Random Forests and Adaptive Nearest Neighbors

Yi LIN and Yongho JEON

In this article we study random forests through their connection with a new framework of adaptive nearest-neighbor methods. We introduce a concept of potential nearest neighbors (k-PNNs) and show that random forests can be viewed as adaptively weighted k-PNN methods. Various aspects of random forests can be studied from this perspective. We study the effect of terminal node sizes on the prediction accuracy of random forests. We further show that random forests with adaptive splitting schemes assign weights to k-PNNs in a desirable way: for the estimation at a given target point, these random forests assign voting weights to the k-PNNs of the target point according to the local importance of different input variables. We propose a new simple splitting scheme that achieves desirable adaptivity in a straightforward fashion. This simple scheme can be combined with existing algorithms. The resulting algorithm is computationally faster and gives comparable results. Other possible aspects of random forests, such as using linear combinations in splitting, are also discussed. Simulations and real datasets are used to illustrate the results.

KEY WORDS: Adaptive estimation; Boosting; Classification trees; Randomized trees; Regression trees.

1. INTRODUCTION

In recent years, a number of new regression and classification methods have been proposed in the machine learning literature. Examples include support vector machines, boosting, and random forests. These methods have been very successful in empirical studies, and their working mechanisms are under active investigation. One particularly fruitful approach to understanding these new methods is through discovering their connections with existing methods that are well understood statistically. It has been shown that support vector machines are closely related to the penalty method extensively studied in nonparametric statistics literature, and boosting can be viewed as a form of greedy stagewise modeling. The connections give researchers new perspectives in analyzing and improving these methods. In this article we provide a framework connecting random forests with nearest-neighbor methods, and study the statistical properties of random forests through this connection. We focus mainly on regression problems, but also discuss classification problems.

Consider independent and identically distributed observations $\{(\mathbf{x}_i, y_i), i = 1, \ldots, n\}$ of a random pair $(\mathbf{X}, Y)$. Here $\mathbf{X} = (X^{(1)}, \ldots, X^{(d)}) \in \mathbb{R}^d$ is the input vector and $Y \in \mathbb{R}$ is the response variable. We wish to estimate the regression function, $g(\mathbf{x}) = E(Y|\mathbf{X} = \mathbf{x})$. For any point $\mathbf{x}_0 \in \mathbb{R}^d$, the mean squared error (MSE) at $\mathbf{x}_0$ of an estimator $\hat{g}(\mathbf{x}_0)$ of $g(\mathbf{x}_0)$ is

$$\begin{aligned} MSE[\hat{g}(\mathbf{x}_0)] &= E[\hat{g}(\mathbf{x}_0) - g(\mathbf{x}_0)]^2 \\ &= \big[E\big(\hat{g}(\mathbf{x}_0) - g(\mathbf{x}_0)\big)\big]^2 + \operatorname{var}(\hat{g}(\mathbf{x}_0)) \\ &= \text{bias}^2 + \text{variance}. \end{aligned}$$

The integrated MSE is $IMSE(\hat{g}) = E_{\mathbf{X}} MSE[\hat{g}(\mathbf{X})]$.

Nearest-neighbor methods have been studied extensively in the fields of nonparametric statistics and pattern recognition. (See Fix and Hodges 1951; Dasarathy 1991; Ripley 1996; Hastie, Tibshirani, and Friedman 2001 for summaries of the extensive literature on the topic.) The nearest-neighbor approach is based on the assumption that the regression function is approximately constant in the neighborhood. Given a distance metric, the k nearest-neighbor (k-NN) method estimates $g(\mathbf{x}_0)$ by looking at the k sample points that are closest to $\mathbf{x}_0$: $\hat{g}(\mathbf{x}_0) = \sum_{i=1}^n w_i y_i$, where the weight w_i is $1/k$ for the k nearest neighbors of $\mathbf{x}_0$ and 0 otherwise. Different distance measures can be used in the nearest-neighbor method. Ideally, for a target point $\mathbf{x}_0$, given the number k, the distance measure should be chosen to yield the k nearest neighbors that have mean response values as close to $g(\mathbf{x}_0)$ as possible (see Hastie and Tibshirani 1996 for a discussion).

Example 1. To illustrate, we generated $n = 1{,}000$ sample input points $\{\mathbf{x}_i = (x_i^{(1)}, x_i^{(2)}), i = 1, \ldots, n\}$ uniformly in $[0, 1]^2$. The response was generated according to $Y = g(\mathbf{X}) + \epsilon$, where $g(\mathbf{x}) = [x^{(2)}]^2$, and $\epsilon \sim \text{N}(0, .2^2)$ was simulated independently of $\mathbf{X}$. Figure 1(a) gives a neighborhood of $\mathbf{x}_0 = (.5, .5)$ defined by the ordinary Euclidean metric and containing $k = 100$ sample points. If we knew the underlying regression function, then we could find the neighborhood of $\mathbf{x}_0$ that contains the 100 sample points whose regression function values are the closest to $g(\mathbf{x}_0)$. That ideal neighborhood is shown in Figure 1(b). It stretches out all the way in the $x^{(1)}$ direction.

Several authors have studied the problem of how to adaptively choose the metric distance for nearest-neighbor methods (see Short and Fukunaga 1981; Myles and Hand 1990; Friedman 1994; Hastie and Tibshirani 1996; Domeniconi, Peng, and Gunopulos 2002). The central idea behind these existing adaptive nearest-neighbor methods is to select k-NNs according to an adaptively chosen local metric. In Section 2 we introduce a different paradigm by considering the set of potential nearest neighbors (k-PNNs) that consists of all sample points that can be made a k-NN by choosing a reasonable distance metric. In addition, we show that there is a natural connection between the adaptive weighted PNN approach and random forests.

Random forests are classification and regression methods based on growing multiple randomized trees. Each tree is grown using a randomized tree building scheme. The predictions of M randomized trees are averaged to give the final prediction. Each randomized tree is typically constructed without pruning (Breiman 1999). The tree building continues until each terminal node contains no more than k training sample points for some prespecified k. Different random forests differ in how randomness is introduced in the tree building process. Bagging (Breiman 1996) proceeds by growing trees on bootstrap samples of the training dataset. Randomized outputs (Breiman

Yi Lin is Associate Professor (E-mail: *yilin@stat.wisc.edu*) and Yongho Jeon is a Graduate Student (E-mail: *yjeon@stat.wisc.edu*), Department of Statistics, University of Wisconsin, Madison, WI 53706. This work was supported in part by National Science Foundation grant DMS-01-34987. The authors thank Leo Breiman for helpful comments and suggestions.

Journal of the American Statistical Association
June 2006, Vol. 101, No. 474, Theory and Methods
DOI 10.1198/016214505000001230

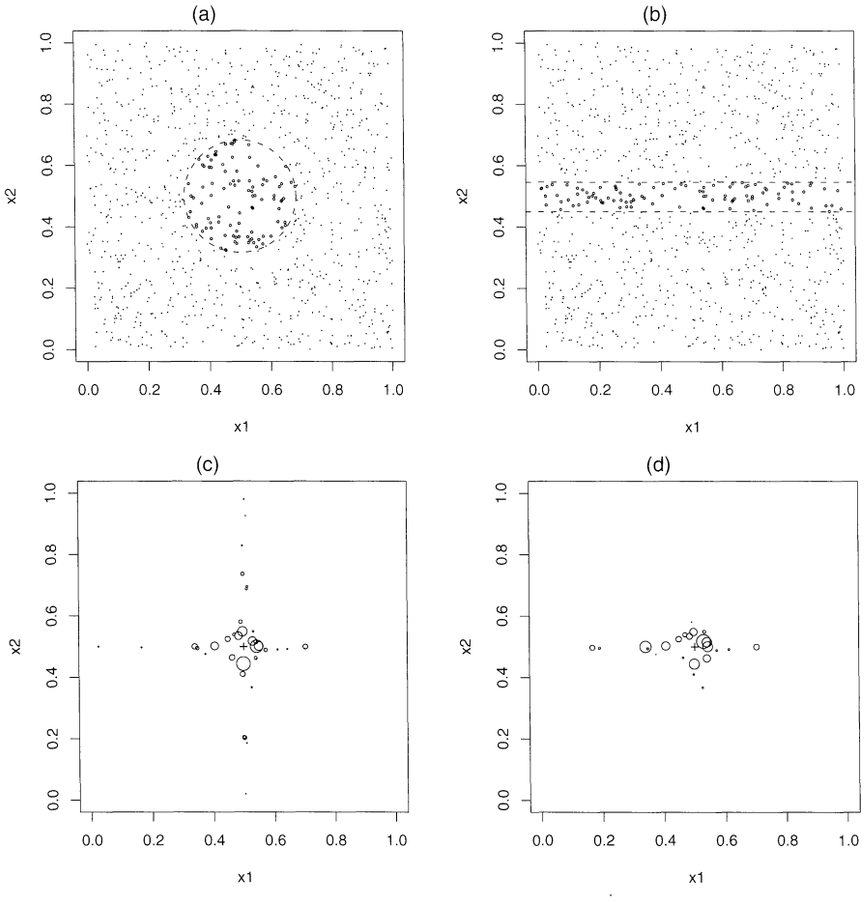

Figure 1. Neighboring Points of $\boldsymbol{x}_0 = (.5, .5)$ Used to Estimate $g(\boldsymbol{x}_0)$ in Example 1. (a) Nearest neighbors according to Euclidean distance. (b) Nearest neighbors according to the ideal distance metric. (c) Voting points and their weights in the random forest with the purely random splitting scheme. The areas of the symbols are proportional to the weights. (d) Voting points and their weights in the random forest with the random side selection.

1998) grows trees on the training set after randomly perturbing the output variable. Random split selection (Dietterich 2000) grows trees on the original training dataset. For a fixed number S, at each node, S best splits (in terms of minimizing deviance) are found, and the actual split is randomly selected from them. Random feature selection (Amit and Geman 1997; Breiman 1999) searches for the best split over a random subset of the features at each node. Random subspace (Ho 1998) grows each tree with a random subset of the features. Perfect random trees (Cutler 1999; Cutler and Zhao 2000) uses an extreme randomness: At each node, randomly choose a variable to split on, and on the chosen variable randomly choose a split point between two randomly chosen points coming from different classes.

