Boosting Noisy Data

Abba Krieger KRIEGER@WHARTON.UPENN.EDU
Department of Statistics, Wharton School,University of Pennsylvania, Philadelphia, PA 19104, USA

Chuan Long CHUAN21@WHARTON.UPENN.EDU
Department of Statistics, Wharton School,University of Pennsylvania, Philadelphia, PA 19104, USA

Abraham Wyner AJW@WHARTON.UPENN.EDU
Department of Statistics, Wharton School,University of Pennsylvania, Philadelphia, PA 19104, USA

Abstract

We study two classes of problems where it is known that boosting will overfit data. The first case occurs when the training data is corrupted by independent label noise, and the second occurs when the regions significantly overlap. We begin by observing that in the proper framework, overlapping regions is a special case of noisy data. Our main contribution is the introduction of a new ensemble learning strategy based on the careful application of both bagging and boosting. We demonstrate experimentally that the performance of this algorithm is superior to boosting when the training set is noisy, and importantly, nearly identical otherwise. We corroborate this on some real data. Finally, by comparing boosting with this new algorithm, we provide evidence for a new explanation for boosting's still remarkable resistance to overfitting.

1. Introduction

While the overall success of AdaBoost (Freund and Schapire 1996) in particular and boosting in general, is indisputable, there is increasing evidence that boosting algorithms are not quite as immune from overfitting as indicated in early reports. Dietterich (2000) and Opitz and Maclin (1999) among others report that boosting is quite susceptible to data corrupted by the introduction of independent label noise. Friedman et. al.(2000) provide another example of a data set for which boosting overfits: concentric spheres with significant overlap. In short, research to date point to two classes of problems where boosting may overfit: 1) independent label noise 2) overlapping regions. To be fair, classification in such situations is hardest to accomplish.

1.1 A Model for Noisy Data

The simplest model for noisy data considers, as in Dietterich (2000), the addition of independent label noise. For example, assume that Y is a binary class label. Suppose there exists region B such that for every feature vector $X \in B$ it follows that $Y = 1$, otherwise $Y = 0$. Now suppose for every instance X the label Y is reversed with independent probability $p < 1/2$. More generally, let $f(x)$ represent the complete density for feature vector X. We model noise by letting $p(x)$ represent the probability that label Y equals 0 given that $X = x$; that is we let $p(x) = P(Y = 0|X = x)$. Intuitively, this model generates data in a two step procedure: first we choose a feature vector X and then, depending on the value of X, we choose the label Y according to $p(x)$.

Now consider a model for overlapping regions. Assume, for simplicity, that Y is binary and that there are two regions. Let $f_0(x)$ be the density of feature vector X with $Y = 0$. Similarly, let $f_1(x)$ be the density of feature vector X with $Y = 1$. If the data contains a fraction ω of points with label $Y = 0$ then the distribution $f(x)$ on the feature vectors is given by the ω *mixture* of $f_0(x)$ and $f_1(x)$:

$$f(x) = \omega f_0(x) + (1 - \omega)f_1(x).$$

Intuitively, the observations are generated in reverse of the two step procedure for noisy data. First, the class label is chosen and then the feature vector is drawn according to f_0 or f_1. The regions are overlapping if there exists features x that are possible under both f_1 and f_0. The mixture model can be transformed into the noisy data model, with

$$p(x) = P(Y = 0|X = x) = \frac{\omega f_0(x)}{\omega f_0(x) + (1-\omega) f_1(x)}.$$

The converse is also true: any noisy model, given by $f(x)$ and $p(x)$ can be transformed into a mixture of two *noiseless* models by letting

$$f_0(x) = p(x) f(x)/\omega$$
$$f_1(x) = (1 - p(x)) f(x)/(1-\omega)$$
$$\omega = \int f(x) p(x) dx.$$

Hence, any noisy model ($f(x)$, $p(x)$) is equivalent to the mixture of the two noiseless models ($f_0(x)$, $f_1(x)$, ω). So, the two classes of problems for which classification can be expected to be hardest and for which boosting algorithms are least effective, are essentially equivalent.

1.2 Preventing Overfitting in Noisy Environments

One way to prevent boosting from overfitting in a noisy environment begins with the belief that boosting fits noise only in its later stages. Thus, a potential preventative measure would be a reduction in the number of iterations, or an adaptive scheme using cross-validation. Another similar suggestion, would allow for weighting functions (of the margin) that increase more slowly than exponential, see for example, the algorithm Gentle-Boost of Friedman et al.(2000).

A second approach is to 'smooth' the boosted classifier. In a noisy environment, this makes obvious sense. What is not entirely clear is that smoothing can be accomplished without reversing the boosting process and producing an underfit. More critically, since the noise level is essentially a hidden variable, the smoothing procedure cannot be so destructive as to render boosting ineffective in zero noise environments.

