SpamBayes: Effective open-source, Bayesian based, email classification system.

T.A Meyer and B Whateley

Meyer:- IIMS, Massey University, Auckland, New Zealand
T.A.Meyer@massey.ac.nz
Whateley:- Dark Indigo, Inc., Fremont, California, USA
brendon@darkindigo.com

Abstract. This paper introduces the SpamBayes classification engine and outlines the most important features and techniques which contribute to its success. The importance of using the indeterminate 'unsure' classification produced by the chi-squared combining technique is explained. It outlines a Robinson/Woodhead/Peters technique of 'tiling' unigrams and bigrams to produce better results than relying solely on either or other methods of using both unigrams and bigrams. It discusses methods of training the classifier, and evaluates the success of different methods. The paper focuses on highlighting techniques that might aid other classification systems rather than attempting to demonstrate the effectiveness of the SpamBayes classification engine.

1. SpamBayes

SpamBayes [1] was born on August 19th 2002, soon after publication of *A Plan for Spam* [2]; Tim Peters and others involved with the Python development community developed code based on Graham's ideas, with the initial aim of filtering python.org mailing list traffic, although this quickly progressed to also filtering personal email streams. SpamBayes separated out from its Python sandbox birthplace to a separate project on September 4th 2002.

The project had initial success with Graham's original combining scheme – a "Naïve Bayes" scheme of sorts; however, the scheme had a number of problems, particularly selection of the 'magic' numbers required by the scheme, and a tendency to produce scores of either one (definite spam) or zero (definite ham), with only a very small middle ground in between. As a result, when the system was wrong, it was completely confident in its (incorrect) score; more recent approaches still make some classification errors, but the system is less confident in its (incorrect) score. Various techniques developed by Gary Robinson [3] alleviated these problems, including a Central Limit Theorem approach, which produced two internal scores, one for ham and one for spam. This meant it was possible to return an "I don't know" response when ham and spam scores were both very low or both very high. Although the central limit approach was dropped in favor of chi-squared combining, this 'unsure' range formed an important addition.

1.1 Applications

The SpamBayes tokenizer [4] and classifier are used by a number of separate applications included in the distributions. The most widely used is a plug-in for Microsoft® Outlook®, which fully integrates into the mailer. Another is a POP3 proxy with a web-based training and configuration interface, which is used to provide filtering for most other mail clients. Various command-line scripts (to integrate with procmail, for example), and an IMAP4 filter are also included.

1.2 Testing Tools

Testing is a core focus of the SpamBayes group and the source distributions include a variety of scripts to aid users in setting up and executing a variety of tests. Many users and group members have ideas which they feel would improve results; in practice, the product's development is at a point at which almost all the ideas have either no effect or actually harm results. This focus on evaluating the effectiveness of ideas before changing the product has helped to anecdotally position SpamBayes as one of the most effective spam products available. The most commonly used test is a simple cross-validation script: the ham and spam corpora are randomly

divided into n sets; each set n is filtered against a classifier trained with sets 1 to n-1, and this is repeated until all sets have been filtered.

While cross-validation testing provides information that is easy to interpret, it does not imitate the way that the classifier is typically used and trained in practice. The cross-validation test imitates a user who carries out a large amount of initial training (and trains on every message), and then does not do any further training. In practice, filtering and training is an iterative process that continues throughout the use of the filter and some messages may not be used for training. SpamBayes provides a script that imitates this form of 'incremental' training. Given ham and spam corpora, sorted by date, and a 'training regime', as described below, the script simulates arrival of each message, performs the appropriate training, records the results, and moves on to the next (in chronological order) message. This form of testing produces more data, so is somewhat harder to interpret, but matches much more closely the use to which the filter is put in practice.

To determine which messages to train on, the tester provides the script with a 'training regime'. This is a function which, given both the estimated and correct classifications of the message, instructs the script whether to train the message as ham, train as spam, or to perform no training with the message. The simplest training regime is to train on all messages with their correct classification; variations include 'training on mistakes' (all false positives, false negatives, and messages classified as unsure), 'non-edge training' (training all messages within a certain score range, such as 0.05 and 0.95), and 'training to exhaustion' [5] (similar to the perceptron algorithm [6]).