Several authors have studied the statistical mechanism of random forests. Breiman (1999) showed that the expected misclassification rate is bounded by an expression related to the expected accuracy of the individual classifiers and inversely related to the expected correlation between pairs of classifiers. This bound is loose, but suggestive. Breiman (2000) showed that in the population space for binary classification problems, random forests are approximately kernel smoothers. Cutler and Zhao (2000) showed that their perfect random trees method is fitting a blockwise linear surface and can be computed directly using a recursive equation. Friedman and Hall (1999) and Bühlmann and Yu (2002) provided some explanations of how bagging works.

In this article we concentrate on random forests that use the original training data (not bootstrap samples) in the construction of randomized trees. Most tree building schemes use splits on the input variables. We concentrate on random forests with such splitting schemes everywhere except Section 5, where we consider random forests using linear combinations of input variables for splitting.

We start by looking at random forests locally at a target point $\mathbf{x}_0$. Given the training data $\{(\mathbf{x}_i, y_i), i = 1, \ldots, n\}$, at a target point $\mathbf{x}_0$, the prediction from the mth randomized tree is $\sum_{i=1}^{n} W_{im} y_i$, with the weight W_{im} as $1/k_m$ if $\mathbf{x}_i$ is among the k_m sample points in the terminal node containing the target point $\mathbf{x}_0$ and 0 otherwise. Averaging over M trees, the random forest prediction at $\mathbf{x}_0$ is $\sum_{i=1}^{n} \bar{W}_i y_i$, with $\bar{W}_i = (1/M) \sum_{m=1}^{M} W_{im}$. Therefore, the random forest can be viewed as a weighted average of the y_i's. Because $\sum_{i=1}^{n} W_{im} = 1$, we have

$$\sum_{i=1}^{n} \bar{W}_i = 1. \tag{1}$$

It is clear that the weight $\bar{W}_i$ is 0 for most of the sample points. The sample points with positive weights are called "vot-

ing points" for estimating $g(\mathbf{x}_0)$. We show in Section 2 that the voting points are all k-PNNs, where k is the terminal node size of the randomized trees.

Random forests with different splitting schemes assign weights to k-PNNs in different ways. A nonadaptive splitting scheme partitions the input space in a way that is independent of the response y_i's given the input vectors $\mathbf{x}_i$'s. One type of nonadaptive splitting scheme is purely random splitting; for each internal node, we randomly choose a variable to split on and choose the split point randomly among all possible split points on that variable. Most splitting schemes used in practice depend on the response y_i's. One example is the random input selection (Breiman 1999); at each node, a small group of F input variables are randomly selected, and the best split is searched for over these F input variables. For convenience, we refer to the random input selection with $F = 1$ as random side selection.

Example 1 (Continued). We applied the purely random scheme and the random side selection to the data in Example 1 and $\mathbf{x}_0 = (.5, .5)$. The terminal node size is $k = 2$. Both random forests were built to contain $M = 1{,}000$ randomized trees. The voting points and their weights for estimating $g(\mathbf{x}_0)$ are shown in Figures 1(c) and 1(d).

It can be seen that in the purely random splitting case, the voting points and weights are roughly equally spread out in the two dimensions. In the random side selection case, the voting weights are spread out in the $x^{(1)}$ direction but are concentrated around .5 in the $x^{(2)}$ direction. This is qualitatively similar to the ideal neighborhood discussed earlier. This suggests that the random forest with random side selection adaptively assigns weights among the voting points.

We study aspects of random forests from the perspective of adaptively weighted PNNs. The key elements in the construction of random forests are the splitting scheme and the specified maximum terminal node size k. In Section 3 we study the effect of the terminal node size k on the statistical accuracy of random forests. The conventional wisdom is that random forests work best using the largest trees possible ($k = 1$). We show that it is advantageous to tune the terminal node size for the best performance of random forests, especially in relatively low-dimensional problems with large sample size, although large trees usually give the best performance in high-dimensional problems. In Section 4 we study the effects of splitting schemes. We show that the random forest with random input selection assigns weights to the k-PNNs of the target point according to the local importance of different input variables. We introduce random point selection, a new, simple randomization scheme that achieves adaptiveness in a more straightforward fashion.

Breiman (1999) introduced a random forest, Forest-RC, that uses linear combinations of input variables in splitting. We consider the impact of using linear combinations in Section 5. The random point selection can be naturally used with linear combinations. The resulting algorithm achieves results similar to those of Forest-RC but is computationally faster.

Boosting is another regression and classification procedure based on multiple trees. Since its inception (Schapire 1990; Freund 1995; Freund and Schapire 1996), boosting has become one of the most successful methods. The popular boosting procedure Adaboost can be viewed as stagewise fitting of additive models under the exponential loss function, and many variants of boosting with different loss function using the gradient descent view have been proposed (Breiman 1998; Mason, Baxter, Bartlett, and Frean 2000; Friedman, Hastie, and Tibshirani 2000; Friedman 2001). An explanation of how boosting and random forests work differently can be given in the bias–variance trade-off framework. Unlike random forests, where the randomized trees are grown independently given the data, in a boost procedure trees enter stagewise without backfitting: each tree is grown on the residuals from existing trees. This makes it possible for boosting procedures to have much smaller bias than a single tree. It has been found that large trees are not necessary for boosting, and small trees are typically used (Friedman 2001). Therefore, this bias reduction aspect of boosting procedure is particularly important, because a single small tree tends to have large bias. In contrast, the averaging in random forests reduces variance but does not reduce bias. Random forests work best with large trees; therefore, the bias for each single tree is small (if the largest tree is used, the bias is close to 0), and hence random forests can still work well.

It also should be clear that the smallest bias does not translate into the best performance. It is well known that boosting forever (until convergence), which has the smallest bias, leads to overfitting, and that some form of regularization (typically early stopping) is necessary. The result in this article that random forests with the largest trees may not be optimal is in a similar spirit.

To give a sense of how the performance of random forests compares with that of other related methods, we have included simulation results of random forests, two boosting procedures, and two adaptive nearest-neighbor methods in Section 5. Discussion and summaries are given in Section 6, and proofs are given in the Appendix.

2. POTENTIAL NEAREST NEIGHBORS AND RANDOM FORESTS

Throughout this article, a hyperrectangle refers to a region of the form $\bigotimes_{j=1}^{d}[a^{(j)}, b^{(j)}]$. A hyperrectangle defined by two points $\mathbf{a}$ and $\mathbf{b}$ is a hyperrectangle with the two points as opposing vertices. In Euclidean space it is reasonable to require that distance measures satisfy the following monotonicity condition: For any two points $\mathbf{a}$ and $\mathbf{b}$, any point $\mathbf{c}$ in the hyperrectangle defined by $\mathbf{a}$ and $\mathbf{b}$ is closer to $\mathbf{a}$ than $\mathbf{b}$ is. This is intuitively clear because $\mathbf{c}$ is closer to $\mathbf{a}$ than $\mathbf{b}$ is in every dimension j, $j = 1, \ldots, d$. We call any distance measure in the Euclidean space satisfying this monotonicity property a "monotone distance" measure. Such measures include most of the common distance measures in the Euclidean space, such as any scaled L_q measures, $0 < q \leq \infty$.

Definition 1. A sample point $\mathbf{x}_i$ is called a *k-potential nearest neighbor* (k-PNN) to a target point $\mathbf{x}_0$ if there exists a monotone distance metric under which $\mathbf{x}_i$ is among the k closest to $\mathbf{x}_0$ among all of the sample points.

Therefore, any k-PNN is a k-NN under a suitably chosen monotone metric. The number of k-PNNs is typically much larger than k and depends on the number and configuration of the sample points.

Existing adaptive nearest-neighbor methods can be seen to choose k sample points from the set of all k-PNNs according to an adaptively chosen local metric. Instead, we may consider looking directly at the set of all of the k-PNNs and adaptively choosing sample points from this set, or adaptively assigning weights to the points in this set (adaptively weighted k-PNN). If K k-PNNs are chosen by an adaptive selection scheme, then in general these K points are not the K nearest neighbors to $\mathbf{x}_0$ under any single distance metric. Therefore, the adaptive PNN approach is different than the existing adaptive nearest-neighbor paradigm.

The proof of the following proposition is straightforward and thus is omitted.

Proposition 1. A sample point $\mathbf{x}_i$ is a k-PNN to $\mathbf{x}_0$ if and only if there are fewer than k sample points other than $\mathbf{x}_i$ in the hyperrectangle defined by $\mathbf{x}_0$ and $\mathbf{x}_i$.

Remark 1. In an exercise (problem 11.6), Devroye, Györfi, and Lugosi (1996) introduced the so-called "layered nearest-neighbor rule" as an example of scale-invariant classification rules. See Figure 2 for an illustration of the layered nearest neighbors. The layered nearest neighbors turn out to be the same as the 1-PNNs.

Consider (regression and classification) random forests with terminal node size k. We can now show that the voting points for a target point $\mathbf{x}_0$ belong to the set of k-PNNs of $\mathbf{x}_0$ regardless of the splitting scheme used. The terminal nodes of each randomized tree define rectangular areas. If a sample point $\mathbf{x}_i$ is not a k-PNN of $\mathbf{x}_0$, then there are more than k sample points (including $\mathbf{x}_i$) in the hyperrectangle defined by $\mathbf{x}_i$ and $\mathbf{x}_0$. Therefore, $\mathbf{x}_i$ cannot be in the terminal node containing $\mathbf{x}_0$. In other words, only k-PNNs can become a voting point. Thus we can view random forests as a weighted k-PNN method. For many splitting schemes, all k-PNNs of the target point have positive probabilities of being in the same terminal node as the target point; therefore, all k-PNNs can become voting points.

The following results give some idea of how many k-PNNs exist for a given target point. These results are used in Section 3. Consider a random sample of n points $\{\mathbf{x}_i, i = 1, \ldots, n\}$ from a density function $f(\mathbf{x})$ supported on $[0, 1]^d$. Let $A_k(n, d, \mathbf{x}_0, f)$ denote the expected number of k-PNNs of a fixed point $\mathbf{x}_0 \in [0, 1]^d$. For example, $A_k(n, d, \mathbf{0}, 1)$ is the expected number of k-PNNs of the origin $\mathbf{0} = (0, \ldots, 0)$ among n uniform random points in $[0, 1]^d$.

