Spam filtering techniques

Related content:
Stamp out spam with SpamAssassin
Subscribe to the developerWorks newsletter
Also in the Linux zone:
Tutorials
Tools and products
Code and components
Articles
Also in the Open source projects zone:
Tutorials
Projects and patches
Code and components
Articles

Six approaches to eliminating unwanted e-mail

David Mertz, Ph.D. (mertz@gnosis.cx)
Analyzer, Gnosis Software, Inc.
September 2002

The problem of unsolicited e-mail has been increasing for years, but help has arrived. In this article, David discusses and compares several broad approaches to the automatic elimination of unwanted e-mail while introducing and testing some popular tools that follow these approaches.

Unethical e-mail senders bear little or no cost for mass distribution of messages, yet normal e-mail users are forced to spend time and effort purging fraudulent and otherwise unwanted mail from their mailboxes. In this article, I describe ways that computer code can help eliminate unsolicited commercial e-mail, viruses, trojans, and worms, as well as frauds perpetrated electronically and other undesired and troublesome e-mail. In some sense, the final and best solution for eliminating spam will probably take place on a legal level. In the meantime, however, you can do some things from a code perspective that can serve as an interim solution to the problem, until (if ever) the laws begin to evolve at the same rate as public frustration.

Considering matters technically -- but also with common sense -- what is generally called "spam" is somewhat broader than the category "unsolicited commercial e-mail"; spam encompasses all the e-mail that we do not want and that is only very loosely directed at us. Such messages are not always commercial per se, and some push the limits of what it means to be solicited. For example, we do not want to get viruses (even from our unwary friends); nor do we generally want chain letters, even if they don't ask for money; nor proselytizing messages from strangers; nor outright attempts to defraud us. In any case, it is usually unambiguous whether a message is spam, and many, many people get the same such e-mails.

The problem with spam is that it tends to swamp desirable e-mail. In my own experience, a few years ago I occasionally received an inappropriate message, perhaps one or two each day. Every day of this month, in contrast, I received *many times* more spams than I did legitimate correspondences. On average, I probably get 10 spams for every appropriate e-mail. In some ways I am unusual -- as a public writer, I maintain a widely published e-mail address; moreover, I both welcome and receive frequent correspondence from strangers related to my published writing and to my software libraries. Unfortunately, a letter from a stranger -- with who-knows-which e-mail application, OS, native natural language, and so on, is not immediately obvious in its purpose; and spammers try to slip their messages underneath such ambiguities. My seconds are valuable to me, especially when they are claimed many times during every hour of a day.

Hiding contact information

For some e-mail users, a reasonable, sufficient, and very simple approach to avoiding spam is simply to guard e-mail addresses closely. For these people, an e-mail address is something to be revealed only to selected, trusted parties. As extra precautions, an e-mail address can be chosen to avoid easily guessed names and dictionary words, and addresses can be disguised when posting to public areas. We have all seen e-mail addresses cutely encoded in forms like "<mertzHIDDEN@NOSPAM.gnosis.cx>" or "echo zregm@tabfvf.pk | tr A-Za-z N-ZA-Mn-za-m".

In addition to hiding addresses, a secretive e-mailer often uses one or more of the free e-mail services for "throwaway" addresses. If you need to transact e-mail with a semi-trusted party, a temporary address can be used for a few days, then abandoned along with any spam it might thereafter accumulate. The *real* "confidantes only" address is kept protected.

In my informal survey of discussions of spam on Web-boards, mailing lists, the Usenet, and so on, I've found that a category of e-mail users gains sufficient protection from these basic precautions.

For me, however -- and for many other people -- these approaches are simply not possible. I have a publicly available e-mail address, and have good reasons why it needs to remain so. I *do* utilize a variety of addresses within the domain I control to detect the source of spam leaks; but the unfortunate truth is that most spammers get my e-mail address the same way my legitimate correspondents do: from the listing at the top of articles like this, and other public disclosures of my address.

