Appears in
Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98),
Madison, WI, July 1998.

Boosting in the limit: Maximizing the margin of learned ensembles

Adam J. Grove and **Dale Schuurmans***
NEC Research Institute
4 Independence Way
Princeton NJ 08540, USA
{grove,dale}@research.nj.nec.com

Abstract

The "minimum margin" of an ensemble classifier on a given training set is, roughly speaking, the smallest vote it gives to any correct training label. Recent work has shown that the Adaboost algorithm is particularly effective at producing ensembles with large minimum margins, and theory suggests that this may account for its success at reducing generalization error. We note, however, that the problem of finding good margins is closely related to linear programming, and we use this connection to derive and test new "LPboosting" algorithms that achieve better minimum margins than Adaboost.

However, these algorithms do *not* always yield better generalization performance. In fact, more often the opposite is true. We report on a series of controlled experiments which show that no simple version of the minimum-margin story can be complete. We conclude that the crucial question as to *why* boosting works so well in practice, and how to further improve upon it, remains mostly open.

Some of our experiments are interesting for another reason: we show that Adaboost sometimes does overfit—eventually. This may take a very long time to occur, however, which is perhaps why this phenomenon has gone largely unnoticed.

1 Introduction

Recently, there has been great interest in ensemble methods for learning classifiers, and in particular in *boosting* [FS97] (or *arcing* [Bre96a]) algorithms. These methods take a given "base" learning algorithm and repeatedly apply it to reweighted versions of the original training data, producing a collection of hypotheses $h_1, ..., h_b$ which are then combined in a final aggregate classifier via a weighted linear vote. Despite their "black box" construction—one typically does not need to modify the base learner at all—these techniques have proven surprisingly effective at improving generalization performance in a wide variety of domains, and for diverse base learners.

For these procedures there are several conceivable ways to determine the example reweightings at each step, as well as the final hypothesis weights. The best known boosting procedure, **Adaboost** [FS97], computes them in a particular way: at each round j, the example weights for the next round $j+1$ are adjusted so that the most recent base hypothesis only obtains error rate 1/2 on the reweighted training set (Figure 1). The intuition behind this is to force the learner to focus on the "difficult" training examples and pay less attention to those that the most recent hypothesis got right. Adaboost then uses a specific formula for hypothesis weights that yields a nice theoretical guarantee about training performance: if the base learner can always find a hypothesis with error bounded strictly below $1/2$ for any reweighting of the training data, then Adaboost is guaranteed to produce a final aggregate hypothesis with zero training set error after a finite number of boosting rounds [FS97].

Adaboost.M1(t training instances $\mathbf{x}$ and labels $\mathbf{y}$,
        base learner H,
        max boosting rounds b)
  $\mathbf{u} := (1/t, ..., 1/t)$;(example weights)
  for $j = 1...b$
    $h_j := H(\mathbf{x}, \mathbf{y}, \mathbf{u})$;(base hypothesis)
    $\epsilon_j := \sum_{i:h_j(x_i)\neq y_i} u_i$;(weighted error)
    if $\epsilon_j > 1/2$, $b := j-1$, break
    $w_j := \log \frac{\epsilon_j}{1-\epsilon_j}$;(hypothesis weight)
    for each u_i
      if $h_j(x_i) \neq y_i$, $u_i := u_i/(2\epsilon_j)$
      else, $u_i := u_i/(2(1-\epsilon_j))$
    end
  end
  return $\mathbf{h} = (h_1, ..., h_b)$, $\mathbf{w} = (w_1, ..., w_b)$, b

Figure 1: Procedure Adaboost

Of course, this only addresses training error, and there is no real reason from this to believe that Adaboost should *generalize* well to unseen test examples. In fact, one would naively expect the opposite: since Adaboost produces increasingly complex hypotheses from a larger space, one would think that Adaboost should quickly "overfit" the training data and produce a final hypothesis with worse test error than the single original hypothesis returned by the base learner. However, there is a growing body of empirical evidence that suggests Adaboost is remarkably effective at reducing the test set error of several well-known learning algorithms, often significantly and across a variety of do-

*Primary affiliation: Institute for Research in Cognitive Science, University of Pennsylvania.

mains (more or less robustly, but with occasional exceptions) [FS96a, Qui96, MO97, BK97].

This raises a central question of the field: why is boosting so successful at improving the generalization performance of already carefully designed learning algorithms? One thing that is clear is that boosting's success cannot be directly attributed to a notion of variance reduction [Bre96b, BK97]. This mystery is further compounded by the observation that Adaboost's generalization error often continues to decrease even after it has achieved perfect accuracy on the training set. What more information could it possibly be obtaining from the training data? If we could "explain" boosting's real-world success satisfactorily, we might hope to construct better procedures based upon that explanation.

Recent progress in understanding this issue has been made by [SFBL97] who appeal to the notion of *margins*. Rather than focus exclusively on classification error, Schapire *et al.* consider the *strength* of the votes given to the correct class labels. They observe that even after zero training error has been achieved, Adaboost tends to increase the vote it gives to the correct class label (relative to the next strongest label vote), and they posit this as an explanation for why Adaboost's test set error continues to decrease.

[SFBL97] examines the effect of Adaboost on the distribution of margins as a whole. However, one of their experimental observations is that Adaboost seems to be particularly effective at increasing the margins of "difficult" examples (those with small margins), perhaps even at the price of reducing the margins of other examples. This suggests the more concrete hypothesis that the size of the *minimum* (worst) margin is the key to generalization performance, and that Adaboost's success is due to its ability to increase this minimum. Supporting this conjecture are two theoretical results, also from [SFBL97]: (1) in the limit, Adaboost is guaranteed to achieve a minimum margin that is at least half the best possible, and (2) given that a minimum margin of $\theta > 0$ can be achieved, then there is a $O(1/\theta)$ bound on generalization error that holds *independently* of the size of the ensemble. (See Section 2 for more about this theory.)

