Spam Filtering using Inexact String Matching in Explicit Feature Space with On-Line Linear Classifiers

D. Sculley, Gabriel M. Wachman, and Carla E. Brodley
Department of Computer Science, Tufts University Medford, MA 02155, USA
{dsculley, gwachm01, brodley}@cs.tufts.edu

Abstract

Contemporary spammers commonly seek to defeat statistical spam filters through the use of word obfuscation. Such methods include character level substitutions, repetitions, and insertions to reduce the effectiveness of word-based features. We present an efficient method for combating obfuscation through the use of inexact string matching kernels, which were first developed to measure similarity among mutating genes in computational biology. Our system avoids the high classification costs associated with these kernel methods by working in an explicit feature space, and employs the Perceptron Algorithm using Margins for fast on-line training. No prior domain knowledge was incorporated into this system. We report strong experimental results on the TREC 2006 spam data sets and on other publicly available spam data, including near-perfect performance on the TREC 2006 Chinese spam data set. These results invite further exploration of the use of inexact string matching for spam filtering.

1. Introduction

Effective spam filtering – automatically blocking unwanted email from the user – is now widely recognized as one of the most important problems in electronic communication. In recent years, statistical spam filters, based on techniques from the field of machine learning, have achieved significant progress. However, as seen in the 2005 TREC Spam Filtering Competition (Cormack and Lynam, 2005b), even state of the art methods yield imperfect results.

Traditional statistical spam filters are built on a foundation of Naive Bayes classification on word-level features (Graham, 2002; Metsis et al., 2006), a combination that has proven efficient and reasonably effective. Recently, character-level features, such as those used by the PPM classifier (Bratko and Filipic, 2005), have improved performance. Character level features offer resistance to standard spam attacks such as *obfuscation*, the practice of intentionally misspelling words to defeat word-based spam filters (Wittel and Wu, 2004). In this paper, we extend character-level features to include methods of *inexact string matching* with the goal of further combating obfuscation attacks. These methods have been applied in computational biology, in the form of inexact string matching kernels with support vector machines on genomic data (Leslie et al., 2002a,b; Leslie and Kuang, 2004), and have also been used for text classification (Lodhi et al., 2002).

Naive application of these methods carries a high computational cost, which is problematic for industrial-scale spam filtering. Our approach reduces this cost in three ways. First, we use *fixed* variants of the wildcard and gappy inexact string matching kernels, and conflate the ideas of wildcards and gaps. Second, we map inexact string matching features into an explicit feature space, enabling fast classification. Finally, we use an on-line linear classifier, the Perceptron Algorithm with Margins (PAM) (Krauth and Mézard, 1987; Li et al., 2002), which has fast training time and good resistance to noise (Khardon and Wachman, 2005).

The remainder of this paper provides details on the methods of efficient inexact string matching (Section 2) and the PAM classifier (Section 3). Experimental results in Section 4 show that this approach has given top level results on the public Chinese email data set from the 2006 TREC Spam Filtering competition.

Viagra	VIAGRA	Viiagrra	viagra	visagra
Vi@gra	Viaagrra	Viaggra	Viagraa	Viiaagra
Via-ggra	Viia-gra	V1AAGRRA	Viiagra	Via-gra
Vi graa	V iagra	via gra	Viagrra	V&Igra
VIAgra	V|agra	Viaaggra	vaigra	V'iagra

Figure 1: Obscured Text. These are the 25 most common variants of the word 'Viagra' found in the 2005 TREC spam data set, illustrating the problem of word obfuscation and tokenization. There are hundreds of other variants for this word alone.

2. Inexact String Matching

Inexact string matching offers one possible solution to the problem of filtering obscured text. Spam filters that rely on traditional word-based features may be defeated by the spam attack of obfuscation, in which words are intentionally obscured through the use of misspellings, character substitutions, insertions, and deletions (Wittel and Wu, 2004). While a case has been made that statistical spam filters will learn to recognize common obfuscations over time, given enough training data (Graham, 2002), there are a very large number of possible obfuscations for any given word, each of which may appear infrequently. Furthermore, consider the effect of joining two words by removing their intervening space: `likethis`. The brute force approach to learning variations is now responsible for a feature space that is suddenly quadratic in the number of dictionary entries, and the combinatorics of this problem quickly grow as more substitutions, insertions, and deletions are employed. Indeed, obfuscations altering of several characters are already common. Evidence for this can be found in the TREC 2005 spam data set (see Figure 1.)

