Explaining the Success of AdaBoost and Random Forests as Interpolating Classifiers

Abraham J. Wyner [*1], Matthew Olson [†1], Justin Bleich, [‡1], and David Mease

[1]Department of Statistics, The Wharton School, University of Pennsylvania

April 30, 2015

Abstract

There is a large literature explaining why AdaBoost is a successful classifier. The literature on AdaBoost focuses on classifier margins and boosting's interpretation as the optimization of an exponential likelihood function. These existing explanations, however, have been pointed out to be incomplete. A random forest is another popular ensemble method for which there is substantially less explanation in the literature. We introduce a novel perspective on AdaBoost and random forests that proposes that the two algorithms work for similar reasons. While both classifiers achieve similar predictive accuracy, random forests cannot be conceived as a direct optimization procedure. Rather, random forests is a self-averaging, interpolating algorithm which creates what we denote as a "spikey-smooth" classifier, and we view AdaBoost in the same light. We conjecture that both AdaBoost and random forests succeed because of this mechanism. We provide a number of examples and some theoretical justification to support this explanation. In the process, we question the conventional wisdom that suggests

[*]Principal Corresponding Author: ajw@wharton.upenn.edu

[†]Corresponding Author: maolson@wharton.upenn.edu

[‡]Corresponding Author: jbleich@wharton.upenn.edu

that boosting algorithms for classification require regularization or early stopping and should be limited to low complexity classes of learners, such as decision stumps. We conclude that boosting should be used like random forests: with large decision trees and without direct regularization or early stopping.

1 Introduction

In the "boosting" approach to machine learning, a powerful ensemble of classifiers is formed by successively refitting a weak classifier to different weighted realizations of a dataset . This intuitive procedure has seen a tremendous amount of success. In fact, shortly, after its introduction, in a 1996 NIPS conference, Leo Brieman crowned AdaBoost (Freund and Schapire, 1996) (the first boosting algorithm) the "best off-the-shelf classifier in the world (Friedman et al., 2000) ." AdaBoost's early success was immediately followed by efforts to explain and recast it in more conventional statistical terms. The statistical view of boosting holds that AdaBoost is a stage-wise optimization of an exponential loss function (Friedman et al., 2000). This realization was especially fruitful leading to new "boosting machines" (Friedman, 2001; Ridgeway, 2006) that could perform probability estimation and regression as well as adapt to different loss functions. The statistical view, however, is not the only explanation for the success of AdaBoost. The computer science literature has found generalization error guarantees using VC bounds from PAC learning theory and margins (Guestrin, 2006). While some research has cast doubt on the ability of any one of these to fully account for the performance of AdaBoost they are generally understood to be satisfactory (Schapire, 2013).

This paper parts with traditional perspectives on AdaBoost by concentrating our analysis on the implications of the algorithm's ability to perfectly fit the training data in a wide variety of situations. Indeed, common lore in statistical learning suggests that perfectly fitting the training data must inevitably lead to "overfitting." This aversion is built into the DNA of a statistician who has been trained to believe, axiomatically, that data can always be decomposed into signal and noise. Traditionally, the "signal" is always modeled smoothly. The resulting residuals represent the "noise" or the random component in the data. The

statistician's art is to walk the balance between the signal and the noise, extracting as much signal as possible without extending the fit to the noise. In this light, it is counterintuitive that any classifier can ever be successful if every training example is "interpolated" by the algorithm and thus fit without error.

The computer scientist, on the other hand, does not automatically decompose problems into signal and noise. In many classical problems, like image detection, there is no noise in the classical sense. Instead there are only complex signals. There are still residuals, but they don't represent irreducible random errors. If the task is to classify images into those with cats and without, the problem is hard not because it is noisy. There are no cats wearing dog disguises. Consequently, the computer scientist has no dogmatic aversion to interpolating training data. This was the breakthrough.

It is now well-known that interpolating classifiers can work, and work well. The AdaBoost classifier created a huge splash by being better than its established competitors (e.g. CART, neural networks, logistic regression) (Breiman, 1998) and substantively better than the technique of creating an ensemble using the bootstrap (Breiman, 1996). The statistics community was especially confounded by two properties of AdaBoost: 1) interpolation (perfect prediction in sample) was achieved after relatively few iterations, 2) generalization error continues to drop even after interpolation is achieved and maintained.

The main point of this paper is to demonstrate that AdaBoost and similar algorithms work not in spite, but because of interpolation. To bolster this claim, we will draw a constant analogy with random forests (Breiman, 2001), another interpolating classifier. The random forests algorithm, which is also an ensemble-of-trees method, is generally regarded to be among the very best commonly used classifiers (Manuel Fernández-Delgado and Amorim, 2014). Unlike AdaBoost, for which there are multiple accepted explanations , random forests' performance is much more mysterious since traditional statistical frameworks do not necessarily apply. The statistical view of boosting, for example, cannot apply to random forests since the algorithm creates decision trees at random and then averages the results - there is no stage-wise optimization. In this paper, we will put forth the argument that both algorithms are effective for the same reason. We consider AdaBoost and random forests as canonical examples of "interpolating classifiers," which we define to be a classifier's algorithmic property

of fitting the training data completely without error. Each of these interpolating classifiers also exhibits a self-averaging property. We attempt to show that these two properties together make for a classifier with low generalization error. While it is easy to see that random forests has both of mechanisms by design, it is less clear that this is true for AdaBoost.

One of our key contributions will be to present a decomposition of AdaBoost as the weighted sum of interpolating classifiers. Another contribution will be to demonstrate the mechanism by which interpolation combined with averaging creates an effective classifier. It turns out that interpolation provides a kind of robustness to noise: if a classifier fits the data extremely locally, a "noise" point in one region won't affect the fit of the classifier at a nearby location. When coupled with averaging, the result is that the fit stabilizes at regions of the data where there is signal, while the influence of noise points on the fit becomes even more localized. It will be easy to see this point holds true for random forests. For AdaBoost, it is less clear, however, and a decomposition of AdaBoost and simulation results in Section 4 will demonstrate this crucial point. We will observe that the error of AdaBoost at test points near noise points will continue to decrease as AdaBoost is run for more iterations, demonstrating the localizing effect of averaging interpolating classifiers. We will also present a theorem in a later section that demonstrates how weighted averaging of classifiers helps to decrease test error.

We will begin in Section 2 by critiquing some of the existing explanations of AdaBoost. In particular, we will discuss at length some of the shortcomings of the statistical optimization view of AdaBoost. In Section 3, we will discuss the merits of classification procedures that interpolate the training data, that is, that fit the training data set with no error. The main conclusion from this section is that interpolation, done correctly, can provide robustness to a fit in the presence of noise. This discussion will be augmented with simulations discussing the performance of random forests, AdaBoost, and other algorithms in a noisy environment. We will then derive our central observation in Section 4, namely that AdaBoost can be decomposed as a sum of classifiers, each of which fits the training data perfectly. The implication from this observation is that for the best performance, we should run AdaBoost for many iterations with deep trees. The deep trees will allow the component classifiers to interpolate the data, while a large number of iterations will lend to a bagging effect.

In Section 5 we will present theory that explains the mechanism by which self-smoothing improves classification error. Finally, we conclude with a brief discussion in Section 6.

2 Competing Explanations for the Effectiveness of Boosting

In this section we will present an overview of some of the most popular explanations for the success of boosting, with analysis of both the strengths and weaknesses of each approach. Our emphasis will focus on the margins view of boosting and the statistical view of boosting, each of which has a large literature and has led to the development of variants of boosting algorithms. For a more extensive review of the boosting literature, one is well-advised to consult Freund (2012).

Before we begin, we will briefly review the AdaBoost algorithm not only to refresh the reader's mind, but also to establish the exact learning algorithm this paper will consider, as there are many variants of AdaBoost. To this end, the reader is invited to review Algorithm 1. In our setting, we are given N training points $(\mathbf{x}_i, y_i)$ where $\mathbf{x}_i \in \mathcal{X}$ and $y_i \in \{-1, +1\}$. On round m, where $m = 1, \ldots, M$, we fit a weak classifier $G_m(\mathbf{x})$ to a version of the dataset reweighted by some weighting vector $\mathbf{w}_t$. We then calculate the weighted misclassification rate of our chosen learner, and update the weighting measure used in the next round, $\mathbf{w}_{t+1}$. The final classifier is output as the sign of a weighted linear combination of classifiers produced from each stage of the algorithm.

2.1 Margin View of Boosting

Some of the earliest attempts to understand AdaBoost's performance predicted that its generalization error would increase with the number of iterations: as AdaBoost is run for more rounds, it is able to fit the data increasingly well which should lead to overfitting. However, in practice we observe that running boosting for many rounds does not overfit. One of the first attempts to resolve this paradox was explored by Freund and Schapire (Schapire et al., 1998), who focused on the *margins* of AdaBoost. The margins can be thought of as

Algorithm 1 AdaBoost Hastie et al. (2009)

1. Initialize the observation weights $w_i = \frac{1}{N},\ i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:
   (a) Fit a classifier $G_m(\mathbf{x})$ to the training data using weights w_i.
   (b) Compute $\text{err}_m = \frac{\sum_{i=1}^{N} w_i I\left(y_i \neq G_t\left(\mathbf{x}_i\right)\right)}{\sum_{i=1}^{N} w_i}$.
   (c) Compute $a_m = \log\left(\frac{1 - \text{err}_t}{\text{err}_t}\right)$.
   (d) Set $w_i \leftarrow w_i \cdot \exp\left(a_t \cdot I\left(y_i \neq G_t\left(\mathbf{x}_i\right)\right)\right)$
   (e) Set $f_i(\mathbf{x}) = \sum_{m=1}^{M} a_m G_m\left(\mathbf{x}\right)$
3. Output $f(\mathbf{x}) = \text{sign}\left(f_M(\mathbf{x})\right)$

a measure of how confident a classifier is about how it labels each point, and one would hypothetically desire to produce a classifier with as large of margins as possible. Schapire et al. (1998) proved that AdaBoost's generalization error decreases as the size of the margins increase. Indeed, in practice one observes that as AdaBoost is run for many iterations, test error decreases while the size of the empirical margins increase. In fact, recent research has demonstrated that AdaBoost can be refomulated exactly as mirror descent applied to the problem of maximizing the smallest margin in the training set under suitable separability conditions (Freund et al., 2013).

While intuitively appealing, the margins explanation of boosting does not provide the whole story. If maximizing margins were the key to AdaBoost's success, then one would expect that other algorithms that focused harder on optimizing the margin would be have better generalization error. However, one can find evidence against this hypothesis in Brieman's arc-gv algorithm and the LP Boost algorithm (Wang et al., 2011). These algorithms are variants of AdaBoost that provably reduce margins by more than AdaBoost, yet have worse generalization error in practice (Wang et al., 2011; Freund, 2012). Furthermore, (Wang et al., 2011) points out that margin-based generalization bounds tend to be too loose to give qualitative predictions, and may not be crucial to generalization error. Breiman (1997) has

also found simple counterexamples that illustrate cases where larger margin leads to higher generalization error. The margins view of AdaBoost is certainly interesting and merits further investigation, but it does not yet provide a full account for the algorithm's performance.

