MITSUBISHI ELECTRIC RESEARCH LABORATORIES
http://www.merl.com

The Spam-Filtering Accuracy Plateau at 99.9% Accuracy and How to Get Past It

Yerazunis, W.S.

TR2004-091 January 2004

Abstract

Bayesian filters have now become the standard for spam filtering; unfortunately most Bayesian filters seem to reach a plateau of accuracy at 99.9 percent. We experimentally compare the training methods TEFT, TOE, and TUNE, as well as pure Bayesian, token-bag, token-sequence, SBPH, and Markovian ddiscriminators. The results deomonstrate that TUNE is indeed best for training, but computationally exorbitant, and that Markovian discrimination is considerably more accurate than Bayesian, but not sufficient to reach four-nines accuracy, and that other techniques such as inoculation are needed.

MIT Spam Conference 2004

The Spam-Filtering Accuracy Plateau at 99.9% Accuracy and How to Get Past It.

William S. Yerazunis, PhD*

Presented at the 2004 MIT Spam Conference
January 18, 2004
MIT, Cambridge, Massachusetts

Abstract: Bayesian filters have now become the standard for spam filtering; unfortunately most Bayesian filters seem to reach a plateau of accuracy at 99.9%. We experimentally compare the training methods TEFT, TOE, and TUNE, as well as pure Bayesian, token-bag, token-sequence, SBPH, and Markovian discriminators. The results demonstrate that TUNE is indeed best for training, but computationally exorbitant, and that Markovian discrimination is considerably more accurate than Bayesian, but not sufficient to reach four-nines accuracy, and that other techniques such as inoculation are needed.

Keywords: spam, Bayesian, filter, email, e-mail, Markovian,

Introduction: The original heuristic spamfilters such as SpamAssassin and Brightmail are weighted filters whose features are generated by humans working in a reactive way. These filters use a fixed set of feature detectors, and set a threshold score for rejection of an email as "too spammy". The first generation of these filters also had a human-generated weight set; now most heuristic filters use a neural net or other relaxation-based system to optimize the feature weighting. For example, SpamAssassin (http://www.spamassassin.org) uses a set of over three hundred hand-coded Perl routines as feature recognizers.

In the recent past, these heuristic filters been eclipsed by the Bayesian antispam filters. Paul Graham described these filters initially in **A Plan For Spam [i].** These Bayesian filters have essentially dominated spam filtering, because of their significantly greater accuracy and greatly increased difficulty of subversion. Instead of using human-coded ad-hoc "feature recognizers", a Bayesian filter is trained on a pair of text corpi, one of spam, and one of nonspam. The Bayesian filter counts the word frequencies in each of the spam and nonspam corpi, and from that ratio can determine a local probability on any incoming text given only that it contains a particular word.

In any case, the Bayesian filter's first big advantage is already evident- there is no human intervention required to generate the feature recognizers. A simple whitespace delimiter detector can break the incoming text into words, and each word is considered a feature in the database.

Some Bayesian filters incorporate a database clipping function that extracts only a few hundred or thousand of the most spammy or non-spammy words from the database for testing during the execution phase. Most Bayesian filters also use a testing clip, in that only the top ten or twenty words in the spammy or nonspammy category are considered significant.

Whether or not the Bayesian filter does database or test clipping, the next phase is to apply a statistical combining function to the local probabilities of each of the word features. Typically the local probability is something of the form:

$$P_{\text{local-spam}} = N_{\text{spam}} / (N_{\text{spam}} + N_{\text{nonspam}})$$

with appropriate bounds testing to prevent divide-by-zero errors. A more advanced local probability formula takes into account that a local probability certainty should be shaded toward uncertainty (P=0.5), especially when there are low experience counts for that feature in the example texts. For example:

$$P_{\text{local-spam}} = 0.5 + \frac{(N_{\text{spam}} - N_{\text{nonspam}})}{C_1 *(N_{\text{spam}} + N_{\text{nonspam}} + C_2)}$$

which biases the local probability estimates strongly toward 0.5 for low numbers of event counts. For example, if C1 = 2 and C2 = 1, and if there is only one example of a word being spammy, the local probability of an incoming text containing that word being spam is considered to be 0.75, which is only a moderate likelihood of spam If there had been ten examples of the word in spam, and none in nonspam, the local probability of the incoming text found to contain that word being spam would be 95.4%, a reasonably high likelihood for spam.

