CS545 Project Report: Comment Spam Identification

Eric Cheng (eric.cheng@yale.edu)
Eric Steinlauf (eric.steinlauf@yale.edu)

May 5, 2007

1 Introduction

Comment spam is prevalent in blogs, wikis, and every possible web medium which allows people to freely post content. They generally contain links to the spammers' websites, and more subtly, they are also sometimes manipulated to boost the search engine rankings of their target sites. These messages not only annoy web users, but also pollute web pages and waste Internet bandwidth. According to Akismet, 95% of blog comments are spam [1].

Various methods have been proposed to combat comment spam, but each has its drawbacks. For example, to circumvent blacklisting, spammers may register new domain names or simply generate random ones, or even just use a dynamic IP address. CAPTCHAs (Complete Automated Public Turing test to tell Computers and Humans Apart) are popular among modern websites. Although they seem to provide protection from spammers, naively designed CAPTCHAs can be solved by machine learning algorithms, and overly complicated CAPTCHAs place too much burden onto website visitors. Keyword filtering can be effective, but the web site administrator has to manually provide a list of blocked keywords, and it sometimes filters out legitimate comments as well.

We propose a model that has worked well for classifying e-mail spam, which is the naive Bayes algorithm. Statistical approaches to identifying email spam have worked well, such as DSPAM [2] and Spamassassin [3]. Unlike keyword filtering, each word in the comment carries its own weight, and words in a comment collectively determine the classification. This could potentially lower the number of false positives, where a legitimate message contains some "spam" words. In this report we will provide details for our approach, and also show how various parameters contribute to the performance and accuracy of the algorithm.

2 Naive Bayes

Naive Bayes classifiers are based on Bayes' theorem with the assumption that each feature is independent of every other feature. It has been applied to document classification, especially

e-mail spam filters. In this section we will briefly explain the naive Bayes model and show how we apply it to our problem.

Given a conditional model

$$P(C_i|F_1, ..., F_m)$$

over classes C_1 through C_n depending on feature variables F_1 through F_m. By applying Bayes' theorem, we have

$$P(C_i|F_1, ..., F_m) = \frac{P(C_i)P(F_1, ..., F_n|C_i)}{P(F_1, ..., F_n)}.$$

Assume that each feature F_j is independent of every F_k where $j \neq k$, then

$$P(F_1, ..., F_m|C_i) = \prod_{j=1}^{m} P(F_j|C_i).$$

In our case, we want to automatically distinguish between spam and non-spam comments, hence we define two classes, *spam* and *ham*. We break a comment into a list of tokens, and each token is treated as a feature of the comment. The equation then becomes

$$P(spam|comment) = \prod_{i=1}^{n} P(w_i|spam)P(spam)/P(comment)$$

where x is the number of times w_1 appears in all spam messages in the corpus, y is the total number of words in all spam messages in the corpus, and n is the length of the given comment. We naively assume that each word occurs independently, which is generally not the case for most human languages. Nevertheless, this assumption makes computation simple and efficient. We show how we compute each term in the equation.

$P(w_i|spam)$ denotes the probability of w_i occurring given a spam comment. Given a corpus of legit messages and spam messages, we iterate through each word and compute x and y. When a new message of length n is given, we claim that $P(w_i|spam) = 1 - (1 - x/y)^n$. This takes into account that the same word may occur multiple times in a message and the corpus, and messages can be of different length.

$P(spam)$ is approximated by dividing the number of spam messages by the total number of messages in the corpus.

$P(comment)$ is really an unknown value, since it is hard to conceive the probability of a particular comment occurring. Fortunately, this term also exists in the computation of $P(ham|comment)$. Since what we are really interested in are the likelihoods between a comment being spam and ham, we can ignore $P(comment)$ in both terms and just compare the two products without it.

In practice, $P(w_i|C)$ and $P(C)$ may be small values, and underflow errors may occur when multiplying over many terms, especially when the number of features is large. To avoid numerical errors, we compute the logarithm of both sides, which yields

$$\log P(ham|comment) = \sum_{i=1}^{n} \log P(w_i|ham) + \log P(ham) - \log P(comment)$$

and

$$\log P(spam|comment) = \sum_{i=1}^{n} \log P(w_i|spam) + \log P(spam) - \log P(comment).$$

It turns out that we can actually compute $P(ham|comment)$ and $P(spam|comment)$ even if we avoid computing $P(comment)$. It's worth noting that in our special case where there are only two classes, spam and ham, $P(spam|comment) = 1 - P(ham|comment)$. Let $m = \log P(spam|comment) - \log P(ham|comment) = \log P(spam|comment)/P(ham|comment)$. Then $e^m = P(spam|comment)/P(ham|comment)$, hence $P(spam|comment) = e^m P(ham|comment)$. By substituting we get $P(ham|comment) = 1/(e^m + 1)$ and $P(spam|comment) = 1 - P(h)$. Since the value of m does not change whether or not we ignore $P(comment)$, we can get away without computing $P(comment)$.

