ARCING THE EDGE

Leo Breiman
Technical Report 486 , Statistics Department
University of California, Berkeley CA. 94720

Abstract

Recent work has shown that adaptively reweighting the training set, growing a classifier using the new weights, and combining the classifiers constructed to date can significantly decrease generalization error. Procedures of this type were called arcing by Breiman[1996]. The first successful arcing procedure was introduced by Freund and Schapire[1995,1996] and called Adaboost. In an effort to explain why Adaboost works, Schapire et.al. [1997] derived a bound on the generalization error of a convex combination of classifiers in terms of the margin. We introduce a function called the edge, which differs from the margin only if there are more than two classes. A framework for understanding arcing algorithms is defined. In this framework, we see that the arcing algorithms currently in the literature are optimization algorithms which minimize some function of the edge. A relation is derived between the optimal reduction in the maximum value of the edge and the PAC concept of weak learner. Two algorithms are described which achieve the optimal reduction. Tests on both synthetic and real data cast doubt on the Schapire et.al. explanation.

1, Introduction

1.1 Background

There is recent empirical evidence that significant reductions in generalization error can be gotten by growing a number of different classifiers on the same training set and letting these vote for the best class. Freund and Schapire ([1995], [1996]) proposed an algorithm called AdaBoost which adaptively reweights the training set in a way based on the past history of misclassifications, constructs a new classifier using the current weights, and uses the misclassification rate of this classifier to determine the size of its vote.

In a number of empirical studies on many data sets using trees (CART or C4.5) as the base classifier (Drucker and Cortes[1995], Quinlan[1996], Freud and Schapire[1996], Breiman[1996]) AdaBoost produced dramatic decreases in generalization error compared to using a single tree. Error rates were reduced to the point where tests on some well-known data sets gave the result that CART plus AdaBoost did significantly better than any other of the commonly used classification methods (Breiman[1996]).

Meanwhile, empirical results showed that other methods of adaptive resampling (or reweighting) and combining (called "arcing" by Breiman [1996]) also led to low test set error rates. An algorithm called arc-x4 (Breiman[1996]) gave error rates almost identical to Adaboost. Ji and Ma[1997] worked with classifiers consisting of randomly selected hyperplanes and using a different method of adaptive resampling and unweighted voting, also got low error rates. Thus, there are a least three arcing algorithms extant, all of which give excellent classification accuracy.

1. 2 Examples-Arcing Algorithms.

A non-negative combination of classifiers with classifier $h_m(\mathbf{x})$ having vote c_m classifies instances this way: let

$$s(j,\mathbf{x}) = \sum_m c_m I(h_m(\mathbf{x})=j) \tag{1.1}$$

where I denotes the indicator function. Then $\mathbf{x}$ is classified into the class with label = $\text{argmax}_j s(j,\mathbf{x})$. Arcing refers to an algorithm that uses adaptive reweighting of the training set to select the classifiers h_m and determine the votes c_m. Here are three examples:

In the first two examples, suppose we have a fixed classification method, i.e. CART (but many others will do) and a training set $\{(y_n,\mathbf{x}_n)\ n=1, \dots ,N\}$ where y is a class label taking values in $\{1, \dots ,J\}$. Suppose that at the mth iteration, probabilities $p^{(m)}(\mathbf{x}_n)$ are defined for the instances $\mathbf{x}_n$. Construct a classifier $h_m(\mathbf{x})$ using the weighted training set. Depending on the characteristics of $h_1, h_2, \dots ,h_m$ update the probabilities and assign vote c_m to h_m. Start with $p^{(1)}(\mathbf{x}_n) =1/N$.

Example 1. Adaboost (Freund and Schapire ([1995], [1996]).

Let $d(n)=1$ if $h_m(\mathbf{x}_n)\neq y_n$, else $d(n)=0$. Compute

$$\varepsilon_m=\sum_n p^{(m)}(\mathbf{x}_n)d(n).$$

Set $\beta_m=(1-\varepsilon_m)/\varepsilon_m$, and

$$p^{(m+1)}(\mathbf{x}_n)=p^{(m)}(\mathbf{x}_n)\,\beta_m^{d(n)}/S$$

where the /S indicates normalization to sum one. The vote c_m assigned to h_m is $\log(\beta_m)$.

Example 2. Arc-x4 (Breiman [1997])

Define k(n) as the number of misclassifications of $\mathbf{x}_n$ by h1, h2, ... ,hm. Let

$$p^{(m+1)}(\mathbf{x}_n)=(1+k(n)^4)/S$$

Assign vote $c_m=1$ to h_m.

Example 3. Random Hyperplanes (Ji and Ma[1997])

This method applies only to two class problems. The set of classifiers H is defined this way: To each direction in input space and point $\mathbf{x}_n$ in the training set corresponds two classifiers. Form the hyperplane passing through $\mathbf{x}_n$ perpendicular to the given direction. The first classifier classifies all the points on one side as class 1 and on the other as class 2. The second classifier switches the class assignment.

