

# SPARSE BOOSTING

Zhen James Xiang and Peter J. Ramadge

Dept. of Electrical Engineering, Princeton University, Princeton NJ

## ABSTRACT

We propose a boosting algorithm that seeks to minimize the AdaBoost exponential loss of a composite classifier using only a sparse set of base classifiers. The proposed algorithm is computationally efficient and in test examples produces composite classifiers that are sparser and generalization better than those produced by Adaboost. The algorithm can be viewed as a coordinate descent method for the  $l_1$ -regularized Adaboost exponential loss function.

**Index Terms**— Pattern classification, Algorithms, Signal representations, Optimization methods.

## 1. INTRODUCTION AND OUTLINE

AdaBoost [1] is used as a pattern classification algorithm in a wide variety of signal processing applications [2]. AdaBoost constructs a powerful composite classifier as a weighted linear combination of a large set of base classifiers. Although the discriminant power of a single base classifier is usually weak, the composite classifier can achieve an acceptable classification accuracy.

In addition to classification accuracy, sparsity of the composite classifier is a desirable attribute. By this we mean that relatively few base classifiers are assigned a nonzero weight in the linear combination. A sparse classifier is easier to store, process, and interpret and, most importantly, is less prone to over fitting. Through empirical studies, AdaBoost is known to be generally resistant to over fitting, even for a large number of iterations [3, 4]. However, we will show that there are additional gains to be made by adding a sparsity mechanism directly into the boosting algorithm.

Boosting algorithms can be interpreted as iterative gradient descent procedures that minimize a loss function on the training data [5, 6, 7]. In many cases, these schemes add at most one new weight per iteration. Hence early stopping of the boosting process is a simple method to ensure sparsity [8]. It has been shown that early stopping of AdaBoost approximately minimizes an exponential loss function subject to an  $l_1$  constraint on the coefficient vector [9]. This suggests that sparsity can be ensured by imposing  $l_1$ -regularization on the optimization of the loss function. This is in accord with results in compressed sensing, where  $l_1$ -regularization has proved an effective approximation to an  $l_0$  sparsity con-

straint [10]. The idea of  $l_1$ -regularized loss minimization has been explored in [9]. However, direct solution of the convex  $l_1$ -regularized loss problem is computationally too expensive in many real applications. This has led to proposals for indirect methods of solution. To this end,  $\epsilon$ -boosting seeks to solve the regularized problem iteratively adding a small weight  $\epsilon$  to one base classifier each iteration [5]. However,  $\epsilon$ -boosting is too inefficient for practical application. Other work has examined combining Adaboost with smaller  $l_1$ -regularized loss optimization problems from the perspective of maximizing the margin on the training examples [11]. Despite this work, there remains a need for an efficient boosting algorithm that directly incorporates a sparsity mechanism.

Our main contribution is to propose a new, computationally efficient boosting algorithm, called RBoost (short for *Regularized Boost*) that iteratively solves the  $l_1$ -regularized loss minimization problem. RBoost works in a similar fashion to AdaBoost except that it incorporates a simple, intuitive mechanism for  $l_1$ -regularization. Moreover, with one simple change, RBoost reverts to Adaboost. Our empirical studies show that RBoost achieves better generalization than Adaboost with sparser composite classifiers.

In Section 2, we introduce the basic notation of the pattern classification problem and review the AdaBoost algorithm. We then present and analyze the RBoost algorithm in Section 3. The performance of RBoost is examined experimentally in Section 4, followed by conclusions in Section 5.

## 2. PRELIMINARIES

In a typical binary classification problem, one is given training data  $\mathcal{S} = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$  (where  $\mathbf{x}_i \in \mathbb{R}^p$  are instances and  $y_i \in \{-1, +1\}$  are corresponding labels) and a set of base classifiers  $\mathcal{H} = \{h_j\}_{j=1}^{|\mathcal{H}|}$ . A base classifier  $h_j \in \mathcal{H}$  is a mapping from instances to possible labels,  $h_j : \mathbb{R}^p \mapsto \{-1, +1\}$ . The performance of any classifier  $h$  under probability distribution  $\mathbf{w} = \{w_i\}_{i=1}^m$  is measured by its *edge*:  $edge(h, \mathbf{w}) = \sum_{i=1}^m w_i y_i h(\mathbf{x}_i)$ , which is related to the error probability  $P_{err}(h)$  with respect to  $\mathbf{w}$  on the training set by  $edge(h, \mathbf{w}) = 1 - 2P_{err}(h)$