It is this second suggestion that we pursue here. We propose to combine the bagging procedure and the boosting procedure in a careful way, whereby bagging operates to smooth the already boosted classifier. We shall show that this new approach, which we label the $BB(K, M, \rho)$ algorithm (if the parameters are left unspecified we simply use the label BB), helps to fix any resultant overfitting caused by applying boosting to a noisy data set.

1.3 The BB Algorithm

Given a training set T of size N, (bagging) parameters $\rho > 0$ and K, and (boosting) parameter M. For $k = 1, ..., K$:

- Generate a replicate training set $T_k^{'}$ of size $T_k^{'} = \rho N$, by sub-sampling with replacement.
- Generate a boosted classifier, $C_k^*(x)$, based on $T_k^{'}$ using M iterations of the base learner.

Output the aggregate classifier C^*:

$$C^*(x) = \frac{1}{K} \Sigma_{k=1}^{K} C_k^*(x). \qquad (1)$$

In short, the BB-algorithm performs a bootstrap aggregation on the boosted base learner. The parameter selection is fundamental: K is the chosen number of bootstrap replicates, M is the number of iterations of the boosting algorithm. The sub-sample consists of a fraction ρ of the training set. In our experiments, we shall show that good choices of K, M and ρ are 15, 15 and 1/2 respectively. Since the total number of iterations of the base-learner is $K \times M$, it follows the $BB(15, 15, 1/2)$ algorithm is no more complex than AdaBoost using 225 iterations. Notice that we do not allow boosting to run for hundreds of iterations. This is a fundamental feature of our algorithm and it arises from the observation that boosting is a self-smoothing algorithm. We shall show (in section 3) that running boosting for many iterations, well past the point of achieving zero training set error rate, allows the algorithm to smooth itself. Since the smoothing in BB is accomplished directly by bootstrapping (bagging) the large number of iterations are not needed and can be limited to about 15.

To understand how boosting overfits noisy data and also to motivate our algorithm, we look at a two class classification problem as an example. Suppose the probability for any point in a two dimensional unit square belonging to class 1 is 80%, and the probability for it to belong to class 0 is 20%. For convenience, we assume there are 800 points in the training set having class label 1 and 200 points having class label 0. Since boosting is actually a weighted aggregation of multiple trees, we can then think of the result as a partition of the unit square; in other words each point in the training set claims its own territory by putting its class label on it. We know the optimal classification rule for this simple problem is to classify every point to belong to class 1, so the Bayes classifier has error rate 20%. However, due to boosting's specially designed strong ability to focus on hard points, the 200 points from class 0 also have their territory. If we assume the total area of the region corresponding to class 0 is p, then the generalization error rate is $p * 0.8 + (1-p) * 0.2 \geq 0.2$. The error rate for boosting is higher than the optimal, because boosting overfits the training set.

Recall that bagging improves classification by aggregating decision trees trained on bootstrapped replicates. Thus, if we apply bagging after boosting, the noise contained in the training set will hopefully be averaged out (i.e. smoothed) when different boosted classifiers are combined into an ensemble. Actually, we can do better by subsampling a proportion ρ from the original training set. The trouble with subsampling is that real problems are much more complicated than this example. Most problems are mixtures of signal and noise. Thus, it follows that bagging with subsamples provides stronger protection against noise. It also follows that discarding much of the data may cause the boosting step to lose its power to capture the signal. An important problem is to discover the optimal sub-sampling proportion of the training set in order to balance the trade-off between bagging's ability to smooth and boosting's ability to detect signal.

Combining bagging and boosting is an obvious idea and not entirely new. Webb (2000) combines bagging and boosting into the algorithm multiple boosting (MB). Although he shows that MB does result in slightly improved classification error rates compared to boosting, he makes no connection with noisy environments. In our work, we shall show that for noisy data sets BB is a great improvement over boosting and comparable to boosting in noiseless environments (Webb (2000) limited its analysis to datasets in the UCI repository which are mostly noiseless). While MB is similar to BB in that they both combine bagging and boosting without increasing total algorithmic complexity, there are two major differences. First, *BB* uses subsampling to create smaller replicate training sets. The motivation for sub-sampling is that it makes a better smoother. Our experiments support this conjecture. Secondly, the MB algorithm uses wagging instead of bagging which forces every training set instance to be included in every replicate, albeit with randomly chosen positive weight. The *BB* algorithm uses equal weights and simple sampling with replacement. The effect of this is significant since boosting increases the weights of hard-to-classify points. Every point eventually contributes to each iteration which limits the smoothing.

