Support Vector Machines for Spam Categorization

Harris Drucker, *Senior Member, IEEE*, Donghui Wu, *Student Member, IEEE*, and Vladimir N. Vapnik

***Abstract*— We study the use of support vector machines (SVM's) in classifying e-mail as spam or nonspam by comparing it to three other classification algorithms: Ripper, Rocchio, and boosting decision trees. These four algorithms were tested on two different data sets: one data set where the number of features were constrained to the 1000 best features and another data set where the dimensionality was over 7000. SVM's performed best when using binary features. For both data sets, boosting trees and SVM's had acceptable test performance in terms of accuracy and speed. However, SVM's had significantly less training time.**

***Index Terms*— Boosting algorithms, classification, e-mail, feature representation, Ripper, Rocchio, support vector machines.**

I. Introduction

WE DEFINE spam as an e-mail message that is unwanted—basically it is the electronic version of junk mail that is delivered by the postal service. One of the reasons for the proliferation of spam is that bulk e-mail is very cheap to send and although it is possible to build filters that reject e-mail if it is from a known spammer, it is easy to obtain alternative sending addresses. Good online sources and information about spam include http://spam.abuse.net, http://www.cauce.org, and http://www.junke-mail.org. There have been various attempts to use learning machines that classify e-mail [1], [2].

Solutions to the proliferation of spam are either technical or regulatory [3]. Technical solutions include filtering based on sender address or header content. The problem with filtering is that sometimes a valid message may be blocked. Thus, it is not our intent to automatically reject e-mail that is classified as spam. Rather, we envision the following scenario: in the training mode, users will mark their e-mail as either spam or nonspam. After a finite number of examples are collected, the learning machine will be trained and the performance on new examples predicted. The user can then invoke the e-mail classifier immediately or wait until the number of examples is enough such that performance is acceptable. After the training mode is complete, new e-mail will be classified as spam or nonspam. In one presentation mode, a set of new e-mail messages is presented in a manner consistent with the time of delivery and the spam messages color-coded. It is then up to the user to either read the e-mail or trash the e-mail. An alternative presentation mode is to deliver e-mail to the user in decreasing order of probability that the e-mail is nonspam. That is, e-mail with high probability (according to the classifier) of being nonspam is at the top of the list. In either of these modes, the filter does not reject any messages, only indicates whether the message has a high priority of being spam.

It is highly desirable that if the user decides that e-mail messages be rank-ordered by degree of confidence that the rank ordering be reliable. By reliable, we mean that the user can either start at the top of the list of e-mail messages and be fairly confident that they represent nonspam messages or start at the bottom of the list and be confident that the messages are spam. It is only near the middle of the list (low confidence) that it is reasonable to the user that a few nonspam or spam messages may be misclassified. Therefore, it is important that our learning algorithm not only classify the messages correctly but that a measure of confidence is associated with that classification so that the message can be rank ordered.

In the next section a number of design choices are outlined. In Section III we describe the data sets and experiments. In Section IV, we show how variation of the one parameter in SVM's changes performance. The conclusions are presented in Section V.

II. Design Choices

A. Feature Representation

A feature is a word. In the development below, w refers to a word, $\mathbf{x}$ is a feature vector that is composed of the various words from a dictionary formed by analyzing the documents. There is one feature vector per message. $\mathbf{w}$ refers to a weight vector usually obtained from some combination of the $\mathbf{x}$'s. There are various alternatives and enhancements in constructing the $\mathbf{x}$ vectors. We consider some of them:

- TF—Term Frequency: The ith component of the feature vector is the number of times that word w_i appears in that document. In our case, a word is a feature only if it occurs in three or more documents. (This prevents misspelled words and words used rarely from appearing in the dictionary). Sometimes the feature vector is normalized to unit length.
- TF-IDF uses the above TF multiplied by the IDF (inverse document frequency). The document frequency (DF(i)) is the number of times that word w_i occurs in all the documents (excluding words that occur in less than three documents). The inverse document frequency (IDF) is defined as

$$\text{IDF}(w_i) = \log\left(\frac{|D|}{DF(w_i)}\right)$$

Manuscript received January 15, 1999; revised April 29, 1999.

