Naive Bayes Spam Filtering

by Jamin Thornsberry

Abstract

Spam filtering has been an issue ever since email has come into popular use. Many methods have been devised to combat this issue, but most have been circumvented by spammers. In this paper, I discuss the early methods of spam filtering, as well as the method of naive Bayes spam filtering, which has largely resisted the offensive efforts of spammers since its introduction. I present my implementation of a naive Bayes spam filter and the lessons learned from implementing this filter.

1. Introduction

Since the emergence of email in the early 1990s, spam mail, or unsolicited bulk email, has been a major problem. The introduction of spam filters has helped to aid this problem by classifying emails coming into a user's inbox as spam or ham (non-spam). Over the years, several techniques for filtering spam have been devised. [1] Some of the most popular techniques are described below:

List-based spam filtering:

This filtering method involves filtering email based on who sent the message. One method is to use a blacklist of all senders that are considered spam, and then categorize every message from these senders as spam. Another method is to use a whitelist of senders that are considered trusted, and only allow emails from senders on this whitelist, labelling the rest as spam.

Rule-based spam filtering:

This filtering method uses a set of rules to evaluate between an email being spam or ham. For example, one rule that the spam filtering engine might have is that any message containing the word "nigeria" is labelled as spam. Some rule-based systems might use a point system, assigning certain point scores to the satisfaction of certain rules. The final score can then be evaluated against some score threshold to determine whether the message is spam or ham.

Challenge-response spam filtering:

This simple method of spam filtering takes into account the fact that spam is sent out in bulk, usually through some sort of automation. In the challenge-response spam system, whenever a message is received from a sender that has not been pre-approved, a reply is sent back in which the recipient must perform some sort of task to have their message approved. Since spam is sent out many thousands of messages at a time, it would be extremely impractical for the spammer to perform all of these challenges.

2. Disadvantages of popular techniques

Although the methods listed above work in many circumstances, and can often times filter out a large majority of spam emails, they all have their shortcomings. For example, filtering engines that use whitelists are very restrictive in that new trusted senders usually must be added manually. Filters that use blacklists must continually be updated, as spammers can gain access to new sources to send from as fast as those sources of spam are added to the blacklist.

With a rule-based system, if a spammer can find the list of rules that a spam engine uses to filters emails, they can simply design their spam message to avoid all of these rules. Challenge-response spam systems can make it laborious for other users to send emails to someone with the system, and emails that are sent by automated systems, but that are not considered spam (like a mailing list or store receipt) will be erroneously blocked.

3. Naive Bayes Spam Filtering

A method of filtering spam does exist that combats most of the problems that other spam filtering techniques encounter: filtering using a naive bayesian probabilistic approach. This method, first devised by a research group at Microsoft, uses Bayes' theorem to determine the probability that messages are spam given the presence of words. [2] The following steps describe the method of naive bayes spam filtering:

1. Two sets of messages are supplied to the spam system, one with messages that are considered spam and the other with messages that are considered ham.
2. In both of the sets, each message is reduced to a "bag of words" representation, where the individual words are extracted from the message, ignoring any syntactical structure or order of the original message.
3. For all of the words present over the two sets of messages, each word's spamicity is calculated. A word's spamicity is the probability that a message containing that word should be considered spam. The spamicity is computed according Bayes' theorem, with the specialized equation shown here:

$$P(S|W) = \frac{P(W|S) \cdot P(S)}{P(W|S) \cdot P(S) + P(W|H) \cdot P(H)}$$

where:

$P(S|W)$ is the probability a message containing the given word is spam
$P(W|S)$ is the probability that a spam message contains the given word
$P(W|H)$ is the probability that a ham message contains the given word
$P(S)$ is the initial probability that a message is spam
$P(H)$ is the initial probability that a message is ham

Both $P(W|S)$ and $P(W|H)$ are estimated by taking the number of messages from each set (spam or ham) in which the given word is present out of the total number of messages in the set. $P(S)$ and $P(H)$ are predetermined, and are usually both set at 0.5, but can be tweaked as needed to represent the real-word proportion of spam to ham messages that get sent to a user. When a word is present in only the spam or ham set of messages, the

word's spamicity can be set to some arbitrary but symmetrically high or low spamicity, respectively (like 0.99 and 0.01).

4. Once all of the spamicities are calculated, the spam filter has been trained and can begin filtering new messages. For each incoming message, Bayes' theorem is used again, but this time using the product of the spamicities of all the words present in the incoming message. The probability that a message is spam is calculated using the following specialized equation:

$$p = \frac{p_1 \cdot p_2 \ldots p_N}{p_1 \cdot p_2 \ldots p_N + (1 - p_1) \cdot (1 - p_2) \ldots (1 - p_N)}$$

where:

p is the probability that a message is spam

p_N is the probability that a message is spam given some Nth word

Once p is computed, it can be judged against some threshold. If p is greater than this threshold, the message is considered spam. If p is less than this threshold, the message is considered ham.