Another discussion of the combination of boosting and bagging can be found in the discussion paper by Buhlmann and Yu (2000). They suggest the application of bagging and boosting in the opposite direction of our combination. The effect of this is that no matter how efficient or smooth the weak learner is, boosting eventually fits the noise. Consequently, any improvement gained from bagging the weak learner is eventually eliminated in the boosting stage.

2. Empirical Results

We begin by providing empirical evidence to support our main conjecture: that BB helps to reduce overfitting in the presence of noise. Following the approach of Friedman et. al (2000), we use synthetic data and provide pictures to strengthen intuition and vividly demonstrate our principle ideas. We will look at two cases: 1) independent label noise and 2) overlapping regions and mixture models.

2.1 Applying the BB Algorithm with Independent Label Noise

We begin by specifying a model that is simple enough to understand and to display, but complex enough to constitute a sufficient challenge for our base classifier. Specifically, we suppose that there are five dependent feature variables, $X_1, \ldots, X_5$. We generate each feature i.i.d. from a uniform distribution on the unit interval. Furthermore, we assume, for simplicity, that the class label Y is binary and determined only by X_1 and X_2 according to the following:

$$Y = \begin{cases} 1\,, & X_1 \leq X_2 \\ 0\,, & \text{otherwise.} \end{cases}$$

From the definition it is clear that the other three features, X_3, X_4, X_5 are masking variables from which Y is independent. Furthermore, we assume that addition of noise which randomly and independently flips the observed class label with some fixed probability $1-p$. Under this distortion we have that $P(Y = 1|X_1 > X_2) = 1-p$ and $P(Y = 0|X_1 \leq X_2) = 1-p$. The special case where $p = 1$ corresponds to noiseless data. For reference, we label this set-up the unit square model with a *linear* boundary and independent label noise.

We generate a training set consisting of one thousand pairs (Y, X) from the Unit Square model, with $X = (X_1, X_2, .., X_5)$. We set the noise parameter p to be 0.8. Consequently, any classifier is faced with the difficult task of learning the boundary $X_1 = X_2$ not only in the presence of our masking features X_3, X_4 and X_5, but also in the presence of noise, which peppers the regions with dots of 1's in fields of 0's (and vice versa). To test our classifier we provide a noiseless test data set of 10000 points which provides a clean distortion free backdrop to measure performance. We first consider AdaBoost (225 iterations) using C4.5 as its base classifier. To assess the performance of this classifier, as well as others, we consider the projection of the test sets errors into the unit square. We use black dots to refer to misclassified points, i.e points with $x_1 \geq x_2$, that the classifier labels $Y = 1$. Similarly, black dots in the region $x_1 < x_2$ represent the reverse mistake. In Figure 1, we display this so called *error projection* graph for AdaBoost. Observe that most of the errors are near the boundary, but there are still sizeable concentrations of errors throughout the region which correspond to overfitting. The overall error rate, equal to the area of the black dots on the unit square, is 15.5%. For comparison, since the test set is noiseless, the Bayes error rate is 0.

