Combating Spam

Jeff Sturgis - 261437

November 13, 2003

Project Conceived By:

Evangelos Kranakis
COMP 4108
School of Computer Science
Carleton University

Abstract

Unsolicited e-mail is also known as spam. Spam has become a nuisance to nearly all recipients. Using an electronic form of solicitation has the benefits of being fast, cheap, and massive. For these reasons, business owners see it as an advertisement opportunity, lame people see it as a way to distribute viruses, and con artists see it as a way to con. Combating spam is important to ensure the continued use of e-mail. Thirteen speakers proposed possible solutions to spam at ”Spam Conference” held January 17th, 2003 at the Massachusetts Institute of Technology.[1] This essay will examine ten of these protocols, the results of each proposal, as well as further discuss their effectiveness.

Contents

1 Introduction

The problem of unsolicited e-mail is a growing concern in the workplace. Every time a worker must delete an unsolicited e-mail, also known as spam, it takes time. By multiplying this time by the number of e-mails, the number of workers, and number of working days, the cost accumulates rapidly. Spam is expected to cost the U.S. more than $10 billion dollars in 2003. [2]

Spam is a security threat because of the effect on productivity. For one million e-mails, the cost is cheap for the spammer. But, if every recipient takes one second to delete the message, it costs “5 man-weeks”[13]

Paul Graham states that “the lowest rate seems to be about $200 to send a million spams.”[13] The response rate to the spam can be reduced by filtering the spam effectively so that people never see it. This should decrease the income of the spammers. If this income falls low enough, it would not be worth the spammers while to send these messages.[13]

As methods are proposed to detect spam, spammers create methods to fool detection. Spammers have overcome obstacles such as static filters and subject/message hashing. Static filters are mail filters which remove all mail containing certain words. For example, remove all spam containing the word “viagra”. Using static filters on spam are not effective because they could result in legitimate e-mails not arriving. It is also possible to fool static filters by changing the appearance of text via HTML tags or inserting characters between letters. One way a computer program compares two messages for equality is by applying the message to a hash

function. If two messages have the same hash function then it can be said that they are equal. However, inserting a small amount of random text into each message will ensure that the messages have different hash values. The same occurs with subject hashes. Spammers are clever and it must be assumed that they will find weaknesses in spam combating strategies if such a weakness exists.

Since the Internet is not 100% reliable, some e-mails do get lost due to server or other failure. In order to combat spam effectively, some e-mails might get lost in transit. It is also possible to have some spam messages get through. When applying an e-mail filter to detect spam, legitimate e-mails which are captured as spam are said to be false positives. Undetected spam which arrives in a person's in-box is said to be a false negative. The goal in combating spam is to minimize both false positives and negatives, in order to provide the user with as much spam free experience and e-mail loss as possible.

In the following sections, the latest techniques used to attack spam are outlined. These techniques include using static filters such as whitelists, blacklists, and other regular expression recognition used to sort e-mail into groups. There are many Bayesian techniques with effective results. The use of compression to generate features from text is discussed, as well as systems put in place by commercial anti-spam corporations.

2 Gnus vs. Spam

2.1 Protocol

Gnus is used to handle large amounts of incoming messages such as news and e-mail. The strength of Gnus is it's capability to group these messages into several classifications. A package written in Lisp for Gnus called spam.el has the capability of sorting e-mail into spam and ham categories. Ham is classified as being any e-mail that is not spam. Gnus uses static filters and has learning capabilities. Gnus classifies e-mail based on regular expressions matched to a set of rules. Input from various mail sources into Gnus will be applied to user configured settings. This allows comparison to blacklists, whitelists, real-time databases, and uses all existing spam to classify new spam. Messages grouped as spam will remain flagged as spam as long as they are unread in the users spam directory. In fact, all unread messages are by default classified as spam. Spam which has been read can be moved into the spam folder manually. Another important aspect of Gnus is its user definable capability. A user may define any message, or any subject containing a keyword to be grouped together.

Gnus, when properly configured, will call spam-split on the incoming messages. The messages are then classified according to rules defined in spam.el and any additional rules added by the administrator and user. The spam.el package maintains a series of boolean flags which the user may turn on such as use-whitelist. The messages are parsed and sorted according to the configured flags.[3] The user may then examine their e-mail in their sorted groups and make any corrections to ensure more reliable future spam classification.

2.2 Results

I could not find any published results of the effectiveness of spam.el in terms false negatives or positives. Gnus is a sorting program designed to take advantage of all the static filter capabilities. Once Gnus is successful in filtering out all known good and bad mail, it can focus on detecting spam from the remaining messages. The CPU cycles are reduced by this filter cleaning out all the spam and ham it can. Unless of course, the Bayesian filter parses the good and bad mail anyway in order to learn better.