1.3 Testing Methodology

The testing outlined in this paper covers five corpora (see Table 1); four mail streams from SpamBayes users [7] (including all messages received during that time, manually classified into ham and spam by the user) and the five 2003-02-28 SpamAssassin Public Corpus sets [8]. Testing was carried out using SpamBayes code from CVS as of the 1st of April 2004, and, except where specified, uses the default settings for all options. Graphs from the incremental testing setup have been simplified into a "weighted error" score where the average false positive percentage, false negative percentage and unsure percentage for each day are combined, multiplying false positive percentages by ten and unsure percentages by 0.2 (these are the default weights used to calculate the SpamBayes 'real cost' and 'best cost' values).

Corpus	# Ham	# Spam	Ham::Spam Ratio	Date Range
A	499	1785	0.28::1	2004-01-08 to 2004-04-01
B	23454	1923	12.20::1	2004-01-08 to 2004-04-01
C	1391	643	2.16::1	2004-01-08 to 2004-04-01
D	21750	30312	0.72::1	2003-11-13 to 2004-03-19
SpamAssassin Public Corpus	4150	1897	2.19::1	2001-06-26 to 2002-12-04

Table 1. Breakdown of testing corpora.

The SpamAssassin Public Corpus suits cross-validation testing well; however, the uneven clustering of spam and ham messages in the corpus, when taken chronologically, skews results obtained using the incremental testing script. As a result, the SpamAssassin Public Corpus is used only with cross-validation testing in the results outlined in this paper.

2. Chi-squared combining

After his Central Limit Theorem ideas, Robinson suggested a novel, but well-founded, combining scheme using chi-squared probabilities. A chi-squared test calculates the probability that a particular distribution matches a hypothesis (in this case that the message is spam and, separately, that the message is ham). The results of these

two chi-squared tests are then combined and scaled to give an overall message spam score in the range 0 to 1. The 'unsure' middle ground is defined as any message with a final score falling between an upper and lower bound. The default unsure range is a message with a final combined spam score between 0.20 and 0.90; this lack of symmetry reflects an aversion to false positives. Some end cases were improved when Rob Hooft discovered a cleaner way[1] to combine the internal ham and spam scores, which improved detection of the 'middle ground'. The key to this process is that messages tend to score at the extremes of the range, but difficult to classify messages fall nearer the middle.

A remarkable property of chi-combining is that people have generally been sympathetic to its 'unsure' ratings: people usually agree that messages classed unsure really are hard to categorize. For example, commercial HTML email from a company you do business with is quite likely to score as unsure the first time the system sees such a message from a particular company. Spam and commercial email both use the language and devices of advertising heavily, so it is hard to tell them apart. Training quickly teaches the system how to identify desired commercial email by picking up clues, ranging from which company sent it and how they addressed you, to the kinds of products and services it offers.

2.1 The importance of the 'unsure' range

One of the key features that contributes to user acceptance of SpamBayes is the method of dividing incoming email into three groups: Ham (good mail), Unsure (can't be sure of classification) and Spam. For example, in practice, the Corpus D mail stream resulted in zero ham messages classified as spam, 0.36% spam classified as ham and 1% of total email classified as unsure.[2] Table 2 gives a clear indication of what this really means:

Classification	Actual Ham	Ham as Spam	Actual Spam	Spam as Ham	Unsure
Percentage of Mail	41.75%	0.00%	58.25%	0.21%	1.00%
Num Messages	21750	0	30312	109	521

Table 2. Classification results for Corpus D

As can be seen with the real life results from this corpus, most users experience such a low false positive rate that they have no need to check messages classified as spam. This reduces the practical "spam workload", defined as percentage of messages needing manual checking, to just 1.21% of the total mail stream. Any system that exhibits a non-trivial false positive rate requires the user to check all messages classified as spam to ensure that valuable mail is not lost, dramatically reducing the value of the spam filtering technology. This huge improvement in user experience is the direct result of using the unsure classification instead of the more typical "certainty but with errors" approach. SpamBayes allows the user to configure the size and position of the unsure range to ensure the number of messages classified as unsure is consistent with the user's comfort level, training database and risk tolerance of false positives.