This, then, is the background for our work. As we observe in Sections 3 and 4 it is often quite easy to improve upon the minimum margins found by Adaboost by using Linear Programming (LP) techniques in a variety of ways. So *if* minimum margins really are the principle determiner of learning-success, this should lead to even more effective ensemble methods. The truth, though, seems to be more complex. We run a series of controlled experiments to test the significance of margins, using various real world data-sets from the UCI repository. As we discuss in Section 5 the results are at times mixed, but overall it seems clear that the single-minded pursuit of good minimum margins is detrimental.[1]

A different set of experiments, in Section 6, considers the long run behavior of Adaboost. When we boost well beyond the range of previously reported experiments, margins may improve for a long time—but beyond some point, generalization error often deteriorates simultaneously. This is additional evidence against any simple version of the minimum margin story. It also demonstrates that (in the limit) Adaboost is vulnerable to overfitting, which is just as one would expect *a priori*, but perhaps contrary to the lessons one might take from most of the short-run experiments reported in the literature.

We would have been happier to report that the minimum margin story was unambiguously complete and correct—we would then truly "understand" boosting's success *and* be able to improve upon it substantially. Our more negative results, though, are still important. It is always necessary to test theoretical proposals of this type with rigorous experimentation. We conclude that the key problem of discovering properties of training set performance that are predictive of generalization error in real-world practice still demands significant research effort.

2 The minimum margin

We begin by defining the *margin* and other relevant terminology more carefully. An *ensemble* $\mathbf{h} = (h_1, ..., h_b)$ is a finite vector of hypotheses. Given an ensemble, together with a matching set of *weights* $\mathbf{w} = (w_1, ..., w_b)$, where $w_j \geq 0$ and $\sum w_j = 1$, one classifies examples by taking a weighted vote among the individual hypotheses and choosing the label that receives the largest vote.

Let the training set be a collection of labeled examples $(x_1, y_1), ..., (x_t, y_t)$, where the labels come from $L = \{1, \ldots, l\}$. Each individual hypothesis maps an example to a single label[2] in L. Let $v_{i,y}$ be the total vote that the weighted ensemble casts for label y on example x_i; that is $v_{i,y} = \sum_{h_j : h_j(x_i) = y} w_j$. Note that $\sum_{y \in L} v_{i,y} = 1$.

For a given example x_i, we would like more votes to go to the true label y_i than to any other label, because then the ensemble would classify x_i correctly. This suggests defining the margin m_i as $v_{i,y_i} - \max_{y \neq y_i} v_{i,y}$; that is, the total vote for the true label minus the vote for the most popular wrong label. This quantity is in the range $[-1, 1]$ and is positive iff the weighted ensemble classifies x_i correctly. It is 1 when there is a unanimous vote for the correct label. The definition just given can be found in [SFBL97] and [Bre97b].

However, we instead concentrate on a different, but similar, quantity defined by $m_i = v_{i,y_i} - \sum_{y \neq y_i} v_{i,y}$ $(= 2v_{i,y_i} - 1)$. When $|L| = 2$ this is identical to the previous definition; but it is only a lower bound in general (and thus, it can be negative even when the correct label gets the most votes—*i.e.*, when there is a plurality but not a majority).[3] This alternative definition is much easier to work with, since it involves a sum rather than a max. In fact, most of the theoretical work in the literature uses the second notion (or else considers the 2-class case where there is no distinction).[4] For this reason, we will dispense with the first definition entirely, and from this point use the term *margin* always to

[1] We note that [Bre97a] reports a single experiment that corroborates this point, but as noted in [Bre97b], there is some question as to whether this controlled for all relevant factors.

[2] We note that it easy to generalize our results to handle the case where hypotheses map examples to *distributions* over L.

[3] Breiman has sometimes called this second quantity the *edge*.

[4] However, see the extended version of [SFBL97], available at www.research.att.com/~schapire.

refer to $m_i = v_{i,y_i} - \sum_{y \neq y_i} v_{i,y}$. The reader should remain aware of this subtle terminological distinction.

As discussed in the introduction, there is some recent and important theory involving the margin. [SFBL97] show a result bounding generalization error in terms of the margin distribution achieved on the training set. More precisely, if the distribution of margins has at most a fraction $f(\theta)$ below θ, the we get a bound on the test error of $f(\theta) + O\left(1/\sqrt{t}(\frac{\log t \log H}{\theta^2} + \log(1/\delta))^{1/2}\right)$, where t is the size of the training sample, H is the size of the base hypothesis class, and δ is the confidence.[5] This theorem applies if we know the $(100\, f(\theta))$ %-ile margin θ for any $\theta > 0$. However, the only *a priori* theoretical connection to Adaboost we know of involves the minimum margin (*i.e.*, $\theta^* = \sup\{\theta : f(\theta) = 0\}$): [SFBL97] show that Adaboost achieves at least half of the best possible minimum (see Section 6 for more discussion). Recently Breiman has proven a similar generalization theorem [Bre97b], which speaks *only* about the minimum margin—and thereby obtains even stronger bounds. Of course, neither of these results is likely to be accurate in predicting the actual errors achieved in particular real-world problems (among other reasons, because of the $O(\cdot)$ formulation in [SFBL97]); perhaps their real importance is in suggesting the *qualitative effect* of the minimum margin achieved all else being equal.