2.3 Effectiveness

The use of static filters can be very effective especially for their whitelist use. If a user maintains a good whitelist, it will guarantee arrival of those e-mails. Another interesting application of static filters is their capability to receive messages with keywords in their subject line. If a person maintains a web-site and instructs people submitting e-mails to do so with a keyword in the subject line, then it would be very difficult for spammers to keep up with all the keywords. Even if a spammer did maintain a database of keywords, unlike e-mail addresses, keywords can be changed very easily. A user may keep the same address but only accept messages with his currently defined keyword (or keywords). However, if a person without the keyword is trying to get through and cannot, then this can work against the client.

Blacklists are a good way to prevent spammers from using the same e-mail address repeatedly. However, due to services such as Hotmail or Yahoo mail, it is easy for spammers to gain a new address. The real-time databases containing blacklists are

good only if they are up to date. The real-time whitelists needs prevention from spammers getting on the list. An empty whitelist is no use. The more massive the whitelist the better.

By default, all unread mail is classified as spam.[3] This avoids having to click on the message and manually mark it as spam. This could be risky because users who rely heavily on their spam filters and never check their junk-mail, might get false positives and have them classified as spam. A Bayesian filter works on probabilities. If the properties of spam are contaminated with properties of ham, it would alter the probability. The more times it falsely classifies, the greater the chance for false classification of the next e-mail.

3 Sparse Binary Polynomial Hash Message Filtering and The CRM114 Discriminator

3.1 Protocol

Bill Yerazunis is the creator of the CRM114 Discriminator. CRM114 is designed to calculate the overall probability of the message being spam. The message is broken up by using each word as a token. CRM114 calculates using a sliding window of five tokens. For an implementation with a sliding window of size N, the tokens are considered in 2^{N-1} different ways.[4] Let P_j denote each way the tokens can be used. Let Z_j be the 32-bit hash value of P_j. For each Z_j value, it is important to use as many of the bits as possible to avoid collisions. The function for Z_j is designed to use as many bits as possible. The Z_j values are then compared to two buckets, one containing spam hash values and the other ham hash values. The ratio between spam counts and ham counts can be used to calculate an approximate probability of the message being spam. CRM114 uses Naïve Bayesian Chain Rule to calculate the overall classification of the message.[5]

3.2 Results

In their latest newsflash, CRM114 claims to be "4 times faster than SpamAssassin while still retaining our high (better than 99.9%) accuracy after training. From Sep. 1 through 14 2003, I had ZERO errors on over 2500 e-mails on my live incoming e-mail stream."[6]

3.3 Effectiveness

It is important to realize that this program accomplishes a lot. CRM114 generates more features than words. For every word, CRM114 considers 2^{N-1} permutations of how the next $N-1$ words could appear after it. For a sliding window such as 5, for every word, 16 combinations of it followed by the next four words are considered. This will consider 16 times more features. If it is shown that fewer features could be used and still maintain a high average, it will speed this program up . Also, if generating many features proves to be necessary, using the most significant tokens might allow fewer words to be considered.[12]

It is impressive that a filter could catch that much spam. However, I wonder about the scalability. It would be interesting to see statistics on how much processing time is required per megabyte of e-mail and how CRM114 performs on a larger mail server. It should also be noted that CRM114 works by itself without the use of whitelists.[5] CRM114 is definitely worth considering as a free effective spam filter on a personal mail server.

4 Automatic Feature Induction for Text Classification

4.1 Protocol

Jason Rennie proposes using feature induction to learn the characteristics of a spam automatically. This will prevent an "endless cycle" of filter writers keeping up with the latest spam techniques.[8] Let $r = (t, l)$ denote a rule defined by a token t and label l. The minimum description length(MDL) is an algorithm which will calculate the least number of bits required for each token in a set of rules. That is, MDL ensures each token is of minimal length. He believes that this approach is good because it is automatic and can be personalized. He insists that the most general features of the message are what is used for the classification of text.[8]

The MDL algorithm does not remove words, nor does it consider frequency. This method extracts features from text. It uses a greedy algorithm to extract optimal tokens. A decision list is constructed by a tree of rules. Rules can be used to compress the message, and extract features.[7]

Feature induction is faster than a Bayesian approach.[8] Shih et al. feel that "[m]ost highly accurate text classifiers take a disproportionately large amount of time to handle training examples."[9] The number of features used in an e-mail can slow down it's classification. That is, an e-mail with thousands of features would be classified slower than an e-mail with, say, ten features. Classification time could be decreased by using only a subsample of the features within the text.

However, this leads to discarding features of the text.[9]

4.2 Results

In order to test the efficiency of Rennie's protocol, the greedy algorithm must be improved. It will also be necessary to test for frequency of the extracted features.[8] When Shih et al. tested feature selection, they found that it "was a disappointment on both speed and accuracy grounds."[9]

4.3 Effectiveness

This method of spam detection is based on feature extraction. Compressing the text features in the message will facilitate spam decision making. However, this algorithm has not yet been built to handle frequency. This is important because in advertisements, products are listed over and over again. This information is not extracted. It is important to be able to classify the information in a small form. Hash functions are good for doing this but hash functions have collisions and a static key set. Lossless compression does not have collisions and will handle messages more quickly.

