Boosting of Support Vector Machines with application to editing

Pedro Rangel Fernando Lozano Elkin García
Departamento de Ingeniería Eléctrica y Electrónica
Universidad de los Andes
Bogotá, Colombia
{p-rangel,flozano,elkin-ga}@uniandes.edu.co

Abstract

In this paper, we present a weakened variation of Support Vector Machines that can be used together with Adaboost. Our modified Support Vector Machine algorithm has the following interesting properties: First, it is able to handle distributions over the training data. Second, it is a weak algorithm in the sense that it ensures an empirical error upper bounded by 1/2. Third, when used together with Adaboost, the resulting algorithm is faster than the usual SVM training algorithm. Finally, we show that our boosted SVM can be effective as an editing algorithm.

1. Introduction

Adaboost [4] is an algorithm which employs a weak learner (i.e. an algorithm which returns a hypothesis that is little better than random guessing) to find a good hypothesis. In this paper, we present a classification algorithm which uses a variant of Support Vector Machines as the weak learner for Adaboost. The success of Support Vector Machines (SVM) in practical classification problems [9] has been explained from the statistical learning standpoint. In particular, it has been recently shown that for certain kernels SVMs are strong learners [11]. This means that they can achieve generalization error arbitrarily close to the Bayes error, given a large enough training data set.

Some empirical evidence shows that using a strong learner as the base classifier for Adaboost is not a good idea. Wickramaratna, Holden and Buxton apply directly SVM with Adaboost [12] and observe that the performance of the boosted classifier degrades as the number of Adaboost rounds increases. Of course, boosting of a strong learner does not make much sense, from the generalization error point of view. However, we will see that if the SVM is forced to output a weaker hypothesis, boosting still may have some other advantages.

The main drawbacks of the Support Vector Machines are the time and space complexities of the training algorithm. Usually, the SVM training algorithm solves a large Quadratic Programming (QP) Problem. Let m denote the number of training examples. The time complexity of a QP problem is $O(m^3)$. Several researchers have proposed various methods that improve this time complexity. Some of these algorithms include: chunking, the decomposition method and the sequential minimal optimization method [7]. In practice, these algorithms have a time complexity that usually scales like $O(m^2)$.

Pavlov, Mao and Dom [6] propose a method for speeding and scaling up the SVM training algorithm using Adaboost. They implement the SVM training procedure by means of the Sequential Minimal Optimization algorithm. In order to simulate the distributions required by Adaboost, they use bootstrap samples of the original data. In their experiments, the bootstrap samples size was approximately equal to 2-4% of the original data set size. In this way, they reduce substantially the training set in each Adaboost round.

We present a modified version of SVM which has the following properties: First, it is able to handle distributions without using bootstrap samples. Second, it is a weak algorithm in the sense that it ensures an empirical error upper bounded by 1/2. Third, when our algorithm is combined with Adaboost, the resulting algorithm outputs a hypothesis that is also a SVM, but which is trained much faster. Finally, we present empirical evidence that suggests that our Boosted Support Vector Machines algorithm can be used as an *editing algorithm* [1]. Usually, the SVM classification rule is unnecessarily complex (i.e. The SVM hypothesis has linearly dependent support vectors). An *editing algorithm* is a procedure that reduces the training set in order to simplify the representation of the SVM.

2. Preliminaries

Let (x, y) be a random couple, where x is an instance in a space $\mathcal{X}$ and $y \in \{-1, 1\}$ is a label. Let $S = \{\langle x_i, y_i \rangle\}_{i=1}^{m}$ be a labeled set, consisting of i.i.d. copies of (x, y).

Proceedings of the Fourth International Conference on Machine Learning and Applications (ICMLA'05)
0-7695-2495-8/05 $20.00 © 2005 IEEE

Algorithm: Adaboost $(S, D_1, T, Weak)$

Input: $S = \{x_i, y_i\}_{i=1}^m$, D_1, T, $Weak(\cdot, \cdot)$
Output: $H(\cdot)$

for $t = 1$ **to** T **do**
  Get a weak hypothesis using D_t.
  $h_t \leftarrow Weak(S, D_t)$.
  Choose $\alpha_t \in \mathbb{R}$
  Update the distribution D_{t+1}.
end

Output: $H(x) = sign\left(\sum_{t=1}^T \alpha_t h_t\right)$.

Figure 1: Adaboost Algorithm.