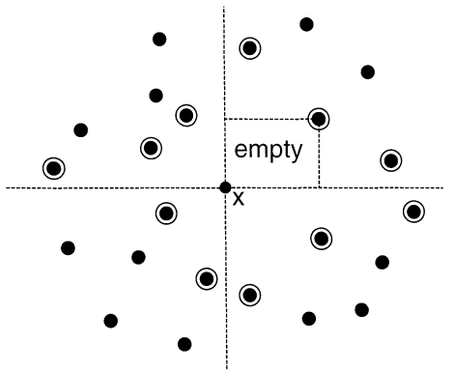

Figure 2. The Layered Nearest Neighbors.

Theorem 1. $A_k(n, d, \mathbf{0}, 1) = \{k(\log n)^{d-1}/(d-1)!\}[1+o(1)]$, as $n \to \infty$.

Lemma 1. $A_k(n, d, \mathbf{0}, 1) \le A_k(n, d, \mathbf{x}_0, 1) \le 2^d A_k(n, d, \mathbf{0}, 1)$, for any $\mathbf{x}_0 \in [0, 1]^d$.

Theorem 2. Assume that the density $f(\mathbf{x})$ is bounded away from 0 and infinity in $[0, 1]^d$. Then there exists $0 < \Lambda_1 \le \Lambda_2$, such that

$$\Lambda_1 k(\log n)^{d-1} \le A_k(n, d, \mathbf{x}_0, f) \le \Lambda_2 k(\log n)^{d-1}$$

for any $\mathbf{x}_0 \in [0, 1]^d$ and n; that is, the expected number of k-PNNs is of order $k(\log n)^{d-1}$.

3. TERMINAL NODE SIZE OF RANDOMIZED TREES

In this section we consider the effect of the maximum terminal node size k on the prediction accuracy of random forests. We first consider random forests with nonadaptive splitting schemes.

Theorem 3. Consider the regression problem $Y = g(\mathbf{X}) + \epsilon$ with $E(\epsilon) = 0$ and $\mathrm{var}(\epsilon) = \sigma^2$. Assume that $\mathbf{X}$ is distributed in $[0, 1]^d$ and that its density is bounded away from 0 and infinity in $[0, 1]^d$. Let $\{(\mathbf{x}_i, y_i), i = 1, \ldots, n\}$ be a random sample. Consider an estimator $\hat{g}$ resulting from a nonadaptive random forest with terminal node size k. There exists $\Lambda_3 > 0$ such that for any n,

$$MSE[\hat{g}(\mathbf{x}_0)] \ge \Lambda_3 k^{-1}(\log n)^{-(d-1)}, \qquad \forall\, \mathbf{x}_0 \in [0, 1]^d, \quad (2)$$

and

$$IMSE(\hat{g}) \ge \Lambda_3 k^{-1}(\log n)^{-(d-1)}. \quad (3)$$

Theorem 3 states that a lower bound to the rate of convergence of the MSE of random forests with nonadaptive splitting schemes is $k^{-1}(\log n)^{-(d-1)}$. On the other hand, it is well known (Stone 1980) that the optimal convergence rate of MSE in regression problems is $n^{-2m/(2m+d)}$, where m is the order of smoothness of the regression function g. The smoothness condition can be made precise by means of Sobolev spaces or Hölder classes, but roughly it means that the mth order derivative of g exists. The convergence rate $n^{-2m/(2m+d)}$ is a standard rate that can be achieved by many nonparametric methods. For smooth functions ($m \ge 1$), the convergence rate $n^{-2m/(2m+d)}$ is clearly much faster than the convergence rate achievable by nonadaptive random forests if the terminal node size k does not vary with the sample size n. To make it possible for nonadaptive random forests to achieve the standard convergence rate, the terminal node size k should be made to increase with the sample size n. Therefore, growing large trees (k being a small constant) does not always give the best performance. Intuitively, random forests with largest trees use only the 1-PNNs of $\mathbf{x}_0$ in the estimation of $g(\mathbf{x}_0)$, and the number of 1-PNNs is of order $O_p[(\log n)^{d-1}]$, which is not large enough for the estimator to achieve the standard rates of convergence for smooth functions.

The foregoing asymptotic argument applies only to situations where the sample size is large compared with the dimension of the problem. In practice, however, we often encounter high-dimensional problems with moderate sample size. In such problems the number $(\log n)^{d-1}/(d-1)!$ may not be smaller than $n^{2m/(2m+d)}$, even if the asymptotics indicate otherwise.

Take $d = 10$ and $m = 2$, for example. When $n = 100{,}000$, $(\log n)^{d-1}/(d-1)! = 9{,}793$, whereas $n^{2m/(2m+d)} = 27$. So $(\log n)^{d-1}/(d-1)!$ is actually much larger than $n^{2m/(2m+d)}$. This is even more true for larger d and smaller n. Therefore, in high-dimensional problems, growing the largest trees often gives the best performance for nonadaptive random forests.

We believe that the results for nonadaptive random forests apply qualitatively to adaptive random forests; growing the largest trees is not optimal in general. We carried out a simulation study to see how the prediction error of random forests varies with the terminal size k. The dataset used (Friedman#2) is available in the R library "mlbench." It originated with Friedman (1991) and was also described by Breiman (1996). There are four independent input variables uniformly distributed over the ranges $0 \le x^{(1)} \le 100$, $40\pi \le x^{(2)} \le 560\pi$, $0 \le x^{(3)} \le 1$, and $1 \le x^{(4)} \le 11$. The response is

$$y = \left[\left(x^{(1)}\right)^2 + \left(x^{(2)}x^{(3)} - \left(1/\left(x^{(2)}x^{(4)}\right)\right)\right)^2\right]^{1/2} + \epsilon,$$

where $\epsilon \sim \mathrm{N}(0, \sigma^2)$. The noise level σ was taken to be 125 to give a signal-to-noise ratio of 3 to 1. In each run we generated $n = 1{,}000$ training examples and 1,000 test examples. The random forest was constructed with 100 randomized trees. The randomized trees were built with random input selection with the size of the random subset being $F = 3$. For each terminal node size k, we calculated the average squared error over the test set. We ran the simulation 100 times and averaged. The result is given in the upper left panel of Figure 3. We can see that the test error goes down as k goes up, the minimum test error is achieved at $k = 20$ or larger.

We then increased the dimension of the regression problem by throwing in six uniform noise input variables, and ran the same experiment on the new problem. The result is given in the upper middle panel of Figure 3. The minimum test error is achieved at $k = 3$.

Instead of increasing the dimension, in the third case we decreased the sample size of the training set to 200 and repeated the experiment. The minimum of the test error is achieved at $k = 5$ (the upper right panel of Fig. 3). These experiments confirm our expectation that growing the largest trees may not be optimal, and that growing the largest trees is often close to optimal when the dimension of the problem is high or the sample size is small.

For the same three cases, we also ran the random input selection with bagging; that is, each tree is built with random input selection on a bootstrap sample of the training data. The results remain qualitatively the same and are shown in the lower panels of Figure 3.

Similar to regression random forests, classification random forests with largest trees may not give optimal performance when the dimension is low and the sample size is large. We illustrate this with a very simple two-dimensional simulation. The positive and the negative class are both normal with a common covariance, identity matrix. The two class centers are $(0, 0)$ and $(1, 1)$. We generated 500 positive examples and 500 negative examples. We used random input selection with $F = 1$ in classification trees and the Gini index when searching for the best splits. We varied the terminal node size k from 1 to 50 and built the random forests with 100 randomized trees. We calculated the misclassification rate on a test set of size 1,000 generated similarly to the training set. We repeated the whole process 200 times and averaged.

The results are shown in Figure 4. The dotted lines give the standard error calculated from the 200 replicates. We see that as the terminal node size increases from 1 to 50, misclassification decreases monotonically, from $>30\%$ to $<26\%$. Therefore, the largest trees are not optimal in this example.

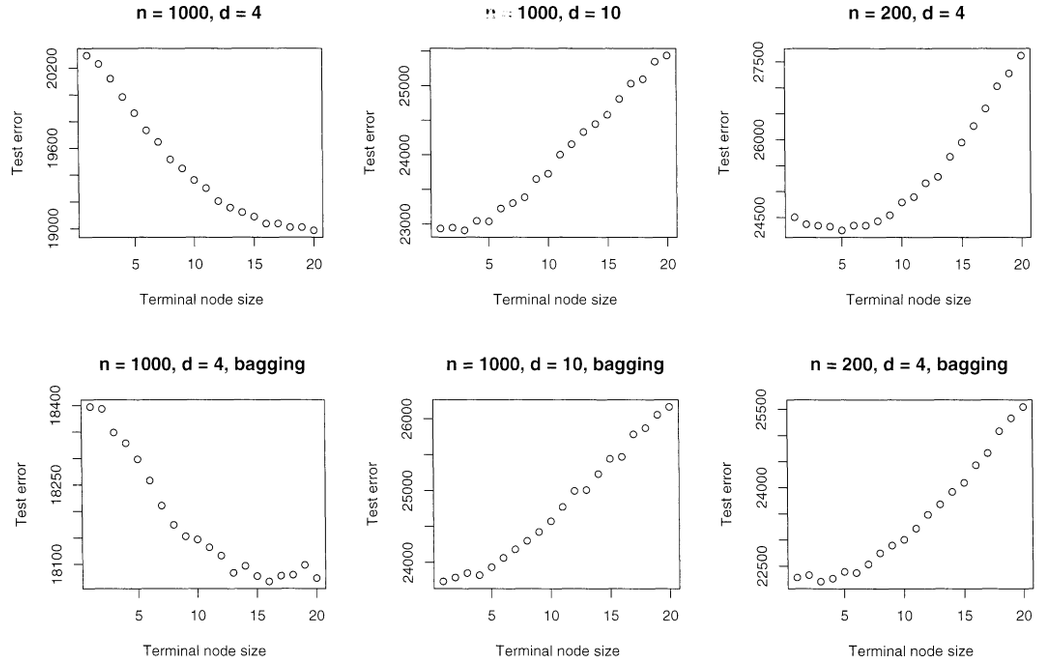

Figure 3. The Prediction Error of Regression Random Forests Varies With the Terminal Node Size k in the Regression Problems Discussed in Section 3. The lower panels give the results for random forests built with bootstrapping.