Intuitively, it seems reasonable to expect that successful methods for genomic data would also have success in the domain of spam, as these domains have similar characteristics. Genomic data are represented as strings in which character level substitution, insertion, and deletion are common for evolutionary reasons. The use of inexact string matching *kernels* with Support Vector Machines has proven effective for learning from genomic data (Leslie et al., 2002a,b; Leslie and Kuang, 2004). However, industrial spam filtering (and the 2006 TREC Spam Filtering Track) has severe limits on computational cost that precludes the direct application of inexact string matching kernels.

To address this cost, we first propose efficient fixed variants of two inexact string matching methods, the *wildcard* kernel and the *gappy* kernel, both of which are built off a foundation of k-mers. We map these kernels to explicit feature space to allow fast classification with the PAM classifier described in Section 3.

2.1 k-mers

The use of character level k-mers is one basic method of inexact string matching (Leslie et al., 2002a). A k-mer is a sequence of k contiguous characters drawn from the alphabet Σ. The set S of all possible k-mers in Σ defines a space X with $|\Sigma|^k$ dimensions, with each unique k-mer $a \in S$ indexing a unique dimension in X. A string s may be represented as a vector $x \in X$ by mapping $\forall a \in S, x_a = c(s, a)$, where $c(s, a)$ returns the number of times that the k-mer a appears in string s, with overlaps allowed.

In spam filtering, there have been several results showing that binary features may be more effective than count-based features (Drucker et al., 1999; Metsis et al., 2006), perhaps because binary features offer greater resistance to the *good word attack* (Wittel and Wu, 2004). A string

may be represented as a binary k-mer vector $x \in X$, by mapping $\forall a \in S, x_a = b(s, a)$, where $b(s, a) = 1$ if a appears anywhere in s, and $b(s, a) = 0$ otherwise.

Even this simple feature space allows powerful inexact string matching. By removing position information, the k-mer feature space allows the insertion or deletion of any number of characters between k-mers. Furthermore, because the k-mers are overlapping, they implicitly capture some level of sequence information that is lost with the standard bag of words model.

The choice of k is an important parameter. Too small a value of k creates ambiguous features, while too high a value of k makes the chance of finding exact matches between strings improbable. Indeed, in a spam classification setting, the optimal value of k may be language dependent, or may vary with the amount of expected obfuscation within the text. Thus, in our model, we choose to focus on a range of k values. An (i, j) k-mer mapping creates a space in which there is a dimension for all possible k-mers, where $i \leq k \leq j$. Indeed, the best spam filter from the 2005 TREC Spam Filtering Track was a PPM-filter (Bratko and Filipic, 2005), which implicitly uses a feature space of (i, j) k-mers (Sculley and Brodley, 2006).

2.2 Wildcards and Gaps

In computational biology, it has been found that k-mers alone are not expressive enough to give optimal classification performance on genomic data. For example, a k-mer feature space will have reduced effectiveness on substrings in which at least two character substitutions, insertions, or deletions occur no more than k positions apart. Two forms of inexact string matching kernels address this issue: *wildcard* kernels and *gappy* kernels (Leslie and Kuang, 2004), which allow k-mers to match even if there have been a small (specified) number of insertions, deletions, or substitutions.