A *classification rule*, also called a *hypothesis*, is a function $h : \mathcal{X} \mapsto [-1, 1]$. The sign of $h(x)$ is interpreted as the predicted label to be assigned to instance x, while the magnitude $|h(x)|$ is interpreted as the "confidence" in this prediction. The goodness of a hypothesis will be evaluated using the *generalization error* R, and the *empirical error* R_{emp}.

We say that $H(x)$ is a *combined classifier* when it is a convex combination of several hypotheses h_i. That is

$$H(x) = \sum_{i=1}^m \alpha_i \cdot h_i(x)$$

Where $\alpha_i \geq 0$ and $\sum_{i=1}^m \alpha_i = 1$. Each hypothesis h_i will be called a *base classifier*.

2.1. Adaboost

Adaboost, first introduced in [4], is a meta-algorithm which has the ability to return a strong hypothesis using a weak algorithm as a subroutine. Figure 1 shows the Adaboost algorithm as presented in [8].

Adaboost takes a labeled examples set S, a discrete distribution D, and a weak learning algorithm $Weak$; and returns a combined classifier. At each iteration t, Adaboost executes $Weak$ over the distribution D_t to get h_t, then it modifies the distribution assigning larger weights to examples misclassified by h_t and smaller weights to examples with high confidence $|h_t(x)|$.

2.2. Support Vector Machines (SVM)

Our algorithm uses the ν formulation of the SVM problem. The ν parameter allows us to control the empirical error of the classifier as we explain below. The optimization problem in the ν formulation is as follows [10]:

$$\begin{aligned} \min_{w,\xi,\rho} \quad & \tau(w, \xi, \rho) = \frac{1}{2}\|w\|_{\mathcal{H}}^2 - \nu\rho + \frac{1}{m}\sum_{i=1}^m \xi_i \qquad (1) \\ \text{s.t.} \quad & y_i\left[\langle\phi(x_i), w\rangle_{\mathcal{H}} + b\right] \geq (\rho - \xi_i), \quad \text{for i=1,...,m} \\ & \xi_i \geq 0,\ \rho \geq 0 \end{aligned}$$

Here $\mathcal{H}$ is the reproducing kernel Hilbert space of a positive definite kernel $k(\cdot, \cdot)$ and ϕ is a mapping from $\mathcal{X}$ to $\mathcal{H}$ such that $k(x_i, x_j) = \langle\phi(x_i), \phi(x_j)\rangle_{\mathcal{H}}$. Using the kernel trick, the dual problem becomes:

$$\begin{aligned} \min_{\alpha} \quad & W(\alpha) = \frac{1}{2}\sum_{i=1}^m\sum_{j=1}^m \alpha_i\alpha_j y_i y_j k(x_i, x_j) \qquad (2) \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq \frac{1}{m} \\ & \sum_{i=1}^m \alpha_i y_i = 0 \quad \sum_{i=1}^m \alpha_i \geq \nu \end{aligned}$$

This optimization problem yields a hypothesis which has the form:

$$h(x) = sign\left(\sum_{i=1}^m \alpha_i y_i k(x, x_i) + b\right) \qquad (3)$$

The training examples are classified in three categories depending on the value of α_i. Within each category, the data margins $y_i h(x_i)$ are prescribed by the Karush-Kuhn-Tucker optimality conditions. The first category consists of all data points such that $\alpha_i = 1/m$ and satisfy $y_i h(x_i) < \rho$. These data points are called *margin errors* or *bouncing support vectors*. Note that the set of *bouncing vectors* includes all the training examples misclassified by the SVM. The second category consists of the data points such that $0 < \alpha_i < 1/m$ and satisfy $y_i h(x_i) = \rho$. They are called *ordinary support vectors*. The third category consists of examples such that $\alpha_i = 0$ and satisfy $y_i h(x_i) > \rho$. These examples play no role in the SVM decision function.

3. Boosting Support Vector Machines

In this section, we present a boosting algorithm which uses SVM as its weak classifier. In order to do that, we need to solve two issues: we need to modify the support vector machine optimization problem so it can deal with distributions, and we need to weaken the performance of the support vector machine.

