The Annals of Statistics
1998, Vol. 26, No. 3, 801–849

ARCING CLASSIFIERS[1]

BY LEO BREIMAN

University of California, Berkeley

Recent work has shown that combining multiple versions of unstable classifiers such as trees or neural nets results in reduced test set error. One of the more effective is bagging. Here, modified training sets are formed by resampling from the original training set, classifiers constructed using these training sets and then combined by voting. Freund and Schapire propose an algorithm the basis of which is to adaptively resample and combine (hence the acronym "arcing") so that the weights in the resampling are increased for those cases most often misclassified and the combining is done by weighted voting. Arcing is more successful than bagging in test set error reduction. We explore two arcing algorithms, compare them to each other and to bagging, and try to understand how arcing works. We introduce the definitions of bias and variance for a classifier as components of the test set error. Unstable classifiers can have low bias on a large range of data sets. Their problem is high variance. Combining multiple versions either through bagging or arcing reduces variance significantly.

1. Introduction. Some classification and regression methods are unstable in the sense that small perturbations in their training sets or in construction may result in large changes in the constructed predictor. Subset selection methods in regression, decision trees in regression and classification and neural nets are unstable [Breiman (1996b)].

Unstable methods can have their accuracy improved by perturbing and combining, that is, generate multiple versions of the predictor by perturbing the training set or construction method, then combine these multiple versions into a single predictor. For instance, Ali (1995) generates multiple classification trees by choosing randomly from among the best splits at a node and combines trees using maximum likelihood. Breiman (1996b) adds noise to the response variable in regression to generate multiple subset regressions and then averages these. We use the label P & C (perturb and combine) to designate this group of methods.

One of the more effective P & C methods is bagging [Breiman (1996a)]. Bagging perturbs the training set repeatedly to generate multiple predictors and combines these by simple voting (classification) or averaging (regression). Let the training set T consist of N cases (instances) labeled by $n = 1, 2, \ldots, N$. Put equal probabilities $p(n) = 1/N$ on each case and using these probabili-

Received May 1996; revised May 1997.
[1]Supported in part by NSF Grant 1-444063-21445.
AMS 1991 *subject classification*. Primary 62H30.
Key words and phrases. Ensemble methods, decision trees, neural networks, bagging, boosting, error-correcting, output coding, Markov chain, Monte Carlo.

ties, sample *with replacement* (bootstrap) N times from the training set T, forming the resampled training set $T^{(B)}$. Some cases in T may not appear in $T^{(B)}$; some may appear more than once. Now use $T^{(B)}$ to construct the predictor, repeat the procedure and combine. Bagging applied to CART gave dramatic decreases in test set errors.

Freund and Schapire (1996, 1997) recently proposed a P & C algorithm which was designed to drive the training set error rapidly to zero. However, if their algorithm is run far past the point at which the training set error is zero, it gives better performance than bagging on a number of real data sets. The crux of their idea is this: start with $p(n) = 1/N$ and resample from T to form the first training set $T^{(1)}$. As the sequence of classifiers and training sets is being built, increase $p(n)$ for those cases that have been most frequently misclassified. At termination, combine classifiers by weighted or simple voting. We will refer to algorithms of this type as "adaptive resampling and combining," or "arcing" algorithms. In honor of Freund and Schapire's discovery, we denote their specific algorithm by arc-fs and discuss their theoretical efforts to relate training set to test set error in the Appendix.

To better understand stability and instability and what bagging and arcing do, we define the concepts of bias and variance for classifiers in Section 2. The difference between the test set misclassification error for the classifier and the minimum error achievable is the sum of the bias and variance. Unstable classifiers such as trees characteristically have high variance and low bias. Stable classifiers like linear discriminant analysis have low variance, but can have high bias. This is illustrated on several examples of artificial data. Section 3 looks at the effects of arcing and bagging trees on bias and variance.

The main effect of both bagging and arcing is to reduce variance. Usually, arcing seems to do better at this than bagging. Arc-fs does complex things and its behavior is puzzling. However, the variance reduction comes from the adaptive resampling and not the specific form of arc-fs. To show this, we define a simpler arc algorithm denoted by arc-x4 whose accuracy is comparable to arc-fs. The two appear to be at opposite poles of the arc spectrum. Arc-x4 was concocted to demonstrate that arcing works not because of the specific form of the arc-fs algorithm, but because of the adaptive resampling.

Freund and Schapire (1996) compare arc-fs to bagging on 27 data sets and conclude that arc-fs has a small edge in test set error rates. We tested arc-fs, arc-x4 and bagging on the 10 real data sets used in our bagging paper and got results more favorable to arcing. These are given in Section 4. The results show that arc-fs and arc-x4 finish in a dead heat. On a few data sets, one or the other is a little better, but both are almost always significantly better than bagging. We also look at arcing and bagging applied to the U.S. Postal Service digit data base.

The overall results of arcing are exciting—it turns a good but not great classifier (CART) into a procedure that seems usually to get close to the lowest achievable test set error rates. Furthermore, the arc-classifier is off-the-shelf. Its performance does not depend on any tuning or settings for

particular problems. Just read in the data and press the start button. It is also, by neural net standards, blazingly fast to construct.

Section 5 gives the results of some experiments aimed at understanding how arc-fs and arc-x4 work. Each algorithm has distinctive and different signatures. Generally, arc-fs uses a smaller number of distinct cases in the resampled training sets and the successive values of $p(n)$ are highly variable. The successive training sets in arc-fs rock back and forth and there is no convergence to a final set of $\{p(n)\}$. The back and forth rocking is more subdued in arc-x4, but there is still no convergence to a final $\{p(n)\}$. This variability may be an essential ingredient of successful arcing algorithms.

Instability is an essential ingredient for bagging or arcing to improve accuracy. Nearest neighbors are stable; Breiman (1996a) noted that bagging does not improve nearest neighbor classification. Linear discriminant analysis is also relatively stable (low variance) and in Section 6 our experiments show that neither bagging nor arcing has any effect on linear discriminant error rates.

Sections 7 and 8 contain remarks, mainly to give understanding of how bagging and arcing work. The reason that bagging reduces error is fairly transparent, but it is not at all clear yet, in other than general terms, how arcing works. Two dissimilar arcing algorithms, arc-fs and arc-x4, give comparable accuracy. It is possible that other arcing algorithms, intermediate between arc-fs and arc-x4, will give even better performance. The experiments here, in Freund and Shapire (1995), in Drucker and Gortes (1997) and in Quinlan (1996) indicate that arcing decision trees may lead to fast and generally accurate classification methods and indicate that additional research aimed at understanding the workings of this class of algorithms will have a high pay-off.

2. The bias and variance of a classifier. In classification, the output variable $y \in \{1, \ldots, J\}$ is a class label. The training set T is of the form $T = \{(y_n, \mathbf{x}_n)\ n = 1, \ldots, N\}$ where the y_n are class labels. Given T, some method is used to construct a classifier $C(\mathbf{x}, T)$ for predicting future y-values. Assume that the data in the training set consists of iid selections from the distribution of $Y, \mathbf{X}$. The misclassification error is defined as

$$PE(C(\ , T)) = P_{\mathbf{X},Y}(C(\mathbf{X}, T) \neq Y)$$

and we denote by $PE(C)$ the expectation of $PE(C(\ , T))$ over T. Denote

$$P(j \mid \mathbf{x}) = P(Y = j \mid \mathbf{X} = \mathbf{x}),$$
$$P(\mathbf{dx}) = P(\mathbf{X} \in \mathbf{dx}).$$

The minimum misclassification rate is given by the "Bayes classifier C^*" where

$$C^*(\mathbf{x}) = \operatorname*{argmax}_j P(j|\mathbf{x})$$

with misclassification rate

$$PE(C^*) = 1 - \int \max_j (P(j|\mathbf{x}))P(\mathbf{dx}).$$

Let

$$Q(j|\mathbf{x}) = P_T(C(\mathbf{x}, T) = j)$$

and define the aggregated classifier as

$$C_A(\mathbf{x}) = \operatorname{argmax}_j Q(j|\mathbf{x}).$$

This is aggregation by voting. Consider many independent replicas $T_1, T_2, \ldots$; construct the classifiers $C(\mathbf{x}, T_1), C(\mathbf{x}, T_2), \ldots$; and at each $\mathbf{x}$ determine the classification $C_A(\mathbf{x})$ by having these multiple classifiers vote for the most popular class.

DEFINITION 2.1. $C(\mathbf{x}, \cdot)$ is unbiased at $\mathbf{x}$ if $C_A(\mathbf{x}) = C^*(\mathbf{x})$.

That is, $C(\mathbf{x}, T)$ is unbiased at $\mathbf{x}$ if, over the replications of T, $C(\mathbf{x}, T)$ picks the right class more often than any other class. A classifier that is unbiased at $\mathbf{x}$ is not necessarily an accurate classifier. For instance, suppose that in a two-class problem $P(1|\mathbf{x}) = 0.9$, $P(2|\mathbf{x}) = 0.1$ and $Q(1|\mathbf{x}) = 0.6$, $Q(2|\mathbf{x}) = 0.4$. Then C is unbiased at $\mathbf{x}$ but the probability of correct classification by C is $0.6 \times 0.9 + 0.4 \times 0.1 = 0.58$. However, the Bayes predictor C^* has probability 0.9 of correct classification.

If C is unbiased at $\mathbf{x}$ then $C_A(\mathbf{x})$ is optimal. Let U be the set of all $\mathbf{x}$ at which C is unbiased, and call U the unbiased set. The complement of U is called the bias set and denoted by B.

