Variable Thresholding In Naïve Bayesian Spam Filters

Sherman Braganza
Department of Electrical and Computer Engineering
University of Waterloo
Waterloo, ON, Canada.
sbraganz@uwaterloo.ca

Abstract – *Email has become an essential means of communication for both business and personal use. However, the proliferation of unwanted email advertising or spam has cost organizations millions of dollars and has reduced the effectiveness of email as a communications medium. Recently, spam filters have been widely adopted as a means of combating these unwanted messages. This paper presents a method for better spam detection by combining the classical naïve Bayesian filter with a neural network that analyzes various characteristics of the email body. The results are analyzed and the method deemed effective in conditions where very strong thresholds must be set or where the training data is not exhaustive.*

Keywords: spam, filtering, naïve Bayes, neural network, variable thresholding

1 Introduction

The proliferation of unsolicited email or 'spam' has led to a resurgence of active research in the domain of text classification. As a result, spam filters, mostly based on the naïve Bayesian assumption, have become highly effective tools in preventing spam from ever reaching user inboxes. However, spammers are becoming increasingly adept at avoiding such filters. To this end, it is becoming necessary to augment simple naïve Bayesian filters with heuristics or other machine intelligence algorithms to achieve sufficient performance.

2 Related Work

The origins of the renewed interest in naïve Bayesian techniques as applied to spam can be traced back to a paper published by Sahami et al.[1] detailing a method for classifying spam. This method focused on a combination of domain specific features and probabilistic weighting to achieve performance numbers in excess of eighty percent. Although providing accuracies lower than rule based filters such as SpamAssassin, this method introduced the concept of asymmetrical weighting in spam filtering, that is, false positives are more unacceptable than false negatives.

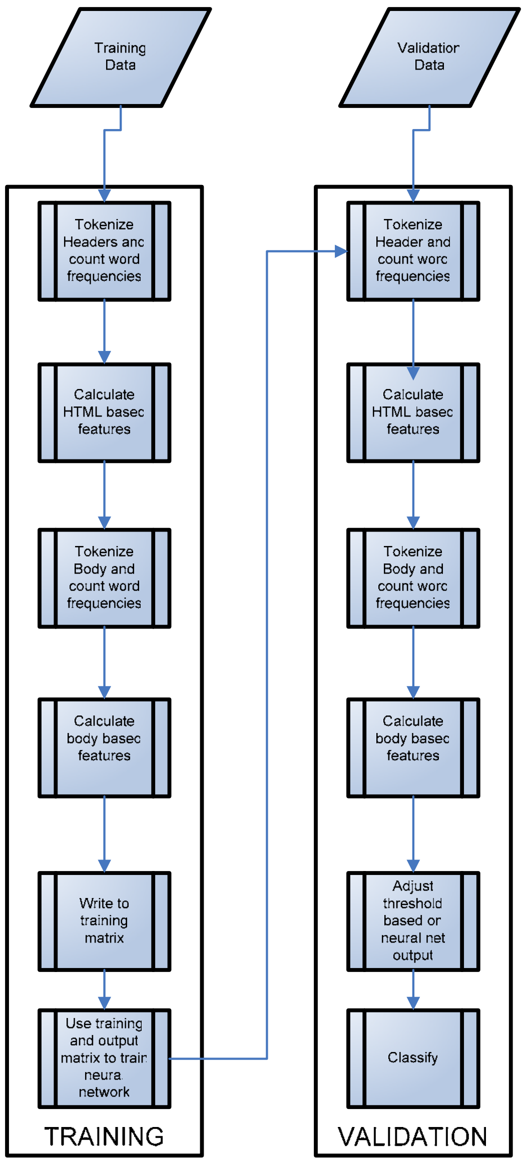
Further work was performed by Androutsopoulos et al.[2][3] in this field, but it was an essay by Paul Graham[4] detailing his technique for building effective spam filters that began to see naïve Bayesian classifiers appearing in commercial grade products. The Graham method involves assigning weights to certain words depending on their 'spamminess'. The classification of an email as unwanted email or 'spam' results from the application of a modified naïve Bayesian equation to the fifteen words most heavily weighted away from an 'uncertain' result of 0.5. This method combined with simple heuristics for analyzing header information generally scores over 90 percent on most publicly available corpora.

The Graham technique has been adapted and modified for better results in filters such as BogoFilter and SpamBayes, mostly by adapting the feature set and using various combining methods such as the chi-square distribution [5]. Other techniques utilizing super-increasing weights and variable window sizes have also been implemented providing accuracies in the 99.9% range. However, most adaptations still use a hardcoded asymmetric weighting or threshold for distinguishing between spam and non-spam. Presented below is a method for dynamically adjusting this threshold based on non-word based features of the email.