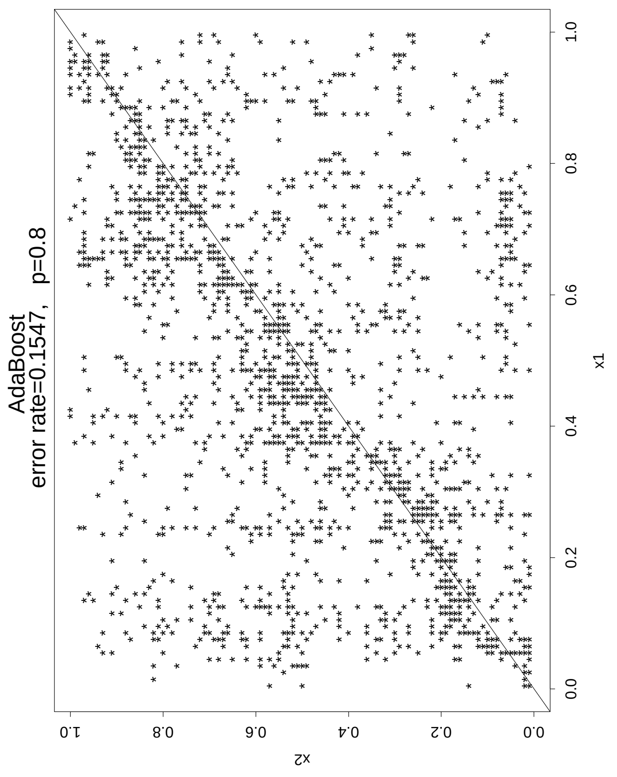

Figure 1. error projection Graph for AdaBoost with p=0.8

To demonstrate BB's performance, we bootstrap 15 subsamples of 333 (out of 1000) observations. We then apply AdaBoost (15 iterations) to each of the 15 subsamples. The error projections for the subsamples are displayed in 15 of the 16 panels in Figure 2 (all but the right panel at the top). The resulting error rates range from 15.8% to 20.09%. Finally, we create our aggregate classifier by voting among the 15 bootstrapped samples. The performance of the aggregate classifier is displayed in Figure 2 in the extreme upper right. Notice, that this panel has much more white than the other 15 panels indicating that the probability of misclassification (6.79%) is much smaller. It is also apparent from this panel that most of the error for the BB classifier is near the boundary where x_1 is close to x_2. We know that the black dots inside each class region in each of the fifteen boosting panels are due to boosting's overfitting of the noise and their locations are completely random, so when we combine all the boosted subsamples the errors are averaged out. In contrast, the black dots close to the boundary are due to the inherent nature of the data and their locations are consistent among the boosted subsamples. Consequently, they will survive the final smoothing step in the BB aggregation.

In Figure 3, boosting is contrasted with BB for different levels of noise. To create a level playing field, we used the same total number of iterations for boosting and the aggregate BB classifier. Specifically, the single boosting algorithm is a combination of 225 base classifiers. The aggregate BB classifier is also a combination of 225 base classifiers, resulting from the combination of the 15 subsamples of 15 bootstrap iterations.

Consider the bottom panels in Figure 3, corresponding to the zero noise case (i.e. $p = 1$). Notice that the performances of AdaBoost and BB are nearly the same (error rates of 2.4% and 2.65% respectively). It will turn out that aggregation does not decrease performance even in a distortion free environment. In the middle set of panels, the probability of distortion is .1. In this context, BB has an error rate of 4.8% and AdaBoost has an error rate of 6.7%. Finally, we consider the top panels of Figure 3, where the noise level is higher ($p = .8$). Here, the distortion probability is .2. In this case, boosting is overwhelmingly outperformed by BB. With noise, AdaBoost overfits and consequently posts an error rate of 15.47% BB does substantially better with an error of 6.79%. The overall pattern is very clear: the more noise in the training set, the larger the improvement of BB over boosting.

So far we have only looked at a noisy dataset. In this environment, bagging also outperforms boosting since it does not overfit the noise. Furthermore, unlike BB which averages out the noise at the final step, bagging ignores the noise in each iteration. Consequently, it is more effective at reducing noise than BB. On the other hand, this advantage comes with a price: noise distorts the real signal. While bagging ignores the noise, it cannot capture the signal as well as boosting. Thus there is apparently a simple trade-off: boosting does better in noiseless environments, while bagging performs better on noisy datasets. We shall soon see that BB can adapt appropriately: it can still boost a weak learner (such as c4.5) when applied to noiseless data, and it can outperform bagging when applied to noisy datasets.

To demonstrate this we compare the base classifier

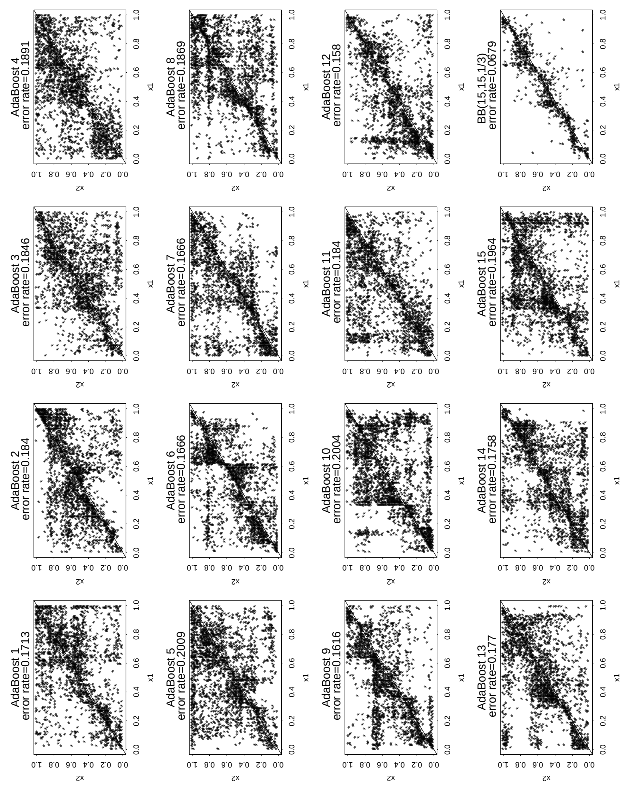
Figure 2. Step by step output of BB(15,15,1/3). The graph in the upper right corner is the error projections of BB(15,15,1/3). The other graphs are the error projections for the 15 applications of Adaboost to the 15 sub-samples. Each point in the pictures represent an error in the test. The noise parameter p is 0.8.

