

---

# Voting Perfect Random Trees

---

**Adele Cutler**

Department of Mathematics and Statistics  
Utah State University  
Logan, UT 84322-3900  
adele@math.usu.edu

**Guohua Zhao**

Department of Mathematics and Statistics  
Utah State University  
Logan, UT 84322-3900  
slzhao@math.usu.edu

## Abstract

The success of Adaboost and other ensemble classifiers has provoked considerable interest in understanding why such methods succeed and identifying circumstances in which they can be expected to produce good results.

We consider a simple “perfect random tree” classifier (PERT). The PERT classifier is specifically designed for ensemble methods. It produces results that are competitive with the best known off-the-shelf classifiers, and is easy to code and very fast to fit. Calculations suggest that one reason why it works so well is that although the individual tree classifiers are relatively weak, they are also almost uncorrelated.

The simple probabilistic nature of the classifier lends itself to theoretical analysis. We show that PERT is actually fitting a continuous posterior probability surface for each class. As such, it can be viewed as a classification-via-regression procedure that fits a continuous interpolating surface. This surface could, in theory, be found using a one-shot procedure, without an ensemble.

## 1 Introduction

It has recently been shown (see, for example, [1]) that ensemble classifiers can give more accurate predictions than using an individual classifier. In bagging [2], an ensemble classifier is constructed by generating bootstrap samples from the original data set, constructing a classifier from each bootstrap sample, and voting to combine. In Adaboost or other arcing algorithms [1, 3], the successive classifiers are

constructed by giving increased weight to those points which have been frequently misclassified up to this time, and the classifiers are combined using weighted voting.

Both bagging and arcing perturb or reweight the training data to generate the individual classifiers for the ensemble. Another approach is to fix the training data and allow the base learner itself to have a random element. One such method is the randomized C4.5 tree [4], which splits a node by finding the 20 best candidate splits and choosing one at random. In situations with little or no classification noise the randomized C4.5 tree was shown to be competitive with (and perhaps slightly superior to) bagging C4.5.

Breiman [5] gives a general framework for tree ensembles in which each tree is determined by a random vector which is independently sampled from the same distribution (“random forests”). He shows that under reasonable conditions, the generalization error for a random forest converges a.s. as the number of trees increases. The accuracy depends on the strength of the individual classifiers and the dependence between them. Accordingly, Breiman [5] studies trees that are constructed by selecting the best split for a randomly chosen feature or linear combination of features. Random selection of features helps to reduce correlation, and optimizing over the split selection maintains the strength of the individual classifiers.

Cutler [6] introduces a technique (PERT) that randomly chooses both the feature on which to split and the split itself. In particular, PERT does not optimize over either the feature or the location of the split. It is very easy to code and very fast to fit. Understandably, the method does not perform well if it is applied alone, but in ensembles it is competitive with Adaboost and random feature selection. Moreover, PERT lends itself to theoretical analysis.

The PERT method is described in Section 2. Section 3 contains comparisons with published results of bagging and arcing CART and random forests. Section 4 gives an explanation of PERT’s performance in terms of strength and correlation. Section 5 contains our theoretical analysis, and Section 6 gives some concluding remarks.

## 2 Perfect Random Trees

The perfect random tree (PERT [6]) fits the training data perfectly and is constructed as follows. To split a node, first note that if all the data points in the node have the same class, the node is terminal and need not be split. Otherwise, randomly choose two data points from the node until we get two belonging to different classes, say  $\mathbf{x} = (x_1, \dots, x_p)$  and  $\mathbf{y} = (y_1, \dots, y_p)$ . Now, randomly choose a feature on which to split, say feature  $j$ , and split at  $\alpha x_j + (1 - \alpha)y_j$ , where  $\alpha$  is generated from a uniform(0,1) distribution.

Ties are taken care of by repeating the procedure until a definitive split is obtained. If no such split is found after 10 tries, the node is declared to be terminal (so in this case, the tree would not perfectly fit the data).

PERT is trivial to code and PERT classifiers are extremely fast to construct because there is no optimality criterion to compute. CART, for example, searches over all possible features and all possible splits to find the best way to split each node. Even random feature selection [5] selects an optimal split for the given choice of feature(s). However, the speed of fitting PERT is somewhat offset by the fact that

Table 1: Test Set Errors (%).