Shih et al. discuss random sampling.[9] Random sampling is the idea that in order to reduce computation, you take a random sample of the text as the representative. Ideally, this will lead to faster classification because the set of data is smaller. Before writing this paper, I thought that random classification would be a good idea to reduce the data set and speed up the process. However, if spammers use lots of legitimate text to disguise the small amount of commercial text and random sampling is done on this set of data, it would likely miss the even fewer

keywords in the commercial text. Clearly, random sampling would not work on every message on its own. But a message such as the one I described might make it past a Bayesian filter as well.

Shih et al. also discuss using probabilities based on mean data instead of summing up or multiplying probability through Bayesian techniques.[9] A spam filter could look at statistics such as, does this message have a word with a frequency close to the frequency of that word in a spam message? I think this goes to show that there are many more ways that we have not even thought of, which we could use to extract classifying features from e-mail.

5 The Spammers' Compendium

5.1 Protocol

John Graham-Cumming is the creator of Popfile. Popfile is designed to:

- Check for e-mail containing no words, only HTML with links, and pictures
- Read domain names in HTML tags
- Strip headers of superfluous information, ignore numbers, ignore text set to the same colour as the background
- Ignore HTML comments or other tags that would be ignored by a browser
- Merge letters separated by common characters into words
- Detect foreign characters and replace them with their English encoding
- Remove long random words

Essentially, Popfile reduces the e-mail to the spam that it is with the addition of a few words unlikely to confuse the Bayesian filter. After each message is reduced, a Bayesian filter is applied to the message. This will classify the messages and place them into their appropriate categories.

5.2 Results

Popfile obtained a 94.11% average accuracy across all the users. Figure 1 represents a pie chart showing the percentage of users who obtained 100, 99, 98, ... ,90, and less than 90% accuracy. These results are good considering many users are still training their filter.[11]

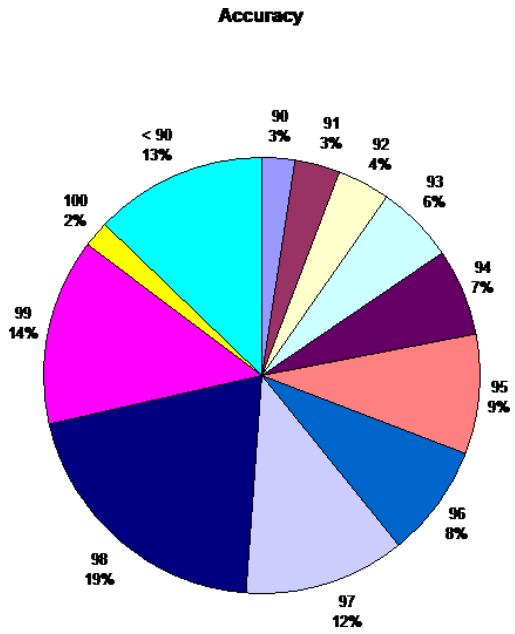

Figure 1: Percentage of users getting percentage of spam[11]

5.3 Effectiveness

Some spammer tricks could be used to fool a Bayesian filter. One trick in particular is inserting a lot of legitimate text to disguise the small amount of spam text. If a Bayesian filter were to read the legitimate text, it would likely want to classify the message as non-spam. When it reads the text of the spam thereafter, it would have a probability close to 1 that it is not spam. The few amounts of remaining text might not shift the probability. In this case, it is good to strip the message of the comments, headers, and background coloured text.

A spammer is easily able to add legitimate text from any source. Since the message becomes ugly and cluttered with all this additional text the spammer will try

to hide this text in comments or using font the same colour as background. When other Bayesian filters comes across these messages, they too would have to take similar precautions to avoid the extra text weighing heavily on the message. Removing it is one solution. Paul Graham proposes another solution which I tend to favour. He takes the fifteen most significant features. This could be the fifteen spammiest features. Based on these fifteen features, the filter will be able to tell if the message is spam, and it will disregard all extra text.

Removing text from a message could damage the Bayesian filter. If the text was left in, the Bayesian filter could learn from it. Removing the text only ensures that the Bayesian filter has no idea how to classify messages filled with extra text. If a spammer can think of a way to hide text that Popfile has yet to discover, it will allow those messages to pass until Popfile is updated.

Popfile might work better as a front-end to immediately detect spam. If all of Popfile's knowledge was focused on checking for hidden text, javascript, HTML properties, and immediately classifying the message as spam, it could save computation time for the Bayesian filter. Besides, who uses javascript and same coloured font as the background in their e-mail? As for reassembling words to their original state such as V*I*A*G*R*A, this word would be caught by a well trained Bayesian filter and will prevent a spammer from using the same trick over and over. Once all possible combinations of V#I#A#G#R#A have been used, there will be no choice but to use the same ones over again. It also forces the spammer to keep track of every permutation of every word they have ever used.

Popfile is designed to catch spammer tricks. By doing this, John Graham-Cumming must commit to keeping up with the latest spam techniques, implementing countermeasures and releasing updated versions of Popfile. Though he is committed to take this task on, he should stop and ask himself if what he is doing is overkill. If there are two solutions to a problem, both yielding nearly perfect results, one solution requires no human intervention and the other needs constant human intervention, then which program would you want?

