Information Systems Security

Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/uiss19

Filtering Spam at E-Mail Server Level with Improved CRM114

Víctor Méndez a, Julio Cesar Hernández a, Jesús Carretero a & Félix García a

a Department of Computer Science, Universidad Carlos III de Madrid, Madrid, Spain

They may be reached at vmendez@inf.uc3m.es, jcesar@inf.uc3m.es, jcarrete@inf.uc3m.es, and fgcarbal@inf.uc3m.es, respectively

Published online: 21 Dec 2006.


To link to this article: http://dx.doi.org/10.1201/1086/44530.13.3.20040701/83069.7

PLEASE SCROLL DOWN FOR ARTICLE

Taylor & Francis makes every effort to ensure the accuracy of all the information (the “Content”) contained in the publications on our platform. However, Taylor & Francis, our agents, and our licensors make no representations or warranties whatsoever as to the accuracy, completeness, or suitability for any purpose of the Content. Any opinions and views expressed in this publication are the opinions and views of the authors, and are not the views of or endorsed by Taylor & Francis. The accuracy of the Content should not be relied upon and should be independently verified with primary sources of information. Taylor and Francis shall not be liable for any losses, actions, claims, proceedings, demands, costs, expenses, damages, and other liabilities whatsoever or howsoever caused arising directly or indirectly in connection with, in relation to or arising out of the use of the Content.

This article may be used for research, teaching, and private study purposes. Any substantial or systematic reproduction, redistribution, reselling, loan, sub-licensing, systematic supply, or distribution in any form to anyone is expressly forbidden. Terms & Conditions of access and use can be found at http://www.tandfonline.com/page/terms-and-conditions
Filtering Spam at E-Mail Server Level with Improved CRM114

Víctor Méndez, Julio Cesar Hernández, Jesús Carretero, and Félix García

This paper explores the possibility of advanced techniques at the server level, and concludes that simple improvements to a well-known Naive Bayes-based technique could turn this algorithm into a much faster and significantly better one that could be used at the server level.

The problem of automatically filtering unwanted e-mail messages is one of increasing importance, because bulk mailers take advantage of the great popularity of the electronic mail communication channel for indiscriminately flooding e-mail accounts with unwanted advertisements. There are many factors that contribute to the proliferation of unsolicited spam, especially the inexpensive cost of sending e-mail, and of obtaining pseudonyms. On the other hand, we have the high cost associated with users receiving spam and the network overflow.

The spam filtering problem can be seen as a particular instance of a text categorization problem where the classes are spam or ham. In recent years, a vast amount of techniques has been applied to solve this problem. Some of the top-performing methods are Learning Rules that classify e-mail based on the RIPPER algorithm, Ensembles of Decision Trees (1999), Support Vector Machines (1998), Boosting (2000), and Instance Based Learning (2000). Nowadays, advanced Naive–Bayes methods are the top-performing ones, coming from Paul Graham’s principles for spam-classifying, and some basic improvements (2003). The false positive rates go from 0.3 to 0.06 percent, and the detected spam from 99.5 to 99.75 percent.

In this article we explain the principles that make CRM114 one of the applications with the best filtering accuracy. We then compare its design features and accuracy with other state-of-the-art applications. After that, we describe our proposal of modifying CRM114, notably its window size, and empirically compare accuracy versus speed and text features extraction; finally we introduce the new concept of virtual feature, prove its usefulness, and propose a modification of the original SBPH polynomials used in CRM114. We also show some experimental results and end by presenting our conclusions.

SPARSE BINARY POLYNOMIAL HASHING (SBPH) CRM114

SBPH works by creating numerous distinctive features of a given text, and then apply-
Using the Naive–Bayes technique to these features, instead of directly to the words. For this purpose, the algorithm slides a word window of length five over the incoming text and, for each window state, generates a set of order-preserving subphrases containing different combinations of these words. These order-preserved subphrases are processed calculating 32-bit hashes, and with all the resulting subphrases the algorithm creates a 32-bit superhash (i.e., hashing of hashes) value that will be used to calculate the Naive–Bayes probabilities. Essentially, each subphrase tries to extract a word feature from the text. With a window size of five words, each word affects $2^{5-1} = 16$ features.