Each word feature generates one such local probability. These local probabilities are then used in a Bayesian chain rule to calculate an overall probability that an incoming text is spam.   As noted above, some Bayesian filters will do database or testing clipping, to minimize the total computation. The Bayesian chain rule is:

$$P\ (\text{in class} \mid \text{feature}) = \frac{P(\text{feat} \mid \text{in class}) * P\ (\text{in class})}{P(\text{feat} \mid \text{class}) * P\ (\text{in class}) + P(\text{feat} \mid \text{not in class}) * P\ (\text{not in class})}$$

which is aplied iteratively to chain each local probability feature into an overall probability for the text in question.

One failing of the Bayesian chain rule is that strictly speaking it is only valid in the case of all features being statistically independent. This is emphatically not the case in text analysis; the words are not chosen randomly but with a very significant correllation. What is remarkable about Bayesian text classifiers is that the Bayesian classifiers work so well even with this gaping fundamental error.

To avoid the error of presumed decorrellation, is possible to use a chi-squared or other combining rules. SpamBayes (http://www.spambayes.org) uses a modified Chi-squared combining rule.
The overall success of Bayesian filters is visible in two ways- first, the large number

of freely available filters. At last check, over twenty different Bayesian-style spamfilters were available on Sourceforge.net. Almost all of these filters quoted demonstrable accuracies on the order of 99.9% accuracy. By comparison, a human is only about 99.84% accurate in filtering spam and nonspam (Yerazunis 2003), so any of these filters is more effective than a human “correspondence secretary”.

The extreme effectiveness of Bayesian filters has not been lost on the one group whose livelihood is most affected by the filters: the spammers themselves. While spammers used to evade simple checksumming filters by adding a few randomly chosen words or gibberish strings to the text and subject, spammers now typically use subterfuges such as:

- “Long Story” spams, where the first page or two of a spam seems to be actual innocent email.
- “Dictionary Salad” spams, where large sections of the text are randomly chosen dictionary words. The spammers apparently are hoping to drown their signal in the random dictionary noise.
- “News Story” spam, where the initial text is copied verbatim from a news report of a current popular news event completely unrelated to the spam “payload”.
- “Habeas Haiku” spams. The Habeas Haiku is a short poem whose copyright is owned by Habeas Inc, a group of lawyers. Initially use of the Habeas Haiku was to be under license only to companies agreeing to obey Habeas' code of email conduct, but the haiku has now been co-opted by spammers to the extent that, statistically speaking, the presence of the haiku is a strong statistical predictor of spam. This switchover from “nonspam” to “spam” indication occurred on one test stream in less than one week of time in early December 2003.

Spammers have also begun another very interesting tactic- penetration of well-credentialled and widely-distributed mailing lists, such as alumni and special-interest mailing lists. These well-credentialled lists are often whitelisted by their subscribers and hence the spam is delivered without significant filtering.

Comparison of Methods and Variations of Bayesian Antispam Filtering

A consistent comparison experiment for different filtering techniques needs to operate against a consistent incoming data stream. Because of the significant variation in spams from day to day, for our experimentation we used the April 2003 standard testing corpus available from the SpamAssassin web site (available at http://spamassassin.org/publiccorpus/ * <added 27-Feb-2004>).

This test set contains 4,147 messages, pre-classified into spam, easy nonspam, and hard nonspam.   Of the messages, 1400 are spam, and the remainder are nonspam. This test is a severe torture test of filtering- the author can rarely score 90% accuracy on this torture test, while in real life the author can achieve about 99.84% accuracy.

To provide a larger test set, the set of 4147 messages was shuffled ten times, with each shuffle providing an effectively random sequencing of the messages.   Each of the ten shuffles was preserved so that every experimental configuration would get the same ten sequences of the messages.

Except as noted, the last 500 messages of each shuffle were “reserved” as the final evaluation testing set, and errors were summed, so each experimental configuration had 5,000 test attempts. Thus, our raw output statistics are in errors per 5,000 in the test.

Just as in real life, the testing procedure ran for the complete set of 4147 messages, and training was performed as specified even during the final 500 (evaluation) messages. After a complete run of 4147 messages were done, the learning databases were deleted so the next run would not only execute against a different shuffle, but also without any a priori training.

