Comparing Content-Filter Techniques For Stopping Spam

Andrew Akehurst, Iain Phillips and Mark Withall
Department of Computer Science
Loughborough University
I.W.Phillips@lboro.ac.uk

Abstract

There are many new theoretical techniques for detecting spam e-mail based upon the message contents. Although Bayesian methods are the most well-known, there are other approaches for classifying information. This paper establishes some criteria for measuring spam filter effectiveness and compares the Boosting and Support Vector Machine approaches with some well-known existing filter software. It also examines ways of transforming e-mail messages into a form which is more readily processable by such algorithms.

1 Introduction

E-mail has been blighted in recent years by a problem known as 'spam': unsolicited junk messages filling up one's storage space, drowning out legitimate messages by their sheer volume.

The problem appears to be increasing over time. Figures from the recent UK Parliamentary All Party Internet Group report on spam estimate that there are 10 billion spam messages sent per day, which accounts for between one third and one half of all e-mail [2]. Brightmail, an e-mail filtering company, estimates there was a 900% increase in the volume of spam from April 2001 to April 2003.

Much research has been concerned with the methods to detect and eliminate spam automatically, thus saving huge amounts of human time and effort.

However, the problem has proved considerably challenging to overcome. Each time a technique is found to reduce the spam problem, the senders of spam ('spammers') discover ways to defeat the technique. The result has been a kind of arms race: a technological battle of wits between spammers and those who oppose them.

In this study possible technological methods to reduce the impact of spam are investigated.

1.1 What Are The Characteristics Of Spam?

The definition of spam as "unsolicited e-mail, usually sent in bulk" is a helpful starting point for discussion but is not specific enough for practical purposes. It is necessary to consider the properties of a typical spam message to see what kinds of features they have.

Spam messages incorporate a number of techniques to hide the real content of the message from computerised spam-idenfication systems.

Firstly, the subject line often has had some additional text appended to it. This is intended to confuse content-filtering systems which rely upon dictionary-based filtering. Such methods often assign a 'spam score' rating to each word: 'free', 'pharmacy' and 'viagra' are likely to have high positive scores, whereas more innocent every-day words will have low or even negative scores. The net effect is that the inclusion of many 'non-spam' words can artificially decrease the spam rating of the overall message, meaning that it is more likely to pass through such filters.

Secondly, there is sometimes additional concealed text within the body which is most likely designed to achieve the same purpose. This can be enclosed in an HTML FONT element which sets the text colour to be white; therefore such text will not show up against a white background and will be invisible to the reader (assuming their mail client can display HTML e-mail).

Thirdly, addition obfuscation techniques are used such as: a dummy HTML tag in the middle of the word and substituting the '@' symbol for the letter 'a'. This is of course because the word 'Viagra' alone is easily detected as probable spam.

Finally, invalid HTML tags are used to break up key words in the text. Such tags are ignored

by an HTML rendering program and are hidden from the user, but make it more difficult for automated filters to determine the true content.

It follows that a major guiding principle in detecting spam should be that, when considering whether a message is spam, it is vital to consider the way in which the message will be displayed to the recipient.

This is not to suggest that the unseen content is unnecessary in spam classification; indeed such features rarely occur in genuine messages. However, looking at the raw message content alone is often insufficient to determine its true nature. A good filtering system will be capable of stripping away unseen HTML content, content-transfer encodings and converting obfuscated character entities into the actual characters they represent, whilst taking the presence of such features into account during classification.

2 Classifying SPAM

2.1 Possible Spam Indicators

One aim of this work was to determine some attributes by which spam may be recognised and to decide which attributes tend to work most effectively.

It is unlikely that such analysis will remain unchanged in future, since the behaviour of spammers tends to adapt over time. However, the intention is to suggest some general types of attribute which will be useful and to determine some criteria which can be used to evaluate future proposed spam indicators.

The number of Spam indicators means they are too numerous to list in this paper, but typically the following are included:

- Specific words and characters in header fields or the body of an email message, e.g. free, pharmacy or accented characters
- HTML syntax embedded in words, or specific HTML tags, including Malformed HTML.
- Statistics about the words in the messages, such as mean or maximum length.
- Attachment types, especially inline images or 'octet-stream' Content-Types.