3 Proposed Technique

Variable threshold levels in spam filters could be used to implement time based filters as well as white-lists, black-lists, feature analysis along with a number of other possibilities. Time-based filters as applied to spam are most applicable to ISP grade email servers where a large number of incoming spam messages may occur in a short period of time as spammers burst sizable quantities of email to various users. White-lists and black-lists can be accomplished by setting the threshold to arbitrarily large or small values. The last application mentioned above, feature analysis, is what will be discussed in this paper. In this implementation, the threshold of a basic naïve Bayesian classifier is adjusted by a neural network trained on certain non-word based email features.

The filter design is very similar to that outlined by Graham. The flow of data is given below for the variable threshold implementation:



Training Data
Validation Data
Tokenize Headers and count word frequencies
Calculate HTML based features
Tokenize Body and count word frequencies
Calculate body based features
Write to training matrix
Use training and output matrix to train neural network
TRAINING
Tokenize Header and count word frequencies
Calculate HTML based features
Tokenize Body and count word frequencies
Calculate body based features
Adjust threshold based on neural net output
Classify
VALIDATION

Figure 1: Filter Design

The purely naïve Bayesian filter skips the threshold adjustment step during validation and instead uses a constant value.

3.1 The Naïve Bayesian Classifier

The spam filtering naïve Bayesian classifier is binary in nature, categorizing input text as either spam or not spam. All incoming input emails are broken down into individual words where each word corresponds to a feature. The features, numbering n in total, are combined into a feature vector denoted by $\vec{X}$. The classification itself is based upon the theorem of total probability and Bayes theorem.

$$P(C = Spam \mid \vec{X} = \vec{x}) = \frac{P(C = Spam)P(\vec{X} = \vec{x} \mid C = Spam)}{\sum_{k \varepsilon spam, non-spam} P(C = k)P(\vec{X} = \vec{x} \mid C = Spam)} \quad (1)$$

The equation given in (1) tends to be very difficult to work with in real life due to the fact that the probability P($\vec{X}$ | C) has a large number of possibilities to deal with. Thus the naïve Bayesian assumption given by (2) is used.

$$P(C = Spam \mid \vec{X} = \vec{x}) = \frac{P(C = Spam)\prod_{i=1}^{n} P(X_i = x_i \mid C = Spam)}{\sum_{k \varepsilon spam, non-spam} P(C = k)\prod_{i=1}^{n} P(X = x \mid C = k)} \quad (2)$$

This simplification is made possible by the assumption that all the features in the feature vector are conditionally independent of each other given the classifier C. Thus the joint probability is the product of the probability of each of features.

Using a naïve Bayesian classifier, the training procedure would work as follows. Given two preferably large corpora of ham and spam, the frequency of individual words in each corpus is measured and stored along with the total number of spam and ham emails. These are then used in the validation phase to generate the probabilities of specific words occurring in spam and ham, which is in turn used by the naïve Bayes equation to classify emails.

3.2 Grahams method

Graham's method as implemented in this project is similar to the method outlined above with a few key implementation based differences. First, the feature vector used here is limited to a maximum of fifteen words. The reasons for including this limitation is that machines have finite word lengths and multiplying a large number of low probability words can lead to the number being rounded to zero. Another problem is that classical naïve Bayesian methods, especially ones based on Graham's algorithm, produce strongly positive or strongly negative results even in the case of false positives as shown in Figure 2.

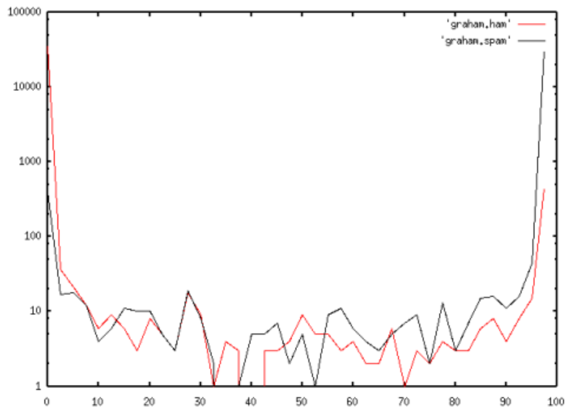


Figure 2: Graham filter output [6]