H. Drucker is with AT&T Labs-Research, Red Bank, NJ 07701 USA, and is also with the Department of Electronic Engineering, Monmouth University, West Long Branch, NJ 07764-1898 USA.

D. Wu is with Rensselaer Polytechnic Institute, Troy, NY12181 USA.

V. N. Vapnik is with AT&T Labs-Research, Red Bank, NJ, 07701 USA.

Publisher Item Identifier S 1045-9227(99)07272-0.

where $|D|$ is the number of documents. Typically, the feature vector that consists of the TF-IDF entries is normalized to unit length.

- Binary representation which indicates whether a particular word occurs in a particular document. A word is a candidate only if it occurs in three or more documents.
- Use of a stop list in addition to any of the above: Words like "of," "and," "the," etc., are used to form a stop list. Words on the stop list are not used in forming a feature vector. The rationale for this is that common words are not very useful in classification. The argument against using a stop list is that it is not obvious which words, beyond the trivial, should be on the stop list. It may be obvious that articles like "a," "an," should be on the stop list. However, should a word like "now" be on the stop list? The choice of words to put on a stop list is probably a function of the classification task and it would be better if learning algorithm itself determined whether a particular word is important or not. Use of word stemming: words such as "build," "builder," and "building" are shortened to the word stem "build." This lowers the size of the feature vector but it may be the case that certain forms of a word (such as the active tense) may be important in classification.

B. Number of Features

The choice is between using some of the features or all of the features. In text recognition, a feature is a word. One possible advantage of using a finite number of features is better generalization. By generalization we mean that good performance on the training set generalizes to good performance on a separate test set. Depending on the learning algorithm, it may be the case that there is an optimum set of features, less than the total number of available features. For example, if the dimensionality of the classification space is greater than the number of examples, then the examples may always be separable by a nonunique hyperplane with zero training error (assuming the patterns are independent). Since there are, in general, an infinite number of separating hyperplanes, one does not obtain the optimal separating hyperplane (the one that has the best test performance).

A few of the mechanisms designed to find the optimum number of features (and the best features) are [4] document frequency thresholding, information gain, mutual information, term strength, and χ^2. In comparing two learning algorithms, Yang and Petersen found that, except for mutual information gain, all these feature selection methods had similar performance and similar characteristics. That is, as the number of features used was decreased from all the features to some smaller number, the test performance improved. Then, once the number of features decreased below a critical value, the test error rate increased.

Thorsten Joachims [5] did similar experiments and compared five learning algorithms including naïve Bayes, Rocchio, k-nearest neighbors, C4.5 [6] and SVM's. He found that, except for SVM and Bayes, the optimum number of features was less than the total number of features. For SVM, using all the features gave better performance than any of the other techniques.

The main disadvantage of searching for the best features is that it requires additional time in the training algorithm. In the case of information gain and mutual information gain, the features are ranked by the feature selection method from high to low, which is linear in the number of examples and linear in the number of features (giving quadratic complexity). Then, to find the optimum number of features, one must apply the learning algorithms to different set sizes to find the minimum error rate. Thus the complexity of the algorithm to find the optimum number of features is at least quadratic times the complexity of the learning algorithm. It would be far better if the learning machine itself either made the feature selection automatically or used all the features. C4.5 has the former characteristic while SVM's have the latter.

C. Performance Criteria

- *Recall and Precision*: In information retrieval tasks, documents could be assigned multiple categories. For instance, a story about the high wages of baseball players could be categorized as belonging to both the term "financial" and the term "sports." When there are multiple categories, performance measures such as recall and precision [4] are used

$$\text{recall} = \frac{\text{categories found and correct}}{\text{total categories correct}}$$
$$\text{precision} = \frac{\text{categories found and correct}}{\text{total categories found}}.$$

  Since our problem is a two-class classification task, recall and precision are not needed here.
- *Error Rate*: Error rate is the typical performance measure for two-class classification schemes. However, two learning algorithms can have the same error rate, but the one which groups the errors near the decision border is the better one.
- *False Alarm and Miss Rate*: We define the false alarm and miss rates as

$$\text{miss rate} = \frac{\text{nonspam samples misclassified}}{\text{total nonspam examples}}$$
$$\text{false alarm rate} = \frac{\text{spam samples misclassified}}{\text{total spam examples}}.$$

  The advantage of the false alarm and miss rates is that that they are a good indicator of whether the errors are close to the decision border or not. Given two classifiers with the same error rate, the one with lower false alarm and miss rates is the better one.