6 Better Bayesian Filtering

6.1 Protocol

Paul Graham uses entire words stripped slightly of excess punctuation. The probability of the word occurring is based on learning data and calculated using a Bayesian filter. If there is no match, then different permutations of the word are considered and the highest value it finds is used. The spam probability is calculated using the 15 most significant tokens. Any result less than some threshold (he uses 0.9) is considered as spam.[12]

6.2 Results

When Paul Graham tested his filter, out of 1750 spams, 99.5% spam was caught, and he had 0.03% false positives. His false positives include two commercial newsletters, and somebody from Egypt writing all in capital letters. Also, one notice that something he bought was back-ordered and the last was a party invitation from Evite. These are noted as bugs which he thinks he will be able to fix. Graham also had two false negatives. His filter has trouble filtering e-mail claiming to be from a woman with a URL to some dating service. This is because of the casual language used rather than a sales pitch. The other false negative message was for contracting programming services. This passed through the filter likely because he is programmer.[12]

6.3 Effectiveness

I like the idea Paul Graham uses of extracting the 15 most significant features from the spam. Computing which 15 features are the most significant should be costly compared to computing the probability of the message being spam based on the

15 features. The results are still impressively high. Even the defying messages which he labels as bugs seem reasonable and excusable for the filter. When you are training something else to read and classify your mail you cannot expect it to be perfect. If he were training a person to filter spam, those messages might have gotten by them also. The only one who knows for sure that the message is spam is the recipient.

The Bayesian approach is very effective. It is clear that Bayesian filters will continue to undergo study for speed and efficiency. There should be a lower bound on speed such that the program will be able to run with 99.5% efficiency but if you make the filter classify any faster it will sacrifice accuracy. Two important questions for the Bayesian filter are:

How will it respond to messages disguised in regular text?
How will it respond to attacks?

I think that if the Bayesian system were put in place widely enough for the spammers to fight against it, the spammers might have some success. What would happen to the Bayesian filter if a spam message got through with one thousand occurrences of the word ”and”? Then thousands poured in with the word ”the”? Next thousands more pour in with the words ”and the” combined? If these messages are marked as spam, then e-mail containing many occurrences of ”and the” could result in false positives. What I would like to see in a Bayesian filter is where it is completely self taught by default and also allows full administrative capabilities to override statistical anomalies. It would also be beneficial for an ad-

ministrator to be able to edit filter settings so that mail can be classified differently. For example, spam containing a lot of legitimate text should be handled in a different way then a brief message with a URL. The long message might be stripped of HTML comments and other things where the brief message could focus mainly on the URL.

The important thing about a filter is its ability to run with minimal human intervention. I think that the Bayesian filter can do that now because the spammers are not attempting to disguise their text to hide it. Possibly, they are focused on larger scale spam filters such as Brightmail.

7 Spam Filtering: From the Lab to the Real World

7.1 Protocol

Joshua Goodman is a researcher at Microsoft. He feels that a Bayesian machine learning approach is the way of the future to fight spam. The spam rates are best calculated using false positives and miss rates(false negatives). Hotmail implements Brightmail. The installation of Brightmail showed a spammer response. It is good to have junk and non-junk buttons handy for easy manual classification. The technique used to sort spam should allow messages to be validated in any language.[14]

7.2 Results

Results for Bayesian approaches can be found in CRM114,[4] Better Bayesian Filtering,[12] and Bayesian Spam Filtering Tweaks.[22] Results for Brightmail can be found in Fighting Spam in Real Time.[17]

7.3 Effectiveness

The effectiveness of several Bayesian methods occur throughout this paper. Since Goodman lists no specific implementation there is no Bayesian approach to evaluate.

Brightmail results are not overly impressive. Some may excuse this because it must deal with such an abundant amount of mail.

8 Heuristics in the Blender

8.1 Protocol

Michael Salib uses a heuristics approach to filtering spam. He build tokens with adjacent punctuation, and does not tokenize by punctuation. Heuristics are used just like tokens to increase spam probabilities. Instead of using a genetic algorithm, the filter implements a linear combination of heuristics to calculate spam probability. Assign spam to 1 and ham to -1. Sum up all the heuristics and if you get a number > 0 classify it as spam. This method is faster and will allow users to define their own corpus. As the corpus grows however, the accuracy of the filter goes down. Salib believes this can be solved by focusing on the latest e-mails rather than the entire set.[15]

8.2 Results

When Salib tested on a 200 message corpus, he obtained 93% effectiveness. When he used his heuristics approach on a 6000 message corpus, it was only 50% effective.[15]

8.3 Effectiveness

This filter is faster than a Bayesian filter. It also takes good heuristics into account. It does not perform as well as a Bayesian filter and when you increase the corpus, probability should not drop that significantly.

Salib raises the issue of users having their own message corpus. This is a good idea because the mail will be filtered specifically for each individual. However, each user having their own corpus will result in multiplicative data. A collective

system will allow one large central corpus and thus have no duplicate data. Both ways have their advantages and disadvantages. In the future it will be likely that whenever possible, a system with small individual corpus' will be implemented. It might also prove infeasible for large scale systems to be implemented in this way thus sharing one large corpus of data.

