ESSAYS ON RANDOM FOREST ENSEMBLES

Matthew Olson

A DISSERTATION

in

Statistics

For the Graduate Group in Managerial Science and Applied Economics

Presented to the Faculties of the University of Pennsylvania

in

Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

2018

Supervisor of Dissertation

Abraham J. Wyner, Professor of Statistics

Graduate Group Chairperson

Catherine M. Schrand, Celia Z. Moh Professor of Accounting

Dissertation Committee

Richard Berk, Professor of Criminology and Statistics

Andreas Buja, Professor of Statistics

ESSAYS ON RANDOM FOREST ENSEMBLES

To my parents.

ACKNOWLEDGEMENT

I thank first my parents - Joan and Mark - to whom I dedicate this dissertation. Your sacrifices and selflessness as parents make a PhD look easy in comparison.

I thank next the Wharton Statistics Department faculty and staff, especially my committee members. Adi, your challenges to conventional views of machine learning shaped a large portion of this thesis. Richard and Andreas, I am grateful for your open doors and insightful conversations. I also thank the faculty and staff for fostering a warm, friendly, environment, which made my PhD studies so enjoyable.

Finally, I thank the wonderful friends and colleagues I met during my time at Penn. I will have fond memories of weekly bowling, as well as long lunches and dinners in the department. Colman and Sameer, I could not have asked for a better pair of friends to get through five years of graduate school together. Jon, I am grateful for our conversations and collaborations while we were roommates - you never fail to teach me something new. Jen, hiking, tennis, and nights on the couch with Felix provided a welcome respite from research.

ABSTRACT

ESSAYS ON RANDOM FOREST ENSEMBLES

Matthew Olson

Abraham J. Wyner

A random forest is a popular machine learning ensemble method that has proven successful in solving a wide range of classification problems. While other successful classifiers, such as boosting algorithms or neural networks, admit natural interpretations as maximum likelihood, a suitable statistical interpretation is much more elusive for a random forest. In the first part of this thesis, we demonstrate that a random forest is a fruitful framework in which to study AdaBoost and deep neural networks. We explore the concept and utility of *interpolation*, the ability of a classifier to perfectly fit its training data. In the second part of this thesis, we place a random forest on more sound statistical footing by framing it as kernel regression with the proximity kernel. We then analyze the parameters that control the bandwidth of this kernel and discuss useful generalizations.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF ILLUSTRATIONS

CHAPTER 1 : Introduction

In his 2001 paper *Statistical Modeling: The Two Cultures* Leo Breiman lamented that the statistics community had become overly reliant on stochastic data models, which had led to "irrelevant theory" and "questionable conclusions" (Breiman et al., 2001). At the same time, an emerging class of data analysts - the machine learners - were applying black-box, algorithmic, models with great success. Among these models were AdaBoost and random forests, ensembles of thousands of decision trees with no clear connection to classical statistics. As new, more successful, algorithmic tools were supplanting the model-based tools of linear regression, statisticians stood on the sidelines.

The situation has changed dramatically since then, as evidenced by the advent of *statistical* machine learning (Hastie et al., 2009). The key bridge between the "black-boxes" of machine learning and stochastic data models was optimization: optimization provided the framework for statisticians to *interpret* these algorithms in the language of maximum likelihood. A canonical example of this bridge was the framing of AdaBoost as a stage-wise approach to additive logistic regression (Friedman et al., 2000). If ensemble of trees algorithms were the black boxes of data analysis in the early 2000s, today's black boxes are deep neural networks, with millions of parameters. Yet, even these models can be *interpreted* in terms of binomial deviance and regularization, language familiar to the statistician.

An interpretation is useful to the extent that it allows others to think fruitfully about an idea, and leads to meaningful generalizations (Buja, 2000). One finds this to be the case with the framing of AdaBoost found in Friedman et al. (2000). Subsequent work led to gradient boosting, an algorithmically explicit method utilizing trees and gradi-

ent descent to maximize a likelihood (Friedman, 2001). Similarly, the training process for deep neural networks has successfully borrowed techniques of regularization that have deep foundations in penalized likelihood estimation.

However, there is no true interpretation of anything (Buja, 2000), and it is often counterproductive to slavishly follow any interpretation to its logical conclusion. Again, one finds this to be the case with the framing of AdaBoost as maximum likelihood. Variants of boosting algorithms that use more aggressive optimization tend to perform worse than the original formulation (Schapire and Freund, 2012). Furthermore, natural extensions of the likelihood interpretation, such as probability estimation, can perform poorly in practice (Mease and Wyner, 2008). In a similar way, accelerated gradient methods used in training deep neural networks can converge to local optima with worse generalization than unmodified stochastic gradient descent (Wilson et al., 2017).

The unifying thread of this thesis is the random forest classification algorithm, an ensemble method that defies the optimization view of machine learning. The mathematics of a random forest are far less elegant than AdaBoost: there is neither a global loss function nor an interpretation of gradient descent. Although a random forest greedily minimizes an impurity criteria at each node, this minimization is done imprecisely as variables are selected at random. Furthermore, additional randomization - and thus less optimization - also leads to good performance (Geurts et al., 2006). We use random forests as an umbrella for studying the performance of successful machine learning methods, specifically AdaBoost and deep neural networks. We also suggest that although a random forest does not fit cleanly into the familiar likelihood framework, we can still frame it in statistically familiar terms as kernel regression.

In the first part of the thesis, we argue that AdaBoost and neural networks produce fits that are qualitatively similar to a random forest. In particular, we focus on the property of all three classifiers to generalize well despite achieving perfect training accuracy. In each case, we provide decompositions of the final algorithm into ensembles low bias, decorrelated classifiers. Although obvious in the case of random forests, these decompositions are more subtle in the case of AdaBoost and neural networks. This view also supports the absence of regularization and early stopping, which are strongly suggested by the optimization perspective.

The second part of this thesis offers a new interpretation of a random forest as kernel regression, which places it squarely back in the statistician's domain. This interpretation was inspired from the analysis of random forest probability estimates, which are simply the fraction of trees in a forest that vote for a certain class. Apriori, an ensemble vote has nothing to do with a probability. We show that in the case of random forest, this fraction amounts to kernel regression with the proximity function. We then analyze the parameters that control the bandwidth of this kernel, as well as construct useful generalizations of the proximity function.

CHAPTER 2 : Interpolating Classifiers

Abstract

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. In the process, we question the conventional wisdom that suggests 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, without regularization or early stopping.

2.1. 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 data set. 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

*Joint work with Adi Wyner, Justin Bleich, and David Mease.

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 do not 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 (for instance, CART, neural networks, logistic regression) (Breiman, 1998) and substantively better than the technique of creating an ensemble using the bootstrap (Breiman, 1996a). 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 (Fernández-Delgado et al., 2014). Unlike AdaBoost, for which there are multiple accepted explanations, random forest's 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 these mechanisms by design, it is less clear that this is true for AdaBoost.

It is worth noting that Breiman noticed the connection between random forests and AdaBoost as well, although his notion of a random forest was more general, including other types of large ensembles of randomly grown trees (Breiman, 2001). In his 2001 *Random Forests* paper, he conjectured that the weights of AdaBoost might behave like an ergodic dynamic system, converging to an invariant distribution. When run for a long time, the additional rounds of AdaBoost were equivalent to drawing trees randomly grown according to this distribution, much like a random forest. Recent work has followed up on this idea, proving that the weights assigned by AdaBoost do indeed converge to a invariant distribution [2] (Belanich and Ortiz, 2012). In this work, the authors also show that functions of these weights, such as the generalization error and margins, also converge. This work certainly complements ours, but we focus on the similarity between AdaBoost and random forests through the lens of the type of decision surfaces both classifiers produce, and ability of both algorithms to achieve zero error on the training set.

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

[2] More specifically, they consider the so called "Optimal AdaBoost" algorithm, which is assume to pick the base classifier with lowest weighted error at each round. They show that the per round average of any measurable function of the training weights converges under mild conditions.

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 will not 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 2.4 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 begin in Section 2.2 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 2.3, 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 2.4, 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. We will then demonstrate this intuition in a real data example in Section 2.5. Finally, we conclude with a brief discussion in Section 2.6.

2.2. 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 Schapire and 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 data set reweighted by some weighting vector $\mathbf{w}_m$. We then calculate the weighted misclassification rate of our chosen learner, and update the weighting measure used in the next round, $\mathbf{w}_{m+1}$. The final classifier is output as the sign of a weighted linear combination of classifiers produced from each stage of the algorithm. In practice, one sometimes limits the number of rounds of boosting as a form of regularization. We will discuss this point more in the next section, and challenge its usefulness in later parts of the paper.

2.2.1. 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

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 $\text{err}_m = \dfrac{\sum_{i=1}^{N} w_i I\left(y_i \neq G_t\left(\mathbf{x}_i\right)\right)}{\sum_{i=1}^{N} w_i}$.
   (c) Compute $a_m = \log\left(\dfrac{1 - \text{err}_t}{\text{err}_t}\right)$.
   (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 $f(\mathbf{x}) = \text{sign}\left(f_M(\mathbf{x})\right)$

does not overfit in most cases. One of the first attempts to resolve this paradox was explored by Schapire et al. (1998), who focused on the *margins* of AdaBoost. The margins can be thought of as a measure of how confident a classifier is about how it labels each point, and one would hypothetically desire to produce a classifier with margins as large 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 reformulated 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).

One could take these observations to suggest that a more effective algorithm might be designed to explicitly optimize margins. However, one can find evidence against this hypothesis in Breiman's *arc-gv* algorithm (Breiman, 1999). Breiman designed the *arc-gv* algorithm to maximize the minimum margin in a data set, and he found that

this algorithm actually had worse generalization error than AdaBoost. Moreover, he developed generalization error bounds based on the minimum margin of a classifier which were tighter than those established by Shapire, casting doubt on the existing margin explanation. Other algorithms designed to maximize margins, such as *LP-Boost* have also been found to perform worse than AdaBoost in practice (Wang et al., 2011). Critics of these supposed counterexamples to the margin view of boosting note that AdaBoost's success likely depends on the entire distribution of margins on the data, not just the smallest margin. More recent work has improved upon the Breiman's generalization bound by taking into account other aspects of the margin that more closely reflect its distribution, adding new life to the margin explanation of AdaBoost (Gao and Zhou, 2013). While the margin explanation of AdaBoost is certainly intuitive, its role in producing low generalization error is still an area of active research.

2.2.2. 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 as 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. 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 account 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. Furthermore, among many similar examples in the literature, Mease and Wyner (2008) present a simulation example in which the the exponential loss is monotonically increasing with the number of iterations of AdaBoost on a test set, while the generalization error decreases. In this example, the value of the exponential loss is uninformative about how well the classifier generalizes. Freund et al. (2013) also provide evidence to this end. They conduct an experiment that compares AdaBoost to two AdaBoost variants that minimize the exponential loss function at differing rates: one performs the minimization very quickly through gradient descent, while the other performs the minimization quite slowly. They find that AdaBoost performed significantly better than these two competitors, suggesting that AdaBoost's strong performance cannot be tied exclusively to its action on the exponential loss function.

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

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 $\hat{C}_{rf}^B(\mathbf{x}^*) =$ majority vote$\{\hat{C}_b(\mathbf{x}^*)\}_1^B$.

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 use of early-stopping as a form of regularization has also been called into question (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 low complexity classifiers may not be able to achieve good accuracy in difficult classification tasks such as speech recognition or image recognition (Cortes et al., 2014). This paper proposes an algorithm called "DeepBoost" which encourages boosting high complexity base classifiers—such as very deep decision trees—but in a "capacity-conscious" way. 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, point 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:

- Boosting works well, perhaps best in terms of performance if not efficiency, with "strong learners" like C4.5 and CART (Niculescu-Mizil and Caruana, 2005).
- The value of exponential loss does not always bear a clear relationship to generalization error (Mease and Wyner, 2008).
- The optimization theory offers no explanation as to why the training error can be zero, yet the test error continues to descend (Freund and Shapire, 2000)

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. 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.

2.3. Interpolating Classifiers

It is a widely held belief by statisticians that if a 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.

2.3.1. 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 opti-

mizing 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 (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.

2.3.2. Local Robustness of Interpolating Classifiers

Let us begin with a definition of interpolation:

Definition: Let X_i be vector observations of predictor variables a 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, $f(X_i) = Y_i$.

The term "interpolation" is likely jarring for some readers. In many contexts, one often thinks about interpolating a set of points with classically smooth functions, such as polynomial splines. However, strictly speaking, there are many other ways that one might interpolate a set of points—through the fit of an AdaBoost classifier, for instance! Since the notion of fitting a set of points without error is central to this paper, and since the common definition of interpolation does not preclude the kinds of fits we consider, we felt it appropriate to proceed with the term.

Many statisticians are not comfortable with classifiers that interpolate the training data: common wisdom suggests that any classifier which fits the training data per-

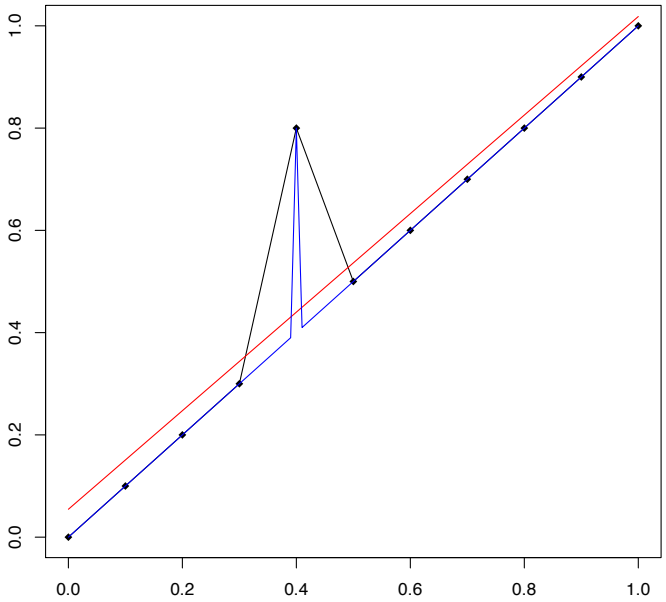


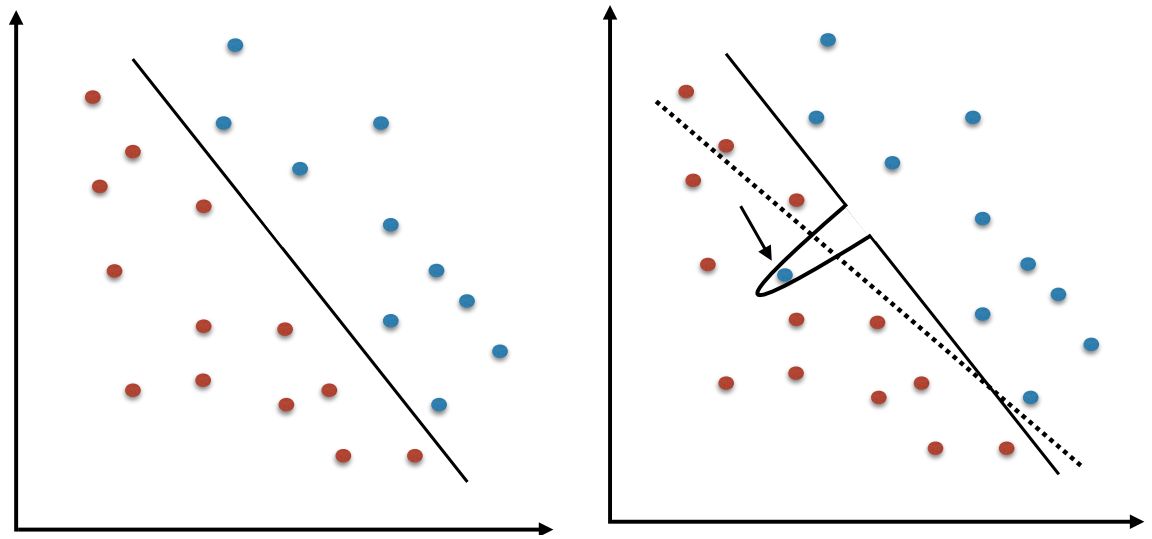
Figure 1: Three estimated regression functions with two interpolating fits (black and blue) and an ordinary least squares fit (red).

fectly 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. Specifically, Cover and Hart (1967) have shown that the asymptotic generalization error for the one-nearest neighbor classifier is at least as large as the Bayes error rate. 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.

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. In order to make this point conceptually clear, it is helpful to put ourselves in familiar territory with a regression example.

Suppose we are trying to predict a continuous response y based on real-valued x observations. Let us assume that the true underlying model is $y = x + \epsilon$, where ϵ is a mixture of a point mass at zero and some heavy-tailed distribution. In other words, we'll assume that most points in a given training set reflect the true linear relationship between y and x, but a few observations will be noise points. This is analogous to the types of probability models we typically consider in classification settings, such as those found in later sections of the paper. Figure 1 shows hypothetical training data: note that the only "noise" point is found at $x = 0.4$. We then consider fitting three models to this data: two interpolating functions, given by the blue and black lines, and an ordinary regression fit given by the red line. The first thing to notice is that the two interpolating fits differ only from the true target mean model $y = x$ only at the noise point $x = 0.4$. In contrast, the fit of the regression line deviates from the underlying target over the entire range of x. The one noise point corrupted the entire fit of the regression line, while the interpolating lines were able to minimize the influence of the noise point by adapting to it only very locally. Moreover, one should note that between the two interpolating fits, the blue line interpolates more locally than the black line, and thus its overall fit is even less influenced by the noise point. This simplified example is of course meant to be didactic, but we will show throughout the rest of this paper that in practice AdaBoost and random forest do indeed produce fits similar to the blue line.

While it is conceptually clear that it is desirable to produce fits like the blue interpolating line in the previous example, one may wonder how such fits can be achieved in practice. In the classification setting, we will argue throughout this paper that this type of fit can be realized through the process of averaging interpolating classifiers. We will refer to the decision surface produced by this process as being *spiked-smooth*.



(a) (b)

Figure 2: An illustration of the robustness to noise of interpolating classifiers. In 2a a classifier (such as a logistic regression) is fit to a set of training data, and the decision boundary produced by this classifier is shown as a solid line. In 2b, the same type of classifier is fit to the training data, except a noise point is added to the data (the blue point marked by the arrow). The one noise point shifts the entire decision boundary, which is shown as the dotted line. On the other hand, the decision boundary produced by a classifier which interpolates very locally is shown as a solid line. It is clear that this classifier is able to adapt locally to the noise point, and the overall fit does not get corrupted.

The decision surface is *spiked* in the sense that it is allowed to adapt in very small regions to noise points, and it is *smooth* in the sense that it has been produced through averaging. A technical definition of a spiked-smooth decision surface would lead us too far astray. It may be helpful instead to consider the types of classifiers that do not produce a spiked-smooth decision surface, such as a logistic regression. Logistic regression separates the input space into two regions with a hyperplane, making a constant prediction on each region. This surface is not spiked-smooth because it does not allow for local variations: in large regions (namely, half-spaces), the classifier's predictions are constrained to be the same sign. Figure 2 provides a graphical illustration of this intuition.

Intuitively, we expect ensembles of interpolating classifiers to generalize well because they are flexible enough to fit a complex signal, and "local" enough to prevent the undue influence of noise points. It is clear that random forests are averaged collections of interpolating classifiers, and we will show in section 2.4 that AdaBoost may be thought of in the same way. Classically smooth classifiers—such as logistic regression or pruned CART trees—are forced to produce fits that are locally constant. It is harder for such classifiers to "recover" from making a mistake at a noise point since the surface of the fit will affect the fit at nearby points. Interpolating classifiers are flexible enough to make mistakes in "small regions." When averaged over many such classifiers, the influence of the noise point can be easily smoothed out. Later examples in the paper will visualize this process, and will demonstrate that the later iterations of AdaBoost have a smoothing effect which shrinks down the influence of noise points on the overall fit of the classifier.

With this discussion in mind, let us consider another conceptual example, this time in 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

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

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 $\mathbb{P}(\mathbf{x})$ 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 3 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 4 shows four hypothetical classifiers that could result from fitting boosted decision tree models to the training data in Figure 3. 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 3 will necessarily be constant throughout; this is the finest resolution of the classifier resulting from boosting decision tress assuming no sub-sampling. Thus, Figure 4a represents the interpolating classifier closest to the Bayes rule. (In these plots, pink

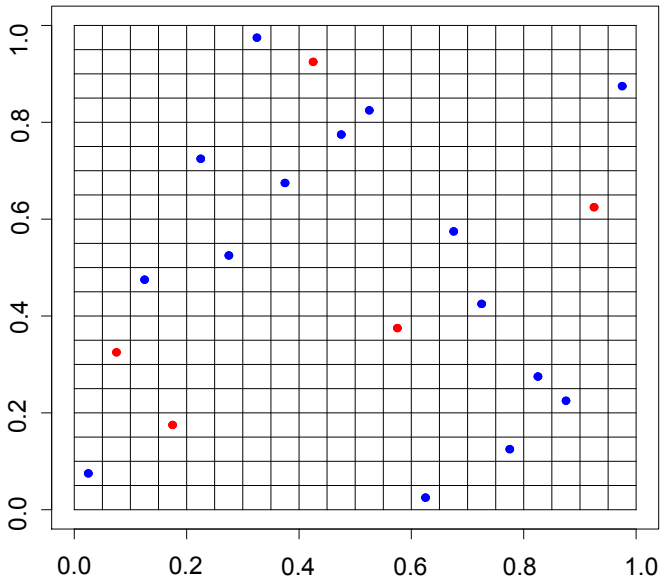


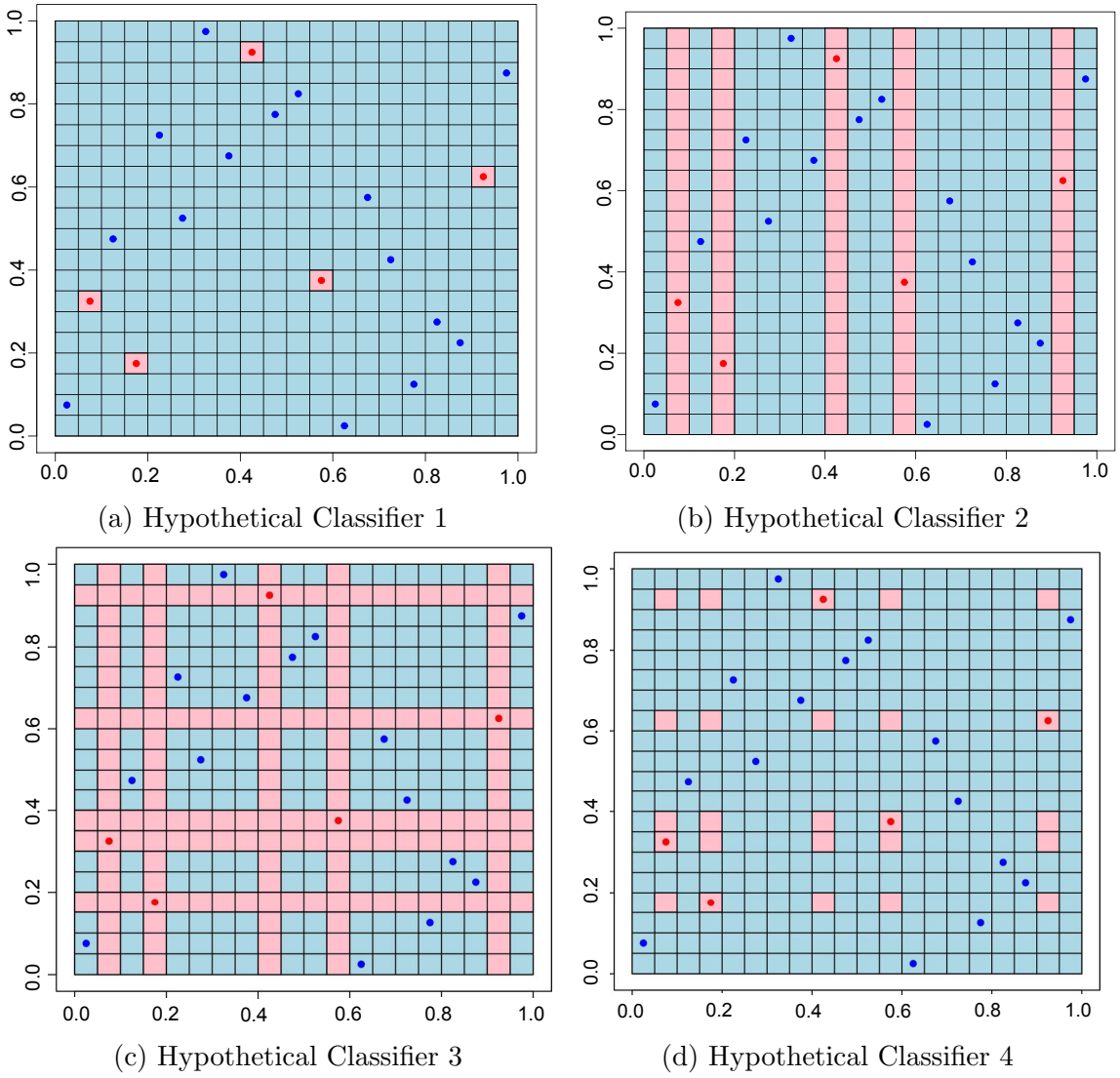
Figure 3: Training Data

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 $p = \mathbb{P}(y = 1|\mathbf{x})$ and d is the dimensionality of the predictor space (for our example, $d = 2$). We will present evidence later that in noisy environments boosting sufficiently large trees does actually tend to find such rules as that in Figure 4a. 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."



(a) Hypothetical Classifier 1

(b) Hypothetical Classifier 2

(c) Hypothetical Classifier 3

(d) Hypothetical Classifier 4

Figure 4: Four different hypothetical classifiers on a pure noise response surface where $\mathbb{P}(y = 1|\mathbf{x}) = 0.75$.

While a classifier such as that in Figure 4a would preform well and is even consistent, many possible interpolators exist, such as the others displayed in Figure 4.

Figure 4b shows the (hypothetical) result of allowing the boosting algorithm to use trees involving only x_1 and not x_2.

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 4c has an even worse error rate, while the classifier in Figure 4d 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 differ 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 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 2.4.3.

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 4a. AdaBoost and random forests average many individually overfit classifiers, similar to the one in Figure 4b. The final result is a robust classifier, that is spiked-smooth; it fits the noise but only extremely locally.

2.3.3. 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 interpolate more locally around error points in 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 $\mathbf{x}$ is 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 $\mathbb{P}(y = 1|\mathbf{x}) = .8$.

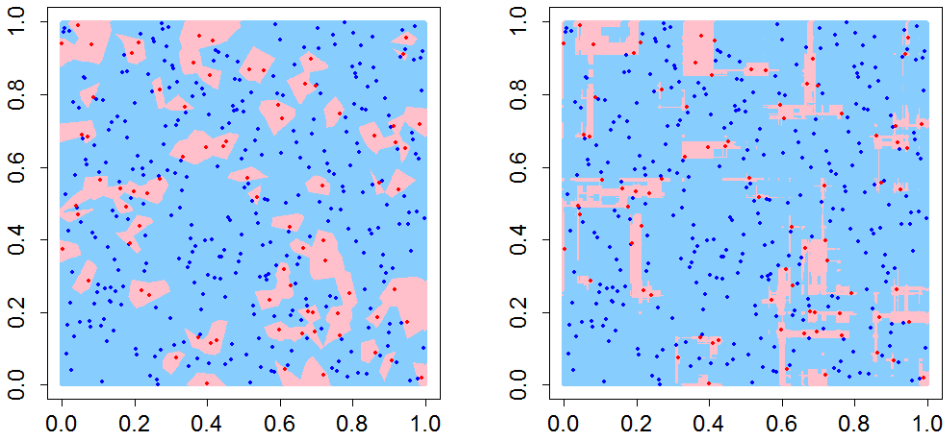
Figure 5 displays the results for the following: (a) one-NN, (b) AdaBoost , and (c) random forests. Regions classified as $+1$ are colored light blue and regions classified as -1 are colored pink. The training data is displayed with blue points for $y = +1$

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 2.3.6, 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.

