Relaxed Online SVMs in the TREC Spam Filtering Track

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

Abstract

Relaxed Online Support Vector Machines (ROSVMs) have recently been proposed as an efficient methodology for attaining an approximate SVM solution for streaming data such as the online spam filtering task. Here, we apply ROSVMs in the TREC 2007 Spam filtering track and report results. In particular, we explore the effect of various sliding-window sizes, trading off computation cost against classification performance with good results. We also test a variant of fixed-uncertainty sampling for Online Active Learning. The best results with this approach give classification performance near to that of the fully supervised approach while requiring only a small fraction of the examples to be labeled.

1. Introduction

It has been nearly ten years since Support Vector Machines (SVMs) were shown to give state of the art performance on high-dimensional classification problems in general, and text classification in particular [11, 12].[1] In that time, the machine learning community has repeatedly suggested that SVMs are a strong choice for content-based spam filtering [7, 13, 18]. However, to our knowledge SVM methodology has yet to be applied in a large-scale spam filtering application, such as an industrial scale email system. Spam filtering practitioners have preferred to apply other machine learning techniques. These methods include several variants of the Naive Bayes classifier [1, 9, 10, 16], linear classifiers such as Logistic Regression [8] and Perceptron Algorithm variants [22], compression-based methods [2, 3], and ensemble methods [15].

Anti-spam practitioners have avoided SVMs for two reasons. First, until recently it was considered unclear whether SVMs actually gave state of the art performance on online spam filtering tasks [4, 5]. However, experiments on the benchmark spam filtering data sets from TREC 2005 and TREC 2006 have been reported that show SVMs indeed give state of the art classification performance on these tasks, exceeding that of all other single classifiers, and well within confidence bounds of the mark set by ensemble methods [21].

Second, SVMs carry high training cost that make them impractical in large-scale settings. Relaxed Online SVMs (ROSVMs) have been proposed that compute an approximate SVM solution at greatly reduced expense [21]. ROSVMs use a sliding window to cap the size of the classical SVM optimization problem for streaming data, such as a (potentially limitless) series of messages to be classified. ROSVMs further reduce cost by updating their model less frequently than classical SVMs, and by allowing early termination by the

1. Space restrictions preclude a general discussion of SVMs in this paper. Those wishing a general review of SVM methods are referred to the excellent text by [19].

iterative SVM solver. ROSVMs have given near state of the art performance on the TREC 2005 and 2006 public data sets; we now test them on TREC 2007 data and tasks.

In the remainder of this paper, we review the ROSVM algorithm and its implementation. We detail the Online Active Learning approach used by our ROSVM in the current tests. We report results on the TREC 2007 Spam Filtering tasks on the `trec07p` data set, and compare these to baseline results given by one standard open-source spam filter. Our concluding discussion focuses on identifying open questions in spam filtering research.

2. Relaxed Online SVMs

The ROSVM methodology gives an approximate Online SVM solution at much reduced cost that does not grow with the amount of data that has been encountered [21]. This is accomplished with three base strategies, which we review briefly in this section.

Sliding Window. Recall that SVMs classify a new example $\mathbf{x}$ with the function:

$$f(\mathbf{x}) = \sum_{i}^{m} \alpha_i y_i < \mathbf{x}_i, \mathbf{x} > +b$$

Here, each of the n training examples $\mathbf{x}_i$ has an associated label y_i and weight α_i. An SVM solver such as Platt's SMO finds optimal values for each of the n alpha values by solving a now classic quadratic optimization problem [17, 19]. As the number of examples in the training data grows, the number of weights to set also grows and the cost of training increases.

In the sliding window strategy, we restrict the size of the optimization problem by only optimizing the weights for the p most recent examples. That is, at step n (where $n > p$) we treat the set of examples $\{\mathbf{x}_{n-p+1}, .., \mathbf{x}_n\}$ as our set of training data, and only optimize the values for $\{\alpha_{n-p+1}, .., \alpha_n\}$. Each weight value $\alpha_i \in \{\alpha_1, .., \alpha_{n-p}\}$ associated with an example outside the sliding window is fixed at the value it was most recently given when $\mathbf{x}_i$ was still in the sliding window. This reduces the size of the optimization problem from one that increases with n to one that is bound by p. Experiments reported in [21] showed that near state of the art results can be produced with a relatively small lookback buffer of size 5000 or less, dramatically reducing computational cost.

