Naïve Bayesian Spam Classifier Report

Tyler Evans, 7770177

December 11, 2017

1 Implementation Notes

The implementation followed the approach outlined in [PL98]. In order to accomplish this, I had two classes of frequency tables, one for spam and one for non-spam. Then, within each of these classes I had a frequency table for just the email headers, and just the email body (removed header). This allowed me to only have to parse the files once (resulting in the header and body tables for each class) and then in the testing phase I could test combinations such as: only consider email bodies, only headers, or both (by simply merging the two tables).

The training function is set up so that you can pass a tokenizer of your choice (for example, the word based split and stem or the trigram based approach) as well as an email modifier that will affect all of the email trained on (a function that either returns the email header, body, or both). This modularity allowed me to test many combinations with the same training code.

In a similar way, the testing function accepts your choice tokenizer and email manipulator as well. The testing also requires to to provide it with the relevant frequency table. For tests to make sense, it must be ensured that you are using the same tokenizer and email manipulator for your testing that was used to generate the frequency table that you are using for the probabilities.

I found that using the feature selection outlined in the paper to filter out the frequency tables had a negative effect on the results. I am not sure if this is due to the use of different data than the paper (which could make the coefficients that they used irrelevant for our data) or my own implementation. Even after trying different coefficients in the feature selection filter, my results were still only negatively impacted. Due to this I decided to not include this strategy in my tests.

Note: After trying and failing to reliably create my own Porter Stemmer, I opted to use one implemented in the python nltk package. My program checks to see if nltk is installed on the system, if not it defaults to not stem words. All of the results for word based tokenization presented in this report were stemmed using the nltk Porter Stemmer.

2 Results

The base classifier uses word based tokenization and word stemming. Alterations to this method as well as their impacts on the results are discussed below. For results of individual alterations as well as certain combinations, see Tables 1-3. The actual output of the program has been provided in the file *output.txt*.

2.1 Alterations

2.1.1 Trigram vs. Word Based Strings

The results are presented in Table 1. As we can see, the trigram approach lead to a slightly higher error rate than the base approach of considering word based tokens. As stated in the SpamCop paper [PL98], this is probably due to the fact that trigrams do not carry as much information as the words do. However, one benefit of using trigrams was that in my implementation, they were much faster to train with and test on than the stemmed words.

2.1.2 Incorporating Header Data

The results are presented in Table 1. Initially, training and testing was done without using the email headers. Then, the same process was run using the full email (both headers and email body), and an improvement of nearly 2% was found for both word based and trigram based tokenizations. Finally (to see how far this would go), I trained and tested only the email headers. I was surprised when the results were significantly improved over both previous cases.

Upon visually searching a couple of spam and non-spam emails, I could not find a pattern that would immediately result in an email being classified as spam, but I am sure that there is some information in the header that is significantly helping out (perhaps all spam messages are tagged as spam somewhere, etc.)

2.1.3 More Recent Corpus

For an additional email corpus, I used the Webb Spam Corpus 2011 found at https://www.cc.gatech.edu/projects/doi/WebbSpamCorpus.html. It is more recent than the SpamAssassin data and worked perfectly for this application since (after extracting with the provided tool) the emails were similar in format to SpamAssassin (raw text, unlike many others that I stumbled upon).

Unfortunately, this data set only had the training emails labeled, so I had to discard the provided test data and form my own by partitioning the training data into a new training set and test set. I ensured that the ratio of spam to non spam as well as training to testing was similar to that of SpamAssassin for a fair comparison. After the repartitioning, I was left with about 3600 training emails and about 700 testing emails. Within each of these classes, there were about twice as many nonspam emails as spam emails.

The subject headers were different than SpamAssassin, and would have required a different method of separating the email body from the header. Because of this, I simply considered the entire email (header and body) and was able to run the existing classifier directly on the files.

See Table 2 for results. Notice that the newer corpus offered better performance for the two common cases that were tested.

2.1.4 New Modification

While training and testing on the email data, I noticed that on average, the spam emails were taking significantly longer to process. I reasoned that this was because they contained more tokens since all of the spam emails that I had visually inspected were comprised mostly of html (lots of symbols, which get broken down into many 3 character tokens). Compared to non spam email which generally did not contain html and were much simpler, I made the assumption that the number of tokens may be a helpful indicator in determining the spam.

To do this I modified the formula used to calculate $P(W|C)$. Initially, I tried multiplying the $1/k$ term in the numerator by the number of tokens in the email, but this ended up negatively affecting the results. Since this modification had too much influence in the formula, I decided to square k so that the new factor did not matter as much. Here is the modified formula that was used

$$P(W|C) = \frac{N(W,C) + T(W)\frac{1}{k^2}}{N(C)+1}$$

where $T(W)$ is the number of tokens in the email that W originated from.

I don't believe that this formula is proper ($P(W|C)$ should only depend on the word and the class, it shouldn't have anything to do with the email) but it was an ad-hoc modification that made sense to me and it turned out that it had a positive effect on the results as well, see Table 3.

References

[PL98] Patrick Pantel and Dekang Lin. Spamcop: A spam classification & organization program. *AAAI Technical Report*, 1998.

Tokens	Header	Spam Misclassified	Non-Spam Misclassified	Total Error
Stemmed Words	No Header	20.26%	0.38%	5.45%
Stemmed Words	Only Header	2.64%	0.22%	0.84%
Stemmed Words	Full Email	11.67%	0.08%	3.03%
Trigram	No Header	22.91%	0.60%	6.29%
Trigram	Only Header	4.63%	0.68%	1.68%
Trigram	Full Email	16.96%	0.38%	4.60%

Table 1: SpamAssassin Base Results

Tokens	Header	Spam Misclassified	Non-Spam Misclassified	Total Error
Stemmed Words	Full Email	13.04%	0.41%	4.43%
Trigram	Full Email	17.39%	0.41%	5.82%

Table 2: CDSMC2010 Results

Tokens	Header	Spam Misclassified	Non-Spam Misclassified	Total Error
Stemmed Words	No Header	16.52%	0.15%	4.32%
Stemmed Words	Only Header	1.98%	0.15%	0.62%
Stemmed Words	Full Email	8.81%	0.08%	2.30%
Trigram	No Header	20.48%	0.60%	5.67%
Trigram	Only Header	3.74%	0.83%	1.57%
Trigram	Full Email	13.88%	0.38%	3.82%

Table 3: SpamAssassin Results with Modification