2.3.4. A Visualization of Spiked-Smoothing

We have argued that local interpolation such as in Figure 4c 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 the process of spiked-smoothing in the case of the random forest classifier from the previous simulation. Each of the first



(a) one-NN

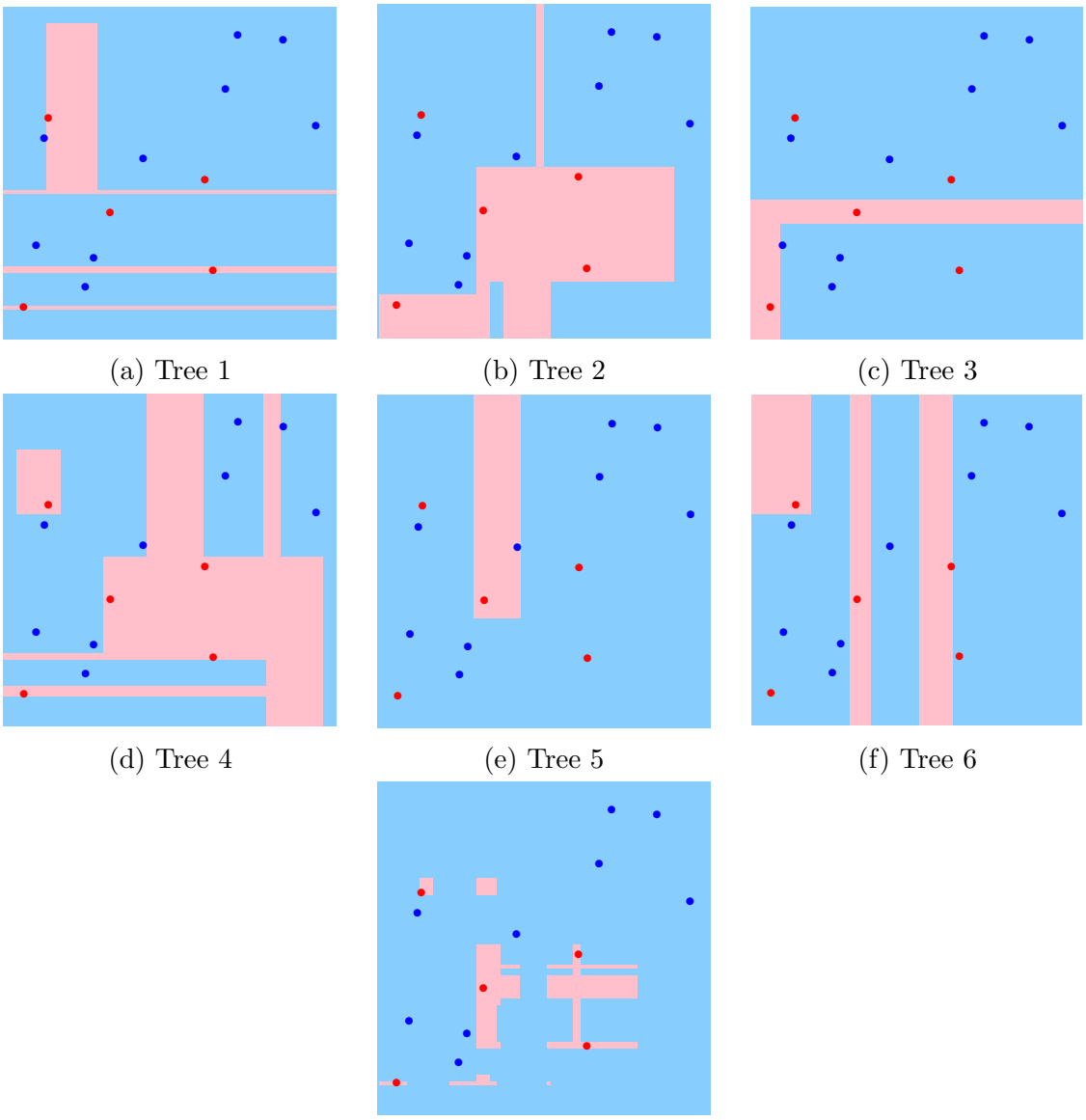
(b) AdaBoost

(c) Random Forests

Figure 5: Performance of one-NN, AdaBoost, and random forests on a pure noise response surface with $\mathbb{P}(y = 1|x) = .8$ and $n = 400$ training points.

six plots in Figure 6 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 6g 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 the 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 6g 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 2.4 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.



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

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

(g) Majority Tree Vote

Figure 6: First six trees from a random forest, along with the classifier created by a majority vote over the trees.

2.3.5. 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 Hypercube 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 7 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 using 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.

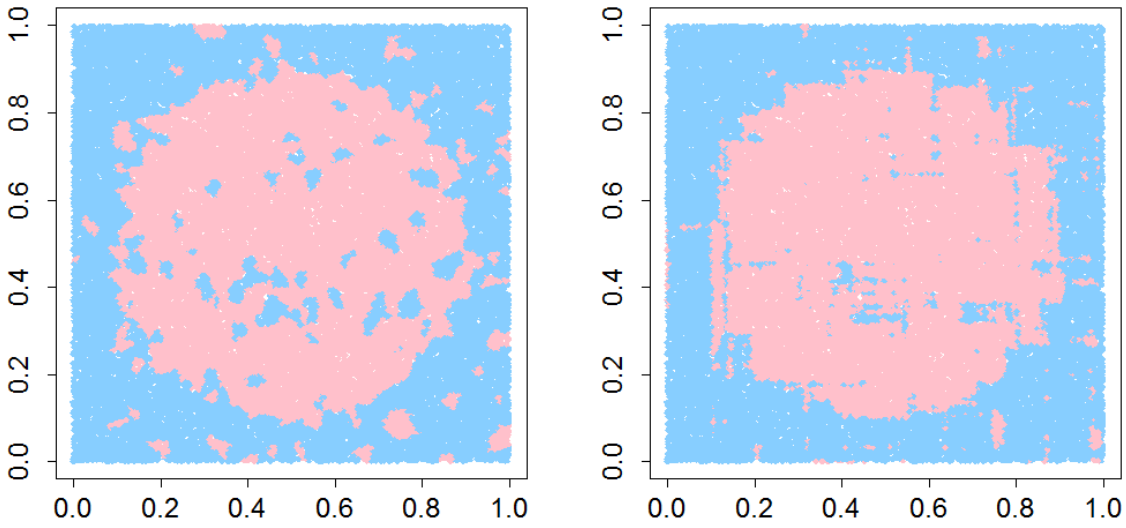
We find that AdaBoost and random forests have an overall error rate of around 0.13, one-nearest neighbor has an overall error rate of 0.20, and CART has an error rate of 0.18. CART fails to perform well in this example because it is not allowed enough

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.

2.3.6. A Twenty-Dimensional Example

We now repeat the simulation in Section 2.3.3 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 20% or 1000 of these points to be -1's with the remaining 4000 to be $+1$'s.

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 8 plots the proportion of points in the hold out sample 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 $+1$ 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



(a) One-NN

(b) AdaBoost

(c) Random Forests

(d) CART

Figure 7: Performance of AdaBoost, random forests, and CART on a response surface where $\mathbb{P}(y = 1|\mathbf{x}) = 0.10$ inside the circle and $\mathbb{P}(y = 1|\mathbf{x}) = 0.90$ outside of the circle. There are $n = 1000$ training points and the Bayes error is 0.10.

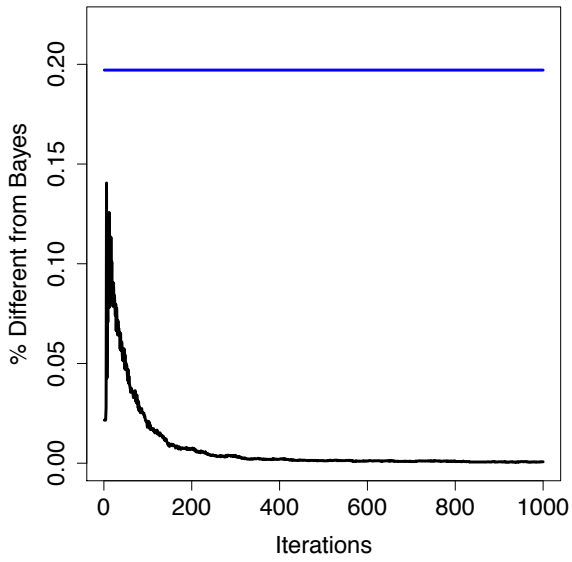


Figure 8: This plot shows the proportion of points in a test set for which the predictions made by AdaBoost and the Bayes rule differ, as a function of the number of boosting rounds (black). The blue line shows this proportion for the one nearest neighbor classifier. Note that the agreement of AdaBoost and the Bayes rule increases with the number of boosting rounds.

99.92% of the points. Recall that AdaBoost fits the training data perfectly, and thus differs from the Bayes rule on 20% of this sample. 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.

2.4. Self-Averaging Property of Boosting

2.4.1. 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 9 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 (for instance, 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 interpolates 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

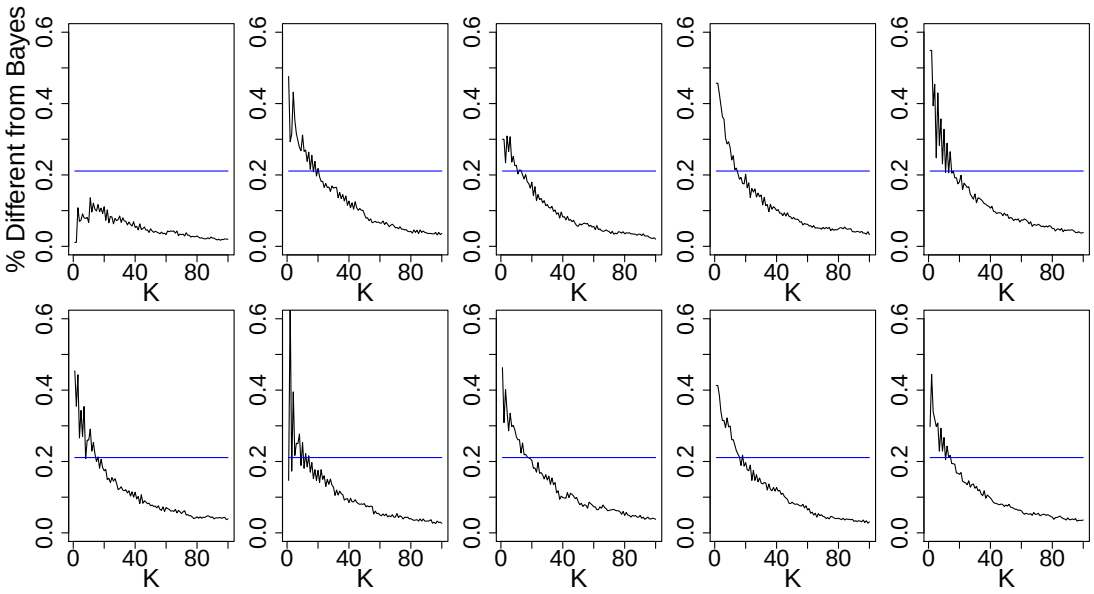


Figure 9: A decomposition of boosting

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 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 labeled 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

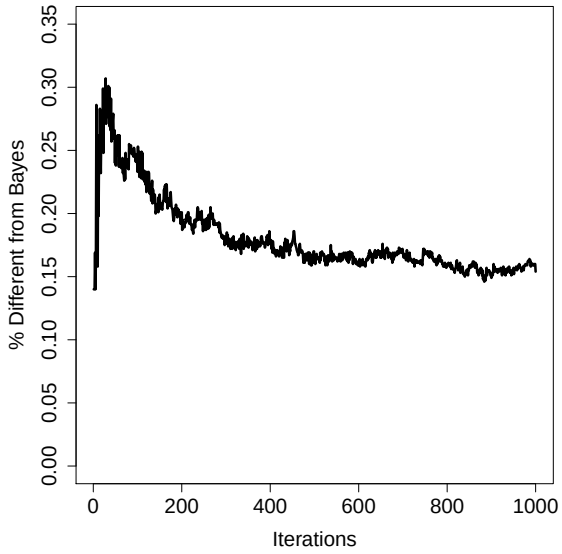


Figure 10: This plot shows the proportion of points in a test set for which the predictions made by AdaBoost and the Bayes rule differ, as a function of the number of boosting rounds (black). The test set has been chosen to contain excess points near the error points in the training set.

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 10) so even points that are quite close to the label errors (the -1 points) eventually become classified correctly as $+1$. Comparison to Figure 8 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.

2.4.2. A Five-Dimensional Example

We will 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

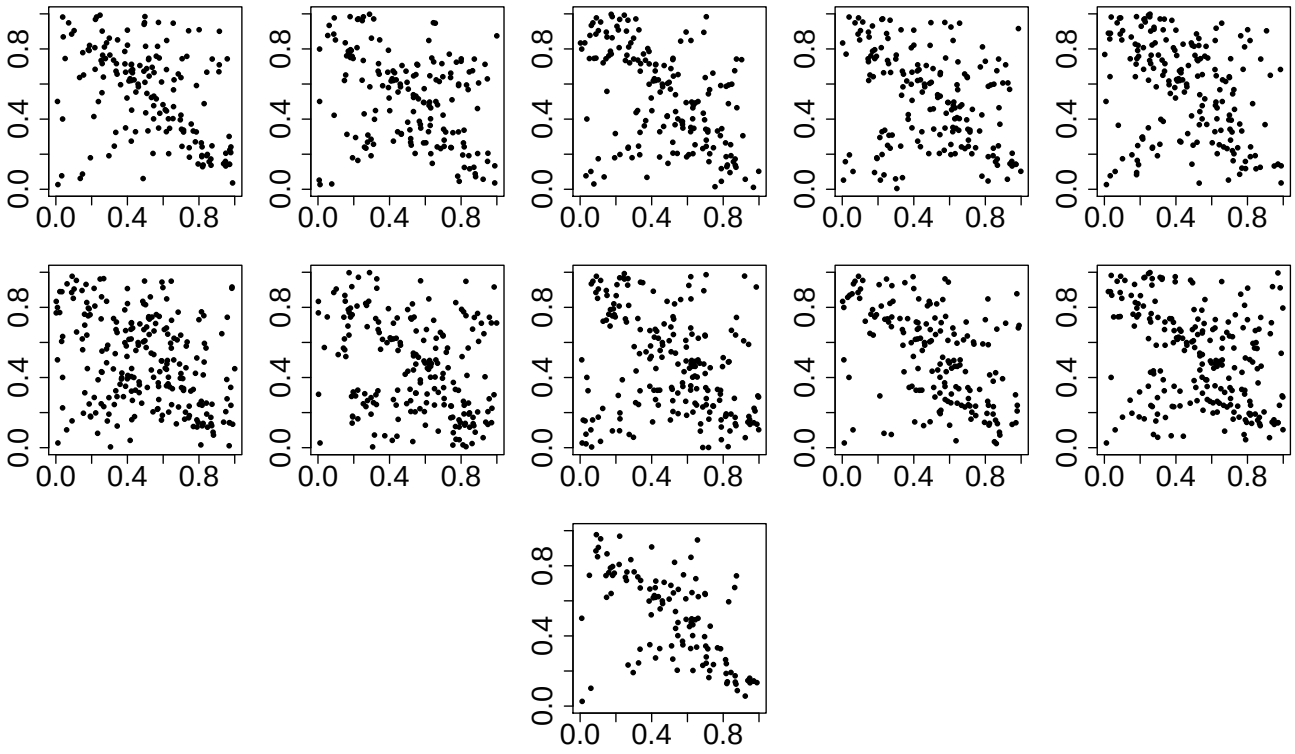


Figure 11: Errors made by each classifier in a decomposition of boosting (first two rows) and errors made by the final classifier (bottom)

simulation is

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

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 11 show the performance of the ten classifiers corresponding to this decomposition with respect to a hold out sample of 1000 points. Each 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).

In this example, we also have evidence that AdaBoost takes some measure to decorrelate the errors made by its base classifiers, as hypothesized in Amit et al. (2000) and suggested in Figure 11. To this end, we computed the correlation between $\mathbb{I}[h_{10}^1(X) = Y], \ldots, \mathbb{I}[h_{10}^{10}(X) = Y]$ over a large test set. The correlations ranged from 0.4 to 0.56, with an average value of 0.488. As a comparison, we also considered a similar calculation for the decomposition produced from a random forest with 500 trees, and from bagging 1000 depth 8 trees. As with AdaBoost, we can also decompose these classifiers into a sum of interpolating classifiers: 25 sub-ensembles of 20 trees in the case of the random forest, and 10 sub-ensembles of trees in the case of bagged trees. The interpolating classifiers produced by the random forest had error correlations ranging from 0.75 to 0.87, with an average of 0.81, while the correlations for the bagged trees ranged from 0.9 to 0.94, with an average value of 0.92. In other words, the sub-ensembles produced by random forests and AdaBoost are less correlated than the bagged trees, with AdaBoost being markedly less so. When voting across constituents in its decomposition, it is clear that such a decorrelating effect would help to increase the effectiveness of the voting mechanism to cancel out error points. While we observed this phenomenon in a few data sets, we cannot state under what conditions it happens more generally. However, it is know that boosting generally outperforms bagging, and that bagged trees will be much more correlated.

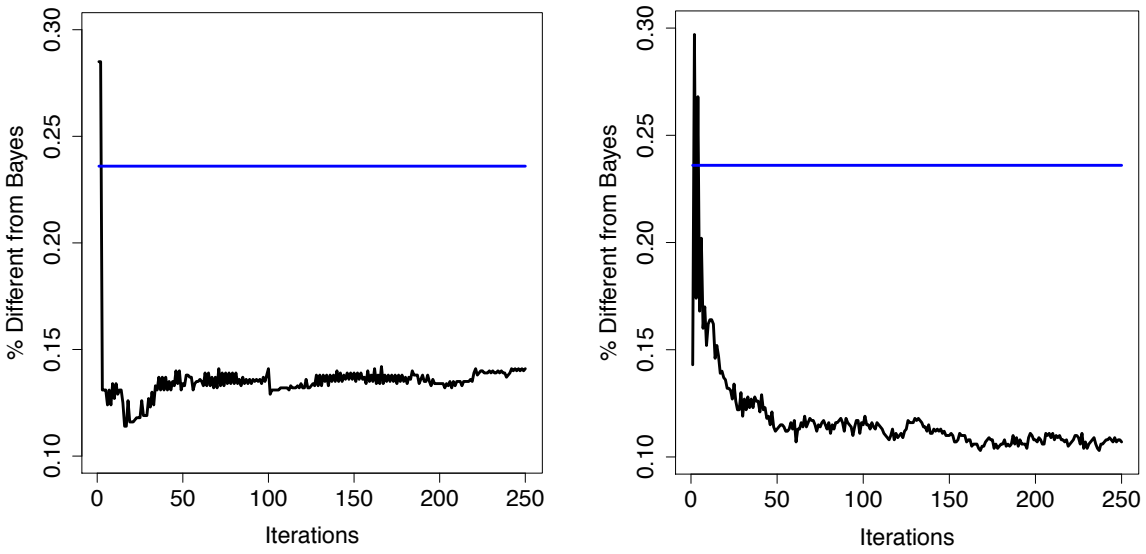
2.4.3. Comparison to Boosted Stumps

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 2.4.2 which has an additive Bayes decision rule. Figure 12 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 data set 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.



(a) Stump (b) 2^8 Node Trees

Figure 12: Comparison of AdaBoost with stumps (a) and 2^8 node trees (b) for the five dimensional simulation. The blue line corresponds to One-NN.

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.

2.5. Real Data Example

In previous sections, we explored the mechanism of spiked smoothing in simulation experiments where it was easy for us to identify noise points. Recall that with a fully

specified probability model, noise points are simply sample points whose sign differs from the Bayes' rule. While simulated examples are certainly good for illustration, one may wonder whether these settings are overly simplistic. Data in the real world is typically generated by more complicated—and unknown—probability models with heteroskedastic noise components. Since boosted trees have been empirically successful in such settings, it pays to move our discussion to the context of a data set arising in the real world.

2.5.1. Phoneme Data

In this section, we will discuss spiked smoothing in the context of a data set arising in a speech recognition task designed to discriminate "nasal" vowels from "oral" vowels in spoken language [3]. For a collection of 5404 examples of spoken vowels, this data set contains the amplitudes of the first 5 harmonic frequencies as collected by a cochlea spectra (hearing aid), along with a label of either nasal or oral (n=5404, p=5). The goal is then to use these harmonic frequencies to correctly identify the type of vowel. While there are many potential data sets we could have considered, this one is convenient to analyze since it consists of a relatively small number of real valued covariates. This makes it easier to talk about "neighborhoods" of points later in our discussion.

We begin by dividing the phoneme data set up into a training set consisting of 70% of the samples, and a testing set consisting of 30% of the examples. Depth eight decision trees boosted for 1000 rounds and a random forest classifier both achieve comparable test error rates of 9.0% and 9.4%, respectively [4]. Figure 13 demonstrates

[3]The original data can be found at the following address: https://www.elen.ucl.ac.be/neural-nets/Research/Projects/ELENA/databases/REAL/phoneme/phoneme.txt.

[4]The error rates reported are the result from repeating the fitting procedure on 100 random train/test set splits and considering the average error on the test set.

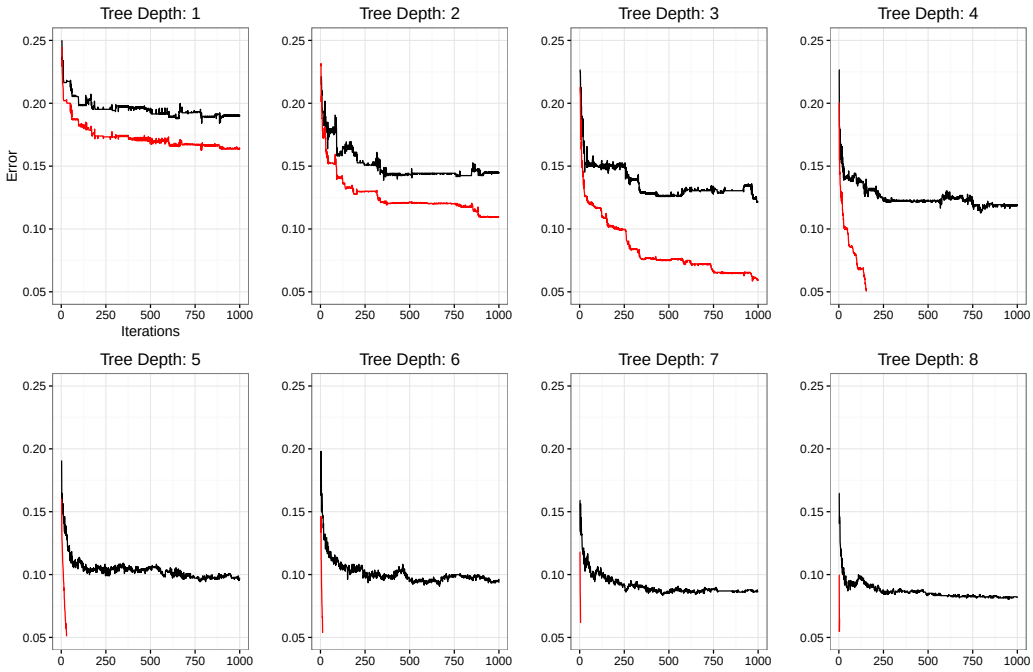


Figure 13: Plots of testing and training error as a function of number of boosting iterations for trees of different depths. The black lines show the test error rates while the red lines show training error rates. Notice that depth seven and eight boosted trees quickly fit the training data, as depicted by the rapidly decreasing red lines.

that boosting deep trees is preferable to shallow trees in this data set. Each frame in the figure shows the testing error in black and the training error in red as a function of the number of boosting iterations for different depth trees. One can readily observe that test error steadily decreases with the depth of tree used in AdaBoost. It is interesting to note that boosting trees of all depths are slow to overfit the data, even with boosted stumps after 1000 iterations. It is also worth noting that boosted depth 8 trees quickly interpolate the training data, as indicated by the sharply decreasing red line. This raises the often asked question: what is AdaBoost doing after it achieves a perfect fit on the training set?

Any discussion of spiked smoothing on real data is complicated by the reality that it is impossible to identify noise points without knowing the underlying probability

model. As a substitute for noise points, we flipped the signs of 100 randomly chosen points in our training data set (about 2% of the data) and analyzed the fits of boosted trees and a random forest classifier around these points. After refitting the models to the perturbed data, we find the the testing errors are 10.2% and 10.0% for boosted depth eight trees and a random forest, respectively. Figure 14 illustrates that even after thousands of rounds of boosting, test error continues to decrease. The punchline is that both algorithms are able to achieve fits with comparable test set error even after flipping the sign of a large number of points in the training set.

As a comparison with another interpolating classifier, we also repeated the same experiment with a one nearest-neighbors classifier. We found the one nearest neighbor achieved a test error rate of 10.5% when fit on the original training set, and a test error rate of 12.6% when fit on the noisy training set. The increase in error rate when noise is added to the training set is larger than that of AdaBoost or a random forest, which coincides with the results form the simulated example in section 2.3.3. Furthermore, two sample t-tests reveal that the increase in error rate for one nearest-neighbors is significant at the 0.01 level in both cases (p-values less than $1e^{-16}$ in both cases).

We will argue that additional rounds of boosting are helping to "smooth out" and "localize" the influence of the 100 noise points that we introduced into our training set. In section 2.3.3, we illustrated the action of spiked smoothing in diagrams which showed AdaBoost's fit on a two dimensional plane. In this slightly higher dimensional example, we instead consider small neighborhoods around each of the noise points and track the fraction of points in these neighborhoods that agree with the sign of the "correct label", that is, before flipping the sign. [5]. Panels (a) and (b) of

[5]We choose each neighborhood to be a rectangle centered at the point of interest, with side lengths chosen in such as way that the neighborhood contains only a small number of training points. We

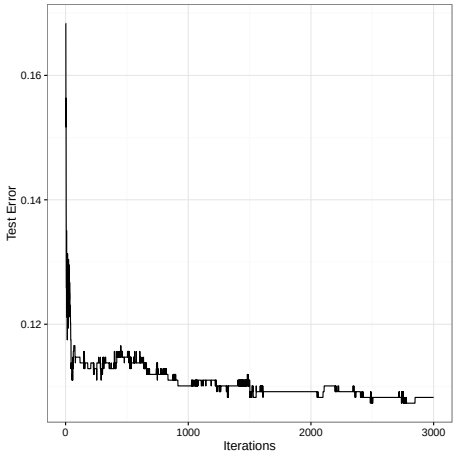
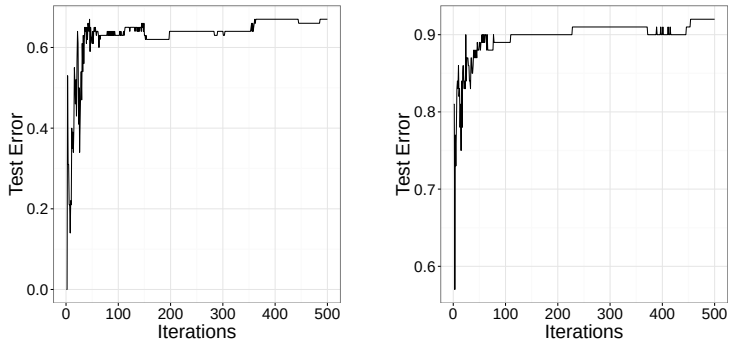


Figure 14: Plot of testing error for depth 8 boosted trees fit to a training set which has flipped labels for 100 randomly selected examples.

Figure 15 plot this fraction as a function of the number of boosting iterations for two representative "noise points." In both panels, it is clear that as the number of iterations increases, the fraction of points in each neighborhood that agrees with the original sign of the training point increases. Recall that in this case AdaBoost still fits its training data perfectly: the in-sample AdaBoost fit agrees with the sign of the flipped training point in both figures. Despite this, the algorithm still fits a majority of points in a neighborhood of each of these noise points in the correct way. The classifier is producing a spiked smooth fit, that is, it fits the data in such a way to localize the influence of noise points. One can interpret the increasing homogeneity in each neighborhood as the result of averaging. The first few iterations of deeply boosted trees produce a fit that interpolates the training data, but this fit is quite complicated in the sense that it assigns large numbers of points in the neighborhood to both '+1' and '-1'. As the number of iterations increases, this fraction increases so that the fit becomes more smooth in the classical sense.

then chose 100 points uniformly at random in this rectangle and computed the AdaBoost classifier at each point: these points are obviously not included in the original data set.



(a) (b)

Figure 15: The fraction of points in a neighborhood that agree with the sign of the original training point before its sign was flipped. Figures (a) and (b) plot this proportion for two different, representative noise points.

2.5.2. Additional Data Sets

We repeated a version of the analysis conducted in Section 2.5.1 for five additional data sets from the UCI repository: Haberman, Wisconsin breast cancer, voting, Pima, and German credit. In our analysis we did the following: we added 5% label noise to the training data, missing values were imputed with the mean prior to model fitting, and all experiments were conducted on 50 random training-testing splits of each data set. Table 1 reports the mean increase in testing error after adding 5% label noise over the 50 random training-testing splits. For example, on the original haberman data set, AdaBoost achieved an average error rate of 34.225%, and on the noisy version of the data set achieved an average error rate of 34.354%. The mean difference of 0.13% is reported in the table. It is also apparent in each setting that AdaBoost and random forest are relatively immune to the addition of label noise. The stars in the Table 1 report the significance level when comparing the increase in error rate for AdaBoost and a random forest with that of one-nearest neighbors using a two-sample t-test. With the exception of the German credit data, the increase in error rate for one nearest neighbors was larger than that of one nearest neighbors with statistical

significance.