Our submissions to TREC 2007 explore three sizes of lookback buffer, with p set to 500, 1000, and 5000, in order to explore the tradeoff between computational cost and classification performance.

Redefining *Well Classified*. For Online SVMs, it is only necessary to update the model when an example is encountered that is not well classified. The formal definition of *well classified* in this setting is defined by the Karush-Kuhn-Tucker (KKT) conditions, which will be satisfied by an Online SVM on all new examples $\mathbf{x}_i$ except those for which $y_i f(\mathbf{x}_i) \leq 1$. Intuitively, we must re-train an Online SVM whenever we encounter an example that lies within the margins or which is mistakenly classified. This will guarantee that our model is consistent with all the data seen so far.

The ROSVM approach relaxes this requirement by re-defining what it means to be well classified. We set a parameter m, where $0 \leq m \leq 1$, and only re-train when we find an example for which $y_i f(\mathbf{x}_i) \leq m$. That is, we allow a model to continue classifying so long as

it has been shown to be only slightly inconsistent with the data seen so far. This approach reduces cost by reducing the number of times we update the model. Experiments reported in [21] show that, for spam data, a value of $m = 0.8$ gives results indistinguishable from state of the art results at half the cost, and values as low as $m = 0$ performs nearly as well with additional savings. In our submissions, we use $m = 0.8$

Reducing Iterations Platt's SMO is an iterative solver, making repeated passes over the data set to find optimal alpha values [17]. The third cost-reducing strategy is to enforce a maximum number of iterations in the outer loop of the SMO algorithm (complete SMO pseudo-code is given in [17]), limiting the total number of passes over the training data for each update. This is similar to setting a loose tolerance for convergence, but is perhaps a more Draconian approach. Our previous tests have shown that restricting the maximum number of iterations to even very low values has negligible effect on classification performance for online spam filtering tasks [21]. This is because there are many successive training updates over the course of online filtering. In our submissions, we set the maximum number of SMO outer-loop iterations to 1.

3. Implementation Issues

In this section, we explore some of the implementation issues involved in making the ROSVMs run fast enough to participate in the TREC spam filtering task.

Binary 4-*mer* Feature Space. Our tests have found that a binary 4-*mer* feature space gives consistently best results on spam data. This is a high dimensional feature space, containing a dimension for each possible substring of length exactly 4 drawn from the alphabet of all possible single-byte characters. This approach allows inexact string matching [14], which provides a measure of robustness against spammer techniques such as word obfuscation [23]. This feature space gives significantly improved performance over a more traditional word-based feature space [21]. This choice of feature space also allows the filtering method to be language-independent, and to learn from non-textual data such as attachments.[2] Our tests have agreed with results reported by [7] and [16], finding that binary valued features out-perform count-based and TFIDF-based feature scoring methods on the spam filtering data sets that we have tested. We map binary 4-*mer* features for the first 3000 characters of a given message string. The feature vectors are normalized using the Euclidian norm during classification and training.

The drawback to this representation is the dimensionality is even higher than standard text-classification problems. In the worst case, there could be 2^{32} unique features present in the data. In practice, we have found 3,700,809 unique 4-*mers* in the `trec07p` data set, 3,332,440 unique 4-*mers* in the `trec05p-1` data set, and 1,424,606 unique 4-*mers* in the `trec06p` data set when considering the first 3,000 characters of each message.[3] Thus, the feature space remains sparse – the vast majority of possible 4-*mers* features do not occur.

2. Interestingly, we have also found that using additional inexact-string matching features, such as gappy features and wildcard features [22] do not give added performance over the simpler 4-*mers* with ROSVMs on spam data.
3. Gordon Cormack has suggested hashing these features to reduce the dimensionality of this problem, which is a suggestion we will explore in upcoming work.

Sparse Binary Inner Products For a linear SVM, computing inner products for two vectors is the core task in an SVM solver such as Platt's SMO [17]. We optimize our ROSVM for the special case of computing inner products on sparse binary vectors, allowing several optimizations over the traditional linked-list implementation of sparse vectors. The inner products are fast enough that a cache of inner-product values [19] is not required, greatly reducing the memory footprint of the system.

In our implementation, each example is stored as an array of integers, each integer indexing a particular non-zero feature. The array of feature indices is stored in sorted order, from lowest to highest. This allows a binary inner product to be computed with no pointer-redirects (as occur with linked-list implementations), excellent memory locality, and using only comparison and increment operators.