Michael Salib's heuristic approach confuses me in some ways. One heuristic in particular is the real-time blacklists(RBLs). He says that you should use the heuristics like tokens and sum up their probability.[15] However, a heuristic such as the sender's name appearing in a blacklist could be an immediate spam classification. It could be argued that if the message is spam, then it should contain many more spam features. When these features are combined with the RBL data, the message will indeed be classified as spam. An implementation could also be created in such a way that it takes into consideration all the heuristics which immediately classify it. It seems as though all the people at the conference are trying to get away from immediate classifying features. Though such filters would classify messages quickly it may result in false positives.

9 Spam vs. Forty Years of Machine Learning for Text Classification

9.1 Protocol

David Lewis proposes a feature extractor used to classify text as spam and ham. This feature extractor will be a supervised learning machine because algorithms work faster than humans. Lewis recommends down-casing text, removing punctuation, numbers, and other mark-up. He stresses the importance of not locking in on any one particular feature. The algorithm uses mathematics such as boolean algebra to calculate probability with fast and simple algorithms. Lewis lays some fundamentals for fighting spam. He believes it is important to know how your program works. He also thinks that more training data should be beneficial to the algorithm. He feels larger corpus' could be shared to increase all the filters knowledge of spam. Lastly, he insists the pressure of the e-mail filters are slowly making spam look legitimate.[16]

9.2 Results

David Lewis has no results. He does not really propose an exact implementation, or strategy. He merely states his opinions on how e-mail should be filtered.

9.3 Effectiveness

Lewis thinks it better to down-case. This reduces the size of the database because it merges many instances to one. That is any instance with a capital will be reduced to its lowercase. The effect of this is a generalization of the information. Removing information from an e-mail is still controversial. Clearly, the Bayesian

filters that do not remove information get great results. Bayesian filters are capable of shrinking the data corpus to a few hash tables of useful information. The focus on implementing Naïve Bayes should be on speed performance while still maintaining high accuracy.

Lewis believes it is important to know how the implementation works.[16] If a mail administrator only knows that a program is supposed to remove 99.5% of spam, and not how the program actually works, then if the filter does not work to standard it poses a major problem. It is also beneficial for the administrator to be able to test and fine-tune their filter. This could be because of speed requirements or the need to represent data in another way. When a filter is written and tested, the conditions for which it was tested will differ on the new machine. If one filter performs with 99% effectiveness, and the same filter at 95% on another machine, customizability might be the factor which brings them both to 99%.

10 Fighting Spam in Real Time

10.1 Protocol

Ken Schneider is in charge of Brightmail's "roadmap".[21] All e-mail entering the Brightmail mail server is forwarded to Brightmail Logistics and Operations Center (BLOC server). On the BLOC server, a gigantic corpus of e-mail is used to compute spam probability. The result is returned to the mail server and put in the user's junk-mail box if it is spam. BLOC is a learning machine capable of creating new rules every few minutes. Every rule is tested for quality assurance.[17]

10.2 Results

Brightmail uses 2 million honey pot accounts to attract 2 billion messages of spam per day. These honey pots obtain an estimated 10% of all Internet e-mail traffic. Brightmail writes 30,000 new blocking rules per day.[19] When Brightmail was put to the test by PC Magazine, it generated zero false positives.[20]

10.3 Effectiveness

Brightmail claims to filter 11% of the worlds e-mail.[18] For a filter that deals with that much e-mail it is incredible that it can generate zero false positives when put to the test. Brightmail is the enterprise solution of choice. They claim to protect 1,400 of the worlds leading enterprises.[20] Clearly, Brightmail is the Microsoft of spam fighting. Large companies can rely on it because it is the standard and offers support. However, they charge $1,499 per year for 49 users.[19] This is an expensive way for a corporation to fight spam. Companies could use their own implementations of spam fighting software at a much lower cost. However,

getting companies to switch to open source free software when they are reliant on other means can cause havoc. Besides companies are obviously willing to pay.

11 Bayesian Spam Filtering Tweaks

11.1 Protocol

Brian Burton decided to implement a Bayesian filter called SpamProbe using C++ as the language of choice and Berkeley DB for his database. His design decisions were:

- Use word pairs
- Use frequency
- Remove HTML tags
- Fold 8 bit characters
- Use tokenizer that allows HTML parsing
- Remove excess punctuation
- Convert to lowercase
- Use some terms according to where they were found

[22]

11.2 Results

SpamProbe catches 99.7% of spam and continues to climb.[22]

11.3 Effectiveness

Brian Burton's Bayesian implementation has many interesting qualities. He uses word pairs as suggested by Paul Graham.[12] The CRM114 uses a sliding window of five tokens.[4] However, Burton yields 99.7% accuracy using only two tokens. Using only two tokens will run eight times faster than CRM114. This proves that the CRM114 uses an overkill Bayesian algorithm.