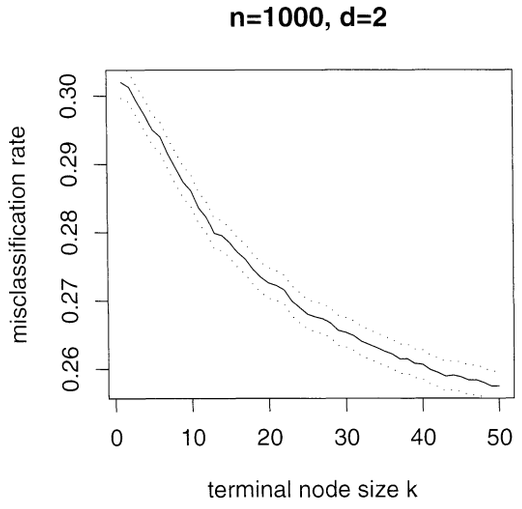

Figure 4. The Misclassification Rate of the Random Forest Varies With the Terminal Node Size k in the Classification Problem Discussed in Section 3. The dotted lines represent the standard error estimate.

Breiman (1999) introduced random forests as general tools for regression and classification in practical problems. In many empirical situations, the datasets are high-dimensional, with the sample size small relative to the dimension. In such situations, the largest trees are usually optimal; in other situations, it is better to tune the tree size for best performance.

4. SPLITTING SCHEMES

Given the terminal node size k, the assignment of weights to the k-PNNs is determined by the splitting scheme of the tree construction. A nonadaptive tree-growing scheme gives rise to a linear smoother, because the weights of the k-PNNs do not depend on the response variable, given the input vector. For most random forests, the tree-growing scheme depends on the response variable. In this section we investigate how these random forests assign weights to k-PNNs. We assume that $\mathbf{X}$ is uniform in $[0, 1]^d$.

We start with the simple random side selection, in which each variable has an equal chance of being split on. On the other hand, Figure 1(d) shows that the random forest with random side selection assigns weights adaptively to the k-PNNs in a desirable way; the weights are spread out in the direction that is not important, but are concentrated in the important direction. This is qualitatively similar to the ideal neighborhood. We argue that the adaptiveness derives from the difference between random splitting and deterministic splitting. We start with a formal characterization of this difference by contrasting the behavior of deterministic bisection and uniform random splitting.

First, we consider deterministic bisection on $[0, 1]$. For a fixed point $t_0 \in [0, 1]$, consider the interval sequence containing t_0 resulting from deterministic bisection. Denote the length of the Jth interval in the sequence by D_J. Then it is clear that $D_J = 2^{-J}$, $\forall J \geq 0$.

Now consider uniform random splitting. Fix $t_0 \in [0, 1]$. Starting from $[0, 1]$, at a current interval containing t_0, the next split is randomly uniform in the interval. Consider the interval sequence containing t_0 resulting from uniform random splitting. Denote the length of the Jth interval in the sequence by R_J; then R_J is a random variable. Simple calculation gives

$$E(R_1) = 1/2 + t_0(1 - t_0).$$

This is larger than D_1 whenever t_0 is not on the boundary of $[0, 1]$. An intuitive explanation of this difference is that for the random splitting scheme, the larger child node is more likely to contain t_0.

Theorem 4.

$$\frac{E(R_J)}{D_J} \geq 1 + 6t_0(1 - t_0)[1.2^J - 1], \qquad \forall J \geq 1.$$

We see that the ratio between the lengths of the intervals containing t_0 resulting from the two splitting schemes increases exponentially for any t_0 that is not on the boundary of $[0, 1]$; deterministic bisection gets to the target point more quickly than random splitting. This sheds light on the working mechanism of random side selection.

In random side selection, each input variable has an equal chance of being chosen to split. When we split on a variable of strong linear signal, weak noise, the best split is close to a deterministic bisection; when we split on a variable of very weak signal, the best split is close to a random split. Thus the splits on variables with strong linear signals are getting to the target point more quickly than the splits on variables with weak signals. Therefore, the terminal node containing the target point is on average narrower in dimensions with strong linear signals than in dimensions with weak signals. Thus random side selection achieves desirable local adaptivity.

To obtain further insight, we consider the linear regression function case, $Y = g(\mathbf{X}) + \epsilon$, with $g(\mathbf{x}) = a^{(0)} + \sum_{j=1}^{d} a^{(j)} x^{(j)}$. This is a situation where the relative importance of the input variables on $[0, 1]^d$ is clear cut: Variables corresponding to larger $|a^{(j)}|$'s are more important. More generally, in a hyperrectangle with the length of the sides being $r^{(j)}$, $j = 1, \ldots, d$, the importance of the jth variables is measured by $|a^{(j)}|r^{(j)}$. This is easily seen by scaling the hyperrectangle into the hypercube $[0, 1]^d$.

For a fixed point $\mathbf{x}_0$, consider the hyperrectangle around $\mathbf{x}_0$ with a fixed volume so that the responses of the points inside the hyperrectangle are the closest to $g(\mathbf{x}_0)$. This corresponds to the ideal hyperrectangle neighborhood of $\mathbf{x}_0$ that contains a fixed number of sample points. Let the length of the sides of this hyperrectangle be $q^{(j)}$, $j = 1, \ldots, d$. It is easy to see that they must satisfy that $q^{(j)}|a^{(j)}| = C$, $\forall j$, for some constant C; that is, in the ideal hyperrectangle neighborhood, all of the variables are equally important.

Now let us consider random side selection in this linear case. In $[0, 1]^d$, as discussed earlier, the lengths of the more important dimensions shrink faster on average. In a node with length $r^{(j)}$ in the jth direction, $j = 1, \ldots, d$, all directions will have the same shrinking rate on average if the $|a^{(j)}|r^{(j)}$ are the same for all directions j. Again, this can be seen by scaling the node to $[0, 1]^d$. In contrast, if the $|a^{(j)}|r^{(j)}$'s are not the same, then, similarly, the variables with larger $|a^{(j)}|r^{(j)}$ tend to shrink faster, and therefore $|a^{(j)}|r^{(j)}$ decreases faster. Thus in equilibrium, the relative lengths of the sides of the node should make all of the $|a^{(j)}|r^{(j)}$'s the same; that is, all variables are equally important in the node. Therefore, at equilibrium we get ideal hyperrectangular neighborhoods.

Example 2. Assume the same setup as in Example 1, except with the regression function $g(\mathbf{x}) = x^{(1)} + 3x^{(2)}$. Then the ideal adaptive hyperrectangular neighborhood should have side lengths of ratio 3:1. We grew random forests with 100 randomized trees, according to random side selection. The terminal node size is $k = 2$. We calculated the weighted spread of the voting points for $\mathbf{x}_0 = (.5, .5)$ in the two directions: $\sum_{i=1}^{n} \bar{W}_i |x_i^{(j)} - .5|$, $j = 1, 2$. Averaging over 100 realizations, the weighted spreads in the two dimensions are .0326 and .0207. The ratio is 1.6:1. We then conducted another experiment. The input vector $\mathbf{X}$ was uniformly generated from $[0, 1] \times [.4, .6]$, while keeping everything else the same. Therefore, the ratio of the length of the two sides in the starting node is 5:1. This time, the weighted spreads of the voting points in the two directions are .0244 and .00663. The ratio is 3.7:1.

This example suggests that the random side selection is heading to the equilibrium (the ideal ratio is 3:1), but does not quite get there. We believe that the random side selection gets to equilibrium so slowly because of the indirect nature in which random side selection achieves adaptivity.

It is possible to design simple splitting schemes that achieve adaptiveness in a more direct way. We consider the following. At the current node, on each input variable we randomly select a split point, and then choose the best split among these d splits. We call this scheme "random point selection." In random point selection, more important variables are more likely to be split. In the linear regression function case $g(\mathbf{x}) = a^{(0)} + \sum_{j=1}^{d} a^{(j)} x^{(j)}$, at equilibrium, all of the $|a^{(j)}| r^{(j)}$'s are the same in the node, where $r^{(j)}$ is the length of the jth direction of the node, $j = 1, \ldots, d$. This can be seen by scaling the node to $[0, 1]^d$. Therefore, at equilibrium, the relative spread of different directions produced by the random-point selection should be close to that of the ideal hyperrectangular neighborhood. It is our experience that random-point selection goes to the equilibrium much faster than random-side selection. When random-point selection is applied in Example 2, the average weighted spreads are .0381 and .0123 (ratio 3.1:1) starting from $[0, 1]^2$ and .0177 and .00654 (ratio 2.7:1) starting from $[0, 1] \times [.4, .6]$. These are close to the ideal ratio of 3:1.

When the regression function $g(\mathbf{x})$ is nonlinear, by approximating $g(\mathbf{x})$ locally with linear functions, we expect the relative spread of the voting weights in different dimensions to be determined by the relative magnitude of the partial derivatives of $g(\mathbf{x})$. The relative magnitude of the partial derivatives of the regression function provides a reasonable measure of the local importance of the input variables.

Example 3. Consider a setup that is similar to Example 2, but with $g(\mathbf{x}) = [x^{(1)}]^2 + [x^{(2)}]^2$. We consider the estimation at three points: (.75, .75), (.25, .75), and (.75, .25). For random forests with the random point selection, we calculate the weighted spread in the two directions of the voting points. Over 100 realizations, the average spreads are (.0219, .0229), (.0367, .0120), and (.0133, .0418) for the three target points. These ratios are close to the reciprocal of the ratio of partial derivatives at these points (1:1, 3:1, and 1:3). When random side selection is used, the corresponding numbers are (.0248, .0249), (.0345, .0156), and (.0174, .0421).

Random side selection is random input selection with random subset of size $F = 1$. For random input selection with $F > 1$, at each node the variables are not equally likely to be split. It is easy to see that more important variables are more likely to be split. In this sense, random input selection with $F > 1$ is similar to random point selection, and it can be shown through an argument similar to that for random point selection that in equilibrium, random input selection with $F > 1$ achieves desirable adaptivity.

5. LINEAR COMBINATIONS

One of the random forest algorithms considered by Breiman (1999), Forest-RC, uses linear combinations of input variables in the construction of randomized trees. For specified numbers L and F, at any internal node, L variables are randomly selected and added together with coefficients that are uniform random numbers on $[-1, 1]$. F such linear combinations are generated, and then a search is made over these for the best split.

