CS 585: Natural Language Processing
Fall 2004
Programming Assignment 2: Naive Bayes Classifier

Out: Tue, September 28, 2004
Due: Tue, October 12, 2004

Spam Filtering using Naive Bayes

Naive Bayes is a simple, effective machine learning solution to the problem of *document classification*. For this assignment, you will implement a Naive Bayes classifier to classify email messages as *spam* (junk mail) or *ham* (legitimate messages).

Data

The data was collected from `http://spamassassin.org/publiccorpus/`. For this assignment, use the slightly modified version found at `http://canberra.cs.umass.edu/~culotta/cs585/ass1-data.tgz` (8.3M). The data consists of 4150 ham messages and 1897 spam messages, with original header information intact.

Tasks

Read data: Read in all messages and store them in some efficient manner. You must decide how to *tokenize* each message – i.e. designate which characters indicate the start of a new "word." See `http://www.paulgrahm.com/spam.html` for one way of doing this (You will probably also assign each unique word an integer index you can use as an index into arrays of word counts and word probabilities). If you choose to do this assignment in Java, you can use the skeleton code for this task from `http://www.cs.umass.edu/~mccallum/courses/inlp2004/pa2-skeleton.tgz`

	Spam	Ham
Spam	TP	FP
Ham	FN	TN

Table 1: Confusion Matrix: TP = "true positive", TN = "true negative", FP = "false positive", 'FN = "false negative".

Split Data: Randomly split the data into a training set (70% of the messages) and a testing set (30%).
Train Classifier: Using the training set only, estimate and store the prior class distributions $P(spam)$ and $P(ham)$, as well as the conditional probability distributions $P(w|spam)$ and $P(w|ham)$. *It is crucial that you store probability measures as log-probabilities to avoid running out of floating-point resolution. This also means you need to do arithmetic in log-space. I.e., multiplications of probabilities become additions of log-probabilities.*
Test Classifier: Classify each message in the testing set as *spam* or *ham* according to the Naive Bayes formulation.
Evaluate Classifier: Evaluate the performance of your classifier using two methods: a *confusion matrix* and a *precision recall graph.* A *confusion matrix* summarizes the types of errors your classifier makes, as in Table **??**. Here, TP is the number of spam messages classified as spam, TN is the number of ham messages classified as ham, FP is the number of ham messages *mis*classified as spam, and FN is the number of spam messages *mis*classified as ham.
A precision-recall graph plots the classifier's precision at various points of recall, where *precision* = (TP + TN) / *testingSetSize*, and recall = TP / (TP + FN). To construct a precision-recall graph, sort the predicted messages in decreasing order of the posterior probability of the predicted class (i.e. in decreasing order of confidence that the classification is correct). Next, iterate through the list in descending order, and at each message, calculate the precision and recall of the classifications of the current message and messages already iterated through. Plot these (recall, precision) points on the graph.

Additional Experiments

Perform **2 of the following 4** experiments, using the same evaluation techniques as in the spam experiment.

- **Effects of train/test split**: Split data into 50% training, 50% testing (instead of 70-30).
- **Different word feature sets:** Try some different word features and investigate their accuracies. Try at least two different word features. Some examples include: (1) Parse each document to extract the `TO`, `FROM`, `CC`, and `SUBJECT` fields, and use only the tokens from these fields to represent each message, (2) Downcase all the words so that case information is lost, (3) Remove MIME attachments, (4) remove HTML tags, and (5) Remove words with fewer than two occurrences in your training set.
- **Alternate priors:** Instead of using "plus one" smoothing, try numbers different than one. Does performance degrade if you use numbers much smaller than one? How about larger than one? Is there some number other than one that gives higher classification accuracy than "plus one"?
- **Prune vocabulary size by information gain:** Instead of using all the words in the training data, use only the top 1000 words with the highest information gain (mutual information with the class label) found from the training set. Try the top 1000, 100, and 10 words, plus some others. Does the test set accuracy change? Why do you think so?

What to turn in

Code: Print out all source code written for the project.
Report: Write a 2 page report that includes the following:

1. Problems encountered - What difficulties did you have and how did you resolve them?

2. Tokenization method - If you did not use the Java skeleton or if you performed experiments with different word feature sets, how did you tokenize the input and why?

3. Experimental results - In addition to the confusion matrix and precision-recall graphs for each experiment, also include the overall accuracy for each.

4. Discussion - Explain your results. What additional experiments do you think would improve your performance?