Data set	AdaBoost	Random Forest	1-NN
Haberman	0.13^{**}	0.52^{*}	**1.55**
breast_cancer	0.20^{***}	0.39^{***}	**2.29**
voting	1.63^{**}	0.30^{***}	**2.71**
Pima	0.56^{***}	0.45^{***}	**1.75**
German	0.29	0.68	**0.68**

Table 1: The increase in average testing error after changing the sign of 5% of the training data. The stars in the table report the significance level when comparing the increase in error rate for AdaBoost and a random forest with that of one-nearest neighbors using a two-sample t-test. One star denotes significance at the 10% level, two stars denotes significance at the 5% level, and three stars denotes significance at the 1% level.

2.6. 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 data sets 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 examples to 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 points. We hope to have cast doubt on the commonly held belief that the later iterations of AdaBoost only serve to overfit the data. Instead, we argue that these later iterations lead to an "averaging effect", which causes AdaBoost to behave like a random forest.

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.

CHAPTER 3 : Deep Neural Networks Generalize on Small Data

Abstract

In this paper, we use a linear program to empirically decompose fitted neural networks into ensembles of low-bias sub-networks. We also show that these sub-networks are relatively uncorrelated which leads to an internal regularization process, very much like a random forest, which can explain why a neural network is surprisingly resistant to overfitting. We then demonstrate this in practice, by applying large neural networks, with hundreds of parameters per training observation, to a collection of 116 real-world data sets from the UCI Machine Learning Repository. This collection of data sets contains a much smaller number of training examples than the types of image classification tasks generally studied in the deep learning literature, as well as non-trivial label noise. We show that even in this setting deep neural nets are capable of achieving superior classification accuracy without overfitting.

3.1. Introduction

A recent focus in the deep learning community has been resolving the "paradox" that extremely large, high capacity neural networks are able to simultaneously memorize training data and achieve good generalization error. In a number of experiments, Zhang et al. (2017) demonstrated that large neural networks were capable of both achieving state of the art performance on image classification tasks, as well as per-

*Joint work with Adi Wyner and Richard Berk.

fectly fitting training data with permuted labels. The apparent consequence of these observations was to question traditional measures of complexity considered in statistical learning theory.

A great deal of recent research has aimed to explain the generalization ability of very high capacity neural networks (Kawaguchi et al., 2017). A number of different streams have emerged in this literature (Neyshabur et al., 2017; Liang et al., 2017; Lin et al., 2017). The authors in Zhang et al. (2017) suggest that stochastic gradient descent (SGD) may provide implicit regularization by encouraging low complexity solutions to the neural network optimization problem. As an analogy, they point out that SGD applied to under-determined least squares problems produces solutions with minimal ℓ_2 norm. Other streams of research have aimed at exploring the effect of margins on generalization error (Bartlett et al., 2017; Liang et al., 2017). This line of thought is similar to the margin-based explanations of AdaBoost in the boosting literature that bound test performance in terms of the classifier's confidence in its predictions. Other research has investigated the sharpness of local minima found by training a neural network with SGD (Dinh et al., 2017; Neyshabur et al., 2017). The literature is extensive, and this review is far from complete.

The empirical investigations found in this literature tend to be concentrated on a small set of image classification data sets. For instance, every research article mentioned in the last section with an empirical component considers at least on of the following four data sets: MNIST, CIFAR-10, CIFAR-100, or ImageNet. In fact, in both NIPS 2017 and ICML 2017, over half of all accepted papers that mentioned "neural networks" in the abstract or title used one of these data sets. All of these data sets share characteristics that may narrow their generality: similar problem domain, very low noise rates, balanced classes, and relatively large training sizes (60k training points at

minimum). In this work, we consider a much richer class of *small* data sets from the UCI Machine Learning Repository in order to study the "generalization paradox." These data sets contain features not found in the image classification data, such as small sample sizes and nontrivial, heteroscedastic label noise.

As part of our investigation we will establish that large neural networks with tens of thousands of parameters are capable of achieving superior test accuracy on data sets with only hundreds of observations. This is surprising, as it is commonly thought that deep neural networks require large data sets to training properly (Rolnick et al., 2017). In fact, we will establish that with minimal tuning, deep neural networks are able to achieve performance on par with random forests, which are considered to have state-of-the-art performance on data sets from the UCI Repository (Fernández-Delgado et al., 2014).

The mechanism by which a random forest is able to generalize well on small data sets is straightforward: a random forest is an ensemble of low-bias, decorrelated trees. Randomization combined with averaging reduces the ensemble's variance, smoothing out the predictions from fully grown trees. It is clear that a neural network should excel at bias reduction, but the way in which it achieves variance reduction is much more mysterious. The same paradox has been examined at length in the literature on AdaBoost, and in particular, it has been conjectured that the later stages of AdaBoost serve as a *bagging* phase which reduces variance (Breiman, 2000a,b).

One of the central aims of this paper is to identify the variance stabilization that occurs when training a deep neural network. To this end, the later half of this paper focuses on empirically decomposing a neural network into an ensemble of sub-networks, each of which achieves low training error and less than perfect pairwise correlation. In this way, we view neural networks in a similar spirit to random forests.

One can use this perspective as a window to viewing the success of recent residual network architectures that are fit with hundreds or thousands of hidden layers (He et al., 2016). These deeper layers might serve more as a bagging mechanism, rather than additional levels of feature hierarchy, as is commonly cited for the success of deep networks (Bengio et al., 2009).

The key takeaways from this paper are summarized as follows:

- Large neural networks with hundreds of parameters per training observation are able to generalize well on small, noisy data sets.
- Despite a bewildering set of tuning parameters (Bengio, 2012), neural networks can be trained on small data sets with minimal tuning.
- Neural networks have a natural interpretation as an ensemble of low-bias classifiers whose pairwise correlations are less than one. This ensemble view offers a novel perspective on the *generalization paradox* found in the literature.

3.2. Ensemble View of Deep Neural Networks

In this section, we establish that a neural network has a natural interpretation as an ensemble classifier. This perspective allows us to borrow insights from random forests and model stacking to gain better insight as to how a neural network with many parameters per observation is still able to generalize well on small data sets. We also outline a procedure for decomposing fitted neural networks into ensemble components of low bias, decorrelated sub-networks. This procedure will be illustrated for a number of neural networks fit to real data in Section 3.4.4.

3.2.1. Network Decomposition

We will begin by recalling some familiar notation for a feed-forward neural network in a binary classification setting. In the case of a network with L hidden layers, each layer with M hidden nodes, we may write the network's prediction at a point $x \in \mathbb{R}^p$ as

$$\begin{aligned} z^{\ell+1} &= W^{\ell+1} g(z^\ell) \;\; \ell = 0, \ldots, L \\ f(x) &= \sigma(z^{L+1}) \end{aligned} \tag{3.1}$$

where σ is the sigmoid function, g is an activation function, $W^{L+1} \in \mathbb{R}^{1 \times M}$, $W^1 \in \mathbb{R}^{p \times M}$, and $W^\ell \in \mathbb{R}^{M \times M}$ for $\ell = 2, \ldots, L$ (and $z^0 \equiv x$). Assume that any bias terms have been absorbed. For the models considered in this paper, $L = 10$, $M = 100$, and g is taken to be the ELU activation function (Clevert et al., 2015). It is also helpful to abuse notation a bit, and to write $z^\ell(x)$ as the output at hidden layer ℓ when x is fed through the network, i.e. $z_0 = x$.

There are many ways to decompose a neural network into an ensemble of sub-networks: one way to do this is at the final hidden layer. Let us fix an integer $K \leq M$ and consider a matrix $\alpha \in \mathbb{R}^{M \times K}$ such that $\sum_{k=1}^{K} \alpha_{m,k} = W_{1,m}^{L+1}$ for $m = 1, \ldots, M$. We can then write the final *logit* output as a combination of units from the final

hidden layer:

$$\begin{aligned} z^{L+1}(x) &= W^{L+1} g(z^L(x)) \\ &= \sum_{m=1}^{M} W_{1,m}^{L+1} g(z_m^L(x)) \\ &= \sum_{m=1}^{M} \sum_{k=1}^{K} \alpha_{m,k} g(z_m^L(x)) \\ &= \sum_{k=1}^{K} \sum_{m=1}^{M} \alpha_{m,k} g(z_m^L(x)) \\ &= \sum_{k=1}^{K} f_k(x) \end{aligned} \tag{3.2}$$

where $f_k(x) = \sum_{m=1}^{M} \alpha_{m,k} g(z_m^L(x))$. In words, we have simply decomposed the final layer of the network (at the *logit level*) into a sum of component networks. The weights that define the k^{th} sub-network are stored in the k^{th} column of α.

We will find in Section 3.4 that networks trained on a number of binary classification problems have decompositions such that each f_i fits the training data perfectly, and such that out-of-sample correlation between each f_i and f_j is relatively small. This situation is reminiscent of how a random forest works: by averaging the outputs of low-bias, low-correlation sub-ensemble. We argue that it is through this implicit regularization mechanism that overparametrized neural networks are able to simultaneously achieve zero training error and good generalization performance.

3.2.2. Ensemble Hunting

The decomposition in Equation 3.2 is of course entirely open-ended without further restrictions on the set of weights stored in α. One constraint that we will impose is that each sub-network should also achieve very high training accuracy. We will

restrict our analysis to networks that obtain 100% training accuracy, and we will demand that each sub-network f_k do so as well.

Another desideratum is that each sub-network should be built from different parts of the full network, as much to the extend that is possible. One strategy for accomplishing this is to require that the columns of α are sparse and have non-overlapping components. In practice, we found that the integer programs required to find these matrices were computationally intractable when coupled with the other constraints we consider. We also found that common ℓ_1 heuristics to finding sparse solutions tended to find α matrices with repeating columns. Our final approach was simply to force a large, randomly selected number of entries in each column to be zero through linear equality constraints.

We will now introduce some notation that will allow us to outline our ensemble search more precisely. First, we fixed the number of target sub-networks to $K = 9$. For each of the K columns of α, we sampled integers $(m_{1,k}, \ldots, m_{60,k})$ uniformly without replacement from the integers 1 to 100, and we included the linear constraints $\alpha_{m_{j,k},k} = 0$. Thus, we constrained each sub-network f_k to be a weighted sum of no more than 40 hidden units from the final hidden layer (in practice, the number of non-zero entries tended to be smaller than this). Under this scheme, two sub-networks, on average, only share at most 16 non-zero hidden units. We then used linear programming to find a matrix $\alpha \in \mathbb{R}^{100\times 9}$ that satisfied the required constraints:

$$\begin{aligned}
\sum_{k=1}^{K} \alpha_{m,k} &= W_{1,m}^{L+1}, && 1 \le m \le M \\
\alpha_{m_{j,k},k} &= 0, && 1 \le j \le 60, \;\; 1 \le k \le K \\
f_k(x_i) y_i &\ge 0, && 1 \le i \le n, \;\; 1 \le k \le K
\end{aligned}$$

In summary, the first set of constraints ensures that the sub-networks f_k decompose the full network, that is, so that $z^{L+1}(x) = \sum_{k=1}^{9} f_k(x)$ for all $x \in \mathbb{R}^p$. The second set of constraints zeros out 60 entries for each column of α, encouraging diversity among the sub-networks. The final set of constraints ensures that each sub-network achieves 100% accuracy on the training data (non-zero margin). Notice that we are simply looking for any feasible α for this system, and these constraints are rough heuristics that our sub-networks are diverse and low-bias. We could have additionally incorporated a loss function which further penalized similarity among the columns of α, such as maximizing pairwise distances between elements. It is also worth mentioning that it is straightforward to extend this program beyond binary classification to multi-class settings at the expense of a few more linear constraints. For simplicity, we avoided this more general setting.

Finally, the goal of our analysis is to examine the accuracy and *diversity* of the ensemble members $f_1, \ldots, f_9$ on a hold out test set. There are a number of metrics one might consider when assessing the diversity of ensemble components, including the *Q-statistic*, *κ-statistic*, or conditional entropy, among others (Zhou, 2012). In the examples that follow in subsequent sections, we use the fraction of test cases for which two ensemble members disagreed, averaged over all $\binom{9}{2}$ pairings.

3.2.3. Model Example

As a first application of *ensemble hunting*, we will consider a simulated model that generates data from the following process:

$$x \sim \mathcal{U}[-0.5, 0.5]^2$$

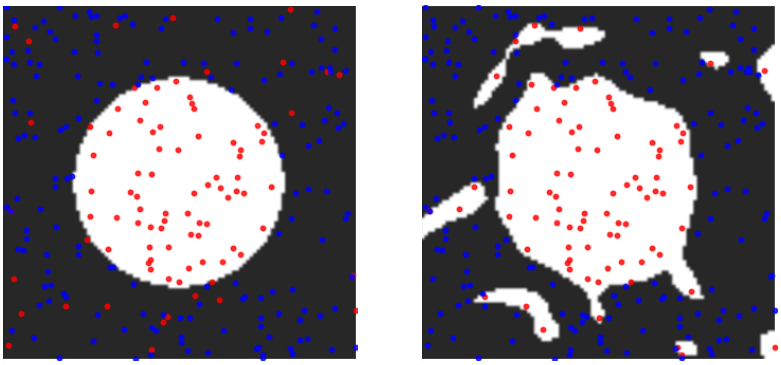
$$p(y = 1|x) = \begin{cases} 1 & \text{if } \|x\|_2 \leq 0.3 \\ 0.15 & \text{otherwise .} \end{cases}$$

The Bayes rule for this model is to assign a label $y = 1$ if x is inside a circle centered at the origin with radius 0.3, and to assign $y = -1$ otherwise. This rule implies a minimal possible classification error rate of 10%.

We draw 250 independent observations from this model, and fit two models: a neural network with $L = 10$ hidden layers and $M = 100$ hidden nodes, and a random forest with 90 trees. All models are trained so that they achieve zero training error. The decision surfaces implied by the neural networks and random forest, as well as the Bayes rule, are plotted in Figure 16.

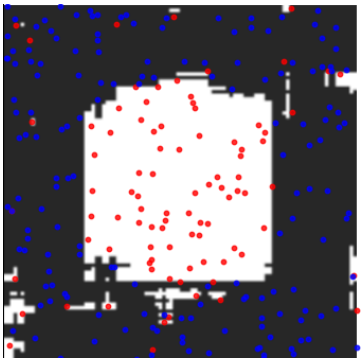
Evaluated on a hold-out set of size $n = 10,000$, the neural network and random forest achieve test accuracies of 85.2% and 88.5%, respectively. Inspecting Figure 16, we see that although each classifier fits the training data perfectly, the fits around the *noise points* outside the circle are confined to small neighborhoods. Our goal is to explain how these types of fits occur in practice.

The bottom two figures in Figure 16 show the decision surfaces produced by the full random forest and as well as a smaller forest consisting of 10 trees. Comparing these surfaces illustrates the power of ensembling: the smaller forest has a smaller accuracy

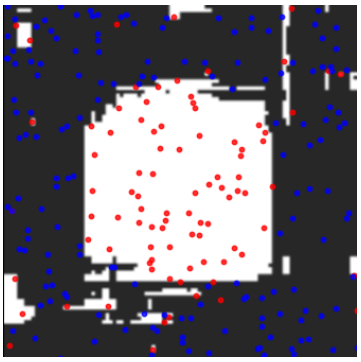


(a) Bayes Rule

(b) Neural Network



(c) RF (90 trees)

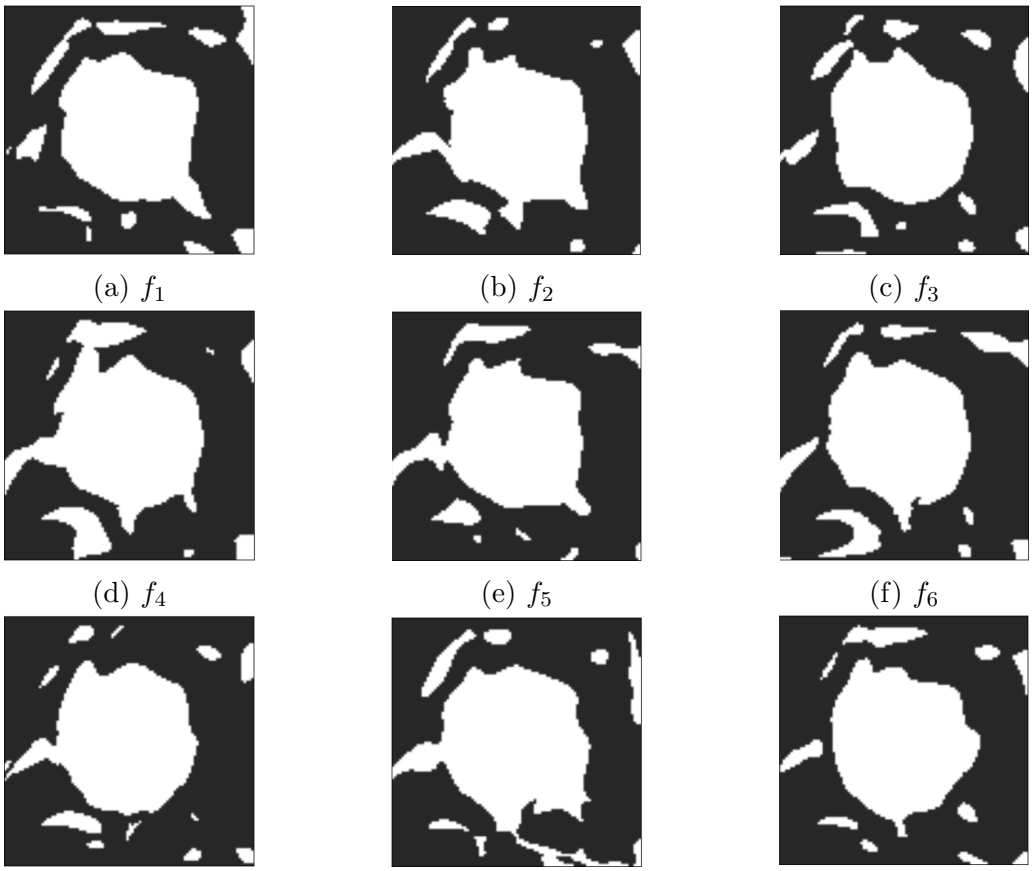


(d) RF (10 trees)

Figure 16: Plots of the decision surfaces from the classifiers fit in Section 3.2.3. In each figure, the white region indicates points for which the classifier returns a label of $y = 1$. Training points with class label $y = 1$ are shown in red, and points with class label $y = -1$ are shown in blue.

than the large forest, as evidenced by the white patches outside of the circle. However, these patches are relatively small, and when averaged with 8 other small forests, much of these get smoothed out to the correct value. Averaging works because each of the 9 smaller forests are diverse by construction. We would like to extend this line of reasoning to explain the fit produced by the neural network.

Unlike a random forest, for which it is relatively easy to find sub-ensembles with low training error and low correlation, the corresponding search in the case of neural networks requires more work. Using the ensemble-hunting procedure outlined in the previous section, we decompose the network into $K = 9$ sub-networks $f_1, \ldots, f_9$,



(a) f_1 (b) f_2 (c) f_3

(d) f_4 (e) f_5 (f) f_6

(g) f_7 (h) f_8 (i) f_9

Figure 17: Decision surfaces implied from a decomposition of the neural network from Section 3.2.3.

and we plot their associated response surfaces in Figure 17. The test accuracies of these sub-networks range from 82% to 86%, and every classifier fits the training data perfectly by construction. When examining these surfaces, it is curious that they all look somewhat different, especially near the edges of the domain. Using the test set, we find that the average pairwise disagreement among these sub-networks is 9.7%. For comparison, the average pairwise disagreement among the 9 smaller random forests comprising the larger one was 7.3%.

One surprising conclusion from this exercise was that the final layer of our fitted neural network was highly redundant: the final layer could be decomposed as 9 distinct

classifiers, each of which achieved 100% training accuracy. Common intuition for the success of neural networks suggests that deep layers provide a rich hierarchy of features useful for classification (Bengio et al., 2009). For instance, when training a network to distinguish cats and dogs, earlier layers might be able to detect edges, while later layers learn to detect ears or other complicated shapes.

Our analysis here suggests that these later layers might serve mostly in a variance reducing capacity. There are no complicated features to learn in this example: the Bayes decision boundary is a circle, which can be easily constructed from a network with one hidden layer and a handful of hidden nodes. The final layer of our deep network is highly redundant and has no problem fitting the training data. The trick is reducing variance, and this happens when the sub-networks get averaged.

3.3. Model Stacking Connection

We can provide some heuristic intuition for how the neural network optimization program relates to ensembling. In order to do so, we will draw a connection to model stacking, which is a well-known method for creating ensembles that are linear combinations of base learners (Breiman, 1996b; Wolpert, 1992). For simplicity, our discussion will focus on the regression problem with squared error loss.

Suppose that we observe a training set $\mathcal{T} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ of independent observations generated from the model $y = h(x) + \epsilon$, where $E\epsilon = 0$ and $E\epsilon^2 = \sigma^2$. Further, suppose that we have a collection of base precictors that were learned from the training data, $\widehat{f}_1, \ldots, \widehat{f}_K$. The goal of model stacking - at the population - is to minimize the expected generalization error using weighted combinations of the base learners, where we assume that the weights sum to one. Let us denote expectation with respect to the sampled training set as $E_{\mathcal{T}}$ and expectation with respect to a new

test point (x, y) as E. We may then write the ensemble problem as:

$$\min_{\alpha} E_{\mathcal{T}} E \left(y - \sum_{k} \alpha_k \widehat{f}_k(x) \right)^2 .$$

Let us further define the *bias* and *covariance* that occurs when fitting our predictors over different realizations of the training set $\mathcal{T}$:

$$\begin{aligned} \text{bias}(\widehat{f}_k)(x) &= h(x) - E_{\mathcal{T}} \widehat{f}_k(x) \\ \text{cov}(\widehat{f}_i, \widehat{f}_j)(x) &= E_{\mathcal{T}} \left(\widehat{f}_i(x) - E_{\mathcal{T}} \widehat{f}_i(x) \right) \left(\widehat{f}_j(x) - E_{\mathcal{T}} \widehat{f}_j(x) \right) . \end{aligned}$$

We may then express the population level model-stacking problem into bias and variance constituents using a bias-variance decomposition for ensembles:

$$\min_{\alpha} \left(\sigma^2 + \sum_{k} \alpha_k^2 \text{bias}^2(\widehat{f}_k)(x) + \sum_{i,j} \alpha_i \alpha_j \text{cov}(\widehat{f}_i, \widehat{f}_j)(x) \right) .$$

If we use low bias, high capacity base learners, then the third *variance* term in the expression dominates, and the the largest reduction from ensembling comes from decreasing the ensemble variance.

Now, recall that the output layer of a neural network is simply a weighted combination of the final hidden layer. Thus, if we map W^{L+1} to α and map $f_k(x)$ to the predictions made by each node in the final hidden layer, the neural network optimization problem can be written in a very similar manner to the stacking problem:

$$\min_{\alpha, f_1, \ldots f_K} E \left(y - \sum_{k} \alpha_k f_k(x) \right)^2 .$$

Note the key difference, that neural networks optimize α and the base learners jointly,

instead of a two stage procedure.

3.4. Empirical Results

In this section we will discuss the results from a large scale classification study comparing the performance of a deep neural network and a random forest on a collection of 116 data sets from the UCI Machine Learning Repository. We also discuss empirical ensemble decompositions for a number of trained neural networks from binary classification tasks.

3.4.1. Data Summary

The collection of data sets we consider were first analyzed in a large-scale study comparing the accuracy of 147 different classifiers (Fernández-Delgado et al., 2014). This collection is salient for several reasons. First, Fernández-Delgado et al. (2014) found that random forests had the best accuracy of all the classifiers in the study (neural networks with many layers were not included). Thus, random forests can be considered a gold standard to which compare competing classifiers. Second, this collection of data sets presents a very different test bed from the usual image and speech data sets found in the neural network literature. In particular, these data sets span a wide variety of domains, including agriculture, credit scoring, health outcomes, ecology, and engineering applications, to name a few. Importantly, these data sets also reflect a number of realities found in data analysis in areas apart from computer science, such as highly imbalanced classes, non-trivial Bayes error rates, and discrete (categorical) features.

These data sets tend to have a small number of observations: the median number of training cases is 601, and the smallest data set has only 10 observations. It is interesting to note that these small sizes lead to highly overparameterized models:

	Categorical	Classes	Features	N
min	0	2	3	10
25%	0	2	8	208
50%	4	3	15	601
75%	8	6	33	2201
max	256	100	262	67557

Table 2: Dataset Summary

on average, each network as 155 parameters per training observation. The number of features ranges from 3 to 262, and half of data sets include categorical features. Finally, the number of classes ranges from 2 to 100. More detail can be found in Table 2.

Preprocessing

We will note here some of the most important preprocessing decisions. A more detailed discussion can be found in Fernández-Delgado et al. (2014). As is common in real world data sets, a number of data sets contain missing values. Missing values are simply replaced by zero. Although arbitrary, this decision shouldn't bias comparison of our classifiers. Ordinal variables are converted to a continuous scale, and all features are centered and standardized.

3.4.2. Experimental Setting

For each data set in our collection, we fit three classifiers: a random forest, and neural networks with and without dropout. Importantly, the training process was completely non-adaptive. One of our goals was to illustrate the extent to which deep neural networks can be used as "off-the-shelf" classifiers.

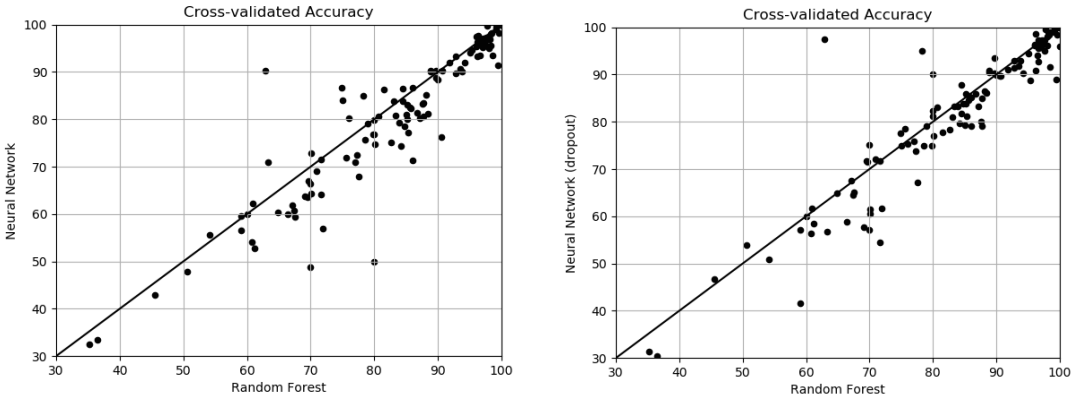


Figure 18: Plots of cross-validated accuracy for neural networks fit with and without dropout, and a random forest. Each plot point corresponds to a data set.

Implementation Details

Both networks shared the following architecture and training specifications:

- 10 hidden layers
- 100 nodes per layer
- 200 epochs of gradient descent using Adam optimizer with a learning rate of 0.001 (Kingma and Ba, 2014).
- He-initialization for each hidden layer (He et al., 2015)
- Elu activation function (Clevert et al., 2015).

Our choice of architecture was chosen simply to ensure that each network had the capacity to achieve perfect training accuracy in most cases. See Section 3.4.3 for observations about the training process.

The only difference between the networks involved the presence of explicit regularization. More specifically, one network was fit using dropout with a keep-rate of

0.85, while the other network was fit without explicit regularization. Dropout can be thought of as a ridge-type penalty is often used to mitigate over-fitting (Srivastava et al., 2014). There are other types of regularization not considered in this paper, including weight decay, early stopping, and max-norm constraints.

Each random forest was fit with 250 trees, using defaults known to work well in practice. In particular, in each training instance, the number of randomly chosen splits to consider at each tree node was $\sqrt{p}$, where p is the number of input variables. We used the random forest implementation found in the `sklearn` machine learning library (Pedregosa et al., 2011).

Evaluation Metrics

We evaluated each of the fits using 5-fold stratified cross-validation. The limited number of observations in some data sets excluded the possibility of more preferable training-test splits. We were chiefly concerned with measuring the accuracy of each classifier, but we also found it interesting to examine performance in terms of F_1 score, and area under the curve (AUC). Recall that the F_1 score is the harmonic mean between a classifier's precision and recall. The three measures we consider present a more complete view of the quality of a classifier's predictions.

