17th Annual Conference on Computational Learning Theory, 2004.

Boosting Based on a Smooth Margin*

Cynthia Rudin[1], Robert E. Schapire[2], and Ingrid Daubechies[1]

[1] Princeton University, Program in Applied and Computational Mathematics
Fine Hall, Washington Road, Princeton, NJ 08544-1000
{crudin,ingrid}@math.princeton.edu
[2] Princeton University, Department of Computer Science
35 Olden St., Princeton, NJ 08544
schapire@cs.princeton.edu

Abstract. We study two boosting algorithms, *Coordinate Ascent Boosting* and *Approximate Coordinate Ascent Boosting*, which are explicitly designed to produce maximum margins. To derive these algorithms, we introduce a smooth approximation of the margin that one can maximize in order to produce a maximum margin classifier. Our first algorithm is simply coordinate ascent on this function, involving a line search at each step. We then make a simple approximation of this line search to reveal our second algorithm. These algorithms are proven to asymptotically achieve maximum margins, and we provide two convergence rate calculations. The second calculation yields a faster rate of convergence than the first, although the first gives a more explicit (still fast) rate. These algorithms are very similar to AdaBoost in that they are based on coordinate ascent, easy to implement, and empirically tend to converge faster than other boosting algorithms. Finally, we attempt to understand AdaBoost in terms of our smooth margin, focusing on cases where AdaBoost exhibits cyclic behavior.

1 Introduction

Boosting is currently a popular and successful technique for classification. The first practical boosting algorithm was AdaBoost, developed by Freund and Schapire [4]. The goal of boosting is to construct a "strong" classifier using only a training set and a "weak" learning algorithm. A weak learning algorithm produces "weak" classifiers, which are only required to classify somewhat better than a random guess. For an introduction, see the review paper of Schapire [13].

In practice, AdaBoost often tends not to overfit (only slightly in the limit [5]), and performs remarkably well on test data. The leading explanation for AdaBoost's ability to generalize is the *margin theory*. According to this theory, the margin can be viewed as a confidence measure of a classifier's predictive ability. This theory is based on (loose) generalization bounds, e.g., the bounds of Schapire et al. [14] and Koltchinskii and Panchenko [6]. Although the empirical

* This research was partially supported by NSF Grants IIS-0325500, DMS-9810783, and ANI-0085984.

success of a boosting algorithm depends on many factors (e.g., the type of data and how noisy it is, the capacity of the weak learning algorithm, the number of boosting iterations before stopping, other means of regularization, entire margin distribution), the margin theory does provide a reasonable qualitative explanation (though not a complete explanation) of AdaBoost's success, both empirically and theoretically. However, AdaBoost has not been shown to achieve the largest possible margin. In fact, the opposite has been recently proved, namely that AdaBoost may converge to a solution with margin significantly below the maximum value [11]. This was proved for specific cases where AdaBoost exhibits cyclic behavior; such behavior is common when there are very few "support vectors".

Since AdaBoost's performance is not well understood, a number of other boosting algorithms have emerged that directly aim to maximize the margin. Many of these algorithms are not as easy to implement as AdaBoost, or require a significant amount of calculation at each step, e.g., the solution of a linear program (LP-AdaBoost [5]), an optimization over a non-convex function (DOOM [7]) or a huge number of very small steps (ϵ-boosting, where convergence to a maximum margin solution has not been proven, even as the step size vanishes [10]). These extra calculations may slow down the convergence rate dramatically. Thus, we compare our new algorithms with arc-gv [2] and AdaBoost* [9]; these algorithms are as simple to program as AdaBoost and have convergence guarantees with respect to the margin. Our new algorithms are more aggressive than both arc-gv and AdaBoost*, providing an explanation for their empirically faster convergence rate.

In terms of theoretical rate guarantees, our new algorithms converge to a maximum margin solution with a polynomial convergence rate. Namely, within $poly(1/\epsilon)$ iterations, they produce a classifier whose margin is within ϵ of the maximum possible margin. Arc-gv is proven to converge to a maximum margin solution asymptotically [2, 8], but we are not aware of any proven convergence rate. AdaBoost* [9] converges to a solution within ϵ of the maximum margin in $2(\log_2 m)/\epsilon^2$ steps (where the user specifies a fixed value of ϵ); there is a tradeoff between user-determined accuracy and convergence rate for this algorithm. In practice, AdaBoost* converges very slowly since it is not aggressive; it takes small steps (though it has the nice convergence rate guarantee stated above). In fact, if the weak learner always finds a weak classifier with a large edge (i.e., if the weak learning algorithm performs well on the weighted training data), the convergence of AdaBoost* can be especially slow.

The two new boosting algorithms we introduce (which are presented in [12] without analysis) are based on coordinate ascent. For AdaBoost, the fact that it is a minimization algorithm based on coordinate descent does not imply convergence to a maximum margin solution. For our new algorithms, we can directly use the fact that they are coordinate ascent algorithms to help show convergence to a maximum margin solution, since they make progress towards increasing a differentiable approximation of the margin (a "smooth margin function") at every iteration.

To summarize, the advantages of our new algorithms, *Coordinate Ascent Boosting* and *Approximate Coordinate Ascent Boosting* are as follows:

– They empirically tend to converge faster than both arc-gv and AdaBoost*.
– They provably converge to a maximum margin solution asymptotically. This convergence is robust, in that we do not require the weak learning algorithm to produce the best possible classifier at every iteration; only a sufficiently good classifier is required.
– They have convergence rate guarantees that are polynomial in $1/\epsilon$.
– They are as easy to implement as AdaBoost, arc-gv, and AdaBoost*.
– These algorithms have theoretical and intuitive justification: they make progress with respect to a smooth version of the margin, and operate via coordinate ascent.

Finally, we use our smooth margin function to analyze AdaBoost. Since AdaBoost's good generalization properties are not completely explained by the margin theory, and still remain somewhat mysterious, we study properties of AdaBoost via our smooth margin function, focusing on cases where cyclic behavior occurs. "Cyclic behavior for AdaBoost" means the weak learning algorithm repeatedly chooses the same sequence of weak classifiers, and the weight vectors repeat with a given period. This has been proven to occur in special cases, and occurs often in low dimensions (i.e., when there are few "support vectors") [11].

Our results concerning AdaBoost and our smooth margin are as follows: first, the value of the smooth margin increases if and only if AdaBoost takes a large enough step. Second, the value of the smooth margin must decrease for at least one iteration of a cycle unless all edge values are identical. Third, if all edges in a cycle are identical, then support vectors are misclassified by the same number of weak classifiers during the cycle.

Here is the outline: in Section 2, we introduce our notation and the AdaBoost algorithm. In Section 3, we describe the smooth margin function that our algorithms are based on. In Section 4, we describe Coordinate Ascent Boosting (Algorithm 1) and Approximate Coordinate Ascent Boosting (Algorithm 2), and in Section 5, the convergence of these algorithms is discussed. Experimental trials on artificial data are presented in Section 6 to illustrate the comparison with other algorithms. In Section 7, we show connections between AdaBoost and our smooth margin function.