3. Tokenizing

Tokenizing (i.e. converting the message string to a feature vector) the message has a profound influence on the overall results of the classification engine [8]. During the development history of SpamBayes, many ideas for message tokenization have been suggested and tried. Most schemes have provided no statistically significant over diverse corpora. SpamBayes allows these ideas to be included as experimental features so that developers

---

[1] The classifier originally determined the final score by dividing the internal spam score by the sum of the internal ham and spam scores. The classifier now uses the internal spam score minus the internal ham score, scaled into [0., 1.]

[2] These numbers are the result of the author extrapolating the performance of his real SpamBayes system (which was the basis of Corpus D) to the actual Corpus D. The difference is that these numbers reflect the actual manual training performed by the author, complete with mistakes, instead of the perfect classification of the Corpus. In other words, the Corpus has the author's real life training mistakes corrected.

and adventurous users can test their effectiveness on their individual corpora. It is important to note that beneficial tokenizing schemes change over time as the nature of spam changes. In other words, both accepted and rejected techniques have to be periodically validated.

Messages are split into several types of tokens; header tokens, body tokens and synthesized tokens.[3] Header tokens are the set of tokens that identify themselves as part of the message header. This allows the classifier to benefit from the difference in significance between "words" occurring in a header and the same source "word" occurring in the message body. In addition, this enables the system to ignore certain header fields that do not provide significant clues and synthesize tokens for significant header information. This can include tokens exploiting header features such as large numbers of recipients, the same recipient user name in multiple different domains, sender claiming to be local, and so on. Tokens that score outside the central 0.4 to 0.6 range are used to score the message. For any message, not more than the 150 most significant tokens are used for scoring.

Body tokens are mostly generated by simply splitting the message body text on white space, but with significant handling being given to words that are longer than a particular threshold, octet streams, uuencoded content, base64 content, URLs and HTML content. Examples include stripping HTML tags, "fixing" URLs that attempt to hide the true destination, and tokenization of email addresses.

Synthesized tokens are ones generated by the tokenizer to provide the classifier with clues that are not directly taken from the message. An example is generating a token to indicate that a message has no subject, no from address, and so on. Special tokens are also generated for "words" that are longer than a set threshold, substituting a "skip:n" token that indicates how big skipped the word was. Some of the synthesized tokens add clues that would otherwise be discarded by the tokenizer; others are an attempt to limit the database size by not introducing excessive numbers of tokens that add very little value.

3.1 Unigrams/Bigrams scheme

Early testing [9] showed that using either character or word n-grams was less effective than simple split-on-white-space unigrams (see Figure 1). However, in late 2002, Robinson and Robert Woodhead independently came up with an idea for using both unigrams and bigrams with a twist to avoid generating highly correlated clues. This idea was cleaned up and implemented by Peters, although it was only added to the SpamBayes code in late 2003. This code is currently in daily use by a number of the SpamBayes group, and is available to users as an experimental option.

This technique mixes single tokens (unigrams) with pairs of adjacent tokens (bigrams); the token stream is 'tiled' into non-overlapping unigrams and bigrams using the strongest tokens. Avoiding overlap is important to prevent a single token from contribution to more than one token probability returned (systematic correlation). Each triplet of tokens in the message generates three unigrams and two bigrams – these are ranked in order of the distance of their score from the neutral 0.5. The strongest is added to the list of tokens used to classify the message, and the remaining tokens that overlap that one are removed, and the process is then repeated with the next triplet of tokens.