We fix the miss rate and try to find the algorithm that minimizes the false alarm rate for that fixed miss rate. These rates will correspond to the error rates averaged using ten-fold cross validation. In ten-fold cross validation, the entire set of examples is divided into ten approximately equal sets. Nine of the ten parts are using for training, one of the ten parts is used for testing. This is repeated ten times, each time using another test set. The total misses and false alarms are counted and divided by ten times the number of test examples. This

is more accurate than averaging the ten false alarms and miss rates.

Recall that that even if an input message is mistakenly misclassified as spam when it is not, it is still presented to the user for the ultimate decision.

D. Training and Classification Speed

The e-mail server may be a unit on which many users have an account. Therefore, the training time must be reasonable and the classification speed must be fast. Generally, this leaves out neural networks which take extensive time for training. We therefore tried the following four algorithms: boosting (using decision trees), Ripper, Rocchio, and linear SVM's. The first two are nonlinear learning algorithms but can be very fast in execution and the last two are linear.

E. Choice of Learning Algorithms

1) Boosting Algorithms: The boosting algorithms (Fig. 1) are techniques to combine a number of weak learners to form an ensemble. The term weak leaner arrives from the PAC (probably approximately correct) [7], [8] learning community and indicates that the learning algorithm can learn with error rate slightly better than 50%. C4.5 classification trees are candidate weak learners even though their error rates can be much better than 50%. This version of boosting works as following: train the first member of the ensemble with N training samples. In order to train the next member of the ensemble, the probability that a training sample will be picked to train the second member of the ensemble is adjusted upwards for "hard" examples and down for "easy" examples. By hard examples, we mean those examples that the first weak learner misclassifies.

Each member of the ensemble is subsequently trained on examples picked from the original training set with their probabilities adjusted upwards or downwards depending on whether the previous members of the ensemble classified the training pattern incorrectly or correctly, respectively.

For our weak learner, we use classification trees built using a version of C4.5. Because classification trees can build disconnected decision regions, they are nonlinear. Classification trees can be very fast in execution. An advantage of C4.5 is that features are picked as part of the training algorithm and therefore there is no need to rank order the features by some other mechanism (like mutual information) first. C4.5 decision trees are built by examining a measure related to information gain and this can be time consuming because it has to be done multiple times (equal to the number of nodes in a tree). Boosting has been shown to drive the error rate far below that of one weak learner. Both the boosting algorithms itself and the building of decision trees make training long unless there is a small number of trees in the ensemble. However, there is no way to predict in advance how many trees should be in the ensemble.

In Fig. 1 WeakLearn refers in our case to a C4.5 type algorithm. When the distribution is recalculated, a typical method of picking a training set for this member of the ensemble is as follows.

Input: N documents and labels
$\langle (d_1, y_1), \ldots, (d_N, y_N) \rangle$ where $y_i \in \{-1, 1\}$
Integer T specifying number of iterations

Initialize $D_1(i) = 1/N$ for $i = 1, \ldots, N$

Do for s=1,2,...T (until training error is zero)

- Call WeakLearn and get a weak hypothesis h_s
- Calculate the error rate $\varepsilon_s = \sum_{i:h_s(d_i) \neq y_i} D_s(i)$
- Set $\alpha_s = \frac{1}{2} \ln \left\{ \frac{1 - \varepsilon_s}{\varepsilon_s} \right\}$
- Update distribution

$$D_{s+1}(i) = \frac{D_s(i)}{Z_s} \times \begin{cases} \exp(-\alpha_s) & \text{if } h_s(d_i) = y \\ \exp(\alpha_s) & \text{if } h_s(d_i) \neq y_i \end{cases}$$

where Z_s is a normalization factor that makes D_{s+1} a distribution.

Output the final hypothesis:

$$h_{fin}(d) = sign\left\{ \sum_{s=1}^{T} \alpha_s h_s(d) \right\}$$

Fig. 1. Boosting algorithm.