Frequency is important. Burton's decision to use frequency is wise because a message containing one instance of "viagra" is much less incriminating than one containing multiple instances.

Burton acknowledges that removing information from the e-mail was his most controversial decision. Burton decides to remove HTML tags out of fear that these tags might immediately classify messages as spam. In order to ensure there are no false positives from HTML tags, he discards them.[22] Whether or not removing HTML tags is the correct approach is a matter of opinion. Burton can obviously back up his reasoning for removal.

Folding 8-bit characters is an interesting way to fight foreign spam. He converts characters that are not in the ASCII English range to 'z'. This allows for huge e-mail compression of such messages and easy classification.[22] However, the drawback is that foreigners cannot use his filter for themselves.

It just goes to show, you can do many things to a Bayesian filter and still get high accuracy. Many of his design decisions differ from other Bayesian implementations. However, with 99.7% accuracy, why not convert to down-case to shrink the database? Why not try everything we can to make the Bayesian filter faster and rely on a small corpus or hash table?

12 Conclusion

Bayesian filters need to be trained on a large corpus of data. As it is trained, the results should continue to climb. Paul Graham feels that in order to combat spam, you have to understand what the spammer needs to do to break your filter. He believes that using a Bayesian method will limit spammers to creating e-mail that is indistinguishable from legitimate e-mail. They would have to do this while still getting their message across. Graham regards this as severe constraint.[13] However, it should also be assumed that the spammer has access to the filtering software and possibly a test account. They would be able to test their messages out on such a filter before distributing them. If they know the threshold of the filter, they can tune their messages accordingly. They know that there is no point in sending one million messages that would not get through so they find a message that will.

If the spammer must resort to sending a friendly e-mail with a URL, they will. When they do, how will the Bayesian filter perform?
Will it
a) Let the messages through as long as they are of that form?
b) Get too many spam of that form and start blocking all messages of that form?
c) Start creating false positives for all valid e-mail of that form?
d) All of the above?

Just because the spam gets uglier with lots of legitimate text disguising the spam does not guarantee the spammer will give up. And, since the spam is right there, one click away, the response rate might not go down. It is not a difficult problem to

create a legitimate e-mail. We do it all the time with every non-spam. Spammers so far have only had to change a few things to allow their spam to get through. They have not yet resorted to using friendly messages with a URL. If they do and the response rate is still satisfactory, then the result is legitimate e-mail of the form spammers choose to be blocked. Of course this is worst case.

Assume case 'a' above. All e-mail makes it to the in-box which is good because your friends can send e-mail of that form. This can be accomplished by turning the Bayesian filter off before spam that has been disguised as legitimate mail is used. When we find spam in our in-box we would likely know it is spam because we do not recognise the e-mail address.

If we get to case 'b', then we are in trouble. In case 'b', we assume legitimate e-mail makes it through the filter. However, we would have to do something for legitimate e-mail to get through. This thing that we do cannot be static across many users or it would act as a weakness. There would need to be some form of "I am legitimate" password in the e-mail that would allow the message through.

False positives are not acceptable. These must be minimized at all costs. We must take action in 'b' or implement case 'a' so that we do not have to deal with false positives. So with only a Bayesian filter we either have to deal with small amounts of spam getting through or setting up a system to allow legitimate messages through.

Consider a system where 100% of spam gets filtered correctly. There are no false

positives or false negatives and every spam that is caught is placed in a user's junk-mail which will expire messages every few days. This is the ideal system. Now, how do we accomplish it? Let us assume that this system was nearly perfect and let one spam through. This spam would have to be detected by a user. I propose the answer to the perfect filter would be the answer to how the user goes about coming across this message and realizing that the spam got through. Firstly, the user would look at the address. They do not recognise the address or person whom it is from. The person may just delete the message if they did not know who it was from. Secondly, the subject has nothing to do with anything they have ever talked about by e-mail. In that case, they would delete it as spam or maybe the recipient thinks they know somebody who might have that address and might use that subject. Lastly, the user opens the e-mail which is some form of spam or leads to spam.

Spammers harvest addresses. If you create a unique e-mail address, the spammer could not spam you unless they knew that address. The user of this new e-mail address should take the time between getting the new account and when the spammer obtains their address to establish a whitelist. By adding everybody who e-mails you, and everybody you e-mail to a list, you can establish a trusted group of users. Once the first spam arrives, the user should turn on the spam filter.[13] This way any replies from people will automatically be accepted. The whitelist is a fast and effective way to reduce the number of e-mails that go to the Bayesian filter. In this step, other heuristics such as real-time blacklists could be used to automatically classify the messages as spam.