3 Optimizations for Comment Filtering

Various methods were used in order to tune the Bayesian filter specifically for comment spam. The idea was to find the way to best utilize the available information. The first consideration was finding the best way to break a message into individual tokens. The simplest separator would be whitespace, but it is likely that the punctuation and other symbols left in tokens would result in many occurrences of words that a human would identify as being the same. There was also the possibility that certain symbols would be good predictors of spam. Angle brackets seemed like a good candidate for inclusion in tokens since they are used in HTML tags. Knowing that a word is part of an HTML tag may be valuable. In general, comments are posted in response to an article. It seems reasonable that the comments should be related to the article. It was hoped that including the article as an example of a clean comment would improve the false positive rate. Another possible method of improving the false positive rate was to double count ham comments [5]. By counting each token in ham comments twice, they should be stronger predictors of ham. This would increase the probability of a message being classified as ham while decreasing the probability that it is spam. This may be an acceptable tradeoff, if spam comments still contain words that are strong predictors of spam.

In addition to the body of the comment, there is also valuable information contained in the comment header. This header varies from site to site, but in general it contains the poster's name, a subject, and possibly a link to a site of the poster's choice. This information can be included in various ways. Tokens that appear in the header can be treated the same as tokens that appear in the body or they can be tagged as being part of the header [5]. This may be useful if a certain token, such as a certain domain name, is often found in the body of ham comments, but never in the header of ham comments. Header token could be included using both methods in order to capture all possible correlations.

There are many words that appear in all comments with equal probability. The filter can just focus on the words that are strong predictors of spam or ham [5]. This is done to avoid dilution of the a strong result. If a comment contains a few strong spam indicators,

the presence of lots of articles and prepositions should not detract from its classification as spam.

A boosting algorithm was used on top of the standard Bayesian classifier. Boosting applies the Bayesian classifier repeated while varying the weights of each message used for classification. The first time the classifier is run, each message is weighted equally. The resulting model is then used to predict the value for each comment in the training set. Each comment that was correctly classified has its weight decreased and the classification is run again. This is done repeatedly in order to improve the classification of comments that were initially classified incorrectly. This produces a series of Bayesian models, each with an associated error value. Together this creates a weighted voting model. When it is used to predict if a comment is spam, each individual Bayesian model make as prediction. Each model is weighted by its accuracy and the prediction that has the largest weighted vote is selected. [6]

4 Results

The data set used was obtained from [7]. It consisted of 1020 individual comments collected from 49 blog pages. Each comment was manually classified as either spam or ham. The best overall accuracy attained was 94.1%. This corresponded to a false positive rate of 1% and a false negative rate of 15.2%. This was obtained by including letters, <, and > as valid token characters. Also, the header was included in both the tagged and untagged form, the article body was not included as a clean comment example, clean comments were not double counted, no boosting was used, and all words were used to classify each comment. The following results show how various parameters effected the accuracy of the filter. They show how shifting away from this base effects the results.

Figure 1 compares the results of filtering with and without double counting the clean comments. On this test set, double counting has very little effect. Figure 2 compares the results of using different methods of including the words in the comment header. *Include* means that the header tokens were included with no special tags and were treated equivalently to tokens in the body of the comment. *Tag* means that the tokens were tagged as being from the header so that the Bayesian filter would treat them as being different tokens. *Both* means that both of these methods were used to evaluate the tokens in the header. Figure 3 shows the results of including and not including the main article body as an example of a clean comment. Again, this has a very weak effect.

Figure 4 shows the effects of using various regular expressions to define the separators between tokens. The left regular expression treats anything other than a letter or an angle bracket as a separator character. The middle treats anything other than a letter as a separator, and the right expression treats non-letters and non-digits as separators. Again the effect of this parameter is small. Figure 5 shows the results of including varying numbers of significant words. *None* indicates that no filtering was performed and therefore all words in the message were used. The best results were clearly obtained by using all words. Finally, figure 6 shows the results of using different levels of boosting.