3.1. Modified Support Vector Machines

We modify the formulation of the ν-SVM to introduce the distribution D by multiplying the slack variables ξ_i by the corresponding weight D_i:

$$\min_{w,\xi,\rho} \quad \tau(w,\xi,\rho) = \frac{1}{2}\|w\|^2_{(H)} - \nu\rho + \sum_{i=1}^{m} D_i\xi_i \qquad (4)$$
$$\text{s.t.} \quad y_i\left[\langle x_i, w\rangle_{\mathcal{H}} + b\right] \geq D_i(\rho - \xi_i), \quad \text{for i=1,...,m}$$
$$\xi_i \geq 0,\ \rho \geq 0$$

In this way, we encourage solutions of the optimization problem that do not err on examples with larger weights.

The dual problem becomes:

$$\min_{\alpha} \quad W(\alpha) = \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y_i y_j k(x_i,x_j) \qquad (5)$$
$$\text{s.t.} \quad 0 \leq \alpha_i \leq 1$$
$$\sum_{i=1}^{m}\alpha_i y_i = 0$$
$$\sum_{i=1}^{m} D_i\alpha_i \geq \nu \qquad (6)$$

We can use several optimization algorithms to solve efficiently this problem. We say that $SV_{k,\nu}(S,D)$ is a learning algorithm that solves the optimization problem (5), and returns a hypothesis $h_{w,b}$.

The ν parameter in (5), lets us control the empirical error of the classifier as is explained in the next lemma.

Lemma 1 *Let $h^*_{w,b}$ be a hypothesis returned by $SV_{k,\nu}(S,D)$. If w and b are a feasible point of (4) with $\rho > 0$, then $R_{emp}(h^*_{w,b},S,D) \leq \nu$ (i.e. The ν parameter is an upper bound in the empirical error of the classifier).*

Proof By the Karush-Kuhn-Tucker optimality conditions, ρ greater than zero implies that (6) becomes an equality. Hence

$$\nu = \sum_{i=1}^{m} D_i\alpha_i \geq \sum_{i:\alpha_i=1} D_i \geq \sum_{i=1}^{m} D_i[\![sign(h(x_i)) \neq y_i]\!]$$

where the last inequality follows from the fact that all examples with $\xi_i > 0$ satisfy $\alpha_i = 1$ (if not, α_i could grow further to reduce the value of ξ_i).

□

Then, if we can guarantee that ρ is greater than zero, we can guarantee that the empirical error is upper bounded by ν.

We define a special kind of kernels called *universal kernels*. Let ϕ be a transformation, and let k be the kernel of ϕ. We say that the kernel k is universal if for any uniquely labeled data set $S = \{\langle x_i, y_i\rangle\}_{i=1}^m$, the image of S through ϕ is linearly separable. When we use an universal kernel in (5), we can always find a solution with $\rho > 0$.

Lemma 2 *Let k be a universal kernel, and let $h^*_{w,b}$ be a hypothesis returned by $SV_{k,\nu}(S,D)$, then the empirical error of $h^*_{w,b}$ is always upper bounded by ν.*

The proof of this lemma follows immediately from lemma 1 and the universality of the kernel.

Algorithm: WSV $(S, D, k, \gamma, \lambda)$
Input: $S = \{\langle x_i, y_i\rangle\}_{i=1}^m$, D, k, γ, λ
Output: $h(\cdot)$
begin
Set $\nu = \lambda \cdot (1/2 - \gamma)$.
Set $\mu = (1-\lambda)\cdot(1/2-\gamma)$.
Select J such that $\sum_{j\in J} D(j) \leq \mu$, and it has maximum cardinality.
Set $S^* = \{\langle x_j, y_j\rangle\}_{J'}$.
Set $D^* = D_{J'}$.
end
Output the hypothesis $h(\cdot) \leftarrow SV_{k,\nu}(S^*, D^*)$

Figure 2: Weakened Support Vector Machine Algorithm.

3.2. Weakening Support Vector Machines

As mentioned in the introduction, using Adaboost in combination with Support Vector Machines can be counterproductive. Nevertheless, we claim that using a weakened Support Vector Machine in conjunction with Adaboost has some advantages over a single SVM trained in the usual fashion. The usefulness of the method we propose is twofold. First, it potentially improves the performance of the Support Vector Machines. Second, it speeds up the training algorithm.

Our weakened SVM algorithm, that we will refer to as WSV, takes as inputs a labeled examples set S, a distribution D, a kernel k, and a "weakness" constant γ. WSV returns a hypothesis with empirical error upper bounded by $\epsilon = \frac{1}{2} - \gamma$. Moreover, it runs faster than the original support vector machine training algorithm. The basic idea of WSV is to reject some examples in the training set S, and train the support vector machine using the remaining examples. Figure 2 shows the WSV algorithm.