Set two parameters $\alpha>0, \eta>0$ such that $.5-\eta<\alpha<.5$. After h_m is selected, let

$$p^{(m+1)}(\mathbf{x}_n)=I(k(n)>\alpha m)/S$$

where $I(\cdot)$ is the indicator function. Select a hyperplane direction and training set instance at random. Compute the classification error for each of the two associated classifiers using the probabilities $p^{(m+1)}$. If the smaller of the two errors is less than $.5-\eta$ then keep the corresponding classifier and give it vote one. Otherwise, reject both and select another random hyperplane

1.3 The Schapire et. al. Theorem and Explanation

As the arcing continues, more classifiers are combined. The VC-dimension of the combination increases as more classifiers are added, but what is observed empirically is that the test set error continually decreases. Schapire et. al[1997] worked with a function of convex combinations of classifiers which they called the *margin*. and defined as follows: if $y(\mathbf{x})=j$ then $\text{margin}(\mathbf{c},\mathbf{x})= s(j,\mathbf{x})-\max(s(i,\mathbf{x}); i\neq j)$. Thus, for any number of classes, the probability of erroneous classification is $P(\text{margin}(\mathbf{c})<0)$.

The Schapire et.al. paper proves the important result that for the two class case with y a deterministic function of $\mathbf{x}$, if the $\{\mathbf{x}_n\}$ are independently drawn from a distribution P* on a larger space and P is equidistributed on the $\{\mathbf{x}_n\}$, then with probability greater then $1-\delta$, for all $\mathbf{c}$ and $\theta>0$, the generalization error $P^*(\text{margin}(\mathbf{c})<0)$ can be bounded by $P(\text{margin}(\mathbf{c})<\theta)$ plus a function depending only on θ, δ, d, N. Here d is the VC-dimension of the set of classifiers H from which the individual classifiers in the combination are selected.

This result shifted focus from trying to minimize $P(\text{margin}(\mathbf{c})<0)$ to minimizing $P(\text{margin}(\mathbf{c})<\theta)$ for positive values of θ. In particular, the Schapire et.al. paper then explains the success of Adaboost in lowering test set error as compared to bagging (Breiman, [1996b]) and to error-correcting output coding (Kong and Dietterich [1995]) by Adaboost's ability to produce generally higher margins.

1.4 Our Study

One question is: what do the three arcing algorithms described in Section 1.2 have in common? In this paper we give a simple framework that begins by defining a function on the training set called the edge which is similar to the margin. We then show that there is a large class of algorithms, including all of the arcing examples in Section 1.2, that work by minimizing a function of the edge. This framework leads to the construction of algorithms leading to optimal reductions in the size of the maximum value of the edge. But then, applying one of these algorithms to a variety of data sets gives results which are the opposite of what we would expect given the Schapire et. al. explanation of why arcing works.

2. Framework and Edge

Assume that there is a space of N input vectors $\mathbf{X}=\{\mathbf{x}_n\}$ such that to each $\mathbf{x}_n$, is associated a class label y_n which has a value in $\{1, ... ,J\}$. We are given an ensemble H of M classifiers $\{h_m(\mathbf{x})\}$ such that each $h_m(\mathbf{x})$ also takes values in $\{1, ... ,J\}$. In addition, there is a probability distribution $P(\mathbf{x}_n)$ on $\mathbf{X}$ and we assume $P(\mathbf{x}_n)>0$ for all n. Call the set $e_m=\{\mathbf{x}_n; h_m(\mathbf{x}_n) \neq y_n\}$ the error set of h_m, and let $i(\mathbf{x},e_m)=I(\mathbf{x} \in e_m)$ be the indicator function of e_m.

Definition 1.1. *Let $\mathbf{c}=\{c_m\}$ be a collection of non-negative numbers (votes) summing to one. The edge($\mathbf{c}$) is the function of $\mathbf{x}$ given by the convex combination*

$$\text{edge}(\mathbf{c}) = \sum_m c_m i(\mathbf{x},e_m)$$

The combined classifier with votes $\{c_m\}$ predicts $\text{argmax}_j\ s(j,\mathbf{x})$ (see 1.1). For two classes, margin($\mathbf{c}$)=1-2edge($\mathbf{c}$) and, ignoring ties, the probability of an erroneous classification on $\mathbf{X}$ is P(edge($\mathbf{c}$)>1/2). In general margin($\mathbf{c}$)$\geq$1-2edge($\mathbf{c}$), so P(margin($\mathbf{c}$)<θ)$\leq$P(edge($\mathbf{c}$)>(1-θ)/2). The implication of the Schapire et.al. work is that values of $\mathbf{c}$ that give generally low values of edge($\mathbf{c}$) on the training set will also lead to low generalization error. The intuitive idea is clear: if, in the training set, it is rare that over 1/4th, say, of the votes in a combined classifier are wrong, then the probability that in the parent space, over 1/2 of the weighted votes are wrong, is small. In this paper, we study algorithms producing low edge values.

The objective, given the set H of classifiers and the class labels $\{y_n\}$ is to find $\mathbf{c}$ values minimizing P(edge($\mathbf{c}$)>ϕ) for small values of ϕ. Define the bottom edge ϕ^* as the minimum value of ϕ such that $\inf_{\mathbf{c}}$P(edge($\mathbf{c}$)>ϕ)=0. In Section 3 ϕ^* is characterized, using the dual of a linear program, in a way that gives insight into the PAC concept of weak learning.

There are some fairly obvious algorithms available to minimize P(edge($\mathbf{c}$)>ϕ) or, more generally, to minimize Ef(edge($\mathbf{c}$)) for f(x) an increasing function, For instance, a linear programming method can be used to minimize P(edge($\mathbf{c}$)>ϕ). But there are side conditions on this objective. The algorithms permitted are only those which can be implemented in more realistic situations. This class, which we call arcing algorithms, are defined in Section 4.