---

[3] Interested readers should refer to the SpamBayes source file tokenizer.py [4] for comprehensive treatment of techniques and history.
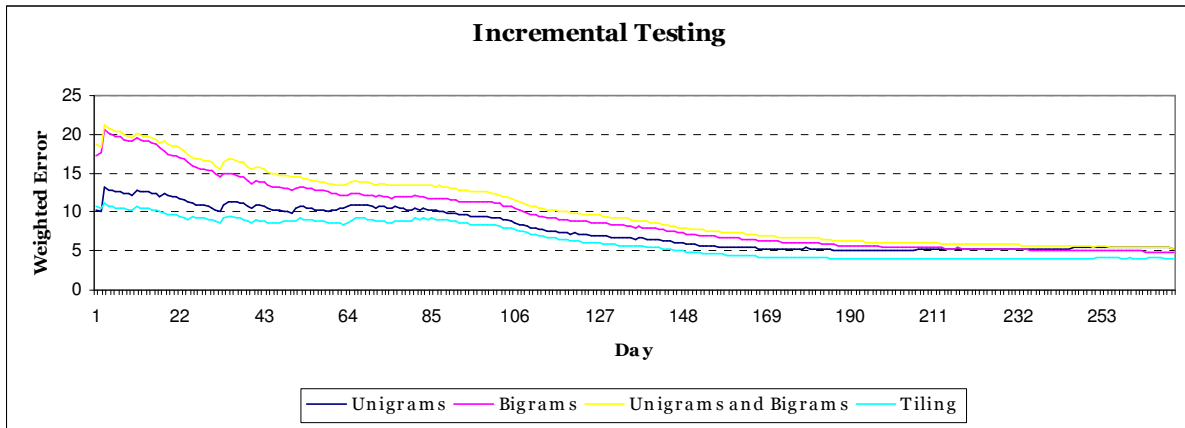
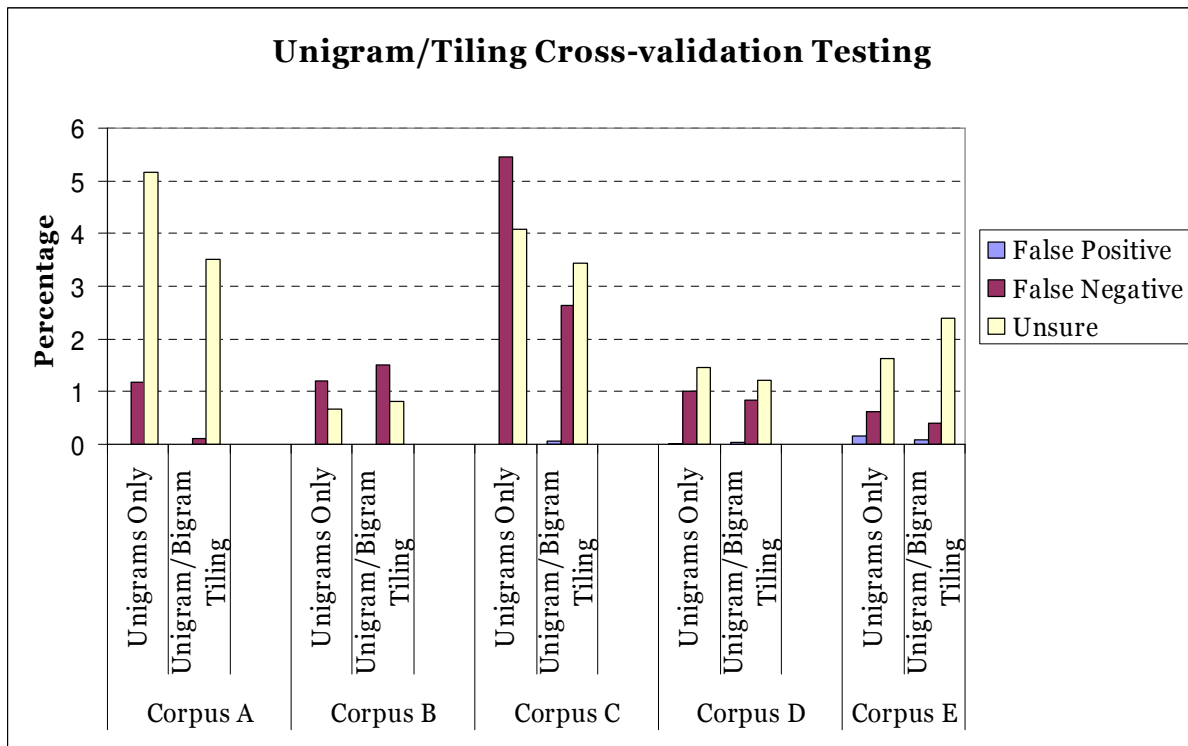**Fig. 1.** Comparison of unigrams, bigrams, unigrams and bigrams, and the tiling technique.

