;login:

THE MAGAZINE OF USENIX & SAGE

June 2003 • volume 28 • number 3

inside:

USENIX & SAGE

The Advanced Computing Systems Association &
The System Administrators Guild

ISPadmin

Bayesian Spam-Filtering Techniques

by Robert Haskins

Robert D. Haskins is an independent consultant specializing in the Internet Service Provider (ISP) industry.

rhaskins@usenix.org

and Rob Kolstad

Rob Kolstad is currently Executive Director of SAGE, the System Administrators Guild. Rob has edited *;login:* for over ten years.

kolstad@sage.org

Introduction

In this installment, Rob Kolstad and I look at the arrival of what has become known as the "Bayesian" spam-filtering technique. The reason Bayesian filtering is so intriguing is its high accuracy rate (for Haskins, 97.6% as tracked by POPFile; for Kolstad, approaching 99.8%) and low false-positive rate (estimated by Haskins at 1% or less, again with POPFile; Kolstad is getting 0.1%), making it one of the most useful anti-spam tools available to date. Before getting into how specific Bayesian implementations work, it is useful to have a good understanding of how Bayesian theory (which has been around since the 18th century) works.

While the vast majority of the discussion revolves around spam, the Bayesian technique can easily be used to augment (or replace) the "usual" filters found in mail client programs. Also, it is likely this sort of filtering functionality will be integrated into the email client program, eliminating the need for proxying except under nonstandard circumstances.

Bayesian Theory

Regrettably, "Bayesian theory" is not precisely the heart of the statistical spam-filtering methodology under discussion, but the name is used universally for this technology. Briefly, let's enter the world of probability theory for a quick refresher. If you are not a math or statistics aficionado, please skip to the next section.

Recall that probabilities assign a numerical value to the belief (or observation) that an event might or might not happen. The probability of a coin flip ending up "heads" is 0.50 – half the time one flips a coin, one should get "heads." (Certain coins are weighted so that this doesn't happen, but they are less interesting to use for this example.) The notation for simple probability is also simple:

```
P('heads') = 0.5
```

This is read "The probability of 'heads' is 0.5."

Probabilities are interesting when they help one deduce facts or understand information better. Knowing that using pharmaceutical P1 cures disease D1 99% of the time is a good piece of data to have when one has disease D1. Knowing that pharmaceutical P2 cures D1 only 1% of the time might very well influence a decision to use P1 instead of P2.

Conditional probability is just a little more complex. Conditional probability addresses the probability of an event's occurrence **given that some other event has occurred**. Consider the probability that one has cancer given that one is a regular smoker of cigarettes. This is written:

```
P (cancer | smoker) = [some number]
```

This is read "The probability of 'cancer' given 'smoker' is . . . ."

Some interesting math about conditional probabilities. Consider the two probabilities:

```
P (cancer | smoker)
```

P (cancer | nonsmoker)

For this discussion, these are all the possibilities that were "observed" or "measured." Summing these with the probabilities of smoking tells us the total probability of cancer:

P(cancer) = P (cancer | smoker) x P(smoker) + P (cancer | nonsmoker) x P(nonsmoker)

Rev. Thomas Bayes published an article in 1763 with a more advanced formula. It concerns computing probabilities for a hypothesis given a potentially new set of observations. Consider a new observation "O" and its impact on a hypothesis "H".

$$P(H \mid O) = \frac{P(H) \times P(O \mid H)}{[P(H) \times P(O \mid H) + P(\text{not-}H) \times P(O \mid \text{not-}H)]}$$

Which is read: "The probability of our hypothesis 'H' given the new observation 'O' is . . ." and then the formula. The various components:

- P(H) is the probability we had assessed before the new observation showed up.
- P(O | H) is the probability of the observation given that the hypothesis H is true.
- P(not-H) is just 1 - P(H), the probability that H is not true.
- P(O | not-H) is the probability of the observation in the world where the hypothesis is false.

Let's use an example to demonstrate the formula. Suppose a disease appears in 1% of the population:

P(disease) = 0.01
P(not-disease) = 0.99

and let's presume that a test for the disease is positive 80% of the time when the disease is present and 10% of the time when the disease is not present:

P(positive test | disease) = 0.80
P(positive test | NOT disease) = 0.10

Then we can answer the question, What is the probability of the disease if the test on someone is positive?

Here's the formula:

$$P(H \mid O) = \frac{P(H) \times P(O \mid H)}{[P(H) \times P(O \mid H) + P(\text{not-}H) \times P(O \mid \text{not-}H)]}$$

Filling in the values from above (H is "has the disease"; O is "positive test"):

$$P(\text{disease} \mid \text{positive test}) = \frac{0.01 \times 0.80}{[\,0.01 \times 0.80 + 0.99 \times 0.10]}$$

$$= 0.074766\ldots$$
$$= \sim 0.075$$

Thus, we can conclude using Bayes' theorem that a positive test for the disease gives one only a 7.5% chance of actually having the disease – surprisingly less than one might expect!

Why is it so low? Because 99 out of 100 people don't have the disease, yet 20% of those 99 people will test positive. This huge number dwarfs the small percentage who actually have the disease.