Let  $h_c(x) = \text{sign}(\sum_{j=1}^{|\mathcal{H}|} \alpha_j h_j(x))$  denote the *composite classifier*. We assume that  $h \in \mathcal{H} \Rightarrow -h \in \mathcal{H}$  and require  $\alpha_j \geq 0$ ,  $j = 1, \dots, |\mathcal{H}|$ . Using this notation, AdaBoost can be specified as follows:

**Algorithm 1** AdaBoost

---

```

1:  $\vec{\alpha} \leftarrow \mathbf{0}$ 
2:  $w_i \leftarrow \frac{1}{m}, \quad i = 1, 2, \dots, m$ 
3: for  $t = 1$  to  $T$  do
4:    $k \leftarrow \arg \max_j \text{edge}(h_j, \mathbf{w})$ 
5:    $\varepsilon = \frac{1}{2} \ln \frac{1 + \text{edge}(h_k, \mathbf{w})}{1 - \text{edge}(h_k, \mathbf{w})} = \frac{1}{2} \ln \frac{1 - P_{\text{err}}(h_k)}{P_{\text{err}}(h_k)}$ 
6:    $\alpha_k \leftarrow \alpha_k + \varepsilon$ 
7:    $w_i \leftarrow \begin{cases} w_i e^\varepsilon & \text{if } y_i h_k(x_i) = -1 \\ w_i e^{-\varepsilon} & \text{if } y_i h_k(x_i) = 1 \end{cases}$ 
8:    $\mathbf{w} \leftarrow \frac{\mathbf{w}}{\|\mathbf{w}\|_1}$ 
9: end for
10: Result:  $h_c(x) = \text{sign}(\sum_{j=1}^{|\mathcal{H}|} \alpha_j h_j(x))$ 

```

---

The *margin* of the classifier on example  $i$  is  $\mu_i = \sum_{j=1}^{|\mathcal{H}|} y_i \alpha_j h_j(x_i) / \|\vec{\alpha}\|_1$ . There is empirical and theoretical evidence that AdaBoost achieves good generalization by enlarging the margins on the training examples [12]. Maximizing the margins is closely related to minimizing the following *exponential loss function*:

$$\mathcal{L}(\vec{\alpha}) = \sum_{i=1}^m \exp(-y_i \sum_{j=1}^{|\mathcal{H}|} \alpha_j h_j(x_i)) \quad (1)$$

In [6] and [7] it is shown that AdaBoost is equivalent to a coordinate descent procedure for minimizing (1). Moreover, in [9] it is shown that early stopping of AdaBoost approximately minimizes (1) subject to an  $l_1$  constraint on the coefficient vector  $\vec{\alpha}$ .

In summary, the basic mechanism of AdaBoost is currently understood in terms of minimizing the exponential loss (1) using coordinate descent and this in turn is closely related to enlarging the margins. Minimization of the loss function subject to  $l_1$ -regularization has also been considered [5, 11]. However, obtaining an efficient algorithm to solve the regularized problem remains a problem under study.

### 3. THE RBOOST ALGORITHM

With the motivation provided above, we aim to efficiently solve the regularized loss minimization problem:

$$\min : \quad \mathcal{L}(\vec{\alpha}) = \sum_{i=1}^m \exp(-y_i \sum_{j=1}^{|\mathcal{H}|} \alpha_j h_j(x_i)) \quad (2)$$

$$\text{s.t. :} \quad \sum_{j=1}^{|\mathcal{H}|} |\alpha_j| \leq R \quad (3)$$

Our proposed algorithm, RBoost, for obtaining a solution is defined as follows:

**Algorithm 2** RBoost

---

```

1:  $\vec{\alpha} \leftarrow \vec{\alpha}_0$ 
2:  $w_i = \exp(-y_i \sum_{j=1}^{|\mathcal{H}|} \alpha_j h_j(x_i))$ 
3:  $\mathbf{w} \leftarrow \frac{\mathbf{w}}{\|\mathbf{w}\|_1}$ 
4: for  $t = 1$  to  $T$  do
5:    $h_k = \arg \max_j \text{edge}(h_j, \mathbf{w})$ 
6:    $h_l = \arg \min_{j: \alpha_j > 0} \text{edge}(h_j, \mathbf{w})$ 
7:    $\varepsilon = \min(2\alpha_l, \frac{1}{2} \ln \frac{P(y h_k(x) > y h_l(x))}{P(y h_k(x) < y h_l(x))})$ 
8:    $\alpha_k \leftarrow \alpha_k + \frac{\varepsilon}{2}$ 
    $\alpha_l \leftarrow \alpha_l - \frac{\varepsilon}{2}$ 
9:    $w_i \leftarrow \begin{cases} w_i e^\varepsilon & \text{if } y_i h_k(x_i) < y_i h_l(x_i) \\ w_i e^{-\varepsilon} & \text{if } y_i h_k(x_i) > y_i h_l(x_i) \\ w_i & \text{if } y_i h_k(x_i) = y_i h_l(x_i) \end{cases}$ 
10:   $\mathbf{w} \leftarrow \frac{\mathbf{w}}{\|\mathbf{w}\|_1}$ 
11: end for
12: Result:  $h_c(x) = \text{sign}(\sum_{j=1}^{|\mathcal{H}|} \alpha_j h_j(x))$ 

```

---

Because the loss function (1) decreases when we scale up  $\vec{\alpha}$ , (3) must be an equality at an optimum solution. Therefore we start with an initial point  $\vec{\alpha}_0$  on the boundary of the feasible set ( $\|\vec{\alpha}_0\|_1 = R$ ). Such initialization can be obtained by using AdaBoost until  $\|\vec{\alpha}\|_1$  reaches  $R$  or (more naively) by assigning all regularization budget  $R$  on a single classifier.

Unlike AdaBoost, which only “adds” base classifiers iteratively, a key component of RBoost is that it can also “remove” classifiers at the same time. In each iteration, RBoost not only chooses a best base classifier  $h_k$  (step 5), but also chooses a worst active (meaning with nonzero coefficient) classifier  $h_l$  (step 6). The  $l_1$  budget on the coefficients is then transferred from the worst classifier  $h_l$  to the best classifier  $h_k$  (step 8), while keeping  $\sum_{j=1}^{|\mathcal{H}|} |\alpha_j| = R$ .

This major difference ensures that RBoost will produce a sparser composite classifier than AdaBoost. The reason for this is twofold: First, the new algorithm promptly removes poor classifiers and relocates resources to better classifiers. This results in a fewer active classifiers, with each single classifier being more effective. Second, the algorithm seeks a solution of the  $l_1$ -regularized loss minimization problem (1) (We will prove this below.). The  $l_1$ -regularization will impose sparsity on the solution and minimizing the loss will ensure good generalization error [9, 12].

Other than this key difference, and the changes it necessitates, RBoost is highly analogous to AdaBoost. This minimal change ensures that RBoost inherits the simplicity and efficiency of AdaBoost while directly incorporating a sparsity mechanism. In this sense it is a natural extension of Adaboost. Indeed, each iteration step of AdaBoost can be viewed as a degenerate case of an iteration step of RBoost. To see this, first note that the AdaBoost step length is  $\varepsilon = \frac{1}{2} \ln \frac{1 - P_{\text{err}}(h_k)}{P_{\text{err}}(h_k)}$ . This reflects the value of the newly chosen classifier. Similarly, in RBoost, the step length  $\frac{1}{2} \ln \frac{P(y h_k(x) > y h_l(x))}{P(y h_k(x) < y h_l(x))}$  can be viewed as a measure of how better  $h_k$  is compared to  $h_l$ .

The important point is that these two step sizes agree when  $h_l = -h_k$ , in which case we have:

$$\begin{aligned} & \frac{1}{2} \ln \frac{P(yh_k(x) > yh_l(x))}{P(yh_k(x) < yh_l(x))} \\ &= \frac{1}{2} \ln \frac{P(yh_k(x) = 1)}{P(yh_k(x) = -1)} = \frac{1}{2} \ln \frac{1 - P_{err}(h_k)}{P_{err}(h_k)} \end{aligned} \quad (4)$$

Now, if we remove the requirement  $\alpha_j > 0$  (meaning  $h_j$  does not have to be an *active* classifier) in step 6 of RBoost, then the worst classifier is just the negation of the best classifier:  $h_l = -h_k$ . In this case,  $\varepsilon$  is the same as that of AdaBoost and all the adjustments in steps 8 and 9 reduce to a normal AdaBoost iteration.