Training Methods Comparison

The first experimental set was to determine the relative merits of various training methods. As we have considerable experience with Sparse Binary Polynomial Hash features and a Bayesian Chain Rule, we set up a SBPH/BCR classifier and ran it with three different training methods:

- **TEFT** - Train Every Thing – for each member of the shuffle set, classify the text, record the output (correct or incorrect), and then force train that piece of text into the database in the correct category.
- **TOE** – Train Only Errors – for each member of the shuffle set, classify the text, record the output (correct or incorrect), and if the text was incorrectly classified, train that text into the database in the correct category.
- **TUNE** – Train Until No Errors – for the first N-500 messages, repeatedly classify and train the messages if incorrect. After this intensive training test and record the final 500 texts, doing training if an error occurred. (Because there is no gaurantee of convergence, training was terminated if only a few messages in the training set were not trained. Of all 42470 messages in the ten training sections, only three were not correctly classified.)

The results are summarized below in Table 1. The execution processor was a Transmeta 666 with Red Hat Linux 7.3 and 128 megabytes of memory. The feature hashtable size was limited to 1,000,000 slots, and slots were randomly decremented when there were no unused slots available.

Training Method	**Errors per 5000**	**Approximate time for all 10 shuffles**
TEFT	149	6 hrs
TOE	69	3 hrs
TUNE	54	20 hrs

Table 1 Comparison of TEFT, TOE, and TUNE training methods

Note that TOE training is almost as good as TUNE training, with only a small fraction of the CPU time expended.

Even though TUNE is somewhat more accurate than TOE training, TUNE requires keeping the entire prior history of both spam and nonspam. In a testing situation with repeated execution, this is acceptable, but in a real-life deployment this would require a huge amount of storage per user. A rough estimate might be as much as 250 megabytes per year per user, even if duplicate spams are dropped from the database.

The additional CPU time needed for TUNE processing is nontrivial. With a back-catalog of 4147 messages, it took approximately two hours to re-classify and retrain sort; this process would need to be repeated whenever any training occurred. For an individual user with a dedicated CPU, this is acceptable, but even a small ISP (with, say, 1000 accounts) could not deploy such a system.

Comparison of Extensions to Bayesian Filtering

Taking the results of table 1 to indicate that TOE (train only errors) training is acceptable in performance and accuracy, we can now consider the relative accuracies of different extensions of a Bayesian antispam filter.

For these experiments, we again use the 4147 corpus messages, in the same ten shuffles, clearing out any learned memory between each shuffle. Each extension to Bayesian filtering will see the same ten shuffles, and each will be trained with in the TOE method. Each method will be allowed at most 1,000,000 feature slots, and the same random deletion of old slots to make room for new data was activated, however none of these extensions needed to purge older data.

The different variations tested were:

- Standard Bayesian - each word is a feature. This is the method proposed by Graham and used in most antispam filters. There was no clipping in either the database or the text; every feature was retained and used in the evaluation.
- Token Grab Bag – A sliding window of five words is moved across the input text. All combinations of those five words (but requiring the first word in the window) are taken in an order-insensitive way. Every such order-insensitive combination is a feature.
- Token Sequence Sensitive – A sliding window of five words is moved across the input text. All combinations of word deletions are applied (except that the first word in the window is never deleted) and the resulting sequence-sensitive set of words is used as a feature.
- Sparse Binary Polynomial Hashing with Bayesian Chain Rule (SBPH/BCR) – As described in Yerazunis [ii]a sliding window of five words is moved across the input. A feature is the sequence-and-spacing-sensitive set of all possible combinations of those five words, except that the first word in the window is always included.
- Peaking Sparse Binary Polynomial Hashing – this is similar to SBPH/BCR, except that for each window position, only the feature with the most extreme probability (furthest from 0.5) is used. The other features generated at that window position are disregarded. This is an attempt to 'decouple" the sequences of words in a text and make the Bayesian chain rule more appropriate.
- Markovian matching – This is similar to Sparse Binary Polynomial Hashing but the individual features are given variable weights. These weights are assigned in a superincreasing way, so that a feature that contains more words than any of it's subfeatures can outweigh all of it's sub-features combined.