2.2 Statistical Optimization View of Boosting

Friedman et al. (2000) take great strides to clear up the mystery of boosting to provide statisticians with a statistical view of the subject. The heart of their article is the recasting of boosting as a statistically familiar program for finding an additive model by means of a forward stage-wise approximate optimization of an exponential criterion. In short, this view places boosting firmly in classical statistical territory by clearly defining it a a procedure to search through the space of convex combinations of *weak* learners or *base* classifiers. This explanation has been widely assimilated and has reappeared in the statistical literature as well as in a plethora of computer science articles. The reader is referred to Mayr et al. (2014) for a recent literature review of boosting algorithms in these areas, especially in biostatisics. Subsequent to the seminal publication of Friedman et al. (2000) there has been a flurry of activity dedicated to theoretical analysis of the algorithm. This was made possible by the identification of boosting as optimization, which therefore admits of a mathematically tractable representation. Research on the optimization properties of AdaBoost and the exponential loss function is still an active area of research, see (Mukherjee et al., 2013), for example.

Although the statistical optimization perspective of AdaBoost is surely interesting and informative, there remain problems. First, we observe that the fact that AdaBoost minimizes an exponential loss may not alone count for its performance as a classifier. Wyner (2003) introduces a variant of AdaBoost called Beta-Boost which is very similar to AdaBoost except that by design the exponential loss function is constant throughout the iterations. Despite this, Beta-Boost was able to demonstrate similar performance to AdaBoost on simulated data sets. (Schapire, 2013) also presents evidence casting doubt on the importance of minimizing the exponential loss in AdaBoost.

We also contend that some of the mathematical theory connected with the statistical optimization view of boosting has a disconnect with the types of boosting algorithms that

work in practice. The optimization theory of boosting insists that overfitting can be avoided by requiring the set of weak learners, to be just that: *weak*. Bühlmann and Yu (2003) argues that one can avoid overfitting by employing regularization with weak base learners. However, empirical evidence points to quite the opposite: boosting deep trees for many iterations tends to produce a better classifier than boosted stumps with regularization (Mease and Wyner, 2008). The thrust of our paper will be to demonstrate why we *should* actually expect boosting with deep trees run for many iterations to have better generalization error. Recent work also suggests that boosting deep trees may improve generalization error, although for very different reasons than the ones we present here (Cortes et al., 2014). One last problem with theory associated with the statistical view of boosting is that by its very nature it suggests that we should be able to extract conditional class probability estimates from the boosted fit, as the procedure is apparently maximizing a likelihood function. (Mease and Wyner, 2008), however, points out a number of examples where the implied conditional class estimates from the boosting fit diverge to zero and one. While boosting appears to do an excellent job as a classifier, it apparently fails to estimate probability quantiles correctly.

We can now summarize the main empirical contradictions with existing theoretical explanations of boosting, which motivates the view we present in this paper:

1. Boosting works well, perhaps best in terms of performance if not efficiency, with "strong learners" like C4.5 and CART.

2. The expected loss (as computed on a test sample) goes up as the number of iterations increases.

3. The optimization theory offers no explanation as to why the training error can be zero, yet the test error continues to descend.

4. Boosting overfits noisy data, but not nearly as much as a classifier that minimizes exponential loss.

This paper will squarely depart from the statistical optimization view by asserting that AdaBoost may be best thought of as a (self) smoothed, interpolating classifier. We will see

that unlike the statistical optimization view, this perspective suggests that for best performance once should run many iterations of AdaBoost with deep trees. This will allow us to draw a number of analogies between AdaBoost and random forests that echoes the similarity between these two algorithms that was posited in Breiman (2001). A key component to this argument will consist of explaining the success of interpolating classifiers in noisy environments. We will pursue this line of thought in the following section.

3 Interpolating Classifiers

Algorithm 2 Random Forests Hastie et al. (2009)

1. For $b = 1$ to B:

   (a) Draw a bootstrap sample $\mathbf{X}^*$ of size N from the training data

   (b) Grow a decision tree T_b to the data $\mathbf{X}^*$ by doing the following recursively until the minimum node size n_{min} is reached:

      i. Select m of the p variables

      ii. Pick the best variable/split-point from the m variables and partition

2. Output the ensemble $\{T_b\}_b^B$

Let $\hat{C}_b(\mathbf{x}^*)$ be predicted class of tree T_b. Then $\hat{C}_{rf}^B(\mathbf{x}^*) = \text{majority vote}\{\hat{C}_b(\mathbf{x}^*)\}_1^B$.

It is a widely held belief by statisticians that if classifier interpolates all the data, that is, it fits all the training data without error, then it cannot be consistent and should have a poor generalization error rate. In this section, we demonstrate that there are interpolating classifiers that defy this intuition: in particular, AdaBoost and random forests will serve as leading examples of such classifiers. We argue that these classifiers achieve good out of sample performance by maintaining a careful balance between the complexity required to perfectly match the training data and a general semi-smoothness property. We begin with a quick review of the random forests classifier, which will be in constant analogy with AdaBoost.

3.1 Random Forests

Random forests has gained tremendous popularity due to robust performance across a wide range of data sets. The algorithm is often capable of achieving best-in-class performance with respect to low generalization error and is not highly sensitive to choice of tuning parameters, making it the off-the-shelf tool of choice for many applications.

Algorithm 2 reviews the procedure for constructing a random forests model. Note that in many popular implementations, such as `R` implementation `randomForest` Liaw and Wiener (2002) built from Breiman's CART software, n_{min} is set to one for classification. This implies that each decision tree is designed to be grown to maximal depth and therefore necessarily interpolates the data in its bootstrap sample (assuming as least one continuous predictor). This results in each tree being a low bias but high variance estimator. Variance is then reduced by averaging across trees, resulting in a "smoothing" of the estimated response surface. The random predictor selection within each tree further reduces variance by lowering the correlation across the trees. The final random forest classifier still fits the entire training data set perfectly, at least with very high probability. To see this is true, consider any given training point. As the number of trees increases, with probability close to one, that point will be present in the majority of the bootstrap samples used to fit the trees in the forest. Thus the point will get the correct training set label when the votes are tabulated to determine the final class label.

We wish to emphasize that despite its success, random forests is not directly optimizing any loss function across the entire ensemble; each tree is grown independently of the other trees. While each tree may optimize a criteria such as the Gini index, the full ensemble is not constructed in any optimization-driven fashion such as is the case for AdaBoost. While there has been recent theoretical work describing the predictive surface of random forests (Wager and Walther, 2015), the analysis required unnatural assumptions that are hard to justify in practice (i.e. such as the growth rate of minimum leaf size). Rather, we postulate that the success of the algorithm is due to its interpolating nature plus the self-averaging mechanism. We next consider the implications of interpolating classifiers more broadly.

3.2 Local Robustness of Interpolating Classifiers

Let us begin with a definition of interpolation:

Definition: Let X_i be vector observations of predictor variables an let Y_i be the observed class label. A classifier $f(X)$ is said to be an *interpolating* classifier if for every training set example, the classifier assigns the correct class label; that is for every i, $f(X_i) = Y_i$.

Many statisticians are not comfortable with classifiers that interpolate the training data: common wisdom suggests that any classifier which fits the training data perfectly must have poor generalization error. Indeed, one of the first interpolating classifiers that might come to one's mind, the one-nearest neighbor, can be shown to be inconsistent and have poor generalization error in environments with noise. In fact, in the simple binary classification example where the true class label is reversed with probability p, it is easy to show that the asymptotic generalization error rate of the one-nearest neighbor classifier is $2p(1-p)$, which is higher than the performance achieved by the optimal Bayes classifier (which has a generalization error of p).

However, the claim that all interpolating classifiers overfit is problematic, especially in light of the demonstrated success of classifiers that perfectly fit the training data, such as random forests. This claim is even more dubious when one considers that any classifier f can be trivially modified to create a classifier f' with the same generalization error that interpolates the training data: let $f'(x_i) = y_i$ for x_i in the training set and $f' = f$ everywhere else. The point is that training data with continuous predictors has measure zero, and changing a classifier on a set of measure zero will not affect its generalization error.

One of our key insights reverses the common intuition about classifiers: interpolation can prevent overfitting. An interpolated classifier, if sufficiently local, minimizes the influence of noise points in other parts of the data. If the interpolations are averaged into a single ensemble, the result is a stable fit in regions of the data where there is signal. The ensemble of interpolating classifiers has been "spiked-smoothed". The basic goal of spiked-smoothing

is to create a decision surface which is essentially smooth, while allowing a small number of "spikes" in small neighborhoods around noise points. In classical statistics, one usually considers classes of functions which are smooth (in the common mathematical sense) everywhere. These are functions that are essentially locally constant - such as linear regression fits or polynomial smoothing splines, for example. Such fits, if interpolated through all the training data, will be unduly influenced by noise points, and will have terrible generalization error. Spiked-smoothing controls generalization error by averaging many interpolating classifiers so the resulting ensemble is mostly smooth but still interpolating. Each interpolating classifier in the ensemble fits the training data very locally which prevents noise points from affecting the fit of the classifier in regions far away from the point. An ensemble average of such classifiers is more stable and classically smooth in regions of the data where there is signal, with a spiked fit around noise points which continues is increasing localized as the number of classifiers and data increases. The result squeezed all the signal from the data without overfitting. In this section we present several conceptual and simulation-based results that illustrate this idea.

Interpolating classifiers that perform poorly, such as one-nearest neighbors, fail precisely because they are not sufficiently local: noise points pull the fit in the wrong direction over relatively large neighborhoods around the error, which increases the generalization error. On the other hand, smooth functions (in the continuous second-derivative sense) are too constrained to capture complex signal since they are not allowed to make rapid changes in any small neighborhoods. The averaging process, which is obvious for and random forests and we will show exists for AdaBoost when run for many iterations, "smooths" the fit in regions where there is signal, and more crucially *localizes* the fit around training set error points. This concept will be explored graphically in Section 3.4.

Let us turn now to a simple conceptual example to reinforce how interpolation can help create robustness in the presence of noise.