Fig. 2. Cross-validation results

For example, given the phrase "now purchase this" and the scores in Table 3, the strongest token is "purchase this", and so would be used for classification. The second strongest is "now purchase", but is unable to be used as it overlaps with "purchase this", as do the two next strongest, "purchase" and "this". The remaining token, "now", does not overlap, and so is used for classification. Obviously in normal use, the two outlying unigrams would also form part of another bigram, and so may or may not be used.

With cross validation testing, all but one of the corpora benefit from the unigrams/bigrams tiling technique (see Figure 2). Corpora A and C in particular benefit from the technique, while with the SpamAssassin Public Archive the main gain is movement from false positives and negatives to the unsure range. The technique does appear more reliant on a reasonably balanced corpus; Corpus B is much more imbalanced (a ham to spam ratio

over 12::1) than the other corpora, and is the only corpus that doesn't gain from the technique. Experiments using random, evenly balanced, selections of Corpus B ham and Corpus B spam all resulted in a net benefit from the tiling technique.

Word	Now	purchase	This	now purchase	purchase this
Score	0.49	0.43	0.47	0.16	0.14
Distance from 0.5	0.07	0.01	0.03	0.34	0.36

Table 3. Example unigram and bigram scores.

A penalty for using the unigrams/bigrams technique is that the token database size typically increases by a multiple of between four and eight; however, given that the technique tends to learn more rapidly, the total number of messages trained is often lower, and so the resulting database is only two to four times larger. Since disk space is relatively inexpensive, this is not prohibitive given that SpamBayes is typically used for client-side filtering rather than server-side. There is also an additional speed penalty (both as a result of the larger database and to generate the bigrams and select which unigrams and bigrams to use); the effect is not noticeable in practice, although this, too, might differ for a server-side process.

4. Training

With any classification system, the selection of training data has a significant effect on the reliability of the system. The naïve approach to training a spam filter like SpamBayes is to simply train on all incoming (filtered) mail; however, testing shows this training regime to be one of the least effective (see Figure 4). A common alternative, and one typically adopted by SpamBayes users, is mistake-based training. Essentially, this involves training only on messages that were incorrectly classified (usually including messages in the 'unsure' range). This regime results in a smaller database, and also offers results either superior, or nearly so, to any other training regime (see Figure 4).
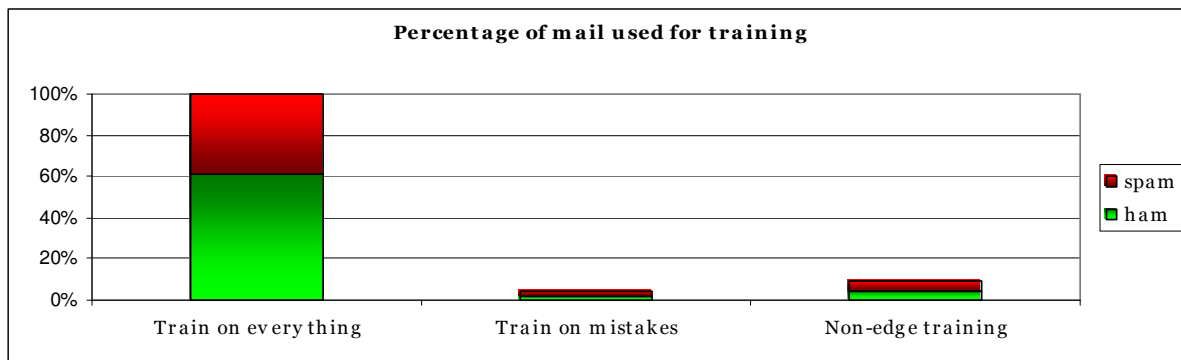