For splitting schemes that use linear combinations of the input variables, the terminal nodes of the randomized trees do not define hyperrectangular areas. Thus the voting points in the random forests may not be the k-PNNs. For such random forests, the voting weights are allowed to be adaptively assigned according to which linear combination is more important. To illustrate, we revisit Example 2.

In Example 2, if we do not insist on hyperrectangular neighborhoods, then the ideal neighborhood of the point $\mathbf{x}_0 = (.5, .5)$, which contains sample points whose regression function values are the closest to $g(\mathbf{x}_0)$, should be parallel to and be centered at the line $x^{(1)} + 3x^{(2)} = 2$ (which goes through $\mathbf{x}_0$). Figure 5(a) shows the voting points and their weights when Forest-RC is applied to the data in Example 2. We used $L = 2$, $F = 25$, and $k = 4$ in the algorithm. The dotted line is the ideal center line, $x^{(1)} + 3x^{(2)} = 2$, and the dashed line is the principal component of the weighted voting points. We can see they are well matched, indicating that the voting weights are assigned according to the ideal direction.

Random point selection can be combined with Forest-RC in a natural way. For specified numbers L and F, at any internal node, F linear combinations are generated in the same way as in Forest-RC. For each of the linear combinations, we randomly use a split point, and then use the best split among these F random splits. This random combination point selection has the advantage of being computationally fast. Figure 5(b) shows the result of the random combination point selection on the data in Example 2. Again, the principal component of the weighted voting points matches the ideal center line.

We tested the random combination point selection on several real and synthetic regression problems. The datasets and the results are given in Table 1. The datasets are the same as those used by Breiman (1999), except that we excluded one dataset whose input variables are all categorical. The Abalone dataset is available at the UCI data repository. The robot arm data are provided by Leo Breiman (who obtained the data from Michael Jordan, according to Breiman 1999). The other datasets are available in the R library "mlbench." For the datasets Friedman#2 and Friedman#3, the noise levels are set so that the signal-to-noise ratio is 3:1. For the first two datasets, the test set error was estimated by training on a random 90% of

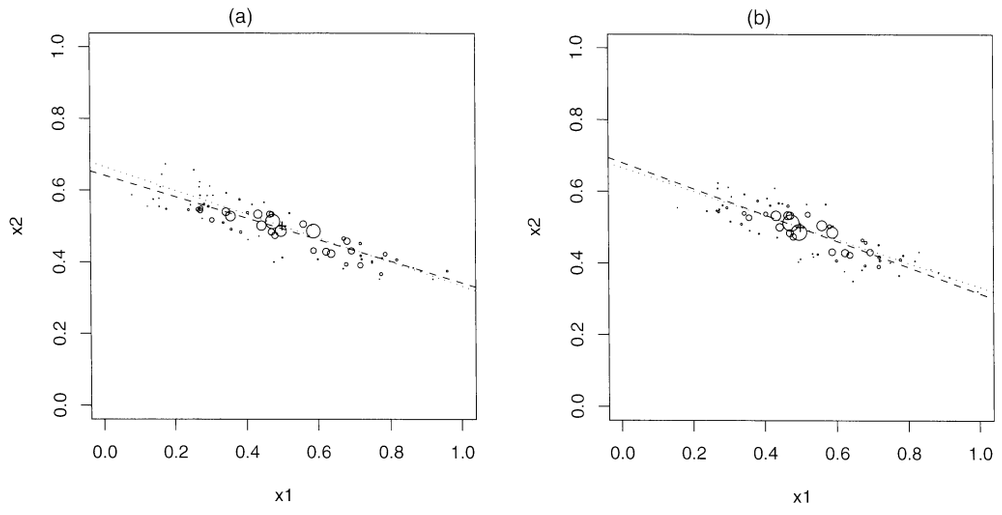

Figure 5. The Voting Points of Random Forests With the Random Linear Combination Selection (a) and the Random Linear Combination Point Selection (b). The dotted line is the ideal line, and the dashed line is the weighted principal component of the voting points.

the data and testing on the rest of the data. This was repeated 100 times, and the test set errors were averaged. The numbers in parentheses are the standard errors based on the repetitions. The Abalone dataset is larger and originally came with a randomly selected 25% of the instances to use as a test set. Following Breiman (1999), we trained on a randomly selected 75% of the data and tested on the rest. We repeated this 10 times. For the robot arm data, we used the given test set, so there is no standard error estimate. We ran the random combination point selection with $L = 2$, $F = 25$, and $k = 4$, the setting used by Breiman (1999) for Forest-RC. Tuning (e.g., k) may improve performance. The results for Forest-RC are copied from Breiman (1999).

We can see the random combination point selection (shown in Table 1 as Forest-RCP with number of cutpoints $Q = 1$) gives similar results to that of Forest-RC. A natural extension of the random combination point selection is to use $Q > 1$ random cutpoints on each linear combination. In fact, Forest-RC can be viewed as Forest-RCP with a very large Q. We tried Forest-RCP with $Q = 3$ and got similar results, possibly slightly better on the larger datasets.

We further tested our algorithm on a number of classification problems. The datasets used and the simulation results are given in Table 2. The datasets are the same as the classification datasets used by Breiman (1999), except that we excluded zip code data and other datasets involving categorical input variables. Our experiment thus include eight smaller datasets, two larger datasets, and four synthetic datasets. The datasets liver, image, and ecoli and the waveform database generator are available at the UCI data repository. The other datasets are available from the R library "mlbench." For the breast cancer dataset, which has 699 records, we deleted the 16 records containing missing values, leaving a sample size of 683. For the diabetes dataset, which consists of 768 records with 8 input variables, following Lim, Loh, and Shih (2000), we deleted the variable serum insulin because it contains many zero values that are physically impossible, and some records that have impossible values in other variables were also deleted. The resulting dataset consists of 532 records with 7 input variables.

We normalized the input variables by subtracting the mean and dividing by the standard deviation before conducting the experiments if the input variables are not commensurable. The image data contain one variable that is a constant for all of the records, we applied our algorithm without that variable.

On each of the 8 smaller sized datasets, we estimated the test set error by training on a random 90% of the data and testing on the rest of the data. We repeated this 100 times, and averaged the test set errors. Each of larger datasets consists of the given training set and test set. For the synthetic datasets, in each of 100 runs, we generated a new training set of size 300 and a test

Table 1. MSE of Forest-RC and Forest-RCP

				Mean squared test set error		
					Forest-RCP	
Dataset	*No. of trainings*	*No. of tests*	*No. of inputs*	*Forest-RC*	$Q = 1$	$Q = 3$
Boston housing	506	10%	12	10.2	$9.26_{(.38)}$	$9.38_{(.42)}$
Ozone	330	10%	8	16.3	$16.73_{(.44)}$	$17.11_{(.51)}$
Abalone	4,177	25%	8	4.6	$4.81_{(.13)}$	$4.58_{(.08)}$
Robot arm -2	15,000	5,000	12	4.2	3.9	3.8
Friedman#1	200	2,000	10	5.7	$5.54_{(.04)}$	$5.34_{(.04)}$
Friedman#2 $+3$	200	2,000	4	19.6	$19.34_{(.10)}$	$19.76_{(.10)}$
Friedman#3 -3	200	2,000	4	21.6	$20.46_{(.25)}$	$20.25_{(.23)}$

NOTE: The $+$ and $-$ following a dataset name indicate the power of 10 to multiply the result by; for example, the prediction error of Forest-RC on robot arm data is 4.2×10^{-2}.

Table 2. Misclassification Rate in Percentage

					Test set error		
						Forest-RCP	
Dataset	*No. of trainings*	*No. of tests*	*No. of inputs*	*Classes*	*Forest-RC*	$k = 1$	$k = 4$
Liver	345	10%	6	2	27.3	27.3	26.8
Ecoli	336	10%	7	8	12.9	13.7	13.6
Diabetes	532	10%	7	2	23.1	23.2	22.4
Glass	214	10%	9	6	24.4	23.5	24.3
Breast cancer	683	10%	9	2	3.0	3.0	2.7
Vehicle	846	10%	18	4	23.1	22.5	23.2
Image	2,310	10%	19	7	1.6	1.6	1.6
Sonar	208	10%	60	2	13.6	13.9	14.0
Letters	15,000	5,000	16	26	3.4	3.0	3.3
Sat images	4,435	2,000	36	6	9.1	8.8	9.2
Waveform	300	3,000	21	3	16.0	15.9	16.0
Twonorm	300	3,000	20	2	3.8	2.9	3.0
Threenorm	300	3,000	20	2	16.8	15.0	14.9
Ringnorm	300	3,000	20	2	4.8	4.9	4.5

set of size 3,000 and then computed the test set error. Breiman (1999) used Forest-RC with $k = 1$, $L = 3$, and $F = 2, 8$, with the choice for F between 2 and 8 decided by the out-of-bag estimate. He found that for the larger datasets, $F = 8$ gave better results and for on the smaller datasets, the results for $F = 2$ and $F = 8$ were very similar. We used $L = 3$, $F = 8$, and $Q = 3$ in the random combination point selection; the results are reported in Table 2. The results for Forest-RC are copied from Breiman (1999), except for the modified breast cancer and diabetes datasets, and we did the simulation for Forest-RC with $L = 3$ and $F = 8$.

Table 2 shows that the performance of the two methods are very close, with Forest-RCP having a slight advantage. The computation count of Forest-RCP is $O(MFLn \log n)$, where M is the number of randomized trees in the random forest. The main computational difference between Forest-RC and Forest-RCP is that for Forest-RC, in each node each random linear combination must be sorted and an exhaustive search performed to get the best split, whereas for Forest-RCP, only a small number of randomly selected splits must be compared, with no sorting or exhaustive search needed. The most commonly used sorting algorithm is the fast-sort procedure, which has an expected operation count $O(N \log N)$ and a worst-case count $O(N^2)$ when N numbers are sorted. Thus the savings of Forest-RCP is on the order of $\log n$ on average, and can be much larger in special cases.

To see the effect of the terminal node size k, we have also included the results of the Forest-RCP with terminal node size $k = 4$. The results confirm our expectation that when the dimension of the problem is high, the largest trees ($k = 1$) tend to be optimal, whereas when the dimension is low, the largest trees can be suboptimal. We use two values, $k = 1$ and $k = 4$, for all datasets, and better results can be achieved if we tune the number k for each dataset. For example, for the ecoli dataset, the misclassification rate is 11.3% when $k = 6$.