Graham's modification of the naive Bayesian equation is as follows:

$$P(C = Spam \mid \vec{X} = \vec{x}) = \frac{\prod_{i=1}^{n} P(X_i = x_i \mid C = Spam)}{\prod_{i=1}^{n} P(X_i = x_i \mid C = Spam) + \prod_{i=1}^{n} (1 - P(X_i = x_i \mid C = Spam))} \quad (3)$$

Equation (3) is based on the assumption that the number of training emails for both spam and ham are equal and that the probability of an email being ham is one minus the probability of it being spam. The probability that any one word is spammy in nature is calculated as in the naïve Bayes case, with the formula

$$P(C = Spam \mid x_i) = \frac{\frac{b}{NumberOfBad}}{\frac{b}{NumberOfBad} + \frac{g}{NumberOfGood}} \quad (4)$$

Where b is the number of times a word occurs in the bad corpus and g is the number of times a word occurs in the good corpus.

3.3 Corpus

The body of training and validation data used in creating this filter was taken from the SpamAssassin public corpus. This provides a publicly available corpus against which the performance of various spam filters can be measured. In addition, the SpamAssassin corpus provides full header listing unlike the LingSpam or PU1 collections.

The SpamAssassin corpus has been generally recognized as a difficult one [7] given that it contains text from various members of the team responsible for designing the popular filter and tends to contain examples of spam in legitimate emails. It thus requires careful thresholding in order to keep the number of false positives minimized. This body of text also has a large number of examples of both spam and valid email to work with.

3.4 Features

The feature set is broken down into space delimited word features and email characteristics that are non word based. The entire header and body are scanned in. HTML tags are not included in the word feature vector as they unnecessarily penalize HTML based email given that the majority of spam is in HTML formatting. Stop words were not utilized since the most common words tend to occur evenly in spam and in non-spam and therefore do not qualify to be included in the top fifteen feature vector. The tokenizer differentiates between upper-case and lower-case and also treats punctuation as valid data, thereby capturing as much information as possible. An approach like this only makes sense however, when there is a large training corpus available so that even relatively rare combinations show up frequently enough to be of statistical use.

The non-word features include the frequency of occurrence of spaces, empty lines, exclamation marks, periods, images, links and a variety of other similar metrics for both the header and the body of the email. These features are used to weigh the threshold in the case of the adaptive filter.

3.5 Neural Network Design

The primary goal of the neural network is to approximate a nonlinear function that can accurately map the input feature vector to the output. Simple regression could not be used since the nature of the function itself is not known.

The neural network implementation was accomplished using the Matlab Neural Network toolbox. For several reasons, the network type chosen was a dual layer feed forward back propagation network. First, such networks provide a significantly faster training time when compared to Perceptrons in Matlab, since they provide access to momentum based gradient descent learning (TRAINLM). Momentum allows the neural network to ignore small features in the error [8]. Secondly, the dual layer architecture allows for a better approximation of the correct non-linear function, given that it has been proven that a dual layer network can approximate any non-linear function [9].

Before training the neural network with the non-word feature vector, Principal Component Analysis (PCA) was performed on the input data. This is necessary given the very large size and correlated nature of the input vector. PCA trims the input data by deleting all highly correlated features. It also orders the data set by the degree of variation and removes the low variation examples. A variation cutoff of 0.4 % was used.

The performance of the network was gauged by running the neural network on a set of validation data after training was completed. This step ensures sufficient generalization for maximal performance. A final network of 10 x 1 was selected and a tan sigmoid transfer function was used to generate an answer in the +1 to -1 range. The topology of the network is depicted below in Figure 3.

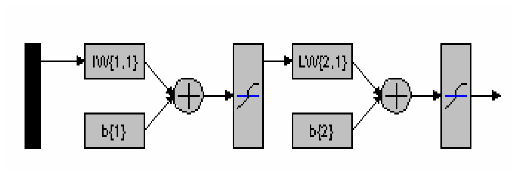


Figure 3: Neural network topology

3.6 Variable Thresholding

As mentioned before, Graham's method as implemented in this project tends to produce results heavily skewed towards 1 or 0, even in the case of false positives or false negatives. Thus it becomes necessary to have a threshold value that can vary over a large range and have a large base value. To this end, the following format for the threshold (λ) is proposed:

$$\lambda = 10 \wedge (b - range*weight) \qquad (5)$$

This format of exponential range for the threshold can be varied depending on the base filter type. In this case, the base value 'b' is set to a static value and the range was set to b/5. The weight value itself was generated by means of a feed forward back propagation neural network as mentioned previously.

The classification itself is performed as in the Graham method outlined above. If the probability of spam is greater than the product of λ and the probability of ham, the email is tagged as spam.