The sections 5 and 6 define two classes of arcing algorithms. One class is called unnormalized and the other normalized . A convergence proof is given for each. We note that Adaboost is a unnormalized arc algorithm, while arc-x4 and random hyperplanes are normalized arc algorithms.

In general, the algorithms that converge to a $\mathbf{c}$ such that P(edge($\mathbf{c}$)>ϕ)=0 use a prespecified value of ϕ. In section 7 we show that there are universal arcing algorithms, i.e. algorithms that converges to a value of $\mathbf{c}$ such that P(edge($\mathbf{c}$)>ϕ)=0 for all $\phi>\phi^*$. Section 8 presents some surprising empirical results and Section 9 gives concluding remarks.

3. <u>The Bottom Edge</u>

<u>Definition 2.1</u> *The bottom edge ϕ^* is the infinum of those values of ϕ such that $\inf_{\mathbf{c}} P(edge(\mathbf{c})>\phi)=0$.*

<u>Theorem 2.2</u> $\phi^*=\max_Q \min_m Q(e_m)$ *where the maximum is over all probabilities Q on $\mathbf{X}$.*

proof: Let $\varphi=\max_Q \min_m Q(e_m)$ and consider minimizing the function g($\mathbf{c}$)=E(edge($\mathbf{c}$)-ϕ)$^+$ where x^+ is the positive part of x, and E is expectation w.r.t. to P on $\mathbf{X}$. If $\phi>\phi^*$ then the minimum is zero. Denoting p_n=P($\mathbf{x}_n$), the minimization can also be casted as the linear programming problem: minimize $\sum_n u_n p_n$ under the constraints

$$u_n \geq 0,\ u_n \geq \sum_m c_m i(\mathbf{x}_n, e_m) - \phi,\ c_m \geq 0,\ \sum_m c_m = 1$$

The dual problem is: maximize $-\phi\sum_n \lambda_n + z$ under the constraints

$$\lambda_n \geq 0,\ \lambda_n \leq p_n,\ z \leq \sum_n \lambda_n i(\mathbf{x}_n, e_m)$$

This is equivalent to: maximize $-\phi\sum_n \lambda_n + \min_m \sum_n \lambda_n i(\mathbf{x}_n, e_m)$ under the constraints $0 \le \lambda_n \le p_n$. Suppose there is a set of $\{\lambda_n\}$ that make this expression positive. Setting $Q(\mathbf{x}_n) = \lambda_n / \sum_n \lambda_n$, then $\min_m Q(e_m) > \phi$ implying $\varphi \ge \phi^*$. On the other hand, take Q so that $\varphi = \min_m Q(e_m)$, and let λ_n = $rQ(\mathbf{x}_n)$ where r >0 is taken so that $\lambda_n \le p_n$, all n. Then the maximum is bounded below by $r(\varphi - \phi)$, so $\varphi \le \phi^*$.

In PAC terminology, H is a weak learner if $\phi^* < 1/2$. Put another way, the bottom edge is <1/2 only if H is a weak learner.

4. Permissible Algorithms are Arcing

Many algorithms can be constructed to minimize the function Ef(edge(**c**)) under the non-negativity and sum one constraints on **c**. We will restrict attention to the class of arcing algorithms--e.g. those algorithms that use adaptive reweighting to determine the selection of the current classifier and its vote.

We assume an iterative algorithm such that at each step:

i) The algorithm defines a probability $Q_k(\mathbf{x}_n)$ depending on the training set misclassifications and votes of the first k-1 classifiers selected.

ii) The kth classifier selected is h_{m^*} where

$$m^* = \text{argmin}_m \; Q_k(e_m)$$

iii) The vote c_{m^*} is determined from the training set misclassifications of h_{m^*} and the training set misclassifications and votes of the first k-1 classifiers selected.

The restriction ii) is the most useful and interesting. It says that given Q, we select that h_m in H having minimum Q-error probability. Given an infinite class H, it is usually not feasible in practice to select the minimum Q-error classifier in H. But various construction and fitting methods are used to try and approximate the minimum.

5. Unnormalized Arc Algorithms

5.1 Description

Let f(x) $\to\infty$ as $x\to\infty$, to 0 as $x \to -\infty$, and have everywhere positive first and second derivatives. For non-negative weights $\{b_m\}$ denote $(\mathbf{b}, \mathbf{i}(\mathbf{x}_n)) = \sum_m b_m i(\mathbf{x}_n, e_m)$ and $|\mathbf{b}| = \sum b_m$. Assuming that $\phi > \phi^*$, we want to minimize $g(\mathbf{b}) = Ef((\mathbf{b},\mathbf{i}) - \phi\,|\mathbf{b}|)$ starting from $\mathbf{b}=0$.

