Creating a Naive Bayes Classifier for Spam Filtering

Nathan Moeller

December 11, 2014

1 Abstract

Every email user who receives spam messages and can attest to how annoying it is. Spam clutters inboxes and makes it difficult to find emails that are actually important. This is why techniques such as spam filtering were made to detect unwanted emails. Due to the similar language of spam emails, I was able to create a simple Naive Bayes spam filter with multiple versions of the Ling-Spam data set [1]. I achieved very successful results for my Naive Bayes classifier, so I concluded that Naive Bayes is a sufficient way to filter out most simple email messages.

2 Introduction

Spam emails have always been an annoying problem. However techniques such as pattern matching and text classification have been able to successfully filter out some spam emails. This is not to say the problem has been completely solved though. Spam emails are still a successful way of Internet marketing. This seems surprising as most people simply delete spam-like emails. However due to their massive audience and low cost, spam still works. According to an article published by MAAWG (Messaging Anti-Abuse Working Group) 16

% of people clicked on spam because they were "interested in the product or service" [8]. Considering the large audience that spam messages can reach, 16 % could generate a lot of revenue for these companies. This revenue comes at the cost to user's and ISP's (Internet Service Provider) because these messages take up bandwidth and local memory. They also may be pornographic and inappropriate for young email users. It is obvious that techniques to block or filter spam messages are essential. One such technique is text classification. Text classification works because the content of most spam messages has similar language. A classifier can then be trained to detect which words are likely to appear in spam messages, and which words are not likely to appear in spam messages. The classifier that I implemented is the Naive Bayes classifier. Naive Bayes uses a "bag of words" approach so it treats each word independently [7]. It then computes the probability that each word appears in a spam or ham (non-spam) message. These probabilities are combined to produce a probability of spam for the entire email, and the probability of ham for the entire email. The probability that is the highest is what the email is classified as. My Naive Bayes classifier is explained in the following sections.

3 Background

One of the first attempts to create a Naive Bayes spam filter was Sahami [9]. In his paper published in 1998, he outlined how he and his colleagues successfully created a Naive Bayes spam filter to detect spam. To my knowledge, this was the earliest successful attempt at a Naive Bayes classifier. This paper also went beyond just a simple Naive Bayes classifier. Sahami included several other variables that were used in the calculation to accurately filter email. He called these variables "domain specific" properties. For instance, Sahami's classifier looked for around 20 specific phrases, such as "FREE!" or "be over 21" that clearly indicated spam. He also looked at the sender of the email, and whether it was sent to an

individual address or many addresses. Spam email is not typically sent from a .edu domain, addressed to an individual person. Sahami and his colleagues used these domain specific properties in their calculation to enhance their Naive Bayes classifier. This implementation decision worked, as they received the best results when computing probabilities on words, phrases, and domain specific properties. I do not include any domain specific features in my calculation, although that would be an area for future work. Another attempt to use text classification to filter spam was Paul Graham [5]. He outlined what he did in his online article, "A plan for spam". Graham did many things that I ended up doing in my solution. For instance, Graham first parsed the email into "tokens" and calculated the spam probability for each token. To calculate the probability, he built hash tables on his training set to count the number of times a word appears in spam emails. I also parsed my data this way, and built similar hash tables. He recognized the problem when a word does not appear in the training set, resulting in a probability of 0. Graham decided to assign a probability of .4 to these words. Unlike Graham, I solved this problem with Laplace smoothing.
Graham also comments on other ways in combination to Naive Bayes to help filter email. One way is each user has a list of words that automatically tells the filter that an email is legitimate. The list would be specific to each user, because everybody has words that they use more often than others. Graham is extremely optimistic that Naive Bayes is the "single most effective" way to filter out spam. This is because as spam evolves over time, the Naive Bayes classifier can also. Training sets can be constantly added to, so if the spammers try to change the language of the emails, the words will simply be added to the training set.

4 Problem Approach

My approach to this problem was to create a Naive Bayes filter from scratch. I did this because learning is the purpose of this project, and I wanted to achieve accurate results.

The Naive Bayes formula is shown below.