It is of interest to compare the performance of random forests and other related methods. Table 3 presents a comparison between random forests and two boosting algorithms, as well as two adaptive nearest-neighbor methods. The boosting algorithms are Adaboost, which minimizes the exponential loss in a stagewise fashion, and Logitboost, which uses the negative log-likelihood as the loss function. We used the boosting algorithm "gbm" package in the free software R contributed by Greg Ridgeway. According to the documentation, the coding closely follows that of Friedman (2001). The numbers of iteration in the boosting procedures between 1 and 1,000 are chosen by five-fold cross-validation. We use small tree sizes for boosting, as recommended by Friedman (2001), and consider tree sizes two and five. We find that the performance are very close, but that tree size five performs slightly better on these datasets for both Adaboost and Logitboost. The results reported in Table 3 are for tree size five. The two adaptive nearest-neighbor methods included in the simulation are Dann (Hastie and Tibshirani 1996) and Adamenn (Domeniconi et al. 2002). These are among the most effective adaptive nearest-neighbor methods. The codes that we use were kindly provided by C. Domeniconi. Dann has three tuning parameters: K_M, the number of nearest neighbors for estimation of the metric; K, the number of neighbors in the final nearest-neighbor rule; and ϵ, the "softening" parameter. Following Hastie and Tibshirani (1996), we fix $K_M = \max\{n/5, 50\}$ and $\epsilon = 1$ and tune K over $\{1, 3, 5, 10, 20\}$ based on five-fold cross-validation. Adamenn has six adjustable tuning parameters, K_0, K_1, K_2, L, K, and c. We fix $K_0 = \max\{.1n, 20\}$, $K_2 = .15n$, and $L = K_2/2$ following the suggestions of Domeniconi et al. (2002) and tune the other parameters over $K_1 \in \{1, 5\}$, $K \in \{1, 5, 20\}$, and $c \in \{1, 5, 10\}$ based on five-fold cross-validation. We concentrate on binary classification problems, so that the simulation can be done on all of the methods considered.

Table 3 shows that none of the methods dominates the other methods. Forest-RCP and the boosting algorithms are stable in that their performances are consistently not far away from the

Table 3. Comparison Among Related Methods

Dataset	*Forest-RCP*	*Adaboost*	*Logitboost*	*Dann*	*Adamenn*
Breast cancer	3.0	3.6	3.4	2.9	3.4
Diabetes	23.2	22.6	21.7	24.6	23.1
Sonar	13.9	16.1	13.7	10.9	14.0
Liver	27.3	27.6	26.6	33.9	36.4
Twonorm	2.9	5.4	5.4	4.2	3.3
Threenorm	15.0	19.0	18.9	16.4	15.3
Ringnorm	4.9	6.2	6.1	19.6	18.9

optimal performance. The adaptive nearest-neighbor methods Dann and Adamenn can sometimes suffer from poor performance on some datasets (e.g., liver and ringnorm).

Breiman (1999) provided a comparison between Forest-RC and Adaboost. In his simulation he used the largest tree for each step of Adaboost and fixed the total number of trees in Adaboost at 50. We use smaller trees in boosting and use five-fold cross-validation to adaptively choose the number of iterations in the boosting procedures.

6. SUMMARY AND DISCUSSIONS

Aspects of random forests can be studied from the perspective of adaptive PNNs. For fixed terminal node size k, the voting points of random forests for the estimation at a target point belong to the set of k-PNNs of the target point. The number of k-PNNs is much larger than k and increases with increasing sample size. In general, tuning k in random forests can enhance estimation accuracy, especially when the dimension of the problem is low and the sample size is large.

Two simple splitting schemes are random side selection and the random point selection. Both achieve desirable adaptivity, but in different ways. Random point selection achieves better adaptivity in a straightforward fashion and is computationally faster. The two basic algorithms can be combined in different ways to develop new algorithms. In particular, the random point selection scheme can be used in existing algorithms to replace the search for the best splits. The resulting algorithms are computationally faster.

One difference between random forests and a single regression tree is that by averaging over a large number of trees, the random forests method stabilizes the result to close to the equilibrium at which the desirable adaptivity is achieved. Another difference is that in a single tree, the prediction for any point in the same terminal node is the same, even when the target point is far from being at the center of the terminal node. In random forests, the target point is always close to the center of the voting points for the target point. This is desirable for enhancing estimation accuracy.

APPENDIX: PROOFS

A.1 Proof of Theorem 1

First, we establish the following lemma.

Lemma A.1. $A_k(n, d, \mathbf{0}, 1) = k + \sum_{\ell=k+1}^{n} A_k(\ell, d-1, \mathbf{0}, 1)/\ell$.

Proof. Let $\mathbf{x}_i, i = 1, \ldots, n$, be n uniform random points in $[0,1]^d$. Sort the points according to the $x^{(d)}$ direction. Let the points after sorting be $\mathbf{x}_{(i)}, i = 1, 2, \ldots, n$. Denote $\mathbf{x}_{(i)}^{(-d)} = (x_{(i)}^{(1)}, \ldots, x_{(i)}^{(d-1)})$. Then $\mathbf{x}_{(i)}^{(-d)}, i = 1, 2, \ldots, n$, are iid uniformly distributed in $[0,1]^{d-1}$. Let $Z_i = \mathbb{1}[\mathbf{x}_{(i)}$ is a k-PNN of $\mathbf{0}]$, that is,

$$Z_i = \begin{cases} 1 & \text{if } \mathbf{x}_{(i)} \text{ is a } k\text{-PNN of } \mathbf{0} \\ 0 & \text{otherwise.} \end{cases}$$

Then we have $A_k(n, d, \mathbf{0}, 1) = E[\sum_{\ell=1}^{n} Z_\ell] = \sum_{\ell=1}^{n} E[Z_\ell]$. Now it is easy to see that $Z_\ell = 1$ for $\ell = 1, \ldots, k$.

For $\ell > k$, we see that $\mathbf{x}_{(\ell)}$ is a k-PNN of $\mathbf{0}$ if and only if $\mathbf{x}_{(\ell)}^{(-d)}$ is a k-PNN of the origin among $\mathbf{x}_{(i)}^{(-d)}$, $i = 1, \ldots, \ell$, in $[0,1]^{d-1}$. The probability of the latter happening is $A_k(\ell, d-1, \mathbf{0}, 1)/\ell$ by symmetry, because the expected total number of k-PNNs of the origin among $\mathbf{x}_{(i)}^{(-d)}$, $i = 1, \ldots, \ell$, in $[0,1]^{d-1}$, is $A_k(\ell, d-1, \mathbf{0}, 1)$. Therefore,

$$E(Z_\ell) = \frac{A_k(\ell, d-1, \mathbf{0}, 1)}{\ell}, \qquad \forall\, \ell > k,$$

and the conclusion of the lemma follows.

Now return to the proof of the theorem. It is clear that $A_k(n, 1, \mathbf{0}, 1) = k$ for any $n \geq k$. Applying the lemma, we have

$$A_k(n, 2, \mathbf{0}, 1) = k\left(1 + \sum_{\ell=k+1}^{n} \frac{1}{\ell}\right),$$

the leading term of which is $k \log n$, because $\sum_{i=1}^{n} 1/i = \log n + \gamma + o(1)$, where $\gamma = .57\ldots$ is Euler's constant. We further get (where the notation $\approx$ keeps track of the leading term)

$$\begin{aligned} A_k(n, 3, \mathbf{0}, 1) &= k + \sum_{\ell=k+1}^{n} \frac{A_k(l, 2, \mathbf{0}, 1)}{l} \\ &\approx k\left(1 + \sum_{\ell=k+1}^{n} \frac{\log l}{l}\right) \\ &\approx k\left(1 + \int_{k+1}^{n} \frac{\log x}{x}\, dx\right) \\ &\approx \frac{k(\log n)^2}{2}. \end{aligned}$$

Continuing with this argument, by induction, we get $A(n, d, \mathbf{0}, 1) \approx k(\log n)^{d-1}/(d-1)!$.

A.2 Proof of Lemma 1

Let $\mathbf{x}_i$, $i = 1, \ldots, n$, be n uniform random points in $[0,1]^d$ and let $\mathbf{x}_0$ be any fixed point in $[0,1]^d$. Let $V_i = \mathbb{1}[\mathbf{x}_i$ is a k-PNN of $\mathbf{x}_0]$. Then $A_k(n, d, \mathbf{x}_0, 1) = \sum_i E(V_i) = nP(\mathbf{x}_1$ is a k-PNN of $\mathbf{x}_0)$. Now for $\mathbf{x}_1$ to be a k-PNN of $\mathbf{x}_0$, fewer than k points among $\mathbf{x}_2, \ldots, \mathbf{x}_n$ should be in the rectangle defined by $\mathbf{x}_0$ and $\mathbf{x}_1$. So, conditioning on $\mathbf{x}_1 = \mathbf{x}$, we have

$$P(\mathbf{x}_1 \text{ is a } k\text{-PNN of } \mathbf{x}_0 | \mathbf{x}_1 = \mathbf{x}) = B\big(k-1; n-1, h(\mathbf{x}, \mathbf{x}_0)\big),$$

where $B(k; n, p)$ is the cumulative distribution function of a binomial (n, p) distribution at k and $h(\mathbf{x}, \mathbf{x}_0) = \prod_{j=1}^{d} |x^{(j)} - x_0^{(j)}|$. Thus

$$A_k(n, d, \mathbf{x}_0, 1) = n \int_{[0,1]^d} B\big(k-1; n-1, h(\mathbf{x}, \mathbf{x}_0)\big)\, d\mathbf{x}. \tag{A.1}$$

In what follows, we proceed in the $d = 2$ case; the general case is similar. Divide the foregoing integral into integrals over $E_1, \ldots, E_4$ (Fig. A.1). Because $B(k; n, p)$ is decreasing in p, by appropriate change of variables, we can see that

$$\begin{aligned} &n \int_{E_i} B\big(k-1; n-1, h(\mathbf{x}, \mathbf{x}_0)\big)\, d\mathbf{x} \\ &\qquad \geq n \int_{E_i} B\big(k-1; n-1, h(\mathbf{x}, \mathbf{0})\big)\, d\mathbf{x}, \qquad i = 1, \ldots, 4. \end{aligned} \tag{A.2}$$

Equality holds in E_1; in other areas, inequality holds. This is also easy to see from a symmetry argument; given that $\mathbf{x}_1$ is in E_1, by symmetry, the probability of $\mathbf{x}_1$ being a k-PNN of the origin is the same as the probability of $\mathbf{x}_1$ being a k-PNN of $\mathbf{x}_0$. In all other areas, the probability of $\mathbf{x}_1$ being a k-PNN of the origin is less than the probability of $\mathbf{x}_1$ being a k-PNN of $\mathbf{x}_0$. By (A.2), we get $A_k(n, d, \mathbf{0}, 1) \leq A_k(n, d, \mathbf{x}_0, 1)$.