Unnormalized Algorithm

At the current value of $\boldsymbol{b}$ *let*

$$Q(\mathbf{x}_n) = f'((\mathbf{b}, \mathbf{i}(\mathbf{x}_n)) - \phi|\mathbf{b}|) / \sum_n f'((\mathbf{b}, \mathbf{i}(\mathbf{x}_n)) - \phi|\mathbf{b}|)$$

and $m^* = \text{argmin}_m Q(e_m)$ *Add* $\Delta > 0$ *to* b_{m^*} *and do a line search to minimize* $g(\boldsymbol{b} + \Delta\,\mathbf{u}_{m^*})$ *over* $\Delta > 0$ *where* $\mathbf{u}_{m^*}$ *is a unit vector in the direction of* b_{m^*}. *If the minimizing value of* Δ *is* Δ^* *then update* $\boldsymbol{b} \to \boldsymbol{b} + \Delta^*\,\mathbf{u}_{m^*}$. *Repeat until convergence.*

Comments. Note that

$$\partial g(\mathbf{b})/\partial b_m = E(\mathbf{i}(e_m)-\phi)\,f'((\mathbf{b},\mathbf{i})-\phi|\mathbf{b}|)$$

so the minimum value of the first partial of the target function g($\mathbf{b}$) is in the direction of b_{m^*}. This value is negative, because it is less than or equal to $(\phi^*-\phi)Ef'((\mathbf{b},\mathbf{i})-\phi|\mathbf{b}|)$. Furthermore, the 2nd derivative of g($\mathbf{b}+\Delta\,\mathbf{u}_{m^*}$) with respect to Δ is positive, insuring a unique minimum in the line search over $\Delta>0$.

5.2 <u>Convergence</u>

<u>Theorem 5.1.</u> *Let* $\mathbf{b}^{(k)}$ *be the successive values generated by the unnormalized arc algorithm, and set* $\mathbf{c}^{(k)}=\mathbf{b}^{(k)}/|\mathbf{b}|$. *Then for* $\mathbf{c}$ *any limit point of the* $\mathbf{c}^{(k)}$*sequence,* $P(edge(\mathbf{c})>\phi)=0$.

proof. It suffices to show that $|\mathbf{b}^{(k)}|\to\infty$, since writing

$$(\mathbf{b}^{(k)},\mathbf{i})-\phi|\mathbf{b}^{(k)}|=|\mathbf{b}^{(k)}|(edge(\mathbf{c}^{(k)})-\phi)$$

shows that unless $P(edge(\mathbf{c}^{(k)})>\phi)$ goes to zero, $g(\mathbf{b}^{(k)})\to\infty$. If $|\mathbf{b}^{(k)}|$ does not go to infinity, then there is a finite limit point $\mathbf{b}^*$. But every time that $\mathbf{b}^{(k)}$ is in the vicinity of $\mathbf{b}^*$, g($\mathbf{b}^{(k)}$) decreases in the next step by at least a fixed amount $\delta>0$. Since g($\mathbf{b}^{(k)}$) is monotonic decreasing, and non-negative, this is not possible.

comments: i) From this argument, its clear that cruder algorithms would also give convergence, since all that is is needed is to generate a sequence $|\mathbf{b}^{(k)}|\to\infty$ such that $g(\mathbf{b}^{(k)})$ stays bounded. In particular, the line search can be avoided (see Section 7). ii) if we take (w.l.o.g) g(0)=1, then at the kth stage $P(edge(\mathbf{c}^{(k)})>\phi)\le g(\mathbf{b}^{(k)})$. ii) The simplest function f(x) satisfying the conditions for the unnormalized algorithm is e^x. We denote unnormalized algorithms based on this function by arc-ex.

5.3. <u>Adaboost is an arc-ex algorithm with ϕ=1/2</u>

For $f(x)=e^x$,

$$f((\mathbf{b},\mathbf{i})-\phi|\mathbf{b}|)=e^{-\phi|\mathbf{b}|}\prod_m e^{b_m i(e_m)}$$

Denote $\pi(n)=\prod_m e^{b_m i(e_m)}$, set $Q(\mathbf{x}_n)=\pi(n)/\sum\pi(n)$, and m*=argmin$_m$Q($e_m$). Set $\varepsilon_m=Q(e_{m^*})$. We find the line search step by setting

$$E(\mathbf{i}(e_m)-\phi)\,f'((\mathbf{b}+\Delta\mathbf{u}_{m^*},\mathbf{i})-\phi|\mathbf{b}|-\phi\Delta)=0.$$

Solving gives $\Delta^*=\log(\phi/(1-\phi))+\log((1-\varepsilon_m)/\varepsilon_m)$. The update for Q is given by $\pi(n)\to\pi(n)e^{\Delta^* i(e_{m^*})}$. For $\phi=1/2$ this is the Adaboost algorithm described in Section 1.

6. Normalized Arc Algorithms

6.1 Description

The normalized arc algorithms work directly with $(\mathbf{b},\mathbf{i})/|\mathbf{b}|$ so the coefficients are normalized to sum one at all steps. The algorithm minimizes $Ef((\mathbf{b},\mathbf{i})/|\mathbf{b}|)$ where f'(x) is non-negative and f''(x) is continuous and non-negative for all $x \in [0,1]$. Let $\mathbf{c}=\mathbf{b}/|\mathbf{b}|$.

Normalized Algorithm

At the current value of $\mathbf{b}$ *, if* $Ef'((\mathbf{c},\mathbf{i}))=0$ *then stop. Otherwise, let*

$$Q(\mathbf{x}_n)=f'((\mathbf{b},\mathbf{i}(\mathbf{x}_n)/|\mathbf{b}|)/\sum_n f'((\mathbf{b},\mathbf{i}(\mathbf{x}_n)/|\mathbf{b}|)$$

and $m^*=\mathrm{argmin}_m\ Q(e_m)$. *If* $Q(e_{m^*})\geq E_Q(\mathbf{c},\mathbf{i})$, *then stop. Otherwise let* $b_{m^*}=b_{m^*}+1$ *and repeat.*

Comments: Note that

$$\partial Ef((\mathbf{b},\mathbf{i})/|\mathbf{b}|)/\partial b_m=\frac{1}{|\mathbf{b}|}E(i(e_m)-(\mathbf{c},\mathbf{i}(\mathbf{x}_n)))f'((\mathbf{b},\mathbf{i})/|\mathbf{b}|).$$

The smallest partial derivative is at m=m*.