Construct line segments of length $D(i)$ with total length $\sum_{i=1}^{N} D(i)$ and pick N numbers at random from that total length. If the number is from interval $D(i)$, then example i is used as one of the training samples for that round of boosting. It may be the case that multiple copies of a particular vector are used in training and no samples of easy training examples are used.

It is also important to note that in calculating ε_s, all the training examples are used in that calculation even if they were all not used in training.

2) Support Vector Machines: SVM's are discussed extensively in this issue. Also see [9]–[11]. The key concepts we want to use are the following: there are two classes, $y_i \in \{-1, 1\}$, and there are N labeled training examples: $(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_N)$, $\mathbf{x} \in R^d$ where d is the dimensionality of the vector.

If the two classes are linearly separable, then one can find an optimal weight vector $\mathbf{w}^*$ such that $\|\mathbf{w}*\|^2$ is minimum; and

$$\mathbf{w}^* \bullet \mathbf{x}_i - b \geq 1 \qquad \text{if } y_i = 1$$
$$\mathbf{w}^* \bullet \mathbf{x}_i - b \leq -1 \qquad \text{if } y_i = -1$$

or equivalently

$$y_i(\mathbf{w}^* \bullet \mathbf{x}_i - b) \geq 1.$$

Training examples that satisfy the equality are termed support vectors. The support vectors define two hyperplanes, one that goes through the support vectors of one class and one goes through the support vectors of the other class. The distance between the two hyperplanes defines a margin [10] and this margin is maximized when the norm of the weight vector $\|\mathbf{w}^*\|$ is minimum. Vapnik has shown we may perform this minimization by maximizing the following function with respect to the variables α_j:

$$W(\alpha) = \sum_{i=1}^{N} \alpha_i - 0.5 \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j (\mathbf{x_i} \bullet \mathbf{x_j}) y_i y_j$$

subject to the constraint: $0 \leq \alpha_j$ where it is assumed there are N training examples, $\mathbf{x_i}$ is one of the training vectors, and $\bullet$ represents the dot product. If $\alpha_j > 0$ then $\mathbf{x_j}$ is termed a support vector. For an unknown vector $\mathbf{x}_j$ classification then corresponds to finding

$$F(\mathbf{x}_j) = \text{sign}\{\mathbf{w} * \bullet \mathbf{x}_j - b\}$$

where

$$\mathbf{w}^* = \sum_{i=1}^{r} \alpha_i y_i \mathbf{x}_i$$

and the sum is over the r nonzero support vectors (whose α's are nonzero).

The advantage of the linear representation is that $\mathbf{w}^*$ can be calculated after training and classification amounts to computing the dot product of this optimum weight vector with the input vector.

For the nonseparable case, training errors are allowed and we now must minimize

$$\|\mathbf{w}^*\|^2 + C \sum_{i=1}^{N} \xi_i$$

subject to the constraint

$$y_i(\mathbf{w}^* \bullet \mathbf{x}_i - b) \leq 1 - \xi \qquad \xi \geq 0$$

ξ is a slack variable and allows training examples to exist in the region between the two hyperplanes that go through the support points of the two classes. We can equivalently minimize $W(\alpha)$ but the constraint is now $0 \leq \alpha_i \leq C$ instead of $0 \leq \alpha_j$. Maximizing $W(\alpha)$ is quadratic in α subject to constraints and may be solved using quadratic programming techniques, some of which are particular to SVM's [12], [13].

The advantage of linear SVM's is that execution speed is very fast and there are no parameters to tune except the constant C. We will show that the performance of the SVM is remarkably independent of the choice of C as long as C is large (over 50). Another advantage of SVM's is that they are remarkably intolerant of the relative sizes of the number of training examples of the two classes. In most learning algorithms, if there are many more examples of one class than another, the algorithm will tend to correctly classify the class with the larger number of examples, thereby driving down the error rate. Since SVM's are not directly trying to minimize the error rate, but trying to separate the patterns in high dimensional space, the result is that SVM's are relatively insensitive to the relative numbers of each class. For instance, new examples that are far behind the hyperplanes do not change the support vectors. The possible disadvantages of SVM's are that the training time can be very large if there are large numbers of training examples and execution can be slow for nonlinear SVM's, neither of these cases being present here.