3 Maximizing margins: A Linear Program

The recent theoretical results just discussed suggest that, beyond minimizing training-set error, we should attempt to make the minimum margin as large as possible. It has already been observed [Bre97a] that this maximization problem can be formulated as a linear program.[6] Here we quickly re-demonstrate this formulation, because it is the starting point and basis to our work.

For a fixed ensemble $\mathbf{h}$ and training set $(\mathbf{x}, \mathbf{y})$, define an *error* matrix Z which contains entries z_{ij} such that $z_{ij} = 1$ if $h_j(x_i) = y_i$ and $z_{ij} = -1$ if $h_j(x_i) \neq y_i$. In terms of Z, the margin obtained on example i corresponds to the simple dot product $m_i = \sum_j w_j z_{ij} = \mathbf{w} \cdot \mathbf{z}_i$.

$$\begin{array}{c|ccc|c}
x_1 & z_{11} & \cdots & z_{1b} & m_1 \\
\vdots & \vdots & & \vdots & \vdots \\
x_t & z_{t1} & \cdots & z_{tb} & m_t \\
 & h_1 & \cdots & h_b & \\
 & w_1 & \cdots & w_b &
\end{array}$$

Our goal is to find a weight vector $\mathbf{w}$ that obtains the largest possible margin subject to the constraints $w_j \geq 0$ and $\sum w_j = 1$. This is a maxi-min problem where we choose $\mathbf{w}$ to maximize $\min_i \mathbf{w} \cdot \mathbf{z}_i$ subject to $w_j \geq 0$ and $\sum w_j = 1$. We turn this into a linear programming problem simply by conjecturing a lower bound, m, on the minimum value and choosing $(m, \mathbf{w})$ to maximize m subject to $\mathbf{w} \cdot \mathbf{z}_i \geq m$, $i = 1, ..., t$, and $w_j \geq 0$, $\sum w_j = 1$. (Note that m is not constrained to be nonnegative.)

[5] These results can be extended to infinite base hypothesis spaces to appealing to the standard VC dimension bounds.

[6] See also [FS96b], which predates the "margin" terminology and also casts the definitions in terms of game theory rather than linear programming, but otherwise makes the same point.

LP-Adaboost$(\mathbf{x}, \mathbf{y}, H, b)$
  $(\mathbf{h}, \mathbf{w}, b) :=$ Adaboost$(\mathbf{x}, \mathbf{y}, H, b)$
  Construct error matrix Z (of dimension $t \times b$)
  $(m, \mathbf{w}) :=$ solve linear program: minimize m
    subject to $\sum_{j=1}^{b} w_j z_{ij} \geq m$,
      $w_j \geq 0$, $\sum w_j = 1$
  return $\mathbf{h}$, $\mathbf{w}$

Figure 2: Procedure LP-Adaboost

Although straightforward, this suggests a simple test of how important "minimum margins" are for real learning problems. Consider the ensemble produced by Adaboost on a given problem. Although Adaboost also provides a weighting over this ensemble, we could simply ignore this and instead re-solve for the weights using the LP just formulated. We call this procedure **LP-Adaboost**; see Figure 2. Clearly this will achieve a minimum margin at least as good as the Adaboost weighting does. In fact, as we see in Section 5, it generally does significantly better. Importantly, this uses the same ensemble as Adaboost and so completely controls for expressive power (which otherwise can be a problem; see [Bre97b]). To the extent to which minimum margins really determine generalization error, we should expect this to improve generalization performance. But as we see in Section 5, this expectation is not realized empirically.

As an aside, we note that LP-Adaboost is clearly more computationally expensive than Adaboost, since it has to construct the Z matrix and then solve the resulting LP. However, this is still feasible for few thousands of examples and hundreds of hypotheses using the better LP packages known today; this range covers many (although definitely not all) experiments being reported in the literature. The deeper question is, of course, whether one would want to use LP-Adaboost at all (even ignoring computational costs).

4 The Dual Linear Program

Before investigating the empirical performance of LP-Adaboost, we first show that the *dual* of the linear program formulated in Section 3 leads to another boosting procedure which uses an alternative technique for computing the *example* reweightings. This procedure can provably achieve the optimal margin over the entire base hypothesis space.

From the work of [Bre97a, FS96b] it is known that the dual of the previous linear program has a very natural interpretation in our setting. (For a review of the standard concepts of primality and duality in LP see [Van96, Lue84].) In the dual problem we maintain a weight u_i for each training example, and a constraint $\sum_i u_i z_{ij} \leq s$ for each hypothesis (*i.e.*, column in Z). Here, s is the conjectured bound on the dual objective. The dual linear program then is to choose $(s, \mathbf{u})$ to *minimize* s subject to $\sum_i u_i z_{ij} \leq s$, $j = 1, ..., b$, and $\sum u_i = 1$, $u_i \geq 0$ [Van96, p70].

