Peter Malmgren
Homework 2: Bayes Classification

I. Algorithm

a. Dictionary construction

The first step in text classification is to build a dictionary. The function to build a dictionary, getFeatures(), takes as input all the examples in the corpus and builds a dictionary with the following steps: First, all non-alphabetic characters are removed, all extra whitespace is removed, and all words are converted to lowercase. The string is then tokenized, and either inserted into the dictionary with a count of 1 if not present, or incremented in the dictionary if present. The returned dictionary is a sorted list of tuples (word,count) in ascending order.

The next step I performed was stop word removal (as per step 2 in the assignment description). Since getFeatures() returns words sorted by their frequencies, the dictionary is truncated, and then transformed into a set for fast lookup.

Note: only the training set is used for dictionary construction.

b. Vectorization of documents

Each document is processed in the exact same way as during dictionary construction (transformed to lower case, alphabetic characters only, tokenized). Every word present in the document and the dictionary was counted to form a feature vector.

c. Classifier training

I considered two options when building my list of posterior probabilities: binning, and fitting to a distribution. Binning takes each attribute, or word, and forms a histogram of occurrences. Take, for example, the occurrences of the word ‘cash’ in the classes spam and ham:

```
>>> itemfreq(spamTable['cash'])
array([[  1., 58.], [  2., 6.],[  3., 2.]])
>>> itemfreq(hamTable['cash'])
array([[  1., 13.]])
```

Because the word ‘cash’ appears more often, and sometimes more frequently than once in examples labeled ‘spam,’ three probabilities are computed: the probability that the word ‘cash’ appears once, twice, or three

times. Even though these probabilities are small, they're still larger than the default (the virtual probability).

A big design choice I made was assuming that if, like the example above, a word occurred in different frequencies in one document than another, I would assign the default probability to frequencies absent from the other class:

```
P(spam|'cash'=1) = 58 + p / |spam| + 1, P(ham|'cash'=1) = 13 + p / |ham| + 1
P(spam|'cash'=2) = 6 + p / |spam| + 1, P(ham|'cash'=2) = default
P(spam|'cash'=3) = 3 + p / |spam| + 1, P(ham|'cash'=3) = default
```

Also, the probability of spam if the word 'cash' occurred more than three times would be equal to the probability of the highest occurrence ('cash' = 3).

Alternatively, representing each word by a continuous distribution involves taking the sample mean and sample standard deviation (or other parameter estimation for different distributions) and estimating the probability P(C|d) with the PDF of the distribution.

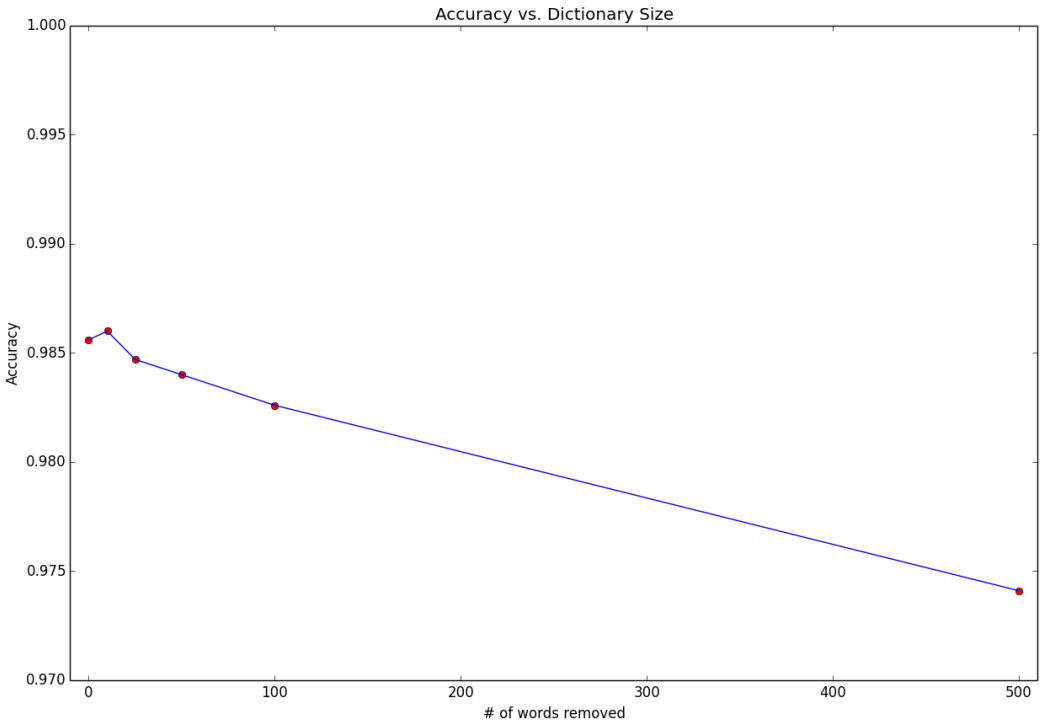
I chose to use binning vs. a distribution because it seemed simpler and worked reasonably well. Also, because most of the words didn't have a large distribution, occurring only once or twice, the distribution method seemed like overkill.

II. Experiments

a. Table of results from all experiments

Cutoff Metric	0 Words	10 Words	25 Words	50 Words	100 Words	500 Words
Specificity	.9983	.9981	.9973	.9958	.9958	.9919
Sensitivity	.9006	.9071	.9029	.9071	.8964	.8575
Precision	.9883	.9867	.9809	.9716	.9717	.9430
Recall	.9006	.9071	.9029	.9071	.8964	.8575
TP Rate	.9006	.9071	.9029	.9071	.8964	.8575
FP Rate	.0017	.0019	.0027	.0042	.0042	.0081
Error	.0144	.0140	.0153	.0160	.0174	.0260
Accuracy	.9856	.9860	.9847	.9840	.9826	.9741

b. Plot of accuracy vs. # of words removed from dictionary



c. Discussion

The most important metrics in this experiment were true positive rate and false positive rate. Designing a system to filter out spam text messages should not filter out ham text messages. It is better to let a few spam messages go through than to delete someone's real messages.

I believe that by using the binning strategy, the classifier I designed was consistently accurate across experiments. Words that appear lots of times in both categories (stop words) would have similar distributions, and the probability that they occur in one class vs. another class should be roughly equal. The first ten words of the dictionary seemed to be useless, but as more and more words were truncated the performance degraded, indicated by a higher false positive rate. Since the ham class had more examples, words that appear more often have a higher probability of being an informative feature of the ham class. Removing these words from our dictionary made it less sensitive to ham.

d. Problems

The main problem my classifier has is speed. Training only takes a few seconds. However, it takes roughly 60-70ms per new example calculation. This is because I used a dictionary, which uses open hashing (and probably had a lot of collisions), instead of a feature vector like a Numpy ndarray. I thought it would give a speed increase because each ‘vector’ just contained a word hashed to it’s count, but it seemed to slow things down.