Bayesian Spam Filtering

Spam prevention methods use Bayes' theorem in its chain form, with ever more observations entering into the formula until a final probability is assessed. Of course, in the spam-prevention case, the hypothesis is "This note is spam." It is the observations that make the result interesting.

The key paradigm in Bayesian spam prevention is the observation of sets of words or combinations of words. Consider observations of the form:

- A single word (e.g., "Viagra" or "enlargement")
- Pairs of words (e.g., "money fast" or "bank account")
- Triples, quads, or longer sequences of words (e.g., "make money fast," *friend@public.com*)
- Header information tokenized in a clever way. Here's a header:
  Message-ID: <20010214170157.C323@ubiqx.mn.org>

One of Paul Graham's experiments tokenizes pairs like this:

```
Message-ID 20010214170157
Message-ID C323
Message-ID ubiqx
Message-ID mn
Message-ID org
```

The imagination is the limit on these!

Bayesian methods read (usually after de-MIMEing) email (in whole or in part) and try to deduce if the mail is spam or not. They tokenize the mail (often removing punctuation; sometimes – but, these days, more often not – removing capitalization), create a zillion observations (e.g., every possible run of 1, 2, 3, 4, or 5 words and every properly ordered subset of that list), and then look up these observations in a huge database. The probabilities are all combined (when they exist), and a final result is determined. This is compared to a threshold to determine whether the program believes the message to be spam.

Note that while this sounds simple, the numbers involved theoretically range from 1.0 down to unbelievably small numbers for combinations that only show up once. The software must be careful that probabilities smaller than 1.0e-350 or so don't end up rounding to 0.

Initial probabilities are calculated by running a set of training mail (both spam and non-spam) through a program that tokenizes and calculates probabilities for the various phrases that appear. This is interesting because some industries use the term "offer" a lot, a word commonly used in spam. By training with longer phrases, a database of probabilities can be built with relatively small amounts of very representative good and bad email.

By way of example, Kolstad's training base now has these statistics:

```
Spam email:  317 messages, 130,770 lines of text (3.3MB)
Non-spam email: 63 messages, 55,133 lines of text (0.75MB)
```

These training bases have been augmented over time by adding to them each time the filter makes a mistake.

The key paradigm in Bayesian spam prevention is the observation of sets of words or combinations of words.

The beauty of a statistical approach to identifying spam is that it "learns" the methods spammers use to get past more static filters.

History of Bayesian Theory as It Relates to Spam

Interestingly, Microsoft was granted a patent for a method of filtering spam using the Bayesian methodology (USPTO patent # 6161130). However, this work was preceded by at least two other published works. One is "ifile," a filter for a number of command-line-based mail programs including MH, Pine, and procmail-compatible clients. It is unclear what, if anything, Microsoft did with this work and associated patent.

The problem in the past with the Bayesian method was that the accuracy of various software implementations wasn't nearly close enough to 100%. Users complained about both false negatives (spam delivered as legitimate mail) and false positives (real email that was incorrectly identified as spam). Few users will tolerate more than a minuscule false positive rate (0.5% may actually be too high). Of course, the more spam there is, the more such tolerance might go up. Now that everyone is inundated, those sending important mail know that it can get lost in the flotsam and jetsam.

What makes newer algorithms (implementations, anyway) different? According to Paul Graham's follow-up paper, "Better Bayesian Filtering," things are improving because of:

- The size of the body of training mail
- Using mail headers as tokens or not
- The message tokenization methodology
- Improved probability calculations
- More intelligent bias against false positives

The beauty of a statistical approach to identifying spam is that it "learns" the methods spammers use to get past more static filters. As a result, spammers are forced to make their messages look just like your regular email, thereby reducing the effectiveness of spam. This could cause spammers to stop spamming!

Likewise, as Paul Graham points out, eliminating 99% of delivered spam would presumably eliminate 99% of the responses a spammer seeks (i.e., to sell a product or service). Few businesses can continue with a 100 times decrease in revenue. Maybe that would cause them to stop spamming. We can only hope.

PC Email Proxy Client (POPFile)

POPFile is a Perl-based POP3 proxy that runs on your PC, between your email client program (such as MS Outlook, Eudora, etc.) and the POP3 server (or anti-virus software). It takes a little bit to set up and train, but once it has a "corpus" of spam, it works very well.

The concept of POPFile is this. First, you set up your email client to work with the program. Consult the POPFile Web site for detailed instructions on how to configure POPFile for various email clients, including MS Outlook, Outlook Express, and Eudora. Also, consult the HOWTO discussion group for other setup notes, including working with virus scanners, setting up Netscape Mail, etc.

Once your email client has been configured, you begin categorizing email per the "buckets" (classifications) you set up. Haskins created two buckets, one called "spam" and one called "notspam." There is no reason why you cannot create other buckets for other classifications of email to enhance your email client's filtering capability.

When your mail client checks mail, POPFile receives the message, runs it through its filters, and either tags the subject line with the bucket name or adds a header to the

message "X-Text-Classification: <*bucket name*>" if your email client can filter based on arbitrary mail headers.

