SpamCop: A Spam Classification & Organization Program

Patrick Pantel and Dekang Lin
Department of Computer Science
University of Manitoba
Winnipeg, Manitoba
Canada R3T 2N2

March 11, 1998

Abstract

We present a simple, yet highly accurate, spam filtering program, called SpamCop, which is able to identify about 92% of the spams while misclassifying only about 1.16% of the nonspam e-mails. SpamCop treats an e-mail message as a multiset of words and employs a naive Bayes algorithm to determine whether or not a message is likely to be a spam. Compared with keyword-spotting rules, the probabilistic approach taken in SpamCop not only offers high accuracy, but also overcomes the brittleness suffered by the keyword spotting approach.

1. Introduction

With the explosive growth of the Internet, so comes the proliferation of spams. Spammers collect a plethora of e-mail addresses without the consent of the owners of these addresses. Then, unsolicited advertising or even offensive messages are sent to them in mass-mailings. As a result, many individuals suffer from mailboxes flooded with spams. Many e-mail packages contain mechanisms that attempt to filter out spams by comparing the sender address of the e-mails to predefined lists of known spammers. Such programs have had limited success since spammers often change their address and new spammers continuously appear. Furthermore, spammers have found ways to send messages with forged headers. For example, the sender address can be made the same as the receiver address. A more general and effective approach is obviously needed.

In [2], Cohen presented an approach to e-mail classification in which a learning program, called RIPPER [1, 3], is used to obtain a set of keyword-spotting rules. If all the keywords in a rule are found in a message, the conclusion in the rule is drawn. For example, RIPPER created the following set of rules to recognize talk announcements:

A message is a talk announcement if it contains:

- 'talk' and 'talk' in 'Subject:' field; or
- '2d416' and 'the'; or
- 'applications' and 'comma' in 'Subject:' field; or
- 'visual'; or
- 'design' and 'transfer' or
- 'place' and 'colon' and 'comma' in 'To:' field; or
- 'doug' in 'From:' field and 'specification'; or
- 'presentation'

Otherwise, the message is not a talk announcement.

Cohen reported that the rules generated by RIPPER have similar accuracy as manually generated rules.

Spams form a semantically much broader class than the categories in Cohen's experiments. Their subjects often range from advertising products to "make money fast" schemes to WEB site for "fun-loving adults". The keyword-spotting rules appear to be too brittle for this purpose. The inadequacy of this method for spam filtering is also evidenced by the fact that even experienced computational linguists are not able to come up with a good set of keyword combinations for this purpose.

In this paper, we presents a spam-filtering program, called SpamCop, which employs a naive Bayesian algorithm to detect spams. The remainder of this paper is organized as follows. The next section describes the SpamCop program. The experimental results and the comparison with RIPPER are presented in Section 3.

2. Description of SpamCop

SpamCop uses a naive Bayesian algorithm to classify messages as regular spam or as nonspam. A message M is classified as a spam if $P(Spam|M)$ is greater than $P(NonSpam|M)$. In most probabilistic approaches to text classification, the attributes of a message are defined as the set or the multiset of words in the message. However, this is not the only viable alternative. For instance, one can also define all the three consecutive letter sequences (trigrams) as the attributes. Once M is represented as a set of attributes $(a_1, \ldots, a_n)$, the classification problem becomes that of finding the larger one of $P(Spam|a_1, ..., a_n)$ and $P(NonSpam|a_1, ..., a_n)$. Since

$$P(Spam|a_1, ..., a_n) = \frac{P(Spam, a_1, ..., a_n)}{P(a_1, ..., a_n)}$$

$$P(NonSpam|a_1, ..., a_n) = \frac{P(NonSpam, a_1, ..., a_n)}{P(a_1, ..., a_n)}$$

the problem becomes determining which is the larger one between $P(Spam, a_1, ..., a_n)$ and $P(NonSpam, a_1, ..., a_n)$, which can be rewritten as:

$$P(Spam, a_1, ..., a_n) = P(a_1, ..., a_n|Spam)P(Spam)$$

$$P(NonSpam, a_1, ..., a_n) = P(a_1, ..., a_n|NonSpam)P(NonSpam)$$

If we further assume that the attributes in a message are conditionally independent given the class of the message, the right hand side of the above equations become:

$$P(Spam, a_1, ..., a_n) = P(a_1|Spam)...P(a_n|Spam)P(Spam)$$

$$P(NonSpam, a_1, ..., a_n) = P(a_1|NonSpam)...P(a_n|NonSpam)P(NonSpam)$$

We now describe how to estimate the probabilities $P(a_i|Spam)$, $P(a_i|NonSpam)$, $P(Spam)$, $P(NonSpam)$. Once these probabilities become available, the above formulas will allow us to determine which class has the higher conditional probability.

