Your task for this programming assignment is to develop a Naive Bayes classifier for email, more specifically a spam filter. Your tasks are to write code to implement the Naive Bayes classifier, to test its performance, and to write an explanatory report. We have provided support code to extract header and body text tokens from email files. We have also assembled a small corpus of spam and ham (legitimate email) for you in `home/comp440/code/corpora`. You are not constrained to use this corpus. You can assemble your own corpus, making sure that you have about 3000 to 4000 examples each of spam and and ham for training, and a test set of about 300 to 400 examples each of spam and ham for testing.

1 The Problem

A great introduction to Bayesian filtering is Better Bayesian Filtering by Paul Graham available at [http://www.paulgraham.com/better.html](http://www.paulgraham.com/better.html) and an earlier article called A Plan for Spam accessible from the above URL. Please read these two articles; they are insightful and well-written.

Spam filtering uses ideas from the growing body of work in text classification. One of the key choices in designing text classification algorithms is the representation of examples. The naive Bayes algorithm requires a fixed length representation of an email message. The standard approach in the text processing community is to use the ‘‘bag of words’’ model, in which each email is treated as a set of words (or tokens) with associated frequencies of occurrence. Given a vocabulary \( V \) of tokens, each email is represented as a vector of size \( |V| \) of the form \((n_1, \ldots, n_{|V|})\), where \( n_i \) is the number of times the \( i^{th} \) vocabulary item \( v_i \in V \) occurs in that email. Clearly the choice of tokens in \( V \) (called the feature selection problem) is critical to the performance of an email classifier.

The email vectors in the training set have labels associated with them: spam or ham. Given a training set of spam and ham vectors, we learn a set of conditional probability distributions for each token in \( V \) for both spam and ham. The (maximum likelihood) estimate of the probability of token \( v_i \in V \) in a given class \( c \in \{\text{spam}, \text{ham}\} \) is

\[
P(v_i|c) = \frac{N_{ci} + 1}{N_c + |V|}
\]

where \( N_{ci} \) is the number of times token \( v_i \) appears in documents of class \( c \) and \( N_c \) is the total number of tokens in class \( c \). This estimate is a smoothed estimate of the conditional probability; it ensures that tokens that do not appear in a given training set are not given zero probability. For example, suppose the word \( \text{free} \) appears in all the spam email in the training set, but not in the hams. Then, \( P(\text{free}|\text{spam}) > 0 \) but \( P(\text{free}|\text{ham}) = 0 \). Suppose you are trying to classify a new email message in which the word \( \text{free} \) appears. While the occurrence of the word \( \text{free} \) should make it more likely that the email is spam, it shouldn’t make us rule out the possibility of it being a ham. Without smoothing, Bayes rule forces us to label this email as spam.

Given the conditional probabilities \( P(v_i|c), v_i \in V \) learned from the training data for \( c \in \{\text{spam}, \text{ham}\} \), we are ready to classify the unseen messages in the test set. Suppose we have a new email vector
If \( e = (m_1, \ldots, m_n) \), we can compute its classification as follows:

\[
c^* = \arg\max_{c \in \{\text{ham, spam}\}} P(c|e)
\]

We use Bayes rule to rewrite \( P(c|e) \) as:

\[
P(c|e) = \frac{P(e|c)P(c)}{P(e)}
\]

yielding

\[
c^* = \arg\max_{c \in \{\text{ham, spam}\}} \frac{P(e|c)P(c)}{P(e)}
\]

The denominator \( P(e) \) is constant, for both choices of \( c \) in the above equation, so we can simplify the classification computation as:

\[
c^* = \arg\max_{c \in \{\text{ham, spam}\}} P(e|c)P(c)
\]

\( P(c) \) is the prior probability of each class \( c \). \( P(\text{spam}) \) is the unconditional probability of an email message being spam, and \( P(\text{ham}) = 1 - P(\text{spam}) \). To compute \( P(e|c) \), we need to make the assumption of conditional independence between the tokens in the message, given the class \( c \). This is the naive Bayes assumption.

\[
c^* = \arg\max_{c \in \{\text{ham, spam}\}} P(c) \prod_i P(v_i|c)^{m_i}
\]

