Bayesian Learning Email Cleansing.

In its original meaning, “spam” was associated with a canned meat from Hormel. In recent years its meaning has changed. Now, an obscure word has become synonymous with unsolicited, unwanted email. It is possible that an unsolicited email is welcome by the recipient. In that case, it is still spam, but spam that hit its mark and was desired.

Spam causes problems for many people, and it costs a lot of money. Interestingly enough, it costs money for almost everyone except the person generating spam. If you wanted to send a thousand full-color glossy fliers via postal service, it would cost you for the paper, printing, and postage. To send a thousand spam email messages filled with whatever message you desire doesn't cost anything except for a few minutes of your time. In fact, you could send out a million copies of your spam message for the same cost as a thousand. So, if it is free to send spam, who does it cost and how? Spam costs you and me, regular users of email, money. It costs bandwidth between computers and networks. It costs CPU time to route, process, and pass it along. It also costs disk space. A typical spam message may be 100 kilobytes and may contain a picture or two. I personally get over two hundred spam email messages every day.

Why we receive spam

You may wonder, why do I get spam? How does the spammer get my email address? Well, there are many ways in which a spammer can get “a hold of you”.

One way is by scanning the World Wide Web for occurrences of strings that look like email addresses. This probably isn’t very efficient, but if you can collect a

million email addresses this way, you can easily spam them all, simply because it is free. There are many email harvester programs that will search the web for email addresses., which makes the inefficiency unimportant to spammers. Another way to get addresses is by reading posts to Usenet. It is a certainty that if you post on Usenet with your real email address it will be harvested by spammers.

Then there are programs that are frequently called “spyware”. These programs install themselves on your local computer, usually without your knowledge, and relay information about you back to a central database. Spyware programs are not limited to email addresses. They spy on all your activities, and generally have access to everything on your computer.

Some shareware download sites require you to enter a valid email, and then they use that email to send you the correct link to download the program you wanted. The primary reason to do this is to verify that you gave them a real email address, which in turn becomes a valuable commodity for them to sell. A validated list of real email addresses sells for much more than a list of unverified ones. A similar trick, commonly used to verify an email address, is a link that is opened automatically in the preview panel which in turn will notify the spammer that this is in fact a good email address.

You can rest assured that almost any time your email address goes out across your internet connection, someone somewhere is going to catch it and try to profit from it by selling it to spammers or spamming you themselves.

In the following paper I discuss classification of spam, most spam can be easily detected and organized into one or two groups. I talk about common methods for decreasing the number of spam email messages as well as a brief survey of other spam

filtering techniques. I then advocate and explain a Bayesian technique for detecting spam. Finally, I discuss variations on the original Bayesian technique.

Classifying Spam

Spam classification helps us understand the problem of how to eliminate it. If we divide spam into meaningful groups, we can talk about automated approaches for sorting out each type. There are as many ways to divide spam as there are spammers, but most spam can be classified. One simple classification is pornographic spam or non-pornographic spam. Since the Internet has become widely available, the porn industry has taken hold and embraced the technology. On Google as of 4/6/2003, a search on the word "porn" returns over 67 million results. With this much competition on the net, spammers try countless ways to get their message out, to try to get you to come to their site. Once they have you there, with endless promises of free products they will attempt to get you to enter your credit card number “for verification purposes only”.

Out of the two hundred spam email messages I get every day, 50% are for a pornographic service of some sort. Traditionally, these spam email messages will contain a selection of colorful words about some particular fetish or sexual adventure. These emails usually contain embedded pictures, normally of an X-rated nature. They also contain short flash animation movies and may contain embedded sound. If you have your email client set to preview your messages (a common feature, in modern email clients), as soon as you open such an email, it will display the text, words, photos, animations and sounds. This can cause problems in a work environment and may turn out to be quite embarrassing. As it turns out, these pornographic types of spam are easy to

filter with mail-server based tools. I will go into much more in-depth discussion about those tools later on.

So, pornographic spam accounts for a substantial percentage of the spam in a person's inbox. What about the rest, you ask? Well, the other spam I receive is sales not related to pornography. The items sold vary from home mortgages to miracle vitamins to insurance. These sales pitches all contain similar vocabulary and tone, and these qualities can be readily detected by filtering techniques.

Another possible classification of spam that is sometimes used consists of checking the email headers for correctness. If you research the "From" fields of spam you will find that in almost every case this field is invalid. This is a useful classification for a couple of reasons. Firstly, there is legislation currently pending that makes it illegal to send spam without proper return routing information. The other reason this can be useful it is easy to throw out email messages that don't have proper return routing info as spam, because legitimate email never comes this way.