3.4.3. Classifier Performance

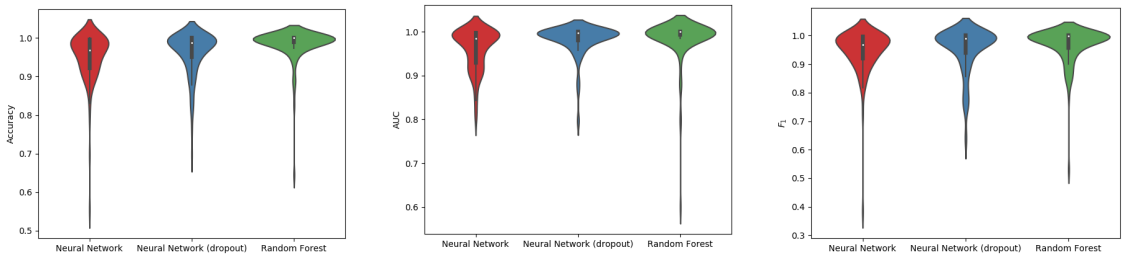
Accuracy

We turn first to Figure 18, which plots the cross-validated accuracy of the neural network classifiers and the random forest for each data set. In the first figure, we see that a random forest outperforms an unregularized neural network on 72 out of 117 data sets, although by a small margin. The mean difference in accuracy is 2.4%,

which is statistically significant at the 0.01 level by a Wilcoxon signed rank test. We notice that the gap between the two classifiers tends to be the smallest on data sets with low Bayes error rate - those points in the upper right hand portion of the plot. We also notice that there exists data sets for which either a random forest or a neural network significantly outperforms the other. For example, a neural network achieves an accuracy of 90.3% on the *monks-2* data set, compared to 62.9% for a random forest. Incidentally, the base-rate for the majority class is 65.0% percent in this data set, indicating that the random forest has completely overfit the data.

Turning to the second plot in Figure 18, we see that dropout improves the performance of the neural network relative to the random forest. The mean difference between classifiers is now decreased to 1.5%, which is still significant at the 0.01 level. The largest improvement in accuracy occurs in data sets for which the random forest achieved an accuracy of between 75% and 85%. It is also worth noting that the performance difference between the neural networks with and without dropout is less than one percent, and this difference is not statistically significant.

While it might not be surprising that explicit regularization helps when fitting noisy data sets, it *is* surprising that its absence does not lead to a complete collapse in performance. Neural networks with many layers are dramatically more expressive than shallow networks (Bengio et al., 2009), which suggests deeper networks should also be more susceptible to fitting noise when the Bayes error rate is non-zero. We find this is not the case. Furthermore, as we illustrated in Section 3.2, the additional capacity of deep neural networks may be directed mostly in a redundant manner, in which multiple components of a network learn essentially the same function.



(a) Accuracy (b) AUC (c) F_1 score

Figure 19: The distribution of relative performance (relative to the best possible over all classifiers) for each classifier over data sets.

Relationship of Data Set Properties with Accuracy

In addition to cataloging the performance differences between these classifiers, we were also interested in determining if any data set characteristics explained these gaps. The characteristics we examined were the number of training observations, the number of features, the number of classes to predict, and the base rates for each class. In order to assess the correspondence between data set characteristics and accuracy, we considered Spearman rank correlation.

We found that the performance gap between a random forest and an unregularized neural network was modestly negatively correlated with the number of training cases (at a 0.01 significance level). That is, the gap between random forests and neural networks decreases on smaller data sets. The magnitude of this correlation persists when considering the gap between neural networks fit with dropout and random forests, but is no longer significant. We also found that the only data set characteristic that was noticeably correlated with the difference between neural network accuracy with and without dropout was the number of classes: dropout networks tended to perform a bit worse on data sets with more classes.

We will briefly comment on the relative performance of these classifiers on the other

performance metrics mentioned in Section 3.4.2. To this end, Figure 19 plots the distribution of a classifier's performance as a fraction of the best performance across all three classifier's on a given data set. This allows a more meaningful comparison across data sets. As was the case for accuracy, we also find slight improvements in AUC and F_1 for neural networks fit with dropout. The random forest had a slight advantage over both neural networks across these metrics.

3.4.4. Ensemble Formation

We will now carry over the ensemble decomposition analysis from Section 3.2 to the binary classifiers fit in Section 3.4. Our analysis will exclude a number of data sets. First, since our ensemble analysis relies on a holdout set, we only considered data sets with more than 500 observations. This allows us to conduct an 80/20 train-test split of the data with a reasonable number of observations for a holdout sample. Secondly, of the 27 binary classification data sets with at least 500 observations, we remove 7 data sets either because the network was unable to achieve perfect training accuracy, or the number of observations was too large to solve the required linear program. As we mentioned in Section 3.2.2, *ensemble hunting* can also be applied in the multi-class setting at the expense of a slightly more complicated linear program, although we do not do so here. For each data set, we extract $K = 9$ sub-networks f_k such that each network fits the training data perfectly, and such that f_i and f_j are constructed from different parts of the network. Again, we remark that the fact that this is even possible to do is not obvious. If it were the case that each hidden unit in the final layer was highly specialized, such a decomposition would be impossible to find. Our choice of $K = 9$ was arbitrary, and in some cases a much larger value of K works just as well. In addition to a random forest with 90 trees, we also construct sub-ensembles of from 90 bagged CART trees of depth 6.

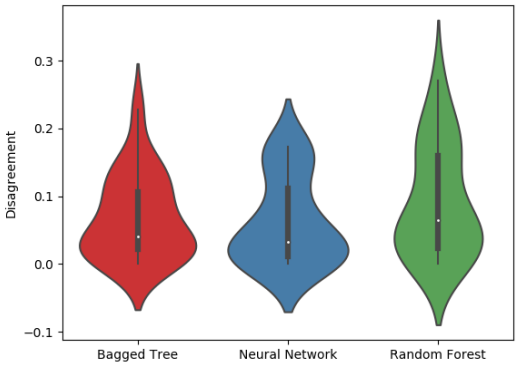


Figure 20: Distribution of average pairwise disagreement of ensemble members over 20 binary classification data sets.

Figure 20 plots the distribution of average pairwise disagreement of ensemble members across the 20 binary classification tasks. We first notice that the mean disagreement tends to be low across all classifiers. This is not entirely surprising, as these classifiers were able to achieve test accuracies exceeding 96% on at least half of these data sets, indicating that not too much label noise may be present. We do notice that the random forest tends to have the smallest ensemble disagreement. Again, this is not surprising since a random forest ensemble is explicitly constructed to have this property. The neural networks and bagged trees tend to have similar disagreement, with neural networks being slightly lower. We point out again that we were by no means exhaustive or optimal in our search for diverse sub-networks, so it is possible that much more diverse decompositions exist.

3.5. Discussion

We established that deep neural networks can generalize well on small, noisy data sets despite memorizing the training data. In order to explain this behavior, we offered a novel perspective on neural networks which views them through the lens of ensemble classifiers. Some commonly used wisdom when training neural networks is to choose an architecture which allows one sufficient capacity to fit the training data, and then

scale back with regularization (Bengio, 2012). Contrast this with the mantra of a random forest: fit the training data perfectly with very deep decision trees, and then rely on randomization and averaging for variance reduction. We have shown that the same mantra can be applied to a deep neural network. Rather than each layer presenting an ever increasing hierarchy of features, it is plausible that the final layers offer a bagging mechanism.

Future work can proceed along a number of dimensions. For one, it is certainly possible to improve the test accuracy of the networks fit in Section 3.4 through more extensive tuning. It may also even be possible to find a better set of default parameters to improve the results demonstrated in this paper. Our analysis is far from the last word on a comparison between random forests and neural networks. One might also improve our analysis in Section 3.2 by incorporating a loss function or better sparsity constraints that aid the search for a decomposition of the final hidden layer.

It is of more interest to analyze whether anything in the network's training procedure can explain the ensembling behavior in the last layer. The randomness present in stochastic gradient descent, or perhaps instability in the training procedure that arises when the loss function is driven near zero and gradients collapse might provide "jittering" to the final output layer. This "jittering" might provide the decorrelation effect hypothesized by Breiman (2000a) in the case of AdaBoost. To this end, one might consider modifications of the training process that involve injecting more randomness into the final layer near the end of training, in the spirit of dropout. As far as we can tell, there is nothing immediately obvious from the equations of backpropogation that suggest another mechanism.

CHAPTER 4 : Random Forest Probability Estimation

Abstract

A random forest is an increasingly popular tool for producing estimated probabilities in machine learning classification tasks. However, the means by which this is accomplished is unprincipled: one simply counts the fraction of trees in a forest that vote for a certain class. In this paper, we forge a connection between random forests and kernel regression. This places random forest probability estimation on more sound statistical footing. As part of our investigation, we develop a model for the proximity kernel and relate it to the geometry and sparsity of the estimation problem. We also provide intuition and recommendations for tuning a random forest to improve its probability estimates.

4.1. Introduction

In classification tasks, one is often interested in estimating the probability that an observation falls into a given class - the conditional class probability. These probabilities have numerous applications, including ranking, expected utility calculations, and classification with unequal costs. The standard approach to probability estimation in many areas of science relies on logistic regression. However, modern data sets with nonlinear or high dimensional structure, it is usually impossible to guarantee a logistic model is well-specified. In that case, resulting probability estimates may fail to be consistent (Kruppa et al., 2014). As a result, researchers have been relying

*Joint work with Adi Wyner.

more on machine learning and other nonparametric approaches to classification that make leaner assumptions.

Random forests have become a widely used tool in "black-box" probability estimation. This technique has been found to be successful in diverse areas such as medicine (Gurm et al., 2014), (Escobar et al., 2015), ecology (Evans et al., 2011), outcome forecasting in sports (Lock and Nettleton, 2014), and propensity score calculations in observational studies (Zhao et al., 2016), (Lee et al., 2010). First proposed in Breiman (2001), the random forests classifier consist of a collection of randomly grown trees whose final prediction is an aggregation of the predictions from individual trees. Random forests enjoy a number of properties that make them suitable in practice, such as trivially parallelizable implementations, adaptation to sparsity, and automatic variable selection, to name a few. The reader is well-advised to consult Biau and Scornet (2016) for an excellent review of recent research in this area.

After fitting a classification random forest to training data, it is common practice to infer conditional class probabilities for a test point by simply counting the fraction of trees in the forest that vote for a certain class. A priori, this is an unprincipled practice: the fraction of votes of classifiers in an ensemble need not have anything to do with a class probability. In the case when base classifiers in the ensemble are highly correlated - such as a collection of bagged tree stumps - the estimated probabilities will necessary converge to 0 or 1.

The next section contains a simulated example for which a random forest produces catastrophically poor probability estimates, yet still manages to obtain the Bayes error rate. These observations should not be surprising. Achieving a low misclassification error rate requires only that the classifier estimates one quantile well: the median. As long as 251 out of 500 trees vote for the correct class, the forest will achieve a low

test error rate. Probability estimation at every quantile simultaneously is obviously a much harder problem. What is surprising is that despite such ad hoc foundations, random forests do tend to produce good probability estimates in practice, perhaps after calibration (Niculescu-Mizil and Caruana, 2005). The goal of this paper is to put random forest probability estimates on more sound statistical footing in order to understand how a voted ensemble is able to estimate probabilities. Moreover, we will exploit this understanding to improve the quality of these estimates in the cases where they are poor.

4.1.1. Motivation

Random forests tend to be excellent classifiers under a wide range of parameter settings (Berk, 2008). The same robustness, however, is not enjoyed by a forest's probability estimates. In fact, it can sometimes be the case that such classifiers achieve the Bayes error rate while producing remarkably bad probability estimates. The following example motivates our analysis of random forest probabilities in the rest of the paper.

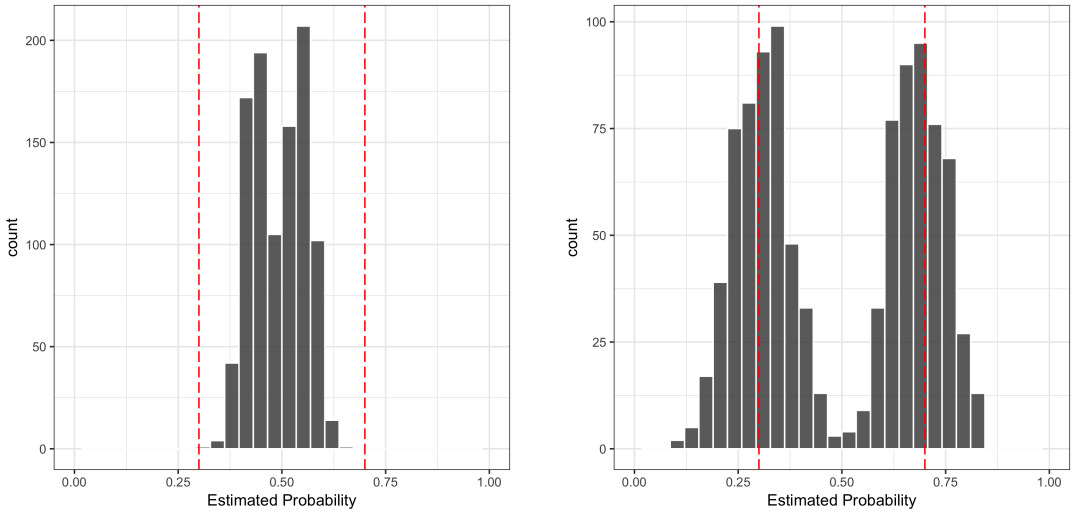
We consider a very simple model. First, draw $n = 1,000$ predictors uniformly at random from $[-1, 1]^{50}$, and then generate class labels according to the conditional class probability $\mathbb{P}(y = 1|x) = 0.3$ if $x_1 < 0$, and $\mathbb{P}(y = 1|x) = 0.7$ if $x_1 \geq 0$. The first dimension contains all of the signal, while the remaining 49 dimensions are noise. Note here that a simple tree stump would produce very good probability estimates. Fit to training data, the stump would split near $x_1 = 0$, and the training data that accumulated in each daughter node would have relative proportions of $y = 1$ labels in the way we would expect. The story is quite different for a random forest.

Figure 21 plots a histogram of estimated probabilities from a classification random

forest under two different parameter settings. The parameter that will concern us most in this paper is *mtry*, the number of randomly chosen candidate predictors for each tree node - the details are spelled out in Section 4.2. Since the population conditional class probability function only takes on two possible values, namely 0.3 and 0.7, we would ideally expect these histograms to consist of two point masses at these values. Figure 21a shows the results from using a random forest with $mtry = 1$, while Figure 21b shows the estimated probabilities when $mtry = 30$ [2]. When comparing these figures, the quality of probability estimates differs drastically. When $mtry = 1$, the probabilities are pushed toward the uninformative value of 0.5, while when $mtry = 30$, the probabilities are centered around their true values of 0.3 and 0.7. However, in both cases, the random forest achieves the Bayes error rate of 0.3! Each random forest is able to achieve similar (optimal) classification performance in terms of test error, but very different performance in probability space. A random forest can produce good probability estimates, but only when tuned properly.

In order to determine why this discrepancy in probability estimation quality differs, we will focus our efforts on the estimation at a single point $x_0 = (0.5, 0.5, \ldots, 0.5)$. Specifically, we will be concerned with the points in the training set that get used to make a prediction at x_0, which is the same to say, the training points that appear in the same terminal nodes of the forest trees as x_0. Such points are referred to in the literature as *voting points*, and were first studied by Lin and Jeon (2006). Figure 22 displays the voting points for the point x_0 projected in the (x_1, x_2) plane. Ideally, voting points should all lie in the half-space $x_1 \geq 0$, since these points all have the same conditional class probability as our target point x_0, and we would hope that a random forest would only consider "similar" points in making a prediction. When comparing

[2]Note that when $mtry = 1$, one randomly chosen predictor is considered at each split, so the trees in the forest are very weak.



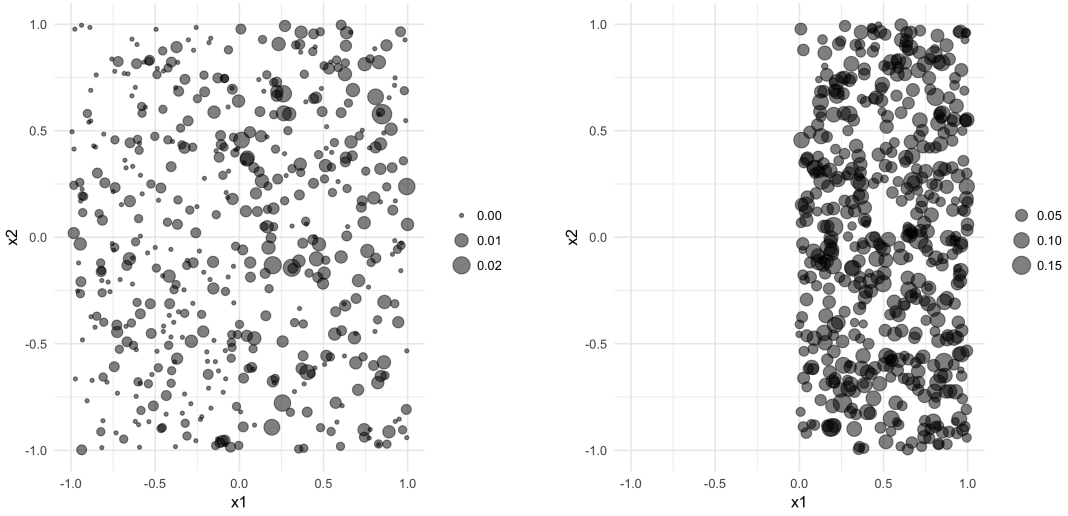
(a) $mtry = 1$ (b) $mtry = 30$

Figure 21: Histogram of estimated probabilities produced by a random forest under two different settings of *mtry*. We expect these probabilities to cluster around 0.3 and 0.7.

Figures 22a and 22b, it is clear that the forest with $mtry = 30$ concentrates all of its voting points in the correct neighborhood, while the forest with $mtry = 1$ does not. This example illustrates that the parameter *mtry* intuitively controls the "tightness" of voting neighborhoods: when *mtry* is large, these neighborhoods concentrate more tightly. We would like the reader to see the analogy to the bandwidth parameter in a kernel regression. This analogy is at the heart of the paper.

4.1.2. Outline

Our contribution is to frame random forest probability estimation in the framework of kernel regression, and to exploit this framework to understand how to use random forests to produce better probabilities. The explicit connection between random forests and kernel methods has recently appeared in Scornet (2016a), although this connection is implicit in earlier works of Leo Breiman (Breiman, 2000a), (Breiman, 2004). This work shows that regression random forests can be viewed as close cousins



(a) $mtry = 1$ (b) $mtry = 30$

Figure 22: Plots indicating the training points contributing to the random forest's prediction at $x_0 = (0.5, 0, \ldots, 0)$ projected on the (x_1, x_2) axis. The size of the plot point indicates the fraction of trees in the forest for which each point appears in the same terminal node as the target point x_0.

of kernel regression, in which the kernel function is the *proximity function*, which will be discussed in more depth later in the paper. We build on this work by studying the shape of the random forest kernel, especially as it relates to the forest's parameter settings. As a key tool to developing intuition about the behavior of this kernel, we develop an analytical approximation to the kernel of a simplified model of a random forest studied in other parts of the literature. This model allows us to bridge intuition about bandwidth selection in Nadaraya-Watson type kernel estimation to understand the role of parameter tuning in random forest probability estimation.

We will begin in Section 4.2 by providing more formal background on random forest probability estimation, as well as notation which will facilitate our discussion throughout the paper. Next, we will introduce the concept of the *proximity function* in Section 4.3, and we will relate random forests to kernel regression methods. This not only allows us to view random forest probabilities in a more principled way, but

also motivates discussion of the random forest proximity function as a fundamental quantity of interest in probability estimation. In order to better understand the shape of the random forest kernel and its connection to tuning parameters, we develop a kernel approximation to a simplified model of a random forest in Section 4.4. This will allow us to bridge our intuition about kernel regression to random forest probability estimation. We will then confirm the intuition developed from our simple model in Section 4.5, and will consider more extensive simulation experiments in Section 4.6.

4.2. Background

In this section, we will establish mathematical notation to facilitate of discussion in the rest of the paper, as well as provide background on the existing literature in probability estimation in random forests.

4.2.1. Setup and Notation

In the standard set-up for binary classification problems, we observe n pairs of training data points $(x_1, y_1), \ldots, (x_n, y_n)$ with $x_i \in \mathcal{X} \subset \mathbb{R}^p$ and $y_i \in \{0, 1\}$, and the goal is to learn a mapping from x_i to y_i. To make the problem more statistically tractable, it is often assumed that these pairs are independent and identically distributed, and that x and y are related according to an unknown conditional class probability function $\mathbb{P}(y = 1|x)$. The goal of the analyst in classification is simply to discriminate whether $\mathbb{P}(y = 1|x) \geq 0.5$ to predict the class of a new test point x. The related problem of directly estimating the probability of class membership $\mathbb{P}(y = 1|x)$ is much more difficult. This is the problem we consider in this paper in the context of random forests.

Recall that a random forest for classification consists of a collection of T un-pruned decision trees, where each tree is grown on a bootstrap sample of the data, and the

split variable at each node of the tree is chosen to be the best among a subset of size *mtry* randomly chosen predictors. The randomness introduced in each tree is governed by a random variable $\theta \in \Theta$ (i.e. the bootstrap sample that was chosen, as well as the candidate subset of random predictors at each node of the tree). The parameter θ will serve us primarily as a way to index the trees in the forest. Each decision tree partitions the input space into hyper-rectangles formed by the terminal nodes. We will denote the terminal node in the tree generated by $\theta \in \Theta$ to which a point x belongs by $\mathcal{R}_\theta(x)$, and we will denote the number of sample points in this node by $\mathrm{N}_\theta(x)$.

To simplify notation, we will assume that the bootstrap step is not used in the tree-growing process. With this caveat, we can define the prediction for a single tree at a single point x_0 by

$$f(\theta, x_0) = \sum_{i=1}^{n} \frac{\mathbb{I}(x_i \in \mathcal{R}_\theta(x_0))\, y_i}{\mathrm{N}_\theta(x_0)}.$$

In words, to make a prediction at a point x_0, we simply determine which terminal node that point belongs to, and then return the fraction of training points in that node for which $y = 1$. A random forest is constructed from a selection of independent random draws $\theta_1, \ldots, \theta_T$ and the associated trees $f(\theta_1, \cdot), \ldots, f(\theta_T, \cdot)$. We will consider two types of random forests for probability estimation, *classification forests*, denoted by $\mathrm{RF}^{\mathrm{class}}(\cdot)$ and *regression forests*, denoted by $\mathrm{RF}^{\mathrm{reg}}(\cdot)$. We can define these as

$$\mathrm{RF}^{\mathrm{reg}}(x_0) = \frac{1}{T} \sum_{t=1}^{T} f(\theta_t, x_0)$$

$$\mathrm{RF}^{\mathrm{class}}(x_0) = \frac{1}{T} \sum_{t=1}^{T} \mathrm{round}(f(\theta_t, x_0)).$$

In the case of a classification random forest, we estimate probabilities simply by

making a class prediction for each tree round $(f(\theta_t, x_0))$, and counting the fraction of trees that vote for a certain class. In practice, classification forest trees are often grown to a terminal node size of one. In later sections, we will be concerned with the extent to which $\text{RF}^{\text{class}}(x_0)$ and $\text{RF}^{\text{reg}}(x_0)$ approximate the conditional class probability $\mathbb{P}(y = 1|x_0)$. It is not apriori obvious that counting the fraction of trees that vote for a certain class, as in a classification random forest, is principled way to estimate this quantity.

4.2.2. Literature on Probability Estimation

There has been relatively little literature on probability estimation in random forests. It is known that classification random forests probabilities are typically uncalibrated, but produce among the best estimates among machine learning classifiers after calibration (Niculescu-Mizil and Caruana, 2005). Other research has investigated the usefulness of correcting probabilities in random forests using Laplace and m-estimates at the nodes (Bostrom, 2007). There also exists limited empirical evidence comparing the efficacy of regression and classification random forest probabilities in a number of simulated and real data settings (Li and Cutler, 2013).

Recently, researchers have argued that regression random forests are appropriate for probability estimation (Malley et al., 2012), (Kruppa et al., 2014). This claim follows from recent work showing that regression random forests are consistent in a number of settings (Scornet et al., 2015). Essentially, the argument is that if one passes the binary random variable Y in a regression forest, then consistency in conditional class probability follows since $\mathbb{E}(Y|X) = \mathbb{P}(Y = 1|X)$. We contend that this argument is too simplistic to explain how to obtain useful probability estimates in practice. For one, random forest probabilities are not always good, even if this theory says otherwise

[3]. Secondly, one must appreciate that classification and regression random forests are extremely similar, except for the default parameter settings and aggregation method. Indeed, we argue in the appendix that in the binary outcome case, the splitting criteria for regression trees is exactly the same as that for classification trees. We assert that performance differences in probability estimation simply result from differences in default parameter settings: regression random forests have a larger default setting for *mtry*, and in many cases this can account for any discrepancy in probability estimation performance. We will argue in the rest of the paper that a kernel perspective on probability estimation is more appropriate since it emphasizes that tuning parameters are critical in practice.

4.3. Probabilities from the Proximity Function

In this section we will introduce the concept of the proximity function and relate it to the task of probability estimation. We will begin by motivating the proximity function as a natural distance measure induced by a random forest, and we will briefly describe some of its traditional uses. Next, we will argue that the probability estimates produced through voting resemble those of kernel regression, where the kernel is given by the proximity function.

4.3.1. The Proximity Function

We will begin by defining a natural distance metric induced by a random forest, the *proximity function*.

Definition 1 (The Proximity Function)**.** *Grow a random forest according to* $\theta_1, \ldots, \theta_T$ *on a training set* $(x_1, y_1), \ldots, (x_n, y_n)$*. The proximity function is a mapping* $K_T : \mathcal{X} \times \mathcal{X} \to [0, 1]$ *where* $K_T(x, z) = \frac{1}{T} \sum_{t=1}^{T} \mathbb{I}\left(x \in \mathcal{R}_{\theta_t}(z)\right)$.

[3]The example from Section 4.1.1 has the same qualitative outcome if we use a regression random forest instead of a classification random forest, even with a very large training set.

In words, the proximity function simply measures the fraction of trees in a forest for which two test points appear in the same terminal node. One can view this quantity as a natural notion of similarity between two points: the more times these two points appear in the same terminal node of a tree, the more similar they are. In the limit of an infinite number of trees, it is also natural to consider a population version of the proximity function given by $K(x,z) = \mathbb{P}_\theta\left(z \in \mathcal{R}_z\left(\theta_t\right)\right)$. It is easy to argue from the law of large numbers that $K_T(x,z) \to K(x,z)$ as $T \to \infty$ almost surely (Breiman, 2000a). Since trees generate partitions of the input space, one also has the interpretation that the proximity function gives the probability that two points are in the the same cell of a randomly chosen partition. The notation K_T and K is also meant to be suggestive: both are self-adjoint, positive definite kernels that are bounded above by one (Breiman, 2000a).

The most common use of the proximity function is for clustering training data. To this end, one can define the *proximity matrix* $P \in \mathbb{R}^{n\times n}$ where $P_{i,j} = K_T\left(x_i, x_j\right)$. In other words, the proximity matrix contains all of the pairwise similarities among the training data. By the kernel properties of the proximity function, one can easily argue that the matrix $D \in \mathbb{R}^{n\times n}$, $D_{i,j} = 1 - P_{i,j}$ defines a Euclidean distance matrix. One can then appeal to multidimensional scaling to a find a lower dimensional representation of the data that approximately respects the similarity measure induced from the proximity function. In addition to clustering, the proximity function has an number of other creative applications including missing data imputation and outlier detection. See Berk (2008) for more discussion.

4.3.2. Proximity Probabilities

Given our motivation of the proximity function as a measure of similarity between points, it is reasonable to explore how we might use this function to create probability

estimates. To this end, let us first recall the form of Nadaraya-Watson kernel-weighted regression. For our purposes, a kernel function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}^+$ is a non-negative function such that $K(x, z)$ is large when x and z are close in some sense. In practice, one often considers parameterized families, such as the Gaussian family $K_\lambda(x, z) = \exp\left(\frac{-||x-z||_2^2}{2\lambda}\right)$. Here, λ is referred to as the bandwidth of the kernel, and controls the bias-variance trade-off in estimation by changing the size of the local neighborhood. A kernel regression estimate of conditional class probability is then given by

$$\mathbb{P}(y = 1|x_0) = \frac{\sum_{i=1}^N K(x_0, x_i)\, y_i}{\sum_{i=1}^N K(x_0, x_i)}.$$

If we let $w_i(x_0) = \frac{K(x_0,x_i)}{\sum_{i=1}^N K(x_0,x_i)}$, we can simply think of a kernel estimate as a weighted combination of training data labels, with weights proportional to the similarity between a given training point and the target points x_0, i.e. $\mathbb{P}(y = 1|x_0) = \sum_{i=1}^n w_i(x_0) y_i$.