6.2 Convergence

Theorem 6.1 *Let* $\mathbf{c}$ *be any stopping or limit point of the normalized arc algorithm. Then* $\mathbf{c}$ *is a global minimum of Ef(edge(*$\mathbf{c}$ *)).*

proof: Let f'(x)>0 for $x>\phi$ and zero for $x\leq\phi$. If $\phi<\phi^*$ or if f'(x)>0 for all x then $Ef'((\mathbf{c},\mathbf{i}))=0$ is not possible. We treat this case first. Suppose the algorithm stops after a finite number of steps because $Q(e_{m^*})\geq E_Q(\mathbf{c},\mathbf{i})$. Then

$$E(i(e_{m^*})f'((\mathbf{b},\mathbf{i})/|\mathbf{b}|)\geq\sum_m c_m E(i(e_m)f'((\mathbf{b},\mathbf{i})/|\mathbf{b}|)|) \tag{6.1}$$

This implies that for all m, either $c_m=0$ or

$$Ei(e_{m^*})f'((\mathbf{b},\mathbf{i})/|\mathbf{b}|)=Ei(e_m)f'((\mathbf{b},\mathbf{i})/|\mathbf{b}|) \tag{6.2}$$

Consider the problem of minimizing $Ef((\mathbf{c},\mathbf{i}))$ under non-negativity and sum one constraints on $\mathbf{c}$. The Kuhn-Tucker necessary conditions are that there exist numbers λ and $\mu_m\geq0$ such that if $c_m>0$, then $\partial Ef((\mathbf{c},\mathbf{i}))/\partial c_m=\lambda$. If $c_m=0$, then $\partial Ef((\mathbf{c},\mathbf{i}))/\partial c_m=\lambda+\mu_m$. These conditions follow from (6.1) and (6.2).

Now suppose that the algorithm does not stop after a finite number of steps. After the kth step, let $\mathbf{c}^{(k+1)}$ be the updated $\mathbf{c}^{(k)}$. Then

$$(\mathbf{c}^{(k+1)},\mathbf{i})-(\mathbf{c}^{(k)},\mathbf{i})=(i(e_{m^*(k)})-(\mathbf{c}^{(k)},\mathbf{i}))/(k+1) \tag{6.3}$$

Denote the right hand side of (6.3) by $\delta_k(\mathbf{x}_n)/(k+1)$. Using a partial Taylor expansion gives

$$Ef((\mathbf{c}^{(k+1)},\mathbf{i}))-Ef((\mathbf{c}^{(k)},\mathbf{i}))=\frac{1}{(k+1)}E\delta_k f'((\mathbf{c}^{(k)},\mathbf{i}))+\frac{\gamma}{(k+1)^2} \tag{6.4}$$

The first term on the right in (6.4) is negative for all k. Since $Ef((\mathbf{c},\mathbf{i}))$ is bounded for all $\mathbf{c}$,

$$\sum_k \frac{1}{(k+1)}E\delta_k f'((\mathbf{c}^{(k)},\mathbf{i}))<\infty \tag{6.5}$$

So, except possibly on a non-dense subsequence of the {k},

$$E\delta_k f'((\mathbf{c}^{(k)},\mathbf{i}))\to 0 \tag{6.6}$$

Take a subsequence of the k for whic h (6.6) holds such that $m^*(k)\to m^*,\mathbf{c}^{(k)}\to\mathbf{c}$. Then the situation of (6.2) is in force. Since f''(x)≥0 on [0,1], then $Ef((\mathbf{c},\mathbf{i}))$ is convex in **c**, and a point **c** satisfying the Kuhn-Tucker conditions is a global minimum. Furthermore, since the first term on the right of (6.4) is negative (non-stopping), then (6.4) implies that the entire sequence $Ef((\mathbf{c}^{(k)},\mathbf{i}))$ converges. Thus, all limits or stopping points of the $\mathbf{c}^{(k)}$ are global minimum points of $Ef((\mathbf{c},\mathbf{i}))$.

Now examine the case $\phi\geq\phi^*$. Clearly f(0) ≤ $Ef((\mathbf{c},\mathbf{i}))$ for any **c**. If there is stopping at any point because $Ef'((\mathbf{c},\mathbf{i}))=0$ then $Ef((\mathbf{c},\mathbf{i}))$=f(0). Otherwise, note that for any **c**, $E(\mathbf{c},\mathbf{i})f'((\mathbf{c},\mathbf{i}))\geq\phi E(\mathbf{c},\mathbf{i})f'((\mathbf{c},\mathbf{i}))$. Hence

$$E(i(e_{m^*})-(\mathbf{c},\mathbf{i}))f'((\mathbf{c},\mathbf{i})\leq(\phi^*-\phi)Ef'((\mathbf{c},\mathbf{i})) \tag{6.7}$$

If $\phi>\phi^*$ the right side of (6.7) is strictly negative and the algorithm never stops. Then the argument used in the proof of Theorem 5.1 gives a subseqence satisfying (6.6). For any limit point **c** and m*, $E(i(e_{m^*})-(\mathbf{c},\mathbf{i}))f'((\mathbf{c},\mathbf{i}))=0$ which implies $P((\mathbf{c},\mathbf{i})>\phi)=0$. If $\phi=\phi^*$ and the algorithm stops, then $E(i(e_{m^*})-(\mathbf{c},\mathbf{i}))f'((\mathbf{c},\mathbf{i}))=0$, implying $P((\mathbf{c},\mathbf{i})>\phi)=0$. If it does not stop, the same conclusion is reached. In either case, we get $Ef((\mathbf{c},\mathbf{i}))$=f(0).