Looking at filtering software

This article looks at filtering software from a particular perspective. I want to know how well different approaches work in correctly identifying spam as spam and desirable messages as legitimate. For purposes of answering this question, I am not particularly interested in the details of configuring filter applications to work with various Mail Transfer Agents (MTAs). There is certainly a great deal of arcana surrounding the best configuration of MTAs such as Sendmail, QMail, Procmail, Fetchmail, and others. Further, many e-mail clients have their own filtering options and plug-in APIs. Fortunately, most of the filters I look at come with pretty good documentation covering how to configure them with various MTAs.

For purposes of my testing, I developed two collections of messages: spam and legitimate. Both collections were taken from mail I actually received in the last couple of months, but I added a significant subset of messages up to several years old to broaden the test. I cannot know exactly what will be contained in next month's e-mails, but the past provides the best clue to what the future holds. That sounds cryptic, but all I mean is that I do not want to limit the patterns to a few words, phrases, regular expressions, etc. that might characterize the very latest e-mails but fail to generalize to the two types.

In addition to the collections of e-mail, I developed training message sets for those tools that "learn" about spam and non-spam messages. The training sets are both larger and partially disjoint from the testing collections. The testing collections consist of slightly fewer than 2000 spam messages, and about the same number of good messages. The training sets are about twice as large.

A general comment on testing is worth emphasizing. False negatives in spam filters just mean that some unwanted messages make it to your inbox. Not a good thing, but not horrible in itself. False positives are cases where legitimate messages are misidentified as spam. This can potentially be *very* bad, as some legitimate messages are important, even urgent, in nature, and even those that are merely conversational are ones we do not want to lose. Most filtering software allows you to save rejected messages in temporary folders pending review -- but if you need to review a folder full of spam, the usefulness of the software is thereby reduced.

1. Basic structured text filters

The e-mail client I use has the capability to sort incoming e-mail based on simple strings found in specific header fields, the header in general, and/or in the body. Its capability is very simple and does not even include regular expression matching. Almost all e-mail clients have this much filtering capability.

Over the last few months, I have developed a fairly small number of text filters. These few simple filters correctly catch about 80% of the spam I receive. Unfortunately, they also have a relatively high false positive rate -- enough that I need to manually examine *some* of the spam folders from time to time. (I sort probable spam into several different folders, and I save them all to develop message corpora.) Although exact details will differ among users, a general pattern will be useful to most readers:

- Set 1: A few people or mailing lists do funny things with their headers that get them flagged on other rules. I catch something in the header (usually the From:) and whitelist it (either to INBOX or somewhere else).

- Set 2: In no particular order, I run the following spam filters:
  - Identify a specific bad sender.
  - Look for "<>" as the From: header.
  - Look for "@<" in the header (lots of spam has this for some reason).
  - Look for "Content-Type: audio". Nothing I want has this, only virii (your mileage may vary).
  - Look for "euc-kr" and "ks_c_5601-1987" in the headers. I can't read that language, but for some reason I get a *huge* volume of Korean spam (of course, for an actual Korean reader, this isn't a good rule).

- Set 3: Store messages to known legitimate addresses. I have several such rules, but they all just match a literal To: field.

- Set 4: Look for messages that have a legit address in the header, but that weren't caught by the previous To: filters. I find that when I am only in the Bcc: field, it's almost always an unsolicited mailing to a list of alphabetically sequential addresses (mertz1@..., mertz37@..., etc).

- Set 5: Anything left at this point is probably spam (it probably has forged headers to avoid identification of the sender).

2. Whitelist/verification filters

A fairly aggressive technique for spam filtering is what I would call the "whitelist plus automated verification" approach. There are several tools that implement a whitelist with verification: TDMA is a popular multi-platform open source tool; ChoiceMail is a commercial tool for Windows; most others seem more preliminary. (See Resources later in this article for links.)