2 Notation and Introduction to AdaBoost

The training set consists of examples with labels $\{(\mathbf{x}_i, y_i)\}_{i=1,...,m}$, where $(\mathbf{x}_i, y_i) \in \mathcal{X} \times \{-1, 1\}$. The space $\mathcal{X}$ never appears explicitly in our calculations. Let $\mathcal{H} = \{h_1, ..., h_n\}$ be the set of *all possible* weak classifiers that can be produced by the weak learning algorithm, where $h_j : \mathcal{X} \to \{1, -1\}$. We assume that if h_j appears in $\mathcal{H}$, then $-h_j$ also appears in $\mathcal{H}$ (i.e., $\mathcal{H}$ is symmetric). Since our classifiers are binary, and since we restrict our attention to their behavior on a finite training set, we can assume that n is finite. We think of n as being large, $m \ll n$, so a gradient descent calculation over an n dimensional space is impractical; hence AdaBoost uses coordinate descent instead, where only one weak classifier is chosen at each iteration.

We define an $m \times n$ matrix $\mathbf{M}$ where $M_{ij} = y_i h_j(\mathbf{x}_i)$, i.e., $M_{ij} = +1$ if training example i is classified correctly by weak classifier h_j, and -1 otherwise. We assume that no column of $\mathbf{M}$ has all $+1$'s, that is, no weak classifier can classify all the training examples correctly. (Otherwise the learning problem is trivial.) Although $\mathbf{M}$ is too large to be explicitly constructed in practice, mathematically, it acts as the only "input" to AdaBoost, containing all the necessary information about the weak learner and training examples.

AdaBoost computes a set of coefficients over the weak classifiers. The (unnormalized) coefficient vector at iteration t is denoted $\boldsymbol{\lambda}_t$. Since the algorithms we describe all have positive increments, we take $\boldsymbol{\lambda} \in \mathbb{R}^n_+$. We define a seminorm by $|||\boldsymbol{\lambda}||| := \min_{\boldsymbol{\lambda}'}\{\|\boldsymbol{\lambda}'\|_1 \text{ such that } \forall j : \lambda_j - \lambda_{\tilde{j}} = \lambda'_j - \lambda'_{\tilde{j}}\}$ where $\tilde{j}$ is the index for $-h_j$, and define $s(\boldsymbol{\lambda}) := \sum_{j=1}^n \lambda_j$, noting $s(\boldsymbol{\lambda}) \geq |||\boldsymbol{\lambda}|||$. For the (non-negative) vectors $\boldsymbol{\lambda}_t$ generated by AdaBoost, we will denote $s_t := s(\boldsymbol{\lambda}_t)$. The final combined classifier that AdaBoost outputs is $f_{Ada} = \sum_{j=1}^n (\lambda_{t_{max},j}/|||\boldsymbol{\lambda}_{t_{max}}|||)h_j$. The *margin of training example* i is defined to be $y_i f_{Ada}(\mathbf{x}_i)$, or equivalently, $(\mathbf{M}\boldsymbol{\lambda})_i/|||\boldsymbol{\lambda}|||$.

A boosting algorithm maintains a distribution, or set of weights, over the training examples that is updated at each iteration, which is denoted $\mathbf{d}_t \in \Delta_m$, and $\mathbf{d}_t^T$ is its transpose. Here, Δ_m denotes the simplex of m-dimensional vectors with non-negative entries that sum to 1. At each iteration t, a weak classifier h_{j_t} is selected by the weak learning algorithm. The *probability of error* of h_{j_t} at time t on the weighted training examples is $d_- := \sum_{\{i:M_{ij_t}=-1\}} d_{t,i}$. Also, denote $d_+ := 1 - d_-$, and define $\mathcal{I}_+ := \{i : M_{ij_t} = +1\}$ and $\mathcal{I}_- := \{i : M_{ij_t} = -1\}$. Note that $d_+, d_-, \mathcal{I}_+$, and $\mathcal{I}_-$ depend on t; the iteration number will be clear from the context. The *edge* of weak classifier j_t at time t is $r_t := (\mathbf{d}_t^T\mathbf{M})_{j_t}$, which can be written as $r_t = (\mathbf{d}_t^T\mathbf{M})_{j_t} = \sum_{i\in\mathcal{I}_+} d_{t,i} - \sum_{i\in\mathcal{I}_-} d_{t,i} = d_+ - d_- = 1 - 2d_-$. Thus, a smaller edge indicates a higher probability of error. Note that $d_+ = (1 + r_t)/2$ and $d_- = (1 - r_t)/2$. Also define $\gamma_t := \tanh^{-1} r_t$.

We wish our learning algorithms to have robust convergence, so we will not require the weak learning algorithm to produce the weak classifier with the largest possible edge value at each iteration. Rather, we only require a weak classifier whose edge exceeds ρ, where ρ is the largest possible margin that can be attained for $\mathbf{M}$, i.e., we use the "non-optimal" case for our analysis. AdaBoost in the "optimal case" means $j_t \in \text{argmax}_j(\mathbf{d}_t^T\mathbf{M})_j$, and AdaBoost in the "non-optimal" case means $j_t \in \{j : (\mathbf{d}_t^T\mathbf{M})_j \geq \rho\}$.

To achieve the best indication of a small probability of error (for margin-based bounds), our goal is to find a $\tilde{\boldsymbol{\lambda}} \in \Delta_n$ that maximizes the minimum margin over training examples, $\min_i (\mathbf{M}\tilde{\boldsymbol{\lambda}})_i$ (or equivalently $\min_i y_i f_{Ada}(\mathbf{x}_i)$), i.e., we wish to find a vector $\tilde{\boldsymbol{\lambda}} \in \text{argmax}_{\bar{\boldsymbol{\lambda}}\in\Delta_n} \min_i(\mathbf{M}\bar{\boldsymbol{\lambda}})_i = \text{argmax}_{\boldsymbol{\lambda}\in\mathbb{R}^n} \min_i(\mathbf{M}\boldsymbol{\lambda})_i/|||\boldsymbol{\lambda}|||$. We call the minimum margin over training examples (i.e., $\min_i(\mathbf{M}\boldsymbol{\lambda})_i/|||\boldsymbol{\lambda}|||$) the *margin* of classifier $\boldsymbol{\lambda}$, denoted $\mu(\boldsymbol{\lambda})$. Any training example that achieves this minimum margin is a *support vector*. Due to the von Neumann Min-Max

Theorem, $\min_{\mathbf{d}\in\Delta_m}\max_j(\mathbf{d}^T\mathbf{M})_j = \max_{\bar{\boldsymbol{\lambda}}\in\Delta_n}\min_i(\mathbf{M}\bar{\boldsymbol{\lambda}})_i$. We denote this value by ρ.

Figure 1 shows pseudocode for AdaBoost. At each iteration, the distribution $\mathbf{d}_t$ is updated and renormalized (Step 3a), classifier j_t with sufficiently large edge is selected (Step 3b), and the weight of that classifier is updated (Step 3e).

1. **Input:** Matrix $\mathbf{M}$, No. of iterations t_{max}
2. **Initialize:** $\lambda_{1,j} = 0$ for $j = 1, ..., n$
3. **Loop for** $t = 1, ..., t_{max}$
   (a) $d_{t,i} = e^{-(\mathbf{M}\boldsymbol{\lambda}_t)_i} / \sum_{\bar{i}=1}^{m} e^{-(\mathbf{M}\boldsymbol{\lambda}_t)_{\bar{i}}}$ for $i = 1, ..., m$
   (b) $\begin{cases} j_t \in \text{argmax}_j (\mathbf{d}_t^T\mathbf{M})_j & \text{"optimal" case} \\ j_t \in \{j : (\mathbf{d}_t^T\mathbf{M})_j > \rho\} & \text{"non-optimal" case} \end{cases}$
   (c) $r_t = (\mathbf{d}_t^T\mathbf{M})_{j_t}$
   (d) $\alpha_t = \frac{1}{2}\ln\left(\frac{1+r_t}{1-r_t}\right)$
   (e) $\boldsymbol{\lambda}_{t+1} = \boldsymbol{\lambda}_t + \alpha_t\mathbf{e}_{j_t}$, where $\mathbf{e}_{j_t}$ is 1 in position j_t and 0 elsewhere.
4. **Output:** $\boldsymbol{\lambda}_{t_{max}} / |||\boldsymbol{\lambda}_{t_{max}}|||$

Fig. 1. Pseudocode for the AdaBoost algorithm.