Notice that these constraints have a natural interpretation. The vector $\mathbf{u}$ is constrained to be (formally) a probability distribution over the *examples*, and the j'th column of Z corresponds to the sequence of predictions (1 if correct, -1

DualLPboost($\mathbf{x}$, $\mathbf{y}$, H, tolerance ϵ, max iterations $b_{\max}$)
    $\mathbf{u} := (1/t, ..., 1/t), b := 0, m := -1, s := 1$
    repeat
        $b := b + 1$
        $h_b := H(\mathbf{x}, \mathbf{y}, \mathbf{u})$
        $s := \sum_{i=1}^{t} u_i z_{ib}$;(score of h_b on $\mathbf{u}$)
        if $s - m < \epsilon$ or $b > b_{\max}$, $b := b - 1$, break
        $(m, \mathbf{w}, s, \mathbf{u})$:= solve primal/dual linear program:
            maximize m s.t. $\sum_{j=1}^{b} w_j z_{ij} \geq m$, $w_j \geq 0, \sum w_j = 1$
            minimize s s.t. $\sum_{i=1}^{t} u_i z_{ij} \leq s$, $u_i \geq 0, \sum u_i = 1$
    end
    return $\mathbf{h}$, $\mathbf{w}$, b

Figure 3: Procedure DualLPboost

if wrong) made by hypothesis h_j on these examples. Thus, $\sum_i u_i\, z_{ij}$ is simply the (weighted) *score* achieved by h_j on the reweighting of the training set given by $\mathbf{u}$. (*I.e.*, score on a scale where -1 means it gets all the examples incorrect and $+1$ means all were correct.) Thus, for each h_j, we have a constraint that h_j achieves score of at most s. Minimizing s means to find the worst possible best score. We can therefore rephrase the dual problem as follows: Find a reweighting of (*i.e.*, probability distribution over) the training set, such that the score s of the best hypothesis in the ensemble is as small as possible. Basically, we are looking for a *hard* distribution.

By duality theory, there is a correspondence between the primal and dual problems; it is enough to solve just one of them, and a solution to the other is easily recovered. Moreover, the optimal objective value is the *same* in both the primal and the dual. For us, this implies: *The largest minimum margin achievable for given Z* (by choosing the best weight vector over the ensemble) *is exactly the same as the smallest best score achievable* (by choosing the "hardest" reweighting of the training set). This remarkable fact, an immediate consequence of duality theory, also appears in [Bre97a, Bre97b] and [FS96b].

This notion of duality can extend to the entire base hypothesis space: if we implicitly consider an ensemble that contains every base hypothesis and yet somehow manage to identify the hardest example reweighting that yields the lowest maximum score over all base hypotheses, then this will correspond to a (hopefully sparse) weight vector over the entire base hypothesis space that yields the best possible margin. One way of attempting to do this leads to our next boosting strategy, **DualLPboost** (Figure 3).

This procedure follows a completely different approach to identifying hard example reweightings than Adaboost. The idea is to take a current ensemble $h_1, ..., h_{b-1}$, solve the resulting LP problem, and take the dual solution vector $\mathbf{u}$ as the next example reweighting. It then calls the base learner to add another base hypothesis to the ensemble.

By construction, $\mathbf{u}$ is maximally hard for the given ensemble, so either a new hypothesis can be found that obtains a better score on this reweighting, or we have converged to an optimal solution. This gives us a convergence test—if we cannot find a good enough base hypothesis, then we know that we have obtained the best achievable margin for the entire space, even if we have only seen a small fraction of the base hypotheses.[7] Contrast this with Adaboost, which only stops if it cannot find a hypothesis that does better than chance (*i.e.*, score of exactly 0). In practice, Adaboost may never terminate.

Proposition 1 *Suppose we have a base learner H that can always find the best base hypothesis for any given reweighting. Then, if the base hypothesis space is finite, DualLPboost is guaranteed to achieve optimal weight vector $\mathbf{w}$ after a finite number of boosting rounds.*

The key point to realize is that DualLPboost shares the most important characteristic of boosting algorithms: it only accesses a base learner by repeatedly sending it reweighted versions of the given training sample, and incrementally builds its ensemble. The only difference is that DualLPboost keeps much more state between calls to the base learner: it needs to maintain the entire Z matrix, whereas Adaboost only needs to keep track of the current reweighting. How much of a problem this causes depends on the computational effort of LP solving vs. the time taken by the base learner.

5 Generalization performance

We tested these procedures on several of the data sets from the UCI repository ([MM, KSD96]). Generally, we trained on a randomly drawn subset of 90% of the examples in a data set, and tested on the other 10%; we repeated this 100 times for each set and averaged the results.[8] For all procedures we set $b_{\max} = 50$, although they could terminate with a smaller ensemble if any of the various stopping conditions were triggered. By construction, LP-Adaboost uses the same ensemble as Adaboost.

We report average error rates for each method. We omit confidence intervals in the tables, but instead report winning percentages for each method against Adaboost as the baseline; this allows for a statistically weaker but distribution-free comparison. This number is the percentage of the 100 runs in which each method had higher accuracy than Adaboost (allocating half credit for a tie). In general, these winning percentages correlate well with test error.

We tested two base learners. The first, [FS96a]'s FindAttrTest, searches a very simple hypothesis space. Each classifier is defined by a single attribute A, a value V for that attribute, and three labels $l_{yes}, l_{no}, l_?$. For discrete attributes A, one tests an example's value of A against V; if the example has no value for A predict $l_?$, if the example's value for A equals V then predict l_{yes}, otherwise predict l_{no}. For continuous attributes V functions as a threshold—we predict l_{yes} if an example's value for A is $\leq V$; otherwise we predict l_{no} or $l_?$ as appropriate. This simple hypothesis class has some nice properties for our experiments: First, it is fast to learn,

[7] Note that this depends crucially on the base learner always being able to find a sufficiently good hypothesis if one exists; see Section 5 for further discussion of this issue.

[8] However, for some large data sets, *chess* and *splice*, we inverted the train/test proportions.