Wildcards. The (k, w) wildcard kernel maps each k-mer to a set of k-mers in which up to w "wildcard" characters replace the characters in the original k-mer (Leslie and Kuang, 2004). For example, the $(3, 1)$ wildcard mapping of the k-mer `abc` is the set {`abc, *bc, a*b, ab*`}. The wildcard character is a special symbol that is allowed to match with any other character. Naturally, allowing wildcards increases computational cost. In our testing with spam data, however, we have found that allowing a *fixed* wildcard variant gives equivalently strong results to the standard wildcard kernel. A fixed (k, p) wildcard mapping maps a given k-mer to a set of two k-mers: the original, and a k-mer with a wildcard character at position p. Note that we designate the first position in a string as position 0. Thus, the fixed $(3, 1)$ wildcard mapping of `abc` is {`abc, a*c`}. This fixed mapping gives more flexibility to the k-mer feature space, but only increases the size of the feature space by a constant factor of 2.

Gaps. The (g, k) gappy kernel (where $g \leq k$), allows g-mers to match with k-mers by inserting $k - g$ gaps into the g-mer (Leslie and Kuang, 2004). Note that this is equivalent to allowing k-mers to match with g-mers by deleting up to $k - g$ characters from the k-mer. Thus, the $(2, 3)$ gappy mapping of the string `acbd` includes positive values for features indexed by {`acb, cbd, ac, ab, cb, cd, bd`} . As with the wildcard mappings, we reduce computational cost with a *fixed* (k, p) gappy variant, in which a k-mer is is mapped to a set of k-mers: the original k-mer, and a k-mer in which the character at position p has been removed. For clarity, we define a function REMOVE(a, p) which removes a single character from string a at position p. Thus, if a is a k-mer, REMOVE returns a $(k - 1)$-mer.

Merging Gaps and Wildcards. The function of gaps and wildcards are similar, with the difference that implementing the gappy mappings with the REMOVE function changes the length of the k-mers. However, our (i, j) k-mer space contains k-mers of different lengths. This allows us to reduce computational cost by defining the wildcard character as *a character of length zero.* Furthermore, we used the fixed variants, and only allow wildcard replacement and gaps to occur at

```
GIVEN (i, j, p) MAP STRING s TO VECTOR x:

FOR k FROM i TO j DO:
        FIND SET K OF ALL k-MERS IN s
        FOR EACH k-MER a ∈ K DO:
                x_a := 1
                b := REMOVE (a,p)
                x_b := 1
        DONE
DONE
NORMALIZE: x := x / ||x||
```

Figure 2: Pseudo code for fixed (i, j, p) inexact string feature mapping.

position p in all k-mers. We call this a fixed (i, j, p) inexact string feature mapping, pseudo-code for which is given in Figure 2. The resulting feature space has the following properties:

- k-mers may match other k-mers exactly
- $(k + 1)$-mers may match other $(k + 1)$-mers with a mismatch at position p.
- k-mers may match $(k + 1)$-mers with a gap at position p.

Thus, this feature space has a measure of robustness to a number of obfuscation attacks, including character substitution, insertion, and deletion.

Normalization. Because email messages may be of varying lengths, we normalize all data vectors using the Euclidean norm. Additionally, normalizing provides some resistance to the *sparse data attack* (Wittel and Wu, 2004), which attempts to defeat a spam filter using very short messages.

2.3 Explicit Feature Mapping

For efficient classification, we represent the features in the inexact feature space explicitly, using a sparse feature/value representation. This approach is similar to that recommended by Sonnenburg et al. (2005), who show that explicit feature mapping is preferable to implicit feature mapping (using, for example, suffix trees) for support vector machine training and classification of strings, when using small k-mers. This allows the PAM classifier to classify a new data example by computing a single (sparse) inner product. On-Line training is equally fast, requiring only a single (sparse) vector addition. Thus, although the feature space is of high dimensionality, the sparsity of the explicit feature vectors allows fast training and classification with manageable space cost.

3. On-Line Linear Classifiers

In the previous section, we showed an efficient method for mapping email messages to explicit vectors in an (i, j, p) inexact string feature space. Because the vast majority of machine learning methods are designed to accept explicit feature vectors as input, we have a wide range of training and classification methods at our disposal, including Naive Bayes classifiers and support vector machines. In this competition, we choose to classify the data with the PAM classifier (Krauth and Mézard, 1987; Li et al., 2002), which learns a linear classifier with tolerance for noise (Khardon and Wachman, 2005).