AdaBoost is known to be a coordinate descent algorithm for minimizing $F(\boldsymbol{\lambda}) := \sum_{i=1}^{m} e^{-(\mathbf{M}\boldsymbol{\lambda})_i}$ [1]. The proof (for the optimal case) is that the choice of weak classifier j_t is given by: $j_t \in \text{argmax}_j \left[-dF(\boldsymbol{\lambda}_t + \alpha\mathbf{e}_j)/d\alpha\Big|_{\alpha=0}\right] = \text{argmax}_j(\mathbf{d}_t^T\mathbf{M})_j$, and the step size AdaBoost chooses at iteration t is α_t, where α_t satisfies the equation for the line search along direction j_t: $0 = -dF(\boldsymbol{\lambda}_t + \alpha_t\mathbf{e}_{j_t})/d\alpha_t$. Convergence in the non-separable case is fully understood [3]. In the separable case ($\rho > 0$), the minimum value of F is 0 and occurs as $|||\boldsymbol{\lambda}||| \to \infty$; this tells us nothing about the value of the margin, i.e., an algorithm which simply minimizes F can achieve an arbitrarily bad margin. So it must be the *process* of coordinate descent which awards AdaBoost its ability to increase margins, not simply AdaBoost's ability to minimize F.

3 The Smooth Margin Function $G(\boldsymbol{\lambda})$

We wish to consider a function that, unlike F, actually tells us about the value of the margin. Our new function G is defined for $\boldsymbol{\lambda} \in \mathbb{R}_+^n$, $s(\boldsymbol{\lambda}) > 1$ by:

$$G(\boldsymbol{\lambda}) := \frac{-\ln F(\boldsymbol{\lambda})}{s(\boldsymbol{\lambda})} = \frac{-\ln\left(\sum_{i=1}^{m} e^{-(\mathbf{M}\boldsymbol{\lambda})_i}\right)}{\sum_j \lambda_j}. \tag{1}$$

One can think of G as a smooth approximation of the margin, since it depends on the entire margin distribution when $s(\boldsymbol{\lambda})$ is finite, and weights training examples with small margins much more highly than examples with larger margins. The function G also bears a resemblance to the objective implicitly used for ϵ-boosting [10]. Note that since $s(\lambda) \geq |||\boldsymbol{\lambda}|||$, we have $G(\boldsymbol{\lambda}) \leq -(\ln F(\boldsymbol{\lambda}))/|||\boldsymbol{\lambda}|||$. Lemma 1 (parts of which appear in [12]) shows that G has many nice properties.

Lemma 1.

1. *$G(\boldsymbol{\lambda})$ is a concave function (but not necessarily strictly concave) in each "shell" where $s(\boldsymbol{\lambda})$ is fixed. In addition, $G(\boldsymbol{\lambda})$ becomes concave when $s(\boldsymbol{\lambda})$ becomes large.*
2. *$G(\boldsymbol{\lambda})$ becomes concave when $|||\boldsymbol{\lambda}|||$ becomes large.*
3. *As $|||\boldsymbol{\lambda}||| \to \infty$, $-(\ln F(\boldsymbol{\lambda}))/|||\boldsymbol{\lambda}||| \to \mu(\boldsymbol{\lambda})$.*
4. *The value of $G(\boldsymbol{\lambda})$ increases radially, i.e., $dG(\boldsymbol{\lambda}(1+a))/da\Big|_{a=0} > 0$*

It follows from 3 and 4 that the maximum value of G is the maximum value of the margin, since for each $\boldsymbol{\lambda}$, we may construct a $\boldsymbol{\lambda}'$ such that $G(\boldsymbol{\lambda}') = -\ln F(\boldsymbol{\lambda})/|||\boldsymbol{\lambda}|||$. We omit the proofs of 1 and 4. Note that if $|||\boldsymbol{\lambda}|||$ is large, $s(\boldsymbol{\lambda})$ is large since $|||\boldsymbol{\lambda}||| \leq s(\boldsymbol{\lambda})$. Thus, 2 follows from 1.

Proof. (of property 3)

$$me^{-\mu(\boldsymbol{\lambda})|||\boldsymbol{\lambda}|||} = \sum_{i=1}^{m} e^{-\min_\ell (\mathbf{M}\boldsymbol{\lambda})_\ell} \geq \sum_{i=1}^{m} e^{-(\mathbf{M}\boldsymbol{\lambda})_i} > e^{-\min_i (\mathbf{M}\boldsymbol{\lambda})_i} = e^{-\mu(\boldsymbol{\lambda})|||\boldsymbol{\lambda}|||},$$

$$\text{hence,} \quad -(\ln m)/|||\boldsymbol{\lambda}||| + \mu(\boldsymbol{\lambda}) \leq -(\ln F(\boldsymbol{\lambda}))/|||\boldsymbol{\lambda}||| < \mu(\boldsymbol{\lambda}). \tag{2}$$

□

The properties of G shown in Lemma 1 outline the reasons why we choose to maximize G using coordinate ascent; namely, maximizing G leads to a maximum margin solution, and the region where G is near its maximum value is concave.

4 Derivation of Algorithms

We now suggest two boosting algorithms (derived without analysis in [12]) that aim to maximize the margin explicitly (like arc-gv and AdaBoost*) and are based on coordinate ascent (like AdaBoost). Our new algorithms choose the direction of ascent (value of j_t) using the same formula as AdaBoost, arc-gv, and AdaBoost*, i.e., $j_t \in \operatorname{argmax}_j (\mathbf{d}_t^T \mathbf{M})_j$. Thus, our new algorithms require exactly the same type of weak learning algorithm.

To help with the analysis later, we will write recursive equations for F and G. The recursive equation for F (derived only using the definition) is:

$$F(\boldsymbol{\lambda}_t + \alpha \mathbf{e}_{j_t}) = \frac{\cosh(\gamma_t - \alpha)}{\cosh \gamma_t} F(\boldsymbol{\lambda}_t). \tag{3}$$

By definition of G, we know $-\ln F(\boldsymbol{\lambda}_t) = s_t G(\boldsymbol{\lambda}_t)$ and $-\ln F(\boldsymbol{\lambda}_t + \alpha \mathbf{e}_{j_t}) = (s_t + \alpha) G(\boldsymbol{\lambda}_t + \alpha \mathbf{e}_{j_t})$. From (3), we find a recursive equation for G:

$$(s_t+\alpha)G(\boldsymbol{\lambda}_t+\alpha\mathbf{e}_{j_t}) = -\ln F(\boldsymbol{\lambda}_t) - \ln\left(\frac{\cosh(\gamma_t-\alpha)}{\cosh\gamma_t}\right) = s_t G(\boldsymbol{\lambda}_t) + \int_{\gamma_t-\alpha}^{\gamma_t} \tanh u \, du. \tag{4}$$