Fig. 3. Percentage of mail used for training (averaged across corpora)

As a result of the chi-squared combining method, the majority of messages are at the edges of the score range (0 to 0.05 and 0.95 to 1, for example). The 'non-edge' training regime uses this feature to train on all messages 'inside the edges' (e.g. 0.05 to 0.95), reasoning that the edge messages, already successfully classified, have a lesser amount of valuable training data. This is similar to the concept of Support Vector Machines [10], except that no attempt is made to automatically determine the ideal boundaries. Like mistake-based training, 'non-edge' training results in a greatly reduced database size (see Figure 3), and is often the most effective training regime (see Figure 5).

4.1 Training regimes and unigram/bigram tiling

Interestingly, the unigram/bigram tiling technique produces less appealing results when tested with the incremental setup than with cross-validation (see section 3.1). Using the tiling technique slightly improves results when training on everything or using mistake based training, but (with all corpora) causes the 'non-edge' training results to decrease in accuracy. Surprisingly, given the cross-validation results, the most accurate incremental testing results can be achieved using the 'non-edge' regime, without the unigram/bigram tiling technique. In the future, investigation will be carried out into the cause of this, and whether a training regime can be found that suits unigram/bigram tiling well enough to result in higher accuracy than 'non-edge' training.
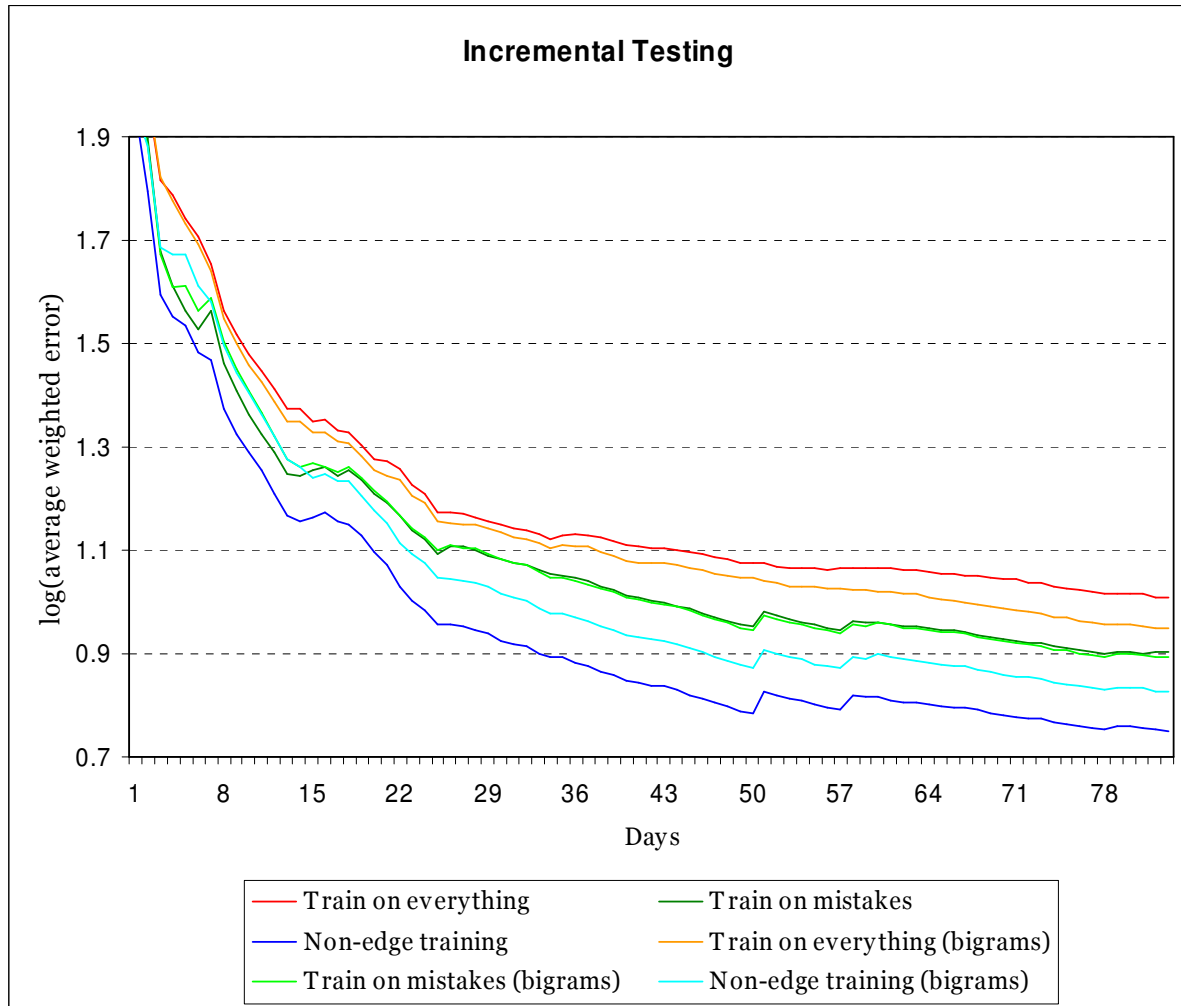