A whitelist filter connects to an MTA and passes mail only from explicitly approved recipients on to the inbox. Other messages generate a special challenge response to the sender. The whitelist filter's response contains some kind of unique code that identifies the original message, such as a hash or sequential ID. This challenge message contains instructions for the sender to reply in order to be added to the whitelist (the response message must contain the code generated by the whitelist filter). Almost all spam messages contain forged return address information, so the challenge usually does not even arrive anywhere; but even those spammers who provide usable return addresses are unlikely to respond to a challenge. When a legitimate sender answers a challenge, her/his address is added to the whitelist so that any future messages from the same address are passed through automatically.

Although I have not used any of these tools more than experimentally myself, I would expect whitelist/verification filters to be very nearly 100% effective in blocking spam messages. It is conceivable that spammers will start adding challenge responses to their systems, but this could be countered by making challenges slightly more sophisticated (for example, by requiring small human modification to a code). Spammers who respond, moreover, make themselves more easily traceable for people seeking legal remedies against them.

The problem with whitelist/verification filters is the extra burden they place on legitimate senders. Inasmuch as some correspondents may fail to respond to challenges -- for any reason -- this makes for a type of false positive. In the best case, a slight extra effort is required for legitimate senders. But senders who have unreliable ISPs, picky firewalls, multiple e-mail addresses, non-native understanding of English (or whatever language the challenge is written in), or who simply overlook or cannot be bothered with challenges, may not have their legitimate messages delivered. Moreover, sometimes legitimate "correspondents" are not people at all, but automated response systems with no capability of challenge response. Whitelist/verification filters are likely to require extra efforts to deal with mailing-list signups, online purchases, Web site registrations, and other "robot correspondences".

3. Distributed adaptive blacklists

Spam is almost by definition delivered to a large number of recipients. And as a matter of practice, there is little if any customization of spam messages to individual recipients. Each recipient of a spam, however, in the absence of prior filtering, must press his own "Delete" button to get rid of the message. Distributed blacklist filters let one user's Delete button warn millions of other users as to the spamminess of the message.

Tools such as Razor and Pyzor (see Resources) operate around servers that store digests of known spams. When a message is received by an MTA, a distributed blacklist filter is called to determine whether the message is a known spam. These tools use clever statistical techniques for creating digests, so that spams with minor or automated mutations (or just different headers resulting from transport routes) do not prevent recognition of message identity. In addition, maintainers of distributed blacklist servers frequently create "honey-pot" addresses specifically for the purpose of attracting spam (but never for any legitimate correspondences). In my testing, I found *zero* false positive spam categorizations by Pyzor. I would not expect any to occur using other similar tools, such as Razor.

There is some common sense to this. Even those ill-intentioned enough to taint legitimate messages would not have samples of *my* good messages to report to the servers -- it is generally only the spam messages that are widely distributed. It is *conceivable* that a widely sent, but legitimate message such as the developerWorks newsletter could be misreported, but the maintainers of distributed blacklist servers would almost certainly detect this and quickly correct such problems.

As the summary table below shows, however, false negatives are far more common using distributed blacklists than with any of the other techniques I tested. The authors of Pyzor recommend using the tool in *conjunction* with other techniques rather than as a single line of defense. While this seems reasonable, it is not clear that such combined filtering will actually produce many more spam identifications than the other techniques by themselves.

In addition, since distributed blacklists require talking to a server to perform verification, Pyzor performed far more slowly against my test corpora than did any other techniques. For testing a trickle of messages, this is no big deal, but for a high-volume ISP, it could be a problem. I also found that I experienced a couple of network timeouts for each thousand queries, so my results have a handful of "errors" in place of "spam" or "good" identifications.

4. Rule-based rankings

The most popular tool for rule-based spam filtering, by a good margin, is *SpamAssassin*. There are other tools, but they are not as widely used or actively maintained. SpamAssassin (and similar tools) evaluate a large number of patterns -- mostly regular expressions -- against a candidate message. Some matched patterns add to a message score, while others subtract from it. If a message's score exceeds a certain threshold, it is filtered as spam; otherwise it is considered legitimate.