Alpha Seeding. As reported by [6], alpha seeding can speed up the training time of SVMs considerably. The main idea is that starting with a good initial guess α_i' that is close to the optimal value α_i for each example $\mathbf{x}_i$ dramatically reduces the amount of work required for an iterative SVM solver, such as SMO, to converge to the optimal solution. Because we are working in an online setting where updates happen incrementally, we use the assumption that the set of alpha values found in the previous update is a good initial starting point for the next update. In practice, this means that we can simply keep the alpha values in memory after the SVM solver has finished updating its hypothesis.

Sliding Window. We use a ring buffer store the p most recent examples in the data stream. Each time we encounter a new example, it takes the place of the oldest example in the buffer. The alpha value of the new example is initially set to 0. All SVM updates are done using this buffer as the training data set.

3.1 Linear Hypothesis.

Because we deal only with the linear kernel, we follow the suggestion of [17] and rewrite the dual form SVM classification function to the primal form, using a single aggregate weight vector to store the hypothesis:

$$f(\mathbf{x}) =< \mathbf{w}, \mathbf{x} > +b$$

Naturally, $\mathbf{w} = \sum_i^m \alpha_i y_i \mathbf{x_i}$. Because the SMO algorithm uses the current hypothesis to determine convergence and to select candidates for optimization, it is necessary to maintain a current aggregate weight vector at all times. Thus, every time an alpha value α_i for an example $\mathbf{x}_i$ is changed during training to a new value α_i', the weight vector is updated with:

$$\mathbf{w} = \mathbf{w} + (\alpha_i' - \alpha_i) y_i \mathbf{x}_i$$

When an example $\mathbf{x}_j$ leaves the sliding window of examples, we do not alter the weight vector $\mathbf{w}$, but we do remove the example from the ring buffer of examples and no longer update its alpha value. Thus, α_j is fixed permanently at its most recent optimal value.

4. Online Active Learning

This year's TREC Spam Filtering track included an Online Active Learning task, rather than the previous pool-based Active Learning task from 2006. The structure of this task was

FILTER	SLIDING WINDOW	UPDATE MARGIN	MAX ITERS	COST C	WALL CLOCK (`full`)
TFTS1F	500	0.8	1	100	224s
TFTS2F	1000	0.5	1	100	214s
TFTS3F	5000	0.8	1	100	3087s

Table 1: Parameter settings and wall-clock computation time on `full` task for tftS filters.

similar to the setting explored in [20] – a filter was shown one message at a time and asked to make a classification prediction. After prediction, the filter was allowed to request a ground-truth label for that message, with the goal of achieving strong classification performance with as few label requests as possible. The TREC scenario added two additional factors: the maximum amount of label requests a filter could make was fixed in advance by a label quota, and the filter was told how many messages remained in the online filtering task at a given point.

Our approach to the `active` task is based on the fixed margin sampling method proposed in [20]: a threshold t was fixed, and label requests were made whenever a message was classified with $|f(x)| < t$ until the label quota was exhausted. We used a slight variant of this idea: when there were at least 50 label requests available, we set $t = 0.8$, and when there were fewer than 50 remaining, we set $t = 0.2$. The goal of this heuristic was to reserve some label requests for particularly difficult messages appearing later in the data stream. However, our results show that this approach was not ideal, as in two cases not all the labels were used.

5. Experiments

The experiments we report here are all performed on the `trec07p` data set of 50,199 spam emails and 25,220 ham emails, and the private `mrx3` corpus of 8082 ham and 153893 spam, each set in a canonical ordering for online evaluation. We applied three filters, all of which were ROSVM-based filters with parameter settings as shown in Table 1, which were fixed before any tests were run on this data set. For a rough estimate of speed, Table 1 also shows wall-clock times for a complete run on the `trec07p full` task, using a Sunfire x86_64 machine with 8G RAM with no other load. Note that these timing results were performed using a pre-tokenized form of this data set, in order to remove overhead common to all filtering systems. All other results in this paper were found using the interface from the TREC Spam toolkit, which was significantly slower.