Data set	FindAttrTest error%	win%	Adaboost error%	margin	LP-Adaboost error%	win%	margin	DualLPboost error%	win%	margin
Audiology	52.30	50.0	52.30	-1.0	52.30	50.0	-1.0	54.70	47.0	-0.804
Banding	27.00	19.5	18.88	-0.080	22.37	31.0	0.021	23.88	25.0	0.032
Chess	32.60	0.0	5.24	-0.099	6.49	14.0	0.0	6.59	17.5	0.010
Colic	18.68	44.5	17.95	-0.179	23.08	22.5	-0.005	23.59	19.5	0.002
Glass	47.80	50.0	47.80	-1.0	47.80	50.0	-1.0	45.80	55.5	-0.427
Hepatitis	18.38	49.0	18.19	-0.026	21.44	36.0	0.063	21.44	34.0	0.071
Labor	24.00	14.0	6.50	0.255	7.00	48.0	0.295	6.83	49.5	0.298
Promoter	28.09	4.5	8.82	0.165	9.45	47.5	0.212	9.36	46.5	0.223
Sonar	27.29	12.0	16.76	0.052	17.81	42.5	0.113	19.76	37.0	0.099
Soybean	69.50	50.0	69.50	-1.0	69.50	50.0	-1.0	71.70	37.0	-0.733
Splice	37.70	0.0	10.56	-0.695	17.10	7.5	-0.415	12.34	25.0	-0.170
Vote	4.16	40.0	3.43	-0.056	4.00	41.5	0.002	5.50	24.5	0.019
Wine	34.40	0.5	4.00	0.011	3.06	55.5	0.073	5.00	41.5	0.081

Figure 4: FindAttrTest Results

Data set	C4.5 error%	win%	Adaboost error%	margin	LP-Adaboost error%	win%	margin	DualLPboost error%	win%	margin
Audiology	22.70	17.0	16.39	0.446	16.48	49.0	0.501	18.09	38.5	0.370
Banding	25.58	12.5	15.00	0.528	15.42	45.5	0.565	22.50	20.0	0.430
Chess	4.18	12.5	2.70	0.657	2.74	46.5	0.730	2.97	37.0	0.560
Colic	14.46	67.5	17.03	0.051	18.97	31.5	0.182	18.16	44.0	0.108
Glass	30.91	22.0	23.95	0.513	23.91	49.5	0.624	26.86	38.0	0.386
Hepatitis	21.06	38.0	18.94	0.329	17.56	59.0	0.596	20.00	45.5	0.385
Labor	15.33	43.0	12.83	0.535	13.83	47.0	0.684	15.17	42.0	0.599
Promoter	21.09	10.5	7.55	0.599	8.00	47.0	0.694	13.55	29.5	0.378
Sonar	28.81	16.0	18.10	0.628	18.62	48.0	0.685	25.00	23.0	0.478
Soybean	8.86	28.5	6.97	-0.005	6.55	62.0	0.017	8.41	33.5	0.003
Splice	16.18	0.0	6.83	0.535	7.00	25.0	0.569	11.01	0.0	0.393
Vote	4.95	51.0	5.02	0.723	5.30	44.5	0.795	5.27	44.5	0.756
Wine	9.11	27.0	4.61	0.869	4.89	47.5	0.912	4.50	50.5	0.814

Figure 5: C4.5 Results

so we can easily boost for hundreds of thousands of iterations (see Section 6). We can also explicitly solve the LP for this space to determine the optimal ensemble (according to the minimum margin criterion). But most importantly, [FS96a] have shown that the benefits of Adaboost are particularly decisive for FindAttrTest.

The second learning method we considered is a version of Quinlan's decision tree learning algorithm C4.5 [Qui93].[9] This is at the other end of the spectrum of base learners in that it produces hypotheses from a very expressive space. Adaboost is generally effective for C4.5 but, as [FS96a] point out, the gains are not as dramatic and in fact there is sometimes significant deterioration [Qui96].

In Tables 4 and 5 we present the statistics discussed, along with the average minimum margins obtained over 100 runs. (Note that the base learners invariably obtain margins of -1.) First, consider the behavior of LP-Adaboost vs. Adaboost. When C4.5 is used as the base learner, LP-Adaboost always improves Adaboost's margins, by an average of about 0.1. Similarly, there is a fairly consistent increase in the margins with FindAttrTest; generally around 0.05 with greater relative increase. And yet we do not see any consistent improvement in generalization error! For FindAttrTest, LP-Adaboost is almost always worse, sometimes much so. For C4.5, the typical case shows only slight deterioration—we are not being hurt so much here, but there is definitely no gain on average. Since all else is controlled for in this experiment, this seems to decisively refute any simple version of the minimum margin story. (Of course, this immediately raises the question of whether there is some other property of the margin distribution which is more predictive of generalization performance; see [SFBL97].)

Next, consider the behavior of DualLPboost with FindAttrTest. With one exception (explained below) this yields even better margins. And yet, in comparison with Adaboost, we are frequently hurt even more. (Note however, that DualLPboost constructs a different ensemble, so this comparison is not as tightly controlled.)

The behavior of DualLPboost with C4.5 is also curious. There is a seeming anomaly here: why are the margins sometimes *worse*? There are two reasons. First, even un-

[9] We hedge by saying "a version of C4.5" because it is in fact a re-implementation of this program, based closely on the presentation in [Qui93]. However, it produces identical output in direct comparisons. Our program corresponds to all C4.5's default settings *including pruning*.

der ideal conditions DualLPboost can take many rounds to achieve the best minimum margin, and we stopped it after only 50. So here at least, its convergence rate seems slower. Second, DualLPboost relies crucially on the assumption that the base learner finds a good hypothesis whenever one exists. If the base learner is unable (or unwilling) to find such a hypothesis, then DualLPboost can easily become stuck with no way of continuing. For a classifier like C4.5, which may not even *try* to find the best hypothesis (because of pruning), it is not surprising that DualLPboost can show inferior performance at finding large minimum margins.