Unlike some other filtering programs, it specifically does not come with a preloaded corpus of spam. Instead, the program builds a very specific corpus for your incoming email, making it very accurate.

POPFile's weaknesses include:

- No support for IMAP
- Issues with clients who leave their mail on the server
- No mechanism for inputting a preexisting corpus of spam and non-spam to "jump start" the filtering

POPFile is certainly an excellent implementation. If you have a POP3 mailbox, and don't store your mail on the server, we heartily recommend it!

It is very likely that Bayesian-style filtering will be implemented in other email client software. The Mozilla folks are hard at work implementing Bayesian filtering in their Mail application. Check out Mozilla version 1.3 alpha if you are interested. While initial reports indicate the functionality doesn't work perfectly, it is a step in the right direction. The SpamAssassin folks have also started using Bayesian filtering.

Command-Line Email Client Support

Bayesian solutions exist for the server and command-line email clients such as Pine. Most of these are hooked in via procmail, so if you host your mail on your own server, then you are set. If not, ask your provider if it can activate procmail for you.

Some Bayesian command-line email client filters include ifile, SpamProbe and bogofilter. More information on these filters is available on their respective Web sites.

Server Side

There are several server implementations of Bayesian filtering. Most are easiest to implement via SpamAssassin or procmail.

Probably the best known is CRM114, which, incidentally, did not start out as a Bayesian filter but as a rather large regular expression matcher. However, Bayesian filtering was added to the mix as of November 2002.

Your best bet is probably to implement the native SpamAssassin Bayesian filters available in 2.50, or to attempt the CRM114 hooks to provide Bayesian filtering for an entire server. Be aware that this is bleeding-edge software, and plan accordingly.

Experience with CRM114

Kolstad's mail shows up the old-fashioned way: It is delivered to a mailbox on his desktop. He configured his .forward file like this:

```
"| /home/kolstad/bin/nospam"
```

and then created a simple Perl program to run the mail against CRM114:

- Read in the message.
- Run the message through CRM114 with the spamfilter configuration file.
- Examine the exit code to deduce if the message is spam.
- Non-spam messages are appended to the mailbox, with locking; biff is called with a seven-line networking sequence.

REFERENCES

POPFile home page: *http://popfile.sourceforge.net/*

Paul Graham's "A Plan for Spam": *http://www.paulgraham.com/spam.html*

International Society for Bayesian Analysis (ISBA): *http://www.bayesian.org/*

Microsoft Bayesian spam-filtering patent: *http://patft.uspto.gov/netacgi/nph-Parser?patentnumber=6161130*

Old ifile README: *http://www.ai.mit.edu/~jrennie/ifile/old/README-0.1A*

New ifile site: *http://www.nongnu.org/ifile/*

The RAND MH Message Handling System: *http://www.ics.uci.edu/~mh/*

Pine: *http://www.washington.edu/pine/*

Procmail: *http://www.procmail.org/*

Learning to Filter Spam E-Mail: A Comparison of a Naive Bayesian and a Memory-Based Approach: *http://citeseer.nj.nec.com/androutsopoulos00learning.html*

POPFile HOWTO forum on SourceForge: *http://sourceforge.net/forum/forum.php?forum_id=234504*

SpamProbe: *http://spamprobe.sourceforge.net*

Bogofilter: *http://bogofilter.sourceforge.net/*

Mozilla page regarding anti-spam features in 1.3a: *http://www.mozilla.org/mailnews/spam.html*

SpamAssassin: *http://spamassassin.org/*

CRM114: *http://crm114.sourceforge.net*

Better Bayesian Filtering: *http://www.paulgraham.com/better.html*

- Spam messages are either tagged (for testing!) or diverted to another file for later review.

He has been running this system unmodified for about two months. Two additional scripts augment the two training databases ("this is spam"; "this is not spam") with mail that was misclassified.

He gets 250 emails per day, unless something special is happening (e.g., broken salary survey, some other SAGE member interaction, etc.). Right now, the spam filter is missing about one spam per day (false negative) and tagging one or two emails per week as spam. Happily, those emails really do resemble spam and would not have been missed.

The nospam program itself requires an average of 55 ms per message; CRM114 averages 270 ms. Incoming mail suffers only the smallest of delays.

Note well that the CRM114 distribution out-of-the-box did not perform as hoped (though it compiled with no problems). Some of the scripts have been rewritten (one had a huge regular expression converted to a Perl program that ran 100–1000 times faster) and converted for spam handling. Likewise, the new scripts dramatically ease the user interface.

The absolute best part of CRM114 (and Bayesian spam detection in general) is that it does not require the relatively frequent updating that SpamAssassin really needs. Half-a-dozen "retrainings" (a single keystroke on clever mail readers) per week keep it in good shape.

If you're interested in packaging this solution for simple and more widespread distribution, please contact Kolstad.

Conclusion

The Naive Bayesian method as outlined by Paul Graham is a very effective way to filter spam. The algorithm has a very high accuracy rate and a decreasingly small false positive rate, making it the best currently available method to identify spam. There are PC client implementations (including POPFile) as well as UNIX-based packages (CRM114, bogofilter, SpamAssassin plug-in) available.