DEFINITION 2.2. The bias of a classifier C is

$$\text{Bias}(C) = P_{\mathbf{X},Y}(C^*(\mathbf{X}) = Y, \mathbf{X} \in B) - E_T P_{\mathbf{X},Y}(C(\mathbf{X}, T) = Y, \mathbf{X} \in B)$$

and its variance is

$$\text{Var}(C) = P_{\mathbf{X},Y}(C^*(\mathbf{X}) = Y, \mathbf{X} \in U) - E_T P_{\mathbf{X},Y}(C(\mathbf{X}, T) = Y, \mathbf{X} \in U).$$

This leads to the *fundamental decomposition*

$$PE(C) = PE(C^*) + \text{Bias}(C) + \text{Var}(C).$$

Note that aggregating a classifier and replacing C with C_A reduces the variance to zero, but there is no guarantee that it will reduce the bias. In fact, it is easy to give examples where the bias will be increased. Thus, if the bias set B has large probability, $PE(C_A)$ may be significantly larger than $PE(C)$. As defined, bias and variance have the following properties:

1. Bias and variance are always nonnegative.
2. The variance of C_A is zero.
3. If C is deterministic, that is, does not depend on T, then its variance is zero.
4. The bias of C^* is zero.

The proofs of (1)–(4) are immediate from the definitions. The variance of C can be expressed as

$$\mathrm{Var}(C) = \int_U \left[\max_j P(j|\mathbf{x}) - \sum_j Q(j|\mathbf{x}) P(j|\mathbf{x}) \right] P(d\mathbf{x}).$$

The bias of C is a similar integral over B. Clearly, both bias and variance are nonnegative. Since $C_A = C^*$ on U, its variance is zero. If C is deterministic, then on U, $C = C^*$, so C has zero variance. Finally, it is clear that C^* has zero bias.

Use of the terms "bias" and "variance" in classification is somewhat misleading, since they do not have properties commonly associated with bias and variance in predicting numerical outcomes, that is, regression [see Geman, Bienenstock and Doursat (1992)]. The most important aspect of our definition is that the variance is the component of the classification error that can be eliminated by aggregation. However, the bias may be larger for the aggregated classifier than for the unaggregated.

Friedman (1996) gives a thoughtful analysis of the meaning of bias and variance in two class problems. Using some simplifying assumptions, a definition of "boundary bias" at a point $\mathbf{x}$ is given and it is shown that at points of negative boundary bias, classification error can be reduced by reducing variance in the class probability estimates. If the boundary bias is not negative, decreasing the estimate variance may increase the classification error. The points of negative boundary bias are exactly the points defined above as the unbiased set. Other definitions of bias and variance in classification are given in Kong and Dietterich (1995), Kohavi and Wolpert (1996) and Tibshirani (1996).

2.1. *Instability, bias and variance.* Breiman (1996a) pointed out that some prediction methods were unstable in that small changes in the training set could cause large changes in the resulting predictors. I listed trees and neural nets as unstable, nearest neighbors as stable. Linear discriminant analysis (LDA) is also stable. Unstable classifiers are characterized by high variance. As T changes, the classifiers $C(\mathbf{x}, T)$ can differ markedly from each other and from the aggregated classifier $C_A(\mathbf{x})$. Stable classifiers do not change much over replicates of T, so $C(\mathbf{x}, T)$ and $C_A(\mathbf{x})$ will tend to be the same and the variance will be small.

Procedures like CART have high variance, but they are "on average, right," that is, they are largely unbiased—the optimal class is usually the winner of the popularity vote. Stable methods, like LDA achieve their stability by having a very limited set of models to fit to the data. The result is low variance. But if the data cannot be adequately represented in the available set of models, large bias can result.

2.2. *Examples.* To illustrate, we compute bias and variance of CART for a few examples. These all consist of artificially generated data, since otherwise

C^* cannot be computed nor T replicated. In each example, the classes have equal probability and the training sets have 300 cases.

Waveform: This is 21 dimension, 3 class data. It is described in the CART book [Breiman, (1984)] and code for generating the data is in the UCI repository (ftp address: `ftp.ics.uci.edu` directory `pub/machine-learning-databases`).

Twonorm: This is 20 dimension, 2 class data. Each class is drawn from a multivariate normal distribution with unit covariance matrix. Class 1 has mean $(a, a, \ldots, a)$ and class 2 has mean $(-a, -a, \ldots, -a)$; $a = 2/(20)^{1/2}$.

Threenorm: This is 20 dimension, 2 class data. Class 1 is drawn with equal probability from a unit multivariate normal with mean $(a, a, \ldots, a)$ and from a unit multivariate normal with mean $(-a, -a, \ldots, -a)$. Class 2 is drawn from a unit multivariate normal with mean at $(a, -a, a, -a, \ldots a)$; $a = 2/(20)^{1/2}$.

Ringnorm: This is 20 dimension, 2 class data. Class 1 is multivariate normal with mean zero and covariance matrix four times the identity. Class 2 has unit covariance matrix and mean $(a, a, \ldots, a)$; $a = 1/(20)^{1/2}$.

Monte Carlo techniques were used to compute bias and variance for each of the above distributions. One hundred training sets of size 300 were generated and a CART tree grown on each one. An additional set of size 18,000 was generated from the same distribution and the aggregated classifier computed for each point in this latter set. The Bayes predictor can be analytically computed, so each point in the 18,000 can be assigned to either the bias set or its complement. The results of averaging over this set are in Table 1.

These problems are difficult for CART. For instance, in twonorm the optimal separating surface is an oblique plane. This is hard to approximate by the multidimensional rectangles used in CART. In ringnorm, the separating surface is a sphere, again difficult for a rectangular approximation. Threenorm is the most difficult, with the separating surface formed by the continuous join of two oblique hyperplanes. Yet in all examples CART has low bias. The problem is its variance.

We will explore, in the following sections, methods for reducing variance by combining CART classifiers trained on perturbed versions of the training set. In all of the trees that are grown, only the default options in CART are used. No special parameters are set, nor is anything done to optimize the performance of CART on these data sets.

TABLE 1
Bias, variance and error of CART (%)

Data set	$PE(C^*)$	Bias	Variance	Error
Waveform	13.2	1.7	14.1	29.0
Twonorm	2.3	0.1	19.6	22.1
Threenorm	10.5	1.4	20.9	32.8
Ringnorm	1.3	1.5	18.5	21.4

3. Bias and variance for arcing and bagging. Given the ubiquitous low bias of tree classifiers, if their variances can be reduced, accurate classifiers may result. The general direction toward reducing variance is indicated by the classifier $C_A(\mathbf{x})$. This classifier has zero variance and low bias. Specifically, on the four problems above, its bias is 2.9, 0.4, 2.6, 3.4. Thus, it is nearly optimal. Recall that it is based on generating independent replicates of T, constructing a CART classifier on each replicate training sets and then letting these classifiers vote for the most popular class. It is not possible, given real data, to generate independent replicates of the training set. But imitations are possible and do work.

3.1. *Bagging*. The simplest implementation of the idea of generating quasi-replicate training sets is bagging [Breiman (1996a)]. Define the probability of the nth case in the training set to be $p(n) = 1/N$. Now sample N times from the distribution $\{p(n)\}$. Equivalently, sample from T *with replacement*. This forms a resampled training set T'. Cases in T may not appear in T' or may appear more than once. Here T' is more familiarly called a bootstrap sample from T.

Denote the distribution on T given by $\{p(n)\}$ as $P^{(B)}$. Then T' is iid from $P^{(B)}$. Repeat this sampling procedure, getting a sequence of independent bootstrap training sets. Construct a corresponding sequence of classifiers by using the same classification algorithm applied to each one of the bootstrap training sets. Then let these multiple classifiers vote for each class. For any point $\mathbf{x}$, $C_A(\mathbf{x})$ really depends on the underlying probability P that the training sets are drawn from $C_A(\mathbf{x}) = C_A(\mathbf{x}, P)$. The bagged classifier is $C_A(\mathbf{x}, P^{(B)})$. The hope is that this is an approximation to $C_A(\mathbf{x}, P)$ good enough that considerable variance reduction will result.

3.2. *Arcing*. Arcing is a more complex procedure. Again, multiple classifiers are constructed and vote for classes, but the construction is sequential, with the construction of the $(k + 1)$st classifier depending on the performance of the k previously constructed classifiers. We give a brief description of the Freund–Schapire arc-fs algorithm. Details are contained in Section 4.

At the start of each construction, there is a probability distribution $\{p(n)\}$ on the cases in the training set. A training set T' is constructed by sampling N times from this distribution (or by reweighting, see Section 8.2). Then the probabilities are updated depending on how the cases in T are classified by $C(\mathbf{x}, T')$. A factor $\beta > 1$ is defined which depends on the misclassification rate —the smaller it is, the larger β is. If the nth case in T is misclassified by $C(\mathbf{x}, T')$, then put weight $\beta p(n)$ on that case. Otherwise define the weight to be $p(n)$. Now divide each weight by the sum of the weights to get the updated probabilities for the next round of sampling. After a fixed number of classifiers have been constructed, they do a weighted voting for the class.

The intuitive idea of arcing is that the points most likely to be selected for the replicate data sets are those most likely to be misclassified. Since these

are the troublesome points, focusing on them using the adaptive resampling scheme of arc-fs may do better than the neutral bagging approach.

3.3. *Results*. Bagging and arc-fs were run on the artificial data set described above. For each one, the procedure consisted of generating 100 replicate training sets of size 300. On each training set bagging and arc-fs were run 50 times using CART as the classifier. For each training set this gave an arced classifier and a bagged classifier. An additional 18,000-member Monte Carlo set was generated and the results of aggregating the 100 bagged and arced classifiers computed at each member of this latter set and compared with the Bayes classification. This enabled the bias and variance to be computed. The results are given in Table 2 and compared with the CART results.

Although both bagging and arcing reduce bias a bit, their major contribution to accuracy is in the large reduction of variance. Arcing does better than bagging because it does better at variance reduction.