Returning to the generalization comparisons, *why* is all our extra effort to optimize margins hurting us? In many cases the answer seems to be the intuitive one; namely we are sometimes increasing the minimum margin at the expense of all the other margins. Essentially, the margin distribution becomes more concentrated near its lower end, which counteracts the fact that the lower margin is higher. Table 6 shows the (average) difference between the 10'th lower-percentile margin and the minimum margin for a few of the data sets; note how these are much closer together these are for LP-Adaboost. Quite frequently 10% of the points (or more) all have margins within 0.001 of the minimum. This table also shows the *median* margin: note that even as the minimum margin is always improved by LP-Adaboost, the median never improves by as much and (especially for FindAttrTest) often *decreases*.

The final illustration of the effect of our optimization on the overall margin distribution is given in Figure 7. Here we plot the cumulative margin distributions obtained by the three procedures on a few data sets. These plots show the distribution obtained from a single run, along with the "mean" distribution obtained over 100 runs (formed by sorting the margins for each of the 100 runs and averaging corresponding values). These plots graphically show the concentration effect discussed above. (The location of the minimum margins are highlighted by the short vertical lines.)

Our observations about the effects of margin distributions on test error emphasize the central open question raised by this work: precisely *what* characteristic of the margin distribution should we be trying to optimize in order to obtain the best possible generalization performance? The minimum margin is not the answer.

6 Boosting in the limit

Our second set of experiments investigates what happens when one runs Adaboost far beyond the 10's to 100's of iterations normally considered. Here we focus on the FindAttrTest base learner, because of its greater tractability, and also because we are able to exactly solve the corresponding LP over the entire hypothesis space. In fact, we do this by running DualLPboost to completion, which generally takes a few hundred rounds at most. (This always works because, for FindAttrTest, we can certainly find the best hypothesis for each reweighting.)

The results show several interesting phenomena. First, we often continue to see significant changes in performance even after 10^5–10^6 boosting rounds. We observe that the minimum margin increases more or less monotonically (excluding fluctuations near the beginning). Furthermore, in our experiments Adaboost's minimum margin always asymptotes at the optimal value found by LP.[10] The theory we are aware of only guarantees half the value—so are we seeing an artifact of FindAttrTest, or does Adaboost always have this property? If the latter, Adaboost can presumably turned into a general, albeit very slow, LP solver.[11] Moreover, if this is the case, then Adaboost would be expected to do exactly as well as DualLPboost (or other exact LP solvers) in the limit, and thus its empirical superiority would have to be explained (somehow) using the fact that it is typically *not* run to anywhere near its asymptotic limit. We believe that this theoretical question is one of the most significant issues raised by our work.

Data/Learner	Adaboost		LP-Adaboost	
	$\Delta_{\min,10\%}$	median	$\Delta_{\min,10\%}$	median
Band–C4.5	0.022	0.584	0.000	0.606
Chess–C4.5	0.025	0.752	0.000	0.859
Colic–C4.5	0.185	0.353	0.021	0.382
Hep–C4.5	0.130	0.626	0.000	0.760
Labor–C4.5	0.018	0.064	0.000	0.749
Splice–C4.5	0.025	0.599	0.000	0.607
Vote–C4.5	0.049	0.890	0.002	0.933
Wine–C4.5	0.018	0.933	0.000	0.968
Band–FindAtt	0.044	0.165	0.000	0.171
Chess–FindAtt	0.110	0.187	0.008	0.142
Colic–FindAtt	0.178	0.173	0.000	0.029
Hep–C4.5	0.031	0.247	0.000	0.171
Labor–FindAtt	0.008	0.368	0.000	0.339
Splice–FindAtt	0.594	0.074	0.104	0.171
Vote–FindAtt	0.198	0.402	0.098	0.350
Wine–C4.5	0.083	0.257	0.000	0.228

Figure 6: Properties of the margin distributions

Another important aspect of our results is the correlation of minimum margin with generalization error. The minimum margin always increases, but the test error often deteriorates with large numbers of boosting rounds (although this can sometimes take tens of thousands of rounds before it becomes apparent). One implication is that these experiments give another, independent, refutation of the straightforward minimum-margin story—because, again, we see errors sometimes increase even as the margin gets better. But this phenomenon is also interesting because it counters what seems to be very common "folklore" that boosting is strongly resistant to overfitting. To a certain extent this may be true, but our experiments suggest that this is perhaps only the case in certain regimes (which presumably differ with the problem and the base learner being considered), and should not be relied on in general.

Figures 8, 9, and 10 illustrate three typical asymptotic

[10] But note that there are typically many different solutions that achieve the same best minimum margin. Thus, even though the asymptotic margins coincide, there is no guarantee that Adaboost and our particular LP solver will find the same weighted ensembles.

[11] I.e., an LP solver that does not require any tuning based on prior knowledge of the LP's solution value. It is in this sense that such a result, if true, would strengthen [FS96b].