2.2 How Are Spam Indicators Used?

The problem of identifying spam is one of classification. Suppose that any e-mail message could be represented by an instance of some data type M.

In order to construct an efficient classification algorithm, it is necessary to extract the useful relevant features of a message and discard all other information. Each feature can be represented by a number (with Booleans mapped onto 0 or 1). The collection of all such feature numbers for a given message can be presented as an n dimensional vector, where n is the number of features.

Thus for every message m of type M it will be possible to find such a vector. In the general case, the elements of the vector will be real numbers and so the vector has type $\mathbb{R}^n$. Each message can be regarded as being a point in n dimensional vector space.

Effectively three things are required:

1. A feature space function $f_s : M \rightarrow \mathbb{R}^n$ which can map individual e-mail messages onto a vector of n feature values which characterise that message. In order to behave consistently, this must be strictly deterministic.

2. A learning algorithm A_L which, given some pre-classified e-mail examples, can divide the feature space into two regions: spam and non-spam. Each message must first be mapped onto its point in feature space using f_s and then that space will be partitioned.

   Input: Two finite sets of message feature vectors: one set of spam and the other of legitimate messages.

   Output: The regions of feature space which comprise spam and non-spam. This is expressed as a function (f_c) which can map a feature vector onto a class label.

   Thus $A_L : \mathcal{F}(\mathbb{R}^n \times \mathbb{B}) \rightarrow (\mathbb{R}^n \rightarrow \mathbb{B})$

3. The classification function f_c which, given some feature vector $v \in \mathbb{R}^n$, can classify m as being either spam or legitimate:

   $f_c : \mathbb{R}^n \rightarrow \mathbb{B}$

   $f_c(v) \triangleq$ "if v represents spam then True else False fi" (semi-formally)

   Here, a Boolean label is used to indicate the class. One could also map v onto the sets {0, 1} or {-1, 1}.

   Some classification algorithms also output a (positive) number whose magnitude reflects how confident the system is about the accuracy of the classification. One example of

this type of algorithm is the Boosting algorithm, which will be discussed later. In that case, $f_c : \mathbb{R}^n \rightarrow \mathbb{B} \times \mathbb{R}^+$

2.3 Classification Techniques

There is a range of techniques which can be used for categorising text into known classes. In this section, Bayesian Classification, Boosting Algorithms and Support Vector Machines will be discussed.

2.3.1 Bayesian Classification

There has been much research into Bayesian methods for e-mail classification. For example, Sahami et al. described such a method for filtering spam [6]. Since the authors compared some existing Bayesian implementations with other techniques, it is worth examining the principles behind the approach.

Paul Graham's article "A Plan For Spam" is a practical guide to spam filtering[1] and gives some interesting examples of how Bayesian methods can detect spam. For instance, consider Table 1 which lists the probabilities of individual words being spam indicators. The values represent the words that have a probality of indicating spam furthest from 0.5, for a given message.

The word "madam" appears in unsolicited messages which start "Dear Sir or Madam" whilst the term "republic" commonly occurs in so-called Nigerian scam messages. Graham cites the example of a message containing "shortest", "madam" and "promotion". Now "shortest" is a very good indicator of legitimate messages, according to Graham's own collection. However, the other two are spam-like words, so it is potentially difficult to classify such mixed content. Applying Bayes Theorem gives a probability that the message is spam of 0.9027.

Interestingly, Graham's results also show that the string "FF0000" (the HTML hexadecimal code used for red fonts) is just as good a spam indicator as the word "sexy". Of course this depends upon the training set used, but it shows that good spam indicators are not always obvious to the recipient.

2.3.2 Boosting Algorithms

Boosting algorithms are a class of algorithms which can combine separate weak classification rules into a composite whole. There is assumed to exist a "weak learning procedure" to generate hypotheses which may be only partially accurate. A boosting algorithm aims to find a set

Term	Probability
madam	0.99
promotion	0.99
republic	0.99
enter	0.91
quality	0.89
investment	0.86
valuable	0.82
very	0.14
sorry	0.08
shortest	0.05
mandatory	0.02

Table 1: Words and message spam probabilities. Adapted from A Plan For Spam

of weak hypotheses whose combined accuracy is greater than that of any individual hypothesis.

An example of this approach is that of Carreras and Màrquez [3], whose work is a special case of Schapire and Singers approach [7].