GIVEN: SET OF EXAMPLES AND THEIR LABELS
$\mathcal{Z} = ((x_1, y_1), \ldots, (x_m, y_m)) \in (\mathbb{R}^n \times \{-1, 1\})^m$, τ

INITIALIZE $w := 0^n$
FOR EVERY $(x_j, y_j) \in \mathcal{Z}$ DO:
    IF $y_j(\langle w, x_j \rangle) < \tau$
        $w := w + \eta y_j x_j$
DONE

Figure 3: The On-Line Perceptron Algorithm with Margins

We chose to use PAM for several reasons. First, it offers fast on-line training and classification. Second, strong performance has been achieved by other linear classifiers on spam such as a variant of logistic regression (Goodman and Yin, 2006) and linear support vector machines (Drucker et al., 1999; Rios and Zha, 2004); since Khardon and Wachman (2005) demonstrate experimentally that PAM is competitive with SVM on many data sets, it is reasonable to expect PAM to perform well on spam. Third, it has been shown that while generative models such as Naive Bayes may have steeper (faster) learning curves, discriminative models such as linear classifiers have lower asymptotic error (Ng and Jordan, 2002). In the remainder of this section, we will review the basic Perceptron Algorithm, and explore relevant features of the PAM variant that makes it an attractive choice for on-line spam filtering.

3.1 The Perceptron Algorithm

The perceptron algorithm (Rosenblatt, 1958) takes as input a set of training examples in $\mathbb{R}^n$ with labels in $\{-1, 1\}$. Using a weight vector, $w \in \mathbb{R}^n$, initialized to 0^n, it predicts the label of each training example x to be $y = sign(\langle w, x \rangle)$. The algorithm adjusts w on each misclassified example by an additive factor.[1] An upper-bound on the number of mistakes committed by the perceptron algorithm can be shown both when the data are linearly separable (Novikoff, 1962) and when they are not linearly separable (Freund and Schapire, 1999).

3.2 Perceptron Algorithm with Margins (PAM)

The classical perceptron attempts to separate the data but has no guarantees on the separation margin obtained. The Perceptron Algorithm with Margins (PAM) (Krauth and Mézard, 1987; Li et al., 2002) attempts to establish such a margin, τ, during the training process. Following work on support vector machines (Boser et al., 1992) one may expect that providing the perceptron with higher margin will add to the stability and accuracy of the hypothesis produced (Cristianini and Shawe-Taylor, 2000).

To establish the margin, instead of only updating on examples for which the classifier makes a mistake, PAM also updates on x_j if $y_j(\langle x_j, w \rangle) < \tau$. When the data are linearly separable, the margin of the classifier produced by PAM can be lower-bounded (Krauth and Mézard, 1987; Li et al., 2002). The algorithm is summarized in Figure 3.

It is important to select a reasonable value for τ. If τ is too large, the algorithm will not be able to find a stable hypothesis until the norm of w grows large enough at which point individual updates will have little effect; if τ is too small, the margin of the hypothesis will be small and the performance may suffer.

1. The classical perceptron includes a bias term θ. In our preliminary experiments, we found our simplified version to perform as well as the classical one on spam data.

Table 1: Parameter settings for tufS filters.

FILTER	$(i,\ j,\ p)$	τ
TUFS1F	(2, 4, 1)	100
TUFS2F	(2, 5, 1)	100
TUFS3F	(2, 6, 1)	100
TUFS4F	(2, 7, 1)	100

The learning rate, η, controls the extent to which $\mathbf{w}$ changes on a single update; too large of a value causes the algorithm to make large fluctuations, and too small of a value results in slow convergence to a stable hypothesis and hence many mistakes. Note that we can eliminate η in our case by scaling τ by $\frac{1}{\eta}$, but we leave it in for discussion as it is part of the classical algorithm.

PAM enables fast classification and on-line training. The classification time of PAM is dominated by the computation of the inner product $\langle w, x \rangle$. A naive inner product takes $O(m)$ time, where m is the number of features. When x is sparse, containing only $s \ll m$ non-zero features, we can compute this inner product in $O(s)$ time. Similarly, the time for an on-line update is dominated by updating the hypothesis vector w, which can be done in $O(s)$ time as well. Moreover, PAM does not require training updates for well-classified examples. Thus, the total number of updates is likely to be significantly less than the total number of training examples.

