



Faculty of Science

<b>Course:</b>	CSCI 2020U: System Development and Integration
<b>Component:</b>	Assignment #1
<b>Topic:</b>	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. Larger groups will be considered with the proviso that the marker will mark your assignment with higher expectations. 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 exam time.

## 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 E-Mails, 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.

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

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

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

**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 E-Mails in `test/ham` are not spam, and the E-Mails 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.

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

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

**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
<b>Math.E</b>	The constant e
<b>Math.log(x)</b>	Calculates the natural logarithm of x: $\ln(x)$
<b>Math.pow(x, y)</b>	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.

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

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

File	Actual Class	Spam Probability
00001.1a31cc283af0060967a233d26548a6ce	Ham	0.00000
00002.5a587ae61666c5aa097c8e866aedcc59	Ham	0.00000
00003.19be8acd739ad589cd00d8425bac7115	Ham	0.00000
00004.b2ed6c3c62bbdfab7683d60e214d1445	Ham	0.00000
00005.07b9d4aa9e6c596440295a5170111392	Ham	0.00039
00006.654c4ec7c059531accf388a807064363	Ham	0.00000
00007.2e086b13730b68a21ee715db145522b9	Ham	0.00000
00008.6b73027e1e56131377941ff1db17ff12	Ham	0.00022
00009.13c349859b09264fa131872ed4fb6e4e	Ham	0.00000
00010.d1b4dbbad797c5c0537c5a0670c373fd	Ham	0.00093
00011.bc1aa4dca14300a8eec8b7658e568f29	Ham	0.09739
00012.3c1ff7380f10a806321027fc0ad09560	Ham	0.00000
00013.245fc5b9e5719b033d5d740c51af92e0	Ham	0.99727
00014.8e21078a89bd9c57255d302f346551e8	Ham	0.00000
00015.d5c8f360cf052b222819718165db24c6	Ham	1.00000
00016.bc1f434b566619637a0de033cd3380d1	Ham	0.00000
00017.010c50001ff611655404107333f	Ham	0.00033

Accuracy:

Precision:

Figure 1: Sample output of the spam detector

### Secondary Instructions

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.

### References

- [1] [https://en.wikipedia.org/wiki/Bag-of-words\\_model](https://en.wikipedia.org/wiki/Bag-of-words_model)
- [2] [https://en.wikipedia.org/wiki/Naive\\_Bayes\\_spam\\_filtering](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 instructor will be added as an author to the repository (or you can simply use a public repository). To submit the assignment, create a single file 'README.txt' that contains instructions on how to download, compile, and run your application. A .zip, .7z, or .rar file will not be acceptable.

**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.