We shall look at two different algorithms; in the first, we assign to α_t the value α that maximizes $G(\boldsymbol{\lambda}_t+\alpha\mathbf{e}_{j_t})$, which requires solving an implicit equation. In the second algorithm, inspired by the first, we pick a value for α_t that can be computed in a straightforward way, even though it is not a maximizer of $G(\boldsymbol{\lambda}_t + \alpha \mathbf{e}_{j_t})$. In both cases, the algorithm starts by simply running AdaBoost until $G(\boldsymbol{\lambda})$ becomes positive, which must happen (in the separable case) since:

Lemma 2. *In the separable case (where $\rho > 0$), AdaBoost achieves a positive value for $G(\boldsymbol{\lambda}_t)$ in at most $\ulcorner -2\ln F(\boldsymbol{\lambda}_1)/\ln(1-\rho^2)\urcorner + 1$ iterations.*

The proof of Lemma 2 (which is omitted) uses (3). Denote $\boldsymbol{\lambda}_1^{[1]},...,\boldsymbol{\lambda}_t^{[1]}$ to be a sequence of coefficient vectors generated by Algorithm 1, and $\boldsymbol{\lambda}_1^{[2]},...,\boldsymbol{\lambda}_t^{[2]}$ to be generated by Algorithm 2. Similarly, we distinguish sequences $\alpha_t^{[1]}$ and $\alpha_t^{[2]}$, $g_t^{[1]} := G(\boldsymbol{\lambda}_t^{[1]})$, $g_t^{[2]} := G(\boldsymbol{\lambda}_t^{[2]})$, $s_t^{[1]}$, and $s_t^{[2]}$. Sometimes we compare the behavior of Algorithms 1 and 2 based on one iteration (from t to $t+1$) as if they had started from the same coefficient vector at iteration t; we denote this vector by $\boldsymbol{\lambda}_t$. When both Algorithms 1 and 2 satisfy a set of equations, we will remove the superscripts $^{[1]}$ and $^{[2]}$. Although sequences such as j_t, r_t, γ_t, and d_t are also different for Algorithms 1 and 2, we leave the notation without the superscript.

4.1 Algorithm 1: Coordinate Ascent Boosting

Rather than considering coordinate descent on F as in AdaBoost, let us consider coordinate ascent on G. In what follows, we will use only positive values of G, as we have justified above. The choice of direction j_t at iteration t (in the optimal case) obeys: $j_t \in \operatorname*{argmax}_j dG(\boldsymbol{\lambda}_t^{[1]} + \alpha\mathbf{e}_j)/d\alpha\Big|_{\alpha=0}$, that is,

$$j_t \in \operatorname*{argmax}_j \left[\frac{\sum_{i=1}^m e^{-(\mathbf{M}\boldsymbol{\lambda}_t^{[1]})_i} M_{ij}}{F(\boldsymbol{\lambda}_t^{[1]})}\right]\frac{1}{s_t^{[1]}} + \frac{\ln(F(\boldsymbol{\lambda}_t^{[1]}))}{\left(s_t^{[1]}\right)^2}.$$

Of these two terms on the right, the second term does not depend on j, and the first term is simply a constant times $(\mathbf{d}_t^T\mathbf{M})_j$. Thus the same direction will be chosen here as for AdaBoost. The "non-optimal" setting we define for this algorithm will be the same as AdaBoost's, so Step 3b of this new algorithm will be the same as AdaBoost's.

To determine the step size, ideally we would like to maximize $G(\boldsymbol{\lambda}_t^{[1]} + \alpha\mathbf{e}_{j_t})$ with respect to α, i.e., we will define $\alpha_t^{[1]}$ to obey $dG(\boldsymbol{\lambda}_t^{[1]} + \alpha\mathbf{e}_{j_t})/d\alpha = 0$ for $\alpha = \alpha_t^{[1]}$. Differentiating (4) with respect to α (while incorporating $dG(\boldsymbol{\lambda}_t^{[1]} + \alpha\mathbf{e}_{j_t})/d\alpha = 0$) gives the following condition for $\alpha_t^{[1]}$:

$$G(\boldsymbol{\lambda}_{t+1}^{[1]}) = G(\boldsymbol{\lambda}_t^{[1]} + \alpha_t^{[1]}\mathbf{e}_{j_t}) = \tanh(\gamma_t - \alpha_t^{[1]}). \tag{5}$$

There is not a nice analytical solution for $\alpha_t^{[1]}$, but minimization of $G(\boldsymbol{\lambda}_t^{[1]} + \alpha\mathbf{e}_{j_t})$ is 1-dimensional so it can be performed quickly. Hence we have defined the first of our new boosting algorithms: coordinate ascent on G, implementing a line search at each iteration. To clarify the line search step at iteration t using (5) and (4), we use $G(\boldsymbol{\lambda}_t^{[1]})$, γ_t, and $s_t^{[1]}$ to solve for $\alpha_t^{[1]}$ that satisfies:

$$s_t^{[1]}G(\boldsymbol{\lambda}_t^{[1]}) + \ln\left(\frac{\cosh\gamma_t}{\cosh(\gamma_t - \alpha_t^{[1]})}\right) = (s_t^{[1]} + \alpha_t^{[1]})\tanh(\gamma_t - \alpha_t^{[1]}). \tag{6}$$

Summarizing, we define Algorithm 1 as follows:

– First, use AdaBoost (Figure 1) until $G(\boldsymbol{\lambda}_t^{[1]})$ defined by (1) is positive. At this point, replace Step 3d of AdaBoost as prescribed: $\alpha_t^{[1]}$ equals the (unique) solution of (6). Proceed, using this modified iterative procedure.

Let us rearrange the equation slightly. Using the notation $g_{t+1}^{[1]} := G(\boldsymbol{\lambda}_{t+1}^{[1]})$ in (5), we find that $\alpha_t^{[1]}$ satisfies the following (implicitly):

$$\alpha_t^{[1]} = \gamma_t - \tanh^{-1}(g_{t+1}^{[1]}) = \tanh^{-1} r_t - \tanh^{-1}(g_{t+1}^{[1]}) = \frac{1}{2}\ln\left[\frac{1+r_t}{1-r_t}\frac{1-g_{t+1}^{[1]}}{1+g_{t+1}^{[1]}}\right]. \quad (7)$$

For any $\boldsymbol{\lambda} \in \mathbb{R}_+^n$, from (2) and since $|||\boldsymbol{\lambda}||| \leq s(\boldsymbol{\lambda})$, we have $G(\boldsymbol{\lambda}) < \rho$. Consequently, $g_{t+1}^{[1]} < \rho \leq r_t$, so $\alpha_t^{[1]}$ is strictly positive. On the other hand, since $G(\boldsymbol{\lambda}_{t+1}^{[1]}) \geq G(\boldsymbol{\lambda}_t^{[1]})$, we again have $G(\boldsymbol{\lambda}_{t+1}^{[1]}) > 0$, and thus $\alpha_t^{[1]} \leq \gamma_t$.