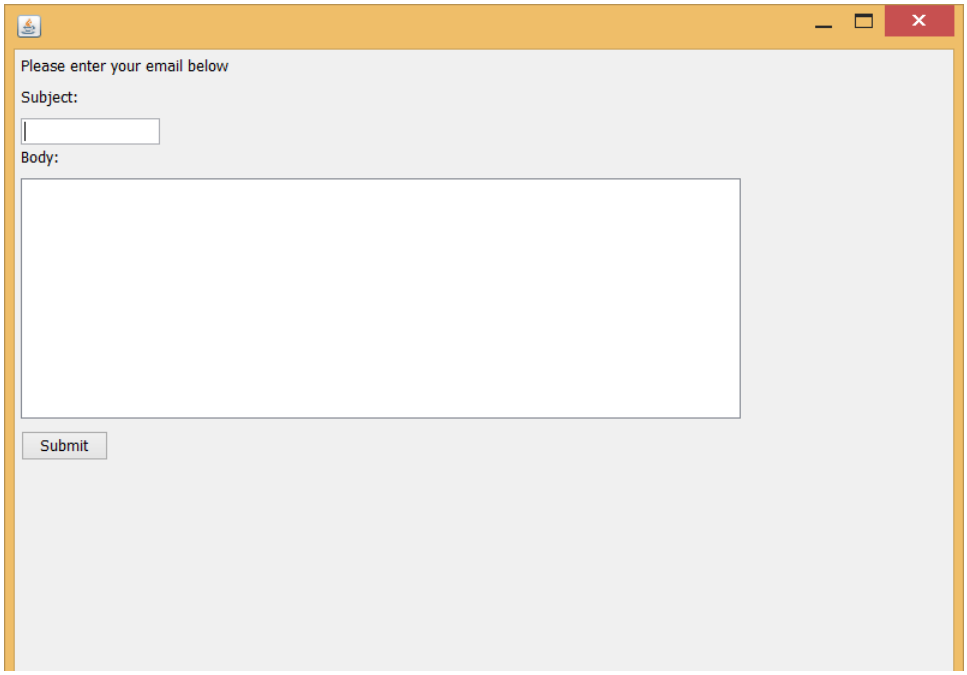
$$Pr(A|B) = \frac{Pr(B|A) * Pr(A)}{Pr(B|A) * Pr(A) + Pr(B|\neg A) * Pr(\neg A)} \tag{1}$$

This formula can be used to calculate the probability of a certain class, given that it belongs to another class. As I said above, Naive Bayes uses a "bag of words" approach, meaning each word is independent of every other word. This is generally an invalid assumption, as each word has a high likelyhood of depending on a previous word. This is an important detail, although Naives Bayes ignores it in implementation. In the context of spam filtering, $Pr(A|B)$ is the probability a message is spam given it contains a certain word. The $Pr(B|A)$ is the probability the certain word appears in spam messages. $Pr(A)$ is the probability that any message is spam, and $Pr(\neg A)$ is the probability any message is legitimate. Finally, $Pr(B|\neg A)$ is the probability the certain word appears in legitimate messages. To use this formula, the values on the right side of the equation need to be computed using a training set of emails. A training set is a large set of previously classified emails that the classifier uses to make decisions. To do this, I searched the Internet for a free to use training set of previously classified emails. I found many training sets but I eventually decided to use the Ling-spam data set [1] as it was used in "An Evaluation of Naive Bayesian Anti-Spam Filtering" [4]. I also used it because it came in four different kinds: bare, lemm, lemm-stop, and stop. Each type is the same data, but different parsing was done to the data. The bare data set is exactly how the original email was sent in text form. The lemm data set put each word in each email through a "lemmatizer" which converts each word to its base form. For example, the word "run" can be represented in forms such as "ran" or "running". The lemmatizer will convert these two words back to its base "run". This is helpful when calculating probabilities because two words with essentially the same meaning like "run" and "running", will be treated as one word. The third data set used the lemmatizer and

a stop-list. A stop list is a list of words that are removed from the email because they do not provide any information (e.g "the", "a", "on"). The list of words that were filtered out in this data set was not provided. Finally the last data set only used a stop-list, and no lemmatizer. I used the bare data set and the lemm-stop data set to compare the combined effects of the lemmatizer and the stop-list.