We can also make a simple argument that kernel regression is an intuitively appealing method for estimating conditional class probabilities. Suppose we estimate marginal and conditional probabilities in the following manner: $\widehat{\mathbb{P}}(y = 1) = \frac{n_1}{n}$, $\widehat{\mathbb{P}}(x_0) = \frac{1}{n}\sum_{i=1}^n K_\lambda(x_0, x_i)$, $\widehat{\mathbb{P}}(x_0|y = 1) = \frac{1}{n_1}\sum_{i:y_i=1} K_\lambda(x_0, x_i)$, where n_1 is the number of training points in the data set for which $y_i = 1$. Notice that the last two expressions are just kernel density estimates of $\mathbb{P}(x_0|y = 1)$ and $\mathbb{P}(x_0)$, respectively. We can then write an estimate for the conditional class probability using Bayes rule, plugging in kernel density estimates for the appropriate quantities as follows:

$$\begin{aligned}
\mathbb{P}\left(y=1|x_0\right) &= \frac{\mathbb{P}\left(y=1\right)\mathbb{P}\left(x_0|y=1\right)}{\mathbb{P}\left(x_0\right)} \\
&\approx \frac{\widehat{\mathbb{P}}\left(y=1\right)\widehat{\mathbb{P}}\left(x_0|y=1\right)}{\widehat{\mathbb{P}}\left(x_0\right)} \\
&= \frac{\frac{n_1}{n}\frac{1}{n_1}\sum_{i:y_i=1} K_\lambda\left(x_0,x_i\right)}{\frac{1}{n}\sum_{i=1}^{n} K_\lambda\left(x_0,x_i\right)} \\
&= \frac{\sum_{i=1}^{n} K_\lambda\left(x_0,x_i\right) y_i}{\sum_{i=1}^{n} K_\lambda\left(x_0,x_i\right)}.
\end{aligned}$$

We will now make a connection between the probabilities generated from voting and the proximity function. Note that the connection between regression random forests and kernel regression was pointed out in Scornet (2016a). Our focus here is to cast probability estimation in this light in order to explain the role of random forest parameters in controlling the quality of probability estimates, and we believe it is most natural to do this in the kernel regression setting. In later sections, we will devote effort to explaining how tuning parameters such as *mtry* and the number of nodes in each tree act as bandwidth parameters for this estimate.

Recall from Section 4.2 that a random forest estimate of probabilities is given by $\mathbb{P}\left(y=1|x_0\right) = \frac{1}{T}\sum_{t=1}^{T} f\left(x_0,\theta_t\right)$. We can plug in the definition of $f\left(x_0,\theta_t\right)$ to re-write this estimate in a more enlightening form:

$$\begin{aligned}
\mathbb{P}\left(y=1|x_0\right) &= \frac{1}{T}\sum_{t=1}^{T}\sum_{i=1}^{n}\frac{\mathbb{I}\left(x_i \in \mathcal{R}_{\theta_t}\left(x_0\right)\right) y_i}{\mathrm{N}_{\theta_t}\left(x_0\right)} \\
&= \sum_{i=1}^{n} y_i \left(\frac{1}{T}\sum_{t=1}^{T}\frac{\mathbb{I}\left(x_i \in \mathcal{R}_{\theta_t}\left(x_0\right)\right)}{\mathrm{N}_{\theta_t}\left(x_0\right)}\right) \\
&\approx \sum_{i=1}^{n} y_i \left(\frac{\sum_{t=1}^{T}\mathbb{I}\left(x_i \in \mathcal{R}_{\theta_t}\left(x_0\right)\right)}{\sum_{t=1}^{T}\mathrm{N}_{\theta_t}\left(x_0\right)}\right) \\
&= \sum_{i=1}^{n}\frac{K(x_0,x_i)y_i}{\sum_{i=1}^{n}K(x_0,x_i)}.
\end{aligned}$$

We will refer to the quantity in the last step as the *proximity probability* estimator of conditional class probability.

Definition 2 (Proximity Probabilities)**.** *The proximity probability estimate of conditional class probability at a test point x_0 is defined to be*

$$RF^{prox}\left(x_0\right) \equiv \sum_{i=1}^{n}\frac{K(x_0,x_i)y_i}{\sum_{i=1}^{n}K(x_0,x_i)}.$$

In the third step, we use the approximation that $\frac{1}{T}\sum_{t=1}^{T}\frac{\mathrm{a}_t}{\mathrm{b}_t} \approx \frac{\frac{1}{T}\sum_{t=1}^{T}\mathrm{a}_t}{\frac{1}{T}\sum_{t=1}^{T}\mathrm{b}_t}$ for any positive sequence of numbers a_t and b_t [4]. It is interesting to consider this approximation in the context of positive random variables X and Y: the approximation is merely assuming that $\mathbb{E}\left(\frac{X}{Y}\right) \approx \frac{\mathbb{E}(X)}{\mathbb{E}(Y)}$. Note that we simply use this approximation as intuition

[4] We also use the fact that

$$\begin{aligned}
\frac{1}{T}\sum_{t=1}^{T}\mathrm{N}_{\theta_t}\left(x_0\right) &= \frac{1}{T}\sum_{t=1}^{T}\sum_{i=1}^{n}\mathbb{I}\left(x_i \in \mathcal{R}_{\theta_t}\left(x_0\right)\right) \\
&= \sum_{i=1}^{n}\frac{1}{T}\sum_{t=1}^{T}\mathbb{I}\left(x_i \in \mathcal{R}_{\theta_t}\left(x_0\right)\right) \\
&= \sum_{i=1}^{n} K\left(x_0,x_i\right)
\end{aligned}$$

.

for motivating a different way of aggregating probabilities. In particular we make no claims about the quality of this approximation. When this is the case, $\mathrm{RF}^{\mathrm{prox}}(x_0)$ will be similar to the estimated probability estimated in the usual way.

4.3.3. Intuition

We will close this section with a few final thoughts on the similarities and differences between the different ways of using a random forest for probability estimation. First, it will be helpful to define a quantity which gives a conditional class probability estimate for a point x_0 for a given tree generated by θ_t: $p_t(x_0) = \sum_{i=1}^{n} \frac{\mathbb{I}\left(x_i \in \mathcal{R}_{\theta_t}(x_0)\right) y_i}{\mathrm{N}_{\theta_t}(x_0)}$. With this notation in hand, we can give three expression for probability estimates generated by a regression, classification, and proximity random forest:

$$\begin{aligned}
\mathrm{RF}^{\mathrm{reg}}(x_0) &= \frac{1}{T}\sum_{t=1}^{T} p_t(x_0) \\
\mathrm{RF}^{\mathrm{prox}}(x_0) &= \sum_{t=1}^{T} \frac{\mathrm{N}_{\theta_t}(x_0)}{\sum_{t'=1}^{T} \mathrm{N}_{\theta_{t'}}(x_0)} p_t(x_0) \\
\mathrm{RF}^{\mathrm{class}}(x_0) &= \frac{1}{T}\sum_{t=1}^{T} \mathrm{round}\left(p_t(x_0)\right).
\end{aligned}$$

First, one should note that in classification setting, the typical default is to grow a tree to node purity, so $\mathrm{RF}^{\mathrm{prox}}(x_0)$ agrees with $\mathrm{RF}^{\mathrm{class}}(x_0)$ in this case. In the case where nodes are not grown to purity, there is an interesting difference between the way regression random forest probabilities and proximity probabilities differ. In particular, regression random forests average node probabilities across trees with equal weights, while the proximity probability averages probabilities with weights proportional to the number of points in each node. Stated in another way, $\mathrm{RF}^{\mathrm{prox}}(x_0)$ forms a probability estimate by collecting all points in the training set (with multiplicity) that share a terminal node with the target point, and taking a grand mean. If one really does

believe that $p_t(x_0)$ is an estimate of a conditional class probability, this might be a reasonable strategy when some nodes have small counts. The form of $\text{RF}^{\text{class}}(x_0)$ seems to suggest that it is unlikely that $p_t(x_0)$ do in fact estimate probabilities: if we consider $p_t(x_0)$ to be a random variable with expectation $\mathbb{P}(y = 1|x_0)$ and very small variance, $\text{RF}^{\text{class}}(x_0)$ would tend to produce estimates that are all close to 0 or 1.

4.4. Kernel Intuition

We argued in the previous section that random forest probabilities can be fruitfully viewed from a kernel regression point of view. Central to kernel regression is the shape of the kernel and associated bandwidth parameter. Unfortunately, the mathematics of the original random forest algorithm forest are complicated and make analytical expressions for the kernel intractable. In this section, we derive the kernel in a simplified setting considered in Breiman (2000a), Breiman (2004), Biau (2012). In particular, we demonstrate that a weighted Laplacian kernel is a good analogy to a random forest, and we leverage this analogy to provide intuition for how the parameter choice in a random forest informs the quality of its probability estimates.

4.4.1. A Naive Model of a Random Forest Kernel

While the mechanics are quite simple, the mathematics of a random forest make analytical expressions for quantities of interest intractable. As a result, the literature tends to focus on stylized models that make analysis more amenable. In this vein, we construct a simplified model of a random forest by borrowing elements from models considered in Breiman (2000a), Breiman (2004), Biau (2012). Closed for expressions for other random forest models have been considered elsewhere, but our model contains richer elements that more closely capture adaptive splitting (Scornet, 2016a). As we will demonstrate, adaptive splitting is key to constructing a kernel which adapts

to the shape and sparsity of the conditional class probability function.

Algorithm 3 Simplified Random Forest Tree

1. Specify the number of leafs M; initialize $\texttt{leafs} = \{\mathtt{t}_{root}\}$.
2. For $m = 1 : M$:
   (a) Select a terminal node $\mathtt{t} \in \texttt{leafs}$ uniformly at random.
   (b) Split $\mathtt{t}$ into daughter nodes $\mathtt{t}_L$, $\mathtt{t}_R$
      (i) Choose *mtry* predictors at random $\mathcal{F} \subseteq \{1, \ldots, p\}$
      (ii) Select split variable uniformly among the S_* signal variables in $\mathcal{F}$
      (iii) Choose split point uniformly at random
   (c) Replace $\mathtt{t}$ with $\mathtt{t}_L$ and $\mathtt{t}_R$ in $\texttt{leafs}$

The simplified model we will consider is given in Algorithm 3. To begin, we assume that the domain of the predictor variables is $\mathcal{X} = [0, 1]^p$. Of the p predictors, S of them are related to the response variable, and we call such predictors *strong*, and the remaining W predictors are unrelated to the response, and we call these weak. One may equivalently call strong variables "signal" variables, and weak variables "noise" variables. We fix a number M of tree nodes before tree growing begins. At each stage of the growing process, we select a node at random to split on, and randomly select *mtry* predictors. Among the *mtry* variables chosen, we split only on the strong ones with equal probability (unless only weak variables are chosen). This mechanism mimics adaptivity in the forest, and one can indeed verify in simulated examples that the predictors which are related to the response do indeed tend to be split on with higher probability (Biau, 2012). Finally, the chosen predictor is split on uniformly at random.

Note that the above algorithm gives a probability model for a fixed tree in the forest. We are interested in the probability that such a randomly drawn tree contains two fixed points in one of its terminal nodes, which is precisely the interpretation for a random forest proximity kernel. In this setting, we can derive an approximate

expression for the proximity kernel, which is given in the following proposition.

Proposition 3. *Suppose a random forest is grown to a size of M nodes. Under the above setting, we can derive an approximation for the proximity function $K(0,x)$:*

$$K(0,x) \approx \exp\left\{-\log M\left(p_{\mathcal{S}}\sum_{s\in\mathcal{S}} x_s + p_{\mathcal{W}}\sum_{w\in\mathcal{W}} x_w\right)\right\}$$

where

$$p_{\mathcal{S}} = \sum_{k=1}^{S\wedge mtry} \frac{\binom{S-1}{k-1}\binom{W}{mtry-k+1}}{k\binom{p}{mtry}}$$
$$p_{\mathcal{W}} = \frac{1-Sp_{\mathcal{S}}}{W}.$$

Proof. (See Appendix 4.8.2) □

It is clear that even in this simple setting, the kernel will clearly not be translation invariant. However, one can make the further approximation, as in Scornet (2016a), that $K(x,z) \approx K(0,|x-z|)$.

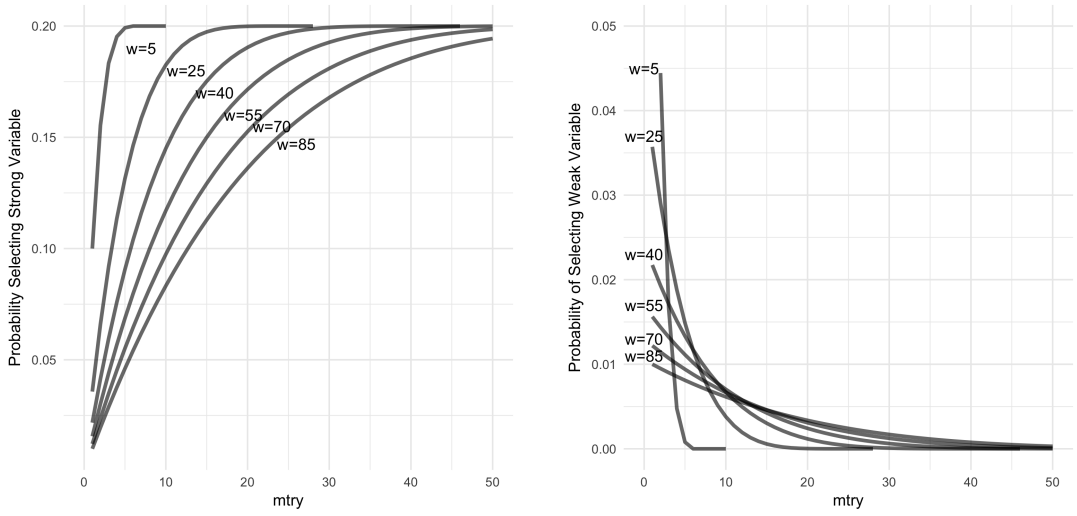
In a 2000 paper, Breiman derived an approximation to the proximity function in a simplified model in which predictors were selected uniformly at random at each stage of the growing process, with no distinction between "strong" and "weak" variables (Breiman, 2000a). The form of this kernel was correspondingly more simple, and took the form of a symmetric Laplacian density: $\exp\left(-\frac{\log M}{p}||x-z||_1\right)$. The advantage of our formulation is that it is evident that variables receive different importance weights in the proximity function. We will demonstrate in Section 4.5 that the proximity function does tend to adapt to the shape of the underlying conditional class probability function, especially in regards to sparsity. In our model, the mechanism by which this adaptation takes place is through weights p_s and p_w from Proposition 3.

In particular, the kernel tends to be more narrow in the direction of "strong" variables and flatter in the direction of "weak" variables. Correspondingly, our model captures this phenomenon by assigning higher weights to signal variables than weak variables, and the relative widths are controlled by the *mtry* parameter. It is also possible to change the shape of the kernel by assigning different weights to different strong variables, but makes the analysis a bit more tedious.

4.4.2. Naive Kernel Parameters

In this section, we will discuss how the value of *mtry* and the number of tree nodes affects the shape of the kernel derived in the previous section. The values of p_s and p_w are visualized as a function of *mtry* in Figure 23. We fix the number of strong variables as $S = 5$, and plot the weights for varying numbers of noise variables W. Notice first that for any value of W, the weight for strong variables p_S is a strictly increasing function of *mtry* and the weight for weak variables p_w is a strictly decreasing function of *mtry*. Furthermore, for large enough values of *mtry*, one can prove the weights for strong variables asymptote to $1/S$, and the weights for weak values converge to zero. These qualitative findings accord with what we would expect in a random forest. When *mtry* is large, the selection of candidate predictors likely contains a large number of signal variables, and such variables are preferred by tree splitting criteria since they lead to more pure daughter nodes. The consequence of a larger weight on such variables is that the kernel is more narrow in directions of signal, which again accords with what we will empirically demonstrate in Section 4.5. Furthermore, we can see that the value of *mtry* needed to filter out noise variables increases with the number of weak variables.

The number of terminal nodes also affects the shape of the proximity function derived in the previous section. In particular, a larger number of nodes M shrinks the diam-



(a) (b)

Figure 23: The plots above show the probability of selecting strong and weak variables as a function of *mtry* in the naive model. In both figures, the number of strong variables is fixed at 5, while the number of weak variable ranges from 5 to 85.

eter of the kernel by a multiplicative factor of $\log M$, making it more concentrated. In our model, the number of nodes M is a predetermined parameter, but in practice it depends on a number of characteristics of the data, such as the size of the data set and the value of *mtry*. For a data set of fixed size n, the number of tree nodes tends to decrease with larger values of *mtry*. The intuition is simple: an increase in *mtry* increases the number of signal variables appearing at each split, producing more shallow trees. Figure 24 shows a plot of the average number of nodes in a random forest tree for a simulated example as a function of *mtry*, which confirms our intuition.

We can summarize the main conclusions from our simplified model as follows:

- The value of *mtry* controls the relative width of the kernel in signal and noise dimensions.
- An increase in the number of nodes in the tree shrinks the kernel equally in all directions.

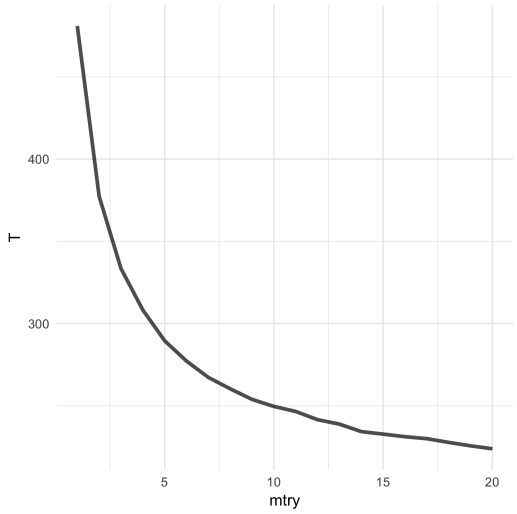


Figure 24: The number of terminal nodes as a function of *mtry* for a simulated example.

- Larger values of *mtry* are needed to sufficiently "zero out" the weight of noise variables in high dimensions.

4.4.3. Laplace Kernel Regression

Given the form of the kernel we derived in Section 4.4.2 and the implications for its parameters explored in Section 4.4.2, we now develop some intuition for a optimal choices for kernel regression with a weighted Laplace kernel. The idea here is to explore the role of kernel weights in producing good probability estimates, and then to tie these weights to random forest parameters. In kernel regression, one typically selects a kernel before looking at the data. We would like to demonstrate that a kernel that adapts to the data is much more powerful, and is in fact essential in higher dimensions.

We will consider here kernels of the form

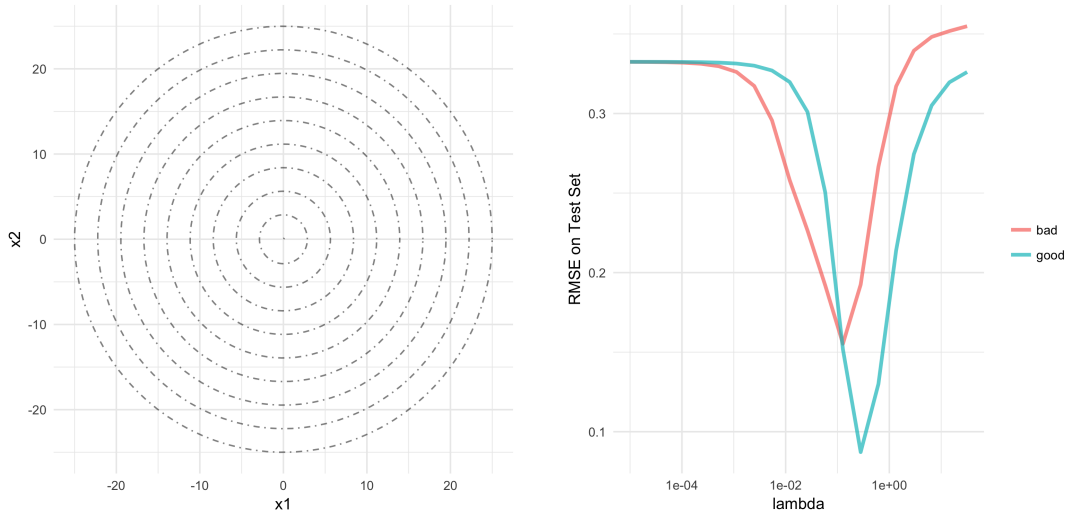
$$K_{\lambda,w}(x,z) = \exp\{-\lambda \sum_{j=1}^{p} w_j |x_j - z_j|\}$$

where w_i are non-negative weighs summing to one. The kernel weights w_i determine the shape of the kernel, and λ controls the concentration. It is clear that in the case of infinite data, the kernel weights do not matter - however, this is not the case in finite data. An appropriately shaped kernel can drastically affect the quality of probability estimates.

In the following example, we will compare conditional probability estimates using kernels with different shapes. We will begin by drawing $n = 500$ points uniformly at random from the square $[-25, 25]^2$, and then a label $y \in \{0, 1\}$ with the conditional probability

$$\mathbb{P}(y = 1|x) = \begin{cases} 1 & \text{if } r(x) < 8 \\ \frac{28 - r(x)}{20} & \text{if } 8 \geq r(x) \geq 28 \\ 0 & \text{if } r(x) \geq 1. \end{cases} \tag{4.1}$$

where $r(x) = \sqrt{(x_1^2 + x_2^2)}$. This is the "circle model" from Mease et al. (2007). Note that the level sets of the conditional class probability model are simply concentric circles, which are shown in Figure 25. We consider probability estimation with two different kernels. The first is $K_{\lambda,(1,1)}(x, z) = \exp(-\lambda(|x_1 - z_1| + |x_2 - z_2|))$ and the second is $K_{\lambda,(10,1)}(x, z) = \exp(-\lambda(10|x_1 - z_1| + |x_2 - z_2|))$. Note that the first kernel is has symmetric level sets - which is well-suited for this problem - and the second has level sets which are very skewed in the x_1 direction. In Figure 25 we plot the RMSE for probability estimation on a holdout set using each of these kernels for various values of the bandwidth parameter λ. The best RMSE achieved by the symmetric kernel is 0.07, while the best achieved by the skewed kernel is 0.15. This comes as no surprise: with a finite amount of data, the shape of the kernel is critical. Interestingly, both of these estimators achieve the same misclassification rate when used for class estimation. We would again like to emphasize the point that misclassification is



(a) (b)

Figure 25: The plot on the left shows the level sets for the conditional class probability function. The plot on the right shows the out of sample root mean squared error for probability estimation for two different kernels. The red line corresponds to a kernel with poor shape relative to the underlying probability density function, while the blue line shows a well-suited kernel.

generally a much more forgiving task than probability estimation: tuning matters!

We will close this section with a final point about kernel shape and sparsity. Suppose we observe n training points (x_i, y_i), where $x \in [0,1]^p$ and $y \in \{0,1\}$ with $\mathbb{P}(y = 1|x)$ a function only of the first coordinate. Our goal will be to estimate the probability that $y = 1$ at some point x_0. In order to do this, we will consider counting the fraction of points in neighborhoods of x_0 that have volume $0 < s < 1$, but different shapes - for simplicity we will consider hyperrectangles with different side lengths. If we consider a square, the side lengths of a neighborhood that capture a fraction s of the training data will have expected side length $s^{1/p}$ in the first coordinate. Now, consider a rectangle which has side lengths in the ratio of $\lambda : 1 : \cdots : 1$. The expected side length in the first coordinate to capture a fraction s of training points is now $\lambda \left(\frac{s}{\lambda}\right)^{1/p}$. Thus, the ratio of requires lengths in the signal direction between the square

and rectangle is $\lambda^{1/p-1}$. When $\lambda < 1$, that is, when the rectangle is more narrow in the signal direction, the square requires about $1/\lambda$ times more data points to estimate the probability with the same precision as the rectangle.

4.5. Empirical Properties of the Proximity Function

We will now present some simulated examples to demonstrate some of the qualitative findings from the previous section. In particular, we will show empirically that the random forest proximity matrix adapts to level sets of conditional class probability functions, and sparsity. These findings should resonate with the intuition for probability estimation presented in Section 4.4.3.

4.5.1. Adaptation to CCPF Shape

We will begin by illustrating that the proximity function's shape reflects the local geometry of the conditional class probability function. While not implied directly by Proposition 3, one could easily modify the Proposition's assumptions to reflect preferential splitting among the strong variables. This is future work, but we find it to be enlightening to present the current material to the extent that it reflects our discussion in Section 4.4.3, and has direct implications for the quality of probability estimation. Furthermore, in the next three examples we include a *completely random forest* as a straw man for which to compare the actual random forest algorithm. The *completely random forest* is nonadaptive: it operates exactly as a random forest, except the predictor to split on and split value are chosen uniformly at random. The proximity function derived from this algorithm is most similar to the one suggested in Breiman's analysis mentioned in Section 4.4.2.

Our first example consists of a piecewise logistic model, where the form of the class probability is given by Equation 4.2. On the domain $x_1 < 0$, the probability model

places more weight on the first coordinate, while on the domain $x_2 \geq 0$ the model places more weight on the second coordinate. The structural break at x_0 will allow us the opportunity to investigate the extent to which the random forest locally adapts to changes in the shape of the probability function. We draw $n = 2,000$ points on the square $[-1, 1]^2$, and we draw a class label $y \in \{0, 1\}$ according to Equation 4.2.

$$\mathbb{P}(y = 1|x) = \begin{cases} \frac{1}{1+\exp(-3x_1 - x_2)} & \text{if } x_1 < 0 \\ \frac{1}{1+\exp(-x_1 - 3x_2)} & \text{if } x_1 \geq 0 \end{cases} \tag{4.2}$$

In Figure 26 we plot level sets for the random forest and completely random forest proximity functions at different points. In particular, the top set of figures show level sets for the random forest, and the bottom set of figures show level sets for the completely random forest. The left set of figures shows level sets centered at $x_0 = (-0.25, -0.25)$, and the right set of figures shows level sets centered at $x_0 = (0.25, 0.25)$. Let us first turn our attention to Figure 26a. Since the first coordinate of the target point $x_0 = (-0.25, -0.25)$ is negative, the probability model places more weight on the first coordinate. Correspondingly, we see that the level set for the random forest kernel is most narrow in the direction of the first coordinate. Symmetrically, we see that the kernel is most narrow in the second coordinate in Figure 26b, in which the kernel is centered at $x_0 = (0.25, 0.25)$. When parsing these findings, it is good to keep in mind the circle example from Section 4.4.3 in which we demonstrated the importance of the shape of the kernel.

One should compare the shape of the random forest kernel in each case with that of the *completely random forest*, which is plotted in the bottom set of figures in Figure 26. At both test points, the level sets are symmetric, regardless of the shape

of the class probability function. Of course, this is not surprising: the *completely random forest* is nonadaptive by design. Finally, averaged over 50 replications, the average difference in root mean square error between the *completely random forest* and the random forest (using the kernel approach) at the point $x_0 = (-0.25, -0.25)$ is 0.002, while at $x_0 = (0.25, 0.25)$ is 0.007.

4.5.2. Adaptation to Sparsity

In our naive kernel model from Section 4.4.2 we argued that a random forest attached greater weights to signal variables than noise variables, and it is through this mechanism that it adapts to sparsity. A random forest's ability to adapt to sparsity has been argued in other places in the literature, as well as from a good track record of prediction in high dimensional problems (Biau, 2012), (Scornet et al., 2015), (Díaz-Uriarte and De Andres, 2006). We would like to illustrate through two examples that this adaptivity is reflected in the shape of the proximity function, which is narrow in directions of signal and flat in directions of noise.