Each hypothesis $h(x)$ can be regarded as a function based on some predicate p evaluated upon an e-mail message x:

$$h(x) = \begin{cases} c_0 & \text{if } p \text{ holds in } x \\ c_1 & \text{otherwise} \end{cases}$$

Here, c_0 and c_1 are real numbers which are calculated by the training algorithm. Their derivation will be explained later.

Carreras and Màrquez describe an algorithm known as AdaBoost in which the weak learning procedure is invoked repeatedly in a series of t rounds. A weight value α_t is computed for each weak hypotheses h_t thus obtained, such that the combined hypothesis may be calculated via Equation 1. Here, x is an element of the feature space, i.e. a message to be classified. The sign of $c(x)$ gives the class label and the magnitude is regarded as the level of confidence in the prediction. Similarly each h_t has the same properties.

$$c(x) = \sum_{i=1}^{t} \alpha_i h_i(x) \tag{1}$$

Training occurs using a set S of pre-classified messages, of which there are m in total.

S can be regarded as a set of pairs in the form (x_i, y_i) where x_i is the message itself and $y_i \in \{-1, +1\}$ is the corresponding class label. Alternatively it could be represented as a pair

1. `http://www.paulgraham.com/spam.html`

of disjoint sets in line with the definition of A_L given earlier. This design choice does not affect the operation of the algorithm.

AdaBoost operates by maintaining a vector of weight values $\vec{D}$. The vector $\vec{D_i}$ represents the values of that vector after round i of the algorithm. $\vec{D}$ is used by the weak learning procedure to find a weak hypothesis which has a low error with respect to those weights. Note that $\vec{D}$ begins as a uniform distribution and is then adjusted exponentially at each round. There is one weight value in $\vec{D}$ for each message in the training set.

The use of h_t to compute weights for the next round ensures a feedback system in which the adjustment of weights is in proportion to the confidence with which that round's weak hypothesis h_t asserts its prediction y_i for the ith message in the training set.

The values used for α_t depend upon the type of weak learning algorithm employed. It can be shown that the training error of AdaBoost is at most equal to the product of Z_t over all rounds, where Z_t is the normalisation factor used to ensure that $\vec{D}$ remains a valid distribution [7].

It is necessary to find values of c_0 and c_1 for each weak predicate which minimise Z_t, in order to produce a minimal overall training error. Schapire and Singer also showed that one can minimise Z_t by choosing $\alpha_t = 1$ and letting:

$$c_j = \frac{1}{2} \ln \left(\frac{W_{+1}^j}{W_{-1}^j} \right)$$

The value of W_b^i is equal to the total weight in $\vec{D_t}$ of all training messages which are in class b and which are in partition j according to the predicate. As stated above, the class is either -1 (spam) or +1 (legitimate).

Given the above definition of $h(x)$, partition 0 is the subset X_0 of the training set for which p holds true; partition 1 is the subset X_1 for which p is false. Thus there are four possible subsets of the training data, which correspond to W_{+1}^0, W_{+1}^1, W_{-1}^0 and W_{-1}^1.

More formally, the W values are defined below. Here, $[q] : \mathbb{B} \rightarrow \{0, 1\}$ evaluates to 1 if and only if q is True.

$$W_b^j = \sum_{i=1}^{m} \vec{D_t}(i)[x_i \in X_j \wedge y_i = b]$$

2.3.3 Support Vector Machines

Support vector machines (SVMs) are a promising type of learning machine. SVM algorithms are a special case of a more general class of learning algorithms known as kernel-based learning algorithms. A general and highly mathematical overview of such algorithms is given by Müller et al [5]. The original SVM concept is due to Vapnik [8].

The basic use of SVMs is to map each message onto a point in feature space and then attempt to find a hyperplane which separates them consistently. In general there might be many such hyperplanes, so the problem is to determine the optimal hyperplane which can most reliably separate spam from non-spam. Drucker et al specifically discussed the use of SVMs for e-mail classification [4].

The general equation of a hyperplane is $\vec{w}.\vec{x} + b = 0$. Here, $\vec{w}$ and $\vec{x}$ are vectors and b is a scalar constant[2]. In this case, $\vec{w}$ defines a constant vector of weights and $\vec{x}$ can be regarded as the feature vector of an e-mail message.