4.2 Algorithm 2: Approximate Coordinate Ascent Boosting

The second of our two new boosting algorithms avoids the line search of Algorithm 1, and is even slightly more aggressive. It performs very similarly to Algorithm 1 in our experiments. To define this algorithm, we consider the following approximate solution to the maximization problem (5):

$$G(\boldsymbol{\lambda}_t^{[2]}) = \tanh(\gamma_t - \alpha_t^{[2]}), \quad \text{or more explicitly,} \quad (8)$$

$$\alpha_t^{[2]} = \gamma_t - \tanh^{-1}(g_t^{[2]}) = \tanh^{-1} r_t - \tanh^{-1}(g_t^{[2]}) = \frac{1}{2}\ln\left[\frac{1+r_t}{1-r_t}\frac{1-g_t^{[2]}}{1+g_t^{[2]}}\right]. \quad (9)$$

This update still yields an increase in G. (This can be shown using (4) and the monotonicity of tanh.) Summarizing, we define Algorithm 2 as the iterative procedure of AdaBoost (Figure 1) with one change:

– Replace Step 3d of AdaBoost as follows:

$$\alpha_t^{[2]} = \frac{1}{2}\ln\left(\frac{1+r_t}{1-r_t}\frac{1-g_t^{[2]}}{1+g_t^{[2]}}\right), g_t^{[2]} := \max\{0, G(\boldsymbol{\lambda}_t^{[2]})\},$$

where G is defined in (1). (Note that we could also have written the procedure in the same way as for Algorithm 1. As long as $G(\boldsymbol{\lambda}_t^{[2]}) \leq 0$, this update is the same as in AdaBoost.)

Algorithm 2 is slightly more aggressive than Algorithm 1, in the sense that it picks a larger relative step size α_t, albeit not as large as the step size defined by AdaBoost itself. If Algorithm 1 and Algorithm 2 were started at the same position $\boldsymbol{\lambda}_t$, with $g_t := G(\boldsymbol{\lambda}_t)$, then Algorithm 2 would always take a slightly larger step than Algorithm 1; since $g_{t+1}^{[1]} > g_t$, we can see from (7) and (9) that $\alpha_t^{[1]} < \alpha_t^{[2]}$.

As a remark, if we use the updates of Algorithms 1 or 2 from the start, they would also reach a positive margin quickly. In fact, after at most $\ulcorner 2\ln F(\boldsymbol{\lambda}_1)/[-\ln(1-\rho^2) + \ln(1-G(\boldsymbol{\lambda}_1))]\urcorner + 1$ iterations, $G(\boldsymbol{\lambda}_t)$ would have a positive value.

5 Convergence of Algorithms

We will show convergence of Algorithms 1 and 2 to a maximum margin solution. Although there are many papers describing the convergence of specific classes of coordinate descent/ascent algorithms (e.g., [15]), this problem did not fit into any of the existing categories. The proofs below account for both the optimal and non-optimal cases, and for both algorithms.

One of the main results of this analysis is that both algorithms make significant progress at each iteration. In the next lemma, we only consider one increment, so we fix $\boldsymbol{\lambda}_t$ at iteration t and let $g_t := G(\boldsymbol{\lambda}_t)$, $s_t := \sum_j \boldsymbol{\lambda}_{t,j}$. Then, denote $g_{t+1}^{[1]} := G(\boldsymbol{\lambda}_t + \alpha_t^{[1]})$, $g_{t+1}^{[2]} := G(\boldsymbol{\lambda}_t + \alpha_t^{[2]})$, $s_{t+1}^{[1]} := s_t + \alpha_t^{[1]}$, and $s_{t+1}^{[2]} := s_t + \alpha_t^{[2]}$.

Lemma 3.

$$g_{t+1}^{[1]} - g_t \geq \frac{\alpha_t^{[1]}(r_t - g_t)}{2s_{t+1}^{[1]}}, \quad \textit{and} \quad g_{t+1}^{[2]} - g_t \geq \frac{\alpha_t^{[2]}(r_t - g_t)}{2s_{t+1}^{[2]}}.$$

Proof. We start with Algorithm 2. First, we note that since tanh is concave on $\mathbb{R}_+$, we can lower bound tanh on an interval $(a, b) \subset (0, \infty)$ by the line connecting the points $(a, \tanh(a))$ and $(b, \tanh(b))$. Thus,

$$\int_{\gamma_t - \alpha_t^{[2]}}^{\gamma_t} \tanh u \, du \geq \frac{1}{2}\alpha_t^{[2]} \left[\tanh \gamma_t + \tanh(\gamma_t - \alpha_t^{[2]})\right] = \frac{1}{2}\alpha_t^{[2]}(r_t + g_t), \quad (10)$$

where the last equality is from (8). Combining (10) with (4) yields:

$s_{t+1}^{[2]} g_{t+1}^{[2]} \geq s_t g_t + \dfrac{1}{2}\alpha_t^{[2]}(r_t + g_t)$, thus $s_{t+1}^{[2]}(g_{t+1}^{[2]} - g_t) + \alpha_t^{[2]} g_t \geq \dfrac{1}{2}\alpha_t^{[2]}(r_t + g_t)$,
and the statement of the lemma follows (for Algorithm 2). By definition, $g_{t+1}^{[1]}$ is the maximum value of $G(\boldsymbol{\lambda}_t + \alpha \mathbf{e}_{j_t})$, so $g_{t+1}^{[1]} \geq g_{t+1}^{[2]}$. Because $\alpha/(s + \alpha) = 1 - s/(\alpha + s)$ increases with α and since $\alpha_t^{[1]} \leq \alpha_t^{[2]}$,

$$g_{t+1}^{[1]} - g_t \geq g_{t+1}^{[2]} - g_t \geq \left(\frac{\alpha_t^{[2]}}{s_{t+1}^{[2]}}\right) \frac{(r_t - g_t)}{2} \geq \left(\frac{\alpha_t^{[1]}}{s_{t+1}^{[1]}}\right) \frac{(r_t - g_t)}{2}. \qquad \square$$

Another important ingredient for our convergence proofs is that the step size does not increase too quickly; this is the main content of the next lemma. We now remove superscripts since each step holds for both algorithms.

Lemma 4. $\lim_{t\to\infty} \alpha_t / s_{t+1} \to 0$ *for both Algorithms 1 and 2.*

If $\lim_{t\to\infty} s_t$ is finite, the statement can be proved directly. If $\lim_{t\to\infty} s_t = \infty$, our proof (which is omitted) uses (4), (5) and (8).

At this point, it is possible to use Lemma 3 and Lemma 4, to show asymptotic convergence of both Algorithms 1 and 2 to a maximum margin solution; we defer this calculation to the longer version. In what follows, we shall prove two different results about the convergence rate. The first theorem gives an explicit a priori upper bound on the number of iterations needed to guarantee that $g_t^{[1]}$ or $g_t^{[2]}$ is within $\epsilon > 0$ of the maximum margin ρ. As is often the case for uniformly valid upper bounds, the convergence rate provided by this theorem is not optimal, in the sense that faster decay of $\rho - g_t$ can be proved for large t if one does not insist on explicit constants. The second convergence rate theorem provides such a result, stating that $\rho - g_t = \mathcal{O}\left(t^{-1/(3+\delta)}\right)$, or equivalently $\rho - g_t \le \epsilon$ after $\mathcal{O}(\epsilon^{-(3+\delta)})$ iterations, where $\delta > 0$ can be arbitrarily small.

Both convergence rate theorems rely on estimates limiting the growth rate of α_t. Lemma 4 is one such estimate; because it is only an asymptotic estimate, our first convergence rate theorem requires the following uniformly valid lemma.