3.4. *The effect of combining more classifiers*. The experiments with bagging and arcing above used combinations of 50 tree classifiers. A natural question is "What happens if more classifiers are combined?" To explore this, we ran arc-fs and bagging on the waveform and twonorm data using combinations of 50, 100, 250 and 500 trees. Each run consisted of 100 repetitions. In each run, a training set of 300 and a test set of 1500 were generated, the prescribed number of trees constructed and combined and the test set error computed. These errors were averaged over 100 repetitions to give the results shown in Table 3. Standard errors were less than 0.16%.

Arc-fs error rates decrease significantly out of 250 combinations, reaching rates close to the Bayes minimum (13.2% for waveform and 2.3% for twonorm). One standard of comparison is linear discriminant analysis, which should be

TABLE 2
Bias and variance (%)

Data set	CART	Bagging	Arcing
Waveform			
bias	1.7	1.4	1.0
var	14.1	5.3	3.6
Twonorm			
bias	0.1	0.1	1.2
var	19.6	5.0	1.3
Threenorm			
bias	1.4	1.3	1.4
var	20.9	8.6	6.9
Ringnorm			
bias	1.5	1.4	1.1
var	18.5	8.3	4.5

TABLE 3
Test set error (%) for 50, 100, 250, 500 combinations

Data set	50	100	250	500
Waveform				
arc-fs	17.8	17.3	16.6	16.8
bagging	19.8	19.5	19.2	19.2
Twonorm				
arc-fs	4.9	4.1	3.8	3.7
bagging	7.3	6.8	6.5	6.5

almost optimal for twonorm. It has an error rate of 2.8%, averaged over 100 repetitions. Bagging error rates also decrease out to 250 combinations, but the decreases are smaller than for arc-fs.

4. Arcing algorithms. This section specifies the two arc algorithms and looks at their performance over a number of data sets.

4.1. *Definitions of the arc algorithms.* Both algorithms proceed in sequential steps with a user defined limit of how many steps until termination. Initialize probabilities $\{p(n)\}$ to be equal. At each step, the new training set is selected by sampling from the original training set using probabilities $\{p(n)\}$. After the classifier based on this resampled training set is constructed, the $\{p(n)\}$ are updated depending on the misclassifications up to the present step. On termination, the classifiers are combined using weighted (arc-fs) or unweighted (arc-x4) voting. The arc-fs algorithm is based on a boosting theorem given in Freund and Schapire (1997). Arc-x4 is an ad hoc invention.

Arc-fs specifics.

1. At the kth step, using the current probabilities $\{p^{(k)}(n)\}$, sample with replacement from T to get the training set $T^{(k)}$ and construct classifier C_k using $T^{(k)}$.
2. Run T down the classifier C_k and let $d(n) = 1$ if the nth case is classified incorrectly; otherwise zero.
3. Define

$$\varepsilon_k = \sum_n p^{(k)}(n)d(n), \qquad \beta_k = (1 - \varepsilon_k)/\varepsilon_k$$

and the updated $(k + 1)$st step probabilities by

$$p^{(k+1)}(n) = p^{(k)}(n)\beta_k^{d(n)}/\sum p^{(k)}(n)\beta_k^{d(n)}.$$

After K steps, the $C_1, \ldots, C_K$ are combined using weighted voting with C_K having weight $\log(\beta_k)$. We made two minor revisions to this algorithm. If ε_k becomes greater than $1/2$, then the voting weights β_k go negative. We found that good results were then gotten by setting all $\{p(n)\}$ equal and restarting. If ε_k equals zero, making the subsequent step undefined, we again set the

probabilities equal and restart. The definition given above of arc-fs differs in appearance from the original definition given by Freund and Schapire, but is mathematically equivalent.

A referee pointed out that the updating definition in arc-fs leads to the interesting result that $\sum_n p^{(k+1)}(n)d(n) = 1/2$. That is, the probability weight at the $(k + 1)$st step is equally divided between the points misclassified on the kth step and those correctly classified.

Arc-x4 specifics.

1. Same as for arc-fs.
2. Run T down the classifier C_k and let $m(n)$ be the number of misclassifications of the nth case by $C_1, \ldots, C_k$.
3. The updated $k + 1$ step probabilities are defined by

$$p(n) = \left(1 + m(n)^4\right) / \sum \left(1 + m(n)^4\right).$$

After K steps the $C_1, \ldots, C_K$ are combined by unweighted voting.

After a training set T' is selected by sampling from T with probabilities $\{p(n)\}$, another set T'' is generated the same way. T' is used for tree construction, T'' is used as a test set for pruning. By eliminating the need for cross-validation pruning, 50 classification trees can be grown and pruned in about the same cpu time as it takes for 5 trees grown and pruned using 10-fold cross-validation. This is also true for bagging. Thus, both arcing and bagging, applied to decision trees, grow classifiers relatively fast. Parallel bagging can be easily implemented but arc is essentially sequential.

Here is how arc-x4 was devised. After testing arc-fs, I suspected that its success lay not in its specific form but in its adaptive resampling property, where increasing weight was placed on those cases more frequently misclassified. To check on this, I tried three simple update schemes for the probabilities $\{p(n)\}$. In each, the update was of the form $1 + m(n)^h$, and $h = 1, 2, 4$ was tested on the waveform data. The last one did the best and became arc-x4. Higher values of h were not tested so further improvement is possible.

4.2. *Experiments on data sets.* Our experiments used the six moderate sized data sets and four larger ones used in the bagging paper [Breiman (1996a)] plus a handwritten digit data set. The data sets are summarized in Table 4.

Of the first six data sets, all but the heart data are in the UCI repository. Brief descriptions are in Breiman (1996a). The procedure used on these data sets consisted of 100 iterations of the following steps:

1. Select at random 10% of the training set and set it aside as a test set.
2. Run 50 steps each of arc-fs and arc-x4 on the remaining 90% of the data.
3. Get error rates on the 10% test set.

The error rates computed in (3) are averaged over the 100 iterations to get the final numbers shown in Table 5. We note that using arc-fs in the soybean data set, frequently $\varepsilon_k > 0.5$, causing restarting.

TABLE 4
Data set summary

Data set	# Training	# Test	# Variables	# Classes
Heart	1395	140	16	2
Breast cancer	699	70	9	2
Ionosphere	351	35	34	2
Diabetes	768	77	8	2
Glass	214	21	9	6
Soybean	683	68	35	19
Letters	15,000	5000	16	26
Satellite	4,435	2000	36	6
Shuttle	43,500	14,500	9	7
DNA	2,000	1,186	60	3
Digit	7,291	2,007	256	10

The five larger data sets came with separate test and training sets. Again, each of the arcing algorithms was used to generate 50 classifiers (100 in the digit data) which were then combined into the final classifier. The test set errors are also shown in Table 5.

The first four of the larger data sets were used in the Statlog Project (Michie, Spiegelholter and Taylor (1994)], which compared 22 classification methods. Based on their results, arc-fs ranks best on three of the four and is barely edged out of first place on DNA. Arc-x4 is close behind.

The digit data set is the famous U.S. Postal Service data set as preprocessed by Le Cun, Boser, Denker, Henderson, Howard, Hubbard and Jackel (1990) to result in 16×16 grey-scale images. This data set has been used as a test bed for many adventures in classification at AT & T Bell Laboratories. The best two-layer neural net gets 5.9% error rate. A five-layer network gets down to 5.1%. Hastie and Tibshirani (1994) used deformable prototypes and

TABLE 5
Test set error (%)

Data set	Arc-fs	Arc-x4	Bagging	CART
Heart	1.1	1.0	2.8	4.9
Breast cancer	3.2	3.3	3.7	5.9
Ionosphere	6.4	6.3	7.9	11.2
Diabetes	26.6	25.0	23.9	25.3
Glass	22.0	21.6	23.2	30.4
Soybean	5.8	5.7	6.8	8.6
Letters	3.4	4.0	6.4	12.4
Satellite	8.8	9.0	10.3	14.8
Shuttle	0.007	0.021	0.014	0.062
DNA	4.2	4.8	5.0	6.2
Digit	6.2	7.5	10.5	27.1

got to 5.5% error. Using a very smart metric and nearest neighbors gives the lowest error rate to date—2.7% [Simard, Le Cun and Denker (1993)]. All of these classifiers were specifically tailored for this data.

The interesting Support Vector machines described by Vapnik (1995) are off-the-shelf, but require specification of some parameters and functions. Their lowest error rates are slightly over 4%. Use of the arcing algorithms and CART requires nothing other than reading in the training set, yet arc-fc gives accuracy competitive with the hand-crafted classifiers. It is also relatively fast. The 100 trees constructed in arc-fs on the digit data took about four hours of CPU time on a Sparc 20. Some uncomplicated reprogramming would get this down to about one hour of CPU time.

Looking over the test set error results, there is little to choose between arc-fs and arc-x4. Arc-x4 has a slight edge on the smaller data sets, while arc-fs does a little better on the larger ones.

5. Properties of the arc algorithms. Experiments were carried out on the six smaller sized data sets listed in Table 1 plus the artificial waveform data. Arc-fs and arc-x4 were each given lengthy 1000-step runs on each data set. In each run, information on various characteristics was gathered. We used this information to better understand the algorithms, their similarities and differences. Arc-fs and arc-x4 probably stand at opposite extremes of effective arcing algorithms. In arc-fs the constructed trees change considerably from one step to the next. In arc-x4 the changes are more gradual.

5.1. *Preliminary results.* Resampling with equal probabilities from a training set, about 37% of the cases do not appear in the resampled data set—put another way, only about 63% of the data is used. With adaptive resampling, more weight is given to some of the cases and less of the data is used. Table 6 gives the average percent of the data used by the arc algorithms in constructing each classifier in the sequence of 1000 steps. The third column is the average value of beta used by the arc-fs algorithm in constructing its sequence.