If the two classes are linearly separable then there exists an optimal weight vector $\vec{w^*}$ such that $||\vec{w*}||^2$ is a minimum and for any example message x_i of class y_i:

Either $\vec{w^*}.\vec{x_i} + b \geq 1$ if $y_i = 1$ (legitimate)
or $\vec{w^*}.\vec{x_i} + b \leq -1$ if $y_i = -1$ (spam)

These equations define two distinct hyperplanes. The support vectors of the spam class form one hyperplane; the support vectors of the non-spam class form the other. The distance between the hyperplanes defines a margin which can be maximised by minimising $||\vec{w^*}||^2$. When the margin it at a maximum, the separation between the classes is as well defined as possible.

The above can be combined into a single inequality:

$$y_i((\vec{w^*}.\vec{x_i}) + b) > 1$$

Once $\vec{w^*}$ has been found, the classification decision for any unseen message x_i is to compute $\text{sign}((\vec{w^*}.\vec{x_i}) + b)$.

One reason that SVMs can be effective using linear classification is that it is common to map each data point into a feature space of higher dimensionality. Thus points which might not be linearly separable in n dimensional space may well be so in N dimensional space ($N \in \mathbb{N}$, $N > n$). This is usually achieved with some mapping Φ. Of course, computing the scalar product $\vec{w^*}.\vec{x_i}$ in N dimensional space could be very costly for many dimensions.

2. The reader may recognise this as a generalisation of the two-dimensional straight-line equation: $w_1 x + w_2 y + b = 0$ where $\vec{x} = (\ x \quad y\ )$ and $\vec{w} = (\ w_1 \quad w_2\ )$.

In practice a kernel function is used to calculate the scalar product in n dimensional space instead. A variety of well-known Φ and corresponding kernel functions exist; the choice of kernel depends upon Φ.

For example, consider the case where some data are arranged such that one of the sets of points falls entire on the inside of an ellipse and the other set falls entirely on the outside. Clearly no flat plane can separate the different classes of points in only two dimensional space. Note that these data could be separated by an ellipse of equation:

$$\frac{x^2}{p^2} + \frac{y^2}{q^2} = 1$$

Where p and q are constants.

If it were possible to transform each point to include x^2 and y^2 terms, then the data could be properly classified. Müller et al give the following example mapping:

$$\Phi : \mathbb{R}^2 \rightarrow \mathbb{R}^3$$

$$\begin{pmatrix} x_1 & x_2 \end{pmatrix} \mapsto \begin{pmatrix} x_1^2 & \sqrt{2}x_1x_2 & x_2^2 \end{pmatrix}$$

They demonstrate that the higher feature space scalar product $\Phi(\vec{x}).\Phi(\vec{y})$ can be computed as $(\vec{x}.\vec{y})^2$ and thus that the general kernel function for this class of Φ mappings is $(\vec{x}.\vec{y})^n$ where n is the original feature vector size.

In this example, the corresponding hyperplane equation would be:

$$\begin{pmatrix} \frac{1}{p^2} & 0 & \frac{1}{q^2} \end{pmatrix}.\Phi(\vec{x_i}) + b = 0$$

Hence the classification inequality would be:

$$y_i \left(\begin{pmatrix} \frac{1}{p^2} & 0 & \frac{1}{q^2} \end{pmatrix}.\Phi(\vec{x_i}) + b = 0 \right) \geq 1$$

The general SVM inequality is:

$$y_i((\vec{w^*}.\Phi(\vec{x_i})) + b) \geq 1$$

Here, the weight vector $\vec{w^*}$ is assumed to be already in N dimensional space. If this were not the case it would be simple to compute $\Phi(\vec{w^*})$ as a one-time calculation.

So far, only linearly separable classification has been considered. However, this may not hold true: errors may occur and some training examples may exist in the region between the classes. Thus a more general inequality exists in which slack variables ξ_i are used to allow for less rigid class boundaries:

$$y_i((\vec{w^*}.\Phi(\vec{x_i})) + b) \geq 1 - \xi_i$$

Müller explains how this latter equation can be solved for $\vec{w^*}$ and $\vec{\xi}$. The aim is to achieve a balance between the complexity of classification and the risk of classification error. Central to the algorithm is an optimisation problem to minimise the following sum:

$$\frac{1}{2}||\vec{w^*}||^2 + C\sum_{i=1}^{m} \xi_i$$

Here, C is a constant which represents a cost penalty for misclassification. Each ξ_i is only non-zero if the corresponding training sample x_i has been misclassified.