The performance of the CRM114 Mailfilter from November 1 to December 1, 2002 was 0.068 percent of false negatives and zero false positives. Its filtering speed on classification is less than 20 Kbytes per second (on a Transmeta 666 MHz), which obviously hinders the use of CRM114 for filtering at the mail server. For this purpose, an average network needs a classification speed of at least 60 kb/sg.

CRM-114 and Other State-of-the-Art Filtering Applications
Every application has distinctive characteristics which are summarized in Table 1 where we also show some of the most relevant features for these state-of-the-art anti-spam applications.

All these applications are based on the Naive–Bayes algorithm, except SpamAssassin, which is based on a combination of a genetic algorithm and rules, which is probably the reason for its having the worse rate of false positives. The false positive rates are taken after different learning cycles, depending on the application approach to the optimum value. We can see this data as a kind of “how good can it do it” upper level. The classifying experiments were made with a different number of mails; every author has used different sets, but always with a number of mails on the order of thousands. It is clear from Table 1 that CRM114 has the better false positive rate.

Automatic feature extraction for text is very important for the detection of new techniques, greatly simplifying the network engineer’s task of coping with the increasing number of new tactics. It is also known that both CRM114 and SpamProbe use a similar window word philosophy, which is explained below.

All the applications except gnus-emacs use some type of HTML processing. This is becoming increasingly important due to the fact that many spammers’ techniques are based on HTML use. We can also see that...
gnus-emacs does not have an implicit design to manage false positives.

In addition, some kind of free software license is needed to enable the network engineer to escalate or update the code, or to tune in a production domain without strategic dependency on a particular software developer. If not a completely open code license, at least a specific generic filter language that allows for some level of implementation-specific tuning should be offered. This is especially important on spam-classify applications because they generally don’t have good generalization features, but are able to successfully operate in a real-world domain after only small design changes. From our point of view, the main drawback of SpamAssassin for our purpose is that it neither has a free software license or an open generic filter language, so it works by outsourcing the spam-filter at the Internet level, a solution that is not appropriate for a corporate implementation at the server level. On the other hand, CRM114 could filter at both the MTA client level and also at the server level, but only if we could greatly increase its classification speed. The rest of the applications could run only on the client side.

THE WINDOW PHILOSOPHY

The CRM114 algorithm uses a window size of five words. Most researchers such as Brian Burton\(^8\) indicate that window sizes over two words generally produce no better accuracy or false positives rates, and in fact may well generate worse rates because they could lead to an overflow of features that could make it harder to find the relevant ones. They also greatly decrease the filtering speed. It is obvious that CRM114, which is a combination of an advanced Naive-Bayes method and polynomial hashing, has a very different window philosophy: bigger windows give polynomial techniques a more relevant weight in the final combined method. If we set the window size to two words, we use a two-variable polynomial that is less suitable than a bigger polynomial for feature extraction.

The first consideration is that following the CRM114 principles of relationship between window size and features, for a two-word window size we may extract only two features, and this seemed not to be enough. So we tried different empirical experiments, playing with both window size and word features. For this purpose we transformed the static CRM114 compiler into a dynamic matrix of pipelines (window size) and superhash phrases (number of word features) to observe the results on false positive service-level and classify-rate decisions.

THE VIRTUAL FEATURES

How are we going to extract a different number of features than those given by the SBHP algorithm? With the SBHP and a window size of \(N\) we should get \(2^N-1\). It is clear that we cannot follow this same approach when having a two-word window size. For this case, we repeat the original SBHP coefficient patterns, \(2^{2-1} = 2\) patterns up to 20, and call them virtual features.

