EECS 349 Fall 2015
Jason Brown - Homework 5

Naive Bayes Classifier

1) The key steps in building the dictionary were:

   i) Convert the word to all-lowercase, as "Buy" rarely has a different meaning than "buy." Remove all symbols, like "?" or ".", from the end of the word.

   ii) If the word had more than 8 symbols in it, like "align=3Dleft>
< /B>
", ignore it, don't fill the dictionary with html formatting. The symbol count excludes "@" and ".", which are used in email addresses and are valid additions to the dictionary.

   iii) If the word in the spam training email wasn't already in the dictionary, add it, and assign it probability 1/(# of spam training emails)

   iv) If the word in the training email was already in the dictionary, increment it's numerator. I used python's fraction library.

   v) Once all words have been added, increment all numerators and denominators by one, to account for black swans. Just because it didn't show up in the training data doesn't make the probability zero.

2) The classifier determined whether the MAP point estimate of the email being spam was more likely than the MAP point estimate of the email being legitimate. It classified the email as spam if, for words a_1 through a_n,

$$(P(spam) * P(a_1|spam) * ... * (P(a_n|spam)) \geq (P(ham) * P(a_1|ham) * ... * (P(a_n|ham))$$

3) Yes, adding logs of numbers x & y will yield the same comparison as multiplying x & y, given the following equivalence: $b^{\log_b x + \log_b y} = x \cdot y$. I verified this in my algorithm as well by raising the sum of the logs to their base (2) and confirming this was equivalent to their products. I did not raise the sum of the logs to their base in the final algorithm to avoid underflow. Floating point numbers do not have infinite precision, even within denormalized value ranges. Therefore, given a number close enough to zero, it will eventually round to zero. This is underflow and an example of how rounding in the c language can influence outcomes. That is why sum of logs is used.

4) Design and Run an Experiment

   a) The goal of the experiment was to show that the null hypothesis, that there is no difference between the naive bayesian methodology and the prior probability methodology, is false. For building the dictionary, I used the spam and ham training data from SpamAssassin. Initially, I chose training data aligned with the percentage of spam, or 70% [200 spam emails, 85 legitimate emails]. However, upon testing this dictionary on a training set, it marked nearly every email, spam or ham, as spam! This was because the the prior probability factors heavily on the likelihood of a test email being spam. Since I didn't want to bias the dictionary, and to give it enough ham emails to learn, the split was 50/50 [200 spam training emails, 200 ham training emails]. This vastly

improved the performance of the spam filter which will be described belowTo get spam emails that we might see in the real world, a larger test set by a factor of 10, from different sources, would be better. Each test set was size 30 emails, comprised of 21 spam emails (70%, like in the real world).

b) The performance of the filter can be thought of as a bernoulli trial, with the email classification being either correct or incorrect. Each IID was of size 30 emails, and there were fourteen IIDs, representing the number of correct classifications: methodbayes = [29, 26, 18, 29, 27, 26, 30, 28, 29, 27, 30, 28, 28, 28]. The prior probability method classified everything as spam since prior probability was 70%, so it got 21 correct classifications for each IID. Using a paired t-test, since both methodologies were run over the same set of test emails, yielded the following result: The t-statistic is 7.990 and the p-value is 0.000. This p-value is statistically significant since it is $\leq 1\%$. Cross validation would have been used if this experiment was automated, but as it was, it took time to count, move, check results for each IID.To do each run, 9 good emails and 21 spam emails were pulled from the set of emails from SpamAssassin that were not used from training. The following script was run to change all the good emails to start with "g", so it would be easy to see whether they were put in the correct sorted folder: *for f in * ; do mv "f""g_f" ; done.*

c) After modifying the training set to be 50/50 spam/ham, yes, the spam filter does better than prior probability. With the training set split of 70/30, it did not work a training set of 285 emails. The results of the experiment was a p-value of 0.000 that is statistically significant since it is $\leq 1\%$, thus, we can reject the null hypothesis. The spam filter never classified ham emails as spam, but incorrectly classified on average 15-20% of spam emails as legitimate.