There are some issues to do with underfitting and overfitting data; a good SVM implementation must be able to produce sensible results which are not unduly swayed by small anomalies or outliers during training. One of the main advantages of SVM classification is that it eliminates extreme anomalous values during training; only points close to the margin are retained.

Drucker et al compared the e-mail classification capabilities of SVMs with three other classification algorithms (including boosting trees) and concluded that boosting and SVM algorithms gave the fastest and most accurate classification. Yet they also found that SVMs required significantly less training time than the other algorithms, which makes them worthy of further study.

2.4 Testing Methodology

Machine learning algorithms are likely have both false negative and false positive results. Content-classification systems must balance the likely risk of either situation. A loose filter would have a low chance of mislabelling genuine messages, but would most likely allow a large number of spam messages through: false negatives. This is undesirable because the sole purpose of a filter is to prevent the user having to sift through large numbers of messages manually. However, a stricter filter might risk catching genuine messages, creating false positives and causing the user to miss real e-mail.

Since classification systems must learn by being shown pre-classified examples, it is necessary to have two training sets (corpora) of messages: one of spam and one of legitimate messages. Once the system has been trained to recognise messages correctly, it should then be shown two further sets of unseen pre-classified messages, again spam and legitimate. Since the class of these messages is known in advance, the

accuracy of classification can be automatically computed.

A number of well-known spam corpora already. For example, some of the SpamAssassin test data are publicly available[3]. There are also some academic corpora such as LingSpam[4] and PU1[5], which are often used by text classification researchers.

For the purposes of these tests, the authors accumulated some 4,000 messages ("Akehurst corpus") consisting of 2,000 spam and 2,000 genuine messages. Additional tests were run with the SpamAssassin and LingSpam corpora.

The method used for testing was ten-fold cross-validation. The classification corpus was divided into ten equal parts, each part having the same proportion of spam as the whole. Ten rounds of learning and classification were performed, each using nine of the parts for training and the tenth for testing, with a different part for testing in each round; the results were then averaged across all rounds using an arithmetic mean.

2.5 Statistical Measures

Adopting the notational conventions from Carreras and Màrquez [3], we define the sets S and L as sets of spam and legitimate messages. We futher define S_+ as the number of correctly classified messages in S_+ and S_- as the number incorrectly classified messages. There are similar definitions for L_+ and L_-.

Two critical measures of classification performance are the False Negative Ratio and False Positive Ratio. A good classification system will have small (near zero) values for these ratios:

$$FNR = \frac{S_-}{S_+ + S_-} \quad FPR = \frac{L_-}{L_+ + L_-}$$

Two other common measures used are precision and recall, which are often used in information retrieval theory:

$$P = \frac{S_+}{S_+ + L_-} \quad R = \frac{S_+}{S_+ + S_-}$$

Precision represents the proportion of blocked messages which are actually spam, while recall gives the proportion of spam messages correctly classified out of all the spam. Clearly it is desirable for precision and recall to be as close to 1 as possible.

As each of the above measures only gives a partial indicator of performance, an additional accuracy measure will be defined for summary purposes:

$$A = \frac{S_+ + L_+}{|S| + |L|}$$

3 Results

For comparison we have tested existing spam implementations using our statistical measures, and compared these with the newly introduced techniques of boosting and SVMs.

3.1 Existing Systems

The first section of Table 2 shows the test results achieved by the existing anti-spam implementations. Clearly, SpamAssassin's Bayesian classifier has the lowest false negative score and hence the highest recall of all (96.9%). Its low false positive score is also impressive, giving it a good overall accuracy ($> 98\%$). The basic SpamAssassin classifier is good, but the Bayesian component clearly yields better results, albeit at the cost of slightly lower throughput.

Mozilla's Bayesian classifier is noteworthy for being the only implementation not to misclassify a single genuine e-mail, hence its 100% precision score. It also had the second-highest overall accuracy score.