5 Problem Design

After I found my data set, I decided to upload it to a MySQL database. I choose to do this because of the ability to switch training sets easily by just specifying the table to choose from. I created three different MySQL tables that each represented a training set. The first table, had the bare Ling-Spam data. The second table had the lemm-stop Ling-Spam data. The third table was a subset of the lemm-stop Ling-Spam data. The third table contained 50 % spam messages, and 50 % legitimate messages. I did this because changing the distribution of messages changes how "suspicious" the classifier is. For instance, if the training set has 99 % spam messages, the classifier is very likely to classify emails as spam. My first two training sets had 16.6 % spam emails, and the third had 50 % spam emails. The reason the first two training sets had 16.6 % spam messages is because it is generally considered worse to lose a legitimate message due to a spam filter, than to successfully filter all spam messages [5]. So my first two training sets are a lot less suspicious of spam compared to my third training set. Each training set's results are analyzed in the analysis section. My classifier code and application code were both written in java. My application's interface is shown in the figure below.



The user simply inputs the subject and body of the email and it is classified as spam using one of the three training sets, and my Naive Bayes classifier code.

While the application code is straight forward, the Naive Bayes code was more complicated. I wrote two methods within the Naive Bayes class. One method downloaded the training set into memory, and computed some constants. For example, it computed $Pr(A)$ and $Pr(\neg A)$, which for training sets one and two is 16.6 % and 83.4 % respectively. After this finishes the second method computes probability of spam for the subject and body independently. I calculated each probability independently because the subject and body are not the same thing. In fact, the subject represents the body so it should contain similar words that the body contains. The two probabilities are combined at the end to produce one probability for the whole message.

Computing the probability for the subject or body was not as straight forward as Equation 1 above. This is for two reasons. First, $Pr(B|A)$ needs to be computed for every word. As I said above, $Pr(B|A)$ is the probability that a given word is in a spam message. In many cases a word may only appear two or three times. This leads to a very small probability. The

Naive Bayes formula does this for each word, and then combines all of them at the end. If each word has a probability close to zero, combining them can lead to arithmetic underflow. To avoid this, I used logarithms. This was a handy mathematical trick to combine small probabilities [2]. The second reason this was more complicated was that in some rare cases the $Pr(B|A)$ is 0. This would be the case that the word never appears in spam messages in the training set. This led to an error because $log(Pr(B|A))$ when $Pr(B|A)$ is 0 results in an error. To avoid this, I used Laplace smoothing. This ensured that the number of times the word appears in spam messages is not zero. All Laplace smoothing does is add 1 to the count of the word, and it does not interfere with the probability calculation. Lastly, I figured out quickly that the denominator of equation 1 does not need to be computed. This is because it is a constant and neglecting it does not interfere with the computation. So after these three changes, the formula to compute the probability that a word appears in a spam email is:

$$log(SpamWordCount + 1/totalSpamWords) + log(Pr(A)) \tag{2}$$

SpamWordCount is the number of times that word appears in spam messages in the training set, and totalSpamWords is the total number of words in spam messages in the training set. One can see that $log(SpamWordCount + 1/totalSpamWords) = log(Pr(B|A))$. This formula is used to calculate logrithmic probabilities for every word. Finally, after each probability is calculated, they are summed up to compute the logrithmic probability that the subject or body is spam. This is repeated a second time, but for legitimate messages. When all of this is done each sum is compared. Whichever sum is higher is the more probable class, so that is the class the email is assigned to.

6 Analysis

Naive Bayes classifiers are typically evaluated by two measures, precision and recall. Precision is based on the number of false positives, while recall is based on the number of false negatives [6]. In the context of spam filtering, a false positive is a email that was classified as spam but is actually legitimate. For spam filters, precision is the most important metric because people do not want to lose their legitimate email due to a spam filter [3]. False negatives are less important because those are just the emails that make it through the spam filter. A low recall percentage does not have the same consequences as a low precision percentage. Because of this fact, I wanted to get precision as close as possible to 100 %, while maximizing recall.