4 Results

In total, 2416 examples of ham and 2532 examples of spam were used resulting in a total corpus of 4948 emails. Of these, 1000 spam and 1000 ham emails were used for training. Another 1000 spam and 1000 ham were used for validation. The remaining 416 ham and 532 spam emails were used for testing.

The training method for the neural net was set to TRAINLM as mentioned before. The performance function was the Mean Squared Error (MSE). The neural network was trained for 100 epochs with a goal of 0. Various other parameters and design decisions are discussed above in section 3.5.

The performance graph of the neural net is given below in Figure 4. After a hundred epochs, the performance settles to 0.097. Even though a number much closer to zero could be achieved by increasing the number of neurons in the hidden layer, such solutions lead to dramatic over-fitting. In this case, output values very close to perfect spam/non-spam are reached for the majority of the test cases when the number of neurons is increased beyond 10. This leads to increased false positives as the neural net by itself possesses insufficient information to make a fully informed classification. Ideally, some uncertainty should be preserved, that is, some output values should be around zero on a -1 to +1 scale.

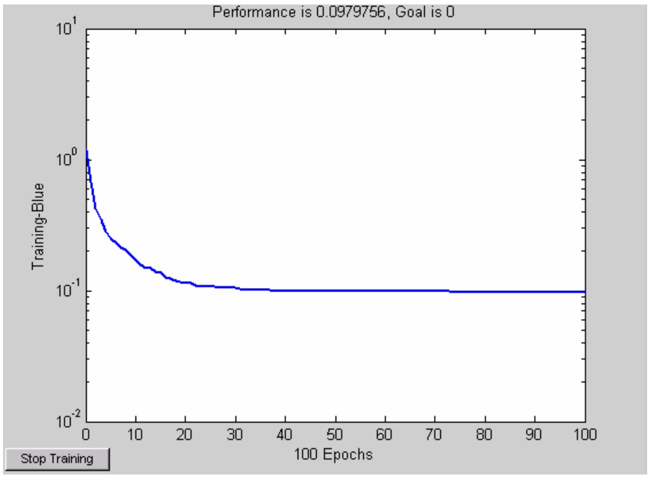


Figure 4: Neural Network performance

This network was deemed acceptable after much iteration through the training/validation cycle. Overall it provided the best balance between generalization and performance.

After training the naïve Bayesian filter and the neural network, a series of validation experiments were performed against a variety of static and dynamic threshold values. The performance of the filter as it varies with threshold is given below, for the static threshold case.

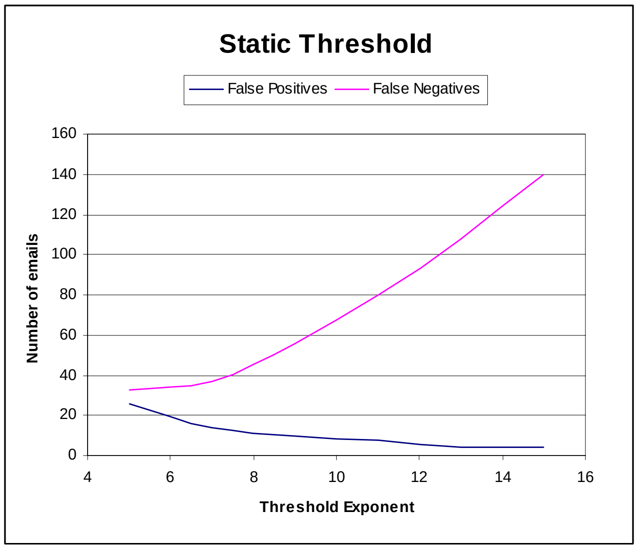


Figure 5: Static Threshold

As can be seen, the performance degrades drastically after passing the 10 ^ 7 mark. The number of false negatives starts to increase rapidly while the number of false positives remains relatively unchanging.

