CMPSCI 585
Programming Assignment 1

Out: February 5, 2003

Due: February 17, 2003

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 either *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. To do this 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 proabilities.)
Split Data: Randomly split the messages 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 $P(w|ham)$.
Test Classifier: Classify each message in the testing set as *spam* or *ham* according to the Naive Bayes formulation. Note that you will most likely run out of floating-point resolution unless you do the product over words

	Spam	Ham
Spam	TP	FP
Ham	FN	TN

Table 1: Confusion Matrix: TP = “true positive”, TN = “true negative”, FP = “false positive”, ‘FN = “false negative”.

in log-space, (i.e. as a sum of log-probabilities), then rescale before exponentiating and normalizing.
Evaluate Classifier: Evaluate the accuracy of your classifier on both the training data and the testing data 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 1.
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 / *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, split the list into n bins ($n = 10$ is usually sufficient). For each bin, calculate the precision and recall including *only* classifications in this bin *or* in bins with higher posterior probabilities. Plot these n points on the graph.

Additional Experiments

Perform at least **2 of the following 5** 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). Consider trying even less training data. What happens to training set accuracy? Why?
- **Different word feature sets:** Try some different word features and investigate their accuracies. For example, try at least two of the following, plus some additional variation you devise. (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, (5) Remove words that occur less than 2 times.

- **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 much larger than one? Is there some number other than 1 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). Try the top 10000, 100 words, top 10, and some others. Does the test set accuracy change? Why do you think so?
- **Create an automatic foldering tool:** If you organize your Inbox into folders, train and test a Naive Bayes classifier to classify your email based on the folder they belong to. The only difference between this and the spam experiment are the data and categories you use, and that you are expected to classify into more than two categories. How does the accuracy here compare with the spam data? Make some analysis of the differences.

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. Implementation experience - What questions and issues arose during your implementation. If there were difficulties, how did you resolve them?

2. Tokenization method - 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 - Discuss your experiments, explain your results. What additional experiments do you think would improve your performance?

Suggested timeline

- Thursday, February 5: Assignment given

- Tuesday, February 10: Implementation finished, baseline experiments in progress.

- Thursday, February 12: Two or more additional experiments in progress
- Tuesday, February 17: Report written, code printed, hand it in.