Decreasing the number of spam email messages

There are many automated methods for determining and blocking spam. One of these is the use of white and black lists. This method is probably the simplest to implement, and it's fairly effective. First, you create a list of people allowed to send you email. Anything from them should be delivered to your inbox. Another list of people, who are known to send spam, is kept. You blacklist those, sending their emails to the trashcan or to a spam folder. This list is often ineffectual because spammers almost always use an invalid email address and never send two spam email messages from the

same address. Lastly, you put all of the mail, not covered by the white or black list into a holding folder, where you can make a determination on which list this email belongs to and update the lists accordingly.

After a short time most anyone who sends you legitimate email will be in your white list. Your spam list will be huge, but mostly unused. Your holding folder is where most all your spam will go. The problem is that some good email will also be sent here. These emails are known as "false positives." False positives in spam detection and filtering cause you to lose information that you might have been expecting. From this point on I talk about ways to improve our filters, to decrease the number of spam email messages without increasing the number of false positives.

One method of decreasing the number of spam email messages you see is to analyze more parts of the email than just the "To" and "From" header fields. You can make white lists based on "To", "CC", "BCC", or subject fields. Another method is to block entire domains from known spammers. This has an undesired side effect of blocking possibly legitimate email from that domain.

We can generalize this technique to scan not only the headers but also the body of an email. You find words that frequently appear in spam and rarely in legitimate email. Then create a black list of these words. For example, you might block every email that contains the word "mortgage" because you never want to receive information about home mortgages. A worthwhile refinement to this method would be to look for two-word phrases, thus making your filtering rules more efficient and less prone to false positives.

Implementing a Bayesian Filter

After reading Paul Graham's "A Plan for Spam" (August 2002), [1]I thought his approach would work. Also I could think of some improvements to his version that may well increase the effectiveness of the tool. I needed to look at a new email as essentially neutral, scan through it, pick out important words, and then use these words to assign a probability that the email is spam. This method works well when combined with Bayesian artificial intelligence learning algorithms. A Bayesian algorithm is one in which we look at a set of known data. This data has already been classified manually. Using this data we then build rules which can in turn answer questions about future or unknown data.

Procedurally, for the end user, such a system is easy to operate. All that is required is changing the way email is deleted. The end user needs to make two folders for deleted email and one folder for spam. I chose folder names of __Kill, __Read and __Spam. Once these folders are set up, email can be read as before. When deleting or archiving email, the user needs to evaluate it. If an email was something that the user wanted to see, it needs to be moved or copied to the __Read folder. If not - copied into the __Kill folder. That is all there is to it from the end users' perspective.

The Program

On the mail server I set up three programs and three databases to handle the back-end work. Each of these items can be adjusted to affect performance.

[1] http://www.paulgraham.com/spam.html

Program 1: Tokenizer

The tokenizer scans email files and breaks them up into "tokens". Each token can be a single word, a pair of words, or a phrase. Then the tokenizer records the number of times a particular token is seen into a database. The tokenizer program is executed from a cron process once an hour for each folder __Kill and __Read. I found that, in early runs, single words worked as well as tokens. Each group of non-white space characters was considered a word. I allowed HTML tags to be considered as words. This turned out to be a very good indicator for me, since most people that send me non-spam emails don't send HTML-based messages. Some HTML tags like the <STRONG> tag, <FONT> tag and the color red are all excellent indicators that a given email is, in fact, spam. I throw out all tokens that are longer than 60 characters. This had the effect of ignoring mime-encoded attachments. After two months of working with the system, I changed tokens to be two words each, instead of one. This has the effect of finding word-phrases and allowed me to predict spam more accurately than with single word tokens. The drawback of double word tokens is that your database grows slower (because tokens are combined). For example, the sentence: *the dog ran fast*, would be tokenized into 3 tokens instead of four (thedog dogran ranfast) vs. (the dog ran fast). The net result is that you will need larger training sets to come up with the same accuracy as with single word tokens.

Another adjustment to the tokenizer is what to count as white space. I experimented with changing white space characters to include everything from only true

white space to everything non-alphanumeric and a myriad of things in between. These turned out to have almost no effect on the system's ability to predict spam.

Program 2: Hash compute

This program looks at the two databases created by the tokenizer and computes the probability that a particular token occurs in spam and the probability that it occurs in non-spam. It keeps track of how many email messages it has seen, how many times a particular token appeared in spam, and how many times a particular token appeared in non-spam. Using a Bayesian algorithm hash-compute assigns probabilities to each token.