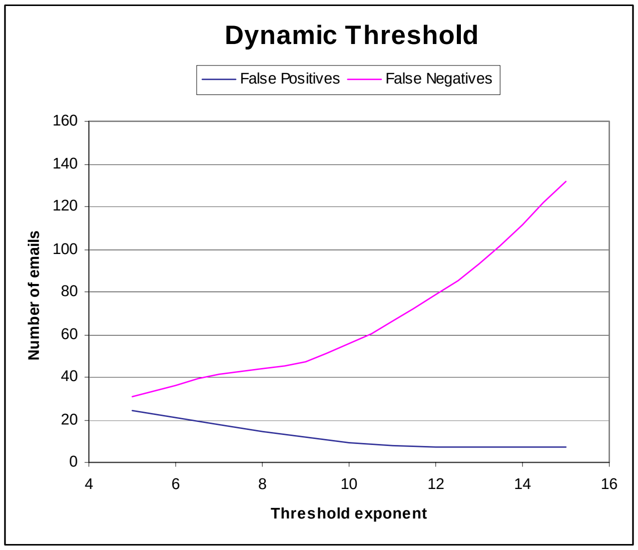


Figure 6: Dynamic Threshold

The situation here is similar to the case of static threshold, however the rate at which the number of false positives increases by is less pronounced. This implies that at larger threshold values, that is, with very strong weighting against false positives, the augmented filter has better performance.

In both cases, with two thousand validation emails and a reasonable threshold value of less than 10^12, overall performance for both filters was greater than ninety percent on the validation data. A base threshold value of 10^ 7 was chosen based on the above graphs. At this point false negatives appear approximately twice as frequently as false positives and the error rate is about 97%. In terms of time performance, Matlab classified two thousand emails in a little under ten minutes which corresponds to three to four emails per second, sufficient for small email servers but not for ISP grade applications.

As a final test, the filter was used to classify another 948 previously unseen test emails of mixed ham and spam based on the threshold above. The static filter made 5 false positive and 117 false negative mistakes, which corresponds to an overall error of 87.13%. The dynamic filter made 4 false positive and 108 false negative classifications. This corresponds to an overall error of 88.19%. Both of these percentages are significantly lower than that of the validation data. After manually checking over the errors, it quickly became apparent that the nature of the spam email was different in the test body than in the training data, containing higher ratios of encoded email and other spam-avoidance tricks. Additionally, there are more spam emails than ham in the test corpus, which skews the value of the number of false negatives. In such cases, both filters erred on the side of caution and classified the uncertain emails as spam, but the augmented filter outperformed the static one by a small margin in both false positives and false negatives.

5 Conclusions

The static filter performs well until the threshold between spam and non-spam is set to a large value, beyond which point the number of spam emails delivered to the user's inbox starts to grow rapidly. The dynamic threshold scenario suffers from the same weakness; however the rate at which spam emails get through is not quite as pronounced. The augmented filter also performs better in situations where the input email is significantly different than any examples covered in the training data. The overall impact of the dynamic thresholding as implemented here is not very significant, mostly due to the limited non-word based feature set, but the general trends discussed in this paper indicate that this technique is fairly promising given the conditions described above.

References

[1] Mehran Sahami, Susan Dumais, David Heckerman and Eric Horvitz. ``A Bayesian Approach to Filtering Junk E-Mail." Proceedings of AAAI-98 Workshop on Learning for Text Categorization, 1998

[2] Ion Androutsopoulos et al., "An Evaluation Of Naïve Bayesian Anti-Spam Filtering", Proceedings of the workshop on Machine Learning in the New Information Age, G. Potamias, V. Moustakis and M. van Someren

(eds.), 11th European Conference on Machine Learning, Barcelona, Spain, pp. 9-17, 2000.

[3] I. Androutsopoulos, G. Paliouras, V. Karkaletsis, G. Sakkis, C.D. Spyropoulos and P. Stamatopoulos, "Learning to Filter Spam E-Mail: A Comparison of a Naive Bayesian and a Memory-Based Approach". In H. Zaragoza, P. Gallinari, and M. Rajman (Eds.), Proceedings of the Workshop on Machine Learning and Textual Information Access, 4th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD 2000), Lyon, France, pp. 1-13, 2000.

[4] Paul Graham, “A plan for spam”, http://www.paulgraham.com/spam.html, August 2002.

[5] Gary Robinson, “A Statistical Approach To The Spam Problem”, The Linux Journal, http://www.linuxjournal.com/article/6467, 2003.

[6] SpamBayes, “Background: Combining and Scoring”, http://spambayes.sourceforge.net/background.html. 2003.

[7] William S. Yerazunis, “The Spam Filtering Accuracy Plateau At 99.9% And How To Get Past IT”, MIT Spam Conference, 2004

[8] Mathworks, “Documentation: Neural Net toolbox”, http://www.mathworks.com/access/helpdesk/help/toolbox/nnet/, 2005

[9] Hornik, Stinchcombe, White, “Multilayer feedforward networks are universal approximators”, *Neural Networks*, Volume 2 Issue 5 pp. 359-366, 1989.

[10] Ethem Alpayadin, “*Introduction to Machine Learning*”, The MIT Press, Cambridge, London, 2004.