Precision and recall were computed the following way. The training set was split into ten parts. Each part had and equal distribution of spam messages, and legitimate messages. Then, starting at part 1 and looping to part 10, each part's emails are classified using the other nine parts as the training set. The number of false positives and false negatives are counted, and precision and recall is calculated for each part. The precision and recall forumlas are shown below.

$$Precision = \frac{numCorrectLegitimate}{numCorrectLegitimate + numFalsePositives} \tag{3}$$

$$Recall = \frac{numCorrectLegitimate}{numCorrectLegitimate + numFalseNegatives} \tag{4}$$

NumCorrectLegitimate is the number of emails correctly classified as ham, numFalsePositives is the number of false positives, and finally numFalseNegatives is the number of false negatives. At the end, precision and recall are averaged over each part to end up with a precision and recall percentage for the entire training set. I did this for all of my three training sets, and my results can be seen in the table below.

	trainingSet1	**trainingSet2**	**trainingSet3**
Precision	1.0	1.0	.9896
Recall	.845	.8966	.9838

trainingSet1 = Bare Ling-Spam Data

trainingSet2 = Lemm-Stop Ling-Spam Data

trainingSet3 = Lemm-Stop Ling-Spam Data with 50 % spam messages, 50 % ham messages

As you can see, each training set has precision values that equal 1.0 or close to 1.0. The reason training sets 1 and 2 both have a precision value of 1.0 is because of the distribution of messages in the training set. They both have 16.6 % spam messages, so it is harder to classify a given message as spam. Training set 3 has a lower precision value because of the 50 % spam messages in the training set. Unlike precision, the recall values for each training set differed. There was about a 5 % increase between training set 1 and 2, and about a 10 % increase between 2 and 3. The percentage difference between one and two was because of the lemmatizer, and the stop-list. It is obvious from these results that the lemmatization and stop-lists help decrease false negatives, with no effect to false positives. The percentage difference between training sets 2 and 3 was because of the distribution of messages again. Since it is easier to classify a message as spam, it makes sense that the recall percentage goes up, although this came at the cost of a decrease in precision. Overall, I was extremely satisfied with my results. Each training set got values that reflect the parsing of the data and the distribution of messages. In practice, training set 2 would most likely be used since it had a precision value of 1.0, and the highest recall value.

7 Conclusion

My results suggest that a Naive Bayes spam filter can block large amounts of spam email. However, a Naive Bayes spam filter is not necessarily the solution for all spam. For instance, my Naive Bayes filter cannot detect HTML, images, or other attachments to emails. Most spam isn't in just text form, so this would be a needed improvement in the future. Another improvement could be to weight the probability for the subject and body differently. In my classifier, I averaged the subject and body probabilities so they are each weighted evenly. A more complete solution might weight the subject more because it represents the body. My filter was also based on only one training set. To further test how well my Naive Bayes filter performs, more training sets would have to be used to see if similar results are achieved.

References

[1] Csmining group. `http://csmining.org/index.php/ling-spam-datasets.html`. Accessed: 2014-11-17.

[2] Naive bayes for classifying text. `http://www.cs.nyu.edu/faculty/davise/ai/bayesText.html`. Accessed: 2014-12-05.

[3] Ion Androutsopoulos, John Koutsias, Konstantinos V. Cb, and Constantine D. Spyropoulos. An experimental comparison of naive bayesian and keyword-based anti-spam filtering with personal e-mail messages. In *In Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 160–167. ACM Press, 2000.

[4] Ion Androutsopoulos, John Koutsias, Konstantinos Chandrinos, Georgios Paliouras, and Constantine D. Spyropoulos. An evaluation of naive bayesian anti-spam filtering. *CoRR*, cs.CL/0006013, 2000.

[5] Paul Graham. A plan for spam. Available on: http://paulgraham.com/spam.html, August 2003.

[6] Jacob Perkins. Text classification for sentiment analysis precision and recall. `http://streamhacker.com/2010/05/17/text-classification-sentiment-analysis-precision-recall/`. Accessed: 2014-12-05.

[7] Jason D. M. Rennie, Lawrence Shih, Jaime Teevan, and David R. Karger. Tackling the poor assumptions of naive bayes text classifiers. In *In Proceedings of the Twentieth International Conference on Machine Learning*, pages 616–623, 2003.

[8] Insights World Reseach. A look at consumers' awareness of email security and practices or "of course, i never reply to spam except sometimes". *MAAWG*.

[9] Mehran Sahami, Susan Dumais, David Heckerman, and Eric Horvitz. A bayesian approach to filtering junk e-mail, 1998.