It should also be noted that RBoost has the same theoretical foundation as AdaBoost: it is a ‘‘coordinate descent’’ procedure for solving (2,3). To illustrate this, consider all possible adjustments along the coordinate directions from point  $\vec{\alpha}$ . For any two indices  $j_1, j_2$ , We can subtract  $a/2$  from coefficient  $\alpha_{j_1}$  and add it to  $\alpha_{j_2}$  to keep  $\sum_{j=1}^{|\mathcal{H}|} |\alpha_j| = R$ . Denote the value of the loss function after such adjustment as:

$$\mathcal{L}_{j_1 \rightarrow j_2}(\vec{\alpha}, a) = \mathcal{L}(\alpha_1, \dots, \alpha_{j_1} - \frac{a}{2}, \dots, \alpha_{j_2} + \frac{a}{2}, \dots, \alpha_{|\mathcal{H}|})$$

Notice that  $\mathcal{L}_{j_1 \rightarrow j_2}(\vec{\alpha}, 0) = \mathcal{L}(\vec{\alpha})$ .

The following theorem shows that RBoost operates on a diagonal of the two-coordinate plane yielding the largest gradient descent. Moreover, the step size is chosen to achieve the minimum along that direction.

**Theorem 1.** *In each iteration of Algorithm 2, the choice of  $k, l$  and  $\varepsilon$  satisfy the following properties:*

$$(l, k) = \arg \max_{(j_1, j_2): \alpha_{j_1} > 0} \left( - \frac{\partial \mathcal{L}_{j_1 \rightarrow j_2}(\vec{\alpha}, a)}{\partial a} \Big|_{a=0} \right) \quad (5)$$

$$\varepsilon = \arg \min_{a: a/2 \leq \alpha_l} \mathcal{L}_{l \rightarrow k}(\vec{\alpha}, a) \quad (6)$$

*Proof.* First, the weight distribution on each example is always proportional to the exponential loss of the example:

$$w_i = \frac{1}{Z} \exp(-y_i \sum_{j=1}^{|\mathcal{H}|} \alpha_j h_j(x_i)) \quad (7)$$

where  $Z$  is a normalization factor. Equation (7) is clearly true for the initialization step (step 2). Every time we remove  $\frac{\varepsilon}{2}$  from  $\alpha_l$  to  $\alpha_k$ , the weight adjustments in step 9 guarantee that equation (7) continues to hold.

The loss function can now be written as:  $\mathcal{L}(\vec{\alpha}) = \sum_{i=1}^m Z w_i$ . We then divide the sum  $\sum_{i=1}^m$  into four groups, according to whether  $(y_i h_{j_1}(x_i), y_i h_{j_2}(x_i)) = (1, 1), (-1, 1), (1, -1)$  or  $(-1, -1)$ . These cases correspond to whether  $h_{j_1}$  and  $h_{j_2}$  are right or wrong on an example. After substituting

$\alpha_{j_1}$  with  $\alpha_{j_1} - a/2$ , and  $\alpha_{j_2}$  with  $\alpha_{j_2} + a/2$ , we obtain

$$\begin{aligned} \mathcal{L}_{j_1 \rightarrow j_2}(\vec{\alpha}, a) &= \sum_{\substack{i: \\ y_i h_{j_1}(x_i) = y_i h_{j_2}(x_i)}} Z w_i + \\ & \sum_{\substack{i: y_i h_{j_1}(x_i) = 1 \\ y_i h_{j_2}(x_i) = -1}} Z e^a w_i + \sum_{\substack{i: y_i h_{j_1}(x_i) = -1 \\ y_i h_{j_2}(x_i) = 1}} Z e^{-a} w_i \end{aligned} \quad (8)$$

So

$$\begin{aligned} - \frac{\partial \mathcal{L}_{j_1 \rightarrow j_2}(\vec{\alpha}, a)}{\partial a} &= Z e^{-a} P(yh_{j_1} = -1, yh_{j_2} = 1) \\ & \quad - Z e^a P(yh_{j_1} = 1, yh_{j_2} = -1) \end{aligned} \quad (9)$$