Let J be the index set of the samples that we eliminate, and let J' be the complement of J. Let μ be an upper bound on the weight of the rejected samples (i.e. $\mu \geq \sum_J D(j)$). WSV runs $SV_{k,\nu}(\cdot,\cdot)$ using a modified labeled examples set with less data. It selects J such that $\sum_{j\in J} D(j) \leq \mu$,

and has maximum cardinality. We can show that this algorithm guarantees an upper bound on the empirical error. To see this, notice that the allowed error can be split into two parts. The first part is due to the original algorithm $SV_{k,\nu}(S, D)$, and we can control it using the ν parameter (see lemma 2). The second part comes from the discarded portion of the training set. If we assume a worse case scenario in which the hypothesis returned by SV errs on the whole discarded set, then the empirical error of the algorithm is upper bounded by $\nu + \mu$. Now, let λ be a real number between 0 and 1. If we choose $\nu = \lambda\epsilon$ and $\mu = (1-\lambda)\epsilon$, we can guarantee that the empirical error of WSV is upper bounded by ϵ.

Since the time complexity of the training algorithm of SVM scales as $O(m^2)$, the training time of WSV is reduced. The number of rejected data depends on two factors: First, it depends on the distribution over the original data set. For example, if the distribution is uniform (as is the case in the first round of Adaboost) the percentage of points rejected is about $1 - \mu$. On the other hand, if the distribution is askew (as in further rounds of Adaboost) the percentage of discarded points becomes larger. Hence, if we use WSV as the weak learner, we can execute many rounds of Adaboost in a small amount of time. The second factor which affects the number of rejected data is the value of μ. The larger μ is, the more data is rejected. In the WSV algorithm, μ can be increased in two ways: decreasing γ, or decreasing λ.

3.3. Boosting SVM as an Editing Algorithm

During both the training and prediction stages, the performance of a SVM is highly influenced by the number of support vectors. When a SVM classifier does not have enough support vectors, it is not able to make predictions with high accuracy. On the other hand, the more support vectors, the slower is the prediction. Ideally, the image of the support vectors in the feature space should be linearly independent, giving a more compact representation.

There are several methods to reduce the number of support vectors without degrading the generalization error of a SVM classifier. Most of these techniques are focused on reducing the number of support vectors after the training procedure [3]. Since these techniques require to compute the SVM solution before being applied, they do not improve the training time. On the other hand, there are methods that selectively reduce the training set before running the training algorithm [1].

In order to obtain a good SVM classifier with a reduced training set, the data points in the new training set must look as if they were drawn from a distribution that has the same Bayes decision boundary of the original problem, but with Bayes error equal to zero. So, the idea is to eliminate the training examples located on the wrong side of the Bayes decision boundary.

Algorithm: WSVE $(S, D, k, \gamma, \nu, \mu, \sigma)$
Input: $S = \{\langle x_i, y_i\rangle\}_{i=1}^{m}$, D, k, γ, ν, μ, σ
Output: $h(\cdot)$
begin
Select J such that $\sum_{j\in J} D(j) \leq \sigma$, and it has minimum cardinality.
Set $S^* = \{\langle x_j, y_j\rangle\}_J$.
Set $D^* = D_J / \sum D_J$.
$h(\cdot) \leftarrow$ MWSV $(S^*, D^*, k, \gamma, \nu, \mu)$.
end
Output the hypothesis $h(\cdot)$

Figure 3: Weakened SVM Algorithm for editing.

A priori, we do not know which training examples are badly placed. We propose to use the distributions generated by Adaboost to estimate the probability of a point being rejected. Adaboost gives high weights to the examples that are misclassified in various rounds. Hence, the points with large weights are more likely to be in the wrong side of the Bayes decision boundary. So, if we reject the examples with the largest weights, we can use the Boosting SVM algorithm to perform editing. The Weakened SVM algorithm for editing is shown in Figure 3. This modified algorithm creates a training set which rejects the points with large weights, then it runs the MWSV algorithm to get an hypothesis with high accuracy in this modified data set.