Program 3: Scanner

Before an email gets into the end user's inbox, the scanner runs on each new email that is seen by the system. The first thing the scanner does is tokenize the email message. Then it reads the databases created by the two preceding programs, and picks out the 15 most interesting tokens in the message. Interesting tokens are tokens that have probability that is farthest from 50%. Tokens with probability close to zero are a strong indication of non-spam. Tokens with a probability close to one are a strong indication of spam. I have found that after a relatively small number of emails (somewhere around 3000), most of the tokens considered interesting by the system have probabilities greater than 98% or less than 1%. This polarizing effect gave me a clear picture of what the system thinks spam is and what it thinks isn't. This shows that the technique is always conclusive. It either classifies an email as spam or as not, never in the middle.

After the scanner has run, it comes up with its list of tokens. It computes an average of the percentages of the interesting tokens. This percentage is the probability

that the email is spam. In my implementation I classify everything with a percentage over 95% as spam, everything else non spam. This 95% was picked essentially at random. I varied it throughout the testing period, to try to find an optimal percentage. The changing of this cut-off percentage had almost no effect on what is classified as spam because the results are so bi-polar, meaning either very close to one or very close to zero.

Example

The following examples should help to clarify exactly how the system works. These examples are small but clearly illustrate the effectiveness of the filters. For this example we will ignore all words in the sample data set except for the word "mortgage."

Word: Mortgage				
Database		**Value**	**Variable Name**	**Description**
WordCountNoSpam		10	**G**	Number of times the word mortgage occurred in non spam emails.
WordCountSpam		1000	**B**	Number of times the word mortgage occurred in spam emails.
Control	Goodcount	500	**GC**	Count of number of non spam emails processed so far.
	Badcount	1000	**BC**	Count of number of spam emails processed so far.

From the above table we can calculate the probability a future email that contains the word "mortgage" is in fact spam. We perform the calculation in the following way:

$$P = \frac{(B \div BC)}{((B \div BC) + (2 \times G \div GC))}$$

The doubling of the G value is to bias the system against false positives. It gives positive occurrences a higher weight than they have "earned".

$$.96 = \frac{(1000 \div 1000)}{((1000 \div 1000) + (2 \times 10 \div 500)}$$

We calculate probabilities in this fashion for each token we see and then record them in the probability database.

Variations

Some interesting variations on the above system were tried.

Variation 1: I allowed for three-word tokens. This had no noticeable effect on outcomes and no benefits over two-word tokens. I found that with one-word tokens the filter could accurately predict 98% of spam emails correctly with training sets 50% smaller than training set size for two-word tokens. The actual training set cardinalities for one-word tokens were 450 and for two-word tokens were 889.

Variation 2: Instead of having just two classifications, spam and non-spam, the system could sort the emails into any number of folders based on the content. For example you could have the system set up with four folders: __Spam, __Baseball, __Work, and __Personal. The scanner, tokenizer, and hash-compute functions will have to be modified, so they would check each of these folders and then could sort the email based on multiple criteria.

There are some problems that must be overcome by the system in order for this to work. When sorting spam from non-spam, you can safely assume that if something isn't spam, then it is non-spam. When using this technique to filter emails into multiple folders, you must keep track of percentages for each possible classification. I implemented a prototype of this system and found that it has some difficulty

distinguishing between the various types of legitimate email. I think the cause of this was that spam is all basically the same thing, a sales pitch of some form or another. The legitimate emails contain various messages that are very close to each other. To my system, conversations about baseball and basketball appear to be very similar, and the differences are difficult to detect. This occurs because these two conversations contain similar vocabularies (score, ball player, team, and so forth). I abandoned this approach because of simpler viable alternatives such as filtering email based on keywords found in the "Subject", "From", and "To" header fields.

Conclusion

I have come to the conclusion that to most effectively deal with spam email messages, a combination of the above techniques works the best. First I filter all my email from mailing lists, automated systems emails, and other personal email with known header properties (whitelisting). With the remaining email I use the Bayesian technique described above with two-word tokens, to determine if something is spam or not. If it is determined to be spam, I file it as such, otherwise I deliver it to my inbox. This method works very well with one exception. I don't ever receive any false positives (not one in the past 3 months). However, sometimes I receive emails in my inbox that would have ideally been blocked – that is, false negatives. The one characteristic they all have is that they are short, one-line emails, usually containing only a link. The Bayesian filter will in time learn to properly classify these.

As long as we allow for trusted email (accepting email from anyone), there will be people who send spam. The Bayesian filter approach is dynamic, easily implemented

on a variety of systems, and learns as time goes on, making it an ideal solution. By taking Paul Graham's work and improving on the tokenizer, adding two-word tokens and experimenting with other tokens, I have been able to make a very effective text filter that catches spam almost 100%. There are still improvements to be made in the field of spam email detection; I plan to continue my research.