These tests were run on the same system as the training tests. Run times did not differ substantially from TOE-trained SBPH/BCR in table 1;

Evaluator	**Errors per 5000**
Pure Bayesian matching	92
Peak Window Value	80
Token Sequence Sensitive	78
Token Grab Bag	71
Sparse Binary Polynomal Hash	69
Markovian matching	56

Table 2 Accuracies of various Evaluators

Interestingly, Token Sequence Sensitive (TSS) scored as a weaker system than Token Grab Bag (TGB), by a statistically significant amount. One might assume that the extra information available to a TSS filter would give higher performance, but apparently in the face of spammer countermeasures such as random word insertion, we find that TSS does not perform as well as TGB. Interestingly, SBPH and TGB are very close in performance.

The best performance of all of the filters was found to be the Markovian matching filter, so some further details are in order.

The Markovian matching filter does not attempt to generate the actual Markov model of the incoming texts, for computational tractability. Instead, it uses a hashed representation of the known example texts in each corpus as local sections of the large (and mostly unknown) Markov models of spam and nonspam.

The implementation of this is quite simple. Instead of chaining tokens (as described in Zdziarski [iii], the features are weighted to give greatly increased credence to evidence found in longer features.

For example, in the text “The quick brown fox jumped”, the features and their associated weights would be

Feature	Weight
The	1
The quick	4
The <skip> brown	4
The quick brown	16
The <skip> <skip> fox	4
The quick <skip> fox	16
The <skip> brown fox	16
The quick brown fox	64
The <skip> <skip> <skip> jumped	4
The quick <skip> <skip> jumped	16
The <skip> brown <skip> jumped	16
The quick brown <skip> jumped	64
The <skip> <skip> fox jumped	16
The quick <skip> fox jumped	64
The <skip> brown fox jumped	64
The quick brown fox jumped	256

Table 3 Example Features and Weights for Markovian matching

In this experiment, we used superincreasing weights as determined by the formula

$$\text{Weight} = 2^{2N}$$

Thus, for features containing 1, 2, 3, 4, and 5 words, the weights of those features would be 1, 4, 16, 64, and 256 respectively. These weights are used to bias short feature local probabilities toward 0.5 such as in

$$P_{local\text{-}spam} = 0.5 + \frac{(N_{spam} - N_{nonspam}) * \text{Weight}}{C_1 *(N_{spam}+N_{nonspam} + C_2) * \text{WeightMax}}$$

A Theoretical Justification for the Observed Markovian Accuracy

One might ask why a Markovian filter would be significantly more accurate than the very closely related Sparse Binary Polynomial Hash (SBPH) filter. One hypothesis is that the SBPH filter is still a linear filter, while a Markovian filter with superincreasing weights is nonlinear. One can envision the filtering domain of SPBH in a high-dimensional hyperspace (one hyperspatial dimension per feature). However, the partition between spam and nonspam is still a flat plane in this hyperspace; texts on one side of the plane are judged to be spam, those on the other side are judged to be nonspam.

Minsky and Papert [iv]showed that such a linear filter cannot implement anything even as complex as a mathematical XOR operation. That is, linear filters cannot learn (or even express!) anything as complex as "A or B but NOT BOTH". The classic Bayesian spam filter is still a linear filter; as an example, a Bayesian cannot be trained to allow "viagra" or "pharmacy" but not both.

The Markovian filter with superincreasing weights is no longer a linear filter; because any feature of length N can override all features of length 1, 2, ... N-1 a Markovian filter can easily learn situations where A or B but NOT BOTH is the correct partitioning. As implemented with a sliding window of length 5, the Markovian partitions the feature hyperspace along a quintic (fifth-order) curved surface, which may not even be fully connected.

This theoretical justification is, unfortunately, not proven. It may be some other issue of implementation that causes a Markovian filter to have significantly better accuracy in these tests than a Bayesian. Therefore, the matter should be considered "likely", but by no means "closed" in a scientific sense.

Also, we should note that the accuracy advantage of a Markovian over a Bayesian is significant- the Markovian made about 40% fewer errors on the same data streams. But a 40% improvement in filtering accuracy is only a few months respite at the current month-to-month increase rate of increase in spam. Even a Markovian is no defense against the spam attack where a spammer joins a well-credentialed list; this attack is becoming more and more common.