There were four main tasks for each of our three filters on these data sets. The `full` task gave supervised feedback to the filter after every prediction. The `delay` task gave supervised feedback to the filter for only the first 10,000 messages in the corpus. The `partial` task provided feedback for 30,388 messages in the corpus that belong to a subset of users. Finally, the `active` task allowed the filter to request a label for any message immediately following classification during online filtering. A maximum of 1000 labels were provided; all other messages were unlabeled for this task. Further details of the experimental design are available in the TREC 2007 Spam Track overview.

Results on `full` task. The results for all three ROSVM filters were quite strong on the full feedback tasks, with (1-ROCA)% scores of about 0.01 on `trec07p` and as low as 0.0042

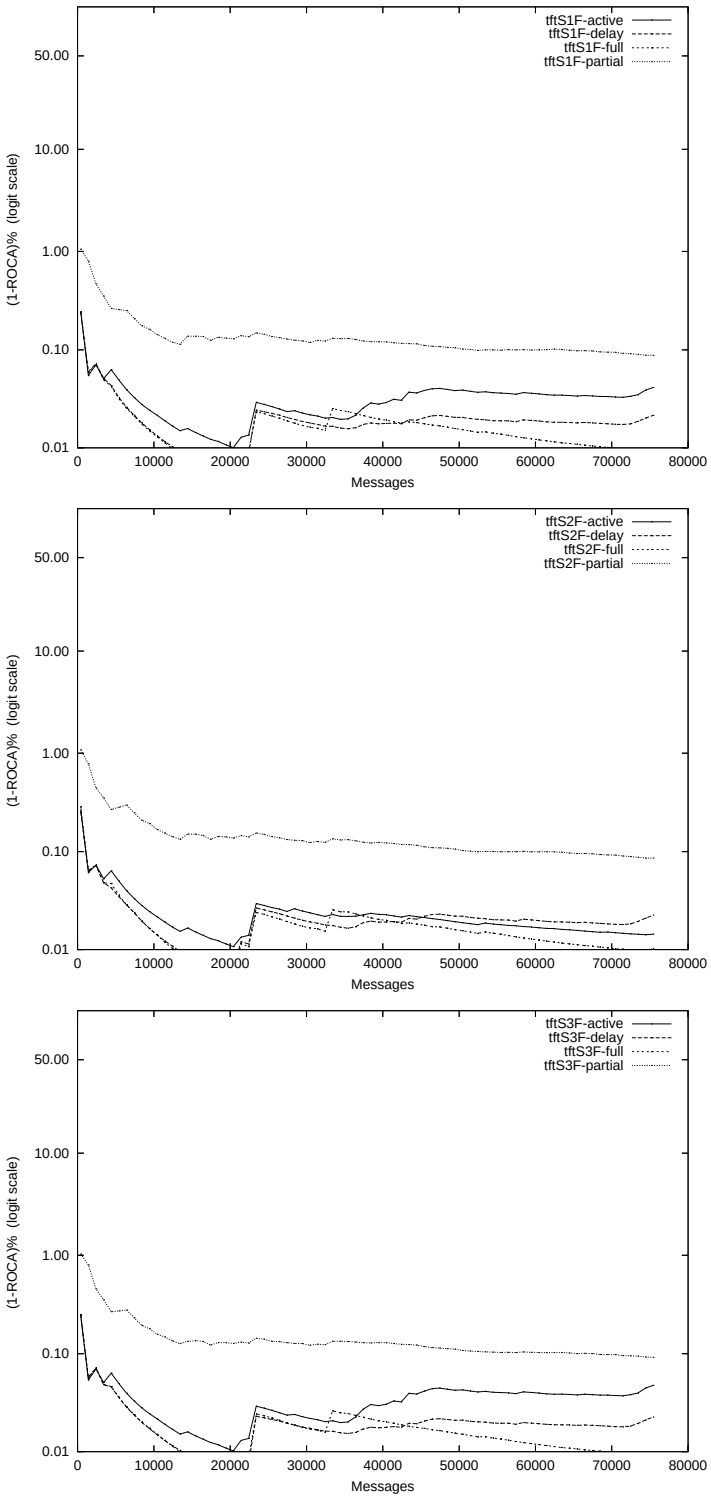

Figure 1: Learning curves for tftS1F, tftS2F, and tftS3F on `full`, `delay`, `partial`, and `active` tasks using `trec07p` data set.

