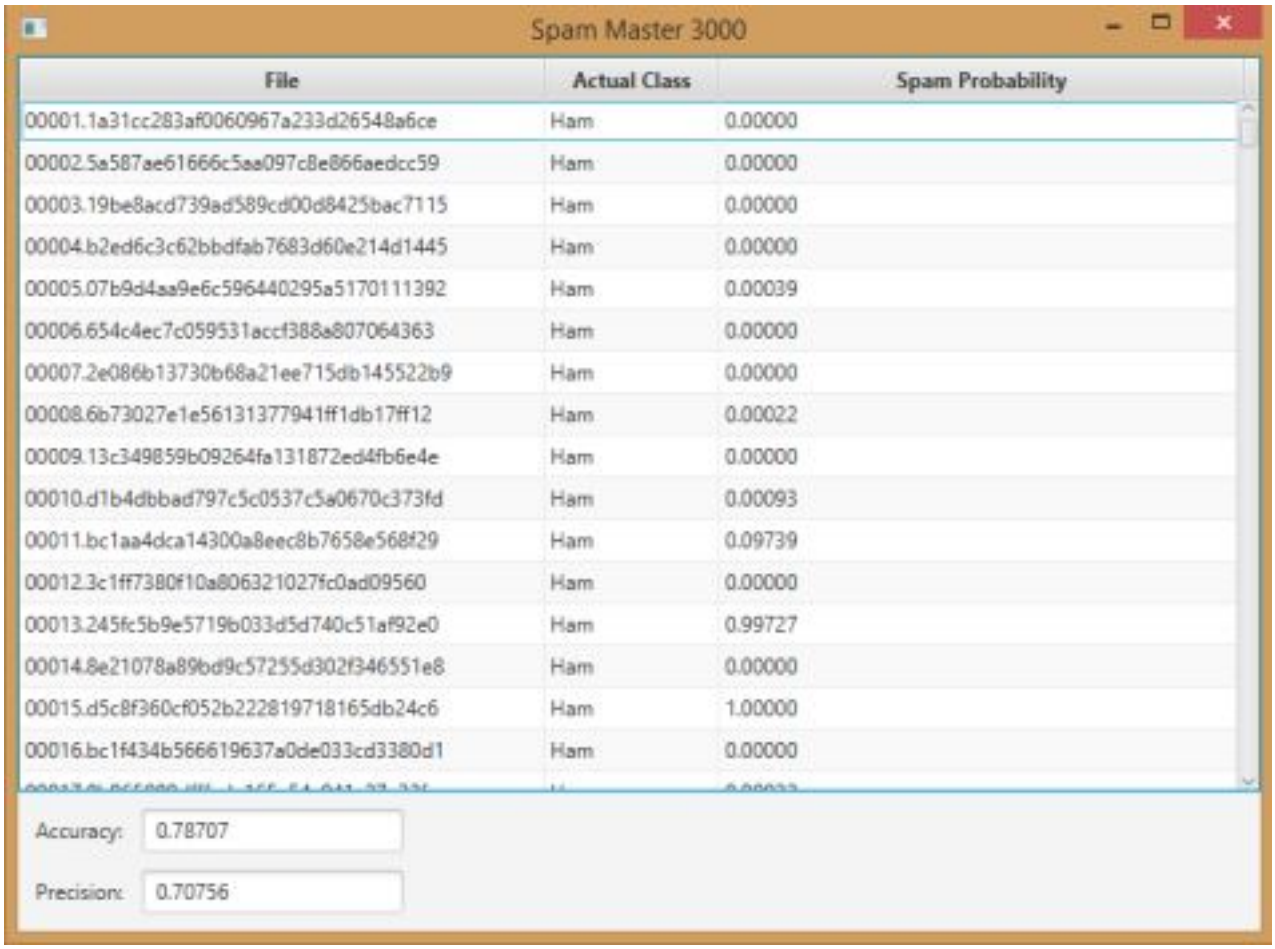

Figure 1: Sample output of the spam detector

Secondary Instructions

1. **The interface.** While figure 1 shows the most basic version of your UI for the spam detector; you are expected to put effort into improving the aesthetics of the screen.
2. **The model.** Can you think of any way to improve the spam detector? Do some research and see how much you can improve your score in your final product. Start with a copy of your bag of words, Bayesian spam detector, above, and modify it to implement your ideas and verify that your ideas work.
3. **README.md.** Make sure your project includes a readme.md file containing the following sections:
    a. Project information: short textual description of your project, and at least one screenshot of your application running.
    b. Improvements: briefly describe the improvements you made to the interface and/or the model (if any).
    c. How to run: step-by-step information on how one can successfully clone and run your application.
    d. Other resources: any references to other materials/libraries that you might have used on your solution or model improvement.

References

[1] https://en.wikipedia.org/wiki/Bag-of-words_model
[2] https://en.wikipedia.org/wiki/Naive_Bayes_spam_filtering

How to Submit

You will maintain a **git repository** for this assignment, shared among all group members, and the TA (@`Desousak`) will be added as an author to the repository (or you can simply use a public repository).

To submit the assignment,

1. Upload your project's cloned .zip file.
2. In the submission comment include the URL of your repository on github (remember the TA will be previously added as a project member), and the name of all student collaborators in this assignment.

***Note**: Comments are mandatory. Failure to properly document your program will result in a deduction on the marks you receive for this (and any other) assignment.*