Some ranking rules are fairly constant over time -- forged headers and auto-executing JavaScript, for example, almost timelessly mark spam. Other rules need to be updated as the products and scams advanced by spammers evolve. Herbal Viagra and heirs of African dictators might be the rage today, but tomorrow they might be edged out by some brand new snake-oil drug or pornographic theme. As spam evolves, SpamAssassin must evolve to keep up with it.

The README for SpamAssassin makes some very strong claims:

> In its most recent test, SpamAssassin differentiated between spam and non-spam mail correctly in 99.94% of cases. Since then, it's just been getting better and better!

My testing showed nowhere near this level of success. Against my corpora, SpamAssassin had about 0.3% false positives and a whopping 19% false negatives. In fairness, this only evaluated the rule-based filters, not the optional checks against distributed blacklists. Additionally, my spam corpus is not purely spam -- it also includes a large collection of what are probably virus attachments (I do not open them to check for sure, but I know they are not messages I authorized). SpamAssassin's FAQ disclaims responsibility for finding viruses; on the other hand, the below techniques do much better in finding them, so the disclaimer is not all that compelling.

SpamAssassin runs much quicker than distributed blacklists, which need to query network servers. But it also runs much slower than even non-optimized versions of the below statistical models (written in interpreted Python using naive data structures). For more on the pros and cons of SpamAssassin, read "Stamp out spam with SpamAssassin" (*developerWorks*, September 2002).

5. Bayesian word distribution filters

Paul Graham wrote a provocative essay in August 2002. In "A Plan for Spam" (see Resources later in this article), Graham suggested building Bayesian probability models of spam and non-spam words. Graham's essay, or any general text on statistics and probability, can provide more mathematical background than I will here.

The general idea is that some words occur more frequently in known spam, and other words occur more frequently in legitimate messages. Using well-known mathematics, it is possible to generate a "spam-indicative probability" for each word. Another simple mathematical formula can be used to determine the overall "spam probability" of a novel message based on the collection of words it contains.

Graham's idea has several noteworthy benefits:

1. It can generate a filter automatically from corpora of categorized messages rather than requiring human effort in rule development.
2. It can be customized to individual users' characteristic spam and legitimate messages.
3. It can be implemented in a very small number of lines of code.
4. It works surprisingly well.

At first blush, it would be reasonable to suppose that a set of hand-tuned and laboriously developed rules like those in SpamAssassin would predict spam more accurately than a scattershot automated approach. It turns out that this supposition is dead wrong. A statistical model basically just works better than a rule-based approach. As a side benefit, a Graham-style Bayesian filter is also simpler and faster than SpamAssassin.

Within days -- perhaps hours -- of Graham's article being published, many people simultaneously started working on implementing the system. For purposes of my testing, I used a Python implementation created by a correspondent of mine named John Barham. I thank him for providing his implementation. However, the mathematics are simple enough that every other implementation is largely equivalent.

There are some issues of data structures and storage techniques that will effect operating speed of different tools. But the actual predictive accuracy depends on very few factors -- the main significant factor is probably the word-lexing technique used, and this matters mostly for eliminating spurious random strings. Barham's implementation simply looks for relatively short, disjoint sequences of characters in a small set (alphanumeric plus a few others).

6. Bayesian trigram filters

Bayesian techniques built on a word model work rather well. One disadvantage of the word model is that the number of "words" in e-mail is virtually unbounded. This fact may be counterintuitive -- it seems reasonable to suppose that you would reach an asymptote once almost all the English words had been included. From my prior research into full text indexing, I know that this is simply not true; the number of "word-like" character sequences possible is nearly unlimited, and new text keeps producing new sequences. This fact is particularly true of e-mails, which contain random strings in Message-IDs, content separators, UU and base64 encodings, and so on. There are various ways to throw out words from the model (the easiest is just to discard the sufficiently infrequent ones).

I decided to look into how well a much more starkly limited model space would work for a Bayesian spam filter. Specifically, I decided to use trigrams for my probability model rather than "words". This idea was not invented whole cloth, of course; there is a variety of research into language recognition/differentiation, cryptographic unicity distances of English, pattern frequencies, and related areas, that strongly suggest trigrams are a good unit.