Vipul's Razor is interesting for its different approach to the other systems. It relies upon matching against fingerprints of known reported spam. Unfortunately it missed the largest proportion of spam, perhaps due to variability in spam reporting. However it has a respectable precision score. Of the false positives it did find, not a single one was a personal e-mail. Certain messages like the New Scientist newsletter have perhaps been mistakenly reported as spam.

It is noticeable that the false-positive ratios of all the implementations are very low, the highest being SpamAssassin's 1.4%. This suggests that all these implementations are optimised to reduce misclassification of genuine e-mail, since that may have a more severe cost to the typical (non-technical) user than the arrival of a few spam messages in their in-box. The large precision scores (all $\geq 98.3\%$) support this hypothesis.

Some authors have done a cost-based analysis of misclassification [1]. Many assume that false positives are more serious than false nega-

3. `http://spamassassin.org/publiccorpus/`

4. `http://www.aueb.gr/users/ion/lingspam_public.tar.gz`

5. `http://www.aueb.gr/users/ion/pu1_encoded.tar.gz`

tives, due to the potential economic loss to business recipients of genuine potential sales leads. However, the cumulative cost to recipients in time lost deleting spam manually can also be large. In this work, all misclassification is regarded as equally undesirable.

3.2 New Implementations

Two new anti-spam classifiers were written based on the Boosting and SVM techniques described earlier. They were tested with the same methodology and corpus as the other implementations.

3.2.1 Feature Vectors

Both approaches require the conversion of each e-mail message into a feature vector which characterises it. Thus the design of an efficient and effective feature space function is important. The elements of the feature vector used are those defined in Section 2.1. Most of these are numbers representing fixed features such as counts of certain HTML tags.

However, some features represent the number of times certain key indicator words appear in a message. Since the vocabulary used in all e-mail is so large and diverse, it is impossible to include all word frequencies in the feature vector. Therefore only the top w most frequently occurring words are included in the vector.

Words were taken to be strings of at least two alphanumeric characters, separated by whitespace and certain punctuation symbols (such as commas). All words were converted into lower case and embedded HTML elements in the middle of words were ignored. Thus the earlier example of "Via<!Qj>gr@" might be read as "viagr@".

Further word matching was done by glyph-based similarity matching. Each character is assigned to a character class, according to how similar the symbols used to render those characters would look to the casual reader. Thus characters such as "á", "ä", "â", "á" and "@" belong to the "a" class and would be matched accordingly. Hence "Via<!Qj>gr@" would actually be recognised as "viagra".

Some experiments involved "differential" frequency, rather than using the top w most frequently-occurring words in the training corpus. The frequencies of each word in spam and legitimate e-mail (f_0 and f_1 respectively) were found; the words were then sorted in non-increasing order of $|f_0 - f_1|$, with the top w in the list being chosen for inclusion in the feature vector.

Here, the aim was to pick words which were potentially strong indicators for either class of message. Words with similar frequencies in both classes will have $|f_0 - f_1|$ close to zero and so they will appear low down in the sorted list. Thus weaker classifiers are less likely to be chosen. Also, infrequent words are unlikely to appear high in such a list, so their presence should not skew the results.

Some further tests were also performed using simply word-based information in the feature vector. These are shown by the phrase "words only" in table 2.

3.2.2 Pre-processing

Before generating the feature vectors, each messge undergoes pre-processing as follows:

- Decoding any headers encoded according to RFC 2047
- Decoding any body text in quoted-printable or base64 content transfer encoding, according to RFC 2045
- Removing the binary data of any attachments in the message. Such data does not contribute to classification and only slows down later processing

The purpose of transforming messages is to ensure that what the filter sees is the same essential information which the user sees.

3.2.3 Boosting Results

In the system described, there are typically over 1,000 elements in the feature vector. It would be impractical to use all of them, and indeed many may turn out to be poor spam indicators. The AdaBoost algorithm will select the best predicates for the training corpus.

Carreras and Màrquez found that training in excess of 150 rounds (i.e. 150 weak rules) yielded little benefit in classification performance. However, this is a result which can be tested with the new implementation; there is still some scope for tuning t, the number of rounds. Typically 150 rounds of boosting were needed in our experiments also.