In comparison with Naive Bayes and linear support vector machines, PAM has the same classification cost $O(m)$, but will have lower overall training time than either method. Naive Bayes requires $O(m)$-cost updates on every example in the training set, while PAM does not train on well classified examples. A linear support vector machine that trains in $O(sn)$ time has recently been proposed (Joachims, 2006), but in practice the constant is quite high and PAM training is significantly faster.

4. Experimental Results

In this section, we report experimental results on spam classification for the 2006 TREC Spam Filtering track. Our general approach was to map email messages to feature vectors, using the fixed (i, j, p) inexact string feature space described in Section 2. On-Line training and classification were performed using the PAM algorithm, as described in Section 3. We set η to 0.1, and τ to 100.

Experimental Setup. We tested this filter at four settings, as shown in in Table 1. Each filter was given a unique setting of the maximum k-mer size, specified by j in the fixed (i, j, p) inexact string feature space. The value of $\tau = 100$ was chosen by parameter search, using the `SpamAssassin` public corpus[2] as a tuning set. No preprocessing of the email messages was performed. We simply used the first n characters from the raw text of the email (including any header information and attachments) as an input string, and created a sparse feature vector from that string. Our initial filters used a maximum of 200K characters, and performed successfully on initial tests for the trec05p-1 data set (Cormack and Lynam, 2005a), and on all private data sets from the 2006 competition. However, two filters with larger maximum k-mer sizes failed to complete testing on the `pe{i,d}` data set, due to lack of memory. When we reduced the maximum input string length from 200K to the first 3000 characters, this problem was eliminated – and performance for all filters improved. For example, on the `pei` tests, TUFS1F improved from 0.062 to 0.040 on (1-ROCA)% using the first 3000 characters. However, note that the official results for the 2006 competition were with TUFS filters using the first 200K characters.

2. http://spamassassin.apache.org/publiccorpus

FILTER	PCI	PCD	PEI	PED	X2	X2D	B2	B2D	TREC05P-1
BEST	0.003	0.01	0.03	0.1	0.03	0.03	0.1	0.3	0.019
TUFS1F	0.002	0.008	0.060	0.211	0.095	0.199	0.390	0.836	0.020
TUFS2F	0.003	0.010	0.060	0.203	0.069	0.145	0.338	0.692	0.018
TUFS3F	0.004	0.012	0.042*	0.132*	0.063	0.126	0.335	0.614	0.018
TUFS4F	0.005	0.011	0.041*	0.136*	0.075	0.131	0.320	0.570	0.017
MEDIAN	0.03	0.3	0.3	0.3	0.1	0.1	0.3	1	0.4

Table 2: Summary of Results on (1-ROCA)% measure. Results are reported for the tests on TREC 2006 public Chinese corpus `pci` and `pcd`, public English corpus `pei` and `ped`, Mr. X private corpus `x2` and `x2d`, b2 private corpus `b2` and `b2d`, and the 2005 TREC public corpus `trec05p-1`. Results on sets ending in `d` are for delayed feedback experiments; others are for incremental learning experiments. Results marked with * were produced using variant that only considered first 3000 characters, rather than the first 200K.

Results. For our initial tests, run before the 2006 competition, we tested our filters on the `trec05p-1` data set, and found that our filters were competitive with the best filters from the TREC 2005 Spam Filtering track (see Table 2) (Cormack and Lynam, 2005b).

Our results from the TREC 2006 competition were strong (see Table 2). In particular, our method achieved extremely strong performance on the public corpus of Chinese email, with a steep learning curve and a $1 - ROCA$ score of .0023 for tufS1F , and .0031 for tufS2F on the incremental task, `pci`, which the initial report suggests are at or near the top level of performance for the 2006 competition, and are an order of magnitude better than the reported median. The results for the delayed learning task on Chinese email, `pcd`, were also very strong. Although absolute performance on the delayed task was reduced, due to the more difficult nature of training on delayed feedback, our methods still gave top-level results. However, these results must be taken with a grain of salt. At the TREC 2006 conference meeting, it was noted that results on this corpus of Chinese email were very strong with many filters. This raises the possibility that this corpus may be a flawed benchmark data set.