3) Ripper: Ripper is a program for inducing classification rules from a set of examples. Unlike the other algorithms, it does not need a feature vector. It forms if–then rules which are disjunctions of conjunctions, e.g.,

A document d is considered to be spam if and only if

(word FREE appears in d) OR
(word low appears in d AND word cost appears in d)
OR
.
.
(word !!!! appears in d).

Ripper [14] works by adding rules to cover positive examples and then prunes those rules to form a best fit to a separate pruning set. The advantages of Ripper are as follows.

- The if–then clauses are easy for humans to understand.
- Ripper is very fast in training and testing.
- Ripper allows users to supply prior knowledge constraints.
- Ripper allows values to be nominal, continuous, or set-valued.
- Ripper is nonlinear.

4) Rocchio: This type of classifier [15], [16] uses normalized TF-IDF representation of the training vectors. A prototype vector $\mathbf{w}$ is formed

$$\mathbf{w} = \frac{1}{N_{\text{spam}}} \sum_{i \in \text{spam}} \mathbf{x_i} - \beta \frac{1}{N_{\text{nonspam}}} \sum_{i \in \text{nonspam}} \mathbf{x_i}$$

where N indicates the number of documents that are classified as spam or nonspam. Elements of the prototype vector that are negative are set to zero and then $\mathbf{w}$ is normalized to unit length. Classification is performed by the dot product of the prototype vector and the candidate test vector. Those with large positive dot products are spam and those with large negative values are nonspam. Unlike any of the algorithms discussed previously, there is no natural threshold for the dot product. That is, the algorithm does not tell us for what values above a critical value of the dot product should we classify the document as spam. This critical value must be obtained by rank ordering the outputs of dot products of the prototype vector with all the training vectors and finding that critical value that minimizes the training error. We emphasize that the test vectors should not be used to find that threshold. Similarly, the optimum value of β should not be obtained from the test set. It should be obtained from the training set and it is that β that minimizes the training error. The advantage of Rocchio's algorithm is that it is fast in training and testing. The disadvantage is that

TABLE I
FOR DATA SET I, FALSE ALARM RATES CORRESPONDING TO A 5% MISS RATE. x INDICATES THAT RIPPER WAS UNABLE TO OBTAIN THE 5% MISS RATE FOR THESE TWO DATA SETS AND THUS NO FALSE ALARM OR ERROR RATES ARE REPORTED

	SVM (TF features)	SVM (binary features)	Boosting (TF features)	Ripper	Rocchio (TF-IDF features)
bodnostop	.0964	.0929	.0403	.1468	.4929
bodstop	.1204	.1153	.0417	.1646	.2576
subbodnostop	.0176	.0152	.0124	.0788	.0470
subbodstop	.0270	.0317	.0188	.0858	.0588
subnostop	.5491	.5493	.5186	x	.8423
substop	.7188	.7576	.6373	x	.8471

one has to search for the optimum threshold and the optimum β on the training set which takes extra training time and does not necessarily generalize well to the test set.

Further text categorization methods may be found in [17] and [18].

III. DATA SETS

We used two data sets. Data set I was collected by an AT&T staff member and consists of 850 messages that he considered spam and 2150 messages that were nonspam. All e-mail messages consist of a subject and body and the various algorithms were tried on either the subject alone, the body alone, or the subject and body together. In addition, a stop list was either used or not used and all words were converted to lower case. The 1000 best features in each case were used (ranked using mutual information). The original feature entries are TF (term frequency) and could later be converted to TF-IDF or binary features.

Thus there were six data sets constructed from the original 3000 messages:

- *bodnostop*—words from the body only without using a stop list.
- *bodstop*—words from the body only and the stop list was used.
- *subbodnostop*—words from the subject and body without using a stop list
- *subbodstop*—words from the subject and body using a stop list.
- *subnostop*—words form the subject without using a stop list.
- *substop*—words from the subject using a stop list.

Results are shown in Table I. It can therefore be seen that the smallest error rates are given by using the subject and body without using a stop list (subbodnostop) and that SVM should be used with binary features. To put the error rates in context, it should be noted that the test sample size is 300 and thus a difference of approximately 0.003 can be attributed to a numerical difference of one error per run of the ten-fold cross validation error rate estimate. Thus there is not much difference on subbodnostop between SVM using binary features and boosting. It should be emphasized that the spam filter will never actually reject any messages classified as spam. The use of the false alarm and miss rates is a mechanism to compare performance.