Lemma 5.

$$\alpha_t^{[1]} \le c_1 + c_2 s_t^{[1]} \text{ and } \alpha_t^{[2]} \le c_1 + c_2 s_t^{[2]}, \text{ where } c_1 = \frac{\ln 2}{1-\rho} \text{ and } c_2 = \frac{\rho}{1-\rho}. \quad (11)$$

Proof. Consider Algorithm 2. From (4),

$$s_{t+1}^{[2]} g_{t+1}^{[2]} - s_t^{[2]} g_t^{[2]} = \ln\cosh\gamma_t - \ln\cosh(\gamma_t - \alpha_t^{[2]}).$$

Because $\frac{1}{2}e^{\xi} \le \frac{1}{2}\left(e^{\xi} + e^{-\xi}\right) = \cosh\xi \le e^{\xi}$ for $\xi > 0$, we have $\xi - \ln 2 \le \ln\cosh\xi \le \xi$. Now,

$$s_{t+1}^{[2]} g_{t+1}^{[2]} - s_t^{[2]} g_t^{[2]} \ge \gamma_t - \ln 2 - (\gamma_t - \alpha_t^{[2]}), \text{ so}$$
$$\alpha_t^{[2]}(1-\rho) \le \alpha_t^{[2]}(1 - g_{t+1}^{[2]}) \le \ln 2 + s_t^{[2]}\left(g_{t+1}^{[2]} - g_t^{[2]}\right) \le \ln 2 + \rho s_t^{[2]}.$$

Thus we directly find the statement of the lemma for Algorithm 2. A slight extension of this argument proves the statement for Algorithm 1. ◻

Theorem 1. *(first convergence rate theorem) Suppose $R < 1$ is known to be an upper bound for ρ. Let $\tilde{1}$ be the iteration at which G becomes positive. Then both the margin $\mu(\boldsymbol{\lambda}_t)$ and the value of $G(\boldsymbol{\lambda}_t)$ will be within ϵ of the maximum margin ρ within at most*

$$\tilde{1} + 1 + \ulcorner (s_{\tilde{1}} + \ln 2)\, \epsilon^{-(3-R)/(1-R)} \urcorner \quad \textit{iterations, for both Algorithms 1 and 2.}$$

Proof. Define $\Delta G(\boldsymbol{\lambda}) := \rho - G(\boldsymbol{\lambda})$. Since (2) tells us that $0 \le \rho - \mu(\boldsymbol{\lambda}_t) \le \rho - G(\boldsymbol{\lambda}_t) = \Delta G(\boldsymbol{\lambda}_t)$, we need only to control how fast $\Delta G(\boldsymbol{\lambda}_t) \to 0$ as $t \to \infty$. That is, if $G(\boldsymbol{\lambda}_t)$ is within ϵ of the maximum margin ρ, so is the margin $\mu(\boldsymbol{\lambda}_t)$.

Starting from Lemma 3,

$$\rho - g_{t+1} \le \rho - g_t - \frac{\alpha_t}{2s_{t+1}}(r_t - \rho + \rho - g_t), \text{ thus}$$
$$\Delta G(\boldsymbol{\lambda}_{t+1}) \le \Delta G(\boldsymbol{\lambda}_t)\left[1 - \frac{\alpha_t}{2s_{t+1}}\right] - \frac{\alpha_t(r_t - \rho)}{2s_{t+1}} \le \Delta G(\boldsymbol{\lambda}_{\tilde{1}}) \prod_{\ell=\tilde{1}}^{t}\left[1 - \frac{\alpha_\ell}{2s_{\ell+1}}\right]. \quad (12)$$

We stop the recursion at $\boldsymbol{\lambda}_{\tilde{1}}$, where $\boldsymbol{\lambda}_{\tilde{1}}$ is the coefficient vector at the first iteration where G is positive. We upper bound the product in (12) using Lemma 5.

$$\prod_{\ell=\tilde{1}}^{t}\left[1-\frac{\alpha_\ell}{2s_{\ell+1}}\right]=\prod_{\ell=\tilde{1}}^{t}\left[1-\frac{1}{2}\frac{s_{\ell+1}-s_\ell}{s_{\ell+1}}\right]\le\exp\left[-\frac{1}{2}\sum_{\ell=\tilde{1}}^{t}\frac{s_{\ell+1}-s_\ell}{s_{\ell+1}}\right]$$
$$\le\exp\left[-\frac{1}{2}\sum_{\ell=\tilde{1}}^{t}\frac{s_{\ell+1}-s_\ell}{s_\ell+\frac{\rho}{1-\rho}s_\ell+\frac{\ln 2}{1-\rho}}\right]=\exp\left[-\frac{1-\rho}{2}\sum_{\ell=\tilde{1}}^{t}\frac{s_{\ell+1}-s_\ell}{s_\ell+\ln 2}\right]$$
$$\le\exp\left[-\frac{1-\rho}{2}\int_{s_{\tilde{1}}}^{s_{t+1}}\frac{dv}{v+\ln 2}\right]=\left[\frac{s_{\tilde{1}}+\ln 2}{s_{t+1}+\ln 2}\right]^{(1-\rho)/2}. \tag{13}$$

It follows from (12) and (13) that

$$s_t\le s_t+\ln 2\le(s_{\tilde{1}}+\ln 2)\left[\frac{\Delta G(\boldsymbol{\lambda}_{\tilde{1}})}{\Delta G(\boldsymbol{\lambda}_t)}\right]^{2/(1-\rho)}. \tag{14}$$

On the other hand, using some trickery one can show that for all t, for both algorithms, $\alpha_t\ge(\Delta G(\boldsymbol{\lambda}_{t+1}))/(1-\rho g_{\tilde{1}})$, which implies:

$$s_t\ge s_{\tilde{1}}+(t-\tilde{1})\frac{\Delta G(\boldsymbol{\lambda}_t)}{1-\rho g_{\tilde{1}}}. \tag{15}$$

Combining (14) with (15) leads to:

$$t-\tilde{1}\le\frac{(1-\rho g_{\tilde{1}})s_t}{\Delta G(\boldsymbol{\lambda}_t)}\le\frac{(1-\rho g_{\tilde{1}})(s_{\tilde{1}}+\ln 2)\left[\Delta G(\boldsymbol{\lambda}_{\tilde{1}})\right]^{2/(1-\rho)}}{\left[\Delta G(\boldsymbol{\lambda}_t)\right]^{1+[2/(1-\rho)]}}, \tag{16}$$

which means $\Delta G(\boldsymbol{\lambda}_t)\ge\epsilon$ is possible only if $t\le\tilde{1}+(s_{\tilde{1}}+\ln 2)\epsilon^{-(3-\rho)/(1-\rho)}$. Therefore, $\Delta(G(\boldsymbol{\lambda}_t)<\epsilon$ whenever t exceeds

$$\tilde{1}+1+(s_{\tilde{1}}+\ln 2)\epsilon^{-(3-R)/(1-R)}\ge\tilde{1}+1+(s_{\tilde{1}}+\ln 2)\epsilon^{-(3-\rho)/(1-\rho)}. \qquad\square$$

In order to apply the proof of Theorem 1, one has to have an upper bound for ρ, which we have denoted by R. This we may obtain in practice via the minimum achieved edge $R=\min_{\ell\le t}r_\ell<1$.

An important remark is that the technique of proof of Theorem 1 is much more widely applicable. In fact, this proof used only two main ingredients: Lemma 3 and Lemma 5. Inspection of the proof shows that the exact values of the constants occurring in these estimates are immaterial. Hence, Theorem 1 may be used to obtain convergence rates for other algorithms.

The convergence rate provided by Theorem 1 is not tight; our algorithms perform at a much faster rate in practice. The fact that the step-size bound in Lemma 5 holds for all t allowed us to find an upper bound on the number of iterations; however, we can find faster convergence rates in the asymptotic regime by using Lemma 4 instead. The following lemma holds for both Algorithms 1 and 2. The proof, which is omitted, follows from Lemma 3 and Lemma 4.