The section section of Table 2 shows the results for Boosting. Four experiments are shown, which are the best achieved performance for: the simple feature vector set; the same set complemented by features based on the differential frequency method for choosing words; simply using

Program	FNR %	FPR %	Precision %	Recall %	Accuracy %
Mozilla Bayesian	5.5	0	100.0	94.5	97.3
SpamAssassin Default	15.7	1.4	98.4	84.3	91.5
SpamAssassin Bayesian	3.1	0.7	99.3	96.9	98.1
Vipul's Razor	55.0	0.8	98.3	45.1	72.1
Boosting	0.6	0.5	99.5	99.4	99.4
Boosting (diff sel)	0.6	0.5	99.5	99.4	99.4
Boosting (words only)	2.4	3.1	96.9	97.6	97.3
Boosting (words only, diff sel)	16.4	42.2	66.4	83.6	70.7
SVM	4.8	3.2	96.7	95.2	96.0
SVM (diff sel)	0.7	0.2	99.8	99.3	99.6
SVM (words only)	12.8	50.1	63.5	87.2	68.6
SVM (words only, diff sel)	12.0	46.0	65.7	88.0	71.0

Table 2: Results, Akehurst Corpus

the words as features; and simply using words chosen by differential selection. Typically 500 words were used except in the best performance for differential selection with words only where 10 were sufficient.

3.2.4 SVM Results

An existing SVM code library was used. This is known as LIBSVM and is available from `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`. LIBSVM was written by Chih-Chung Chang and Chih-Jen Lin of the National Taiwan University.

Experiments concentrated upon the Gaussian Radial Basis Function (RBF) kernel, since early experiments indicated that this provides excellent results in many situations.

Results are shown in Table 2. In these experiements a value of 7 was used for C and the number of words 500, expect for the words only methods where w was 50 for non-differential selection and 25 for differential selection.

4 Conclusions

Firstly, it is clear that the Boosting and SVM approaches can classify e-mail at least as accurately as existing implementations, often exceeding the standard of existing tools.

Use of word features alone is insufficient. When the indicators listed in Section 2.1 were included in the feature vectors, both Boosting and SVM classification accuracy were clearly higher than without such elements. Systems which use only word tokens are discarding invaluable information. Since the behaviour of a classifier depends greatly upon the training set, it is difficult to recommend specific features to include in all cases.

The differential frequency method for selecting words can give good results provided non-word features are also included.

Where Support Vector Machines are used, a Gaussian RBF kernel with $1 \leq C \leq 7$ gives good accuracy. When Boosting is used, 150 rounds is a reasonable choice.

Further work could concentrate on removing "noise" caused by weak classifiers. SVMs are affected by noise since they use all dimensions in training and classification. Boosting might be more resistant to noise since it picks only the best dimensions from the vector. Further study is needed on the effects of noise and how to reduce it. Also, the Boosting algorithm outputs a confidence level for how well a message fits the predicted class. This information is discarded by the implementation, but it could be useful in tuning classification or sorting messages within a MUA.

References

[1] I. Androutsopoulos, J. Koutsias, K. Chandrinos, and D. Spyropoulos. An experimental comparison of naïve bayesian and keyword-based anti-spam filtering with personal e-mail messages. In *Proc. 23rd ACM SIGIR Annual Conference*, pages pp.160–167, 2000.

[2] APIG. Spam: A report of the all-party internet group, 2003. `http://www.apig.org.uk/spam_report.pdf`.

[3] X. Carreras and L. Màrquez. Boosting trees

for anti-spam email filtering. In *Proceedings Of RANLP-01, 4th Intl. Conference on Recent Advances in Natural Language Processing*, 2001.

[4] H. Drucker, D. Wu, and V. Vapnik. Support vector machines for spam categorization. *IEEE Transactions on Neural Networks*, Vol. 10(No. 5):pp.1048–1054, 1999.

[5] K. Müller, S. Mika, G. Rätsch, K. Tsuda, and B. . Schölkopf. An introduction to kernel-based learning algorithms. *IEEE Transactions in Neural Networks*, Vol. 12(No. 2):pp.181–201, March 2001 2001.

[6] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz. A bayesian approach to filtering junk e-mail. *Learning for Text Categorization – Papers from the AAAI Workshop*, pages pp. 55–62, 1998.

[7] R. E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):pp. 297–336, 1999.

[8] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, New York, 1995.