We first tokenize the message. A token is either a consecutive sequence of letters or digits, or a consecutive sequence of non-space, non-letter and non-digit characters (we limit the length of the second kind of token to be at most three characters long). The spaces are ignored. We then remove the suffixes from the tokens using an implementation of the Porter stemmer [5] by Frakes and Cox. The frequency counts of the suffix-removed tokens are then accumulated in a frequency count table. For each word W in the training messages, the frequency table contains the count $N(W, Spam)$, and $N(W, NonSpam)$, which is the number of times the word W occurred in the documents that belong to class C. The frequency table also records the total number of words (not necessarily unique) in the spam and nonspam messages: $N(Spam)$ and $N(NonSpam)$. Table 1 illustrates a subset of the frequency table.

Once the frequency table is created, we use the m-estimate method [4] to estimate the conditional and prior probabilities of the words. M-estimate can be viewed as mixing the sample population in the frequency table with m uniformly distributed virtual examples. In our experiments, we used m=1 and the probability of a word in the virtual example is $\frac{1}{K}$ where K is the number of unique words in the training messages. In other words,

$$P(W|C) = \frac{N(W, C) + \frac{1}{K}}{N(C) + 1}$$

where C is $Spam$ or $NonSpam$ and W ranges over the set words in the training messages.

Some words are not good indicators of the classification of the message in which they occur. We applied a filter to remove such words from the frequency table so that the classification of a message will not be affected by the accumulation of noise. A word W is removed from the frequency table if one of the following conditions are met:

Table 1: Sample entries from the frequency table

word	spam	nonspam	word	spam	nonspam
`___`	2183	703	`report`	215	64
`|||`	60	0	`mail`	358	167
`$`	716	295	`ship`	36	0
`adult`	52	0	`:__`	36	0
`000`	178	26	`you`	1165	1210
`million`	69	2	`///`	251	103
`order`	253	60	`email`	212	77
`###`	44	0	`address`	239	99
`bulk`	43	0	`your`	581	458
`monei`	127	19	`busi`	122	30

- $N(W, Spam) + N(W, NonSpam) < 4$; or
- $\frac{P(W|Spam)}{P(W|Spam)+P(W|NonSpam)} \in [0.45, 0.55]$.

3. Experimental Results

3.1. Setup

Our training data consists of 160 spams that were sent to one of the authors (DL) and 466 nonspam messages in DL's mailbox. The testing messages consist of 277 spams obtained from the Internet[1] and 346 NonSpam e-mails in DL's mailbox from a different (but adjacent) time period. The header information is removed from the messages. The classification is completely based on the body of the messages.

There are a total of 230449 words in the training messages with 60434 in spams and 170015 in nonspams. There are 12228 entries in the frequency table. Applying the filtering rules from the previous section reduces the number of entries to 3848.

3.2. Evaluation Measures

Let

- $TrueCount(Spam)$ and $TrueCount(NonSpam)$ denote the number of spam and nonspam messages in the testing data.

[1] `http://pantheon.cis.yale.edu/ jgfoot/junk.html`

- $CorCount(Spam)$ and $CorCount(NonSpam)$ denote the number of messages that are correctly classified as Spam and NonSpam by SpamCop.

We use three measures to evaluate the performance of SpamCop: false positive rate R_{fp}, false negative rate R_{fn}, and error rate R_e:

$$R_{fp} = 1 - \frac{CorCount(NonSpam)}{TrueCount(NonSpam)}$$

$$R_{fn} = 1 - \frac{CorCount(Spam)}{TrueCount(Spam)}$$

$$R_e = 1 - \frac{CorCount(Spam) + CorCount(NonSpam)}{TrueCount(Spam) + TrueCount(NonSpam)}$$

The false positive rate is the percentage of nonspam messages that are incorrectly classified as spam. It measures how safe the filter is. The false negative rate is the percentage of spam messages that pass through the filter as nonspams. It measures how effective the filter is. The error rate measures the overall performance.

3.3. Results

Table 2 summarizes our results. It can be seen that although naive Bayes algorithm is extremely simple, it achieved very high accuracy, especially with respect to the nonspam messages. The filtering algorithm reduced the frequency table to 1/3 of its original size and resulted in a slightly higher false positive rate, a much lower false negative rate and a lower overall error rate.