In contrast, it is clear that

$$\begin{aligned} &\int_{E_i} B\big(k-1; n-1, h(\mathbf{x}, \mathbf{x}_0)\big)\, d\mathbf{x} \\ &\qquad \leq \int_{[0,1]^d} B\big(k-1; n-1, h(\mathbf{x}, \mathbf{0})\big)\, d\mathbf{x}, \qquad i = 1, \ldots, 4; \end{aligned}$$

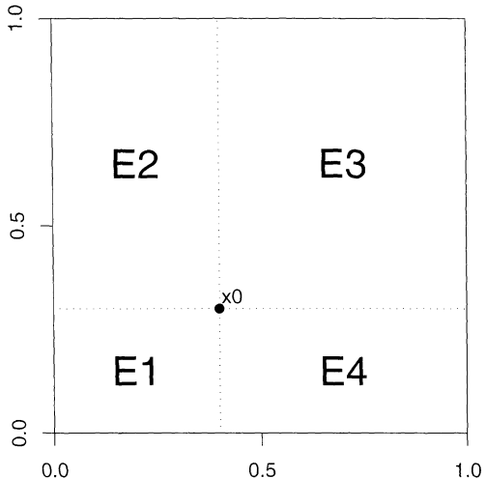

Figure A.1. The Illustration Figure Used in the Proof of Lemma 1.

for example,

$$\int_{E_3} B\big(k-1; n-1, h(\mathbf{x}, \mathbf{x}_0)\big)\, d\mathbf{x} = \int_{E_3-\mathbf{x}_0} B\big(k-1; n-1, h(\mathbf{x}, \mathbf{0})\big)\, d\mathbf{x} \le \int_{[0,1]^d} B\big(k-1; n-1, h(\mathbf{x}, \mathbf{0})\big)\, d\mathbf{x}.$$

The other areas can be treated similarly. This gives $A_k(n, d, \mathbf{x}_0, 1) \le 2^d A_k(n, d, \mathbf{0}, 1)$.

A.3 Proof of Theorem 2

Because the density $f(\mathbf{x})$ is bounded away from 0 and infinity, there exist positive constants $0 < C_1 \le 1 \le C_2 < \infty$, such that $C_1^d < f(\mathbf{x}) < C_2^d$. The conclusion of the theorem follows from Theorem 1 and the fact that for any $\mathbf{x}_0 \in [0, 1]^d$ and n,

$$(C_1/C_2)^d A_k(n, d, \mathbf{0}, 1) \le A_k(n, d, \mathbf{x}_0, f) \le 2^d (C_2/C_1)^d A_k(n, d, \mathbf{0}, 1). \quad \text{(A.3)}$$

In what follows, we prove (A.3). Following the proof of (A.1), we have

$$A_k(n, d, \mathbf{x}_0, f) = n \int_{[0,1]^d} B\big(k-1; n-1, h_1(\mathbf{x}, \mathbf{x}_0)\big) f(\mathbf{x})\, d\mathbf{x}, \quad \text{(A.4)}$$

with $h_1(\mathbf{x}, \mathbf{x}_0) = \int_{R(\mathbf{x},\mathbf{x}_0)} f(\mathbf{u})\, d\mathbf{u}$, where $R(\mathbf{x}, \mathbf{x}_0)$ is the hyperrectangle defined by $\mathbf{x}$ and $\mathbf{x}_0$. Now, because $B(k; n, p)$ is decreasing in p, we get, from (A.4),

$$n \int_{[0,1]^d} B\big(k-1; n-1, C_2^d h(\mathbf{x}, \mathbf{x}_0)\big) C_1^d\, d\mathbf{x} \le A_k(n, d, \mathbf{x}_0, f) \le n \int_{[0,1]^d} B\big(k-1; n-1, C_1^d h(\mathbf{x}, \mathbf{x}_0)\big) C_2^d\, d\mathbf{x},$$

with $h(\mathbf{x}, \mathbf{x}_0) = \prod_{j=1}^d |x^{(j)} - x_0^{(j)}|$. An argument similar to that used in the proof of Lemma 1 leads to

$$n \int_{[0,1]^d} B\big(k-1; n-1, C_2^d h(\mathbf{x}, \mathbf{x}_0)\big) C_1^d\, d\mathbf{x} \ge n \int_{[0,1]^d} B\big(k-1; n-1, C_2^d h(\mathbf{x}, \mathbf{0})\big) C_1^d\, d\mathbf{x}$$

and

$$n \int_{[0,1]^d} B\big(k-1; n-1, C_1^d h(\mathbf{x}, \mathbf{x}_0)\big) C_2^d\, d\mathbf{x} \le 2^d n \int_{[0,1]^d} B\big(k-1; n-1, C_1^d h(\mathbf{x}, \mathbf{0})\big) C_2^d\, d\mathbf{x}.$$

Now

$$\begin{aligned} & n \int_{[0,1]^d} B\big(k-1; n-1, C_2^d h(\mathbf{x}, \mathbf{0})\big) C_1^d\, d\mathbf{x} \\ &= n \int_{[0,C_2]^d} B\big(k-1; n-1, h(\mathbf{v}, \mathbf{0})\big) \left(\frac{C_1}{C_2}\right)^d d\mathbf{v} \\ &\ge n \left(\frac{C_1}{C_2}\right)^d \int_{[0,1]^d} B\big(k-1; n-1, h(\mathbf{v}, \mathbf{0})\big)\, d\mathbf{v} \\ &= \left(\frac{C_1}{C_2}\right)^d A_k(n, d, \mathbf{0}, 1) \end{aligned}$$

and

$$\begin{aligned} & n \int_{[0,1]^d} B\big(k-1; n-1, C_1^d h(\mathbf{x}, \mathbf{0})\big) C_2^d\, d\mathbf{x} \\ &= n \int_{[0,C_1]^d} B\big(k-1; n-1, h(\mathbf{v}, \mathbf{0})\big) \left(\frac{C_2}{C_1}\right)^d d\mathbf{v} \\ &\le n \left(\frac{C_2}{C_1}\right)^d \int_{[0,1]^d} B\big(k-1; n-1, h(\mathbf{v}, \mathbf{0})\big)\, d\mathbf{v} \\ &= \left(\frac{C_2}{C_1}\right)^d A_k(n, d, \mathbf{0}, 1). \end{aligned}$$

Therefore, (A.3) is proved.

A.4 Proof of Theorem 3

We need only prove (2), because (3) follows from (2). Let $K = K(\mathbf{x}_0; \mathbf{x}_i, i = 1, \ldots, n)$ denote the number of k-PNNs of $\mathbf{x}_0 \in [0, 1]^d$ among $\{\mathbf{x}_i, i = 1, \ldots, n\}$. Then $K(\mathbf{x}_0; \mathbf{x}_i, i = 1, \ldots, n)$ is a random number depending on the configuration of $\{\mathbf{x}_i, i = 1, \ldots, n\}$. By Theorem 2, there exists $\Lambda_2 > 0$ such that

$$EK(\mathbf{x}_0; \mathbf{x}_i, i = 1, \ldots, n) \le \Lambda_2 k (\log n)^{d-1}, \quad \text{(A.5)}$$

for any $\mathbf{x}_0 \in [0, 1]^d$ and n. We prove (2) with $\Lambda_3 = \Lambda_2^{-1}\sigma^2$.

Fix any $\mathbf{x}_0 \in [0, 1]^d$. Denote the responses of the k-PNNs of $\mathbf{x}_0$ by $y_{(\ell)}, \ell = 1, \ldots, K$. Conditional on $\{\mathbf{x}_i, i = 1, \ldots, n\}$, for random forests with node size k, only k-PNNs can become voting points; that is, $\hat{g}(\mathbf{x}_0) = \sum_{\ell=1}^K \bar{W}_{(\ell)} y_{(\ell)}$. Because the splitting scheme is nonadaptive, the weights, $\bar{W}_{(\ell)}$'s, are independent of the y_i's, given $\{\mathbf{x}_i, i = 1, \ldots, n\}$; therefore,

$$\begin{aligned} & E\big[\big(\hat{g}(\mathbf{x}_0) - g(\mathbf{x}_0)\big)^2 \big| \{\mathbf{x}_i, i = 1, \ldots, n; \bar{W}_{(p)}, p = 1, \ldots, K\}\big] \\ &\ge \operatorname{var}\big[\hat{g}(\mathbf{x}_0) \big| \{\mathbf{x}_i, i = 1, \ldots, n; \bar{W}_{(p)}, p = 1, \ldots, K\}\big] \\ &= \operatorname{var}\left[\sum_{\ell=1}^K \bar{W}_{(\ell)} y_{(\ell)} \Big| \{\mathbf{x}_i, i = 1, \ldots, n; \bar{W}_{(p)}, p = 1, \ldots, K\}\right] \\ &= \sum_{\ell=1}^K \bar{W}_{(\ell)}^2 \operatorname{var}\big[y_{(\ell)} \big| \{\mathbf{x}_i, i = 1, \ldots, n; \bar{W}_{(p)}, p = 1, \ldots, K\}\big] \\ &= \sum_{\ell=1}^K \bar{W}_{(\ell)}^2 \sigma^2 \\ &\ge \frac{\sigma^2}{K}. \end{aligned}$$

The last inequality follows from the fact that $\sum_{\ell=1}^K \bar{W}_{(\ell)} = 1$ [from (1)] and the Cauchy–Schwarz inequality. We then have

$$MSE[\hat{g}(\mathbf{x}_0)] \ge \sigma^2 E\left(\frac{1}{K}\right) \ge \frac{\sigma^2}{E(K)} \ge \Lambda_3 k^{-1} (\log n)^{-(d-1)}.$$

We used Jensen's inequality in the second inequality and (A.5) in the third inequality.