Suppose we are trying to predict a continuous response y based on real-valued x observations. The eleven points in Figure 1 make up the training data. The true model is $y = x + \epsilon$ for some (extremely) heavy tailed symmetric distribution for ϵ. In the training data only the y value at $x = .4$ contains a non-zero error ϵ. Thus, the two interpolation models given by

the blue and black lines only deviate from the target mean model ($y = x$) near this point. Moreover, the fit represented by the blue line is more spiked. It interpolates, but more locally than the fit represented by the black line. Consequently, the fit is less influenced by the single noisy observation. The spiked-smooth blue interpolator is more robust to label noise than the less local and less spiked-smooth black fit. In fact, as the width of the spike approaches zero, these interpolators converge to the target model $y = x$. We will show that this is analogous to the way boosting provides a robust fit by interpolating noise extremely locally. In contrast, the least squares regression fit in red has substantial deviation over the entire x distribution, caused by the single observation with large error. The (non-interpolating) regression fit fails because it allows the error to distort the fit everywhere, if only by a little bit. The linear model is powerful when it can borrow strength; but here it borrows weakness.

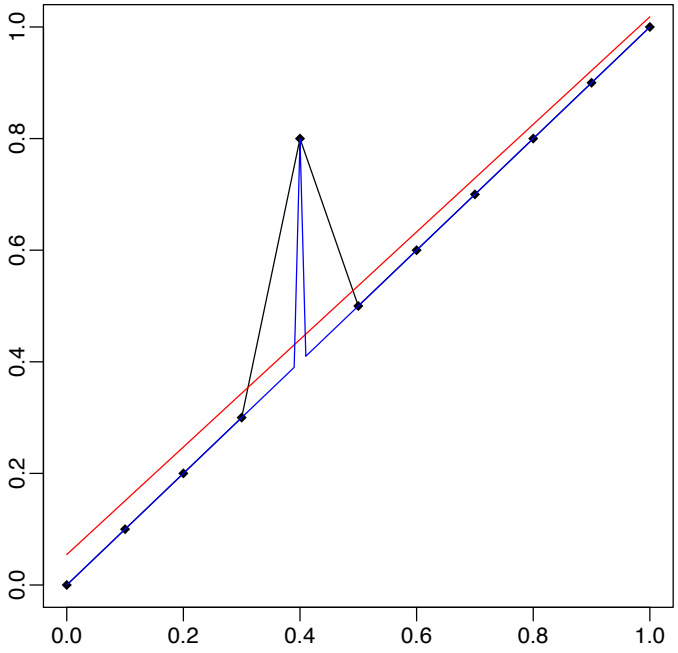


Figure 1: Three estimated regression functions with two interpolating fits (black and blue) and an ordinary least squares fit (red).

Returning to the classification setting, suppose we have have two predictors x_1 and x_2 distributed independently and uniformly on $[0, 1]^2$ and $y \in \{-1, +1\}$. Further suppose that the true conditional class probability function is

$$p(\mathbf{x}) = p(y = 1|\mathbf{x}) = p = .75$$

for all $\mathbf{x}$. This is a pure noise model with no signal, but in general one could view this as a subspace of a more complex model in which the $p(\mathbf{x})$ function is approximately constant. Since the Bayes decision rule is to classify every point as a "+1", we would desire an algorithm that will match the Bayes rule as close as possible. Again, we stress that this closeness should be judged with respect to the population or a hold-out sample. On this training data, any interpolating classifier will necessarily differ from the Bayes rule for $1-p = 25\%$ of the points on average.

Figure 2 shows a possible sample of training data from this model of size $n = 20$. The blue points represent the "+1"'s and the red points represent the "-1"'s. There are $5/20 = 25\%$ red points. The training data was sampled according to a Latin Hypercube design using the midpoints of the squares so that the points would be evenly spaced, but that is not essential.

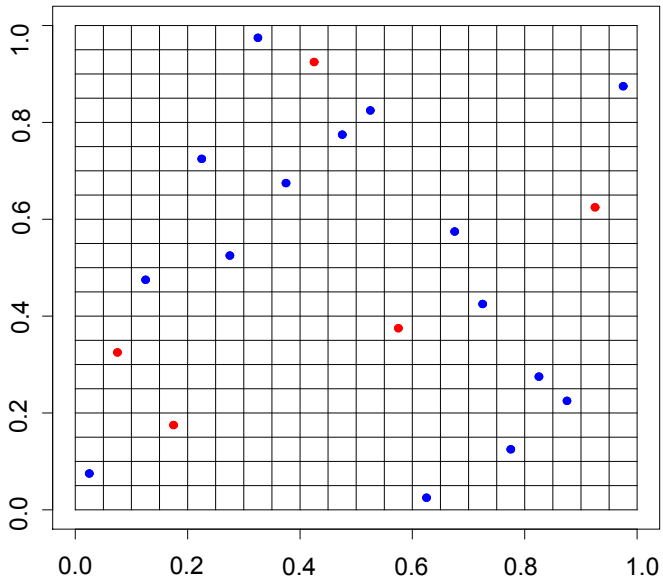


Figure 2: Training Data

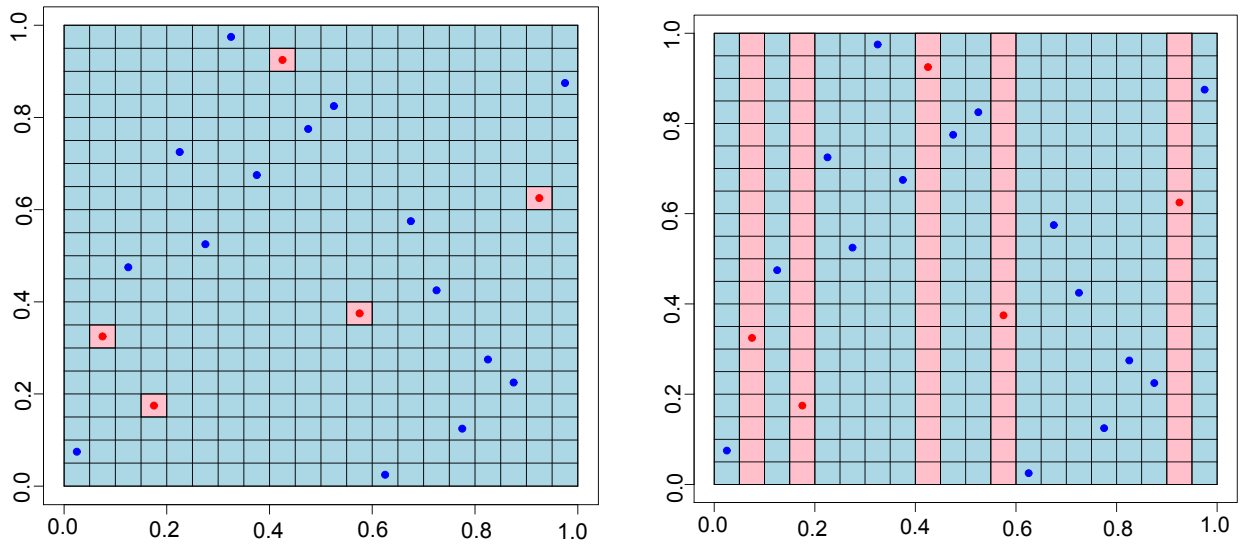
Figure 3 shows four hypothetical classifiers that could result from fitting boosted decision tree models to the training data in Figure 2. When decision trees are used as base learners for data with continuous predictors, it is a common convention to restrict the split points of the trees to be the midpoints of the predictors in the training data. Consequently, the classifier in each small square shown in Figure 2 will necessarily be constant throughout; this is the finest resolution of the classifier resulting from boosting decision tress assuming

no sub-sampling . Thus, Figure 3a represents the interpolating classifier closest to the Bayes rule. (In these plots, pink squares represent "-1"'s and light blue squares represent "+1"'s.) Note that the interpolation is in fact quite local; the estimated function varies rapidly in the small neighborhoods of the pink squares. For such a classification rule the percentage of points in a hold-out sample that would differ from the Bayes rule (in expectation over training sets) would be $(1-p)n/n^d$ where $p = p(Y = 1|\mathbf{x})$ and d is the dimensionality of the predictor space (for our example, $d = 2$). We will present evidence latter that in noisy environments boosting sufficiently large trees does actually tend to find such rules as that in Figure 3a. Interestingly, since

$$\lim_{n\to\infty}(1-p)n/n^d = 0$$

such rules are in fact consistent. This illuminates the point that interpolation does not rule out consistency. By allowing the decision boundary to be "mostly" smooth, with spikes of vanishing measure, it is possible to obtain consistency in the limit as $n \to \infty$, even while classifying every training point correctly. This stands in direct contrast to the conclusion of others such as Bickel et al. (2006) who have observed that the "empirical optimization problem necessarily led to rules which would classify every training set observation correctly and hence not approach the Bayes rule whatever be n."

While a classifier such as that in Figure 3a would preform well and is even consistent, many possible interpolators exist, such as the others displayed in Figure 3. Figure 3b shows the (hypothetical) result of allowing the boosting algorithm to use trees involving only x_1 and not x_2. The resulting classifier returns estimates that differ from the Bayes rule with rate $1-p$. Consequently, the one-nearest neighbor classifier has an error rate of $2p(1-p)$, which Jiang (2002) shows is the consequence of boosting in one dimension. It is interesting to note that this classifier has severely overfit, even though it is a simpler model, depending on only one of the two predictors. The classifier in Figure 3c has an even worse error rate, while the classifier in Figure 3d differs from the Bayes rule with rate $((1-p)n)^2/n^2$. This final example illustrates the type of structure and error rate that occurs when stumps are used as the weak learner. In fact, Mease and Wyner (2008) show that the additive nature of stumps results in boosted classifiers that differs from the Bayes rule at a rate of at least $(1-p)^d(1-1/d)^d$ and hence is not consistent. The reason for this is that using linear combinations of stumps does



(a) Hypothetical Classifier 1

(b) Hypothetical Classifier 2

(c) Hypothetical Classifier 3

(d) Hypothetical Classifier 4

Figure 3: Four different hypothetical classifiers on a pure noise response surface where $p(y = 1|\mathbf{x}) = 0.75$.

not provide enough flexibility to interpolate locally around points for which the observed class differs from the Bayes rule. In contrast, boosting larger trees, such as those grown in random forests interpolating with spikes of increasingly smaller size. Some simulations demonstrating the superior performance of larger trees over stumps are given in Mease and Wyner (2008) and here in Section 4.3.

The different classification rules represented by the four plots all interpolate the training data; however, their performances on the population vary considerably due to different degrees of local interpolations of noise. In the sequel, we will show how random forests and boosted ensembles of large trees results in classifiers that are robust to noise. The classifiers behave in noisy regions as in Figure 3a. AdaBoost and random forests average many individually overfit classifiers, similar to the one in Figure 3b. The result is a final robust classifier, that is spiked-smooth; it fits the noise but only extremely locally.

3.3 A Two-Dimensional Example with Pure Noise

We will begin with an easy to visualize example that demonstrates how fine interpolation can provide robustness in a noisy setting. In particular, we compare the performance of AdaBoost, random forests and one-nearest neighbors, which are all interpolating classifiers. We will see see graphically that AdaBoost and random forests are more locally interpolated around error points into the training data than the one-NN classifier. Consequently, AdaBoost and random forests are less affected by noise points as one-NN and have lower generalization error. We will show that the self-averaging property of AdaBoost and random forests is crucial. This property will be discussed in subsequent sections.