Letting  $a = 0$  yields:

$$\begin{aligned} & - \frac{\partial \mathcal{L}_{j_1 \rightarrow j_2}(\vec{\alpha}, a)}{\partial a} \Big|_{a=0} \\ &= Z P(yh_{j_1} = -1, yh_{j_2} = 1) - Z P(yh_{j_1} = 1, yh_{j_2} = -1) \\ &= Z P(yh_{j_1} = -1) - Z P(yh_{j_2} = -1) \\ &= \frac{Z}{2} \text{edge}(h_{j_2}, \mathbf{w}) - \frac{Z}{2} \text{edge}(h_{j_1}, \mathbf{w}) \end{aligned}$$

To maximize this quantity under the constraint of  $\alpha_{j_1} > 0$ , we must choose  $j_2 = \arg \max_j \text{edge}(h_j, \mathbf{w})$ ,  $j_1 = \arg \min_{j: \alpha_j > 0} \text{edge}(h_j, \mathbf{w})$ . This is exactly the way we choose  $l$  and  $k$  in the algorithm. So equation (5) is proved.

From equation (9), with  $j_1 = l, j_2 = k$ ,  $\mathcal{L}_{j_1 \rightarrow j_2}(\vec{\alpha}, a)$  achieves minimum when  $Z e^{-a} P(yh_{j_1} = -1, yh_{j_2} = 1) - Z e^a P(yh_{j_1} = 1, yh_{j_2} = -1) = 0$ . Therefore

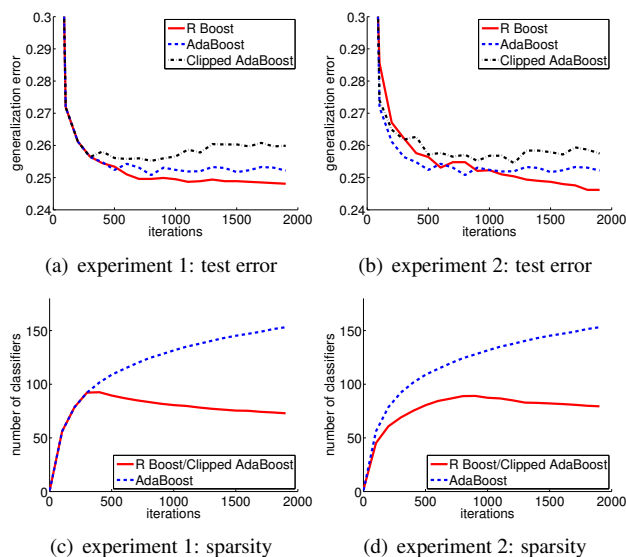
$$\begin{aligned} a &= \frac{1}{2} \ln \frac{Z P(yh_l(x) = -1, yh_k(x) = 1)}{Z P(yh_l(x) = 1, yh_k(x) = -1)} \\ &= \frac{1}{2} \ln \frac{P(yh_k(x) > yh_l(x))}{P(yh_k(x) < yh_l(x))} \end{aligned} \quad (10)$$

That is how  $\varepsilon$  is selected in the algorithm, with the additional constraint  $\varepsilon/2 \leq \alpha_l$  to ensure all coefficients remain positive. This proves (6).  $\square$

## 4. EXPERIMENTS

In all experiments, we used 100 training examples and 500 testing examples. We used single stumps as base classifiers. We first compare RBoost with AdaBoost using a benchmark data set called *ringnorm*. The ringnorm examples are generated from two 20-dimensional Gaussian distributions, one  $N(\mathbf{0}, 4I)$ , the other  $N(\mu, I)$  with  $\mu = (a, a, \dots, a)$  and  $a = 1/\sqrt{20}$ . We performed two experiments; each with 20 trials.

In experiment 1, we used AdaBoost to initialize RBoost by first running AdaBoost until  $\sum_k \alpha_k = R$ , then running RBoost from that initial condition. The results, Fig1(a) and 1(c), show that RBoost sparsifies the result of AdaBoost while continuing to lower the test error. For fair comparison, we



**Fig. 1.** Comparison of RBoost and AdaBoost on ringnorm

also included the generalization results for *clipped AdaBoost*, which simply drops the base classifiers with the smallest coefficients in the AdaBoost ensemble in order to match the number of active classifiers in RBoost. Fig1(a) shows that this naive method of obtaining sparsity sacrifices classification accuracy. In experiment 2, we used an even simpler initialization of RBoost (assign all weight  $R$  to a single classifier) and set  $R = 100$ . The results, Fig1(b) and 1(d), show that RBoost recovers from this naive initialization and eventually surpasses AdaBoost in both sparsity and accuracy.