Data Set	Adaboost	Forest-RI		PERT
	CART	Multiple	Single	
glass	22.0	20.6	21.2	21.4
breast c.	3.2	2.9	2.7	2.8
diabetes	26.6	24.2	24.3	24.5
sonar	15.6	15.9	18.0	16.2
vowel	4.1	3.4	3.3	2.3
ionosphere	6.4	7.1	7.5	6.8
vehicle	23.2	25.8	26.4	27.4
soybean	5.8	6.0	6.5	6.5
Ger. credit	23.5	24.4	26.2	24.9
image	1.6	2.1	2.7	2.6
ecoli	14.8	12.8	13.0	14.5
votes	4.8	4.1	4.6	4.4
liver	30.7	25.1	24.7	27.9
letters	3.4	3.5	4.7	4.4
sat-images	8.8	8.6	10.5	10.2
zipcode	6.2	6.3	7.8	7.6
waveform	17.8	17.2	17.3	17.8
twonorm	4.9	3.9	3.9	3.6
threennorm	18.8	17.5	17.5	17.6
ringnorm	6.9	4.9	4.9	12.8

it produces very large trees (because it fits the training data perfectly), so it tends to be slow to evaluate. Some timing comparisons for PERT are given in [6]. Our focus here is not on computational efficiency but on understanding why a tree-based classifier that makes no attempt at finding optimal splits can produce competitive results. First, we show that PERT does indeed produce competitive results.

### 3 Experimental Results

We compared the accuracy of PERT to published results [5] for Adaboost with CART as the base learner and random feature selection (Forest-RI).

For comparability, we followed the same procedure as [5] to obtain the results for PERT. For the real datasets, 10% of the data were randomly chosen and set aside as a test set. The ensemble method was fit to the remaining data and then used to classify the test set. The entire procedure was repeated 500 times and the test set errors were averaged. For the synthetic data, a new training set of size 300 and test set of size 3000 were generated for each of the 500 repetitions. Most of the datasets are available from the UCI repository [7]. The synthetic datasets are described in [2].

Table 2: Test Set Errors (%) for Larger PERT Ensembles.

Data Set	100	200	400	1600
waveform	17.8	17.3	17.1	16.8
twonorm	3.6	3.2	3.1	3.0
threenorm	17.6	16.4	15.8	15.3
ringnorm	12.8	11.8	11.1	10.7

For the Adaboost results (from [5]), 50 trees were used, and 100 trees were used for all the other ensembles, with the exception of the zipcode data, for which these numbers were doubled.

The results (Table 1) are surprisingly good for a method as simple and fast as PERT. For most of the examples, PERT falls within the range of the other three methods. The PERT results for vowel and twonorm are somewhat better than those for the other three methods, while PERT’s results for vehicle are somewhat worse. The only striking difference is for ringnorm, where PERT does not do well. It turns out that we can get slightly better results if we combine more classifiers (Table 2), but the results for ringnorm are still not good.

We believe that PERT is suffering from the curse of dimensionality. Ringnorm is a mixture of two multivariate normal distributions. The covariance matrix for class 1 is four times the identity, while that for class 2 is the identity. The Bayes rule classifies a region around the class 2 mean as class 2, with the entire surrounding (“tail”) region as class 1. PERT has such a high misclassification rate because it tends to make mistakes in this tail region.

## 4 Strength and Correlation

For obvious reasons, a single PERT tree is not a good classifier. Compared to, say, CART, it works very poorly. However, in ensembles it appears to be competitive with some of the best methods available. In order to help understand why, consider Breiman’s bound on the generalization error:

$$PE^* \leq \bar{\rho}(1 - s^2)/s^2,$$

where  $s$  is a measure of strength of the individual classifiers in the forest and  $\bar{\rho}$  is a measure of the correlation between them in terms of the raw margin (for details, see [5]). While the bound is loose, it suggests that a reasonable strategy for random forests is to find trees that give the correct class on average (high  $s$ ) but make mistakes in different places (low  $\bar{\rho}$ ).

We calculated  $s$  and  $\bar{\rho}$  for PERT and bagged CART for the four synthetic datasets and obtained the results given in Table 3. These results suggest that part of the reason why PERT performs well is that although the individual PERT classifiers are relatively weak (low values of  $s$ ), they have very low correlation (low values of  $\bar{\rho}$ ). Presumably, if we could strengthen the PERT classifier without substantially increasing the correlation, an even better ensemble classifier could be developed.

Table 3: Strength and Correlation for bagged CART and PERT.

Data Set	Strength, $s$		Correlation, $\bar{\rho}$	
	CART	PERT	CART	PERT
waveform	.37	.20	.22	.06
twonorm	.54	.50	.14	.07
threernorm	.31	.20	.13	.04
ringnorm	.48	.27	.14	.05

## 5 Posterior Probabilities

Following [5], consider an ensemble of random tree classifiers (a “random forest”),  $h(X, \theta_1), \dots, h(X, \theta_N)$ , where  $\theta_1, \dots, \theta_N$  are independent and identically distributed. For PERT,  $\theta$  denotes the vector of random numbers used to select random features and random splits.

The posterior probability that a random tree predicts class  $j$  at  $X$ , given the training data, is

$$Q_j(X) = P_\theta (h(X, \theta) = j).$$

In practice, we estimate  $Q_j$  using

$$\hat{Q}_j(X) = \frac{1}{N} \sum_{k=1}^N I(h(X, \theta_k) = j),$$

where  $I$  denotes the indicator function. The ensemble predicts the class at  $X$  by

$$\hat{h}(X) = \arg \max_j \hat{Q}_j(X).$$

The definition of  $Q_j$  can be extended to non-random classifiers by defining  $Q_j$  to be 1 if the classifier predicts class  $j$  and 0 otherwise. For example, a single-nearest-neighbor classifier results in  $Q_j$  being 1 in a voronoi region around each class  $j$  example from the training data, and zero elsewhere.