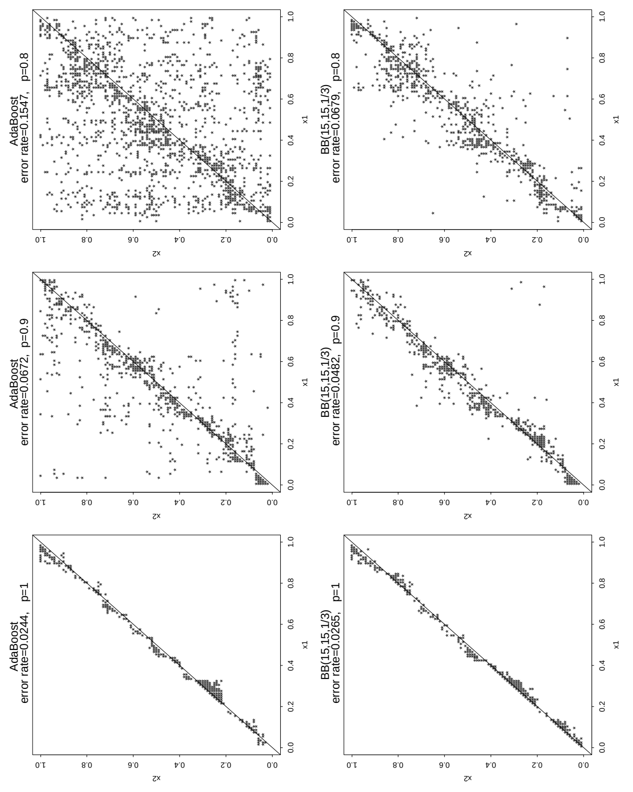
Figure 3. Comparison of AdaBoost vs. BB(15,15,1/3). The noise parameter is p. The total number of base classifiers used in AdaBoost and BB are the same.

C4.5, bagging, boosting and BB in a more elaborate simulation study. We move away from the Unit Square model, to a more complicated model whose regions are defined by spheres. In our experiments, we allow ρ to be 1/3, 1/2 or 1, but set K and M both equal to 15. For simplicity, we let $BB(\rho)$ equal $BB(15, 15, \rho)$.

As in the unit square model with a linear boundary, we let the dependent feature variables, $X_1, \ldots, X_5$ be i.i.d. uniform on the unit interval. The class label Y is again binary. The more difficult *spherical* boundary defines the region where $Y = 1$ as follows:

$$Y = \begin{cases} 1\,, & X \in B \\ 0\,, & \text{otherwise.} \end{cases}$$

where B is the five dimensional sphere centered at $(\frac{1}{2}, ..., \frac{1}{2})$ with radius $r = 0.62$. We again distort the data with noise so that $P(Y = 0|X \in B) = 1 - p$ and $P(Y = 1|X \notin B) = 1 - p$.

We generate a training set consisting of one thousand pairs (Y, X). We allow the noise parameter p to be in the range .7 to 1. The training set is used to create the various classifiers. In order to validate a classifier we apply it to a test set also consisting of one thousand pairs (Y, X), but with $p = 1$. The reason $p = 1$ is used for validation is to ensure that the performance reported is due purely to misclassification, and is not confounded by noise.

Table 1 presents the percentage of errors on the test sample. Since the whole experiment is replicated ten times, the top entry in each cell is the average percentage error. The number below is the standard error of the mean of the error rates across the 10 replications. We summarize the main conclusions thus far:

- Each of the classifiers outperforms C4.5. This is because the region that distinguishes one class from another is spherical rather than rectangular.
- When p is close to 1 (i.e., little noise) AdaBoost outperforms bagging. Conversely, when p is small bagging does better.

Table 1. Error Rate

	$p=1$	$p=0.9$	$p=0.8$	$p=0.7$	AVE
C4.5	18.7	23.8	27.5	30.6	25.1
	0.6	0.6	1	0.7	
Bag	11.7	13.9	17.1	21.4	16
	0.4	0.4	0.5	0.6	
AdaB	8.8	14.7	20.2	26.8	17.6
	0.2	0.4	0.5	0.5	
BB(1)	8.6	12.9	16.9	23.2	15.4
	0.3	0.3	0.5	0.6	
BB(1/2)	7.9	12.1	17.2	21.7	14.7
	0.2	0.4	0.4	0.9	
BB(1/3)	8.1	12	16.2	21	14.3
	0.3	0.3	0.6	0.7	

- BB is adaptive (with respect to the noise level); AdaBoost and bagging are not. When $\rho = 1/3$, BB *uniformly* outperforms both bagging and AdaBoost.

- The last column is the average percentage error over all values of p considered. This column shows that BB is, on average, the best classifier.

2.2 BB Applied to Mixture Models

Any noisy model can be regarded as a mixture model, so we expect to observe similar improvements of BB over boosting. Again we use pictures to convey the main idea.