One version of the normalized algorithm starts with a function g(x) defined on [-1,1] such that g'(x) is zero for x≤0, positive for x>0, and g'' continuous, bounded and non-negative. For $\phi>\phi^*$ define f(x)=g(x-ϕ). Applying the algorithm to this f(x) gives the following result:

Corollary 6.2 *For **c** any limit or stopping point of the normalized algorithm, P(edge(**c**,)>ϕ)=0.*

proof: P(edge(**c**)>ϕ)=0 is necessary and sufficient for a global minimum of g((**c**,**i**)).

comments: i) A line search for the minimum in the direction of increasing b_{m^*} could also be done in the normalized algorithms and would speed convergence. Whether it is worth it computationally is unclear. ii) Taking (w.o.l.g) g(0)=1 gives the upper bound

6.3 Arc-x4 is a normalized arc algorithm

In normalized arcing, each new classifier get a unit vote. Hence, the proportion of misclassifications of $\mathbf{x}_n$ is $(\mathbf{c},\mathbf{i}(\mathbf{x}_n))$. At each stage in arc-x4, the current probability $Q(\mathbf{x}_n)$ is proportional to $(\mathbf{c},\mathbf{i}(\mathbf{x}_n))^4$. Hence, the arc-x4 algorithm is minimizing $E(edge(\mathbf{c}))^5$.

6.4. Random Hyperplanes is (almost) a normalized arc algorithm

Take $\phi > \phi^*$ and consider trying to minimize $E((\mathbf{c},\mathbf{i})) - \phi)^+$ using the normalized algorithm. At each stage, the current probability $Q(\mathbf{x}_n)$ is proportional to $I((\mathbf{c},\mathbf{i}(\mathbf{x}_n)) - \phi > 0)$ where $I\ (\bullet)$ is the indicator function, and this is the Ji-Ma reweighting. In the standard form of the normalized algorithm, e_{m^*} minimizes $Q(e_m)$ and the corresponding b value is increased by one. Because x^+ does not have a bounded 2nd derivative and because the Ji-Ma algorithm does only a restricted search for the minimizing e_m, the normalized arc algorithm has to be modified a bit to make the convergence proof work.

Take $\varepsilon > 0$ small, and g(x) on [-1,1] such that g'(x)=0 for $x \leq 0$, g'(x)=1 for $x > \varepsilon$, and g'(x) in $[0, \varepsilon]$ rising smoothly from 0 to 1 so that g''(x) is continuous, bounded and non-negative on [-1,1]. Now let $\phi > \phi^*$ and consider minimizing $Eg((\mathbf{c},\mathbf{i}) - \phi)$. Take $\delta > 0$ and at each stage, search randomly to find a classifier h_m such that $Q(e_m) \leq \phi^* + \delta$. Then as long as $\phi^* + \delta - \phi < 0$, the result of Theorem 4.2 holds.

The original Ji-Ma algorithm sets the values of two parameters $\alpha > 0, \eta > 0$. In our notation $\phi = \alpha, \phi^* + \delta = .5 - \eta$. Ji and Ma set the values of α, β by an experimental search. This is not surprising since the key value ϕ^* is unknown.

7. **Universal Arcing Algorithms**

The algorithms in Sections 4 and 5 for finding a **c** such that P(edge(**c**)> ϕ)=0 depend on setting the value of the parameter ϕ. They assume knowledge of ϕ^*, select a fixed $\phi > \phi^*$ and find a **c** such that P(edge(**c**)> ϕ)=0. Assuming we do not know ϕ^* is there an arcing algorithm that will converge to a **c** such that P(edge(**c**)> ϕ)=0 for all $\phi > \phi^*$? We describe two such algorithms which we call arc-u1 and arc-u2. Both are of unnormalized type using the function e^x, but do not use line searches to set the step size.

arc-u1 algorithm

Define a sequence of step sizes Δ_k so that $\Delta_k \to 0$, $\sum_k \Delta_k = \infty$. After the kth step, denote $\pi(n) = \prod_m e^{b_m i(e_m)}$, set $Q(\mathbf{x}_n) = \pi(n) / \sum \pi(n)$, and $m^* = \text{argmin}_m Q(e_m)$. Increase b_{m^*} by Δ_k. The update for Q is given by $\pi(n) \to \pi(n) e^{\Delta_k i(e_{m^*})}$.