In general, our results on other data sets were encouraging, giving second place aggregate results in the 2006 TREC spam competition. On the public English corpus, our methods gave results well above the median for both the incremental learning task `pei` and the delayed learning task `ped`. Our results on the two private corpora, `x2` and b2, were less strong. At present, we do not know the nature of these data sets, and cannot yet speculate on the cause of the relative decrease in performance on these data sets.

5. Discussion

These results raise a number of interesting questions. First, we would like to investigate the use of our filter on other data sets of Chinese email. We believe our methods should be robust to the several encoding schemes for Chinese email, each of which employs multiple characters per word. Because Chinese does not employ a phonetic alphabet, short character-level sequences may carry more information in this language than they do in English. We conjecture that it is this fact that enables the strong results achieved by our methods. Indeed, one of the strengths of character-level pattern recognition for spam filtering is that no assumptions are made about language or encoding scheme. Future work should include additional experiments comparing performance on phonetic and ideographic emails in a range of other languages, to ensure that this conjecture holds true.

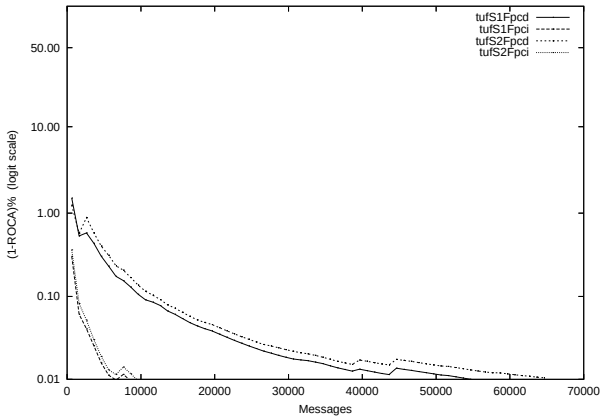

Figure 4: Learning curve for public Chinese corpus: incremental learning `pci` and delayed learning `pcd`, with filters TUFS1F and TUFS2F

.

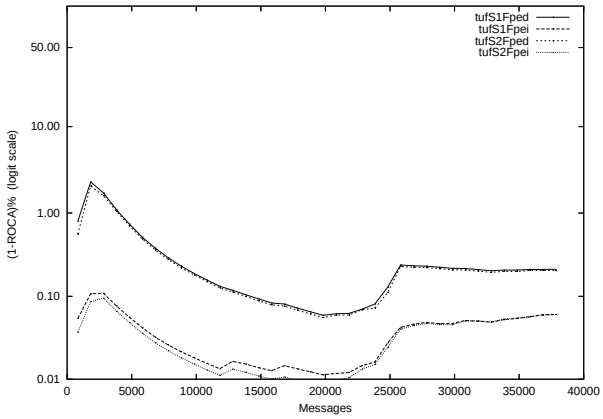

Figure 5: Learning curve for public English corpus: incremental learning `pei` and delayed learning `ped`, with filters TUFS1F and TUFS2F.

Secondly, we draw attention to the learning curve results for the `pei` and `ped` experiments, which hit very low (*i.e.,* good) levels of the $1 - ROCA$ measure between 10,000 and 25,000 messages, and then rises sharply to a plateau of lower performance for both the incremental and delayed learning tasks. This is not the convergent behavior we would expect to see after many thousands of examples, and requires investigation. For example, the problem could be caused by concept drift, an over-correction for an anomalous email, or perhaps an email with a noisy label. These issues might be addressed by varying the learning rate η or the margin τ as the number of seen training examples increases as in Gentile (2001).

Third, we note that our initial choice of filtering based on the first 200K characters in the email message was not optimal. Later tests using only the first 3,000 characters from each email not only reduced both memory usage and computational cost, but also improved performance on all versions of our filter.