There were several decisions I made along the way. The biggest choice was deciding what a trigram is. While this is somewhat simpler than identifying a "word", the completely naive approach of looking at every (overlapping) sequence of three bytes is non-optimal. In particular, considering high-bit characters -- although occurring relatively frequently in multi-byte character sets (in other words, CJK) -- forces a much bigger trigram space on us than does looking only at the ASCII range. Limiting the trigram space even further than to low-bit characters produces a smaller space, but not better overall results.

For my trigram analysis, I utilized only the most highly differentiating trigrams as message categorizers. But I arrived at the chosen numbers of "spam" and "good" trigrams only by trial and error. I also picked the cutoff probability for spam rather arbitrarily: I made an interesting discovery that no message in the "good" corpus was assigned a spam probability above .0071 other than two false positives in the .99 range. Lowering my cutoff from an initial 0.9 to 0.1, however, allowed me to catch a few more message in the "spam" corpus. For purposes of speed, I select no more than 100 "interesting" trigrams from each candidate message -- changing that 100 to something else can produce slight variations in the results (but not in an obvious direction).

Summary
Given the testing methodology described earlier, let's look at the concrete testing results. While I do not present any quantitative data on speed, the chart is arranged in order of speed, from fastest to slowest. Trigrams are fast, Pyzor (network lookup) is slow. In evaluating techniques, as I stated, I consider false positives very bad, and false negatives only slightly bad. The quantities in each cell represent the number of correctly identified messages vs. incorrectly identified messages for each technique tested against each body of e-mail, good and spam.

Table 1. Quantitative accuracy of spam filtering techniques

Technique	**Good corpus** (correctly identified vs. incorrectly identified)	**Spam corpus** (correctly identified vs. incorrectly identified)
"The Truth"	1851 vs. 0	1916 vs. 0
Trigram model	1849 vs. 2	1774 vs. 142
Word model	1847 vs. 4	1819 vs. 97
SpamAssassin	1846 vs. 5	1558 vs. 358
Pyzor	1847 vs. 0 (4 err)	943 vs. 971 (2 err)

Resources

- The TDMA home page provides more information about the Tagged Message Delivery Agent.
- You can get more information about ChoiceMail from DigitalPortal Software.
- Pyzor is a Python-based distributed spam catalog/filter.
- Vipul's Razor is a very popular distributed spam catalog/filter. Razor is optionally called by a number of other filter tools, such as SpamAssassin.
- SpamAssassin is the most popular rule-based spam filter. Read more about it in "Stamp out spam with SpamAssassin" (*developerWorks*, September 2002).
- Read Paul Graham's essay "A Plan for Spam."
- Eric Raymond has created a fast implementation of Paul Graham's idea under the name "bogofilter." In addition to using some efficient data representation and storage strategies, bogofilter tries to be smart about identifying what makes a meaningful word.
- My own trigram-based categorization tools are still at an early alpha or prototype level. However, you are welcome to use

them as a basis for development. They are public domain, like all the tools I write for *developerWorks* articles.

- Lawrence Lessig has written a number of books and articles that insightfully contrast what he metonymically calls "west-coast code" and "east-coast code," in other words, the laws passed in Washington D.C. (and elsewhere) versus the software written in Silicon Valley (and elsewhere). I've written a short review of Lessig's *Code and Other Laws of Cyberspace*. See Lessig's Web site for more to think about.
- Find more Linux articles in the *developerWorks* Linux zone.

About the author

David Mertz dislikes spam. He wishes to thank Andrew Blais for assistance in this article's testing, as well as for listening to David's peculiar fascination with trigrams and their distributions. David may be reached at mertz@gnosis.cx; his life pored over at http://gnosis.cx/publish/. Suggestions and recommendations on this, past, or future articles are welcome.

e-mail it!

What do you think of this article?

○ Killer! (5) ○ Good stuff (4) ○ So-so; not bad (3) ○ Needs work (2) ○ Lame! (1)

Comments?

Submit feedback