Our first example is a simple logistic regression model, which we choose for visual ease. We draw $n = 500$ predictors $\mathbf{x}$ uniformly at random from the square $[-1, 1]^2$, and then we draw a label $y \in \{0, 1\}$ according to the probability

$$\mathbb{P}(y = 1|x) = \frac{1}{1 + \exp\{-3x_1\}}.$$

Note that only the first coordinate of the predictor x matters: the second coordinate is a noise variable. In this setting, when considering neighborhoods of a point x_0 to use for probability estimation, the best neighborhoods consist rectangles that are much more thin in the x_1 direction than the x_2 direction. In Figure 27, we plot the level curves for the proximity function centered at $x_0 = (0, 0)$ using a random forest

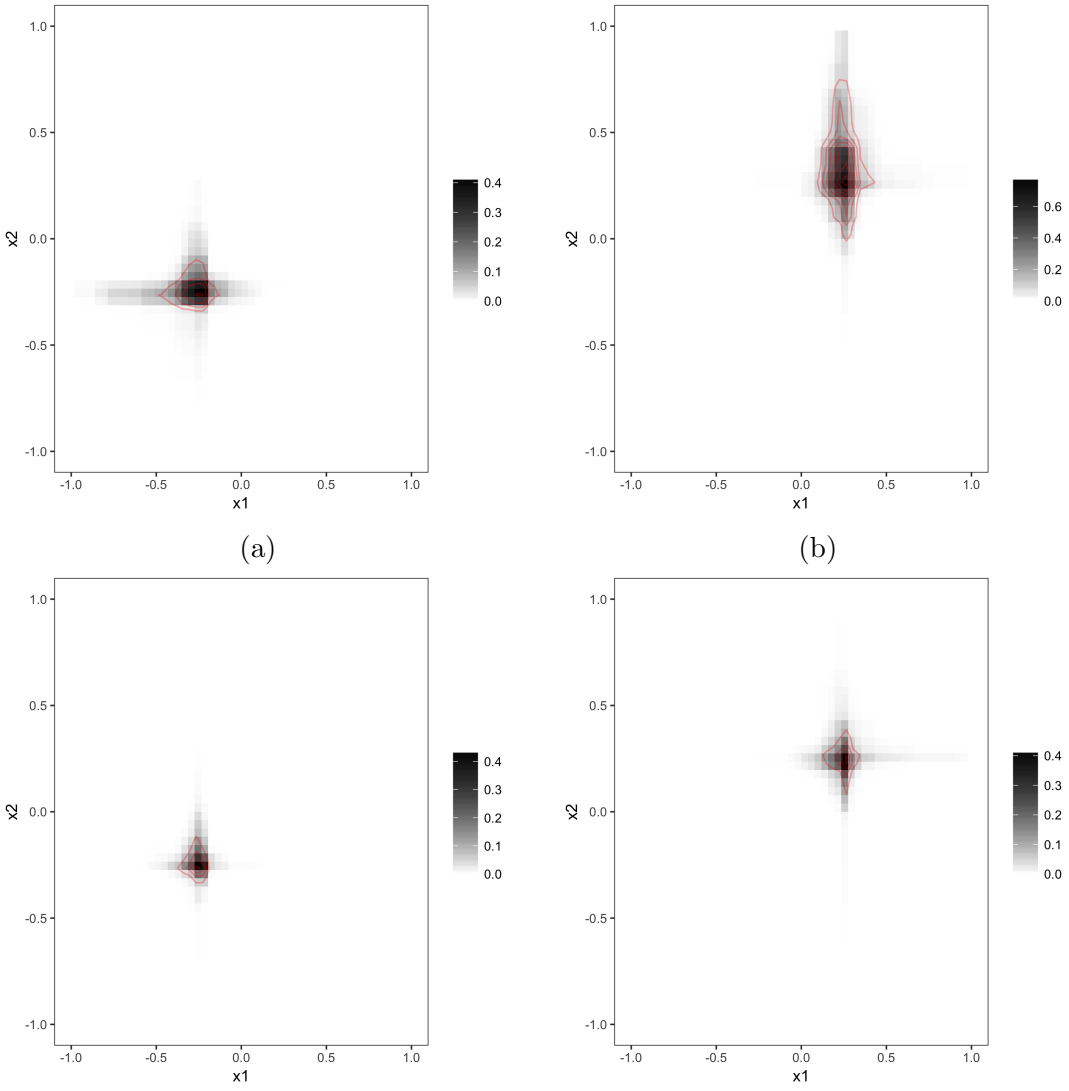
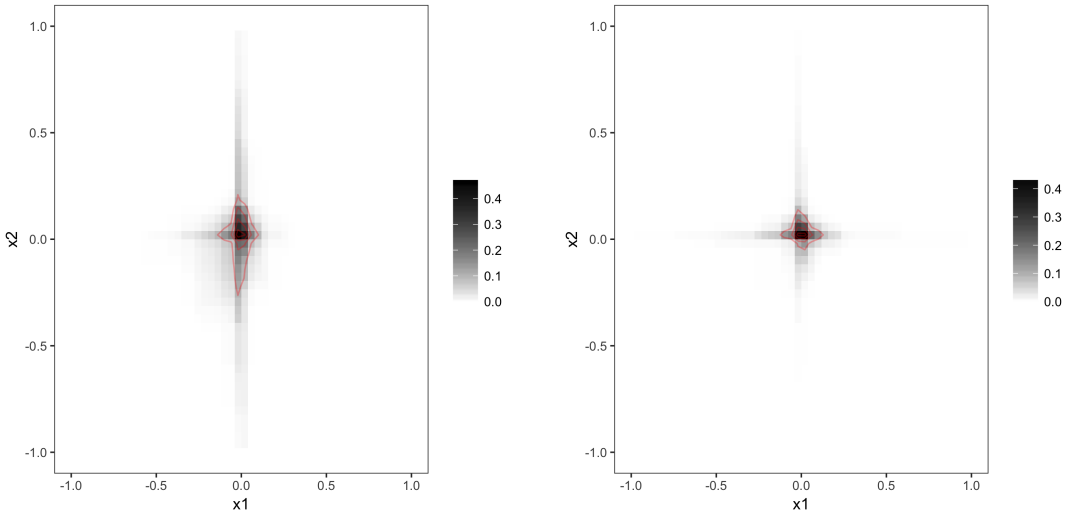


Figure 26: Level sets for the estimated probability density at a point x_0 for both random forests (sub-figures (a) and (b)) and completely random forests (sub-figures (c) and (d)). Note that in the left set of figures, the target point is $x_0 = (-0.25, -0.25)$, while in the right set of figures $x_0 = (0.25, 0.25)$. It is clear that that the level sets produced by the completely random forests are much more symmetric than the usual random forest.



(a) Random forest (b) Completely random forest

Figure 27: Level sets for the estimated probability density at the point $x_0 = (0, 0)$. The level set for the random forest stretches out in the signal dimension x_1, while the level set for the completely random forest is symmetric.

and a *completely random forest* in the left and right plots, respectively. The random forest proximity function has the shape we would expect at the origin: a thin vertical strip. On the other hand, the *completely random forest* has a neighborhood that is symmetric in both the x_1 and x_2 directions. In higher dimensions, one pays a price for this symmetry, as will be explored in more extensive simulation results in Section 4.6.

In our next example, we continue using a logistic model, but with a greater number of noise variables. We draw $n = 1,000$ predictors $\mathbf{x}$ uniformly at random from $[-1, 1]^{22}$, and conditional on $\mathbf{x}$, we draw y with probability

$$\mathbb{P}(y = 1|x) = \frac{1}{1 + \exp\{-2x_1 - 2x_2\}}.$$

Notice that only the first two dimensions contain signal about the class label y. The remaining 20 coordinates consist of noise variables. Unlike the previous example, the level sets of the proximity function are difficult to visualize, so we instead investigate

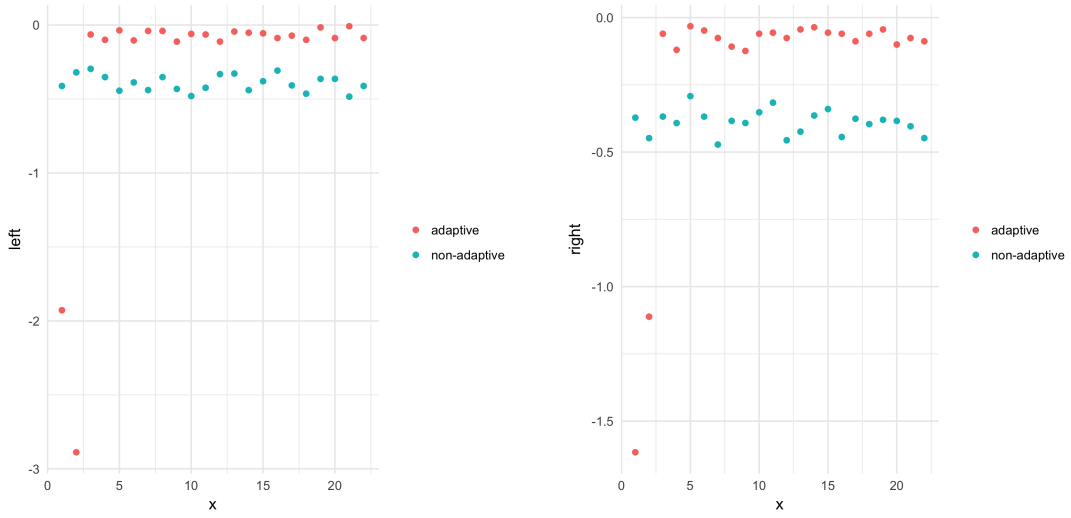
approximate directional derivatives of the proximity function centered at the origin

$$D_{j+}K(0,0) \approx \frac{K(he_j,0) - K(0,0)}{h}$$
$$D_{j-}K(0,0) \approx \frac{K(-he_j,0) - K(0,0)}{h}$$

where we take $h = 0.25$. Figure 28 plots the directional derivatives for each of the 22 directions in predictor domain. The points in red are the values of the derivative for the random forest, and those in blue are the values for the *completely random forest*. Turning our attention first to the red points in Figure 28a, the derivatives in the the x_1 and x_2 directions for the random forest kernel are -1.9 and -2.85, respectively, while the derivatives in the other dimensions are close to zero. In other words, the kernel is very peaked in the signal dimensions, and very flat in the noise dimensions. On the other hand, the directional derivatives for the *completely random forest* kernel are all approximately the same, indicating a symmetric, flat shape. The same phenomenon holds when analyzing left hand derivatives in Figure 28b. It further holds that as we increase the value of *mtry*, the kernel becomes increasingly peaked in the directions of signal as predicted by our naive model. Analyzing such directional derivatives might be an interesting alternative to variable importance type measures in a random forest.

4.5.3. Kernel Shape: the Spam Data Example

In Section 4.4 we proposed a simple model of a random forest kernel that connected the shape of the kernel to the *mtry* parameter. Specifically, we argued that as *mtry* increased, the kernel became more concentrated on signal variables, and more *flat* in noise dimensions. This intuition was also reflected on our motivating example in Section 4.1.1. Here, we will provide an illustration of this point on the *spam* dataset.



(a) Left derivative (b) Right derivative

Figure 28: Plots of the left and right directional derivatives of the estimated probability density function produced by random forests (adaptive) and completely random forests (non-adaptive).

The *spam* data set consists of $n = 4601$ emails, along with $p = 57$ predictors, with each predictor giving the frequency of certain words in that email. Each example is attached with a label indicating whether than email is spam or not. We are interested in determining the shape of the random forest kernel in the direction of noise variables. Since we do not know these variables ahead of time in this data set, we add in 50 additional "junk" predictors in the data that consist of randomly permuted predictors. By permuting the predictors, we break any association with our newly constructed predictor and the response.

We fit a classification random forest for different values of *mtry*, and we compute directional derivatives of the kernel as in Section 4.5.2. More precisely, we estimate directional derivatives at the point x_0, where the $p = 107$ components of x_0 are the sample means for each predictor (including the "junk" predictors). The directional

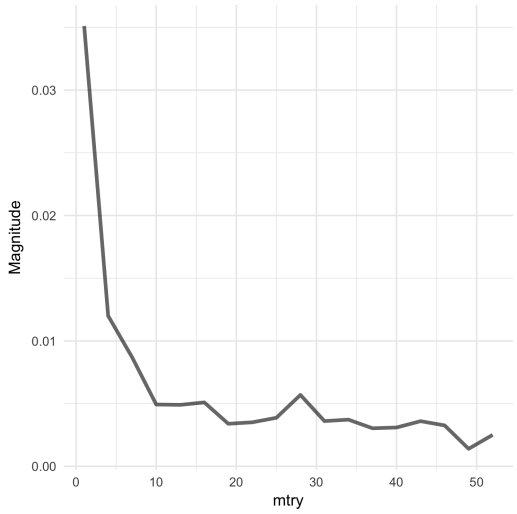


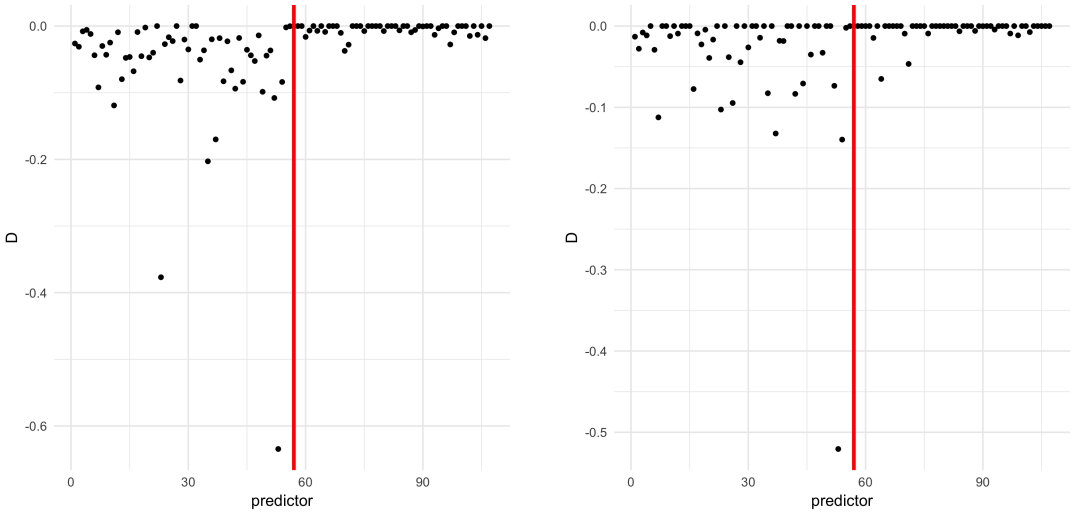
Figure 29: The average absolute value of the magnitude of directional derivative in the artificial noise variables in the *spam* data set.

derivative estimates are given by

$$D_{j+}K(x_0, x_0) \approx \frac{K(x_0 + h_j e_j, x_0) - K(x_0, x_0)}{h_j}$$

where h_j is equal to 0.2 times the standard deviation of the j^{th} predictor. Figure 29 plot the average absolute value of the directional derivative in the 50 "junk" directions as a function of *mtry*. As predicted, the typical size of this derivative decreases as the value of *mtry* increases. This is not surprising: larger values of *mtry* make it more likely that signal variables are used in tree splits, decreasing the influence of noise variables. One can visualize this effect as a "flattening" of the kernel in noise dimensions as *mtry* increases.

We provide another illustration of the kernel shape in Figure 30. In each figure, the vertical axis relates the value of the directional derive for each coordinate $j = 1, \ldots, 107$, given by the horizontal axis. Figure 30a shows this relationship for a random forest fit with *mtry* set to the classification forest default of $\sqrt{p} = 10$, while



(a) $mtry = 10$ (b) $mtry = 34$

Figure 30: Directional derivatives for each variable in the *spam* data set. Variables with indices to the right of the red line are artificially contructed noise variables.

Figure 30b shows this relationship for the regression forest default of $p/3 = 34$. In each figure, it is immediate to notice that the last 50 predictors have very small directional derivatives, given by black dots close to zero. When *mtry* increases from 10 to 34, the values of all directional derivatives get shrunken to zero, with the exception of a few large values for predictors in the original data set.

4.6. Probability Comparisons

We will now undertake an empirical investigation to determine the extent to which random forest parameter tuning influences its probability estimates in a number of simulated and real data examples. Specifically, we will be interested in studying the affect of *mtry* on probability estimates, as well as the type of forest used, i.e. regression, classification, or proximity. In particular, we will consider regression and classification random forests with values at *mtry* set at both $p/3$ and $\sqrt{p}$, proximity random forests tuned for the best value of *mtry*, completely random forests, and

bagged CART trees.

4.6.1. Real Data Sets

We begin by considering probability estimation in six data sets taken from the UCI machine learning repository: *spam*, *splice*, *tic*, *parkinsons*, *australian credit*, and *ionosphere*. A description of the original data is contained in Appendix 4.8.3. Unlike the simulated examples presented earlier in the paper, we do not have the actual probabilities in our data sets, so we will evaluate our probability estimates according to an empirical root mean squared error:

$$\sqrt{\sum_{i=1}^{n} (\widehat{p}_i - y_i)^2}.$$

This quantity is computed for 50 random 80/20 splits of the data, and we report the

	RF^{class}	RF^{reg}	RF^{class}	RF^{reg}	Bagged	Random	RF^{prox}
mtry	$p/3$	$p/3$	$\sqrt{p}$	$\sqrt{p}$	p	One	Best
spam	0.200	0.201	0.196	0.198	0.208	0.211	0.207
splice	0.119	0.120	0.159	0.161	0.122	0.241	0.124
tic	0.163	0.179	0.183	0.200	0.135	0.287	0.184
parkinsons	0.257	0.260	0.257	0.259	0.268	0.280	0.313
australian credit	0.310	0.308	0.312	0.312	0.315	0.328	0.308
ionosphere	0.227	0.230	0.225	0.227	0.245	0.287	0.262

Table 3: Table of root mean squared errors for probability estimates for each of 6 data sets from the UCI repository. Values of RMSE were computed as an average over 50 random training and testing splits for each dataset. The probability estimation methods considered are random forests in regression and classification modes with different settings of *mtry*, bagged trees, completely random forests, and proximity random forests.

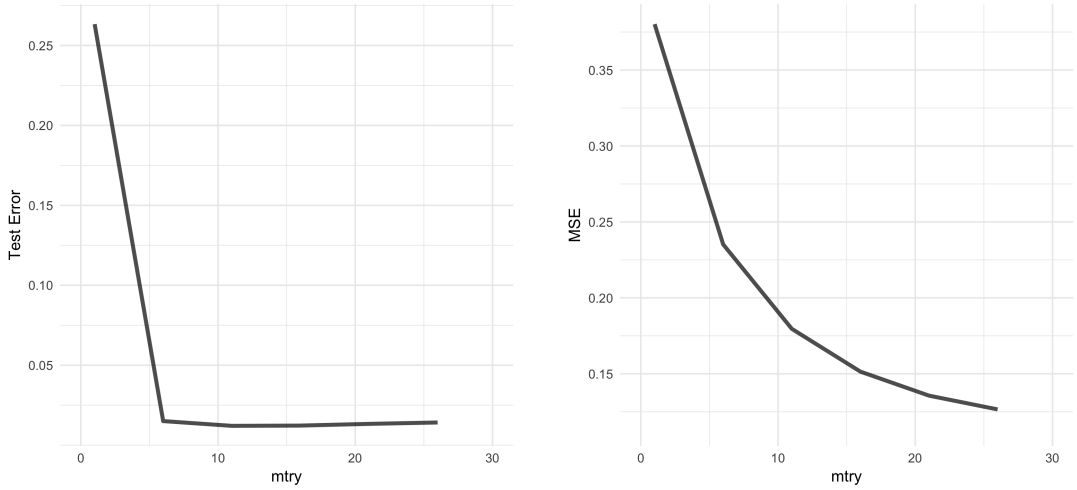
average. Furthermore, we add in 50 "junk" predictors to each data-set in the manner described in Section 4.5.3. We added in noisy predictors to investigate the efficacy of random forests in relatively sparse settings, as well as to consider a wider variety of

mtry settings in data sets with small numbers of predictors.

The first set of results are reported in Table 3. Here, we record the root mean square error for the seven methods previously described, evaluated on six data sets. At a high level, no method uniformly dominates any other. However, the completely random forest performs the worst in each experiment. This is unsurprising, especially in light of our discussion of the shape of the completely random forest kernel described earlier in the paper. Each predictor and each split point are chosen uniformly at random, so the forest does not adapt to the shape of the data. Bagged trees perform the best on one data set, and RF_{prox} performs the best one one data set. The reader may recall that our random forest model predicted that the bagged trees would have a kernel tht was the most narrow in the signal dimensions since it exhaustively searched over predictors at each split. This is not neccesarily at odds with our results: bagged trees are also much more shallow as a result of the better splits, which results in a smaller effective bandwidth.

One of the aims of this paper was to argue that the tuning parameter *mtry* matters much more for succesful probability estimation than the type of the forest. There have been claims in the literature that regression random forests are preferred for probability estimation because of their interpretation as 'conditional expectation machines' (Malley et al., 2012), (Kruppa et al., 2014). However, these experiments ignore the fact that regression and classification forests have different default settings of *mtry* in common software, and we argue this is a crucial confounding factor. For each experiment, we conducted a paired t-test comparing the root mean square error for regression and classification forests for the two different levels of *mtry* used as defaults in existing software: $p/3$ and $\sqrt{3}$. In all but the *tic* dataset, we found the root mean square errors produced by classification and regression forests to be indistinguishable.

We also considered a more in-depth analysis of the *splice* data set. Figure 31 displays the misclassification error (test error) and root mean square error as a function of *mtry*. It is clear from Figure 31a that test error is relatively immune to the value of *mtry*: for values larger than 4, test error is roughly the same. The story is quite different for RMSE, as shown in Figure 31b. Here, the fall in RMSE is much more gradual as a function of *mtry*, plateauing near a value of 30. It is of interest to note that if one were to optimize the forest for test error and used $mtry = 4$, the corresponding value of RMSE would be around 0.22, which is far away from the best value of 0.1. This example reiterates our point that tuning random forest parameters can matter substantially depending upon the quantities one wishes to estimate. In general, test error is much less sensitive to tuning parameters than probability estimation error.



(a) (b)

Figure 31: Test error and root mean square probability estimation error for the *splice* data as a function of *mtry*. Test error is generally much less sensitive to this parameter value.

4.6.2. Synthetic Data

In the previous experiments, we did not know the actual probability in each case, so we needed to use a surrogate measure for the quality of our probability estimates. Here, we will analyze six simulated data sets in order to compare the different methods of obtaining probability estimates. In each instance we will draw a training set of size $n = 500$, fit seven different models, and evaluate each model on a test set of size $n = 1000$. This process is repeated 50 times, and we consider the average root mean square error over these repetitions.

The first three models in our experiments have appeared elsewhere in this paper: the *circle model* was mentioned in Section 4.4.3 and a simple model with signal in only one dimension was used as a motivating example in Section 4.1.1. We also consider a simple logistic regression model in three dimensional space, as well as a more complicated logistic model that contains multiplicative interactions, the *10d Model*. Our last model generated data according to the *XOR* function in two dimensions. Please see Appendix 4.8.3 for mathematical details. This set of models captures a range of different statistical phenomena, ranging from simple probability density surfaces in low dimensions to more complicated nonlinear response surfaces. The results are reported in Table 4. The RF^{prox} method has much better performance on the synthetic data sets, achieving the lowest average root mean squared error on four of six examples. In some settings, such as the *XOR* or *1d Model*, it suffers only half as much RMSE as its nearest competitor. Bagged trees and the completely random forest fail to achieve the best performance on any of the six data sets considered here. The difference in error between classification and regression random forests for fixed values of *mtry* are larger than in the previous section, but one method does not dominate the other. The best type of forest and tuning parameter *mtry* is heavily problem dependent.

	RF^{class}	RF^{reg}	RF^{class}	RF^{reg}	Bagged	Random	RF^{prox}
mtry	$p/3$	$p/3$	$\sqrt{p}$	$\sqrt{p}$	p	One	Best
Circle Model	0.161	0.149	0.179	0.168	0.179	0.150	0.137
1d Model	0.207	0.189	0.223	0.208	0.223	0.181	0.097
1d Model Sparse	0.121	0.120	0.138	0.137	0.132	0.181	0.126
10d Model	0.370	0.371	0.370	0.371	0.369	0.383	0.396
Logistic Model	0.162	0.151	0.177	0.168	0.184	0.136	0.115
XOR	0.215	0.197	0.229	0.213	0.229	0.193	0.136

Table 4: Table of root mean squared errors for probability estimates for each of 6 simulated models . Values of RMSE were computed as an average over 50 random training and testing splits for each dataset. The probability estimation methods considered are random forests in regression and classification modes with different settings of *mtry*, bagged trees, completely random forests, and proximity random forests.

4.7. Conclusion

In the statistics literature, the way in which one accomplishes probability density estimation in a nonparametric setting is through the use of (Parzen) kernels. The canonical reference for this methodology is the Parzen (1962), 'On Estimation of a Probability Density Function and Mode.' Random forests have also been found to estimate probabilities well in some settings, but they do so in an ostensibly different way, by averaging the votes of trees. The connection that this paper makes is to frame random forest probability estimation in a familiar statistical setting.

As discussed in the Parzen paper, the kernel and its associated bandwidth parameter are crucial quantities in determine the quality of density estimate. In a random forest, it is not obvious what these two quantities are, or how one might tune them. Our paper extracts the kernel from the forest, and identifies the relevant tuning parameter. We establish a model of the kernel which gives the user intuition for how changing this parameter affects the kernel's shape. As with any kernel procedure in practice, the bandwidth parameter requires tuning: different sizes work better in different settings,

completely dependent upon the problem at hand.

The practical implications of this paper are that for random forest probability estimation, tuning matters. This point is easy to overlook because random forests misclassification error rate tends to be very robust to parameter settings. In practice, it is tempting to trust that a random forest that produces a low misclassification error rate will also produce reasonable probabilities, but the examples in this paper have illustrated that this need not be the case.

Finally, the 'kernel' view of random forest also suggests extensions to the way random forests are used for classification and probability estimation. As argued in Section 4.3, one can view random forests as kernel regression with the proximity function. The proximity function is just one possible measure of 'closeness' that can be extracted from the forest. As a simple example, instead of a simple binary measure of whether two training points occupy the same terminal node of a tree, one might instead take into consideration the number of tree splits separating these points.

4.8. Appendix

4.8.1. Equivalence of Splitting Criteria

It is a trivial fact that the mean squared error and Gini splitting criteria are equivalent when used with a binary outcomes $y \in \{0, 1\}$. As a consequence, random classification and regression forests fir to binary data only differ in the default parameter settings and aggregation method across trees. We will present a simple argument for this fact.

Suppose that a candidate split of some node results in left and right daughter nodes, denoted by R_L and R_R, each of which contains N_L and N_R training points, respec-

tively. The fitness criteria for such a split according to mean squared error is

$$\sum_{x \in R_L} (y_i - p_L)^2 + \sum_{x \in R_R} (y_i - p_R)^2 \tag{4.3}$$

while the fitness for the Gini criteria is

$$N_L p_L (1 - p_L) + N_R p_R (1 - p_R) \tag{4.4}$$

where $p_L = \frac{1}{N_L} \sum_{x \in R_L} y_i$ and $p_R = \frac{1}{N_R} \sum_{x \in R_R} y_i$. We can expand the squares in the mean squared error criteria in Equation 4.3 as follows:

$$\begin{aligned}
\sum_{x \in R_L} (y_i - p_L)^2 + \sum_{x \in R_R} (y_i - p_R)^2 &= N_L p_L - 2N_L p_L^2 + N_L p_L^2 + N_R p_R - 2N_R p_R^2 + N_R p_R^2 \\
&= N_L p_L - N_L p_L^2 + N_L p_L - N_L p_L^2 \\
&= N_L p_L (1 - p_L) + N_R p_R (1 - p_R) .
\end{aligned}$$

4.8.2. Naive Approximation

Suppose a random forest is grown to a size of M nodes. Under the setting of Algorithm 4, we will derive an approximation to the proximity function $K(0, z)$, where $z \in [0, 1]^p$. Our argument will borrow substantially from Breiman (2000a) and Scornet (2016b). Under completely random splitting (no notion of strong and weak variables), Breiman (2000a) found an approximation to the proximity function of the form $K(0, x) = \exp\left(-\log M / p \sum_{i=1}^{p} x_i\right)$ (although it is worth noting that the argument presented there claimed to approximate $K(x, z)$ for all $x, z \in [0, 1]$, but it contained an error).

Algorithm 4 Simplified Random Forest Tree

1. Specify the number of leafs M; initialize $\texttt{leafs} = \{\mathfrak{t}_{root}\}$.
2. For $m = 1 : M$:
   (a) Select a terminal node $\mathfrak{t} \in \texttt{leafs}$ uniformly at random.
   (b) Split $\mathfrak{t}$ into daughter nodes $\mathfrak{t}_L$, $\mathfrak{t}_R$
      (i) Choose *mtry* predictors at random $\mathcal{F} \subseteq \{1, \ldots, p\}$
      (ii) Select split variable uniformly among the S_* signal variables in $\mathcal{F}$
      (iii) Choose split point uniformly at random
   (c) Replace $\mathfrak{t}$ with $\mathfrak{t}_L$ and $\mathfrak{t}_R$ in $\texttt{leafs}$

Specifically, we will assert that

$$K(0, z) \approx \exp\left\{ -\log M \left(p_{\mathcal{S}} \sum_{s \in \mathcal{S}} z_s + p_{\mathcal{W}} \sum_{w \in \mathcal{W}} z_w \right) \right\} \tag{4.5}$$

where

$$\begin{aligned} p_{\mathcal{S}} &= \sum_{k=1}^{S \wedge mtry} \frac{\binom{S-1}{k-1}\binom{W}{mtry-k+1}}{k\binom{p}{mtry}} \\ p_{\mathcal{W}} &= \frac{1 - S p_{\mathcal{S}}}{W}. \end{aligned}$$

First, we will calculate the probability p_s that a given strong variable x_s is selected at a given node when there are S strong variables, W weak variables, and *mtry* predictors are considered at a time. Let R denote the number of strong variables selected among the *mtry*, and Q denote an indicator for whether x_s is selected . We will compute $p_{\mathcal{S}}$ by conditioning on R:

$$\begin{aligned} p_{\mathcal{S}} &= \sum_{k=1}^{S \wedge mtry} p\left(Q = 1, R = k\right) \\ &= \sum_{k=1}^{S \wedge mtry} \frac{\binom{S-1}{k-1}\binom{W}{mtry-k+1}}{k\binom{p}{mtry}}. \end{aligned}$$

It is then easy to see that the probability of selecting a given weak variable is just $\frac{1-Sp_s}{W}$.