TABLE 6
Percent of data used

Data set	Arc-x4	Arc-fs	Average beta
Waveform	60	51	5
Heart	49	30	52
Breast cancer	35	13	103
Ionosphere	43	25	34
Diabetes	53	36	13
Glass	53	38	11
Soybean	38	39	17

Arc-x4 data use ranges from 35% to 60%. Arc-fs uses considerably smaller fractions of the data—ranging down to 13% on the breast cancer data set—about 90 cases per tree. The average values of beta are surprisingly large. For instance, for the breast cancer data set, a misclassification of a training set case lead to amplification of its (unnormalized) weight by a factor of 103. The shuttle data (unlisted) leads to more extreme results. On average, only 3.4% of the data is used in constructing each arc-fs tree in the sequence of 50 and the average value of beta is 145,000.

5.2. *A variability signature.* Variability is a characteristic that differed significantly between the algorithms. One signature was derived as follows: in each run, we kept track of the average value of $N^*p(n)$ over the 1000-step runs for each n. If the $\{p(n)\}$ were equal, as in bagging, these average values would be about 1.0. The standard deviation of $N^*p(n)$ for each n was also computed. Figure 1 gives plots of the standard deviations vs. the averages for the six data sets and for each algorithm. The upper point cloud in each graph corresponds to the arc-fs values; the lower to the arc-x4 values. The graph for the soybean data set is not shown because the frequent restarting causes the arc-fs values to be anomalous.

For arc-fs the standard deviations of $p(n)$ is generally larger than its average and increase linearly with the average. The larger $p(n)$, the more volatile it is. In contrast, the standard deviations for arc-x4 are quite small and only increase slowly with average $p(n)$. Further, the range of $p(n)$ for arc-fs is 2 to 3 times larger than for arc-x4. Note that, modulo scaling, the shapes of the point sets are similar between data sets.

5.3. *A mysterious signature.* In the 1000-step runs, we also kept track of the number of times the nth case appeared in a training set and the number of times it was misclassified (whether or not it was in the training set). For both algorithms, the more frequently a point is misclassified, the more its probability increases, and the more frequently it will be used in a training set. This seems intuitively obvious, so we were mystified by the graphs of Figure 2.

For each data set, number of times misclassified was plotted vs. number of times in a training set. The plots for arc-x4 behave as expected. Not so for arc-fs; their plots rise sharply to a plateau. On this plateau, there is almost no change in misclassification rate vs. rate in training set. Fortunately, this mysterious behavior has a rational explanation in terms of the structure of the arc-fs algorithm.

Assume that there are K iterations and that β_k is constant equal to β (in our experiments, the values of β_k had moderate sd/mean values for K large). For each n, let $r(n)$ be the proportion of times that the nth case was misclassified. Then

$$p(n) \cong \beta^{Kr(n)} / \sum \beta^{Kr(n)}$$

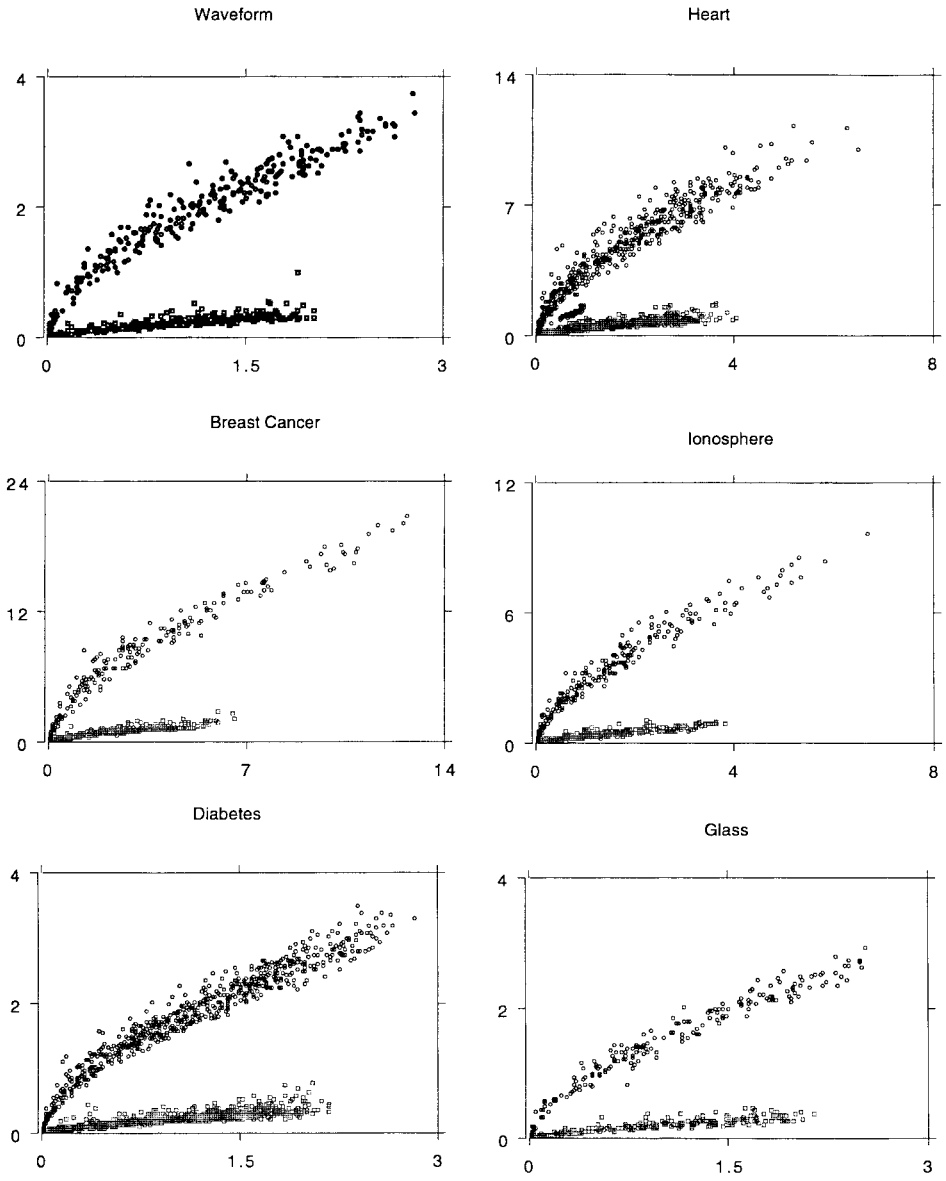

FIG. 1. *Standard deviations vs. averages for resampling probabilities.*

Let $r^* = \max_n r(n)$, L the set of indices such that $r(n) > r^* - \varepsilon$ for a small positive ε, and $|L|$ the cardinality of L. If $|L|$ is too small, then there will be an increasing number of misclassifications for those cases not in L that are not accurately classified by training sets drawn from L. Thus, their misclassification rates will increase until they get close to r^*. To illustrate this, Fig-

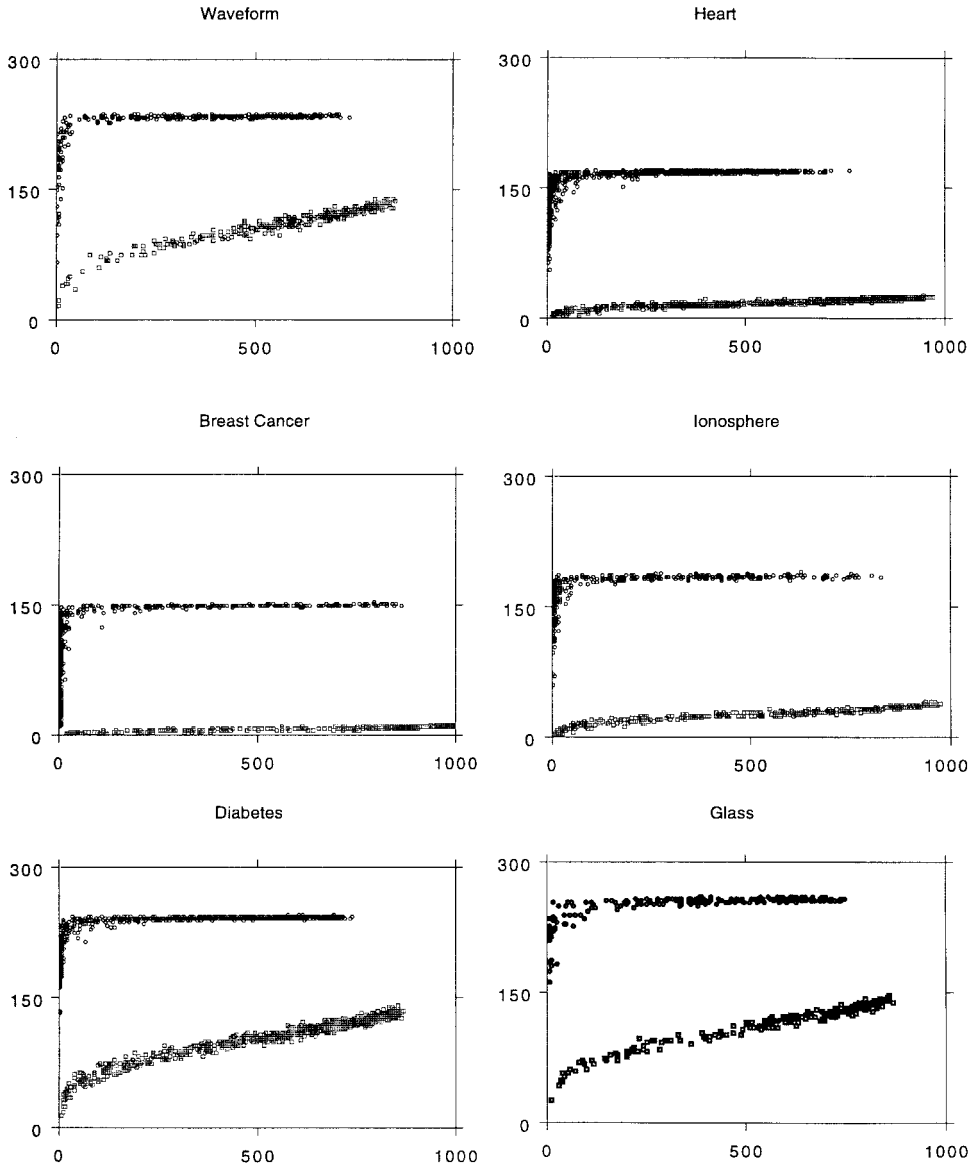

FIG. 2. *Number of misclassifications vs. number of times in training set.*

ure 3 shows the misclassification rates as a function of the number of iterations for two cases in the twonorm data discussed in the next subsection. The top curve is for a case with consistently large $p(n)$. The lower curve is for a case with $p(n)$ almost vanishingly small.