The data we are going to use for the mixture model is two dimensional two norm data with three masking variables. The two classes are drawn with equal probability from a two dimensional Normal distribution with unit covariance matrix. Class 1 has mean (0,0) and class 0 has mean (a, a). The parameter a controls the extent of overlap between the two classes. For any fixed a, the optimal separating surface is the straight line $x1 + x2 = a$. In the following experiment, we select a value for a by solving the equation $P(X_1 + X_2 > a) = 1 - p$, where X_1, X_2 are i.i.d. standard Normal random variables and p is the parameter which controls the Bayes error rate. The three masking variable, (X_3, X_4, X_5), for both classes, are generated i.i.d. from a Uniform distribution on the interval $[-4, a+4]$.

The training set consists of one thousand points generated according to the above description. To test the classifiers, we generate the validation data set according to the Bayes rule. Specifically, we generates five X values with the same distribution as in the training set, but we assign each point a class label according to the following formula:

$$Y = \begin{cases} 1\,, & x_1 + x_2 \leq a \\ 0\,, & \text{otherwise.} \end{cases}$$

Figures 4 and 5 follow the same pattern as in Figures 2 and 3. It is very clear that bagging smooths out the overfitted noise effect in each of the fifteen boosts. The only difference from the noisy model is that as $1-p$ increases the relative improvement of BB decreases. The reason is that in the mixture model, $p(x) = P(Y = 1|X = x)$ is not a constant as in the noisy model. Bagging's smoothing ability depends on $p(x)$, the closer $p(x)$ is to 0.5, the weaker bagging becomes in term of its smoothing power. When $1-p$ is large (the two Normals are close to each other), there are more points which are close to the optimal separating line and those points are hard to be fixed by bagging. The points that can be fixed by bagging are those which are far away from the separating line, and they have less weight when $1-p$ is large.

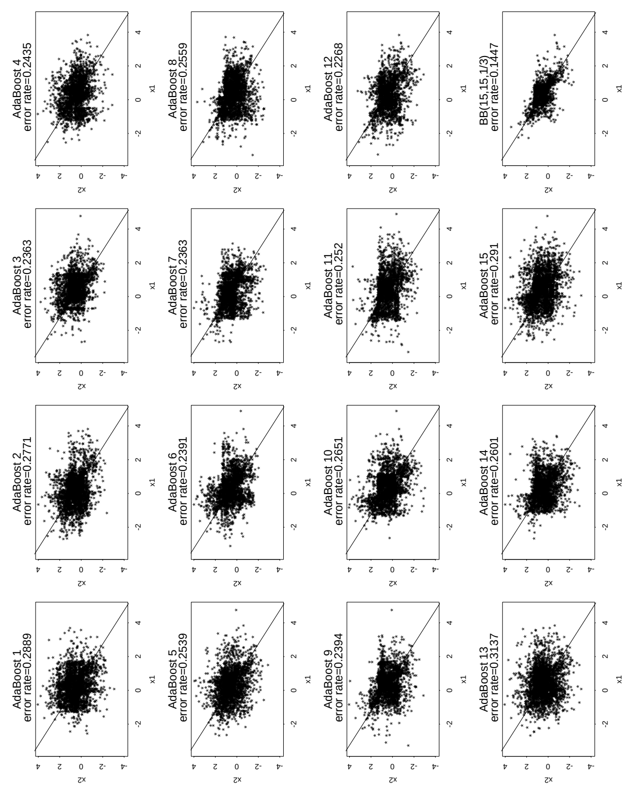

Figure 4. Step by step output of BB with twonorm data. The upper right corner is the final BB classifier, the other graphs are the 15 replications of Adaboost. Each point in the picture represent an error in the test. The optimal separating surface is the straight line $x1 + x2 = a$. The overlap parameter p is 0.8.