The second data set was collected from members of the AT&T technical staff who forwarded their spam and nonspam to the authors. Since the spam had already passed through the AT&T firewall, it would be expected that the spam would be harder to classify. There were 314 spam and 303 nonspam messages. Rather than limiting the feature vector to a finite size we used all words in the messages as long as they occurred in at least three messages. Furthermore, stemming was not used. Ripper was unable to achieve either a 5% or 1% miss rate.

TABLE II
ERROR AND FALSE ALARM RATES FOR DATA SET II USING A DICTIONARY CONSISTING OF UPPER- AND LOWERCASE WORDS

	error rate	false alarm for 1% miss rate	false alarm for 5% miss rate
SVM(TF)	.0327	.0446	.0229
SVM(binary)	.0213	.0229	.0159
Rocchio (TF IDF)	.0327	.0964	.0546
Boosting (TF)	.0278	.0501	.0245

TABLE III
ERROR AND FALSE ALARM RATES FOR DATA SET II USING A DICTIONARY CONSISTING OF LOWERCASE WORDS ONLY

	error rate	false alarm for 1% miss rate	false alarm for 5% miss rate
SVM(TF)	.0344	.0344	.0223
SVM(binary)	.0213	.0222	.0127
Rocchio (TF IDF)	.0360	.0771	.0546
Boosting (TF)	.0180	.0285	.0126

This data set had two versions. In Version I (Table II), words that were all capitals were retained but words that were a mixture of upper and lower case were translated to lower case. The rationale for this is that we could conjecture that words like "FREE" would more likely to occur in spam than words like "free." Thus, these two words would be two separate features in this version of the database. Furthermore the word "free" if it occurred in the subject at least three times and the body at least three times would be two separate features. The net result is that the feature vector of Version I was of size 7458.

Once again, use of binary features is preferred for SVM and boosting give approximately equal performance if performance is based on raw error rate. With approximately 60 messages in the test set, one error corresponds to a numerical difference of 0.016 67. As pointed out previously, the false alarm rates give a measure of dispersion of the errors. Therefore, on this basis although boosting and SVM using binary features are comparable in error rate performance, SVM is much better in terms of dispersion of errors.

TABLE IV
ERROR, FALSE ALARM, AND MISS RATES FOR DATA SET II USING A DICTIONARY CONSISTING OF LOWERCASE WORDS ONLY

	error rate	false alarm rate due to 1% miss rate	miss rate due to 1% false alarm rate	equal rates
SVM (binary)	.0213	.0222	.0236	.0193
Rocchio (TF IDF)	.0360	.0771	.0602	.0495
Boosting (TF)	.0180	.0285	.0338	.0223

In Version II of this data set, all words were converted to lower case which lowered the dimensionality to 6577. The results for this data set are shown in Table III. First, in comparing Tables II and III, we see that there is no advantage in keeping words that are all upper case. Second, it appears that once again, it is better to use binary features with SVM machines. Finally, although boosting has the better performance in terms of raw error rate, the spread of errors is better using SVM with binary features.

Operating points other than the false alarm rate due to a fixed miss rate may be of interest. Therefore, in Table IV, we present the miss rate due a fixed false alarm rate and the operating point where the false alarm rates and miss rate are equal (last column). These results are consistent with the conclusions reach previously, namely that boosting is superior in terms of test error rate but that SVM is better in terms of the spread of errors.

The running times for training and testing have not been optimized for either boosting machines or SVM's. However, since the SVM only has to execute one dot product, it would be expected to be faster than boosting C4.5 decision trees. Both have acceptable speeds, in the order of milliseconds for boosting decision trees and microseconds for SVM. This does not include the time to parse a message to build up its feature set. However, there is a remarkable difference in training time for trees against that of SVM's. In our research version of these algorithms, boosting trees take hours to train and SVM's, in the order of minutes. The average tree size for this database is approximately 11 trees which take inordinately long to build. Thus based on a combination of training time and performance, SVM's are superior.