Theorem 7.1 *Any limit point **c** of the arccu1 algorithm satisfies P(edge(**c**)> ϕ)=0 for all $\phi > \phi^*$.*

proof: Note that

$$P((\mathbf{b},\mathbf{i}) > \phi|\mathbf{b}|) \leq e^{-\phi|\mathbf{b}|} E \prod_m e^{b_m i(e_m)}$$

After k steps, denote the right hand side by f_k and $\varepsilon_k = Q(e_{m^*})$. Then

$$f_{k+1} = e^{-\phi|\mathbf{b}|} e^{-\phi \Delta_k} E e^{\Delta_k i(e_{m^*})} \prod_m e^{b_m i(e_m)}$$

so

$$f_{k+1}=e^{-\phi\Delta_k}(1+(e^{\Delta_k}-1)\varepsilon_k)f_k. \tag{7.1}$$

Then

$$f_{K+1}=\prod_{k=1}^{K}e^{-\phi\Delta_k}(1+(e^{\Delta_k}-1)\varepsilon_k)$$

and

$$\log(f_{K+1})=-\sum_{k=1}^{K}(\Delta_k(\phi-\varepsilon_k)-O(\Delta_k^2)) \tag{7.2}$$

Since $\varepsilon_k \le \phi^*$, (7.2) implies that $f_{K+1}\to 0$ for all $\phi>\phi^*$.

We note that arc-u1, with its small b_{m^*} increases, may be slow to converge compared to the unnormalized algorithms utilizing line searches. Using a small constant increase Δ will produce a **c** such that P(edge(**c**) > ϕ)=0 for all $\phi>[\phi^*/(1-\Delta)]+O(\Delta^2)$.

<u>algorithm arc-u2</u>

This algorithm is the same as arc-u1 except for the step size. Fix some upper bound b<1 for ϕ^*, say, for instance, b=.9. At the current step, let ε=Q(e_{m^*}), and let s=min($\max_n(\mathbf{c},\mathbf{i}(\mathbf{x}_n))$, b). Define the step size as

$$\Delta=\log(s/(1-s))+\log((1-\varepsilon)/\varepsilon)$$

Update Q as in arc-u1.

Theorem 7.2 *Any limit point **c** of the arc -u2 algorithm satisfies P(edge(**c**)>ϕ)=0 for all $\phi>\phi^*$*

proof: At the kth step, let s_k=min($\max_n(\mathbf{c}_k,\mathbf{i}(\mathbf{x}_n))$, b) and ε_k=Q(e_{m^*}). Then, from (7.1)

$$f_{k+1}/f_k=\left(\frac{1-\varepsilon_k}{1-s_k}\right)^{1-\phi}\left(\frac{\varepsilon_k}{s_k}\right)^{\phi}$$

Taking logs and using the fact that $\varepsilon_k \le \phi^*$ gives

$$\log(f_{k+1}/f_k)\le(1-\phi)\log(1-\phi^*)+\phi\log(\phi^*)-(1-\phi)\log(1-s_k)-\phi\log(s_k)$$

Take $\phi>\phi^*$. The function

$$\theta(s)=(1-\phi)\log(1-\phi^*)+\phi\log\phi^*-(1-\phi)\log(1-s)-\phi\log s$$

is negative for $\phi^*<s<h(\phi)$ where $h(\phi)>\phi$. Let $\bar{s}=\limsup(s_k)$. Since $\bar{s}\le$ b, $f_k\to 0$ for $\phi\le h^{-1}(b)<b$, hence $\bar{s}\le h^{-1}(b)$. Repeating this argument leads to the conclusion that $\bar{s}=\phi^*$.

Comments: Schapire et.al.[1997] note that the Adaboost algorithm produces a **c** such that P(edge(**c**) > ϕ)=0 for all

$$\phi>(\log 2+\log(1-\phi^*))/(-\log(\phi^*)+\log(1-\phi^*)) \qquad (7.3)$$

(my version of their remark). If , for instance, $\phi^*=.25$, the lower limit in (7.3) is .37.

8. **Empirical Results**

In realistic settings, the number of classifiers is infinite. For instance, we will work with the set H of all CART classification trees with a minimum node size L. Given a probabilty Q on the training set, it is not computationally feasible to find the classification tree in H that minimizes $Q(e_m)$. Instead, procedures such as CART use a stepwise method to find an approximate minimizer. The trees constructed in the experiments described below were based directly on the weighted training set and not on a training set formed by random resampling..

One can use the framework of the preceding sections to study what happens on a number of synthetic and real data sets. In particular, the Schapire et. al. explanation implies that the for an algorithm procducing a convex combination **c** of classifiers, the lower the value of top(**c**)= $\max_n(\mathbf{c},\mathbf{i}(\mathbf{x}_n))$, the lower the test set error. This implication can be checked empirically by comparing the results of running Adaboost and arc-u2.

Arc-u2 converges most rapidly if the bound b is not too much larger than ϕ^*. In all of our runs, we took b=.5. A minor modification was made to arc-u2. Since an exact minimum of $Q(e_m)$ is not guaranteed, then for the classifier h_{m^*} selected, $Q(e_{m^*})$ may be larger than the current value of top(**c**). If so, the step size is negative. To prevent this, we take $\delta>0$ small such that if the computed step size is smaller than δ, it is set equal to δ.

In the runs we made, L was set equal to 10. The first set of runs involved four sets of synthetic data, of dimension 21,20,20,20 with 300 instances in the training sets and using a 3000 member test set. The fist data set had three classes, the others two. For definitions, see Breiman[1996]. The runs were repeated 10 times with newly generated data and 100 steps of each algorithm were taken in each run. Both algorithms were given the same training and test sets. At the end of a run, top(**c**) and the test set errors were computed. These were averaged over the 10 repetitions and the results given in Table 1.