4. Experiments

4.1. Boosting SVM

In this section, we report some experiments with the Boosting SVM algorithm proposed above. We implement the SVM training algorithm using the sequential minimal optimization algorithm introduced by Platt [7] and modified by Chang and Lin [2] for the ν formulation. We modify this algorithm so we can solve problem 5. This implementation performs caching of the most frequently used support vectors. Note that if we remove the sign function in (3), then a combined classifier of SVM is also a SVM. We use this fact to create a SVM as the output of our Adaboost algorithm that we refer to as the reduced model. In preliminary experiments, the resulting SVM has a similar decision function, but it has a number of support vectors that does not increase with the number of data points in the training set.

We show results for an artificial data set and a real life data set. The artificial data set is called Fournorm. This is a twenty dimensional binary classification problem. Data for the first class is drawn with equal probability from one of two multivariate normal distributions

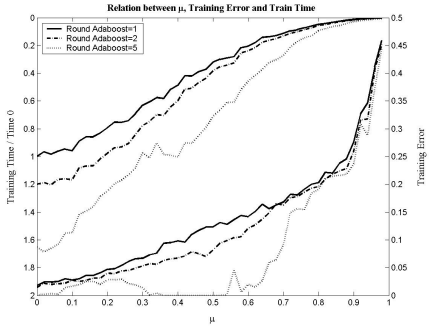

Figure 4. Training time and training error vs. weight of rejected data (fournorm).

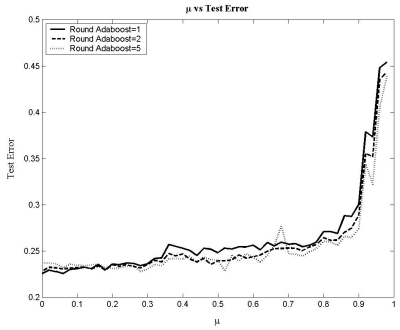

Figure 5. Generalization error vs. Weight of rejected data (fournorm)

with identity covariance matrix and means $(a, a, \ldots, a)$ and $(-a, -a, -a, -a, \ldots, -a)$, while data for the other class is drawn with equal probability from one of two multivariate normal distributions with means $(a, -a, \ldots, -a)$ and $(-a, a, -a, a, \ldots, a)$ and identity covariances matrices. We set $a = 2/\sqrt{2}$. In Figure 4 we plot training time and training error against the weight of the data that is rejected. Figure 5 shows the generalization error vs. the weight of the data that is rejected.

For our second set of experimets we utilized the MNIST data set of handwritten digits [5]. To reduce the classification problem to a binary problem, we use only the data corresponding to the numbers 8 and 3. We use 5000 training and 1984 test examples, with a polynomial kernel of degree 3 and $\nu = 0.1$. We set the maximum number of iterations to 10000. The Adaboost algorithm runs 10 iterations for several values of μ between 0 and 0.9.

Results for both data sets are similar. We can observe that the training error is a decreasing function of the number of rounds, regardless of the amount of data that is rejected (see Figures 4 and 6). This presents an advantage in terms of the total training time relative to the training time using the whole data set. It is faster to train some rounds of Adaboost rejecting a lot of samples than it is to train with the original data set.For example, for the Fournorm data set, rejecting 58% of the data results in a training error of 1% after 5 rounds of Adaboost but the training time is close to one half of the training time with the whole data set. For the MNIST data set, rejecting 50% of the data and after 2 rounds of Adaboost the training error reaches 0% and the training time is close to one half of the original training time.

Regarding the generalization error, Figure 7 shows that for the MNIST data, the test error with rejected data is less than the test error with the whole data set. After 5 rounds of Adaboost, rejecting 60% of the data set results in generalization error similar to the error with the entire data set with a dramatically reduced training time.

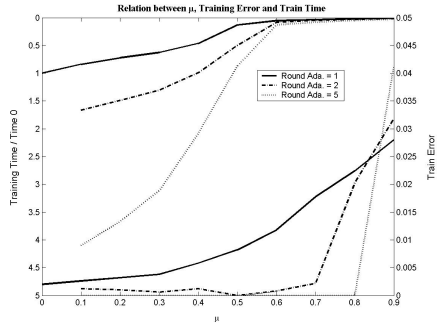

Figure 6. Training time and training error vs. weight of rejected data (MNIST).

4.2. Boosting SVM as an Editing Algorithm

For this experiments, we use a simple two dimensional binary classification problem. The classes are generated from two uniform distributions in squares of side size one with centers at c_1 and c_2 respectively. The position of the centers allow us to control the Bayes error of the problem. In these experiments, we set the Bayes error to 10%.