There are also a number of cases that are more accurately classified by training sets drawn from L. These are characterized by lower values of the

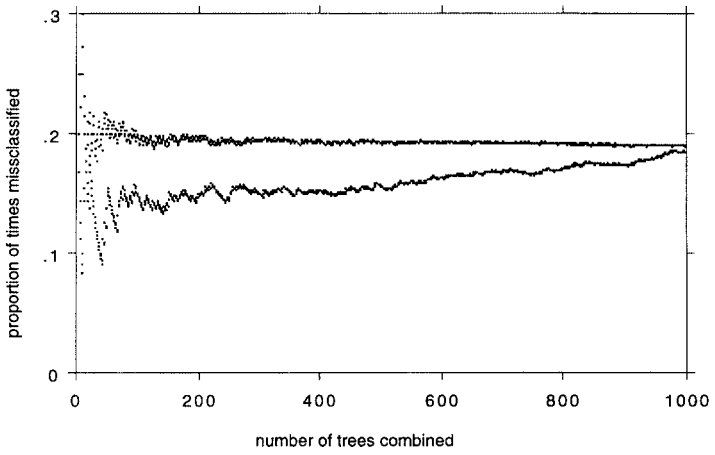

FIG. 3. *Proportion of times misclassified for two cases.*

misclassification rate, and by small $p(n)$. That is, they are the cases that cluster on the y-axes of Figure 2. More insight is provided by Figure 4. This is a percentile plot of the proportion of the training sets that the 300 cases of the twonorm data are in (10,000 iterations). About 40% of the cases are in a very small number of the training sets. The rest have a uniform distribution across the proportion of training sets.

5.4. *Do hard-to-classify points get more weight*? To explore this question, we used the twonorm data. The ratio of the probability densities of the two classes at the point $\mathbf{x}$ depends only on the value of $|(\mathbf{x}, \mathbf{1})|$ where $\mathbf{1}$ is the vector whose coordinates are all one. The smaller $|(\mathbf{x}, \mathbf{1})|$ is, the closer the

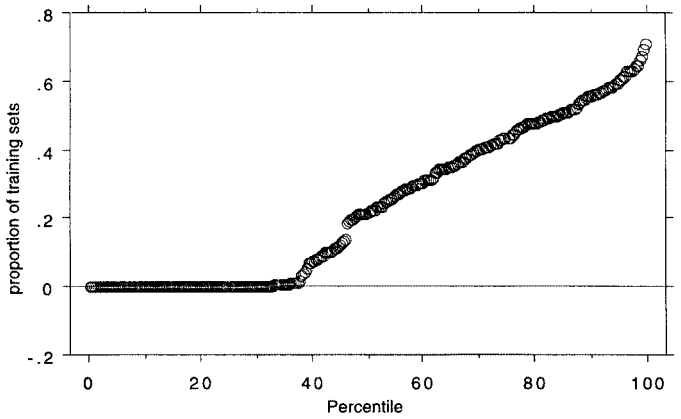

FIG. 4. *Percentile plot—proportion of training sets that cases are in.*

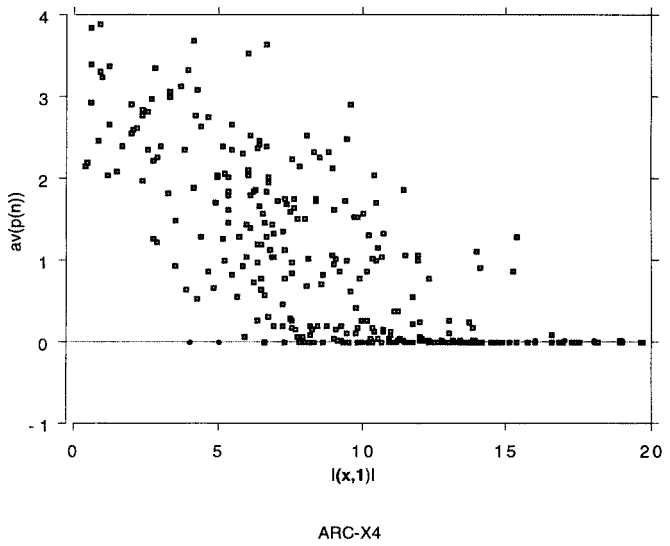

FIG. 5. *Average* $p(n)$ *vs.* $|(\mathbf{x}, \mathbf{1})|$.

ratio of the two densities to one, and the more difficult the point $\mathbf{x}$ is to classify. If the idea underlying the arc algorithms is valid, then the probabilities of inclusion in the resampled training sets should increase as $|(\mathbf{x}, \mathbf{1})|$ decreases. Figure 5 plots the average of $p(n)$ over 1000 iterations vs. $|(\mathbf{x}(n), \mathbf{1})|$ for both arc algorithms.

While av($p(n)$) generally increases with decreasing $|(\mathbf{x}(n), \mathbf{1})|$, the relation is noisy. It is confounded by other factors that I have not yet been able to pinpoint.

6. Linear discriminant analysis is not improved by bagging or arcing. Linear discriminant analysis (LDA) is fairly stable with low variance and it should come as no surprise that its test set error is not significantly reduced by use of bagging or arcing. Here our test bed was four of the first six data sets of Table 1. Ionosphere and soybean were eliminated because the within-class covariance matrix was singular, either for the full training set (ionosphere) or for some of the bagging or arc-fs training sets (soybean).

The experimental set-up was similar to that used in Section 2. Using a leave-out-10% as a test set, 100 repetitions were run using linear discriminant analysis alone and the test set errors averaged. Then this was repeated, but in every repetition, 25 combinations of linear discriminants were built using bagging or arc-fs. The test set errors of these combined classifiers were also averaged. The results are listed in Table 7.

Recall that for arc-fs, if $\varepsilon_k \geq 0.5$, then the construction was restarted with equal $\{p(n)\}$. The last column of Table 7 indicates how often restarting

TABLE 7
Linear discriminant test set error (%)

Data set	LDA	LDA: bag	LDA: arc	Restart frequency
Heart	25.8	25.8	26.6	1/9
Breast cancer	3.9	3.9	3.8	1/8
Diabetes	23.6	23.5	23.9	1/9
Glass	42.2	41.5	40.6	1/5

occurred. For instance, in the heart data, on the average, it occurred about once every nine times. In contrast, in the runs combining trees, restarting was encountered only on the soybean data. The frequency of restarting was also a consequence of the stability of linear discriminant analysis. If the procedure is stable, the same cases tend to be misclassified even with the changing training sets. Then their weights increase and so does the weighted training set error.

These results indicate that linear discriminant analysis is generally a low variance procedure. It fits a simple parametric normal model that does not change much with replicate training sets. To illustrate, we did a Monte Carlo estimation of bias and variance using the synthetic threenorm data. Recall that bias and variance for CART are 1.4% and 20.9%. For LDA they are 4.0% and 2.9%. The problem in LDA is usually bias; when it is wrong, it is consistently wrong, and with a simple model there is no hope of low bias across a variety of complex data sets.

7. Improved bagging. The aggregate classifier depends on the distribution P that the samples are selected from and the number N selected. Letting the dependence on N be implicit, denote $C_A = C_A(\mathbf{x}, P)$. As mentioned in Section 3.1, bagging replaces $C_A(\mathbf{x}, P)$ by $C_A(\mathbf{x}, P^{(B)})$ with the hope that this approximation is good enough to produce variance reduction. Now $P^{(B)}$, at best, is a discrete estimate for a distribution P that is usually smoother and more spread out than $P^{(B)}$. An interesting question is what a better approximation to P might produce.

To check this possibility, we used the four simulated data sets described in Section 3. Once a training set was drawn from one of these distributions, we replaced each $\mathbf{x}_n$ by a spherical normal distribution centered at $\mathbf{x}_n$. The bootstrap training set $T^{(B)}$ is iid drawn from this smoothed distribution. Two or three values were tried for the sd of the normal smoothing and the best one adopted. The results are given in Table 8.

The PE values for the smoothed P-estimates show that the better the approximation to P, the lower the variance. But there are limits to how well we can estimate the unknown underlying distribution from the training set. The aggregated classifiers based on the smoothed approximations had variances significantly above zero, and we doubt that efforts to refine the P

TABLE 8
Smoothed P-estimate bagging—test set errors (%)

Data set	Bagging	Bagging (smoothed)	Arcing
Waveform	19.8	18.4	17.8
Twonorm	7.4	5.5	4.8
Threenorm	20.4	18.6	18.8
Ringnorm	11.0	8.7	6.9

estimates will push them much lower. However, note that even with the better P approximation bagging does not do as well as arcing.