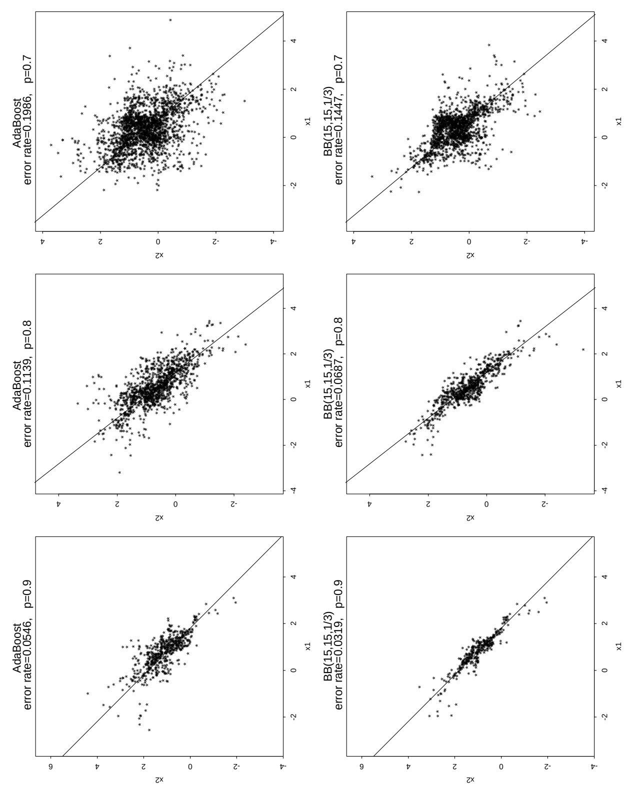

Figure 5. Comparison of AdaBoost vs. BB. The parameter p controls the Bayes error which also reflect how far away the two Normals are from each other. The total number of base classifiers used in AdaBoost and BB is the same.

3. Self-Smoothing: Why boosting resists overfitting

In this section we explore the idea that boosting acts as its own smoother. Our evidence is entirely empirical, but compelling.

We will try to frame our comparison directly to the $BB(15, 15, 1/3)$ algorithm, which constructs 15 bootstrapped replicates each consisting of a sub-sample of size $N/3$. Each of the 15 replicates is then boosted for 15 iterations, for a total of 225 iterations of the base learner. We will consider running Boosting for an identical number of iterations.

In Figure 2 we saw that the main reason that BB improves generalization error is that aggregation of the bootstrap replicates averages out the overfitting in each stage. This is possible since each replicate forms a boosted classifier that overfits in different places. The aggregation of these boosted classifiers smooths out the overfitting. Now in boosting alone, we do not have bootstrap replicates exactly. However, in discrete Adaboost (which is the version of boosting we employ throughout), each iteration is a weighted replicate. It is our conjecture that when the data are noisy the weights move around sufficiently to create replicates that are diverse enough to produce smoothing when averaged together to form the ensemble. If this is indeed the case, classifiers constructed by partitioning the iterations in AdaBoost should produce error projection graphs that look like Figure 2. To see this, we partition the 225 iterations of AdaBoost into 15 subensembles. Specifically, let $F(X)$ be the aggregation of all 225 iterations of AdaBoost. That is

$$F(X) = \sum_{i=1}^{225} \alpha_i h_i(X) = \sum_{i=1}^{15} \alpha_i h_i(X) + \ldots + \sum_{i=211}^{225} \alpha_i h_i(X).$$

Now define f_i, for $i = 1, \ldots, 15$, to be the i^{th} summand in the above. We manufacture 15 individual decisions by letting

$$g_i(X) = \begin{cases} 1\,, & \text{if } f_i(X) > 0 \\ 0\,, & \text{otherwise.} \end{cases}$$

Now we are in a position to directly observe how boosting acts as a self-smoother. In Figure 6, we plot the performance of g_i for $i = 1$ to 15 for the unit square model with a linear boundary and with the addition of independent label noise with $p = .8$. We see that each classifier g_i produces noisy estimates (notice the similarity with Fig. 2). The classifier produces extremely noisy boundaries and clouds of errors that blanket the domain. Nevertheless, the variation among the graphs allows the weighted ensemble average $F(X)$ (upper right corner of Fig. 6) to have a smaller test set error rate. Importantly, this is *not* due to the weights. To see this, we report the performance of the unweighted ensemble average $G(X)$ which classifies according to the simple majority of the g_i. The test set error rate for $G(X)$ is 11.7% compared to 11.5% for Adaboost, when $p = .8$. The implication is that overfitting is prevented not by the weights, but by the resampling procedure.

Furthermore, we duplicated this experiment for $p = 1$ (not pictured). In this example, the error rates for g_i range from 3.3% to 3.8%. These errors typically occur at points around the boundary, $y = x$, as each g_i produces its own jagged estimate of the boundary. In contrast, the ensemble error rates are 2.37% and 2.45% for $G(X)$ and $F(x)$ respectively. This occurs because the ensemble average smooths these jagged boundaries to produce an aggregate classifier which better captures the true boundary.