The implementation of AdaBoost is carried out according to the algorithm described earlier. The base learners used are trees fit by the `rpart` package (Therneau and Atkinson, 1997) in `R`. The trees are grown to a maximum depth of 8, meaning they may have at most $2^8 = 256$ terminal nodes. This will be the implementation of AdaBoost we will consider throughout the remainder of this paper.

We will consider again the "pure noise" model as described in the previous section, where the probability that y is equal to $+1$ for every $\mathbf{x}$ and some constant value $p > .5$. For the training data we will take $n = 400$ points uniformly sampled on $[0, 1]^2$ according to a Latin

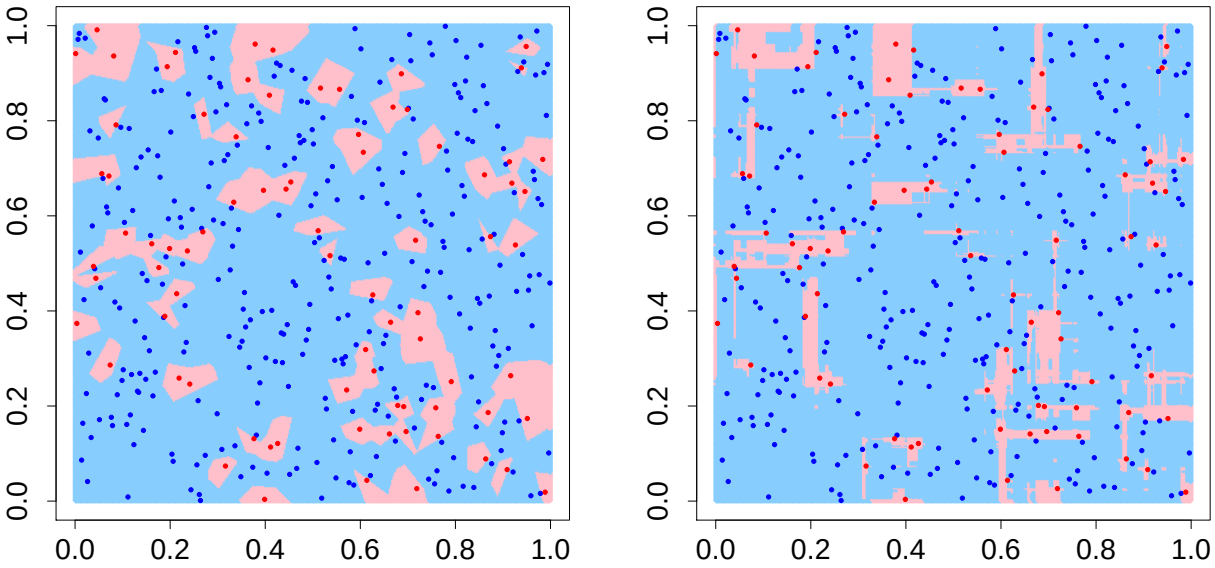
Hypercube using the midpoints as before. For the corresponding y values in training data we will randomly choose 80 points to be -1's so that $p(y = 1|\mathbf{x}) = .8$.

Figure 4 displays the results for the following: (a) one-NN, (b) AdaBoost , and (c) random forests. Regions classified as $+1$ are colored light blue and regions classified as -1 are colored pink. The training data is displayed with blue points for $y = +1$ and red points for $y = -1$. Since the Bayes' rule would be to classify every point as $+1$, we judge the performance of the classifiers by the fraction of the unit square that matches the Bayes' rule . The nearest neighbor rule in this example classifies 79% of the region as $+1$ (we expect $p = 80\%$ on average for the one-NN) while AdaBoost performs substantially better classifying 87% of the square as $+1$ after 100 iterations (which is long after the training error equals zero). This is evidence of boosting's robustness to noise discussed in the previous section. The random forests (with 500 trees) does even better, classifying 94% of the figure as $+1$. Visually, it is obvious that the random forests and AdaBoost classifier is more spiked-smooth than one-nearest neighbors, which allows it to be less sensitive to noise points. AdaBoost and random forests do in fact overfit the noise- but only the noise. They do not allow the overfit to metastasize to modestly larger neighborhoods around the errors. It is interesting to note that there seems to be a large degree of overlap between the regions classified as -1 by both the random forests and AdaBoost; one-NN does not seem to visually follow a similar pattern.

As we will see in the Section 3.6, by increasing the sample size, number of dimensions and iterations the performance is even better. The agreement with the Bayes rule for AdaBoost and random forests converge to practically 100% despite the fact that both algorithms still interpolate the training data without error.

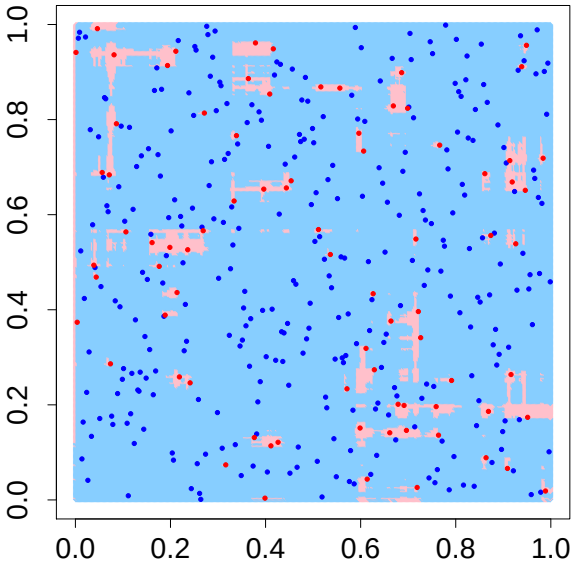
3.4 A Visualization of Spiked-Smoothing

We have argued that local interpolation such as in Figure 3c is desirable, and we have demonstrated that AdaBoost and random forest classifiers can achieve such a fit in the previous simulation. Now, we turn to the crucial point of how these classifiers achieve such a fit. To this end, we will graphically display in the process of spiked-smoothing in the case of the random forest classifier from the previous simulation. Each of the first six plots in



(a) one-NN

(b) Adaboost



(c) Random Forests

Figure 4: Performance of one-NN, Adaboost, and random forests on a pure noise response surface with $p(y = 1|\mathbf{x}) = .8$ and $n = 400$ training points.

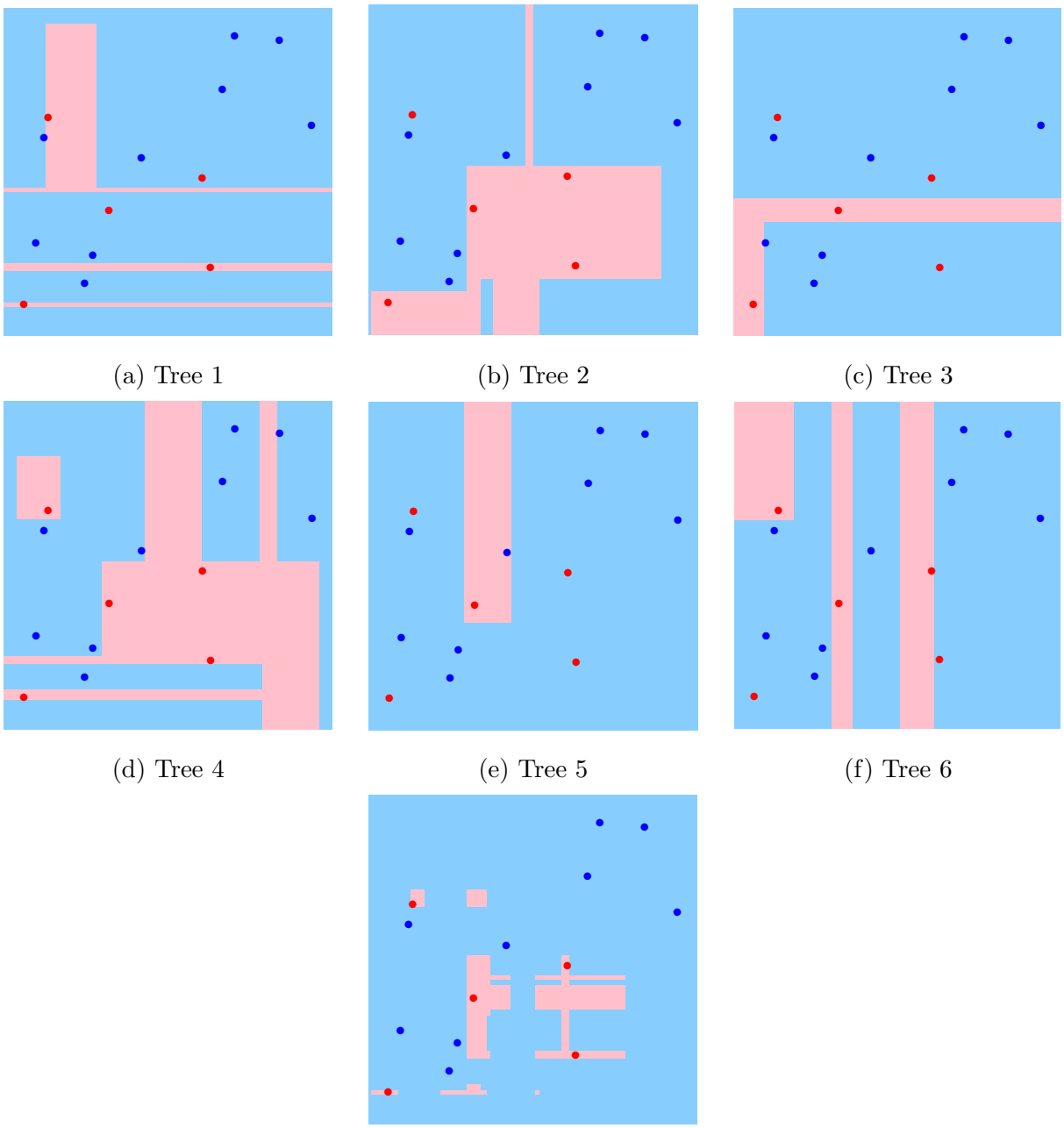
Figure 5 shows the classification rule fit by different decision trees in the random forest. We have restricted each plot to a subset of the unit square to aide in visual ease. The bottom plot, Figure 5g shows the classifier created from a majority vote of each of the six random forest decision trees. As in the previous sections, the light blue regions indicate where a classifier returns $y = +1$, and the pink regions indicate where a classifier returns $y = -1$.