The second step is to analyze the subject. Let us use every string parsing token trick in the book. Subjects should have well-known constraints readily available to people. These constraints are reasonable assumptions to which a subject should adhere to in order to be not automatically classified as spam. Establish rules for subjects such as: They are not more than 20 words. They are not more than 200 characters. The messages which follow these rules then use a method to detect legitimate keys in subject. This can be used to accept messages from people replying to you. If the whitelist is created above, verifying a key could still be useful. For example, consider the case of a user who has their e-mail forwarded to another account. When that user replies to a message it would be from a different e-mail address, not picked up by the whitelist. However, verifying the key would allow the subject to automatically get through. Running expensive operations such as sparse binary polynomial hashing on 20 words seems to be an effective method for further classification. It would be beneficial to have each user maintain their own hash tables to ensure that it detects spam targeted to each user. By this, I mean that somebody who gets Greek spam does not affect the probabilities of some one who gets German spam.

The third step will scan the message. Hopefully, the message has be caught by this step. It is still possible that legitimate e-mail lurks and should be in the inbox. In this step, it would be safe to assume that the messages are from people that you have never e-mailed before, and you know how incriminating their subject was. Paul Graham's proposed method for detecting spam would be used to classify the message. I choose this way because he uses the most significant features of the message to calculate message fate. Graham obtains an excellent catch

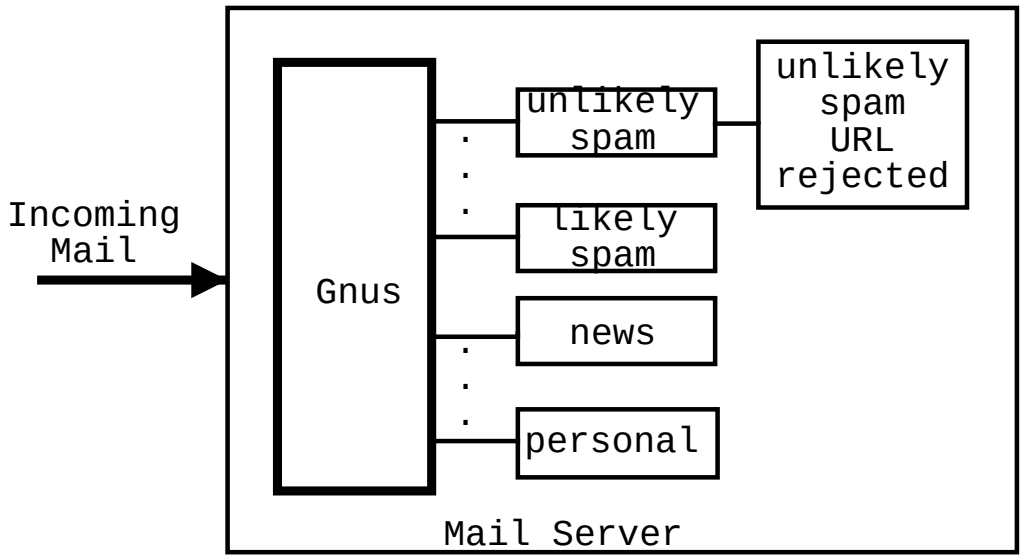

Figure 2: Gnus e-mail setup[11]

rate. He achieves 99.5%.[12] Assuming we achieve the same rate for the small group of messages that were not classified in the first two steps, our rates would be even higher. In my opinion, Gnus is the quickest most effective way of sorting mail. Any mail filtering system should be used in combination with Gnus or a similar sorting utility. You would definitely want to take advantage of the whitelist and its regular expression recognition for subjects. Furthermore, Gnus could sort your mail into groups such as more than 95% likely as spam, 90, 85,..., whatever you want and you can decide your own thresholds of which to check. When I used whitelists, I found it very effective. One tip I can pass on which may be obvious is to check your junk-mail when you are expecting something. False positives often occur when you sign up somewhere and get account information or purchase confirmation after you buy something. Add those people to your whitelist. If you are using Gnus, you can add all messages from that one address to a specific folder.

The ability to be customized is a great software feature. Gnus does well to ensure that your e-mail goes exactly where you want.[3]

Furthermore, to decrease other false positives consider how people are going to obtain your e-mail. When you give your e-mail away, tell them what to put in the subject in order for it to be accepted. For instance, on your web page tell them to put ”bug report” in the subject. This way all the messages with ”bug report” in the subject are accepted and organized for you in their own folder.

I am sure that I have not exhausted every case for which to eliminate false positives. However, whitelists are very fast and efficient. They should be used as much as possible to ensure good mail gets through. When all the mail has been sorted, go over the least incriminating non-white messages. Web-crawl any URLs in the message and re-apply Bayesian to those sites. Incriminating sites should get rejected to another folder.

This system is designed to catch ham automatically. It assumes the majority of legitimate mail travels through whitelists. Messages not caught will be further subjected to a Bayesian filter. Incriminating messages will be placed accordingly and innocent messages separately. Innocent messages will then be subject to further evaluation by re-parsing them and downloading the HTML and URLs within the message. If the URL is incriminating the message will be shipped to a URL rejected folder. This will allow the user to read all their valid e-mail, and to check up on messages that appear legitimate to the Bayesian filter. Hopefully, that will all be ham. The user might want to delete the messages instead of classifying

them as spam since it is likely that any spam that got across, only did so by fluke or by a form which the person would normally willingly accept.