Next, we will compute the probability that 0 and z are in the same terminal node given that there are k_j total splits on coordinate $j = 1, \ldots, p$. Let $c_1, \ldots, c_{k_j}$ denote the randomly chosen split points for coordinate j. Then the probability that these split points do not separate 0 and x_j, $p\left(c_1 \notin [0, x_j], c_2 \notin [0, x_j] \ldots, c_{k_j} \notin [0, x_j]\right)$, is

$$\int_{c_1 \notin [0,x_j]} \int_{c_2 \notin [0,x_j]} \cdots \int_{c_{k_j} \notin [0,x_j]} p(dc_{k_j}|c_{k_j-1}) \cdots p(dc_2|c_1)p(dc_1). \tag{4.6}$$

Now, given c_{k-1}, the distribution of c_k is $c_k|c_{k-1} \sim \mathcal{U}[0, c_{k-1}]$. Thus, it holds that

$$p\left(c_k \notin [0, x_j]|c_{k-1}\right) = 1 - x_j/c_{k-1}.$$

One can then prove inductively that the integral in 4.6 reduces to $1-x_j \sum_{i=0}^{k_j-1} \frac{(-\log x_j)^i}{i!}$. If we let $w_j = -\log x_j$, this is precisely the probability that a Poisson random variable Z_j with parameter w_j is greater than k_j. Next, we need to consider $p\left(k_1, \ldots, k_p\right)$, the joint probability of cutting k_j times on predictor x_j for $j = 1, \ldots, p$. If we condition on the total number of cuts K, then $p\left(k_1, \ldots, k_p|K\right)$ is multinomial with K total trials and success probabilities $(p_s, \ldots, p_s, p_w, \ldots, p_w)$. Finally, the distribution of K is the sum of $M-1$ Bernoulli random variables, the m of which has success probability $1/m$. Putting this all together,

$$K(0, z) = \sum_{k=1}^{T} p(k) \sum_{k_1 + \cdots k_p = k}^{k} p\left(k_1, \ldots, k_p|K = k\right) \prod_{j=1}^{p} p\left(Z_j \geq k_j\right). \tag{4.7}$$

We seek a more tractable approximation to the probability computed in 4.7. Following Breiman (2000a), we appeal to a Poisson approximation. Let us assume that $p > 5$,

$T \leq \exp(p/2)$ and W and S are such that p_s and w_s are both small. If there are currently K nodes, then the probability of selecting the node that 0 and z are both in is $1/K$. So, k_j is achieved by $T-1$ binomial trials, such that the probability of each is p_s is x_j is strong, and p_w if x_j is weak. Thus, we will assume that $k_1, \ldots, k_p$ are independent Poisson random variables with parameter

$$\lambda_s = p_s \sum_{k=1}^{T-1} \frac{1}{k} \approx p_s \log T$$
$$\lambda_w = p_w \sum_{k=1}^{T-1} \frac{1}{k} \approx p_w \log T$$

depending upon whether the predictor is strong or weak. We may then approximate 4.7 by

$$\prod_{s \in \mathcal{S}} U(\lambda_s, w_s) \prod_{w \in \mathcal{W}} U(\lambda_w, w_w) \tag{4.8}$$

where $U(\lambda_s, w_s) = e^{-\lambda_s - w_s} \sum_{k=1}^{\infty} \lambda_s^k p(Z_s \geq w_s)$, and analogously for $U(\lambda_w, w_w)$. Finally, through a Laplace transform argument, Breiman (2000a) argued that $U(\lambda_s, w_s) \approx e^{-\lambda_s w_s}$.

4.8.3. Data Set Descriptions

Data Set	N	Features
australian credit	690	15
ionosphere	351	34
parkinsons	195	22
spam	4601	57
splice	2422	60
tic	958	9
voting	435	16

Table 5: Descriptions of UCI Respository data sets used in Section 4.6.

- 1d Model

$$\mathbb{P}\left(y=1|x\right)=\begin{cases}0.3 & x_1x_2\geq 0\\ 0.7 & x_1x_2<0\end{cases}$$

- Circle Model

$$\mathbb{P}\left(y=1|x\right)=\begin{cases}1, & ||x||_2\leq 8\\ \frac{28-||x||_2}{20}, & 8\leq||x||_2\leq 28\\ 0, & \text{otherwise.}\end{cases}$$

- Logistic Model

$$\mathbb{P}\left(y=1|x\right)=\frac{1}{1+e^{-2(x_1+x_2+x_3)}}.$$

- 10d Model

$$\log\left(\frac{\mathbb{P}\left(y=1|x\right))}{1-\mathbb{P}\left(y=1|x\right)}\right)=0.5(1-x_1+x_2-\cdots+x_6)(x_1+\cdots+x_6)$$

- XOR Model

$$\mathbb{P}\left(y=1|x\right)=\begin{cases}0.3, & x_1x_2\geq 0\\ 0.7, & x_1x_2<0.\end{cases}$$

CHAPTER 5 : Generalizations of the Proximity Kernel

Abstract

A random forest contains an implicit measure of similarity between points in the input space known as the proximity function. The proximity function simply gives the fraction of trees in a forest for which two points fall in the same terminal node. While useful in a great number of applications, this measure is also quite crude, as it ignores all of the information contained in a tree other than its terminal nodes, such as the quality and depth of its splits. In this paper, we construct a rich class of kernels derived from a random forest which generalize the proximity function. Moreover, we demonstrate that these kernels are useful alternatives to the proximity function in probability estimation and visualization examples.

5.1. Introduction

Random forests are among the best off-the-shelf classifiers in existence. A common explanation for their excellent performance relies on ensemble principles: a random forest achieves bias reduction from its deep, un-pruned trees, and variance reduction from averaging de-correlated trees (Breiman, 2001). More recent work has painted random forests through the lens of kernel regression (Breiman, 2000a; Olson and Wyner; Scornet, 2016a). From this point of view, a random forest works well because it is able to detect training data that is *similar* to a target point when making a prediction (Lin and Jeon, 2006). The similarity measure implicit in a random forest is the *proximity function*, which measures the fraction of trees in the forest for which two points appear in the same terminal node.

The proximity function notion of similarity is intriguing since it is learned directly from the data. For example, one finds that in simulated examples with a sparse signal, the proximity function tends to be more "pointed" in the direction of signal variables and "flatter" in the direction of noise variables (Olson and Wyner). In contrast, most kernel methods in machine learning utilize a fixed, predefined kernel, such as a radial basis function. While there have been efforts in the kernel learning literature to learn ideal kernels for different generative processes (Jaakkola and Haussler, 1999), the random forest does this implicitly and automatically.

We will illustrate in this paper that the proximity is far from the only notion of similarity one can extract from a random forest. One obvious shortcoming of the proximity function is that it only utilizes part of the total knowledge encapsulated in a tree, namely, terminal node membership. In later sections we will provide examples in which terminal node membership is largely uninformative. In these examples, split quality, especially near the root of the tree, is much more important. We will propose a new similarity kernel derived from a random forest that incorporates a richer set of information from the tree. Moreover, we will illustrate the practical advantages of this kernel over the proximity function in probability estimation and visualization tasks.

5.2. Background

We will begin with a mathematical framework for a random forest. This framework will make it natural to draw a close connection between random forests and kernel functions. Specifically, we will find that the *proximity function* falls easily out of the definition of a random forest's probability estimates. Subsequent sections will make strides toward generalizing this kernel by incorporating additional information

learned from the training data during the tree growing process.

5.2.1. Random Forest Ensembles

In this section we will describe a random forest as originally defined in Leo Breiman's seminal 2001 paper Breiman (2001). A *random forest* is a collection of T unpruned CART decision trees, where each tree is grown on a bootstrap sample of the data, and the split variable at each node of the tree is chosen to be the best among a subset of F randomly chosen predictors. Software implementations often include F as a parameter named `mtry`, as in the *R* library *randomForest* (Liaw and Wiener, 2002), or `max_features`, as in the *Python* machine learning library *sklearn* (Pedregosa et al., 2011). We model the randomness injected into each tree - such as the bootstrap sample and the subset of variables considered at each node - through a random variable $\theta \in \Theta$. For our purposes, θ will be used primarily as a way to index trees in the forest. Finally, observe that each random forest tree partitions the input space $\mathcal{X}$ into a set of hyper-rectangles. For a tree generated by parameter θ, we will denote by $\mathcal{R}_\theta(x)$ the hyper-rectangle that contains a point $x \in \mathcal{X}$.

We can now leverage this notation to more formally define a random forest tree. We will make a couple of simplifying assumptions to lighten the notation, which are often considered in the literature (Breiman, 2000a). In particular, we assume that each random forest tree is grown to maximal depth so that each terminal tree node contains exactly one sample point. We will also omit the use of bootstrap sampling. After fitting a random forest to training data $(x_1, y_1), \ldots, (x_n, y_n)$, we denote the prediction made by the tree generated by the parameter θ at a point $x \in \mathcal{X}$ by

$$f(\theta, x) = \sum_{i=1}^{n} \mathbb{I}\left(x_i \in \mathcal{R}_\theta(x)\right) y_i.$$

In words, to make a prediction, simply drop x down the tree, find which cell $\mathcal{R}_\theta(x)$ it occupies, and assign it the label y_i corresponding the unique training point x_i that lies in that cell.

A random forest makes a class prediction by taking the majority vote among the trees in the forest. It is also common practice to extract probability estimates from a fitted random forest by considering the fraction of trees in the forest that vote for a certain class (Breiman, 2002; Olson and Wyner; Liaw and Wiener, 2002). In our notation, we may write the random forest probability prediction at a point x by

$$\begin{aligned}\widehat{p}_{rf}(y=1|x) &= \frac{1}{T}\sum_{t=1}^{T} f(\theta_t, x) \\ &= \frac{1}{T}\sum_{t=1}^{T}\sum_{i=1}^{n} \mathbb{I}\left(x_i \in \mathcal{R}_{\theta_t}(x)\right) y_i \\ &= \sum_{i=1}^{n}\left(\frac{1}{T}\sum_{t=1}^{T} \mathbb{I}\left(x_i \in \mathcal{R}_{\theta_t}(x)\right)\right) y_i.\end{aligned}$$

The item in parenthesis on the last line has the straightforward interpretation as the fraction of trees in the forest for which x_i and x occupy the same terminal node. This quantity is important enough to warrant a definition.

Definition 4 (The Proximity Function)**.** *The random forest* ***proximity function*** $K^{prox} : \mathcal{X} \times \mathcal{X} \to [0,1]$ *is defined by*

$$K^{prox}(x,z) = \frac{1}{T}\sum_{t=1}^{T} \mathbb{I}\left(x_i \in \mathcal{R}_{\theta_t}(x)\right).$$

With this definition, we may equivalently write the probability estimate as

$$\widehat{p}_{rf}(y = 1|x) = \sum_{i=1}^{n} K^{prox}(x, x_i) y_i. \tag{5.1}$$

Thus, random forest probability estimates have the satisfying interpretation as a weighted sum of the training data, where the weights are given by a similarity function between the target point x and each training points x_i. The larger the value of $K^{prox}(x, x_i)$, the more weight (x_i, y_i) is given when informing the predicted value.

5.2.2. The Proximity Kernel

The proximity function is an intuitive quantity that measures the similarity between points in the predictor space: the more often that two points x and z appear in the same terminal node of a randomly grown tree, the more similar we expect them to be. Our notation for the proximity function suggests that it is a positive semi-definite kernel (i.e. Mercer kernel), which turns out to be the case (Breiman, 2000a). Furthermore, since the proximity function is an average over a large number of trees, one can also argue that $K^{prox}(x, z)$ is a finite sample approximation to $\mathbb{P}_\theta\left(x \in \mathcal{R}_\theta\left(z\right)\right)$, and that the convergence to this probability happens almost surely as $T \to \infty$ (Breiman, 2000a). Expressed in slightly different terms, the proximity function can then be seen to be an approximation to the probability that a randomly chosen recursive partition of the input space contains the points x and z.

Historically, it has been much more common in the literature to motivate the proximity function through its numerous applications rather than its link to probability estimation. For instance, the proximity function is used for visualization, outlier detection, missing data imputation, archetypal analysis, and clustering Breiman (2002). One distinctive advantage of the proximity function over fixed kernels - such as radial

basis functions - is that its shape is adaptively learned from the data. This makes it especially appealing for situations in high dimensions with sparse signals. More recently, there has been a focus on tying the proximity function to prediction. Some work has evaluated the proximity function's utility as a kernel in support vector machines (Davies and Ghahramani, 2014; Englund and Verikas, 2012), while Olson and Wyner and Scornet (2016a) explore its use in kernel regression.

5.3. Random Tree Kernels

We saw that K^{prox} is one natural notion of distance implied by a random forest ensemble. However, this measure is quite crude in some sense, as the only information it uses from a particular tree in the ensemble is terminal node membership. For a given tree, closeness is binary: two points are either in the same terminal node, or they or not. In this section, we will see that there are many other types of tree-based distance measures that incorporate much richer information, including a tree's structure and split quality. We will then demonstrate that alternative kernels that account for this information can lead to measurable improvements in probability estimation and visualization. A more exhaustive comparison in these areas can be found in Section 5.4.

5.3.1. Topology-Induced Kernels

In Section 5.2, we saw that the terminal nodes of a tree produce a partition of the input space. Moreover, this partition is recursive and hierarchical: one can extract sequentially more coarse partitions from the tree by pruning terminal nodes. As a toy example, consider Figure 32. The first split in the tree separates the data in two vertical halves, while the second level of splits separates the right cell into two further horizontal halves. According to the proximity measure of similarity, the points x and

z have a similarity of 0, despite sharing the same cell of the first partition.

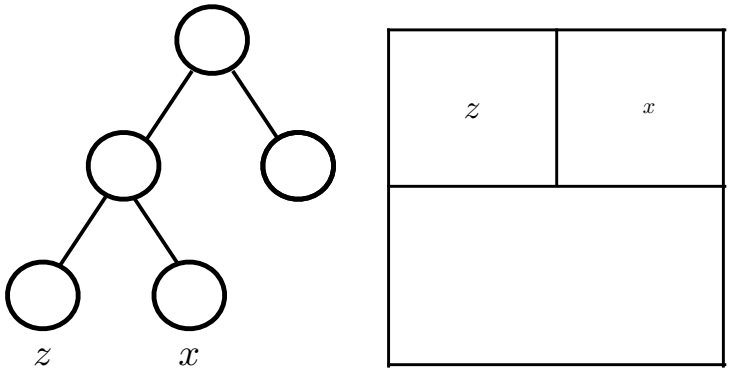


Figure 32: Equivalence between tree representation (a) and training data partition (b).

This binary notion of distance can be too crude in some circumstances. For instance, suppose we want to analyze the performance of school-age children on a standardized test. We might produce a hierarchical partition of these students according to school, grade, and class. We would expect that two students taken from the same school and grade would display some level of similarity on this test, even if they were not in the same class. This motivates the need to consider similarity between partition cells at depths other than terminal nodes.

One such notion of tree-based distance is the *path metric*, which is simply the shortest distance between two nodes in a tree where each edge has a cost of one (Semple and Mike, 2003). We can then define a measure of distance $d : \mathcal{X} \times \mathcal{X} \to \mathbb{N}$ between two points in our input space to be the path distance between the terminal nodes containing each of these points. For instance, in Figure 32, the path distance between x and z in this metric is $d(x, z) = 2$.

Previous work has constructed a tree kernel from the path metric, $K_\lambda^{path}(x, z) = e^{-\lambda d(x,z)}$ (Englund and Verikas, 2012). As the parameter λ grows to infinity, K_λ^{path} reduces to K^{prox}. As λ decreases to zero, the similarity of two points (with respect

to this metric) only depends on the fraction of the trees in the forest for which both points share splits near tree roots. In Section 5.3.2, we will prove that the K_λ^{path} is in fact a kernel, and in Section 5.4 we will analyze its performance when used in kernel regression for probability estimation.

Metrics based solely on *path distance* suffer from the obvious shortcoming that they ignore the informativeness of the splits producing the tree nodes. This problem is especially relevant in a random forest, for which the most informative splits in a given tree tend to occur near the root for large values of F (Scornet et al., 2015).

In order to make this more clear, let us consider a simple probability model in which x is drawn uniformly on $[0,1]^2$ and conditionally on x, $p(y = 1|x_1 > 0) = 0.3$ and $p(y = 1|x_1 \leq 0) = 0.7$. Only the first coordinate of x informs the conditional probability of the response. Suppose we fit a random forest tree to a data set generated by *i.i.d.* (x, y) pairs generated from this probability model. If a node in the tree splits on the first coordinate (and assuming this happens at the optimal population split $x_1 = 0$), all subsequent splits by its descendants will be due to noise alone. Furthermore, note that with high probability, any terminal node will contain an ancestor node that splits on the first coordinate. Thus, the path distance between two nodes is irrelevant, especially near the bottom of the tree. What does matter is the whether two nodes share a certain *informative* split.

One may wonder if there are other types of kernels that can be built from tree topology alone. A natural way to approach this problem is to form the graph adjacency matrix associated with the tree, and to consider various metrics on the rows of this matrix. However, it was shown that practically all metrics one might consider, including Cosine similarity, Tanimoto similarity, Pearson correlation, or resistance measures, all reduce to degenerate measures similar to the proximity matrix (Sologub, 2011).

This occurs since the degree of a tree leaf is one. In other words, one must look elsewhere to find other tree based metrics.

5.3.2. Incorporating Split Information into Similarity Measures

The previous section motivates our search for tree-based similarity measures that incorporate split information at each node. The most natural measure of split quality is the magnitude of the resulting impurity decrease (Breiman et al., 1984). This quantity also has an important role in tree-based variable importance (Breiman et al., 1984; Louppe et al., 2013). Let us recall the node splitting process that occurs when building a CART tree rooted at a node $\mathfrak{t}$. Among all possible candidate splits that produce two daughter nodes $\mathfrak{t}_L$ and $\mathfrak{t}_R$, the CART procedure chooses the split for which the impurity decrease

$$\Delta i(\mathfrak{t}) \equiv \phi(\mathfrak{t}) - (p_L \phi(\mathfrak{t}_L) + p_R \phi(\mathfrak{t}_R)) \tag{5.2}$$

is maximized, where p_L and p_R are the fraction of training points from $\mathfrak{t}$ that are located in nodes $\mathfrak{t}_L$ and $\mathfrak{t}_R$, respectively, and ϕ is a measure of impurity, such as the Gini index.

We propose a new kernel K^{Δ} that measures the similarity between two points x and z in a way can be thought of as a weighted graph distance. For each of these points, we can keep track of the values of the impurity decreases Δi as we traverse from the root node to each of the terminal nodes containing x and z, respectively. Intuitively, we assign a similarity to x and z depending upon what fraction of the total decrease in node purity from root to leaf is shared for both points.

In order to make our discussion concrete, we will develop our definition in the context of the tree in Figure 33. The path leading to the terminal node containing x results

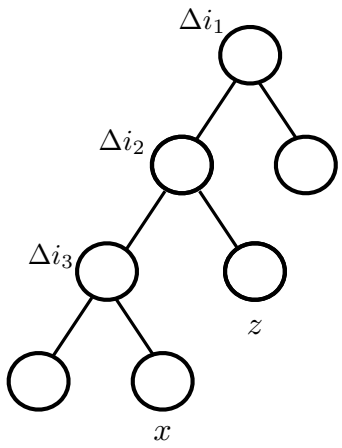


Figure 33: Tree along with node impurity decreases to illustrate the calculations of $K^{\Delta}(x, z)$.

in a total decrease in impurity of $\Delta i_1 + \Delta i_2 + \Delta i_3$, while the path leading to z results in a total decrease of $\Delta i_1 + \Delta i_2$. The terminal cells that contain x and z share only one common split that leads to an impurity decrease of Δi_1. We do not count the decrease Δi_2 which occurs at the least common ancestor node as common for the obvious reason that x and z are assigned different "directions" after passing through this split. Given these quantities, we define, the Δ-similarity between x and z as

$$K^{\Delta}(x, z) \equiv \sqrt{\left(\frac{\Delta i_1}{\Delta i_1 + \Delta i_2 + \Delta i_3}\right)\left(\frac{\Delta i_1}{\Delta i_1 + \Delta i_2}\right)}.$$

An alternative way of viewing this quantity is as follows. The total amount of decrease in node impurity leading to x is $\Delta i_1 + \Delta i_2 + \Delta i_3$, and a fraction $\frac{\Delta i_i}{\Delta i_1 + \Delta i_2 + \Delta i_3}$ of this decrease is shared with z. Conversely, a fraction of $\frac{\Delta i_i}{\Delta i_1 + \Delta i_2}$ of the total path sum of impurity decrease leading to z is shared with x. In order to aggregate these two fractions into one measure, we simply take their geometric mean. We chose the geometric mean as an aggregator because of its connection to the cosine kernel, which appears in the proof of Proposition 6. Later discussion will mention some alternatives.

More formally, fix a tree with M total nodes, and index the nodes according to $m = 1, \ldots, M$. Suppose that $A = \{i_1, \ldots, i_a\}$ indexes the nodes leading from the root to the terminal node containing x, and $B = \{j_1, \ldots, j_b\}$ indexes the nodes leading from the root to the terminal node containing z, and ℓ is the index of the least common ancestor between these terminal nodes. This notation allows us to define the kernel K^{Δ}.

Definition 5 (The Δ Kernel)**.** *The kernel* $K^{\Delta} : \mathcal{X} \times \mathcal{X} \to [0, 1]$ *is defined by*

$$K^{\Delta}(x, z) = \sqrt{\left(\frac{\sum_{k \in C} \Delta i(\mathfrak{t}_k)}{\sum_{a \in A} \Delta i(\mathfrak{t}_a)}\right)\left(\frac{\sum_{k \in C} \Delta i(\mathfrak{t}_k)}{\sum_{b \in B} \Delta i(\mathfrak{t}_b)}\right)}$$

where $C = (A - \{\ell\}) \cap (B - \{\ell\})$.

This new measure of similarity satisfies some comforting properties. First, if the terminal nodes containing x and z contain no common splits, then $K^{\Delta}(x, z) = 0$. If x and z share the same terminal node, then clearly $K^{\Delta}(x, z) = 1$. Furthermore, recall the one-dimensional probability model given in Section 5.3.1 for which only the first coordinate contained information about the response. After splitting on the first coordinate, any further splits will lead to small decreases in impurity since the residual values in nodes are equally likely to have the value $y = 1$ or $y = 0$. Thus, the distance between any points located in a subtree rooted at a node cause by a split on the first dimension should be small. Despite being far apart in "node-space," these points are close with respect to K^{Δ}. Finally, note that we defined K_{λ}^{path} and K^{Δ} for individual trees. Their extensions to an ensemble of trees is obvious: compute these kernels for each tree in the ensemble and average.

We will now argue that the three "kernels" K^{prox}, K_{λ}^{path}, and K^{Δ} are all positive semi-definite Mercer kernels. There are two important reasons for doing this. The

first is that we want to ensure that these functions have the basic properties we would expect from a well-defined similarity measure, such as symmetry. The second reason is numerical: the routines used in popular kernel methods all require positive semi-definite input matrices (such as in the multidimensional scaling applications we consider later in the paper). Note that it has already been proved in the literature that K^{prox} is a Mercer kernel, but from a much different approach than Proposition 6.

Proposition 6. *The functions K^{prox}, K^{path}_{λ}, and $K^{\Delta} : \mathcal{X} \times \mathcal{X} \to [0,1]$ are all positive semi-definite Mercer kernels.*

Proof. The proof will rely on basic facts about Mercer kernels that the reader can find in Smola and Schölkopf (1998) or Shawe-Taylor and Cristianini (2004). In each case, the target kernel is an average of kernels produced by each tree. Since it is the case that if $K_1, \ldots, K_T$ are all kernels so is $K \equiv 1/T(K_1 + \cdots + K_T)$, it remains only to show that the kernel induced by each tree is a Mercer kernel. We will also rely heavily on the fact that if $K : \mathcal{W} \times \mathcal{W} \to \mathbb{R}^+$ is a kernel and $g : \mathcal{X} \to \mathcal{W}$ is a function, then $K_g(x_1, x_2) \equiv K(g(x_1), g(x_2))$ is a kernel. In each case below, we produce a mapping g and a base kernel K.

(a) K^{prox}

Suppose that the tree has M *terminal nodes*, and index each of these nodes by $m = 1, \ldots, M$. Let $g : \mathcal{X} \to [0,1]^M$, where $(g(x))_m$ is the indicator function of whether x lies in the terminal node with index m. Then $K^{prox}(x, z) = \langle g(x), g(z) \rangle$.

(b) K^{path}_{λ}

Suppose that the tree has M *total nodes*, and index these nodes by $m = 1, \ldots, M$. Let $g : \mathcal{X} \to [0,1]^M$, where $(g(x))_m$ is the indicator function of

whether x lies in a terminal node that is a descendant of node m. We will show that $K_\lambda^{path}(x,z) = \exp(-\lambda||g(x) - g(z)||^2)$. In order to see this, notice that $||g(x) - g(z)||^2$ is the path distance $d(x,z)$ between the terminal nodes containing x and z: $||g(x)||^2$ gives the depth of the path leading to the terminal node containing x, $||g(z)||^2$ in the case of z, and $\langle g(x), g(z)\rangle$ is the depth of the least common ancestor of the terminal nodes containing x and z. Finally, $||g(x) - g(x)||^2 = ||g(x)||^2 + ||g(z)||^2 - 2\langle g(x), g(z)\rangle$, which is equivalent to the node distance between x and z in light of the preceding interpretations. The kernel K is taken to be the *Gaussian* kernel $\exp(-\lambda||x - z||^2)$.

(c) K^Δ

Suppose that the tree has M *total nodes*; we will index all these nodes, excluding the root node, by $m = 1, \ldots, M-1$. Suppose $\phi : \{1, \ldots, M-1\} \to \mathbb{R}^+$ is any function mapping tree nodes to positive scalars. In the case of K^Δ, this function can be taken to be $\phi(m) = \sqrt{\Delta i_m}$. Now let $g : \mathcal{X} \to [0,1]^{M-1}$ where $(g(x))_m$ is $\phi(pa(m))$ if the terminal node containing x is a descendant of node m, and 0 otherwise. The kernel K is taken to be the cosine kernel $\frac{\langle x,z\rangle}{||x||||z||}$.

□

We conclude this section by mentioning that the types of kernels described here are far from exhaustive, and there are numerous avenues for refinement. Focusing on the case of K^Δ, the use of the geometric mean to aggregate the fraction of shared decrease in impurity was chosen since it could be easily mapped to the *cosine* kernel mentioned in the proof of Proposition 6. One can also show that the harmonic mean of these two quantities gives rise to a kernel (Belanche Muñoz and Tosi, 2012) (or any of the dozens of kernels mentioned in Shawe-Taylor and Cristianini (2004)). As another

variation, one might consider weighting the decreases in impurity by the depth that these splits occur in the tree. The main point is that the typical way in which one uses trees in a random forest to measure similarity is far from the only one. In the next two sections, we will give simple use-case examples to illustrate the power of this generality.

5.3.3. Kernel Comparison I: Probability Estimation

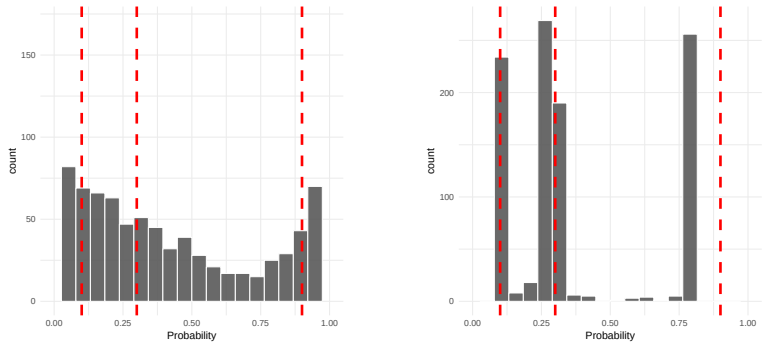
We saw in Section 5.2.1 that the probability estimates produced by a random forest are similar to those produced by kernel regression using the proximity kernel. We can use the class of kernels discussed in the previous section to produce competing kernel regression probability estimates. In particular, in this section we will compare the probability estimates resulting from a random forest to those obtained with kernel regression using K^{Δ} and K_{λ}^{path} for $\lambda = 0.4$, and kernel regression with a radial basis function with variance parameter 0.1. The value of $\lambda = 0.4$ was tuned with out-of-bag (OOB) data. This procedure is described further in Section 5.4.