As before, we remark that they Bayes rule in this case would be to classify every point as $y = +1$, and so agreement with the Bayes rule in the plots below can be visualized as the proportion of the figure that is light blue. The first thing to notice is that each decision tree fails to reproduce the Bayes rule. Indeed, since each tree interpolates its bootstrap sample, each figure is bound to contain regions of pink, since most bootstrap samples will contain at least a few noise points. However, one will also notice that these regions of pink tend to be localized into thin strips (this is especially apparent in trees one, three, five, and six). In other words, noise points tend not to ruin the fit of the decision tree at nearby points.. The magic of spiked-smoothing is revealed in the classifier 5g created by a majority vote of the six decision trees. By itself, each decision tree is a poor classifier (evinced by relatively large regions of pink). However, when voted these regions of pink get shrunk down into smaller regions, indicating better agreement with the Bayes rule. One can easily imagine that if these "thin strips" were actually much wider, as in the case of fitting stumps, averaging would not be able to reduce the influence of these noise points enough. The end effect of averaging is to create a decision surface which is affected only very minimally by the noise points in the training set. A simulation in Section 4 will demonstrate that the additional iterations of AdaBoost serve to "shrink" the fit around noise points, much as the regions of pink in this example became more localized after averaging.

3.5 A Two-Dimensional Example with Signal

In light of the example in the previous section, one might note that certain non-interpolating algorithms, such as a pruned CART tree, would recover the Bayes error rate exactly. In this section, we consider an example where a much more complex classifier is required to recover the signal, yet the self-averaging property is still needed to prevent over-fitting to noise.

We consider $n = 1000$ training points sampled uniformly on $[0, 1]^2$ with the Latin Hy-



(a) Tree 1 (b) Tree 2 (c) Tree 3

(d) Tree 4 (e) Tree 5 (f) Tree 6

(g) Majority Tree Vote

Figure 5: First six trees from a random forest, along with the classifier created by a majority vote over the trees.

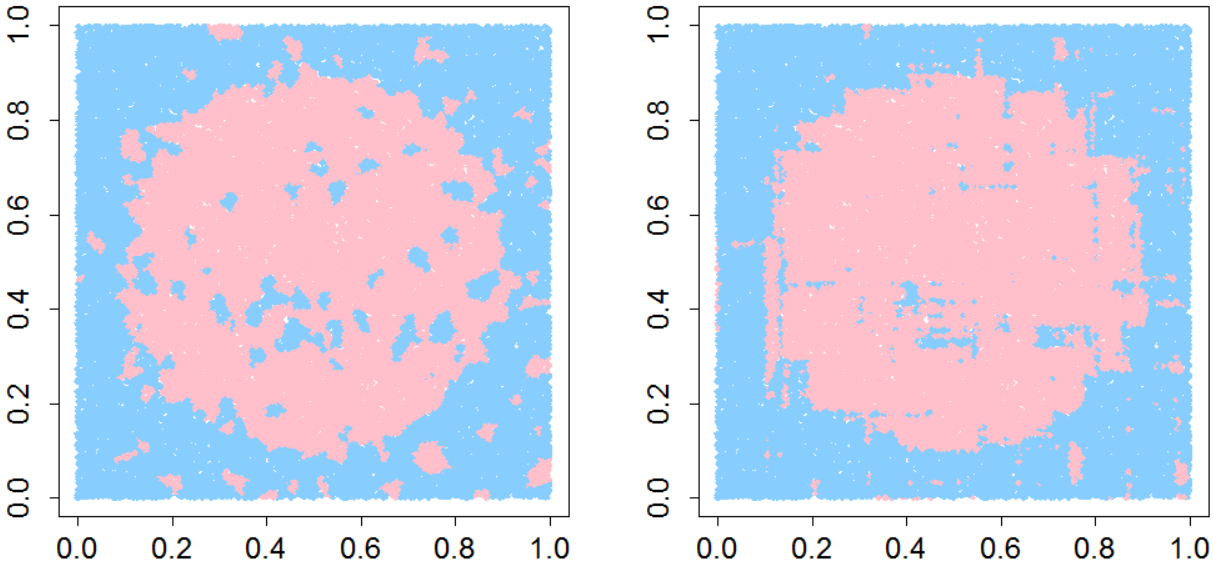
percube design. In this simulation, there is signal present. Inside of a circle of radius 0.4 centered in the square, the probability that $y = +1$ is set to 0.1, while the probability that $y = +1$ outside the circle is set to 0.9.

This simulation setting is similar to the previous one, except that the probability that $y = +1$ varies at different points over the unit square. One can see in Figure 6 that the Bayes rule in this setting is just to label every point inside the circle $y = +1$ and every point outside the circle $y = -1$, which gives a Bayes error rate of 0.1. We can then compare the performance of AdaBoost, random forests, and CART as in the previous section by examining how much of the circle gets classified as $y = +1$ and how much of the outer region is classified as $y = -1$. We run AdaBoost for 500 iterations, fit a random forests model with 500 trees, and build a CART tree that is pruned via cross-validation. Note that we prune the CART tree in order to show how a "classical" statistical model of limited complexity performs on the classification task.

We find that AdaBoost and Random Forests have an overall error rate of around 0.13, one-nearest neighbor has an overall error rate of 0.20, and CART has an error rate of 0.18. CART fails to perform well in this example because it is not allowed enough complexity to capture the circular pattern. To do so via only the splits parallel to the axes allowed by the algorithm would require a very deep tree (as allowed in random forests and AdaBoost), which pruning does not afford. Rather, a shallow tree can only recover a simple rectangular pattern due to its shallow depth. One-NN, on the other hand, again suffers from its inability to keep the interpolation localized. Outside of the circle, one can observe small "islands" of pink surrounding noise points: by failing to localize the fit, test points near these noise points get classified incorrectly. Again, one finds that random forests and AdaBoost have superior performance because they tend to finely interpolate the training data, and the process of spiked-smoothing shrinks down the influence of noise points that affected the

3.6 A Twenty-Dimensional Example

We now repeat the simulation in Section 3.3 with a larger sample size and in 20 dimensions instead of 2. Specifically, the training data now has $n = 5000$ observations sampled according to the midpoints of a Latin Hypercube design uniformly on $[0, 1]^{20}$. We again randomly select



(a) One-NN

(b) AdaBoost

(c) Random Forests

(d) CART

Figure 6: Performance of AdaBoost, random forests, and CART on a response surface where $p(y = 1|x) = 0.10$ inside the circle and $p(y = 1|\mathbf{x}) = 0.90$ outside of the circle. There are $n = 1000$ training points and the Bayes error is 0.10.

20% or 1000 of these points to be -1's with the remaining 4000 to be $+1$'s.

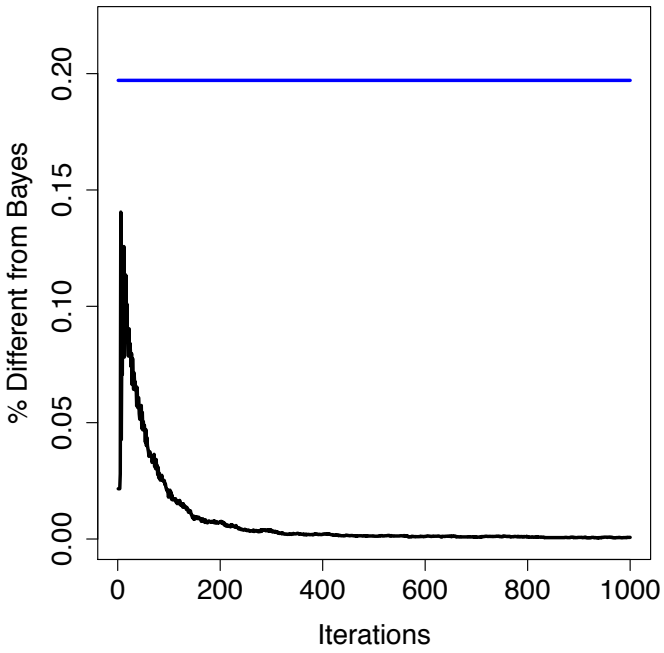


Figure 7: One-NN (blue) and AdaBoost (black) for 20% pure noise

Since in 20 dimensions it is difficult to display the resulting classification rules graphically we instead examine the rules on a hold out sample of 10,000 points sampled uniformly and independently on $[0, 1]^{20}$. Figure 7 plots the proportion of points in the hold out sampled classified by AdaBoost as $+1$ as a function of the number of iterations. This proportion peaks at .1433 at nine iterations but then gradually decreases to .0175 by 100 iterations and is equal to .0008 by 1,000 iterations. The fact that by 1,000 iterations only 8 of the 10,000 points in the hold out sample are classified as $+1$ one means there is very little overfitting. The large number of iterations has the effect of smoothing out the classifier resulting in a rule that agrees with the Bayes rule for 99.92% of the points. This is in spite of the fact that it interpolates the training data completely by 23 iterations and thus necessarily differs from the Bayes rule for exactly 20% of the training data from that point out. We see clearly here that AdaBoost overfits with respect to the training data but not with respect to the population. Again, this is a result of extremely local interpolation of the points in the training data for which the observed class differs from the Bayes rule. A random forests model fit to the training data agrees with the Bayes rule at every point except for one, and

hence has exceptional generalization error.

4 Self-Averaging Property of Boosting

4.1 Boosting is Self-Smoothing

In the previous sections, we have demonstrated simple examples where random forests and AdaBoost yield the strongest performance with respect to the Bayes rule. We have argued that these algorithms are successful classifiers due to the fact that they fit initially complex models by interpolating the training data but also exhibit smoothing properties via self-averaging that stabilizes the fit in regions with signal, while continuing to keep localized the effect of noise points on the overall fit. While this smoothing mechanism is obvious for random forests via the averaging over decision trees, it is less obvious for AdaBoost. In this section we explain why the additional iterations in boosting way beyond the point at which perfect classification of the training data (i.e interpolation) has occurred actually has the effect of smoothing out the effects of noise rather than leading to more and more overfitting. To the best of our knowledge, this is a novel perspective on the algorithm. To explain our key idea, we will recall the pure noise example from before with $p = .8$, $d = 20$ and $n = 5000$.

Recall that the classifier produced by AdaBoost corresponds to $\mathrm{I}[f_M(x) > 0]$ where

$$f_M(x) = \sum_{m=1}^{M} \alpha_m G_m(x)$$

as defined earlier. Taking $M = 1000$ which was successful in our example let us rewrite this as

$$f_{1000}(x) = \sum_{m=1}^{1000} \alpha_m G_m(x) = \sum_{j=1}^{10} \sum_{k=1}^{100} \alpha_{100(j-1)+k} G_{100(j-1)+k}(x) = \sum_{j=1}^{10} \sum_{k=1}^{100} h_k^j(x)$$

where

$$h_k^j(x) \equiv \alpha_{100(j-1)+k} G_{100(j-1)+k}(x).$$

Now define

$$h_K^j(x) \equiv \sum_{k=1}^{K} h_k^j(x)$$

and note that for every $j \in \{1, ..., 10\}$ and every $K \in \{1, ...100\}$ that $\mathrm{I}[h_K^j(x) > 0]$ is itself a classifier made by linear combinations of classification trees. The ten plots in Figure 8 display the performance on the hold-out sample for these ten classifiers corresponding to the ten different values for j as a function of K. Interestingly, each of these 10 classifiers by itself displays the characteristic of boosting the agreement with the Bayes rule increases as more terms are added (i.e. as K is increased).