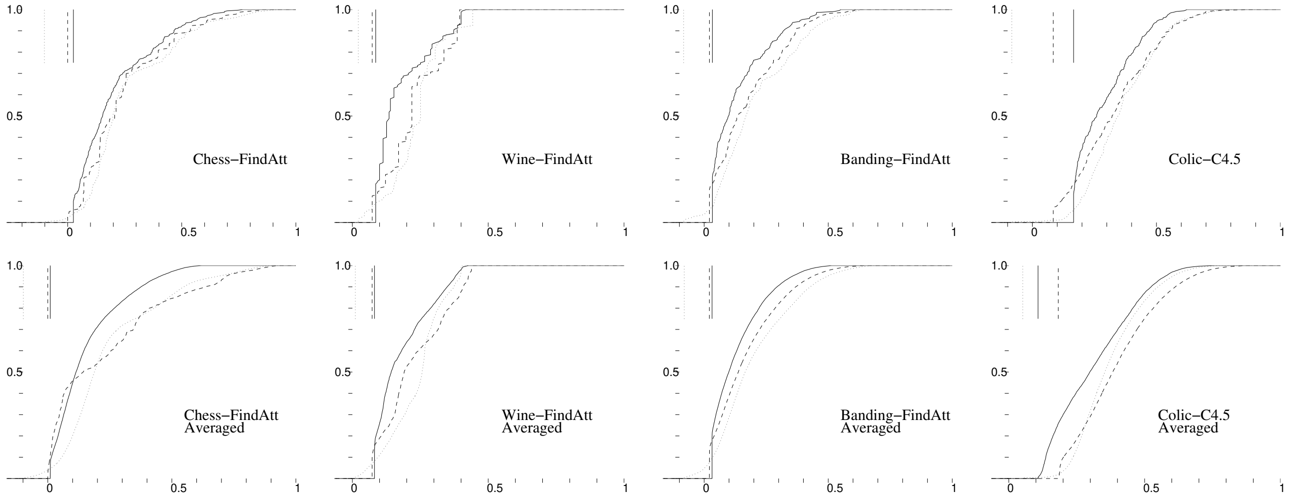

Figure 7: Cumulative margin distributions for (· · · · ·) Adaboost, (– – – –) LP-Adaboost, (——) DualLPboost

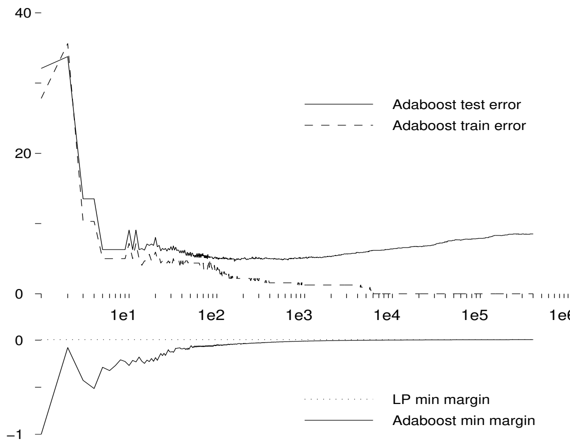

Figure 8: Adaboost with FindAttrTest on "chess"

runs. Here we consider a single train/test split for each problem. The horizontal axis measures boosting rounds on a logarithmic scale. At the top we show training and test error, and below we plot the corresponding minimum margin over the training set. The dotted reference line shows the minimum margin obtained by the exact LP solution.

7 Conclusions

Recent work in the theory of ensemble methods has given us new insight as to why boosting, and related methods, are as successful as they are. But only careful experimentation can tell us how correct or comprehensive this theory is in explaining the "real world" behavior of these methods. The principal contribution of this paper is that we have carried out some of the experiments needed to test the significance of the "minimum margin" in ensemble learning. In the process, we have further elaborated on the deep connection between boosting and linear programming techniques, suggesting new algorithms more directly motivated by LP.

Our experiments prove that increasing the minimum mar-

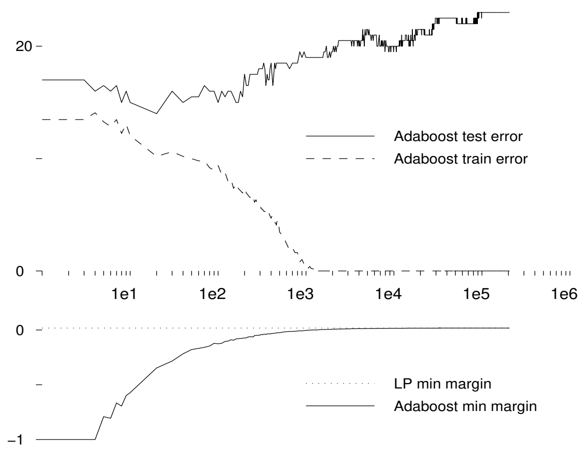

Figure 9: Adaboost with FindAttrTest on "crx"

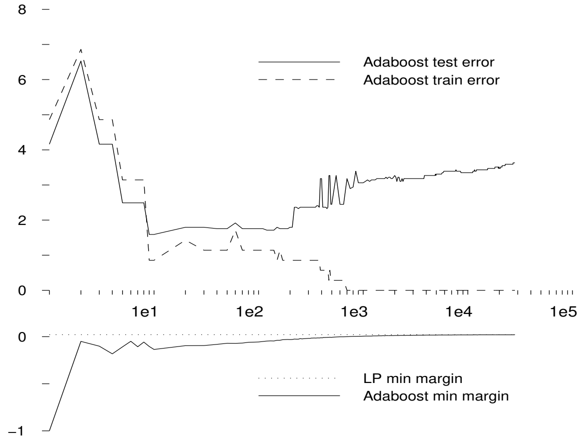

Figure 10: Adaboost with FindAttrTest on "allhypo"

gin is *not* the final, or even the dominant, explanation of Adaboost's success. The question of why Adaboost performs so well on real problems, and whether there are yet better techniques to be found, is still more open than closed.