Inoculation and Minefielding

The next generations of spam filtering can take advantage of the DeSade observation- that is, that one man's pain is another man's pleasure. In this case, we use the pain of one person receiving a spam to train not only their own filters, but also the filters of a number of friends. In this situation, the error rate of the system for discriminatable spam goes with the inverse of the number of subscribers; with ten friends participating, you achieve 10x improvement in filtering. However, this is still a human-mediated training and hence subject to human error.

Dispite this, we have programmed such inoculation-based systems and preliminary results are that they do function well.

A second observation is a site-wide rather than a per-user observation. If one observes the behavior of most spam campaigns, the same spammer will spam all of the accounts known to it on a given site in a very short period of time. In one case, this was every account on a small site in less than ten seconds.

Because of this rapid propagation of the spam, human reaction time is too slow- by the time the first human reads the spam, all of the mailbox filters will have already either accepted or refused the spam.

To counter this style of attack, one can add "email minefield" defenses. An email minefield is constructed by adding a large set of dummy email addresses into the address space of a site. These email addresses are then intentionally leaked to spammers.
Since no human would send email to those extra addresses, any email to those addresses is known a-priori to be spam.

More usefully, spammers usually attempt to falsify their headers and hide their IP addresses. However, during the SMTP transaction from the spammer to the minefield address, the spammer must reveal their actual IP address. The spammer cannot spoof this address, as the SMTP transaction depends on at least the RCPT OK section of the transaction being delivered correctly to the spammer, and that can only happen if the spammer reveals a correct IP address during the socket setup phase.

At this point, the targeted site and any site cooperating with the targeted site can immediately blacklist the offending IP address. This blacklist can be either a “recieve and discard”, or “refuse connection” situation.

Unfortunately, John Graham-Cumming [v] points out, it is possible to use a Bayesian filter against a Bayesian filter. In this process, the “evil bayesian” filter is supplied with a large dictionary of words; it repeatedly sends spam with included random words to the “good” bayesian filter and recieves back negative acknowledgements when the “good” bayesian filter correctly detects the spam. Graham-Cumming reports that the “evil bayesian” will converge on the set of random dictionary words most likely to thwart the “good” bayesian's filtering ability and allow spam to penetrate.

This result is extendable to any filtering algorithm; Graham-Cumming points out that the only true defense for it is to never return any feedback to the sender. Thus, our minefielding system must also perform in the same way to avoid disclosing which accounts are minefield accounts and which are real humans- incoming mail to either kind of account must be accepted and only afterwards deleted if it is known to come from a spamming IP address.

Fortunately, the same rapidity of communication that allows a spammer to hit all of the accounts on a system also allows one system to even more rapidly communicate the source IP address to other systems. In this kind of shared realtime minefield, multiple cooperating sites dynamically blacklist individual IP addresses for short periods of time (a few hours to days) in response to spamming activity to a minefield account.

Further, because the dynamic minefield spans multiple sites for both sensing a spam attack and transmitting blacklist IP addresses, a very large number of accounts can be protected relatively easily and without human intervention to either add IP addresses to the blacklist or remove them from the blacklist.

Dynamic minefielding in this style is currently in testing and is a subject for future work.

Conclusions:

Bayesian filtering may have reached a limit of accuracy; enhancements may be useful but the amount of information in a particular email is limited and an ever-increasing quality of filtering may be impossible.

Fortunately, correllating mail from multiple accounts, either with or without human intervention, will provide a significantly larger source of information, and a source of information with a significantly higher limit.

Thanks and Credits

The author would like to thank all of the members of the CRM114 development mailing list in general, as well as Darren Leigh and Erik Piip,

[i] Graham, Paul, “A Plan For Spam”, 20**03, http://www.paulgraham.com/spam.html)**

[ii] Yerazunis, William S., “Sparse Binary Polynomial Hashing and the CRM114 Discriminator”, MIT Spam Conference 2003, available from http://crm114.sourceforge.net

[iii] Zdzairsky, Jonathan, "Advanced Language Classification using Chained Tokens", MIT Spam Conference 2004

[iv] Minsky and Papert, 1969, PERCEPTRONS

[v] John Graham-Cumming, "How to Beat a Bayesian Spam Filter", MIT Spam Conference 2004