A second interesting fact about these 10 individual classifiers in the decomposition is that each one achieves perfect separation of the training data and thus each one is an interpolating classifier. This result can be expected in general, provided the total number of iterations for each classifier in the decomposition is sufficiently large. This is clear for the first classifier, since it is simply AdaBoost itself and will necessarily achieve zero training error under some standard conditions as discussed in Jiang (2002). The second classifier in the decomposition is simply AdaBoost weight carried over from the first classifier. Since re-weighting the training data does not prevent AdaBoost from obtaining zero training error, the second classifier also interpolated eventually, as does the third, and so on.

Decomposing boosting in this way offers an explanation of why the additional iterations lead to robustness and better performance in noisy environments rather than severe overfitting. In this example, AdaBoost for 100 iterations is an interpolating classifier. It makes some errors, mostly near the points in the training data for which the label differs from the Bayes rule, although these are localized. Boosting for 1000 iterations is thus a point-wise weighted average of 10 interpolating classifiers. The random errors near the points in the training data for which the label differ from the Bayes' rule cancel out in the ensemble average and become even more localized. Of course, the final classifier is still an interpolating classifier as it is an average of 10 interpolating classifiers. In this way, boosting is self-smoothing, self-averaging or self-bagging process that reduces overfitting as the number of iterations increase. The additional iterations provide averaging and smoothing - not overfitting. Empirically this is very similar to random forests and provides evidence that both

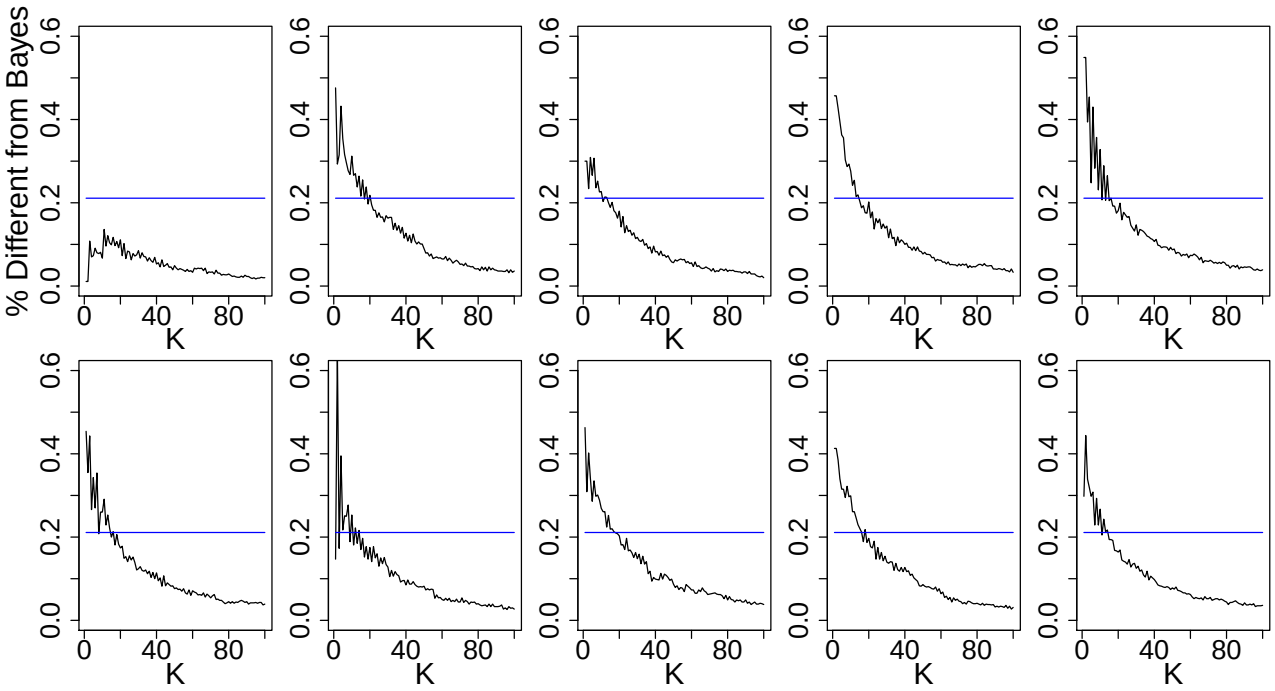


Figure 8: A decomposition of boosting

algorithms, which perform well in our examples, actually do so using the same mechanism.

We further illustrate this phenomenon of increasing localization of the interpolating resulting from this averaging through the following simulation. We take the same training data as before but this time we form the hold out sample by taking a point a (Euclidean) distance of .1 from each of the 1000 points labelled as -1 in the training data in a random direction. Due to the forced (and unnatural) close proximity of the points in the hold out to training set deviations from the Bayes' rule (points with -1 labels), the error rate is much higher than it would be for a random sample . However, the interpolation continues to become more localized as the iterations proceed (see Figure 9) so even points that are quite close to the label errors (the -1 points) eventually become classified correctly as $+1$. Comparison to Figure 7 shows that this localization continues at a steady rate even after the error on the random hold-out sample is practically zero. In contrast, the nearest neighbor interpolator this simulation yielded 100% disagreement with the Bayes' rule.

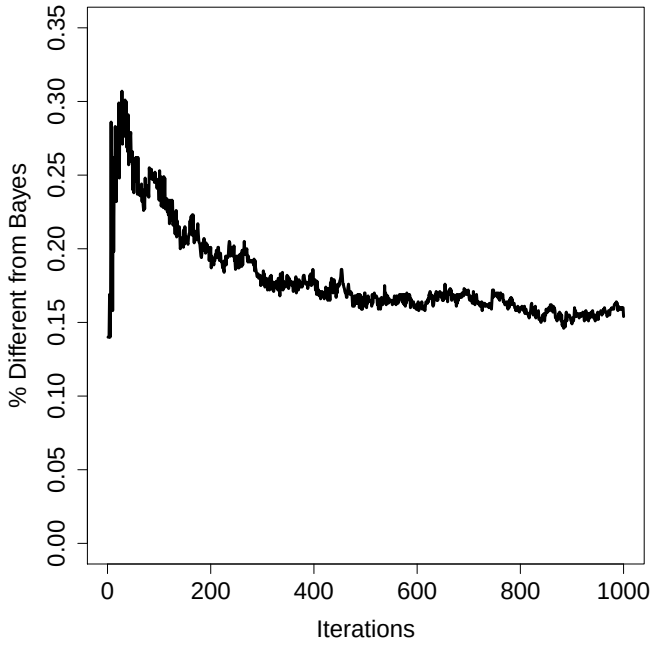


Figure 9: Disagreement with the Bayes rule for points close to noise points plotted using the same vertical scale as Figure 7.

4.2 A Five-Dimensional Example

We now consider a second simulation to further illustrate how this self-averaging property of AdaBoost helps prevent overfitting and improves performance. In this simulation we add signal while retaining significant random noise. Let $n = 400$, $d = 5$ and sample $\mathbf{x}_i$ distributed *iid* uniform on $[0, 1]^5$. The true model from for the simulation is

$$p(y = 1|\mathbf{x}) = .2 + .6 \ \mathrm{I}\left[\sum_{j=1}^{2} x_j > 1\right].$$

The Bayes' error is 0.20 and the optimal Bayes' decision boundary is the diagonal of the unit square in x_1 and x_2. Even with this small sample size, AdaBoost interpolates the training data after 10 iterations. So we boost for 100 iterations which decomposes into ten sets of ten (which is analogous to the 10 sets of 100 from the 20 dimensional example in the previous section).

The ten plots in the first two rows in Figure 10 show the performance of the ten classifiers corresponding to this decomposition with respect to a hold out sample of 1000 points. Each

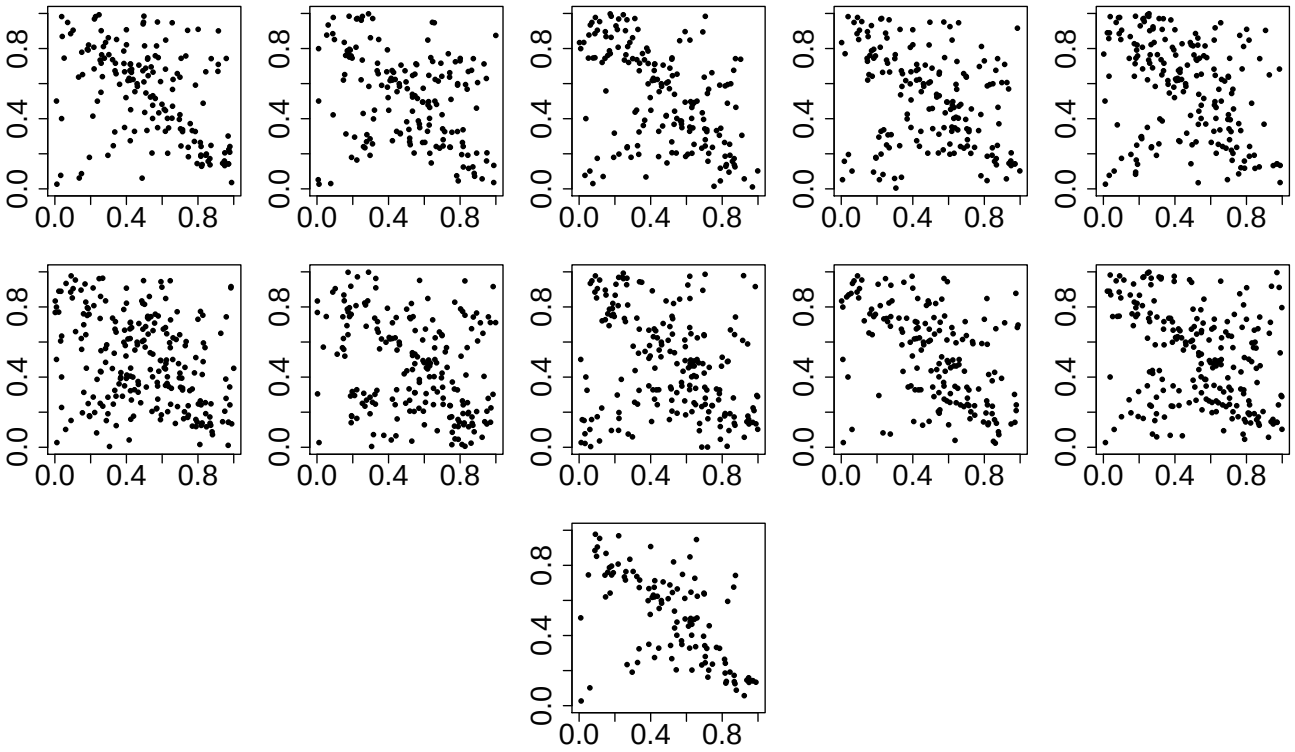


Figure 10: Errors made by each classifier in a decomposition of boosting (first two rows) and errors made by the final classifier (bottom)