FILTER	trec07p full	trec07p partial	trec07p delay	trec07p active	mrx3 full	mrx3 delay
TFTS1F	0.0099	0.0878	0.0214	0.0413	0.0166	0.0685
TFTS2F	0.0103	0.0858	0.0225	0.0144	0.0054	0.0683
TFTS3F	0.0093	0.0919	0.0226	0.0476	0.0042	0.0568
SPAMPROBE	0.0617	0.4870	0.1373	-	-	-

Table 2: Summary of Results on (1-ROCA)% measure for experiments on `trec07p` data set with `full` feedback, `partial` feedback from one user only, `delay` feedback with labels provided only for the first 10,000 messages, and `active` learning selection of 1000 labels. Results for `spamprobe` are included for comparison.

FILTER	PARTIAL FEEDBACK PARTIAL USERS ONLY	PARTIAL FEEDBACK ALL OTHERS	FULL FEEDBACK PARTIAL USERS ONLY
TFTS1F	0.0167	1.8239	0.0171
TFTS2F	0.0190	1.7506	0.0180
TFTS3F	0.0100	1.9411	0.0117

Table 3: Summary of Results on (1-ROCA)% measure for experiments on `trec07p` data set with three partial evaluations. The first column gives results on partial feedback evaluating only those messages that belong to the partial users. The second column shows results on partial feedback for all messages other than those of the partial users. In the third column, we report results evaluating those messages that belong to the partial users when full feedback is given on all messages in the corpus.

on `mrx3` (see Table 2). We were surprised to see that on `trec07p`, using a lookback buffer of size 5000 did not give significantly better results than the buffer of size 500. This is in contrast to the results on `mrx3` and previous results on the `trec06p` and `trec05p-1` data sets, where the larger lookback buffer gave improved performance.

For all filters, there was a performance hit (evident as an upward spike in the learning curves in Figure 1) just after 20,000 messages on `trec07p`. We found that this was caused by message 23337 in the data set, which appeared to have a noisy label. The message is a German-language stock-tip message in all capital letters – apparently spam. All of our filters labeled this message as spam with high confidence, but it was given a ground truth label of ham. The effect of this noisy label did not permanently degrade results, as all filters recovered to the 0.01 (1-ROCA)% level over time.

Results on `delay` task. As shown in Table 2, our filters gave reduced (but still reasonably strong) performance on the `delay` task for both `trec07p` and `mrx3`. This shows that while a filter may be expected to generalize into the future, there is (at least in these data sets) some amount of measurable concept drift associated with spam data.

FILTER	active	first 1000	uniform 1000
TFTS1F	0.0413	0.1095	2.9636
TFTS2F	0.0144	0.1105	2.9541
TFTS3F	0.0476	0.1092	2.8670

Table 4: **Active Learning Comparisons.** Results for tests on `trec07p` with active learning with 1000 label requests, compared to scenarios where only the first 1000 messages were labeled, or 1000 were chosen uniformly at random for labeling.

Results on `partial` task Surprisingly to us, this setting proved the most difficult in the group and our filters had far lower (1-ROCA)% scores on this task than even on the `delay` task, despite getting feedback on more than three times as many messages. Examining the results further shows that this degradation in performance is not due to overall lack of labels, as the performance on the partial user's own messages was very strong. Rather, we find that the model trained specifically on the partial users' messages does not transfer well to messages sent to other people – as shown by the very poor (1-ROCA)% scores when considering only these other messages. (See Table 3.)

Interestingly, we find that training on the other messages (in addition to the partial user's) does not significantly harm performance on the partial user's messages. This suggests that in a multi-user spam-filtering system, it may be equally good to have individual user filters or a single system-wide filter. However, an individually trained filter may not be expected to perform well for different users.

Results on `active` task Our results for the `active` task fall into two groups. The tftS1F filter had a (1-ROCA)% score of roughly 0.014, which approaches the result for full feedback while requesting labels for only one message in seventy-five. For tftS1F and tftS3F, results were significantly worse than that of full feedback – in the 0.04 range on the (1-ROCA)% score. The difference between these two groups was that the strong results were generated by a filter that used its entire label quota, while the weaker results were given by filters that did not exhaust their label quotas. This highlights one of the weaknesses of the fixed-margin sampling method – it is difficult to determine *a priori* exactly how many labels the filter will request for a given choice of t.