Fig. 4. Comparison of training regimes with unigram/bigram tiling (averaging results from all corpora).

5. Conclusion

SpamBayes has achieved high praise from its users due to a combination of factors. The use of a message scoring system that provides an unsure classification greatly reduces the users' exposure to spam messages by almost completely eliminating false positives. The tokenizer has been developed to exploit all clues that are

found to improve results in a statistically significant way. The development group continuously evaluates new techniques, such as the n-gram tiling to seek out performance improvements and to keep the product ahead of the ever-changing attacks that are being used by spammers.

Finally, all ideas that make their way into the SpamBayes product have to survive critical examination by many developers and have proven their value in rigorous testing against a diverse set of corpora. Only after proving significant value for a broad spectrum of users, does a new feature get to move out of the experimental category. This provides an extremely valuable repository for anti-spam techniques, a resource that other products could benefit from.

Acknowledgements

References

[1]   SpamBayes-Development-Team, "SpamBayes: Bayesian anti-spam classifier written in Python", [online] 2003, http://spambayes.org (Accessed: 10 October 2003)
[2]   P. Graham, "A Plan for Spam", [online] 2002, http://www.paulgraham.com/spam.html (Accessed: April 12 2004)
[3]   G. Robinson, "A Statistical Approach to the Spam Problem," *Linux Journal*, vol. 107, 2003.
[4]   SpamBayes-Development-Team, "tokenizer.py", [online] 2004, http://cvs.sourceforge.net/viewcvs.py/*checkout*/spambayes/spambayes/spambayes/tokenizer.py (Accessed: 1 April 2004)
[5]   G. Robinson, "Instructions for Training to Exhaustion", (Gary's Longer Rants), [online] 2004, http://garyrob.blogs.com/garys_longer_rants/2004/02/instructions_fo.html (Accessed: April 12 2004)
[6]   K. Tretyakov, "Machine Learning Techniques in Spam Filtering," *Institute of Computer Science, University of Tartu Data Mining Problem-oriented Seminar, MTAT*, vol. 3, pp. 60-79, 2004.
[7]   T. A. Meyer, "Corpora used for CEAS-04 Testing", [online] 2004, http://www.massey.ac.nz/~tameyer/research/spambayes/ceas.html (Accessed: 12 April 2004)
[8]   "SpamAssassin Public Corpus", [online] 2003, http://spamassassin.org/publiccorpus (Accessed: 13 April 2004)
[9]   T. Peters, "The first trustworthy <wink> GBayes results", (Python-Dev), [online] 2002, http://mail.python.org/pipermail/python-dev/2002-September/028513.html (Accessed: April 12 2004)
[10]   H. Drucker, D. Wu, and V.N.Vapnik, "Support vector machines for spam categorization.," *IEEE Trans. on Neural Networks*, vol. 10, pp. 1048-1054, 1999.