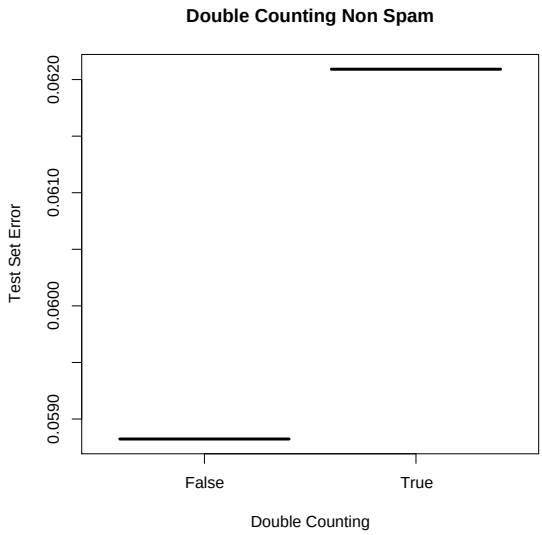

Figure 1: Double counting positive comments.

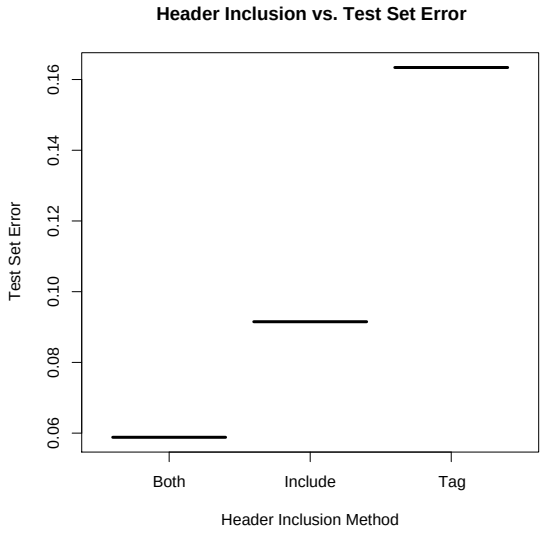

Figure 2: Various methods of including the comment header.

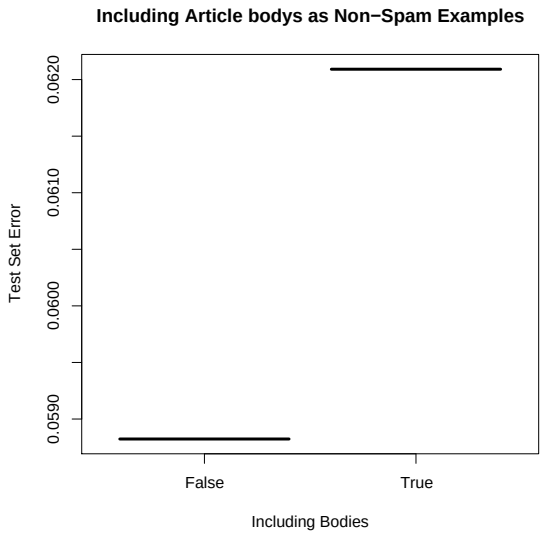

Figure 3: Inclusion of article body.

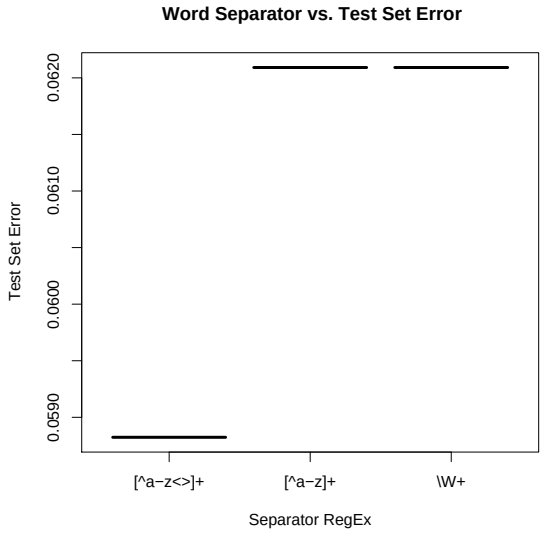

Figure 4: Various methods of separating words.

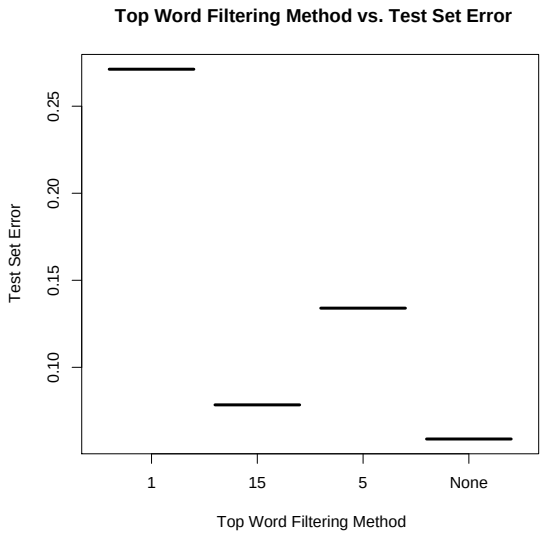

Figure 5: Prediction using most significant words in each comment.