The only attack I can think of for a message to get through is to be a non-incriminating message with a non-incriminating URL. An attached picture might be able to do this because spam filters are not designed to tell if images are spam. It would become extremely obvious when all spam managing to get through has an attached picture. Especially, if all these pictures are from people not on the whitelist. Gnus could easily classify these messages to another directory.

In general, to ensure good e-mail classification, a grouping program like Gnus is essential. Whitelists and white subjects will allow quick ham classification. Since, Bayesian filtering is the most CPU intensive of this system, it would be good to have a Bayesian algorithm that does a fast job. The results of the Bayesian filter should be handled by the grouping program so that the user may disregard definite spam. This should achieve the goal of having the user never see the vast majority of their spam.

References

[1] Spam Conference 2003
<http://spamconference.org/proceedings2003.html>
(cited 30 October 2003)

[2] Jonathan Krim. Spam's Cost To Business Escalates: Bulk E-Mail Threatens Communication Arteries.
<http://www.washingtonpost.com/ac2/wp-dyn/A17754-2003Mar12>
(cited 28 October 2003.)

[3] Teodor Zlatanov. Gnus vs. Spam.
<http://lifelogs.com/spam/spam.pdf>
(cited 26 October 2003)

[4] Bill Yerazunis. Sparse Binary Polynomial Hash Message Filtering and the CRM114 Discriminator.
<http://crm114.sourceforge.net/CRM114_slides_16.ppt>
(cited 30 October 2003)

[5] Bill Yerazunis. Sparse Binary Polynomial Hash Message Filtering and the CRM114 Discriminator.
<http://crm114.sourceforge.net/CRM114_paper.html>
(cited 30 October 2003)

[6] Bill Yerazunis. Sparse Binary Polynomial Hash Message Filtering and the CRM114 Discriminator.
<http://crm114.sourceforge.net/>
(cited 12 November 2003) http://crm114.sourceforge.net/

[7] Jason Rennie. Automatic Feature Induction for Text Classification.
<http://www.ai.mit.edu/~jrennie/papers/aimemo2002.pdf>
(cited 6 November 2003)

[8] Jason Rennie. Automatic Feature Induction for Text Classification.
<http://www.ai.mit.edu/~jrennie/talks/spam03.pdf>
(cited 6 November 2003)

[9] Laurence Shih, Jason D. M. Rennie, Yu-Han Chang, David R. Carger. Text Bundling: Statistic-Based Reduction.
<http://www.ai.mit.edu/~jrennie/papers/icml03-bundlesvm.ps.gz>
(cited 11 November 2003)

[10] John Graham-Cumming. The Spammers' Compendium.
<http://popfile.sourceforge.net/SpamConference011703.pdf>
(cited 6 November 2003)

[11] John Graham-Cumming. The Spammers' Compendium.
<http://popfile.sourceforge.net/stats.html>
(cited 6 November 2003)

[12] Paul Graham. Better Bayesian Filtering
<http://paulgraham.com/better.html>
(cited 6 November 2003)

[13] Paul Graham. A Plan for Spam.
<http://paulgraham.com/spam.html>
(cited 7 November 2003)

[14] Joshua Goodman. Spam Filtering: From the Lab to the Real World.
<http://216.239.39.104/search?q=cache:pskoBPHDcLQJ:
research.microsoft.com/~joshuago/spamconferenceshort.ppt+
Spam+Filtering:+From+the+Lab+to+the+Real+World&hl=en&lr=
lang_en&ie=UTF-8>.(This is a google cached page)
(cited 7 November 2003)

[15] Michael Salib. Heuristics in the Blender.
<http://web.mit.edu/msalib/www/writings/talks/
spam-filtering-conference/>
(cited 7 November 2003)

[16] David Lewis. Spam vs. Forty Years of Machine Learning for Text Classification.
<http://www.daviddlewis.com/publications/slides/
lewis-2003-0117-spamconf-slides.pdf>
(cited 7 November 2003)

[17] Ken Schneider. Fighting Spam in Real Time.
<http://www.brightmail.com/press/2003_MIT_Spam_Conference/>
(cited 7 November 2003)

[18] Press Release. Brightmail Hails Passing of U.S. Senate Bill 877
<http://www.brightmail.com/pressreleases/102203_senate_
bill_877.html>
(cited 7 November 2003)

[19] John Breeden II.Press Release. Brightmail app makes target practice of spam. Government Computer News, 29 September 2003.
<http://www.brightmail.com/pdfs/GCN.pdf>
(cited 7 November 2003)

[20] PC MAGAZINE HONORS BRIGHTMAIL ANTI-SPAM 5.1 WITH EDITOR'S CHOICE AWARD FOR BEST ENTERPRISE CLASS ANTI-SPAM SOFTWARE.
<http://www.brightmail.com/pressreleases/102803_pcmag.html>
(cited 7 November 2003)

[21] Brightmail; Executive Team.
<http://www.brightmail.com/about_us-exec_team.html>
(cited 12 November 2003)

[22] Brian Burton. Bayesian Spam Filtering Tweaks.
<http://spamprobe.sourceforge.net/paper.html>
(cited 7 November 2003)