4. Disadvantages of Naive Bayes

Despite the clear benefits of filtering spam using naive bayesian probabilities, there are still disadvantages. One attack that filters using naive bayes probability might be susceptible to is a method called bayesian poisoning. This method involves spammers injecting non-spammy words into their spam messages in order to trick the filter into classifying them as ham. This method has shown to be quite effective in some filters. [3]

Another method that spammers use to circumvent spam detection is by putting the text of their email into an image, which bayesian spam filters typically ignore. This can be combated by using optical character recognition in order to extract text from images. [4]

5. My Implementation

I have implemented a naive bayes spam filtering engine in Python that performs each of the steps mentioned above. The program takes 4 folder paths as arguments, assuming that the files within each folder representing the following: spam example messages, ham example messages, spam test messages, ham test messages. The test message folders are divided into spam and ham so that the success rates can be determined by evaluated whether each classification is a correct one.

The program also takes 4 other optional arguments that control optimizations to the basic naive bayes filtering method in order to improve results. The arguments are listed as follows:

- **Initial spam probability** (default: 0.5) - the initial probability that a message is spam. The initial probability that a message is ham is determined as the inverse of this probability (1 - initial probability message is spam).

- **Occurrence threshold** (default: 10) - the number of times a word must have appeared overall during the training stage to be considered when evaluating whether a test message

is spam. During the reduction of each message to a "bag of words" (mentioned above in step 2), the multiplicities of each word are kept, and then later considered when evaluating against the occurrence threshold.

- **Score threshold** (default: 0.9) - the score (mentioned above in step 4) over which an incoming message must score to be considered spam.

- **Phrase length** (default: 1) - the maximum number of adjacent words to consider as phrases. By default, during the stage where each message is reduced to a bag of words, what constitutes a word is determined by a regular expression that considers a series of alphabetic characters as a word (taking into account contractions and compound words). With a phrase length greater than 1, up to phrase length adjacent words are considered as "phrases" that are added to the bag of words as their own unique entries. Increasing the phrase length can drastically reduce performance of the program, as it adds many new words to be considered when calculating spamicities.

To test my implementation, I used Apache SpamAssassin's public corpus of spam and ham emails. [5] I took 200 emails from each of the sets (spam and ham), and put them in the test folders. The initial results were encouraging, with around around an 88% spam success rate and a 100% ham success rate using the default parameters. This means that 88% percent of spam was labelled correctly and 100% of ham was labelled correctly. With some tweaking of the optional arguments (increased spam probability, lowered occurrence threshold, increased phrase length), I was able to achieve success rates of about 99% for both ham and spam, which approaches the upper limits of what a spam filter can do, given that there will always be some anomalies.

6. Lessons Learned

During the implementation of my naive bayes spam filter, I tried to do two things to "clean up" the messages before I reduced them to a bag of words. First, I converted all words to lowercase before I put them into the bag of words, thinking that combining words that I considered the same would improve my results. Second, I removed the message headers from all of the messages, assuming they would be garbage and harm my results if I left them. Both of these assumptions turned out to be incorrect. It turns out, leaving the messages completely intact (keeping original capitalization and leaving email headers in the messages) actually improved results, and my naive assumptions about having to "clean up" the data were unfounded, as I was simply removing helpful data. I later found that others had made the exact same mistakes in their implementations, like Paul Graham mentions in his essay "Better Bayesian Filtering. [6]

7. References

[1] B. Satterfield, 'Ten Spam-Filtering Methods Explained', Techsoupcanada.ca, 2006. [Online]. Available: http://www.techsoupcanada.ca/learning_center/10_sfm_explained.

[2] M. Sahami, S. Dumais, D. Heckerman and E. Horvitz, 'A Bayesian Approach to Filtering Junk E-mail', 1998.

[3] H. Stern, J. Mason and M. Shepherd, 'A Linguistics-Based Attack on Personalised Statistical E-mail Classifiers', 2004.

[4] Google.com, 'About Gmail – Google'. [Online]. Available: http://www.google.com/mail/help/intl/en_GB/fightspam/spamexplained.html.

[5] Spamassassin.apache.org, 'Index of /publiccorpus'. [Online]. Available: https://spamassassin.apache.org/publiccorpus/.

[6] P. Graham, 'Better Bayesian Filtering', Paulgraham.com, 2003. [Online]. Available: http://www.paulgraham.com/better.html.