Lemma 6. *For any $0<\nu<1/2$, there exists a constant C_ν such that for all $t\ge\tilde{1}$ (i.e., all iterations where G is positive), $\rho-g_t\le C_\nu s_t^{-\nu}$.*

Theorem 2. *(second convergence rate theorem) For both Algorithms 1 and 2, and for any $\delta > 0$, a margin within ϵ of optimal is obtained after at most $\mathcal{O}(\epsilon^{-(3+\delta)})$ iterations from the iteration $\tilde{1}$ where G becomes positive.*

Proof. By (15), we have $t - \tilde{1} \leq (1 - \rho g_{\tilde{1}})(\rho - g_t)^{-1}(s_t - s_{\tilde{1}})$. Combining this with Lemma 6 leads to $t - \tilde{1} \leq (1 - \rho g_{\tilde{1}}) C_\nu^{1/\nu}(\rho - g_t)^{-(1+1/\nu)}$. For $\delta > 0$, we pick $\nu = \nu_\delta := 1/(2 + \delta) < 1/2$, and we can rewrite the last inequality as: $(\rho - g_t)^{3+\delta} \leq (1 - \rho g_{\tilde{1}}) C_{\nu_\delta}^{2+\delta}(t - \tilde{1})^{-1}$, or $\rho - g_t \leq C_\delta^{'}(t - \tilde{1})^{-1/(3+\delta)}$, with $C_\delta^{'} = (1 - \rho g_{\tilde{1}})^{1/(3+\delta)} C_{\nu_\delta}^{(2+\delta)/(3+\delta)}$. It follows that $\rho - \mu(\boldsymbol{\lambda}_t) \leq \rho - g_t < \epsilon$ whenever $t - \tilde{1} > (C_\delta^{'}\epsilon^{-1})^{(3+\delta)}$, which completes the proof of Theorem 2. □

Although Theorem 2 gives a better convergence rate than Theorem 1 since $3 < 1 + 2/(1 - \rho)$, there is an unknown constant $C_\delta^{'}$, so that this estimate cannot be translated into an a priori upper bound on the number of iterations after which $\rho - g_t < \epsilon$ is guaranteed, unlike Theorem 1.

6 Simulation Experiments

The updates of Algorithm 2 are less aggressive than AdaBoost's, but slightly more aggressive than the updates of arc-gv, and AdaBoost*. Algorithm 1 seems to perform very similarly to Algorithm 2 in practice, so we use Algorithm 2. This section is designed to illustrate our analysis as well as the differences between the various coordinate boosting algorithms; in order to do this, we give each algorithm the same random input, and examine convergence of all algorithms with respect to the margin. Experiments on real data are in our future plans.

Artificial test data for Figure 2 was designed as follows: 50 examples were constructed randomly such that each $\mathbf{x}_i$ lies on a corner of the hypercube $\{-1, 1\}^{100}$. We set $y_i = \text{sign}(\sum_{k=1}^{11} \mathbf{x}_i(k))$, where $\mathbf{x}_i(k)$ indicates the k^{th} component of $\mathbf{x}_i$. The j^{th} weak learner is $h_j(\mathbf{x}) = \mathbf{x}(j)$, thus $M_{ij} = y_i \mathbf{x}_i(j)$. To implement the "non-optimal" case, we chose a random classifier from the set of sufficiently good classifiers at each iteration.

We use the definitions of arc-gv and AdaBoost* found in Meir and Rätsch's survey [8]. AdaBoost, arc-gv, Algorithm 1 and Algorithm 2 have initially large updates, based on a conservative estimate of the margin. AdaBoost*'s updates are initially small based on an overestimate of the margin.

AdaBoost's updates remain consistently large, causing $\boldsymbol{\lambda}_t$ to grow quickly and causing fast convergence with respect to G. AdaBoost seems to converge to the maximum margin in (a); however, it does not seem to in (b), (d) or (e). Algorithm 2 converges fairly quickly and dependably; arc-gv and AdaBoost* are slower here. We could provide a larger value of ν in AdaBoost* to encourage faster convergence, but we would sacrifice a guarantee on accuracy. The more "optimal" we choose the weak learners, the better the larger step-size algorithms (AdaBoost and Algorithm 2) perform, relative to AdaBoost*; this is because AdaBoost*'s update uses the minimum achieved edge, which translates into smaller steps while the weak learning algorithm is doing well.

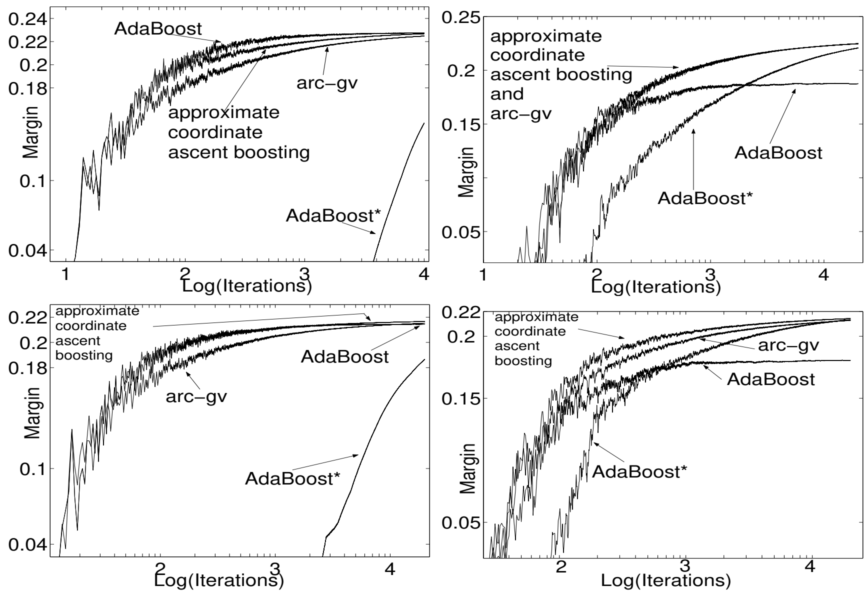

Fig. 2. AdaBoost, AdaBoost* (parameter ν set to .001), arc-gv, and Algorithm 2 on synthetic data. (a-Top Left) Optimal case. (b-Top Right) Non-optimal case, using the same 50×100 matrix $\mathbf{M}$ as in (a). (c-Bottom Left) Optimal case, using a different matrix. (d-Bottom Right) Non-optimal case, using the same matrix as (c).

7 A New Way to Measure AdaBoost's Progress

AdaBoost is still a mysterious algorithm. Even in the optimal case it may converge to a solution with margin significantly below the maximum [11]. Thus, the margin theory only provides a significant piece of the puzzle of AdaBoost's strong generalization properties; it is not the whole story [5, 2, 11]. Hence, we give some connections between our new algorithms and AdaBoost, to help us understand how AdaBoost makes progress. In this section, we measure the progress of AdaBoost according to something other than the margin, namely, our smooth margin function G. First, we show that whenever AdaBoost takes a large step, it makes progress according to G. We use the superscript $^{[A]}$ for AdaBoost.

Theorem 3. $G(\boldsymbol{\lambda}_{t+1}^{[A]}) \geq G(\boldsymbol{\lambda}_t^{[A]}) \iff \Upsilon(r_t) \geq G(\boldsymbol{\lambda}_t^{[A]})$, *where* $\Upsilon : (0,1) \to (0,\infty)$ *is a monotonically increasing function.*

In other words, $G(\boldsymbol{\lambda}_{t+1}^{[A]}) \geq G(\boldsymbol{\lambda}_t^{[A]})$ if and only if the edge r_t is sufficiently large.

