Assignment 2
Spam Filtering Using a Naive Bayes Text Classifier

1 The data and the java class

The data necessary for this programming assignment consists of a collection of e-mail messages arbitrarily divided into 2 directories: *train/* and *test/*. Each of these 2 directories contain 2 subdirectories: *regular/* and *spam/*. These subdirectories in turn contain the e-mail messages as separate ASCII text files. The data is therefore organized as follows:

```
/train/regular/regular-0001.msg
                /regular-0002.msg
                /...
       /spam/spam-0001.msg
             /spam-0002.msg
             /...

/test/regular/regular-1001.msg
               /regular-1002.msg
               /...
      /spam/spam-1001.msg
            /spam-1002.msg
            /...
```

The java class provided for this programming assignment is 'Bayespam.java'. It can be compiled with the command:

```
javac Bayespam.java
```

The resulting '.class' file can be executed with the command:

```
java Bayespam arg1 arg2
```

where arg1 = the train directory, arg2 = the test directory. The two cmd line arguments can be *train/ test/* or vice versa. It is up to the user to decide, but the first argument will always be used for training and the second for testing.

The java program reads the training data, collects all the encountered words into a large vocabulary and separately counts how often a given word was encountered in regular mail (*counterRegular*) and in spam mail (*counterSpam*). The vocabulary is built using a hashtable. All the words from the vocabulary and their corresponding counts (*counterRegular* and *counterSpam*) are printed to standard output for visual inspection.

As you will notice, the method for building up the vocabulary needs to be improved by:

- eliminating punctuation
- eliminating numerals
- converting all characters to lowercase

- eliminating all words with less than 4 letters

Your assignment consists in completing and improving the program.

Observation: Put clear comments in the code to explain your implementation of the algorithms. Mark your comments with three slashes (i.e. this sign: ///) at the beginning of the lines to make them distinguishable.

2 Training stage

In this stage, the counts must be converted to probabilities (logprobabilities).

2.1 Computing *a priori* class probabilities

Count the number of regular mail messages:

$$nMessagesRegular = number\ of\ entries\ in\ the\ regular/\ directory \tag{1}$$

Count the number of spam messages:

$$nMessagesSpam = number\ of\ entries\ in\ the\ spam/\ directory \tag{2}$$

Compute the total number of messages:

$$nMessagesTotal = nMessagesRegular + nMessagesSpam \tag{3}$$

Compute the prior probability for regular:

$$P(regular) = \frac{nMessagesRegular}{nMessagesTotal} \tag{4}$$

Compute the prior probability for spam:

$$P(spam) = \frac{nMessagesSpam}{nMessagesTotal} \tag{5}$$

2.2 Computing *class conditional* word likelihoods

Count the total number of words contained in the regular mail:

$$nWordsRegular = sum\ 'counterRegular'\ over\ all\ the\ words\ in\ the\ vocabulary \tag{6}$$

Count the total number of words contained in the spam mail:

$$nWordsSpam = sum\ 'counterSpam'\ over\ all\ the\ words\ in\ the\ vocabulary \tag{7}$$

For every word w_j in the vocabulary compute two class conditional likelihoods:

$$P(w_j|regular) = \frac{counterRegular}{nWordsRegular} \tag{8}$$

$$P(w_j|spam) = \frac{counterSpam}{nWordsSpam} \tag{9}$$

Zero probabilities must be avoided, they occur when one word has been encountered only in regular, but not in spam, or vice versa. Zero probabilities must be replaced by an estimated small non-zero value, e.g.

$$\frac{\epsilon}{nWordsRegular + nWordsSpam} \tag{10}$$

The value ϵ can also be considered as a tuning parameter. Take $\epsilon = 1$ and then change it to see what effect it has on the final result.

2.3 From probabilities to logprobabilites

Convert all probabilities to logprobabilities (loglikelihoods) to avoid exceeding the dynamic range of the computer representation of real numbers (underflow). Apply the log function to all probabilities (prior and conditional). You may apply the log directly when you compute the probabilities as a ratio of two counts.