Most ensemble classifiers are so complicated that very little can be said about  $Q_j$ . However, PERT has a very simple structure which allows some theoretical development. It is clear that because PERT correctly classifies the training data  $(x_i, y_i), i = 1, \dots, n$ , we know that

$$Q_j(x_i) = \begin{cases} 1 & \text{if } y_i = j \\ 0 & \text{otherwise} \end{cases}$$

Furthermore, we can show [10] that for PERT,  $Q_j$  is continuous over the feature space. In fact, if we let  $x_{(1),k}, \dots, x_{(n),k}$  denote the sorted values of feature  $k$  in the training data, then in any hyper-rectangle

$$I(i_1, \dots, i_m) = [x_{(i_1),1}, x_{(i_1+1),1}] \times \dots \times [x_{(i_p),p}, x_{(i_p+1),p}]$$

$Q_j(X)$  is a multilinear function, i.e. a linear function of each of its arguments, holding the other arguments fixed. These multilinear functions are continuous on the boundaries of the hyper-rectangles  $I(i_1, \dots, i_m)$ .

In fact, given training data,  $Q_j$  could (in theory) be computed directly using a recursive equation given in [10]. In practice, it is faster to simply estimate  $Q_j$  using the ensemble, but PERT appears to be the first competitive ensemble classifier that could be fit in a “one shot” manner.

To better understand PERT, we consider a generic method for classification using linear or nonlinear regression. The linear version dates back to Fisher [8] and the nonlinear version to Breiman and Ihaka [9]. For a 2-class situation, let

$$y_i = \begin{cases} 0 & \text{class 1} \\ 1 & \text{class 2.} \end{cases}$$

Now consider fitting a linear or nonlinear regression to the training data, and classifying a new observation  $x$  as class 2 if  $\hat{y} \geq 0.5$ , where  $\hat{y}$  is the regression estimate of  $y$  at  $x$ .

One way to think about PERT is that instead of using linear or nonlinear regression, PERT is fitting a blockwise multilinear interpolating surface. For regression purposes, such a surface would drastically overfit the data, but for classification purposes it appears to work very well. As Friedman [11] points out, the best methods for regression do not usually translate to the best methods for classification, and PERT is an example of this phenomenon. The natural question is whether there are other ways of interpolating that give even faster, more accurate, one-shot classifiers.

## 6 Concluding Remarks

PERT is a fast, easily coded classifier that gives results comparable to some of the best classifiers available. As well as being useful in its own right, it also helps in understanding a little more about ensemble classifier behavior. An interesting avenue for further research is to revisit the classification-through-regression paradigm to see whether better one-shot classifiers can be found.

### Acknowledgements

Many thanks to Leo Breiman for constant encouragement and frequent helpful discussions.

### References

- [1] Breiman, L. (1998). Arcing Classifiers. *The Annals of Statistics* **26**, 801–849.
- [2] Breiman, L. (1996). Bagging Predictors. *Machine Learning* **26**, 123–140.
- [3] Freund, Y. and Schapire, R. (1996). Experiments with a New Boosting Algorithm. In *Machine Learning: Proceedings of the Thirteenth International Conference* (L. Saitta, ed.) 148–156. Morgan Kaufmann, San Francisco.
- [4] Dietterich, T. (1998). An Experimental Comparison of Three methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization. *Machine Learning* 1–22.
- [5] Breiman, L. (1999). Random Forests - Random Features. *Technical Report* **567**,

Statistics Department, U.C.Berkeley, September 1999.  
[ftp://ftp.stat.berkeley.edu/pub/users/breiman].

[6] Cutler, A. (1999). Fast Classification Using Perfect Random Trees. *Technical Report 5/99/99*, Department of Mathematics and Statistics, Utah State University, May 1999.

[7] Murphy, P. M., & Aha, D. W. (1994). *UCI Repository of machine learning databases* [<http://www.ics.uci.edu/~mllearn/MLRepository.html>]. Irvine, CA: University of California, Department of Information and Computer Science.

[8] Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of Eugenics* 7, 179–188.

[9] Breiman, L. & Ihaka, R. (1984). Nonlinear Discriminant Analysis via ACE and Scaling. *Technical Report 40*, Statistics Department, U.C.Berkeley.

[10] Zhao, G. (2000). *A New Perspective On Classification*, PhD Dissertation, Utah State University.

[11] Friedman, J. H. (1997). On Bias, Variance, 0/1 Loss, and the Curse of Dimensionality. *J. Data Mining and Knowledge Discovery* 1, 55.