There are two reasons we believe that the training time for SVM's with binary features can be reduced. The first is that the vectors are binary and the second is that the feature vectors are sparse (typically only 4% of a vector is nonzero). This allows the dot product in the SVM optimization to be replaced with faster nonmultiplicative routines.

IV. PERFORMANCE AS A FUNCTION OF UPPER BOUND

The upper bound C on α affects performance for the cases where the training data is not separable by a linear SVM. In general there is an optimum value of this constant. However, this best value of C cannot be obtained be examining the training data and it is unfair to do so by examining the test data. Only by having a third set of data, a validation set, would it be possible to optimize for C. In that case, one would maximize $W(\alpha)$ using the training set, and then find the performance on the validation set. Then one would pick another C and repeat until the performance on the validation set is optimum. One could then use the test set to find the final performance. Even if we had the luxury of using a validation set, this iterative process would take took long.

However, it is illustrative to see what happens if we tune C using the test data (Tables V and VI). We see that as C is decreased, the number of support vectors increases and the number of the support vectors that have maximum α increases. However, for the binary case, the performance degrades while for the TF features case, the performance improves and then degrades as C is lowered. The results seem to indicate that one can obtain almost equivalent performance by using the correct C on the TF formatted data. However, in that case we must spend time searching for that optimum, while for the binary features, that is not the case. Furthermore, there is a wide variation in C that gives equivalent performance on the binary data.

TABLE V
ERROR AND FALSE ALARM RATES AS A FUNCTION OF THE UPPER BOUND C FOR BINARY DATA USING LOWERCASE WORDS ONLY

C	error rate	false alarm rate due to 1% miss rate	number of α 's at bound	number of support vectors
50 000	.02132	.02229	1.3	210
5 000	.02132	.02229	1.3	210
500	.02132	.02229	1.3	210
50	.02132	.02229	1.3	210
5	.03113	.02229	4.2	177
.5	.03444	.03184	69.5	247
.05	.05000	.08598	466	509

TABLE VI
ERROR AND FALSE ALARM RATES AS A FUNCTION OF THE UPPER BOUND C FOR TF DATA USING LOWERCASE WORDS ONLY

C	error rate	false alarm rate due to a 1% miss rate	number of α 's at bound	number of support vectors
50 000	.03442	.02229	1.2	160
5 000	.03442	.02229	1.2	160
500	.03442	.02229	1.2	159
50	.03442	.02229	1.2	159
5	.02295	.01592	2.0	218
.5	.04426	. 02229	116	229
.05	.07868	.12101	423	462

V. CONCLUSIONS

Based on examining a number of data sets, different feature representation, and different learning algorithms, we come to the following conclusions.

1) For the best case (all words converted to lower case), SVM's (using binary features) and boosting (using TF features) are the two best candidates. Boosting has a lower error rate but the dispersion of errors is better using SVM's.

2) In a choice of between using a stop list and not using a stop list, it is preferable that a stop list not be used.
3) Based on a review of the literature, all the features should be used rather than a subset. The literature seems to indicate that there is an optimum set of features that depends on both the data and the algorithm. However, searching for the best features takes an unacceptable period of time. Boosting using C4.5 decision trees implicitly uses a choice of best features as part of the algorithm and SVM performance does not degrade if too many features are used.
4) Training time using boosting decision trees is inordinately long.

It should be remembered that these performance figures are based on data sets collected from different individuals. When an individual marks his or her own e-mail, then the from field of the message can be used. This can improve the accuracy in at least two ways: the user can generate a list of acceptable senders that is always noted as nonspam no matter what the subject and body contents. Furthermore, return e-mail that is a response to a user query will always be accepted as nonspam.

REFERENCES

[1] W. W. Cohen, "Learning rules that classify e-mail," in *Proc. 1996 AAAI Spring Symp. Inform. Access.*

[2] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz, "A Bayesian approach to filtering junk e-mail," in *AAAI'98 Wkshp. Learning for Text Categorization*, Madison, WI, July 27, 1998.

[3] L. Faith Cranor and B. H. LaMacchia, "Spam!," *Commun. ACM*, vol 41, no. 8, pp. 74–83, Aug. 1998; also see http:///www.research.att.com/lorrie/pubs

[4] Y. Yang and J. O. Pedersen, "A comparative study of feature selection in text categorization," in *Proc. 14th Int. Conf. Machine Learning*, 1997.