Table 2: Testing results with 277 spams and 346 nonspams

Filter	R_{fp}	R_{fn}	R_e
y	1.16%	8.30%	4.33%
n	0.58%	13.36%	6.26%

We also investigated the effects of the size of the training data on the performance of SpamCop. We divided the training data into 5 even partitions. Each partition has the same spam/nonspam ratio as the whole set. The results are presented in Table 3. The first column is the data size in terms of the number of partitions. For each data size we randomly selected 5 configurations. The average rates of the 5 configurations are shown in last three columns in Table 3. The second column indicates whether the frequency table is filtered or not.

Table 3: Effects of the number of training examples

size	Filter	R_{fp}	R_{fn}	R_e
1/5	y	1.68%	12.35%	6.42%
1/5	n	1.33%	14.08%	7.00%
2/5	y	1.68%	10.97%	5.81%
2/5	n	1.10%	12.64%	6.23%
3/5	y	1.21%	8.66%	4.53%
3/5	n	0.92%	11.84%	5.78%
4/5	y	1.01%	8.94%	4.53%
4/5	n	0.79%	11.64%	5.62%

It can be seen that SpamCop achieves good performance with as few as 32 spam messages and 91 nonspam messages as training examples. Filtering the frequency table consistently produced the same effect: slight increase of false positives, a decrease of false negatives, and a moderate decrease of the error rate.

We also experimented with varying ratios between the number of spam and nonspam messages. The first two columns in Table 4 are the number of spams and nonspams used in training. Compared with the results in Table 2, it appears that a higher ratio of training examples in a category increases the performance in that category. However, it significantly decreases the performance of the other category.

Table 4: Effects of varying ratios of spam and nonspams

spams	nonspams	R_{fp}	R_{fn}	R_e
32	466	0.06%	53.07%	23.63%
160	91	12.60%	1.44%	7.64%

3.4. Using trigrams

Instead of suffix-stripped words, we also used trigrams extracted from words as features. A trigram in a word is a consecutive sequence of three letters in the word. Table 5 illustrates the results of the use of trigram in spam-filtering, using the same training and testing data as the experiment describe in Table 2. Considering the amount information that gets lost during the reduction from words to trigrams, the numbers in Table 5 are remarkably close to the values in Table 2. This might be attributed to the fact that since there are much fewer unique trigram than unique words, the probability estimation for trigrams are more accurate.

Table 5: SpamCop performance using trigrams

Filter	R_{fp}	R_{fn}	R_e
y	4.91%	6.50%	5.62%
n	2.89%	9.03%	5.61%

3.5. Comparison with RIPPER

RIPPER is a rare symbolic learning program that is able to deal with texts. We ran RIPPER with the same training and testing data. RIPPER generated 19 rules with 50 conditions and achieved an error rate of 13.64% on the 623 testing messages.

The top ranked rule in RIPPER is that "if the dollar sign and the word 'only' occurs in a message then it is classified as spam." This rule correctly classified 41 spams and misclassified 2 out of 466 nonspams. Although this rule performed very well, it will misclassify long nonspams which happen to contain these two words. In contrast, our probabilistic algorithm is much more robust.

Another example that demonstrates the advantage of a probabilistic classification over a rule-based classification is the word "you". The word "you" has one of the highest ratio between its conditional probability in spam and nonspam messages. In an extreme case, one of the spams in the training example contained 99 occurrences of "you" or "your" in 112 lines. Therefore, a high frequency of "you" is a definitely good indicator of spams. However, "you" is also a common word in nonspams. A keyword-spotting rule will not be able to use this in classification.

4. Conclusion

We presented a simple, yet highly accurate, spam-filtering program, called SpamCop. It treats an e-mail message as a multiset of words and employs a naive Bayes algorithm to determine whether or not a message is likely to be a spam. Our experiments show that SpamCop is able to identify about 92% of the spams while misclassifying only about 1.16% of the nonspam e-mails. Our experiments also show that high classification accuracy can be achieved with as few as 32 spam examples. Compared with symbolic learning programs such as RIPPER, SpamCop produced higher accuracy and does not suffer from the brittleness associated with keyword-spotting rules.

References

[1] William W. Cohen. Fast effective rule induction. In *Machine Learing: Proceedings of the Twelfth International Conference*, Lake Taho, California, 1995. Morgan Kaufmann.

[2] William W. Cohen. Learning rules that classify e-mails. In *AAAI Spring Symposium on Machine Learning for Information Access*. AAAI, 1996.

[3] William W. Cohen. Learning with set-valued features. In *Proceedings of AAAI-96*, 1996.

[4] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[5] M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, July 1980.