8. More on arcing.

8.1. *Training set error*. Arcing is much less transparent than bagging. Freund and Schapire (1997) designed arc-fs to drive training set error rapidly to zero, and it does remarkably well at this. But the context in which arc-fs was designed gives no clues as to its ability to reduce test set error. For instance, suppose we run arc-fs but exit the construction loop when the training set error becomes zero. The test set errors and number of steps to exit the loop (averaged over 100 iterations of leave-out-10% for the smaller data sets) are given in Table 9 and compared to the stop at $k = 50$ results from Table 2.

For the smaller data sets, we also kept track of the last step at which the training set error was nonzero. Their averages are given in the figures in parentheses in the last column of Table 9. These values show that once the training set error drops to zero, it almost invariably stays at zero. We also ran bagging on the first six data sets in Table 5, exiting the loop when the training error was zero, and kept track of the average number of steps to exit and the average test set error over 100 repetitions of leave-out-10%. These

TABLE 9
Test error (%) and exit times for arc-fs

Data set	Stop: $k = 50$	Stop: error = 0	Steps to exit
Heart	1.1	5.3	3.0 (3.1)
Breast cancer	3.2	4.9	3.0 (3.0)
Ionosphere	6.4	9.1	3.0 (3.1)
Diabetes	26.6	28.6	5.0 (5.1)
Glass	22.0	28.1	4.9 (5.1)
Letters	3.4	7.9	5
Satellite	8.8	12.6	5
Shuttle	0.007	0.014	3
DNA	4.2	6.4	5

TABLE 10
Test error (%) and exit times for bagging

Data set	Stop: $k = 50$	Stop: error = 0	Steps to exit
Heart	2.8	3.0	15
Breast cancer	3.7	4.1	55
Ionosphere	7.9	9.2	38
Diabetes	23.9	24.7	45
Glass	23.2	25.0	22

numbers are given in Table 10 (soybean was not used because of restarting problems).

These results delineate the differences between efficient reduction in training set error and test set accuracy. Arc-fs reaches zero training set error very quickly, after an average of five tree constructions (at most). But the accompanying test set error is higher than that of bagging, which takes longer to reach zero training set error. To produce optimum reductions in test set error, arc-fs must be run far past the point of zero training set error.

8.2. *Nonrandom arcing*. An important question in understanding arcing is what happens if the randomness is taken out. That is, in the definition of arc-fs in Section 4.1, everything remains the same except that instead of randomly sampling from the probabilities $\{p(n)\}$, these probabilities are fed directly into the classifier such that $p(n)$ is the weight for the case $(y_n, \mathbf{x}_n)$. CART was modified to use weights and run using a minimum node size of 10 on all data sets and the same test set error estimation methods used to get the Table 5 results. The results are given in Table 11 as compared to random arcing.

TABLE 11
Test set error arc-fs (%)

Data set	Random	Nonrandom
Heart	1.1	0.33
Breast cancer	3.2	3.0
Ionosphere	6.4	5.7
Diabetes	26.6	25.7
Glass	22.0	22.6
Soybean	5.8	5.7
Letters	3.4	2.8
Satellite	8.8	9.0
Shuttle	0.007	0.014
DNA	4.2	4.5
Digit	6.2	7.1

These results show that randomness is not an important element in arcing's ability to reduce test set error. This sharply differentiates arcing from bagging. In bagging the random sampling is critical, and the weights remain constant and equal.

8.3. *Remarks*. The arcing classifier is not expressible as an aggregated classifier based on some approximation to P. The distributions from which the successive training sets are drawn change constantly as the procedure continues. For the arc-fs algorithm, the successive $\{p(n)\}$ form a multivariate Markov chain. Suppose they have a stationary distribution $\pi(d\mathbf{p})$ (which I suspect is true, but a referee doubts). Let $Q(j|\mathbf{x}, \mathbf{p}) = P_T(C(\mathbf{x}, T) = j)$, where the probability P_T is over all training sets drawn from the original training set using the distribution $\mathbf{p}$ over the cases. Then, in steady-state with unweighted voting, class j gets vote $\int Q(j|\mathbf{x}, \mathbf{p})\pi(d\mathbf{p})$.

It is not clear how this steady-state probability structure relates to the error-reduction properties of arcing, but its importance is suggested by our experiments. The results in Table 3 show that arcing takes longer to reach its minimum error rate than bagging. If the error reduction properties of arcing come from its steady-state behavior, then this longer reduction time may reflect the fact that the dependent Markov property of the arc-fs algorithm takes longer to reach steady-state than bagging in which there is independence between the successive bootstrap training sets and the law of large numbers sets in quickly. But how the steady-state behavior of arcing algorithms relates to their ability to drive the training set error to zero in a few iterations is unknown.

Another complex aspect of arcing is illustrated in the experiments done to date. The diabetes data set gives higher error rate than a single run of CART. The Freund–Schapire (1996) and Quinlan (1996) experiments used C4.5, a tree structured program similar to CART and compared C4.5 to the arc-fs and bagging classifiers based on C4.5. In five of the 39 data sets examined in the two experiments, the arc-fs test set error was over 20% larger than that of C4.5. This did not occur with bagging. It is not understood why arc-fs causes this infrequent degeneration in test set error, usually with smaller data sets. One conjecture is that this may be caused by outliers in the data. An outlier will be consistently misclassified, so that its probability of being sampled will continue to increase as the arcing continues. It will then start appearing multiple times in the resampled data sets. In small data sets, this may be enough to warp the classifiers.

8.4. *Future work*. Arc-fs and other arcing algorithms can reduce the test set error of methods like CART to the point where they are the most accurate available off-the-shelf classifiers on a wide variety of data sets. The Freund–Schapire discovery of adaptive resampling as embodied in arc-fs is a creative idea which should lead to interesting research and better understanding of how classification works. The arcing algorithms have a rich probabilistic structure and it is a challenging problem to connect this struc-

ture to their error-reduction properties. It is not clear what an optimum arcing algorithm would look like. Arc-fs was devised in a different context and arc-x4 is ad hoc. While the concepts of bias and variance seem suitable as an explanation for bagging, and while arcing does effectively reduce variance, the bias-variance setting does not give a convincing explanation for why arcing works. Better understanding of how arcing functions will lead to further improvements.

APPENDIX

The boosting context of arc-fs. Freund and Schapire (1997) designed arc-fs to drive the training error rapidly to zero. They connected this training set property with the test set behavior in two ways. The first was based on structural risk minimization [see Vapnik (1995)]. The idea here is that bounds on the test set error can be given in terms of the training set error where these bounds depend on the VC-dimension of the class of functions used to construct the classifiers. If the bound is tight, this approach has a contradictory consequence. Since stopping as soon as the training error is zero gives the least complex classifier with the lowest VC dimension, then the test set error corresponding to this stopping rule should be lower than if we continue to combine classifiers. Table 10 shows that this does not hold.

The second connection is through the concept of boosting. Freund and Schapire (1997) devised arc-fs in the context of boosting theory [see Schapire (1990)] and named it Adaboost. We follow Freund and Schapire (1997) in setting out the definitions: assume that there is an input space of vectors $\mathbf{x}$ and an unknown function $\text{Co}(\mathbf{x}) \in \{0, 1\}$ defined on the space of input vectors $\mathbf{x}$ that assigns a class label to each input vector. The problem is to "learn" Co.

A classifying method is called a *weaklearner* if there exist $\varepsilon > 0$, $\delta > 0$ and integer N such that given a training set T consisting of $x_1, x_2, \ldots, x_N$ drawn at random from any distribution $P(\mathbf{dx})$ on input space together with the corresponding $j_n = \text{Co}(\mathbf{x}_n)$, $n = 1, \ldots, N$ and the classifier $C(\mathbf{x}, T)$ constructed, then the probability of a T such that $P(C(\mathbf{X}, T) \neq \text{Co}(\mathbf{X}) \mid T) < 0.5 - \varepsilon$ is greater than δ, where $\mathbf{X}$ is a random vector having distribution $P(\mathbf{dx})$.

A classifying method is called a *stronglearner* if for any $\varepsilon > 0$, $\delta > 0$ there is an integer N such that if it is given a training set T consisting of $x_1, x_2, \ldots, x_N$ drawn at random from any distribution $P(\mathbf{dx})$ on input space together with the corresponding $j_n = \text{Co}(\mathbf{x}_n)$, $n = 1, \ldots, N$, and the classifier $C(\mathbf{x}, T)$ constructed, then the probability of a T such that $P(C(\mathbf{X}, T) \neq \text{Co}(\mathbf{X}) \mid T) > \varepsilon$ is less than δ, where $\mathbf{X}$ is a random vector having distribution $P(\mathbf{dx})$.

Note that a stronglearner has low error over the whole input space, not just the training set; that is, it has small test set error. The concept of weak learning was introduced by Kearns and Valiant (1988, 1989), who left open the question of whether weak and strong learnability are equivalent. The question was termed the *boosting problem* since equivalent requires the

method to boost the low accuracy of a weaklearner to the high accuracy of a stronglearner. Schapire (1990) proved that boosting is possible. A *boosting algorithm* is a method that takes a weaklearner and converts it into a stronglearner. Freund and Schapire (1997) proved that an algorithm similar to arc-fs is boosting. Freud and Schapire (1997) prove that Adaboost is boosting.

The boosting assumptions are restrictive. For instance, if there is any overlap between classes (if the Bayes error rate is positive), then there are no weak or strong learners. Even if there is no overlap between classes, it is easy to give examples of input spaces and Co such that there are no weak learners. The boosting theorems really say "if there is a weak learner, then . . ." but in virtually all of the real data situations in which arcing or bagging is used, there is overlap between classes and no weak learners exist. Thus the Freund and Schapire (1997) boosting theorem is not applicable. In particular, it is not applicable in all of the examples of simulated data used in this paper and most, if not all, of the examples of real data sets used in this paper, in Freund and Schapire (1996) and in Quinlan (1996).