point in the figure represents a point classified differently from the Bayes rule. While each of the ten classifiers in the decomposition classifies a number of these 1000 points incorrectly especially along the Bayes boundary, exactly which points are classified incorrectly varies considerably from one classifier to the next. The final classifier (displayed in the last plot), which corresponds to AdaBoost after 100 iterations, makes fewer mistakes than each of the ten individual classifiers. Since AdaBoost is a point-wise weighted average of the 10 classifiers, the averaging over the highly variable error locations made by each classifier reduces substantially the number of errors made by the ensemble. The percentage of points classified differently from the Bayes' rule by the final classifier is 118/1000=0.118 while after the first ten there were still 162/1000=0.162 classified differently from the Bayes rule. Averaged over 200 repetitions of this simulation these numbers are .19 after the first ten iterations and .15 after 100 iterations confirming that the performance does improve by running beyond the point at which interpolation initially occurs as (a result of this self-smoothing). For comparison, the nearest neighbor one (interpolating) classifier differs from the Bayes' rule at rate of .26 over the 200 repetitions of this simulation. Also note that since that every classifier in the decomposition, as well as the final ensemble classifier, are interpolating classifiers, the in-sample expected difference from the Bayes rule is necessarily equal to the Bayes error rate of 0.20. Again, we see that AdaBoost does much worse in-sample than out-of-sample when compared to the Bayes rule.

4.3 Comparison to Boosted Stumps

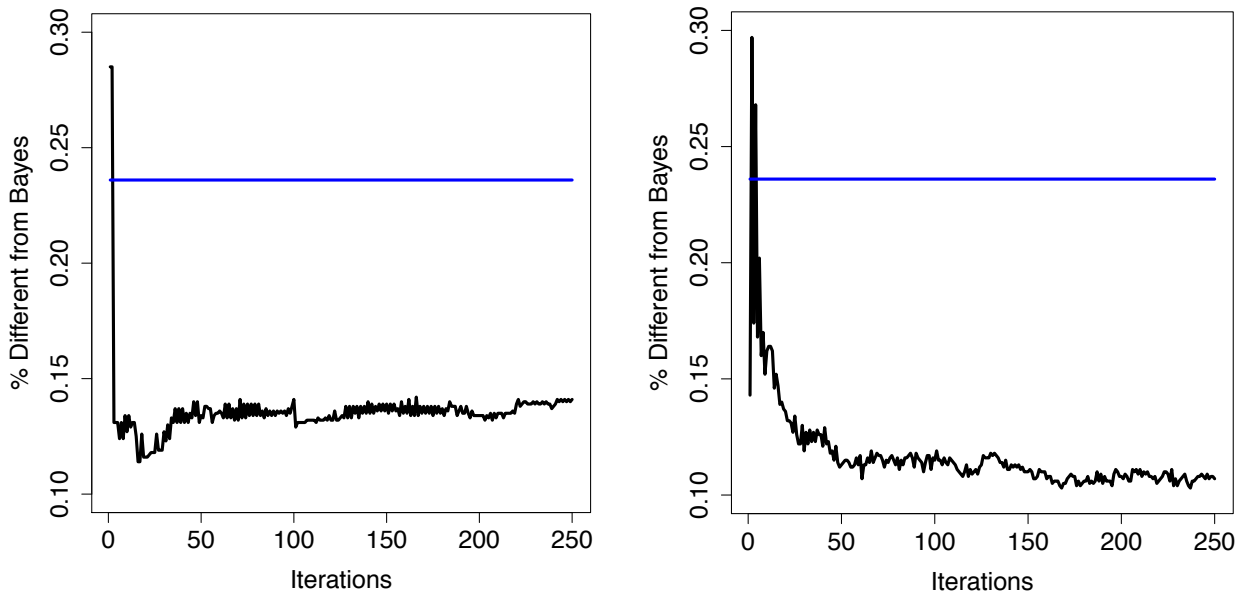
Throughout this paper we have considered AdaBoost with large trees of up to 2^8 terminal nodes. We have shown that AdaBoost with such large trees is a classifier which interpolates the data in such a way that it performs well out of sample for problems in which the Bayes' error rate is substantially larger than zero. In this section we will consider the performance of AdaBoost with stumps as the base learner. The statistical theory predicts that AdaBoost with stumps will overfit less than AdaBoost with larger trees, as expressed, for instance, in Jiang (2002). It is also thought that stumps should be preferable when the Bayes' decision rule is additive in the original predictors. For instance the seminal book by Hastie et al. (2009, chapter 10) advocates using trees of a depth one greater than the dominant level of

interaction, which is generally quite low.

AdaBoost with stumps does not self-smooth nearly as well as with larger trees, likely because the the classifiers in the decomposition are more highly correlated, and the "rough" fits from stumps fail to interpolate the training data locally enough; the fit is not spiked-smooth around the training set error points. Consequently, AdaBoost with stumps as base learners is outperformed by AdaBoost with large trees as base learners, when the Bayes error rate is high. This is the case even when the Bayes rule is additive. This result is matched by random forests which works best with large trees. Its randomly chosen predictors at each splitting opportunity lowers the correlation among the trees and the resulting fit is more spiked-smooth.

To illustrate this with an example, we return to the five dimensional simulation from Section 4.2 which has an additive Bayes decision rule. Figure 11 displays the percentage of points that are classified differently from the Bayes rule in the hold out sample of 1000, as a function of the number of iterations. It can be seen that the stumps (left panel) do not perform as well as the 2^8 node trees (right panel). After 250 iterations, AdaBoost with stumps yields 141/1000 points in the hold out sample classified differently from the Bayes rule, compared to 116/1000 for this same dataset using instead AdaBoost with 2^8 node trees. In fact, the stumps seems to suffer from overfitting when run beyond only 25 iterations, while the 2^8 trees do not have a problem with overfitting as the number of iterations are increased.

These numerical values are based only on a single run of this simulation, but the qualitative finding is reproducible over repeated runs. The result serves to illustrate that the good out-of-sample performance of AdaBoost using large trees resulting from the local interpolation of noise points is not shared by AdaBoost using stumps. The idea that stumps will perform better in noisy environments because they overfit less is not supported by this simulation. While the stumps do overfit less on the training data, as evidenced by the fact that they did not give zero training error, they actually overfit worse than the larger trees out of sample. Again, we attribute this to boosting with stumps lacking the self-smoothing property and not being flexible enough to interpolate the noise locally.



(a) Stump

(b) 2^8 Node Trees

Figure 11: Comparison of AdaBoost with stumps (a) and 2^8 node trees (b) for the five dimensional simulation. The blue line corresponds to One-NN.

5 Theory

We have demonstrated that AdaBoost may in fact be thought of as an average of interpolating classifiers, just as random forests is an average of interpolating classifiers. In this section we develop a mathematical theory to explain why weighted sums of uncorrelated classifiers should improve performance. Combined with the decomposition of AdaBoost in the previous section, this will give a new view on why AdaBoost performs so well in practice.

Consider a fixed point x_0 which is not in the training set for which the corresponding class is equal to "+1" with some probability greater than 1/2. The Bayes' rule for this point will be to classify as a "+1". As we have explained, the AdaBoost rule for classifying test set point x_0 can be expressed as a weighted average of classifiers. The random forests ensemble can be similarly expressed. Let $\{Y_i\}_{i=1}^n$ denote the labels produced by each of the these classifier when evaluated at x_0. Let the corresponding weights be $\vec{z} = \{z_1, z_2, \ldots, z_n\}$. While the classifications for x_0 given by $\{Y_i\}_{i=1}^n$ are not necessarily $i.i.d$, we would expect that they are reasonable uncorrelated conditional on the training set, with roughly comparable error

rates, which would allow an appeal to the following general result.

Suppose we are given an n dimensional vector $\vec{z}$ of positive integers $z_i \geq 0$ with $\vec{z} = \{z_1, z_2, \ldots, z_n\}$ satisfying the constraint that

$$\sum_{i=1}^{n} z_i = M$$

where M is a positive, odd integer less than or equal to n. Now assume that we are given a sequence $\{Y_i\}_{i=1}^{n}$ of i.i.d. Bernoulli(p) random variables with

$$\Pr\{Y_i = 1\} = 1 - \Pr\{Y_i = 0\} = p > \frac{1}{2}.$$

The Y_i represent the class labels assigned by each classifier. The following result shows that averaging *any* weighted ensemble can only increase the probability of correct classification.

Theorem A: Let $\vec{z}$ be any vector of integers satisfying the above constraints. Then for any $p > 1/2$ it follows that

$$\Pr\{\sum_{i=1}^{n} z_i Y_i > M/2\} \geq p. \tag{1}$$

Comment: It follows from this result that weighted ensemble can only reduce misclassification error. This provides theoretical justification for our observation that increasing the number of iterations reduces test set error rates.

Proof: Let $f(\vec{z}) = \Pr\{\sum_{i=1}^{n} z_i Y_i > M/2\}$. If f were Schur-convex then our result would be immediate. Unfortunately, f is *not* Schur convex.

Our proof proceeds by considering an elementary case first. To this end, consider the degenerate vector $\vec{z_0} = \{M, 0, 0, \ldots, 0\}$. Trivially,

$$\Pr\{\sum_{i=1}^{n} z_i Y_i = z_o Y_0 > M/2\} = p.$$

The argument, which we will formalize mathematically, is that z_0 is the worst case: any additional variation in $\vec{z}$ leads to strict inequality in Equation 1.