We then tested RBoost on four real world data sets from the UCI repository: *German* (determining credit risk based on financial data), *Heart* (classifying each subject as having/not having heart disease using vectors of medical attributes), *Sonar* (classifying surface as rock/metal using sonar spectral energy distribution), and *Spam* (identifying spam email based on word frequencies). For each data set we are interested in the best error rate achieved before over fitting occurs.

In each experiment, we averaged the results of 10 trials. We ran 500 iterations for each trial. We used the naive initialization strategy for RBoost with  $R = 40$  (larger  $R$  yield the same result). The results are shown in Table 1. In all data sets, RBoost learns a sparser classifier with equal or better generalization performance.

**Table 1.** Comparison of RBoost and AdaBoost on real data

|               | RBoost     |        | AdaBoost   |        |
|---------------|------------|--------|------------|--------|
|               | error rate | number | error rate | number |
| <i>German</i> | 24.9%      | 47     | 25.3%      | 60     |
| <i>Heart</i>  | 18.1%      | 11     | 19.2%      | 16     |
| <i>Sonar</i>  | 12.1%      | 52     | 13.8%      | 157    |
| <i>Spam</i>   | 10.7%      | 26     | 10.7%      | 59     |

## 5. CONCLUSION

The RBoost boosting algorithm has been introduced to address the need for an efficient solution method for the  $l_1$ -regularized loss minimization problem. We provided both theoretical and experimental evidence that RBoost produces sparse classifiers without compromising classification accuracy or computational efficiency. The algorithm is robust to the selection of the regularization parameter  $R$  and outperforms AdaBoost even when started with very naive initializations. When started from an initial condition produced by Adaboost, it actually sparsifies the Adaboost solution *and* improves generalization. More systematic study of methods for selection of the regularization parameter  $R$  and alternative initialization methods remain questions for further investigation.

## 6. REFERENCES

- [1] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," in *European Conf. on Computational Learning Theory*, 1995, pp. 23–37.
- [2] Y. Freund and R. E. Schapire, "Experiments with a new boosting algorithm," in *Proc. Thirteenth Int. Conf. on Machine Learning*, 1996, pp. 148–156.
- [3] H. Drucker and C. Cortes, "Boosting decision trees," *Advances in Neural Information Processing Systems 8: Proc. 1995 Conference*, 1996.
- [4] Leo Breiman, "Arcing classifiers," *The Annals of Statistics*, vol. 26, no. 3, pp. 801–824, 1998.
- [5] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Annals of Statistics*, vol. 29, pp. 1189–1232, 2001.
- [6] G. Ratsch, T. Onoda, and K. R. Müller, "Soft margins for adaboost," *Machine Learning*, vol. 42, no. 3, pp. 287–320, March 2001.
- [7] L. Breiman, "Prediction games and arcing algorithms," *Neural Comp.*, vol. 11, no. 7, pp. 1493–1517, Oct. 1999.
- [8] Wenxin Jiang, "Process consistency for adaboost," *Annals of Statistics*, vol. 32, 2000.
- [9] S. Rosset, J. Zhu, and T. Hastie, "Boosting as a regularized path to a maximum margin classifier," *Journal of Machine Learning Research*, vol. 5, pp. 941–973, 2004.
- [10] E. J. Candes and T. Tao, "Decoding by linear programming," *Information Theory, IEEE Trans. on*, vol. 51, no. 12, pp. 4203–4215, 2005.
- [11] Y. T. Xi, Z. J. Xiang, P. J. Ramadge, and R. E. Schapire, "Speed and sparsity of regularized boosting," *Technical Report, Dept. Elec. Eng., Princeton University*, 2008.
- [12] R. E. Schapire, Y. Freund, P. Bartlett, and W.S. Lee, "Boosting the margin: A new explanation for the effectiveness of voting methods," *The Annals of Statistics*, vol. 26, no. 5, pp. 1651–1686, 1998.