Depending on the conjunction of window size and virtual features, we obtain different SBHP functions, taking as the algorithmic seed the original Yerazunis function with dimension \(5 \times 16\).\(^2\)

TEST I

Benchmark Corpus

Our benchmark corpus contains the learning mail set to create the .css files (superhash mapped files). Yerazunis\(^3\) recommends a learning corpus of around 0.5 Mb, and following his recommendation we have used the file nonspamtext.txt (695,111 bytes extracted from my personal inbox and public mail lists asfsdevel, SIsedu, or SL-admin) and the file spamtext.txt (536,547 bytes from a public set).\(^11\)

The test set of mail (to test the validity of our approach) came from individual donations,\(^12\) thus the learn/classify sources are independent enough to generalize results at the mail-server filter level. Following Paul Graham’s indications, they have approximately the same ham/spam distribution (ham = 170, spam = 220).
Results
The tests show the result for single-pass learning, without any retraining cycle, so the results do not show a “how good can it do it,” but are enough for comparison between different matrix sizes (see Table 2). The tests were done on an 700-Mhz Intel Pentium III Copermine (c) with 128-Mb memory, and a processor load over 99 percent for the classify process. The times were taken strictly on the classify process part. The matrix size is relative to pipelines (window size) × superhash phrases.

Figure 1 shows the values for the critical parameters. This confirms the worse false positive rates if the window size is over two words. It is important to note that the original static CRM114 matrix size is 5 × 16, so CRM114 has similar behavior on the window size value to the Burton study for the SpamProbe, which obtains better results on one- and two-word window sizes. We also obtain much better classify speed rates but this was an obvious result because, for example, the original CRM114 has to calculate 5 × 16 = 80 hash for a superhash feature, and at the best false positive performance only 2 × 8 = 16 hash should be computed for a superhash feature.

Test I Conclusions
We have proposed a new approach for implementing spam filtering on the e-mail server which is a modification and also an improvement over the state-of-the-art CRM114 technique and leads to much higher speeds, due to the fact that it uses a window word size of only two words and, surprisingly enough, also leads to better classification and false positive rates.

<table>
<thead>
<tr>
<th>TABLE 2</th>
<th>Results for Test I</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5 × 12</td>
</tr>
<tr>
<td>Spam detected</td>
<td>87.61%</td>
</tr>
<tr>
<td>False positive</td>
<td>26.19%</td>
</tr>
<tr>
<td>Sg. training spam</td>
<td>496.67</td>
</tr>
<tr>
<td>Sg. training ham</td>
<td>36.91</td>
</tr>
<tr>
<td>Training rate kb/sg</td>
<td>2.25</td>
</tr>
<tr>
<td>Classify spam</td>
<td>50.48</td>
</tr>
<tr>
<td>Classify ham</td>
<td>62.61</td>
</tr>
<tr>
<td>Classify rate kb/sg</td>
<td>21.27</td>
</tr>
</tbody>
</table>
TEST II
Now we show how our approach works with a bigger test corpus, on the order of thousands of mails from the SpamAssassin public corpus. We also check different values for relearning cycles, following our aim of obtaining zero false positives, that original CRM114 may be accurate, and checking if our conclusions for one cycle of Test I are also valid in a real domain making map processes. For this purpose we first train up to a medium-size corpus; afterwards we relearn every false classified case up to final corpus size, with a bigger ham corpus than a spam one, trying to force more ham weight on maps, for better results on false positives. We finally test from a different corpus for statistics. We combine in a natural way the .css maps and the test corpus, for example, easy ham map and the spam map, testing with easy ham and spam corpus (EASY–EASY); and we also check the cross map–corpus tests, for example, easy maps with a hard test corpus, that are not expected to get good results, but we want to check it.

Benchmark Corpus II
We focus our study on a two-word window size and the original CRM114 5 × 16 matrix, for comparison.

On the first two phases we take mail from the 2003 SpamAssassin public corpus, which has a singular ham source classification with easy to classify ham, and hard ham that usually produces a worse false positive rate. We are going to test with one ham map done with easy ham, and the other with a mix of easy and hard ham on the first phase, and only hard ham on the relearning phase. We call it mix-ham, but it is some kind of hard ham with a little easy ham. After relearning classifying thousands of mails, we obtain map files of the corpus size and mail number, as shown in Table 3.

Test II Results
The tests were done on a 700-Mhz Intel Pentium III Copermine (c) with 128-Mb memory, and a processor load over 99 percent for the classify process. The times were taken strictly on the classify process part. The matrix size was relative to pipeline (window size) × superhash phrases.