Proof. Using AdaBoost's update $\alpha_t^{[A]} = \gamma_t$, $G(\boldsymbol{\lambda}_t^{[A]}) \leq G(\boldsymbol{\lambda}_{t+1}^{[A]})$ if and only if:

$$(s_t^{[A]} + \alpha_t^{[A]}) G(\boldsymbol{\lambda}_t^{[A]}) \leq (s_t^{[A]} + \alpha_t^{[A]}) G(\boldsymbol{\lambda}_{t+1}^{[A]}) = s_t^{[A]} G(\boldsymbol{\lambda}_t^{[A]}) + \int_0^{\alpha_t^{[A]}} \tanh u \, du,$$

$$\text{i.e., } G(\boldsymbol{\lambda}_t^{[A]}) \leq \frac{1}{\alpha_t^{[A]}} \int_0^{\alpha_t^{[A]}} \tanh u \, du,$$

where we have used (4). We denote the expression on the right hand side by $\Upsilon(r_t)$, which can be rewritten as: $\Upsilon(r_t) := -\ln\left(1-r_t^2\right) \Big/ \ln\left(\frac{1+r_t}{1-r_t}\right)$. Since $\Upsilon(r)$ is monotonically increasing in r, our statement is proved. $\square$

Hence, AdaBoost makes progress (measured by G) if and only if it takes a big enough step. Figure 3, which shows the evolution of the edge values, illustrates this. Whenever G increased from the current iteration to the following iteration, a small dot was plotted. Whenever G decreased, a large dot was plotted. The fact that the larger dots are below the smaller dots is a direct result of Theorem 3. In fact, one can visually track the progress of G using the boundary between the larger and smaller dots.

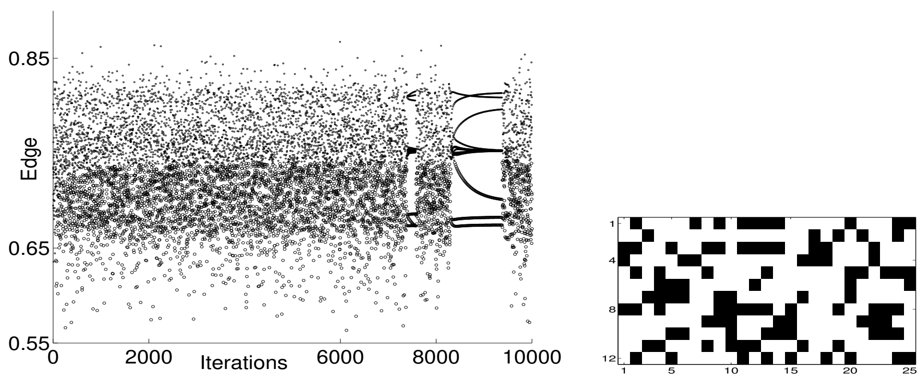

Fig. 3. Value of the edge at each iteration t, for a run of AdaBoost using the 12×25 matrix $\mathbf{M}$ shown (black is -1, white is +1). AdaBoost alternates between chaotic and cyclic behavior. For further explanation of the interesting dynamics in this plot, see [11].

AdaBoost's weight vectors often converge to a periodic cycle when there are few support vectors [11]. Where Algorithms 1 and 2 make progress with respect to G at every iteration, the opposite is true for cyclic AdaBoost, namely that AdaBoost cannot increase G at every iteration, by the following:

Theorem 4. *If AdaBoost's weight vectors converge to a cycle of length T iterations, the cycle must obey one of the following conditions:*

1. *the value of G decreases for at least one iteration within the cycle, or*
2. *the value of G is constant at every iteration, and the edge values in the cycle $r_{t,1}^{(cyc)}, ..., r_{t,T}^{(cyc)}$ are equal.*

In other words, the value of G cannot be strictly increasing within a cycle. The main ingredients for the proof (which is omitted) are Theorem 3 and (4). For specific cases that have been studied [11], the value of G is non-decreasing, and the value of r_t is the same at every iteration of the cycle. In such cases, a stronger equivalence between support vectors exists here; they are all "viewed" similarly by the weak learning algorithm, in that they are misclassified the same proportion of the time. (This is surprising since weak classifiers may appear more than once per cycle.)

Theorem 5. *Assume AdaBoost cycles. If all edges are the same, then all support vectors are misclassified by the same number of weak classifiers per cycle.*

Proof. Let $r_t =: r$ which is constant. Consider support vectors i and i'. All support vectors obey the cycle condition [11], namely: $\prod_{t=1}^{T}(1+M_{ij_t}r)=\prod_{t=1}^{T}(1+M_{i'j_t}r)=1$. Define $\tau_i := |\{t : M_{ij_t} = 1\}|$, the number of times example i is correctly classified during one cycle of length T. Now, $1 = \prod_{t=1}^{T}(1+M_{ij_t}r) = (1+r)^{\tau_i}(1-r)^{T-\tau_i} = (1+r)^{\tau_{i'}}(1-r)^{T-\tau_{i'}}$. Hence, $\tau_i = \tau_{i'}$. Thus, example i is misclassified the same number of times that i' is misclassified. Since the choice of i and i' were arbitrary, this holds for all support vectors. □

References

[1] Leo Breiman. Arcing the edge. Technical Report 486, Statistics Department, University of California at Berkeley, 1997.

[2] Leo Breiman. Prediction games and arcing algorithms. *Neural Computation*, 11(7):1493–1517, 1999.

[3] Michael Collins, Robert E. Schapire, and Yoram Singer. Logistic regression, AdaBoost and Bregman distances. *Machine Learning*, 48(1/2/3), 2002.

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

[5] Adam J. Grove and Dale Schuurmans. Boosting in the limit: Maximizing the margin of learned ensembles. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, 1998.

[6] V. Koltchinskii and D. Panchenko. Empirical margin distributions and bounding the generalization error of combined classifiers. *The Annals of Statistics*, 30(1), February 2002.

[7] Llew Mason, Peter Bartlett, and Jonathan Baxter. Direct optimization of margins improves generalization in combined classifiers. In *Advances in Neural Information Processing Systems 12*, 2000.

[8] R. Meir and G. Rätsch. An introduction to boosting and leveraging. In S. Mendelson and A. Smola, editors, *Advanced Lectures on Machine Learning*, pages 119–184. Springer, 2003.

[9] Gunnar Rätsch and Manfred Warmuth. Efficient margin maximizing with boosting. Submitted, 2002.

[10] Saharon Rosset, Ji Zhu, and Trevor Hastie. Boosting as a regularized path to a maximum margin classifier. Technical report, Department of Statistics, Stanford University, 2003.

[11] Cynthia Rudin, Ingrid Daubechies, and Robert E. Schapire. The dynamics of AdaBoost: Cyclic behavior and convergence of margins. Submitted, 2004.

[12] Cynthia Rudin, Ingrid Daubechies, and Robert E. Schapire. On the dynamics of boosting. In *Advances in Neural Information Processing Systems 16*, 2004.

[13] Robert E. Schapire. The boosting approach to machine learning: An overview. In *MSRI Workshop on Nonlinear Estimation and Classification*, 2002.

[14] Robert E. Schapire, Yoav Freund, Peter Bartlett, and Wee Sun Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *The Annals of Statistics*, 26(5):1651–1686, October 1998.

[15] Tong Zhang and Bin Yu. Boosting with early stopping: convergence and consistency. Technical Report 635, Department of Statistics, UC Berkeley, 2003.