[5] T. Joachims, "Text categorization with support vector machine: Learning with many relevant features," in *European Conf. Machine Learning*, 1998.

[6] C. Ross Quinlin, *C4.5: Programs for Machine Learning.* San Mateo, CA: Morgan Kaufmann, 1988.

[7] Y. Freund and R. E. Schapire, "Experiments with a new boosting algorithm," in *Machine Learning: Proc. 13th Int. Conf.* San Mateo, CA: Morgan Kaufmann, 1996, pp. 148–156.

[8] ——, "Game theory, on-line prediction, and boosting," in *Proc. 9th Annu. Conf. Comput. Learning Theory*, 1996, pp. 325–332.

[9] V. Vapnik, *Estimation of Dependencies Based on Empirical Data.* New York: Springer-Verlag, 1992.

[10] ——, *The Nature of Statistical Learning Theory.* New York: Springer-Verlag, 1995.

[11] H. Drucker, C. J. C. Burges, L. Kauffman, A. Smola, and V. Vapnik, "Support vector regression machines," in *Neural Inform. Processing Syst. 9*, M. C. Mozer, J. I. Joradn, and T. Petsche, Eds. Cambridge, MA: MIT Press, 1997, pp. 155–161.

[12] J. C. Platt, "Sequential minimal optimization: A fast algorithm for training support vector machines," in *Advances in Kernel Method: Support Vector Learning*, Scholkopf, Burges, and Smola, Eds. Cambridge, MA: MIT Press, 1998, pp. 185–208.

[13] E. Osuna, R. Freund, and F. Girosi, "Improved training algorithm for support vector machines," in *Proc. IEEE NNSP'97*, 1997.

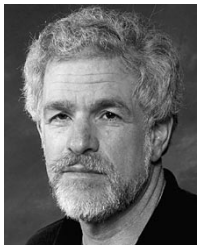
[14] W. W. Cohen, "Fast effective rule induction," in *Proc. 12th Int. Conf. Machine Learning.* San Mateo, CA: Morgan Kaufmann, 1995, pp. 115–123.

[15] R. E. Schapire, Y. Singer, and A. Singhal, "Boosting and Rocchio applied to text filtering," in *Proc. 21st Annu. Int. Conf. Inform. Retrieval, SIGIR*, 1998.

[16] T. Joachims, "A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization," in *Proc. 14th Int. Conf. Machine Learning*, D. Fisher, Ed. San Mateo, CA: Morgan Kaufman, 1997.

[17] W. W. Cohen and Y. Singer, "Context-sensitive learning methods for text categorization," in *Proc. 19th Annu. Int. ACM SIGIR Conf. Res. Development Inform. Retrieval*, 1986, pp. 307–315.

[18] D. D. Lewis, R. E. Schapire, J. P. Callan, and R. Papka, "Training algorithms for linear text classifiers," in *Proc. 19th Annu. Int. ACM SIGIR Conf. Res. Development Inform. Retrieval*, H.-P. Frei, D. Harman, P. Schauble, and R. Wilkinson, Eds. New York: ACM, 1996, pp. 298–306.



Harris Drucker (S'67–M'68–SM'87) received the Ph.D. degree in electrical engineering from the University of Pennsylvania, Philadelphia, in 1967.

He is currently Professor of Electronic Engineering at Monmouth University, West Long Branch, NJ. He has consulted for AT&T, Lucent Technologies, and Bell Laboratories in machine learning. He is coauthor of the first paper that showed a practical implementation of a boosting algorithm, published in 1993. His implementation of boosting algorithms using neural networks and classification trees are present in many commercial check readers.

Donghui Wu (S'99) received the B.S. degree in 1989 from Nanjing Univeristy, China, and the M.S. degree in 1997 from Rensselear Phlytechnic Institute, Troy, NY.

He was a Research and Development Software Engineer at Nanjing Research Institute of Electrical Engineering, China, from 1989 to 1994, where he worked on decision support systems, management information systems, and scientific computing. His current research interests include data mining, machine learning, statistical learning, support vector machines, and operation research.

Vladimir N. Vapnik, for a photograph and biography, see this issue, p. 999.