While there may be a connection between the ability of arcing algorithms to rapidly drive training set error to zero and their steady-state test set reduction, it is not rooted in the boosting context.

Acknowledgments. I am indebted to Yoav Freund for giving me the draft papers referred to in this article and to both Yoav Freund and Robert Schapire for informative E-mail interchanges and help in understanding the boosting context, to Trevor Hastie for making available the preprocessed U.S. Postal Service data, to Harris Drucker who responded generously to my questioning at NIPS95 and whose subsequent work on comparing arc-fs to bagging convinced me that arcing needed looking into, to Tom Dietterich for his comments on the first draft of this paper and to David Wolpert for helpful discussions about boosting. An Associate Editor and the referees made interesting and constructive comments that resulted in an improved paper. In particular, I am indebted to one of the referees for suggesting the experiment with nonrandom arcing reported in Section 8.2.

REFERENCES

ALI, K. (1995). Learning probabilistic relational concept descriptions. Ph.D. dissertation, Dept. Computer Science, Univ. California, Irvine.

BREIMAN, L. (1996a). Bagging predictors. *Machine Learning* **26** 123–140.

BREIMAN, L. (1996b). The heuristics of instability in model selection. *Ann. Statist.* **24** 2350–2383.

BREIMAN, L., FRIEDMAN, J., OLSHEN, R. and STONE, C. (1984). *Classification and Regression Trees*. Chapman and Hall, London.

DRUCKER, H. and CORTES, C. (1996). Boosting decision trees. *Advances in Neural Information Processing Systems* **8** 479–485.

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.

FREUND, Y. and SCHAPIRE, R. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. System Sci.* **55** 119–139.

FRIEDMAN, J. H. (1996). On bias, variance, 0/1-loss, and the curse of dimensionality. *Journal of Knowledge Discovery and Data Mining*. To appear.

GEMAN, S., BIENENSTOCK, E. and DOURSAT, R. (1992). Neural networks and the bias/variance dilemma. *Neural Computations* **4** 1–58.

HASTIE, T. and TIBSHIRANI, R. (1994). Handwritten digit recognition via deformable prototypes. Unpublished manuscript. Available at ftp `stat.stanford.edu/pub/hastie/zip.ps.Z`.

KEARNS, M. and VALIANT, L. G. (1988). Learning Boolean formulae or finite automata is as hard as factoring. Technical Report TR-14-88, Aiken Computation Laboratory, Harvard Univ.

KEARNS, M. and VALIANT, L. G. (1989). Cryptographic limitations on learning Boolean formulae and finite automata. In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*. 433–444. ACM Press, New York.

KOHAVI, R. and WOLPERT, D. H. (1996). Bias plus variance decomposition for zero-one loss functions. In *Machine Learning*: *Proceedings of the Thirteenth International Conference*. (L. Saitta, ed.) 275–283. Morgan Kaufmann, San Francisco.

KONG, E. B. and DIETTERICH, T. G. (1995). Error-correcting output coding corrects bias and variance. In *Proceedings of the Twelfth International Conference on Machine Learning* (A. Prieditis and S. Russell, eds.) 313–321. Morgan Kaufmann, San Francisco.

LE CUN, Y., BOSER, B., DENKER, J., HENDERSON, D., HOWARD, R., HUBBARD, W. and JACKEL, L. (1990). Handwritten digit recognition with a back-propagation network. *Advances in Neural Information Processing Systems* **2** 396–404.

MICHIE, D., SPIEGELHALTER, D. and TAYLOR, C. (1994). *Machine Learning*, *Neural and Statistical Classification*. Ellis Horwood, London.

QUINLAN, J. R. (1996). Bagging, Boosting, and C4.5. In *Proceedings of AAAI* '96 *National Conference on Artificial Intelligence* 725–730.

SCHAPIRE, R. (1990). The strength of weak learnability. *Machine Learning* **5** 197–227.

SIMARD, P., LE CUN, Y. and DENKER, J. (1993). Efficient pattern recognition using a new transformation distance. *Advances in Neural Information Processing Systems* **5** 50–58.

TIBSHIRANI, R. (1996). Bias, variance, and prediction error for classification rules. Technical Report, Dept. Statistics, Univ. Toronto.

VAPNIK, V. (1995). *The Nature of Statistical Learning Theory*. Springer, New York.

DEPARTMENT OF STATISTICS
UNIVERSITY OF CALIFORNIA
367 EVANS HALL #3860
BERKELEY , CALIFORNIA 94720-3860
E-MAIL: leo@stat.berkeley.edu

DISCUSSION

YOAV FREUND AND ROBERT E. SCHAPIRE

AT & T Labs

We thank Leo Breiman for his interest in our work on boosting, for his extensive experiments with the AdaBoost algorithm (which he calls arc-fs) and for his very generous exposition of our work to the statistics community. Breiman's experiments and our intensive e-mail communication over the last two years have inspired us to think about boosting in new ways. These new ways of thinking, in turn, led us to consider new ways for measuring the

performance of the boosting algorithm and for predicting its performance on out-of-sample instances.

It is exciting for us to have this communication channel with such a prominent practical statistician. As computer scientists we try to derive our algorithms from theoretical frameworks. While these frameworks cannot capture all of our prior beliefs about the nature of real-world problems, they can sometimes capture important aspects of the problem in new and useful ways. In our case, boosting was originally derived as an answer to a theoretical question posed by Kearns and Valiant [7] within the PAC framework, a model for the study of theoretical machine learning first proposed by Valiant [15]. We probably would have never thought about these algorithms had the theoretical question not been posed. On the other hand, an experimental statistician such as Leo is usually more interested in the actual behavior of algorithms on existing data sets and pays a lot of attention to the actual values of various variables during the run of the algorithm. Running AdaBoost on several synthetic and real-world data sets, Breiman observed that the algorithm has surprisingly low out-of-sample error, which, while consistent with our theory at the time, was not predicted by it. (Theories usually give only upper and lower bounds on the actual performance of algorithms. The gap between these bounds is a reflection of the degree to which our theories fail to reflect the world and of our shortcomings in the mathematical analysis.)

It is this challenge from the experiments of Breiman reported here, as well as those of Drucker and Cortes [3] and Quinlan [10], that prompted us to think harder about the problem and come up with a new theoretical explanation of the surprising behavior, which we describe in our paper with Bartlett and Lee [13]. This explanation suggests new measurable parameters, which can be tested in experiments, and the adventure continues! Theory suggests new algorithms and experiments, while experiments give rise to new observations which challenge the theory to come up with tighter bounds.

Our communication with Leo has been challenging and exciting. We hope to see further communication developing between researchers in computational learning theory and statistics.

Note: all the theoretical bounds to which we refer in this discussion are *nonasymptotic* and can be used to generate specific numerical bounds for finite sample sizes. However, these bounds are still numerically pretty loose.

1. Boosting and bagging. Breiman's paper is about improving the performance of a *learning algorithm*, sometimes also called a prediction algorithm or classification method. Such an algorithm operates on a given set of *instances* (or *cases*) to produce a *classification rule* which we refer to as a *hypothesis*. The goal of a learning algorithm is to find a hypothesis with low *generalization* or *prediction error*, that is, a low misclassification rate on a separate test set.

Bagging and boosting are two general methods for improving the performance of a given learning algorithm, which we call the *base* learning

algorithm. On a certain level, the algorithms are very similar; they both work by feeding perturbed versions of the training set to the base learning algorithm and combining the resulting rules by a majority vote.

While these similarities are apparent, there are some important differences between the two algorithms. Probably the most important difference is that the perturbation introduced by bagging are *random* and *independent* while the perturbations introduced by boosting (on a given training set) are chosen *deterministically* and *serially*, with the nth perturbation *depending strongly* on all of the previously generated rules.

In his paper, Breiman uses boosting-by-resampling, instead of boosting-by-reweighting and in this way combines the two methods. (In boosting-by-reweighting, we assume that the learning algorithm can work directly with a weighted training sample, while in boosting-by-resampling, training samples are generated by picking examples at random according to the distribution over the training set.) However, in Section 8.2, Breiman reports results from using the deterministic version of boosting and his results indicate that in the experiments reported here randomization is not an important element. On the other hand, Breiman has informed us that there are other, yet unpublished, experimental results regarding boosting of unpruned decision trees, in which boosting-by-resampling seems to have an advantage over boosting-by-reweighting.

It seems to us that the difference between boosting-by-reweighting and bagging should not be overlooked. Breiman analyzes both bagging and boosting in terms of the bias and variance decomposition of the error. We argue that this analysis is not appropriate for boosting. We have proposed a different analysis that seems to be appropriate for boosting but leaves out the effects of randomization. Giving a good theory of these randomization effects and a characterization of the cases in which they are advantageous is an interesting open problem.

The rest of this discussion is organized as follows. In Section 2, we give a historical perspective of the development of boosting algorithms. In Section 3, we summarize our arguments against explaining boosting using the bias-variance decomposition. In Section 4, we sketch our explanation for the small generalization error of boosting (a full description of this analysis appears in [13]). We conclude by describing some practical and theoretical open questions.

2. Boosting in and out of the PAC framework. The differences between boosting and bagging reflect the very different frameworks in which the two algorithms have been developed. Breiman [1, 2] developed bagging in the context of reducing the variance of learning algorithms, while boosting was developed as an answer to a theoretical question posed in [7] within the PAC learning literature. Stated somewhat informally, the question is: suppose we have a computationally efficient learning algorithm that can gener-

ate a hypothesis which is slightly better than random guessing for *any* distribution over the inputs. Does the existence of such a "weak" learning algorithm imply the existence of an efficient "strong" learning algorithm that can generate arbitrarily accurate hypotheses?