3 Testing stage

The classification performance must be evaluated on the test set.

3.1 Classifying a new message

Consider a new e-mail message 'msg' from the test set. Suppose that it contains the words w_1, w_2 ... w_n from the vocabulary. Words that are not in the vocabulary are simply ignored.

Compute the *a posteri* class probabilities:

$$\begin{aligned} P(regular|msg) &= P(regular|w_1, w_2...w_n) && (11)\\ &= \alpha P(w_1, w_2...w_n|regular)P(regular) && (12)\\ &= \alpha P(regular)P(w_1|regular)P(w_2|regular)...P(w_n|regular) && (13) \end{aligned}$$

$$\begin{aligned} P(spam|msg) &= P(spam|w_1, w_2...w_n) && (14)\\ &= \alpha P(w_1, w_2...w_n|spam)P(spam) && (15)\\ &= \alpha P(spam)P(w_1|spam)P(w_2|spam)...P(w_n|spam) && (16) \end{aligned}$$

Using the logprobabilities, the product becomes a sum:

$$logP(regular|msg) = \alpha + logP(regular) + logP(w_1|regular) + logP(w_2|regular) + ... + logP(w_n|regular) \quad (17)$$

$$logP(spam|msg) = \alpha + logP(spam) + logP(w_1|spam) + logP(w_2|spam) + ... + logP(w_n|spam) \quad (18)$$

Classify the mail as regular if

$$logP(regular|msg) > logP(spam|msg) \quad (19)$$

otherwise classify it as spam.

3.2 Computing the performance on the test set

Compute how many of the regular mail messages are correctly classified as regular and also how many are wrongly classified as spam. Similarly for the spam messages. Build the confusion matrix.

3.3 Example runs

The Linux executable 'bayesian-text-classifier' is provided together with the data to have an example of how your program should perform. Run the following:

./bayesian-text-classifier train/ test/

./bayesian-text-classifier test/ train/

./bayesian-text-classifier train/ train/

./bayesian-text-classifier test/ test/

- What happens if you train and test on the same data?

4 Naive Bayes classification based on bigrams

1. This last part of the assignment consists of replacing the vocabulary of unigrams with a vocabulary of bigrams. A bigram is a pair of words occuring together in a text. For instance, in the sentence "I have a gun", "I have" and "a gun" are bigrams. A bigram can be seen as the conditional probability of a word w_1 given that it occurs together with a word w_2. A Naive Bayes bigram classifier can be trained and tested in the same way as the single-word (unigram) Naive Bayes classifier.

   - Create a class BigramBayespam so that it can be executed in the same way as the class Bayespam:

     java BigramBayespam arg1 arg2.

   - Modify your code so that the hashtable containing the vocabulary will now represent all the bigrams and their counters found in the training data. Think about how to represent bigrams.
   - Define a threshold parameter in order to include only the most frequently occuring bigrams in the vocabulary. Also define a parameter for the minimal word size which is included in the vocabulary (see the first section).
   - Modify the training and testing stage of your code so that BigramBayespam will now classify incoming messages based on the logprobabilities of the containing *bigrams*. Build the confusion matrix.

2. Evaluate the obtained classifer after you have applied it to the data set.

   (a) How does the bigram classifier perform when compared to your original unigram classifier? Try to explain the difference in performance (if present).

   (b) Try some different configurations of parameter settings (ϵ, frequency threshold, minimal word size). How do they influence the performance of the classifier? And what is the use of the frequency threshold in real-world applications?

5 Final questions (to be answered individually)

1. The data used in this assignment contains only e-mails in the English language. What happens if an e-mail in Dutch is given to your spam filter trained with English messages? How will the Dutch message be classified? Assume that there are no common words in English and Dutch. Explain your answer.

2. The Naive Bayes assumption is that the attributes (or features) are independent. Are the words in a message really independent? And what can you say about the independence between and within bigrams? Explain your answer in 250 words.