Table 1 Synthetic Data-- Top(**c**) (x100) and Test Set Error(%)

	Adaboost		Arc-u2	
data set	Test Set Error	Top(**c**)	Test Set Error	Top(**c**)
waveform	18.4	24.2	18.6	10.9
twonorm	5.9	23.5	8.8	5.2
threenorm	18.6	24.1	18.3	11.1
ringnorm	7.7	20.5	10.4	6.0

In the next experiment, real data sets were used and Adaboost and arc-u2 compared. The data sets are mostly the same ones used in Breiman [1996] and [1996b] and, except for the heart data, are in the UCI repository. They consist of six moderate sized and three larger data sets. A summary is given in Table 2.

Table 2 Data Set Summary

Data Set	#Training	#Test	#Variables	#Classes
heart	1395	140	16	2
breast cancer	699	70	9	2
ionosphere	351	35	34	2
diabetes	768	77	8	2
glass	214	21	9	6
soybean	683	68	35	19
letters	15,000	5000	16	26
satellite	4,435	2000	36	6
DNA	2,000	1,186	60	3

With the moderate-sized data sets, ten runs were made with 100 steps of each algorithm per run. In each run, 10% of the data was selected at random and held out as a test set. The results in Table 3 are the averages over the 10 runs.

Table 3 Moderate Size Data--Top(**c**) (x100) and Test Set Error(%)

	Adaboost		Arc-u2	
data set	Test Set Error	Top(**c**)	Test Set Error	Top(**c**)
heart	.14	14.9	1.9	2.4
breast cancer	2.9	13.9	3.7	11.3
ionosphere	4.6	13.1	8.3	2.5
diabetes	25.2	26.6	26.0	21.6
glass	26.2	32.6	28.6	30.6
soybean	6.9	45.4	6.9	45.3

The larger data sets were already split into designated training and test sets. The results shown in Table 4 are based on single runs of 100 steps for each algorithm

Table 4 Larger Size Data-- Top(**c**) (x100) and Test Set Error(%)

	Adaboost		Arc-u2	
data set	Test Set Error	Top(**c**)	Test Set Error	Top(**c**)
letters	2.5	28.5	2.9	25.6
satellite	8.6	29.5	9.2	21.4
DNA	4.1	29.7	4.5	29.2

9 **Remarks**

The Schapire et. al. [1997] explanation for the success of Adaboost in producing lower generalization error was that it acted on the training data in a way that increased the margin, i.e. decreased the edge. In particular, since Adaboost is a unnormalized algorithm, it's aimed at producing lower values of top(**c**). On the other hand, as seen in the empirical results of Section 8, arc-u2 produces consistently lower values of top(**c**) than Adaboost, yet has generally higher test set error.

We believe that the reason for this is that the push toward lower values of top(**c**) produces overfitting of the training data. For the moderate sized data sets, Tables 5 compares the test set values and top(**c**) for 50 steps of arc-u2 to the values for 100 steps.

Table 5 Moderate Size Data--Top(**c**) (x100) and Test Set Error(%) for Arc-u2

	50 steps		100 steps	
data set	Test Set Error	Top(**c**)	Test Set Error	Top(**c**)
heart	1.4	4.2	1.9	2.4
breast cancer	3.4	13.0	3.7	11.3
ionosphere	6.9	4.6	8.3	2.5
diabetes	26.4	23.5	26.0	21.6
glass	27.6	32.0	28.6	30.0
soybean	6.9	45.5	6.9	45.3

For most of the data sets, although top(**c**) decreases as we go from 50 to 100 steps of arc-u2, the error rate increases. The Schapire et. al. bound is loose and does not track the the effects of overfitting. The implication that the authors draw from their bound is that the lower the value of top(**c**) the lower the resulting generalization error. But the empirical results do not support this.

The framework given in Section 2 and 3 provides a simple setting for understanding arcing algorithms. But the empirical results suggest complex behavior for the generalization error. We will have to go deeper into the mystery to understand more fully why arcing works.

10 **Acknowlegement**

I am indebted to Yoav Freund for providing me with an early draft of the important work on margins by Schapire et.al. [1997]. This paper helped to crystallize some of my thinking about arcing and resulted in the present study.

References

Breiman, L. [1996] Bias, Variance, and Arcing Classifiers, Technical Report 460, Statistics Department, University of California (available at www.stat.berkeley.edu)

Breiman, L. [1996b] Bagging predictors , Machine Learning 26, No. 2, pp. 123-140

Drucker, H. and Cortes, C. [1995] Boosting decision trees, Advances in Neural Information Processing Systems Vol. 8, pp. 479-485.

Freund, Y. and Schapire, R. [1995] A decision-theoretic generalization of on-line learning and an application to boosting. to appear , Journal of Computer and System Sciences.

Freund, Y. and Schapire, R. [1996] Experiments with a new boosting algorithm, Machine Learning: Proceedings of the Thirteenth International Conference, pp. 148-156

Ji, C. and Ma, S[1997] Combinations of weak classifiers, Special Issue of Neural Networks and Pattern Recognition, IEEE Trans. Neural Networks, Vol. 8, pp. 32-42

Kong, E. and Dietterich, T., [1996] Error-correcting output coding corrects bias and variance, Proceedings of the Twelfth International Conference on Machine Learning, 313-321

Quinlan, J.R.[1996] Bagging, Boosting, and C4.5, Proceedings of AAAI'96 National Conference, on Artificial Intelligence, pp. 725-730.

Schapire, R.,Freund, Y., Bartlett, P., and Lee, W[1997] Boosting the margin, (available at http://www.research.att.com/~yoav look under "publications".)