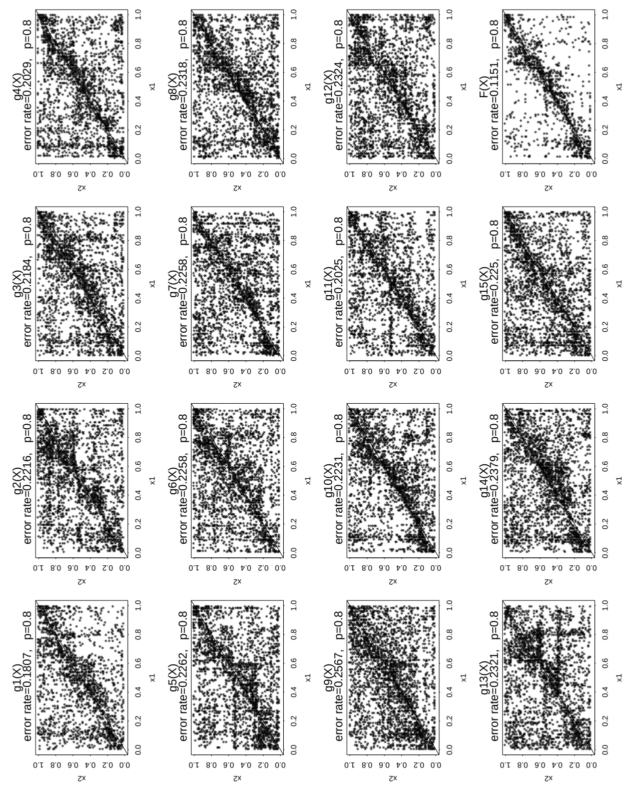

Figure 6. error projection graphs for AdaBoost when chopped into 15 pieces represented by g1(X) to g15(X). The graph in the upper right corner is the error projection graph for $F(X)$- i.e. AdaBoost using 225 iterations. The noise level is $p = .8$

4. Application to Real Data

We considered eight data sets from the UC Irvine repository (see Table 2).The percentage of errors on the test set, averaged for 10 runs, for various classifiers including the proposed bag-boosting are presented in Table 3. For ease of comparison, the error rate ratio for each approach with respect to AdaBoost is given in the second line for each data set. In some data sets, such as hypothyroid, the gain for BB is modest. In other data sets, such as segmentation (see Brodley and Friedl(1999) for an extensive analysis of these data), the gain is impressively about 30%.

Table 2. Description of eight data sets from the UC Irvine Repository

Dataset	Dataset size	Training set size	Attributes Cont/Discr	Classes
BREAST-CANCER-W	699	629	10/0	2
CAR	1728	1000	0/6	4
CREDIT(AUS)	690	621	6/9	2
FLARE	1389	1000	0/10	2
HORSE-COLIC	368	331	8/14	2
HYPOTHYROID	3163	1582	7/18	2
PIMA	768	691	8/0	2
SEGMENTATION	2310	1155	19/0	7

Table 3. Error rate

DATA	C4.5	BAG	ADAB	BB(1)	BB(3/4)	BB(1/2)
BREAST	5.57	3.71	3	2.86	2.86	2.71
	1.86	1.24	1	0.95	0.95	0.9
CAR	10.56	8.87	5.38	4.96	4.42	5.04
	1.96	1.65	1	0.92	0.82	0.94
CREDIT	15.8	14.2	14.49	13.77	14.35	13.33
	1.09	0.98	1	0.95	0.99	0.92
FLARE	18.2	18.64	20.1	19.61	19.15	18.69
	0.91	0.93	1	0.98	0.95	0.93
HORSE	17.57	16.76	19.73	16.49	17.03	18.65
	0.89	0.85	1	0.84	0.86	0.95
HYPO	0.8	0.83	0.96	0.9	0.92	0.94
	0.83	0.87	1	0.95	0.96	0.98
PIMA	28.18	22.73	24.16	24.03	22.47	24.29
	1.17	0.94	1	0.99	0.93	1.01
SEGMENT	4.4	3.34	2.87	2.06	1.9	2.15
	1.53	1.17	1	0.72	0.66	0.75
AVERAGE	12.63	11.14	11.34	10.58	10.39	10.72
	1.28	1.08	1	0.91	0.89	0.92

The percentage error and relative error are row one and row two respectively for each data set.

References

Brodley C., & Friedl M. (1999). Identifying Mislabeled Training Data. *Journal of Artificial Intelligence Research 11* , 131-167.

Buhlmann P., & Yu B. Discussion. (2000). *Ann. Statistics 28(2)*, 377-386.

Dietterich, T. G. (2000). An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting, and randomization. *Machine Learning 40*, 139-158.

Freund, Y. & Schapire, R.E. (1996) Experiments with a New Boosting Algorithm. *In Proceedings of the Thirteenth International conference on Machine Learning* 148-156. San Francisco, CA: Morgan Kaufmann.

Friedman, J., Hastie, T., & Tibshirani, R. (2000). Additive Logistic regression: A Statistical View of boosting(with discussion). *Ann. Statistics 28*, 337-407

Opitz D. & Maclin R. (1999). Popular Ensemble Methods: An Empirical Study. *Journal of Artificial Intelligence Research 11* , 169-198.

Webb, G. (2000) MultiBoosting: A Technique for Combining Boosting and Wagging. *Machine Learning 40 (2)*, 159-196.