We considered the possibility that an optimal strategy would be to simply request many labels at the beginning of the task, so that any learned knowledge would be applied for as long as possible in the online setting. However, when we tested the effect of requesting labels for the first 1000 examples, we found that these results were significantly worse than using 1000 labels selected with active learning. This was even true for the first 10,000 labeled examples (as tested in the `delay` task) in comparison with the best of the three `active` results using 1000 labels. For completeness, we also with 1000 labels sampled uniformly at random, and found that these results were far worse than the active learners. These results are summarized in Table 4.

6. Discussion: Future Directions

The top-level performance at TREC has consistently increased over the three years of the spam filtering track, to the point where performance on the fully supervised online filtering task is approaching perfection, with ROC areas greater than 0.9999 and messages mis-classified at a rate of one in a thousand or better – even when starting from zero knowledge. It seems unlikely that inter-annotator agreement would be high enough from human evaluations to make further improvements measurable in this domain.

However, this scenario of full feedback on all messages is unrealistic in live settings. The `delay`, `partial`, and `active` tasks from this year's competition were all strong steps towards addressing differences between laboratory and real-world settings. We plan to continue work in this vein, especially with regard to noisy labels, malicious labels, and the use of unlabeled data.

Additionally, there are a wide variety of spam domains that are, as yet, unexplored on a TREC-level scale. These include detecting content-based spam in blogs, wikis, social networking sites, SMS text messages, image sharing sites, and collaborative tagging efforts – to name but a few. Continued work transferring and augmenting techniques from email spam filtering to these domains is needed.

References

[1] I. Androutsopoulos, J. Koutsias, K. V. Chandrinos, G. Paliouras, and C. D. Spyropoulos. An evaluation of naive bayesian anti-spam filtering. In *Proceedings of Workshop on Machine Learning in the New Information Age, 11th European Conference on Machine Learning*, 2000.

[2] 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.

[3] A. Bratko, B. Filipič, G. Cormack, T. Lynam, and B. Zupan. Spam filtering using statistical data compression models. *J. Mach. Learn. Res.*, 7:2673–2698, 2006.

[4] G. Cormack and A. Bratko. Batch and on-line spam filter evaluation. In *CEAS 2006: Proceedings of the Third Conference on Email and Anti-Spam*, 2006.

[5] G. Cormack and T. Lynam. On-line supervised spam filter evaluation. *ACM Transactions on Information Systems*, 25(3), July 2007.

[6] D. DeCoste and K. Wagstaff. Alpha seeding for support vector machines. In *KDD '00: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 345–349, 2000.

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

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

[9] P. Graham. A plan for spam. 2002.

[10] P. Graham. Better bayesian filtering. 2003.

[11] T. Joachims. Text categorization with suport vector machines: Learning with many relevant features. In *ECML '98: Proceedings of the 10th European Conference on Machine Learning*, pages 137–142, 1998.

[12] T. Joachims. Making large-scale SVM learning practical. *Advances in Kernel Methods - Support Vector Learning,* B. Scholkopf and C. Burges and A. Smola (ed.), MIT-Press, 1999.

[13] A. Kolcz and J. Alspector. SVM-based filtering of e-mail spam with content-specific misclassification costs. In *Proceedings of the TextDM'01 Workshop on Text Mining - held at the 2001 IEEE International Conference on Data Mining*, 2001.

[14] C. Leslie, E. Eskin, and W. S. Noble. The spectrum kernel: A string kernel for svm protein classification. *Proceedings of the Pacific Symposium on Biocomputing*, pages 564–575, January 2002.

[15] T. Lynam, G. Cormack, and D. Cheriton. On-line spam filter fusion. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 123–130, 2006.

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

[17] J. Platt. Sequenital minimal optimization: A fast algorithm for training support vector machines. In B. Scholkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*. MIT Press, 1998.

[18] 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.

[19] B. Scholkopf and A. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2001.

[20] D. Sculley. Online active learning methods for fast label-efficient spam filtering. In *CEAS 2007: Proceedings of the Fourth Conference on Email and Anti-Spam*.

[21] D. Sculley and G. Wachman. Relaxed online SVMs for spam filtering. In *The Thirtieth Annual ACM SIGIR Conference Proceedings*, 2007.

[22] D. Sculley, G. Wachman, and C. Brodley. Spam filtering using inexact string matching in explicit feature space with on-line linear classifiers. In *The Fifteenth Text REtrieval Conference (TREC 2006) Proceedings*, 2006.

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