This conditional independence assumption is clearly violated in real-world data. However, despite these violations, empirically the naive Bayes classifier does a good job of classifying email. This observation is in part explained by the fact that classification estimation (whether it is ham or spam) depends on the sign of the function estimate and not its exact value. The word independence assumption causes naive Bayes to give extreme (almost 0 or 1) class probability estimates. However, these estimates can still be poor while the classification accuracy remains high. To prevent underflow that would result from the multiplication of many probabilities, it is conventional to convert the above product to a sum using the logarithm transformation.

\[
c^* = \arg\max_{c \in \{\text{ham, spam}\}} \log P(c) + \sum_i m_i \log P(v_i|c)
\]

Now a few words about feature selection. The tokens you can use for representation of email should include words in the body of the email text as well as mail header fields such as sender name and subject. Extracting and using tokens or features specific to headers is very worthwhile. There are other features such as the number and type of HTML or XML tags in an email, as well as the type of domain the message originated from, which can be predictive of the spam. I encourage you to add such features to the vector representation of email discussed above. Another interesting idea is to treat word pair and word triples as a single feature or token and count their occurrences as a unit. Such features are called bigrams and trigrams respectively. It is an interesting empirical question whether bigram features perform better than the single word or token counts (called
unigram features). You will need to write code to extract bigram and trigram counts from the body of the email text. You can modify our flex scripts to do so, or write your own.

As a practical matter, we cannot use all token (or token pairs, triplets) occurring in emails in the classification models. The learned models will be very sparse and not very accurate. The standard approach is to consider a small subset of words (or word pairs or triplets) that have high information content. Paul Graham recommends using a small number like 15, while the standard in the text processing community is to use a 100 or so. One standard measure for feature selection is mutual information $I(v_i; c)$.

$$I(v_i; c) = \sum_{c \in \{spam, ham\}} \sum_{v_i} P(c, v_i) \log \left( \frac{P(c, v_i)}{P(c)P(v_i)} \right)$$

where $P(c)$ is the number of token occurrences in emails of class $c$ divided by the total number of token occurrences in all emails in the training set (spam and ham), $P(v_i)$ is the total number of occurrences of the token $v_i$ divided by the total number of token occurrences in all emails, and $P(c, v_i)$ is the total number of occurrences of $v_i$ in documents of class $c$ divided by the total number of token occurrences in all emails. You can calculate the mutual information measure for the available tokens and pick the top $N$ for the vocabulary vector $V$.

## 2 The Assignment

Your goal is to implement and evaluate the naive Bayes approach to classifying email.

1. (20 points) Implement a simple feature selector. Use the mutual information criterion, and pick the top 10, 100 and 1000 tokens for the following experiments.

   We have provided code to get all tokens out of an email message. Each piece of mail in our ham and spam corpus is in a separate file. Every file contains a header followed by a body. The program `comp440` is a simple tokenizer for these files. It also produces counts of tokens in the file. It is invoked as follows from the Unix prompt %:

   ```
   % comp440 < input > output
   ```

   Provide a table showing the top 10 and top 100 features picked out by your program.

2. (40 points) Create a naive Bayes classifier. Make sure the code is on owlnet (and compilable on owlnet). Report the directory in which we can find your code. Explain how you built the classifier; (e.g., how you smoothed probability estimates). You can implement the Naive Bayes classifier in C, C++, Perl or Java.

3. (20 points) Run your classifier on the set aside test data, using the top 10, 100, and 1000 tokens, and provide the corresponding confusion matrix.

4. (20 points) Explain your results. What is your false positive rate? Paul Graham states that false positive rates over 0.05% are unacceptable. How well does your classifier fare? What kinds of email does your classifier mislabel? What improvement to the feature set can reduce these errors? Implement your suggestion and explain why it does or doesn’t improve classification accuracy.
5. (Extra Credit, 40 points) Performance gains can be made by following the corrections to the naive Bayes probability estimation, feature selection and classification processes suggested in Rennie et. al’s recent paper in ICML-2003 available at the URL: http://www.ai.mit.edu/ jrennie/papers/icml03-nb.ps.gz. Implement the suggestions from Section 3 of their paper (20 points), and from Section 4 of their paper (20 points). Run your new classifier and report the new confusion matrices. How low does the false positive rate get?