Overall, the fixed (i, j, p) inexact string features, represented as sparse explicit feature vectors, used in conjunction with the on-line linear classifier PAM has given strong performance on a number of tests. These results were obtained using inexact string matching in a fairly naive way, without taking domain knowledge into account. In the future, we plan on investigating the use of inexact string matching on email specific features, such as the subject heading and sender information.

References

B. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 144–152, 1992.

A. Bratko and B. Filipic. Spam filtering using compression models. Technical Report IJS-DP-9227, Department of Intelligent Systems, Jozef Stefan Institute, L jubljana, Slovenia, 2005.

G. V. Cormack and T. R. Lynam. Spam corpus creation for TREC. In *Proceedings of the Second Conference on Email and Anti-Spam (CEAS)*, 2005a.

G. V. Cormack and T. R. Lynam. TREC 2006 spam track overview. In *The Fourteenth Text Retrieval Conference (TREC 2005) Proceedings*, 2005b.

N. Cristianini and J. Shawe-Taylor. *An introduction to support vector machines.* Cambridge University Press, 2000.

H. Drucker, V. Vapnik, and D. Wu. Support vector machines for spam categorization. *IEEE Transactions on Neural Networks*, 10(5):1048–1054, 1999.

Y. Freund and R. Schapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 37:277–296, 1999.

C. Gentile. A new approximate maximal margin classification algorithm. *Journal of Machine Learning Research*, 2:213–242, 2001.

J. Goodman and W. Yin. Online discriminative spam filter training. In *Proceedings of the Third Conference on Email and Anti-Spam (CEAS)*, 2006.

P. Graham. A plan for spam. 2002. Available at http://www.paulgraham.com/spam.html.

T. Joachims. Training linear svms in linear time. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 217–226, 2006.

R. Khardon and G. Wachman. Noise tolerant variants of the perceptron algorithm. Technical report, Tufts University, 2005. To appear in Journal of Machine Learning Research; available at http://www.cs.tufts.edu/tr/techreps/TR-2005-8.

W. Krauth and M. Mézard. Learning algorithms with optimal stability in neural networks. *Journal of Physics A*, 20(11):745–752, 1987.

C. Leslie, E. Eskin, and W. S. Noble. The spectrum kernel: A string kernel for svm protein classification. In *Pacific Symposium on Biocomputing*, pages 566–575, 2002a.

C. Leslie, E. Eskin, J. Weston, and W. S. Noble. Mismatch string kernels for svm protein classification. In *Neural Information Processing Systems*, pages 1417–1424, 2002b.

C. Leslie and R. Kuang. Fast string kernels using inexact matching for protein sequences. *Journal of Machine Learning Research*, 5:1435–1455, 2004.

Y. Li, H. Zaragoza, R. Herbrich, J. Shawe-Taylor, and J. Kandola. The perceptrom algorithm with uneven margins. In *International Conference on Machine Learning*, pages 379–386, 2002.

H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins. Text classification using string kernels. *Journal of Machine Learning Research*, 2:419–444, 2002.

V. Metsis, I. Androutsopoulos, and G. Paliouras. Spam filtering with naive bayes – which naive bayes? *Third Conference on Email and Anti-Spam (CEAS)*, 2006.

A. Ng and M. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes, 2002.

A. B. Novikoff. On convergence proofs on perceptrons. *Symposium on the Mathematical Theory of Automata*, 12:615–622, 1962.

G. Rios and H. Zha. Exploring support vector machines and random forests for spam detection. In *Proceedings of the First Conference on Email and Anti-Spam (CEAS)*, 2004.

F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–407, 1958.

D. Sculley and C. E. Brodley. Compression and machine learning: A new perspective on feature space vectors. *DCC: Data Compression Conference*, 0:332–332, 2006. ISSN 1068-0314.

S. Sonnenburg, G. Rätsch, and B. Schölkopf. Large scale genomic sequence svm classifiers. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pages 848–855, 2005.

G. L. Wittel and S. F. Wu. On attacking statistical spam filters. *CEAS: First Conference on Email and Anti-Spam*, 2004.