Finally, our experiments have brought to light other intriguing phenomenon, particularly concerning the very long run behavior of Adaboost. *Eventually*, it seems, Adaboost can overfit (as one would naively expect, but contrary to what one gets by extrapolating most short-run experiments).

8 Further Directions

We close with a few miscellaneous observations about the connection between minimum margins and linear programming which, while interesting, do not directly relate to the main concern of this work. All suggest additional experimental research.

The first concerns the duality result mentioned in Section 4. This suggests a new stopping criteria for boosting algorithms in general, including Adaboost. At each step of a boosting algorithm, we can compute the margin actually achieved by the current weighted ensemble. We can also keep track of the lowest (*i.e.*, worst) score the base learner has been able to achieve (on any reweighting of the training set that it has been presented with). But duality theory, *the optimal achievable margin lies between these values*. Once the gap between these is sufficiently small, one knows that little additional improvement is possible and so can stop. As observed in Section 6, the question of when a procedure like Adaboost has really "converged" can be a difficult one in practice.

Second, and also related to the dual LP formulation, we note the existence of the *ellipsoid algorithm* for solving linear programs, famous because it was the first guaranteed polynomial time algorithm for LP [Kha79, Chv83]. But it has another interesting property that it does not need to see the explicit constraint matrix. Instead, one only needs an "oracle" which—given any proposed assignment to the variables—will produce a violated constraint if one exists. Given such an oracle, the algorithm can find a solution (to some given precision) in polynomial number of calls to the oracle (independent of how many constraints there actually are). It is intriguing that in the dual formulation of our LP *the base learner is such an oracle*: violated constraints correspond to base hypotheses that do better on the current example reweighting than any member of the current weighted ensemble. It therefore seems possible that this idea could lead to another "boosting" algorithm in the style of DualLP-boost, but with perhaps different convergence properties and (at each step) vastly superior computational complexity.

Finally, the idea of minimizing margins is reminiscent of the idea of support vector machines [CV95]. There, however, one tries to find a linear combination that achieves the best worst separation in the sense of Euclidean (*i.e.*, L_2) distance, as opposed to the L_1 notion used to define margins (in a straightforward way but one which we do not formalize here). It turns out that SVMs maximize L_2 margins using quadratic rather than linear programs. But a benefit of maximizing L_1 rather than L_2 margins is that L_1 has a much stronger tendency to produce *sparse* weight vectors. This can yield smaller representations of the learned ensemble, which can be an important consideration in practice [MD97]. In fact our experiments support this. We often find that LP-Adaboost, for instance, ends up giving zero (or negligible) weight to many of the hypothesis in the ensemble and so the *effective* ensemble size is smaller.

Acknowledgments

We would like to thank Leo Breiman, Yoav Freund and Llew Mason for very useful discussions and comments.

References

[BK97] E. Bauer and R. Kohavi. An empirical comparison of voting classification algorithms: bagging, boosting, and variants. http:// robotics. stanford. edu/ users/ ronnyk, 1997.

[Bre96a] L. Breiman. Arcing classifiers. Technical report, Statistics Department, U. C. Berkeley, 1996. http:// www. stat. berkeley. edu/ users/ breiman.

[Bre96b] L. Breiman. Bias, variance, and arcing classifiers. Technical report, Statistics Department, U. C. Berkeley, 1996.

[Bre97a] L. Breiman. Arcing the edge. Technical report, Statistics Department, U. C. Berkeley, 1997.

[Bre97b] L. Breiman. Prediction games and arcing algorithms. Technical report, Statistics Department, U. C. Berkeley, 1997.

[Chv83] V. Chvatál. *Linear Programming*. W. H. Freeman, New York, 1983.

[CV95] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20:273–97, 1995.

[FS96a] Y. Freund and R. Schapire. Experiments with a new boosting algorithm. In *ICML-96*, pages 148–156, 1996.

[FS96b] Y. Freund and R. Schapire. Game theory, on-line prediction and boosting. In *COLT-96*, pages 325–332, 1996.

[FS97] Y. Freund and R. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and Systems Sciences*, 55(1), 1997.

[Kha79] L. Khachian. A polynomial algorithm in linear programming. *Doklady Adademiia Nauk SSSR*, 244:1093–96, 1979. (In Russian) Cited by [Chv83].

[KSD96] R. Kohavi, D. Sommerfield, and J. Dougherty. Data mining using MLC++: A machine learning library in C++. In *Tools with Artificial Intelligence*. IEEE Computer Society, 1996. http:// www. sgi. com/ technology /mlc.

[Lue84] D. Luenberger. *Linear and Nonlinear Programming*. Addison-Wesley, Reading, MA, 1984.

[MD97] D. Margineantu and T. Dietterich. Pruning adaptive boosting. In *ICML-97*, pages 211–218, 1997.

[MM] C. Merz and P. Murphy. UCI repository of machine learning databases. http:// www. ics. uci. edu/ ~mlearn/ MLRepository. html.

[MO97] R. Maclin and D. Opitz. An empirical evaluation of bagging and boosting. In *AAAI-97*, pages 546–551, 1997.

[Qui93] J. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.

[Qui96] J. Quinlan. Bagging, boosting, and C4.5. In *AAAI-96*, pages 725–730, 1996.

[SFBL97] R. Schapire, Y. Freund, P. Bartlett, and W. Lee. Boosting the margin: a new explanation for the effectiveness of voting methods. In *ICML-97*, pages 322–330, 1997.

[Van96] R. Vanderbei. *Linear Programming: Foundations and Extensions*. Kluwer, Boston, 1996.