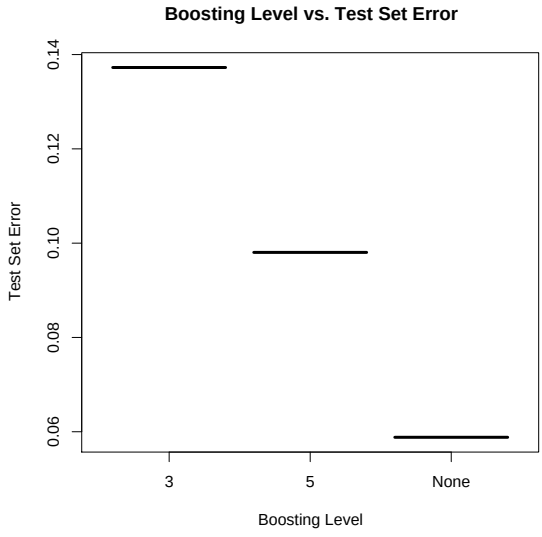

Figure 6: Spam prediction using boosting.

A basic proof of concept web service was also implemented. It provided fields for a user to enter a name and email address and a comment. Bayesian filtering was then applied to the message to in order to predict if it was spam. While this is not particularly useful in itself, it shows that this type of system could easily be integrated into blogs and message boards so that comments are filtered before they are posted.

5 Future Work

In this section we will discuss some directions in which our system can be improved.

First, we believe it will improve the accuracy of the filter if we follow the web links in a comment and take into consideration the content of the target website. This is due to the nature of comment spam where the advertised websites usually contain even more spam words.

The system will also benefit from a more sophisticated word stemming algorithm. Currently, we do not perform any stemming at all. The intuition is that if the algorithm can treat different pluralizations as the same word, the size of the training set required to build a good classifier may decrease.

To make our system ready for general consumption, we still need to add the ability to tune various parameters, the ability to mark any comment as spam or non-spam, and the ability to add incoming comments into the corpus, all in a user-friendly manner.

6 Discussion and Conclusion

In general, Bayesian filtering works well for filtering comment spam, which was expected since it has proved effective as an email spam filter. The attempts to tune the filter produced some surprising results. Double counting clean comments was hypothesized to reduce the occurrence of false positives, possibly at the expense of increased false negatives. This did not appear to be the case. Figure 1 shows a very slight advantage for overall accuracy by not double counting. This technique produced decreased accuracy without improving the false positive rate. The false positive rate was 1% with and without double counting ham comments. Including the article body was also hoped to reduce the rate of false positives and it also did not accomplish this goal.

Boosting also did not prove effective on this test set. There are multiple reasons that boosting may not be an appropriate technique for comment filtering. Since the training set was relatively small, it may just have been clamping to some particular features while not improving the performance on the test set. It also does not lend itself well to the way that a comment spam filter would actually be used. Since the filter is constantly processing new comments, it should be easy to add comments manually flagged as spam or not spam to the body of knowledge in order to improve the performance of the filter. A boosted model is

not easily modified in this way. The simplest way to augment the body with a new message would be to completely re-execute the boosting procedure. This would not be a problem when performed on a a small set of comments such as the training set used here, but this could become an expensive operation as the body of comments grows. The ability to learn and a adapt is one of the greatest assets of a Bayesian classifier. Given this, boosting appears to be unnecessary.

Bayesian classification has worked well for email spam filtering and this study has found that there is sufficient information in comment spam to detect it using a Bayesian classification model. The accuracy rates in this study are believed to be minimum. This is due to the small size of the training and testing sets. Given a large and growing training set, it is believed that this method could achieve even higher accuracy rates. While the attempts to tune the model specifically for comments were generally unsuccessful, the basic model should still be considered a useful and powerful method of dealing with comment spam.

References

[1] http://akismet.com/stats/. Retrieved April 22, 2007.

[2] http://dspam.nuclearelephant.com/

[3] http://spamassassin.apache.org/

[4] Graham, Paul. A Plan For Spam. August 2002.
http://www.paulgraham.com/spam.html

[5] Graham, Paul. Better Bayesian Filtering January 2003.
http://www.paulgraham.com/better.html

[6] Russel, S. and Norvig, P. Artificial Intelligence: A Modern Approach. Prentice-Hall, Upper Saddle River, New Jersey. 2003.

[7] Toy Corpus of Spam in Blog Comments.
http://ilps.science.uva.nl/Resources/blogspam/