Figure 8 shows that the number of support vectors of our methods does not increase very much. This fact implies that our algorithm can be used effectively as an editing algorithms during the training procedure. In addition, our algorithms also reduce the training time as shown in Figure 9. Furthermore, the empirical errors of the three methods are similar.

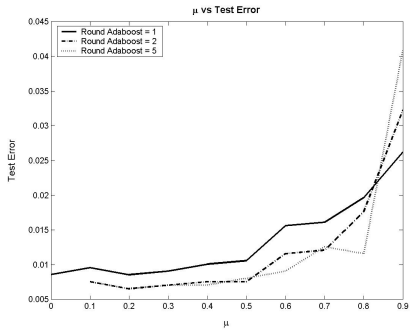

Figure 7. Generalization error vs. Weight of rejected data (MNIST)

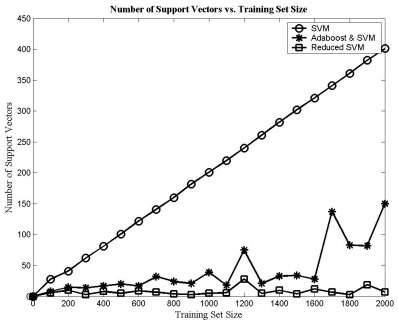

Figure 8. Number of Support Vectors vs. Training set size

5. Conclusions

Although there exists empirical evidence that boosting a strong learner may not be a good idea from the generalization standpoint, our experiments demonstrate that when combined with the reduced set method that we propose, it can lead to some advantages in the running time of the algorithm. Moreover, adaboost is able to identify very effectively the bouncing support vectors in a SVM solution. We exploit this fact to implement an editing algorithm that produces classifiers with a more compact representation.

Topics for future research are to investigate a more theoretical explanation of the editing capabilities of Adaboost, and methods of finding optimal values for all the hyperparameters of our algorithms.

Acknowledgement

This research was funded by a grant from the School of Engineering, Universidad de los Andes.

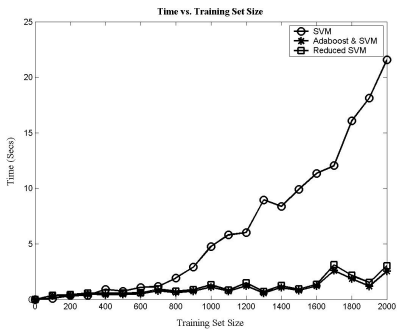

Figure 9. Time vs. Training set size

References

[1] G. Bakır, L. Bottou, and J. Weston. Breaking SVM complexity with cross-training. In *Advances in Neural Information Processing Systems*, volume 17. MIT Press, 2005.

[2] C.-C. Chang and C.-J. Lin. Training ν-support vector classifiers: Theory and algorithms. *Neural Computation*, (9):2119–2147, 2001.

[3] T. Downs, K. E. Gates, and A. Masters. Exact simplification of support vector solutions. *J. Mach. Learn. Res.*, 2:293–297, 2002.

[4] Y. Freund and R. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, Aug. 1997.

[5] Y. LeCun. The MNIST database of handwritten digits. http://yann.lecun.com/exdb/mnist/.

[6] D. Pavlov, J. Mao, and D. Dom. Scaling-up support vector machines using boosting algorithm. *15th International Conference on Pattern Recognition*, 2:219–222, 2000.

[7] J. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 185–208, Cambridge, MA, 1999. MIT Press.

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

[9] B. Schölkopf and A. Smola. *Learning With Kernels*. MIT Press, Cambridge, MA, 2002.

[10] B. Schölkopf, A. Smola, R. Williamson, and P. Bartlett. New support vector algorithms. NeuroCOLT Technical Report NC-TR-98-031, Royal Holloway College, University of London, UK, 1998.

[11] I. Steinwart. Cosistency of support vector machines and other regularized kernel classifiers. *IEEE Transactions on Information Theory*, 51(1):128–142, January 2005.

[12] J. Wickramaratna, S. Holden, and B. Buxton. Performance degradation in boosting. In J. Kittler and F. Roli, editors, *Proceedings of the 2nd International Workshop on Multiple Classifier Systems MCS2001*, volume 2096 of *LNCS*, pages 11–21. Springer, 2001.