Now let us assume that α and β are positive integers with $\alpha + \beta \leq M$. Define $\vec{z} = \{\alpha + \beta, 0, z_3, z_4, \ldots, ..z_n\}$. Now let's perturb $\vec{z}$ be moving β units of mass from the first coordinate to the empty second coordinate, leaving all the other coordinates constant, to create $\vec{z'}$ with

$$\vec{z'} = \{\alpha, \beta, z_3, z_4, \ldots, z_n\}.$$

Let

$$A = \{y \in \{0,1\}^n : \sum_{i=1}^{n} z_i y_i > M/2\};$$

thus A corresponds to the set of n-dimensional binary sequences whose total "weight" exceeds $M/2$ with respect to z. Now let

$$A' = \{y \in \{0,1\}^n : \sum_{i=1}^{n} z'_i y_i > M/2\}$$

be the set of sequences whose weight is greater than $M/2$ with respect to z'. The key fact in the proof, which is interesting in its own right, is the following:

$$\Pr\{A'\} \geq \Pr\{A\}. \tag{2}$$

Assuming the inequality in Equation 2, we can easily prove the theorem: let z be any vector. Let k be the number of non-zero entries. Starting with z_0 it is possible to make k stepwise mass shifts to form z through intermediate sequences $z_1, z_2, ..z_k, z$. Let $A(z_1), A(z_2), .., A(z)$ be the corresponding sets of y whose weight is greater than $M/2$. Equation 2 implies that $\Pr\{A(z)\} \geq \Pr\{A(z_{k-1})\} \ldots, \Pr\{A(z_1)\} \geq p$ which is what is the claim in Theorem A.

Now to prove the inequality in Equation 2, we consider two subsets of binary n-vectors:

$$\begin{aligned} B_1 &= \{y \in \{0,1\}^n : y \in A' \cap A^c\} \\ B_2 &= \{y \in \{0,1\}^n : y \in A \cap A'^c\} \end{aligned}$$

Now any sequence y in B_1 must have $y_2 = 1$ and $y_1 = 0$.

To see this, recall that A has $z_2 = 0$ and A' has $z_2 = \beta > 0$. Thus if $y_2 = 0$ then it

must be that if $y \in A'$ then y must also be in A, so y cannot be in B_1. This shows that $y_2 = 1$. On the other hand, if $y_1 = 1$ as well as $y_2 = 1$, then for any $y \in A$ we would also have $y \in A'$, so y cannot be in B_1 and thus for any $y \in B_1$, it must be that $y_1 = 0$.

Similarly, if $y \in B_2$ then $y_1 = 1$ and $y_2 = 0$. To see this, assume that $y_1 = 0$: then $y \in A$ would imply that $y \in A'$ and hence $y \notin B_2$ so it must be that $y_1 = 1$. Similarly, if $y_2 = 1 = y_1$ then we would have $y \in A$ is also in A' and thus not in B_2 so $y_2 = 0$. Summarizing, we have

$$\{\forall y \in B_1, y_1 = 0,\ y_2 = 1\} \tag{3}$$

and

$$\{\forall y \in B_2, y_1 = 1,\ y_2 = 0\}. \tag{4}$$

The consequence of this fact is that

$$P(A') - P(A) = P(B_1) - P(B_2).$$

A final combinatorial fact is that for every $y \in \{0, 1\}$ if $\sum z_i y_i > M/2$ then $\sum z_i(1 - y_i) < M/2$. This implies that

$$|A| = |A^c|$$

and

$$|A'| = |A'^c|.$$

This in turn implies that

$$|B_1| = |B_2|.$$

Since we have already established that for any y in either B_1 or B_2 it must be that $y_1 + y_2 = 1$ it follows that the probability of B_1 and B_2 respectively is completely determined only by the values of $\{y_3, y_4, \ldots, y_n\}$. Furthermore, since $|B_1| = |B_2|$ it follows that

$$P(B_1) \geq P(B_2) \text{ if } E[Y_i|Y \in B_1] \geq E[Y_i|Y \in B_2],$$

for $i = 3, 4, .., n$. The latter is true only if

$$\sum_{y \in B_1} \sum_{i=3}^{n} y_i \geq \sum_{y \in B_2} \sum_{i=3}^{n} y_i. \tag{5}$$

This is true because y_i are i.i.d. Bernoulli with $Pr\{y_i = 1\} = p > 1/2$. Thus, for any two symmetric divisions of $\{0,1\}^n$ (such as B_1 and B_2), the more probable set is the set with a larger proportion of 1's.

We prove the inequality in Equation 5 by observing that if $y \in B_1$ then two fact emerge from the fact that $y_1 = 0$ and $y_2 = 1$ and the definitions A and A':

$$\beta + \sum_{i=1}^{n} z_i y_i > M/2$$

and

$$\sum_{i=1}^{n} z_i y_i < M/2.$$

Furthermore, for $y \in B_2$, we know from the fact that $y_1 = 1$ and $y_2 = 0$ that

$$\alpha + \beta + \sum_{i=1}^{n} z_i y_i > M/2$$

and

$$\alpha + \sum_{i=1}^{n} z_i y_i < M/2.$$

Thus we have for $y \in B_1$ that

$$M/2 - \beta < \sum_{i=1}^{n} z_i y_i < M/2$$

and for $y \in B_2$ that

$$M/2 - \alpha - \beta < \sum_{i=1}^{n} z_i y_i < M/2 - \alpha.$$

Since the weights z_i are fixed over B_1 and B_2 and since $\alpha \geq \beta$ we have proved the inequality in Equation 5. This in turn implies that

$$P(A') - P(A) \geq 0$$

which proves the theorem.

6 Concluding Remarks

AdaBoost is an undeniably successful algorithm and random forests is at least as good if not better. But AdaBoost is as puzzling as it is successful; it broke the basic rules of statistics by iteratively fitting even noisy datasets until every training set data point was fit without error. Even more puzzling, to statisticians at least, it will continue to iterate an already perfectly fit algorithm which lowers generalization error. The statistical view of boosting understands AdaBoost to be a stage wise optimization of an exponential loss, which suggest (demands!) regularization of tree size and control on the number of iterations. In contrast, a random forest is not an optimization; it appears to work best with large trees and as many iterations as possible. It is widely believed that AdaBoost is effective because it is an optimization, while random forests works—well because it works. Breiman conjectured that "it is my belief that in its later stages AdaBoost is emulating a random forest" (Breiman, 2001). This paper sheds some light on this conjecture by providing a novel intuition supported by simulated examples and some modest theory which show how AdaBoost and random forest are successful for the same reason.

A random forests model is a weighted ensemble of interpolating classifiers by construction. Although it is much less evident, we have shown that AdaBoost is also a weighted ensemble of interpolating classifiers. Viewed in this way, AdaBoost is actually a "random" forest of forests. The trees in random forests and the forests in the AdaBoost each interpolate the data without error. As the number of iterations increase the averaging of decision surface because smooths but nevertheless still interpolates. This is accomplished by whittling down the decision boundary around error point We hope to have cast doubt on the commonly held belief that the later iterations of AdaBoost only served to overfit the data. Instead, we argue that these later iterations lead to an "averaging effect", which causes AdaBoost to behave similarly to random forests.

A central part of our discussion also focused on the merits of interpolation of the training data, when coupled with averaging. Again, we hope to dispel the commonly held belief that

interpolation always leads to overfitting. We have argued instead that fitting the training data in extremely local neighborhoods actually serves to prevent overfitting in the presence of averaging. The local fits serve to prevent noise points from having undue influence over the fit in other areas. Random forests and AdaBoost both achieve this desirable level of local interpolation by fitting deep trees. It is our hope that our emphasis on the "self-averaging" and interpolating aspects of AdaBoost will lead to a broader discussion of this classifier's success that extends beyond the more traditional emphasis on margins and exponential loss minimization.

References

Peter J. Bickel, Ya'acov Ritov, and Alon Zakai. Some theory for generalized boosting algorithms. *The Journal of Machine Learning Research*, 7:705–732, 2006.

Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

Leo Breiman. Prediction games and arcing classifiers. Technical report, Technical Report 504, Statistics Department, University of California at Berkeley, 1997.

Leo Breiman. Arcing classifier (with discussion and a rejoinder by the author). *The Annals of Statistics*, 26(3):801–849, 1998.

Leo Breiman. Random forests. *Machine Learning*, 45:5–32, 2001.

Peter Bühlmann and Bin Yu. Boosting with the l 2 loss: regression and classification. *Journal of the American Statistical Association*, 98(462):324–339, 2003.

Corinna Cortes, Mehryar Mohri, and Umar Syed. Deep boosting. In *Proceedings of the Thirty-First International Conference on Machine Learning (ICML 2014)*, 2014.

Robert M. Freund, Paul Grigas, and Rahul Mazumder. Adaboost and forward stagewise regression are first-order convex optimization methods. *arXiv preprint arXiv:1307.1192*, 2013.

Yoav Freund. *Boosting*. MIT Press, 2012.

Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *ICML*, volume 96, pages 148–156, 1996.

Jerome Friedman. Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, pages 1189–1232, 2001.

Jerome Friedman, Trevor Hastie, Robert Tibshirani, et al. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The Annals of Statistics*, 28(2):337–407, 2000.

Carlos Guestrin. Pac-learning, vc dimension and margin-based bounds. *Machine Learning*, 10701:15781, 2006.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*, volume 2. Springer, 2009.

Wenxin Jiang. On weak base hypotheses and their implications for boosting regression and classification. *Annals of Statistics*, pages 51–73, 2002.

Andy Liaw and Matthew Wiener. Classification and regression by randomforest. *R news*, 2 (3):18–22, 2002.

Senén Barro Manuel Fernández-Delgado, Eva Cernadas and Dinani Amorim. Do we need hundreds of classifiers to solve real world classification problems? *The Journal of Machine Learning Research*, 15(1):3133–3181, 2014.

Andreas Mayr, Harald Binder, Olaf Gefeller, and Matthias Schmid. The evolution of boosting algorithms. *Methods of Information in Medicine*, 53(6):419–427, 2014.

David Mease and Abraham Wyner. Evidence contrary to the statistical view of boosting. *The Journal of Machine Learning Research*, 9:131–156, 2008.

Indraneel Mukherjee, Cynthia Rudin, and Robert E Schapire. The rate of convergence of adaboost. *The Journal of Machine Learning Research*, 14(1):2315–2347, 2013.

Greg Ridgeway. Generalized boosted regression models. *Documentation on the R Package gbm, version 1· 5*, 7, 2006.

Robert E. Schapire. Explaining adaboost. In *Empirical Inference*, pages 37–52. Springer, 2013.

Robert E. Schapire, Yoav Freund, Peter Bartlett, and Wee Sun Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *Annals of statistics*, pages 1651–1686, 1998.

Terry M. Therneau and Elizabeth J. Atkinson. An introduction to recursive partitioning using the rpart routines. Technical report, Technical Report 61. URL http://www. mayo. edu/hsr/techrpt/61. pdf, 1997.

Stefan Wager and Guenther Walther. Uniform convergence of random forests via adaptive concentration. *arXiv preprint arXiv:1503.06388*, 2015.

Liwei Wang, Masashi Sugiyama, Zhaoxiang Jing, Cheng Yang, Zhi-Hua Zhou, and Jufu Feng. A refined margin analysis for boosting algorithms via equilibrium margin. *The Journal of Machine Learning Research*, 12:1835–1863, 2011.

Abraham J. Wyner. On boosting and the exponential loss. In *Proceedings of the Ninth Annual Conference on AI and Statistics Jan*, pages 3–6. Citeseer, 2003.