The classify speed shown in Figure 2 confirms similar conclusions to those extracted from Test I. The relearning classify rate has better results on the original 5 × 16 matrix than on some of the two-word window size matrices. But these data are not very relevant, because the critical classify speed is on production time, not during the relearning phase. On the other hand, we can see that the testing classify rates work better with a two-word window size, especially with 12 and 16 features, and that we obtain the best results on tests that use the hard ham corpus. We also can see better results

<table>
<thead>
<tr>
<th>Mail Class</th>
<th>2 × 2</th>
<th>2 × 4</th>
<th>2 × 8</th>
<th>2 × 12</th>
<th>2 × 16</th>
<th>5 × 16</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Bytes</td>
<td>Mails</td>
<td>Bytes</td>
<td>Mails</td>
<td>Bytes</td>
<td>Mails</td>
</tr>
<tr>
<td>Spam</td>
<td>349107</td>
<td>67</td>
<td>349317</td>
<td>65</td>
<td>331767</td>
<td>60</td>
</tr>
<tr>
<td>Easy-ham</td>
<td>508550</td>
<td>62</td>
<td>500116</td>
<td>62</td>
<td>498545</td>
<td>62</td>
</tr>
<tr>
<td>Mix-ham</td>
<td>414676</td>
<td>48</td>
<td>419989</td>
<td>46</td>
<td>677018</td>
<td>46</td>
</tr>
</tbody>
</table>

Mail:
Spam: 501.
Easy ham: 2551.
Hard ham: 250.
on the original $5 \times 16$ matrix than on the $2 \times 20$, so $2 \times 20$ will not be considered because of speed deficiencies.

For the next graph we consider that during the relearning phase the maps are changing, training the maps with every false classified case of the test. So the accuracy will also change and the data we show is taken from the beginning to the end of the relearning phase, for comparison with the test phase accuracy (see Figure 3).

We confirm better results on the two-word window size, and much better with more features. We also confirm the easy and hard ham SpamAssassin classification, accounting for false positives from around a 0.5 percent value for easy ham, to more than 6 percent with hard ham.

Figure 4 shows the accuracy for natural tests: one is the spam and easy ham map versus the spam and easy ham test corpus; the other is the spam and mix-ham map versus the spam and hard ham corpus.

We get zero false positives for the easy ham case. The best results are in $2 \times 12$ and $2 \times 20$ but we have to remember that $2 \times 20$
has the main disadvantage of speed. The light and dark lines are the false positives and our previous working thesis of “more weight on ham maps for better false positive results” here obtains empirical confirmation if we compare it with relearning accuracy, where the maps at the beginning were of similar size. However, we have very bad results on false negatives. We can clearly see this dependency in the $2 \times 16$, where mix-hard false negatives decrease if we compare them with other matrix sizes, but the false positives increase. The important fact for our study is that we can diminish false positives playing with learning and relearning final mail corpus size, but it increases the false negatives. We may tune for an agreement solution to obtain zero false positives also with mediocre false negatives.

Figure 5 shows that cross map–corpus combinations are not as good as the natural ones shown in Figure 4.
CONCLUSIONS
After the more informative Test II, we confirm the conclusions extracted from the results of Test I. Our approach has better accuracy and speed than the original $5 \times 16$ static matrix size of CRM114. We can also conclude that the ideal rate of zero false positives is within reach, especially if we use relearning on false cases. We have also proposed a valid strategy in two phases for achieving this. An additional conclusion is that we can give more importance to the market-critical parameter “false positives” by using a larger ham corpus size than the spam corpus size. We have also observed that a ham subdivision in hard-ham (hard-ham is made of mail that could be easily mistaken for spam, even for the trained human eye) and easy-ham is good for improving the training and relearning phases.

References
12. From a call for donations for this specific use, at the Universidad Carlos III de Madrid, 2003.
13. Personal communication with Juan Carlos Martinez, Security and Network Manager of EspacioIT, which has over 3000 mail users among different domains and mail servers. October, 2003.