A.5 Proof of Theorem 4

It is easier to start with a general interval $[a, b]$ instead of the interval $[0, 1]$. Consider the uniform random splitting. Let $R_j(a, b, t_0)$ denote the length of the jth interval in the sequence containing t_0, starting from $[a, b]$. Denote $L_j(a, b, t_0) = E[R_j(a, b, t_0)]$. We need the following results.

Lemma A.2.

$$L_{j+1}(a, b, t_0) = (b-a)^{-1}\left[\int_a^{t_0} L_j(x, b, t_0)\,dx + \int_{t_0}^{b} L_j(a, x, t_0)\,dx\right].$$

Proof. Let the first cut be T_1. If T_1 is in (a, t_0), then, conditioning on the first cut, the expected value of the length of the interval after $(k+1)$ cuts is $L_k(T_1, b, t_0)$. If T_1 is in (t_0, b), then, conditioning on the first cut, the expected value of the length of the interval after $(k+1)$ cuts is $L_k(a, T_1, t_0)$. Because the first cut is uniform in (a, b), the result follows.

Lemma A.3.

$$-(1-t)^2\log(1-t) - t^2\log t \le (\log 4)t(1-t) \le 1.4t(1-t), \qquad \forall\, t \in [0, 1].$$

Proof. Consider $f(t) = (\log 4)t(1-t) + (1-t)^2\log(1-t) + t^2\log t$. This is symmetric around $t = .5$. Thus all that we need to show is that $f(t) \ge 0$, $\forall\, t \in [0, .5]$. It is easy to check that $f(0) = 0$, $f'(0) > 0$, $f(.5) = 0$, $f'(.5) = 0$, and $f'(t)$ has only one root in $[0, .5)$. Thus the result follows.

Lemma A.4.

$$\int_a^t \frac{(b-t)(t-x)}{b-x}\,dx + \int_t^b \frac{(x-t)(t-a)}{x-a}\,dx \ge .6(t-a)(b-t), \qquad \forall\, t \in [a, b].$$

Proof. By straightforward calculation, we get

$$\begin{aligned}
&\int_a^t \frac{(b-t)(t-x)}{b-x}\,dx + \int_t^b \frac{(x-t)(t-a)}{x-a}\,dx \\
&= \int_a^t \frac{(b-t)[(b-x)-(b-t)]}{b-x}\,dx \\
&\quad + \int_t^b \frac{[(x-a)-(t-a)](t-a)}{x-a}\,dx \\
&= 2(t-a)(b-t) + (b-t)^2\log\left[\frac{b-t}{b-a}\right] + (t-a)^2\log\left[\frac{t-a}{b-a}\right] \\
&\ge 2(t-a)(b-t) - 1.4(t-a)(b-t) \\
&= .6(t-a)(b-t).
\end{aligned}$$

The second-to-laststep follows from Lemma A.3 by a scaling argument (i.e., scale the interval $[a, b]$ to the interval $[0, 1]$).

Lemma A.5.

$$L_J(a, b, t_0) \ge \frac{1}{2}L_{J-1}(a, b, t_0) + \frac{.6^J(b-t_0)(t_0-a)}{b-a}, \qquad \forall\, J \ge 1.$$

Proof. We prove this lemma by induction. It is easy to see that when $J = 1$,

$$L_1(a, b, t_0) = \frac{1}{2}L_0(a, b, t_0) + \frac{(b-t_0)(t_0-a)}{b-a}.$$

Assume the validity of the theorem for $J = j$. Now, for $J = j+1$, we have

$$\begin{aligned}
&L_{j+1}(a, b, t_0) \\
&\quad = (b-a)^{-1}\left[\int_a^{t_0} L_j(x, b, t_0)\,dx + \int_{t_0}^{b} L_j(a, x, t_0)\,dx\right] \\
&\quad \ge (b-a)^{-1}\int_a^{t_0}\left[\frac{1}{2}L_{j-1}(x, b, t_0) + \frac{.6^j(b-t_0)(t_0-x)}{b-x}\right]dx \\
&\qquad + (b-a)^{-1}\int_{t_0}^{b}\left[\frac{1}{2}L_{j-1}(a, x, t_0) + \frac{.6^j(x-t_0)(t_0-a)}{x-a}\right]dx \\
&\quad = \frac{1}{2}L_j(a, b, t_0) + (b-a)^{-1}0.6^j\left[\int_a^{t_0}\frac{(b-t_0)(t_0-x)}{b-x}\,dx \right. \\
&\qquad \left. + \int_{t_0}^{b}\frac{(x-t_0)(t_0-a)}{x-a}\,dx\right] \\
&\quad \ge \frac{1}{2}L_j(a, b, t_0) + (b-a)^{-1}.6^{j+1}(t_0-a)(b-t_0).
\end{aligned}$$

The first step uses Lemma A.2, the second step uses the induction assumption, the third step uses Lemma A.2, and the fourth step uses Lemma A.4; therefore, the result holds for $J = j+1$, and the lemma is proved.

We now return to the proof of Theorem 4. By Lemma A.5, we have

$$\frac{L_J(0, 1, t_0)}{2^{-J}} \ge \frac{L_{J-1}(0, 1, t_0)}{2^{-(J-1)}} + 1.2^J t_0(1-t_0).$$

Applying this repeatedly, we get

$$\begin{aligned}
\frac{L_J(0, 1, t_0)}{2^{-J}} &\ge L_0(0, 1, t_0) + \sum_{j=1}^{J} 1.2^j t_0(1-t_0) \\
&= 1 + 6t_0(1-t_0)[1.2^J - 1].
\end{aligned}$$

[Received May 2002. Revised September 2005.]

REFERENCES

Amit, Y., and Geman, D. (1997), "Shape Quantization and Recognition With Randomized Trees," *Neural Computation*, 9, 1545–1588.

Breiman, L. (1996), "Bagging Predictors," *Machine Learning*, 26, 123–140.

——— (1998), "Randomizing Outputs to Increase Prediction Accuracy," Technical Report 518, University of California Berkeley, Dept. of Statistics, available at *http://www.stat.berkeley.edu*.

——— (1999), "Random Forests," technical report, University of California Berkeley, Dept. of Statistics, available at *http://www.stat.berkeley.edu*.

——— (2000), "Some Infinity Theory for Predictor Ensembles," Technical Report 577, University of California Berkeley, Dept. of Statistics, available at *http://www.stat.berkeley.edu*.

Bühlmann, P., and Yu, B. (2002), "Analyzing Bagging," *The Annals of Statistics*, 30, 927–961.

Cutler, A. (1999), "Fast Classification Using Perfect Random Trees," Technical Report 5/99/99, Utah State University, Dept. of Mathematics and Statistics.

Cutler, A., and Zhao, G. (2000), "Voting Perfect Random Trees," Technical Report 5/00/100, Utah State University, Dept. of Mathematics and Statistics.

Dasarathy, B. (1991), *Nearest-Neighbor Pattern Classification Techniques*, Los Alamitos, CA: IEEE Computer Society Press.

Devroye, L., Györfi, L., and Lugosi, G. (1996), *A Probability Theory of Pattern Recognition*, New York: Springer-Verlag.

Dietterich, T. G. (2000), "An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization," *Machine Learning*, 40, 139–157.

Domeniconi, C., Peng, J., and Gunopulos, D. (2002), "Locally Adaptive Metric Nearest-Neighbor Classification," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24, 1281–1285.

Fix, E., and Hodges, J. (1951), "Discriminatory Analysis–Nonparametric Discrimination: Consistency Properties," Technical Report 21-49-004, 4, U.S. Air Force, School of Aviation Medicine.

Freund, Y. (1995), "Boosting a Weak Learning Algorithm by Majority," *Information and Computation*, 121, 256–285.

Freund, Y., and Schapire, R. E. (1996), "Experiments With a New Boosting Algorithm," in *Machine Learning: Proceedings of the Thirteenth International Conference*, San Francisco: Morgan Kauffman, pp. 148–156.

Friedman, J. H. (1991), "Multivariate Adaptive Regression Splines" (with discussion), *The Annals of Statistics*, 19, 1–141.

——— (1994), "Flexible Metric Nearest-Neighbor Classification," preprint, available at *http://www-stat.stanford.edu/~jhf/#Reports*.

——— (2001), "Greedy Function Approximation: A Gradient Boosting Machine," *The Annals of Statistics*, 29, 1189–1232.

Friedman, J. H., and Hall, P. (1999), "On Bagging and Nonlinear Estimation," preprint, available at *http://www-stat.stanford.edu/~jhf/#Reports*.

Friedman, J. H., Hastie, T., and Tibshirani, R. (2000), "Additive Logistic Regression: A Statistical View of Boosting" (with discussion), *The Annals of Statistics*, 28, 337–407.

Hastie, T., and Tibshirani, R. (1996), "Discriminant Adaptive Nearest-Neighbor Classification," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18, 607–616.

Hastie, T., Tibshirani, R., and Friedman, J. (2001), *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, New York: Springer-Verlag.

Ho, T. K. (1998), "The Random Subspace Method for Constructing Decision Forests," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20, 832–844.

Lim, T., Loh, W., and Shih, Y. (2000), "A Comparison of Prediction Accuracy, Complexity, and Training Time of Thirty-Three Old and New Classification Algorithms," *Machine Learning*, 40, 203–228.

Mason, L., Baxter, J., Bartlett, P., and Frean, M. R. (2000), "Boosting Algorithms as Gradient Descent in Function Space," in *Neural Information Processing Systems 1999*, Cambridge, MA: MIT Press, pp. 512–518.

Myles, J., and Hand, D. (1990), "The Multiclass Metric Problem in Nearest-Neighbor Classification," *Pattern Recognition*, 23, 1291–1297.

Ripley, B. D. (1996), *Pattern Recognition and Neural Networks*, Cambridge, U.K.: Cambridge University Press.

Schapire, R. E. (1990), "The Strength of Weak Learnability," *Machine Learning*, 5, 197–227.

Short, R., and Fukunaga, K. (1981), "The Optimal Distance Measure for Nearest-Neighbor Classification," *IEEE Transactions on Information Theory*, 27, 655–665.

Stone, C. J. (1980), "Optimal Rates of Convergence for Nonparametric Estimators," *The Annals of Statistics*, 8, 1348–1360.