The probability model we consider generates observations in the following manner:

$$\begin{aligned} z &\sim \text{Unif}(1, 2, 3, 4) \\ x|z &\sim \mathcal{N}(\mu_z,\ I_{2\times 2}) \\ y|z &\sim \text{Ber}(p_z) \end{aligned}$$

where $\mu_1 = (-2, -2), \mu_2 = (-2, 2), \mu_3 = (2, -2), \mu_4 = (2, 2)$ and $p_1 = 0.1, p_2 = 0.3, p_3 = 0.3, p_4 = 0.9$. In words, we draw observations uniformly at random from four clusters, where x is drawn from a cluster dependent normal distribution, and y is a Bernoulli random variable with a cluster dependent success probability. It is worth noting at this point that a simple depth-2 decision tree with four total terminal nodes

is optimal for probability estimation in this problem.



(a) Random Forest (b) K^{Δ}

(c) $K_{0.4}^{path}$ (d) RBF

Figure 34: Histogram of probability estimates using a random forest, and kernel regression with kernels K^{Δ}, $K_{0.4}^{path}$, and a radial basis function with variance parameter 0.1.

We generate $n = 1,000$ observations from this model and compare the resulting probability estimates from each of the four methods on a hold-out set of size $n = 1,000$ ($F = 1$ and 250 trees). Figure 34 plots histograms of the estimated probabilities from each method. Observe first that our underlying probability model only admits three possible probabilities - 0.1, 0.3, and 0.9 - so an ideal estimator would produce a histogram where the mass concentrates at these values. These values are marked in each plot by red vertical lines.

The random forest estimates and the radial basis function probability estimates are

both very similar, and quite disperse, with no clear modes. The remaining two sets of estimates all have three sharp modes, although some bias occurs in each. Note that the quality of probability estimate produced by K_λ^{path} will depend critically on tuning λ correctly. As $\lambda \to \infty$, the kernel will collapse to K^{prox}, while as $\lambda \to 0$, the kernel effectively causes the tree to collapse to its root. The variance parameter for the RBF was chosen to be very small in order to make the kernel overly "sharp," so that the final estimator resembled something like one-nearest neighbors. It is clear that if the kernel is too sharp, the probability estimates will suffer from high variance.

In this light, we can analyze the kernel weights to see why K^Δ and $K_{0.4}^{path}$ perform better than the random forest in this example. The random forest proximity matrix tends to be very sparse, since the training points that impact each prediction - the "voting points" - are concentrated in small neighborhoods (Lin and Jeon, 2006). This is not surprising. The proximity measure of similarity is zero or one depending upon whether two points are in the same terminal node of a tree, and terminal nodes map to rectangles with small volume since each random forest tree is grown deep. The kernel matrix for the RBF is also very nearly sparse, since we chose a very small variance parameter. On the contrary, the weights in the other two kernels are much less sparse, meaning that more training points inform the probability estimates. In this particular example, the neighborhoods in which the probability model takes constant values is relatively large, so one can leverage more training data when producing probabilities.

5.3.4. Kernel Comparison II: Visualization

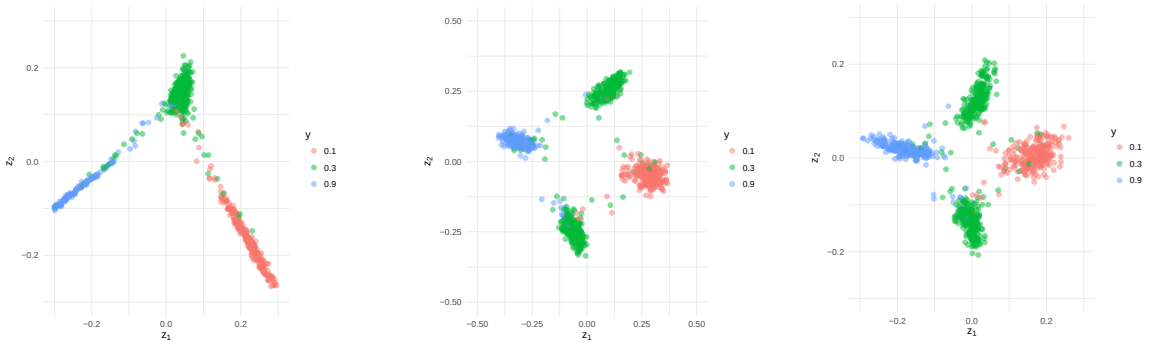
We will also consider a second application for random forests: dimensionality reduction and data visualization (Breiman, 2002). Given the a set of training points $x_1, \ldots, x_n$ and a kernel K, one can form a "distance" matrix $D \in \mathbb{R}^{n \times n}$ where $(D)_{i,j} = 1 - K(x_i, x_j)$. If K is positive semi-definite, bounded above by 1, and

satisfies $K(x_i, x_i) = 1$ for $i = 1, \ldots, n$, it can be shown that there exists an embedding $\psi : \mathcal{X} \to \mathbb{R}^m$ for $m \leq n$ such that $||\psi(x_i) - \psi(x_j)||^2 = D_{i,j}$. One can find such an embedding through multidimensional scaling (MDS). After fitting a random forest, it is common in practice to use the proximity kernel K^{prox} to produce an embedding of the data in $\mathbb{R}^2$ using the first few coordinates from MDS. The resulting plots are known as *proximity plots*.

Proximity plots tend to have a characteristic star-shape, which has caused some to question how well they represent the data. The authors in Hastie et al. (2009) claim that

> "Proximity plots for random forests often look very similar, irrespective of the data, which casts doubt on their utility. They tend to have a star shape, one arm per class, which is more pronounced the better the classification performance."

Returning to discussion in the previous section, since the proximity function has a binary similarity criteria based on node membership, points x and z in the input space can only be considered to be similar if they are also close in a Euclidean sense. In other words, if x and z are far apart in the input space $\mathcal{X}$, the value of $K^{prox}(x, z)$ will also likely be very small, even if $p(y = 1|x)$ is close to $p(y = 1|z)$. The star shape quality of proximity plots is likely to be due to this phenomenon (Hastie et al., 2009). We will illustrate the proximity plots produced by applying MDS to K^{Δ} and K_{λ}^{path} can be qualitatively different that traditional proximity plots. We revisit the probability model from Section 5.3.3 with a slight modification. We keep the set-up exactly the same, except we generate x from a normal distribution in 12 dimensional space instead of 2. The first two coordinates of each mean μ_i are the same as before, but now we add on an additional 10 identical coordinates to each mean. Since our data



(a) Proximity (b) K^{Δ} (c) $K^{path}_{0.4}$

Figure 35: Plots of the first two scaling coordiantes using the proximity kernel, K^{Δ}, and $K^{path}_{0.4}$

now lives in a higher dimensional space, visualization is a more interesting problem.

In Figure 35 we plot the first two scaling coordinates using each of the three kernels, where the plot points are colored according to their true conditional class probability values. As reflected in our discussion, the proximity plot produced by the random forest has its characteristic start shape with three branches. The proximity plots produced by K^{Δ} and $K^{path}_{0.4}$ look very similar and break the data up into four distinct clusters. This representation is appealing in the sense that the data can be perfectly separated by a decision tree of depth two which splits on the first two coordinates. In all cases, points with similar conditional class probability are grouped together.

5.4. Experiments

In this section we will conduct a more extensive comparison between the random forest and the derived kernel methods. Specifically, we will augment the probability estimation study in Section 5.3.3 with eight synthetic data sets, and the visualization study in Section 5.3.4 with five real world data sets from the UCI repository. Our goal here is not to prove that one method uniformly dominates another, but rather to provide evidence that K^{path}_{λ} and K^{Δ} are useful kernels that produce qualitatively

different results from the usual random forest machinery.

	Mease	*One-d*	*One-d (sparse)*	*Friedman*	*Friedman (sparse)*	*Logistic*	*Logistic (sparse)*	*XOR*
K^{Δ}	0.190	0.293	0.302	0.345	0.388	0.337	0.349	0.325
K^{path}	0.189	0.300	0.303	0.321	0.367	0.339	0.344	0.338
random forest	0.206	0.365	0.328	0.286	0.346	0.363	0.350	0.386

(a) Misclassification Error

	Mease	*One-d*	*One-d (sparse)*	*Friedman*	*Friedman (sparse)*	*Logistic*	*Logistic (sparse)*	*XOR*
K^{Δ}	0.206	0.043	0.098	0.398	0.410	0.126	0.155	0.121
K^{path}	0.108	0.079	0.102	0.378	0.401	0.094	0.137	0.123
random forest	0.156	0.196	0.121	0.349	0.385	0.154	0.127	0.207

(b) Brier Score

	Mease	*One-d*	*One-d (sparse)*	*Friedman*	*Friedman (sparse)*	*Logistic*	*Logistic (sparse)*	*XOR*
K^{Δ}	0.875	0.712	0.700	0.736	0.695	0.724	0.723	0.697
K^{path}	0.882	0.710	0.700	0.763	0.709	0.722	0.723	0.688
random forest	0.864	0.670	0.694	0.798	0.723	0.689	0.703	0.650

(c) AUC

Table 6: Table of probability estimation results.

5.4.1. Probability Estimation

We will compare the probability estimates produced by the following three estimators:

(1) $\widehat{p}_{rf}(x) = \frac{1}{T}\sum_{t=1}^{T} f(\theta_t, x)$

(2) $\widehat{p}_{\Delta}(x) = \sum_{i=1}^{n} \frac{K^{\Delta}(x_i,x)y_i}{\sum_{i=1}^{n} K^{\Delta}(x_i,x)}$

(3) $\widehat{p}_{path}(x) = \sum_{i=1}^{n} \frac{K^{path}(x_i,x)y_i}{\sum_{i=1}^{n} K^{path}(x_i,x)}$.

Each estimator is constructed from the same random forest in each simulation, which has $T = 250$ trees and the suggested default value of $F = \sqrt{p}$ for classification, where p is the number of input variables. All models are trained on a data set of size

$n = 500$, and are evaluated in terms of misclassification error rate, Brier score, and AUC over a test set of size $n = 1,000$. The results we report in this section were constructed as averages over 25 repetitions of each experiment. Finally, we note that when constructing K^{path}, we choose λ by tuning over the out-of-bag (OOB) training data to optimize for the Brier score. We consider five model classes, three of which we compare in low and high dimensional settings, for a total of eight simulation settings. The probability models were chosen to span a wide range of phenomena, including additive and non-additive response surfaces, sparse and dense features, as well as more traditional models such as logistic regression.

Mease Model

The first model is the *Mease model* from Mease et al. (2007). Each predictor x is drawn uniformly at random on $[-28, 28]^2$, and the conditional probability of the response y is given by

$$p(y = 1|x) = \begin{cases} 1 & \text{if } ||x||_2 \leq 8 \\ \frac{28 - ||x||_2}{20} & \text{if } x < 0 \\ 0 & \text{otherwise .} \end{cases}$$

One Dimensional Model

We consider two versions of the *one dimensional model* which was used in Olson and Wyner and mentioned in Section 5.3.1. In this model, x is drawn uniformly on $[0, 1]^p$,

and y is drawn according to

$$p(y=1|x) = \begin{cases} 0.3 & \text{if } x_1 \geq 0 \\ 0.7 & \text{if } x_1 < 0. \end{cases}$$

In a low dimensional setting, we take $p = 2$, and in a higher dimensional setting we take $p = 50$. Since all of the signal is contained in the first coordinate, the latter model is very sparse.

Friedman Model

The *Friedman model* is a highly nonlinear model taken from Friedman et al. (2000) which unlike the previous examples, does not have an additive structure. The predictor x is drawn from a p dimensional spherical normal distribution centered at the origin. Condition on x, the response is drawn as

$$\log\left(\frac{p(y=1|x)}{p(y=0|x)}\right) = 2(1 - x_1 + x_2 - \cdots + x_6)(x_1 + \cdots + x_6).$$

We also consider dense and sparse version of this model, taking $p = 10$ and $p = 26$.

Logistic Model

The *logistic model* generates data according to the probability model implied by a logistic regression. We draw x uniformly on $[-1, 1]^p$, where we take $p = 3$ and $p = 23$. In the former case, every variable in the input space is related to the response, where in the latter case the last 20 variables are noise variables.

$$\log\left(\frac{p(y=1|x)}{p(y=0|x)}\right) = x_1 + x_2 + x_3$$

XOR Model

Finally, the *xor model* is a simple data generating process that takes x uniformly on $[-1, 1]$, and the probability of y varies according to the parity of the signs of the coordinates of x:

$$p(y = 1|x) = \begin{cases} 0.3 & \text{if } x_1 x_2 \geq 0 \\ 0.7 & \text{if } x_1 x_2 < 0. \end{cases}$$

Results

The results from these experiments are summarized in Table 6. Aside from the *Friedman model*, the random forest tended to perform slightly worse than the kernel regression with K^{path} and K^{Δ} across all metrics. The largest performance gaps occurred in the dense *one dimensional* and *xor model*. These are two models in particular in which we would expect these performance gaps: the best predictor is given by a depth one tree in the first case and a depth 2 tree in the second case. The action of K^{Δ} is to emphasize informative splits, which would occur near the root of each random tree, and the tuning implicit in K^{path} would encourage effectively smaller trees.

We also notice that K^{path} and K^{Δ} have similar performance, but K^{path} does slightly better across all metrics in a few models. Again, this might not be surprising since K^{path} incorporates explicitly OOB tuning, while K^{Δ} is entirely determined by the training data used in each tree. While no method uniformly dominates the others, we do see noticeable advantages of our alternative kernels, especially in cases where we expect the underlying data to be generated from a simple model.

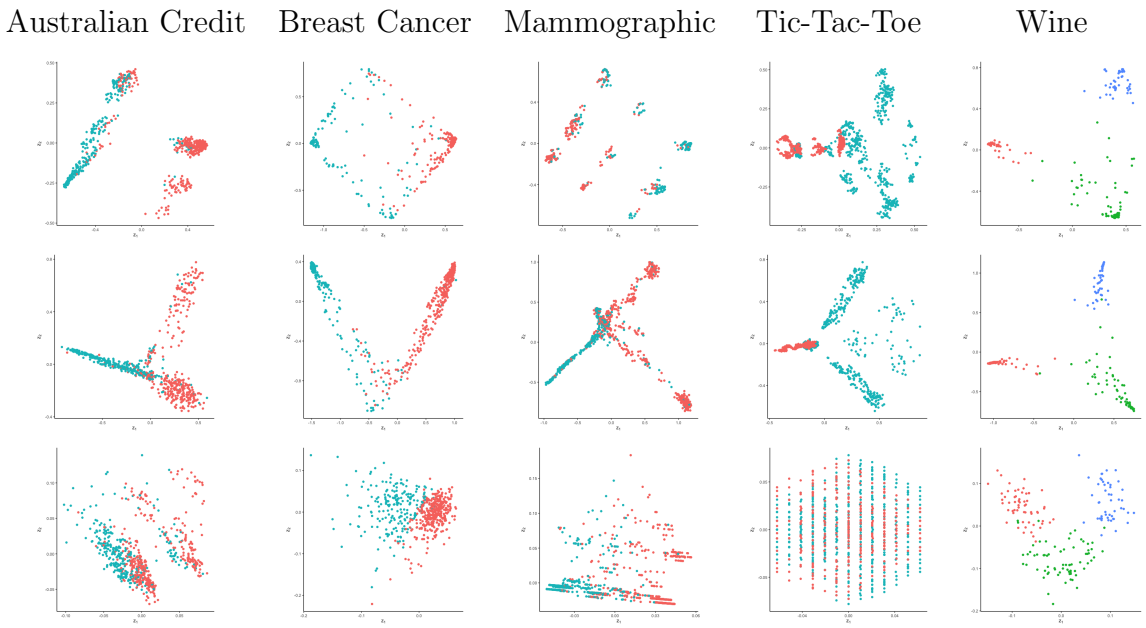


Figure 36: Data set visualizations. The first two rows show proximity plots from a random forest and K^{Δ}, while the last row shows principal components. Points are colored according to class response.

5.4.2. Visualization

Our preliminary analysis with synthetic data in Section 5.3.4 indicated that K^{Δ} (and K^{path}) was able to produce qualitatively different proximity plots than a random forest. Here, we construct proximity plots using this kernel and a random forest on five real data sets from the UCI Machine Learning Repository: *Australian credit*, *Wisconsin breast cancer*, *mammographic*, *tic-tac-toe*, and *wine*. We also compare these visualizations with the first two principle components for each data set. We chose to exclude K^{path}_{λ} from the data since it required an explicit tuning parameter, and for the purposes of visualization it is not clear what an objective criteria for this choice is.

Figure 36 shows the proximity plots produced for each data set. The first row consists of traditional proximity plots produced by a random forest, the second row shows

Name	n	p	Classes
Australian Credit	690	14	2
Breast Cancer	569	30	2
Mammographic	961	5	2
Tic-Tac-Toe	958	9	2
Wine	179	13	3

Table 7: Data set Summary

those from K^{Δ}, and the third row plots the first two principle components of the data. Each plot shows the class label plotted in a different color: all data sets have two classes, with the exception of *wine*, which has three.

Again, we observe that in each case the random forest proximity plots all have the characteristic star-shape, where each branch contains training data with similar labels. The number of arms in each is either one or two. The proximity plots produced by K^{Δ} are qualitatively much different, with a range of shapes and clusters. Interestingly, both the random forest and K^{Δ} both produce visualizations that tend to separate the data according to class label better than PCA. This is to be expected since these kernel methods see class labels at training time, where PCA does not. The difference in quality of visualization is especially apparent in the *tic-tac-toe* data set. One reason for this might be the presence of categorical variables in the data, for which all 9 predictors in this data set consist of categorical variables. Trees are a much more natural structure for dealing with discrete data compared to PCA, which expects its inputs to be real valued.

5.5. Discussion

The goal of this paper was to emphasize the close relationship between random forests and kernel methods. This relationship is both implicit in the way in which a random forest creates probability estimates, and explicit in a number of applications - such as

visualization - that deal directly with the proximity function. Our main contribution was to generalize the proximity kernel to other notions of tree-based similarity that take topology and split information into consideration. We then demonstrate that these generalizations are useful alternatives to traditional random forests in probability estimation, and produce qualitatively different visualizations.

There are a number of avenues for future work. First, one might consider using these kernels in other kernelized tasks, such as support vector machine classification, KernelPCA, or priors for Gaussian processes (Davies and Ghahramani, 2014). It is also natural to consider using these kernels in transfer learning applications. Recall from our previous discussion that one advantage of random forest kernels over fixed kernels (such as radial basis functions) is that they adaptively learn their shape from the training data. Thus, it is plausible that these kernels would perform well on related tasks.

We will also note here that kernel methods based on random forests tend to be computationally expensive, with memory and computational requirements of $\mathcal{O}(n^2T)$. This limited us to data sets and simulation settings with $n = 1,000$ or fewer observations. However, all of the operations in kernel construction can be trivially parallelized over trees which reduces this burden.

Finally, we would like to emphasize that the K^{Δ} kernel is only one possible type of similarity metric that incorporates split information. Our proof of Proposition 6 illustrates that through a simple feature embedding and appropriate choice of base kernel, one can construct many additional variations.

BIBLIOGRAPHY

Y. Amit, G. Blanchard, and K. Wilder. Multiple randomized classifiers: Mrcl. Technical report, 2000.

P. Bartlett, D. J. Foster, and M. Telgarsky. Spectrally-normalized margin bounds for neural networks. *arXiv preprint arXiv:1706.08498*, 2017.

L. A. Belanche Muñoz and A. Tosi. Averaging of kernel functions. In *ESANN 2012 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 25-27 April 2012,.,,* pages 363–368, 2012.

J. Belanich and L. E. Ortiz. On the convergence properties of optimal adaboost. *arXiv preprint arXiv:1212.1108*, 2012.

Y. Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, pages 437–478. Springer, 2012.

Y. Bengio et al. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

R. A. Berk. *Statistical Learning from a Regression Perspective*. Springer Science & Business Media, 2008.

G. Biau. Analysis of a random forests model. *Journal of Machine Learning Research*, 13(Apr):1063–1095, 2012.

G. Biau and E. Scornet. A random forest guided tour. *Test*, 25(2):197–227, 2016.

P. J. Bickel, Y. Ritov, and A. Zakai. Some theory for generalized boosting algorithms. *The Journal of Machine Learning Research*, 7:705–732, 2006.

H. Bostrom. Estimating class probabilities in random forests. In *Machine Learning and Applications, 2007. ICMLA 2007. Sixth International Conference on*, pages 211–216. IEEE, 2007.

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

L. Breiman. Stacked regressions. *Machine learning*, 24(1):49–64, 1996b.

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

L. Breiman. Prediction games and arcing algorithms. *Neural computation*, 11(7): 1493–1517, 1999.

L. Breiman. Some infinity theory for predictor ensembles. Technical report, Technical Report 579, Statistics Dept. UCB, 2000a.

L. Breiman. Special invited paper. additive logistic regression: A statistical view of boosting: Discussion. *The annals of statistics*, 28(2):374–377, 2000b.

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

L. Breiman. Manual on setting up, using, and understanding random forests v3. 1. *Statistics Department University of California Berkeley, CA, USA*, 1, 2002.

L. Breiman. Consistency for a simple model of random forests. 2004.

L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen. *Classification and regression trees*. CRC press, 1984.

L. Breiman et al. Statistical modeling: The two cultures (with comments and a rejoinder by the author). *Statistical science*, 16(3):199–231, 2001.

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

A. Buja. Special invited paper. additive logistic regression: A statistical view of boosting: Discussion. *The Annals of Statistics*, 28(2):387–391, 2000.

D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

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

T. M. Cover and P. E. Hart. Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, 13(1):21–27, 1967.

A. Davies and Z. Ghahramani. The random forest kernel and other kernels for big data from random partitions. *arXiv preprint arXiv:1402.4293*, 2014.

R. Díaz-Uriarte and S. A. De Andres. Gene selection and classification of microarray data using random forest. *BMC bioinformatics*, 7(1):1, 2006.

L. Dinh, R. Pascanu, S. Bengio, and Y. Bengio. Sharp minima can generalize for deep nets. *arXiv preprint arXiv:1703.04933*, 2017.

C. Englund and A. Verikas. A novel approach to estimate proximity in a random forest: An exploratory study. *Expert systems with applications*, 39(17):13046–13050, 2012.

G. J. Escobar, A. Ragins, P. Scheirer, V. Liu, J. Robles, and P. Kipnis. Nonelective rehospitalizations and postdischarge mortality: predictive models suitable for use in real time. *Medical care*, 53(11):916–923, 2015.

J. S. Evans, M. A. Murphy, Z. A. Holden, and S. A. Cushman. Modeling species distribution and change using random forest. In *Predictive species and habitat modeling in landscape ecology*, pages 139–159. Springer, 2011.

M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim. Do we need hundreds of classifiers to solve real world classification problems. *J. Mach. Learn. Res*, 15 (1):3133–3181, 2014.

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

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

Y. Freund and R. Shapire. Discussion of additive logistic regression: A statistical view of boosting. *Annals of Statistics*, 28:337–374, 2000.

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

J. Friedman, T. Hastie, R. 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.

W. Gao and Z.-H. Zhou. On the doubt about margin explanation of boosting. *Artificial Intelligence*, 203:1–18, 2013.

P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006.

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

H. S. Gurm, J. Kooiman, T. LaLonde, C. Grines, D. Share, and M. Seth. A random forest based risk model for reliable and accurate prediction of receipt of transfusion

in patients undergoing percutaneous coronary intervention. *PloS one*, 9(5):e96385, 2014.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*, volume 2. Springer, 2009.

K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

T. Jaakkola and D. Haussler. Exploiting generative models in discriminative classifiers. In *Advances in neural information processing systems*, pages 487–493, 1999.

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

K. Kawaguchi, L. P. Kaelbling, and Y. Bengio. Generalization in deep learning. *arXiv preprint arXiv:1710.05468*, 2017.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

J. Kruppa, Y. Liu, G. Biau, M. Kohler, I. R. König, J. D. Malley, and A. Ziegler. Probability estimation with machine learning methods for dichotomous and multicategory outcome: theory. *Biometrical Journal*, 56(4):534–563, 2014.

B. K. Lee, J. Lessler, and E. A. Stuart. Improving propensity score weighting using machine learning. *Statistics in medicine*, 29(3):337–346, 2010.

C. Li and A. Cutler. Probability estimation in random forests. 2013.

T. Liang, T. Poggio, A. Rakhlin, and J. Stokes. Fisher-rao metric, geometry, and complexity of neural networks. *arXiv preprint arXiv:1711.01530*, 2017.

A. Liaw and M. Wiener. Classification and regression by randomforest. *R News*, 2 (3):18–22, 2002. URL `http://CRAN.R-project.org/doc/Rnews/`.

H. W. Lin, M. Tegmark, and D. Rolnick. Why does deep and cheap learning work so well? *Journal of Statistical Physics*, 168(6):1223–1247, 2017.

Y. Lin and Y. Jeon. Random forests and adaptive nearest neighbors. *Journal of the American Statistical Association*, 101(474):578–590, 2006.

D. Lock and D. Nettleton. Using random forests to estimate win probability before each play of an nfl game. *Journal of Quantitative Analysis in Sports*, 10(2):197–205, 2014.

G. Louppe, L. Wehenkel, A. Sutera, and P. Geurts. Understanding variable importances in forests of randomized trees. In *Advances in neural information processing systems*, pages 431–439, 2013.

J. D. Malley, J. Kruppa, A. Dasgupta, K. G. Malley, and A. Ziegler. Probability machines: consistent probability estimation using nonparametric learning machines. *Methods of Information in Medicine*, 51(1):74, 2012.

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

D. Mease, A. Wyner, and A. Buja. Cost-weighted boosting with jittering and over/under-sampling: Jous-boost. *Journal of Machine Learning Research*, 8:409–439, 2007.

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

B. Neyshabur, S. Bhojanapalli, D. McAllester, and N. Srebro. Exploring generalization in deep learning. *arXiv preprint arXiv:1706.08947*, 2017.

A. Niculescu-Mizil and R. Caruana. An empirical comparison of supervised learning algorithms using different performance metrics. Technical report, Cornell University, 2005.

M. Olson and A. J. Wyner. Making sense of random forest probabilities: a kernel perspective.

E. Parzen. On estimation of a probability density function and mode. *The annals of mathematical statistics*, 33(3):1065–1076, 1962.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011.

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

D. Rolnick, A. Veit, S. Belongie, and N. Shavit. Deep learning is robust to massive label noise. *arXiv preprint arXiv:1705.10694*, 2017.

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

R. E. Schapire and Y. Freund. *Boosting: Foundations and algorithms*. MIT press, 2012.

R. E. Schapire, Y. Freund, P. Bartlett, and W. S. Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *Annals of statistics*, pages 1651–1686, 1998.

E. Scornet. Random forests and kernel methods. *IEEE Transactions on Information Theory*, 62(3):1485–1500, 2016a.

E. Scornet. On the asymptotics of random forests. *Journal of Multivariate Analysis*, 146:72–83, 2016b.

E. Scornet, G. Biau, J.-P. Vert, et al. Consistency of random forests. *The Annals of Statistics*, 43(4):1716–1741, 2015.

C. Semple and A. Mike. *Phylogenetics*. Oxford University Press on Demand, 2003.

J. Shawe-Taylor and N. Cristianini. *Kernel methods for pattern analysis*. Cambridge university press, 2004.

A. J. Smola and B. Schölkopf. *Learning with kernels*. GMD-Forschungszentrum Informationstechnik, 1998.

G. B. Sologub. On measuring of similarity between tree nodes. 2011.

N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958, 2014.

T. M. Therneau and E. 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.

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

L. Wang, M. Sugiyama, Z. Jing, C. Yang, Z.-H. Zhou, and J. Feng. A refined margin

analysis for boosting algorithms via equilibrium margin. *The Journal of Machine Learning Research*, 12:1835–1863, 2011.

A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht. The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems*, pages 4151–4161, 2017.

D. H. Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992.

A. 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.

C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning requires rethinking generalization. *International Conference on Learning Representations*, 2017.

P. Zhao, X. Su, T. Ge, and J. Fan. Propensity score and proximity matching using random forest. *Contemporary clinical trials*, 47:85–92, 2016.

Z.-H. Zhou. *Ensemble methods: foundations and algorithms*. CRC press, 2012.