The answer to this question, given first by Schapire [12], is "yes." The proof is constructive; that is, it describes an efficient algorithm which transforms any efficient weak learning algorithm into an efficient strong one. Later, Freund [5] described a simpler and considerably more efficient boosting algorithm called *boost-by-majority*. Our AdaBoost algorithm is the most recent of the proposed boosting algorithms [6]. While nearly as efficient as boost-by-majority, AdaBoost has certain practical advantages over the preceding boosting algorithms, which we discuss below.

The goal of all boosting algorithms, starting with Schapire's, has always been to generate a combined hypothesis whose *generalization error* is small. One of the *means* for achieving this reduction has been to reduce the error of the combined hypothesis on the training set. The expectation that reducing the training error of the boosted hypothesis will also reduce the test error was justified by appealing to uniform convergence theory [16] (VC theory) or to arguments that rely on sample compression [4]. As a result of this analysis, the expectation was that there would be no point in running boosting beyond the point at which the training error of the combined hypothesis is zero.

It is only with the experimental work in [3] and [10] and the work of Breiman reported here, that it was realized that it is sometimes worthwhile to run the boosting algorithm beyond the point at which the error of the combined hypothesis is zero. The fact that doing so can cause the test error to decrease even further was very surprising to us, and, indeed, completely contradicted our intuitions about the relations between the training error and test error of learning algorithms.

The fact that these discoveries were made on the latest boosting algorithm "AdaBoost" rather than previous boosting algorithms has possibly more to do with the fact that this algorithm is very efficient in practice and less to do with its generalization properties. Indeed, both previous boosting algorithms can be classified as "arcing" algorithms according to Breiman's terminology. It would be interesting to see whether these previous boosting algorithms also decrease the test error after a training error of zero has been reached. In addition, boost-by-majority is quite different than AdaBoost in the way it treats outliers, which suggests that it might be worth exploring experimentally.

The reason that AdaBoost is especially efficient in practice is that it is *adaptive*. Previous boosting algorithms had to receive as input, in advance of observing the data, a parameter $\gamma > 0$. This parameter is the amount by which we believe that our classification method is better than random guessing. More formally, in order for the proof on the prediction error of the combined rule to work, the hypotheses generated by the weak learning algorithm have to all have error smaller than $1/2 - \gamma$. In practice, it is hard

to know how to set γ in advance. There are two aspects to this problem which we discuss in turn.

1. As Breiman remarks in the Appendix, it might be the case that for some distributions over the inputs there is *no* hypothesis whose error is smaller than $1/2$. This reflects a limitation of the original PAC framework, in which boosting was originally analyzed, where the assumption of weak learning is made uniformly with respect to *all* distributions. However, boosting algorithms can be analyzed outside this framework, as was done for AdaBoost where the theoretical framework of the analysis was expanded to better reflect the real-world problems. The cost of this expansion is that in the more general framework we lose the clear and simple definition of a "weak learner." In other words, the degree to which a learning algorithm can be boosted is characterized only as "an algorithm that gives small errors when run under AdaBoost." This is a very unsatisfying characterization because it involves both the base classification method and the boosting algorithm, while we would like to have a characterization that involves only the classification method. A somewhat different framework was used to give an extended analysis of boost-by-majority which, to some degree, overcomes this difficulty [5], Section 4.1.
2. It may be possible to use boost-by-majority in cases where the weak learning algorithm depends on the input distribution. The fact that we need to know γ in advance may not really be such a big problem because we might be able to run the algorithm several times and perform a binary search for the largest value of γ that works. However, there is an important *computational efficiency* problem. The problem is that the number of iterations required by boost-by-majority grows like $1/\gamma^2$ so, for example, if $\gamma = 0.01$ we need several tens of thousands of boosting iterations. This problem was fixed by AdaBoost, which can take advantage of the iterations in which the error of the weak hypotheses is smaller than $1/2 - \gamma$ and gain a large computational efficiency from that. While this last point might seem subtle, it is the main reason that AdaBoost is very efficient on real-world problems and this is ultimately the reason that it has so far played the dominant role in the application of boosting to real-world problems.

The development of boosting algorithms is a result of continuous interaction of practical and theoretical considerations, which demonstrates the importance of the interaction between these two modes of research. Breiman's work and our response to it represent the most recent chapter of this interaction.

3. The bias / variance explanation. Breiman's analysis of bagging and boosting is based on a decomposition of the expected error of the combined classifier into a bias term and a variance term. There are several difficulties

with the use of this analysis for boosting (more details are given in our paper with Bartlett and Lee [13]):

1. The bias-variance decomposition originates in the analysis of quadratic regression. Its application to classification problems is problematic, as reflected in the large number of suggested decompositions [8], [9], [14], in addition to the one given by Breiman in this paper. One unavoidable problem is that voting over several independently generated rules can sometimes increase, rather than decrease, the expected error.
2. Even in those cases where voting several independent classifiers is guaranteed to decrease the expected error, it is not always the case that voting over several bagged classifiers will do the same. The analysis of bootstrap estimation, which underlies bagging, has only asymptotic guarantees that might not hold for real-world sample sizes.
3. We have performed an analysis of the behavior of bagging and boosting on top of both Quinlan's C4.5 decision-tree algorithm [11] and an algorithm which we call "stumps," which generates the best rule which is a test on a single feature (i.e., a one-level decision tree or decision "stump"). We computed the bias and variance of these methods on some of the synthetic problems used by Breiman. We used the definitions of bias and variance of both Kong and Dietterich [9] and of Breiman. The results are summarized in Table 1. It seems clear that while bagging is mostly a variance reducing procedure, boosting can reduce both variance and bias. This is evident mostly in the experiments using stumps, which is a learning algorithm

TABLE 1
*Bias-variance experiments using boosting and bagging on synthetic data**

	Kong and Dietterich [9] definitions						**Breiman definitions**					
	Stumps			**C4.5**			**Stumps**			**C4.5**		
Name	**—**	**|Boost|**	**Bag**	**—**	**|Boost|**	**Bag**	**—**	**|Boost|**	**Bag**	**—**	**|Boost|**	**Bag**
Waveform												
bias	26.0	3.8	22.8	1.5	0.5	1.4	19.2	2.6	15.7	0.9	0.3	1.4
var	5.6	2.8	4.1	14.9	3.7	5.2	12.5	4.0	11.2	15.5	3.9	5.2
error	44.7	19.6	39.9	29.4	17.2	19.7	44.7	19.6	39.9	29.4	17.2	19.7
Twonorm												
bias	2.5	0.6	2.0	0.5	0.2	0.5	1.3	0.3	1.1	0.3	0.1	0.3
var	28.5	2.3	17.3	18.7	1.8	5.4	29.6	2.6	18.2	19.0	1.9	5.6
error	33.3	5.3	21.7	21.6	4.4	8.3	33.3	5.3	21.7	21.6	4.4	8.3
Threenorm												
bias	24.5	6.3	21.6	4.7	2.9	5.0	14.2	4.1	13.8	2.6	1.9	3.1
var	6.9	5.1	4.8	16.7	5.2	6.8	17.2	7.3	12.6	18.8	6.3	8.6
error	41.9	22.0	36.9	31.9	18.6	22.3	41.9	22.0	36.9	31.9	18.6	22.3
Ringnorm												
bias	46.9	4.1	46.9	2.0	0.7	1.7	32.3	2.7	37.6	1.1	0.4	1.1
var	−7.9	6.6	−7.1	15.5	2.3	6.3	6.7	8.0	2.2	16.4	2.6	6.9
error	40.6	12.2	41.4	19.0	4.5	9.5	40.6	12.2	41.4	19.0	4.5	9.5

*Columns labeled with a dash indicate that the base learning algorithm was run just once.

with very high bias. Moreover, in the experiment on the "ringnorm" data using boosting on stumps actually *increases* the variance, while at the same time it decreases the bias sufficiently to reduce the final error. These experiments demonstrate that variance-reduction cannot completely explain the performance of boosting.

4. The margins explanation. In [13], we describe an alternative explanation for the fact that boosting can often decrease the test error even after the training error is zero. Here is a sketch of the explanation. Assume, for the sake of simplicity, that the problem is a *binary* classification problem and that the two possible labels are -1 and $+1$. Denote the input by x, the prediction of the ith rule by $h_i(x)$ and the correct label by $y \in \{-1, +1\}$. Then the weighted vote rule generated by AdaBoost can be written as $\text{sign}(\sum_{i=1}^n \alpha_i h_i(x))$, where $\sum_{i=1}^n |\alpha_i| = 1$. The vote is correct on the example (x, y) if $y \sum_{i=1}^n \alpha_i h_i(x) > 0$. It is natural to think of $|\sum_{i=1}^n \alpha_i h_i(x)|$ as a measure of the *confidence* of the prediction. With this intuition in mind, we define $m(x, y) \doteq y \sum_{i=1}^n \alpha_i h_i(x)$ to be the *margin* of (x, y). A large positive margin indicates a confident correct prediction; a large negative margin indicates a confident but incorrect prediction and a small margin indicates unconfident predictions.

The claim of our explanation is that after boosting achieves zero training error, it goes on to generate a combined hypothesis whose margin is large on all of the examples in the training set and that it is this large margin that causes a decrease in the generalization error.

For example, in one experiment, we ran AdaBoost on top of C4.5 on the "letters" dataset, used also by Breiman in his paper. On the left of Figure 1, we have shown the training and test error curves (lower and upper curves, respectively) of the combined hypothesis as a function of the number of trees combined. The test error of C4.5 on this dataset (run just once) is 13.8%. The test error of boosting 1000 trees is 3.1%. (Both of these error rates are

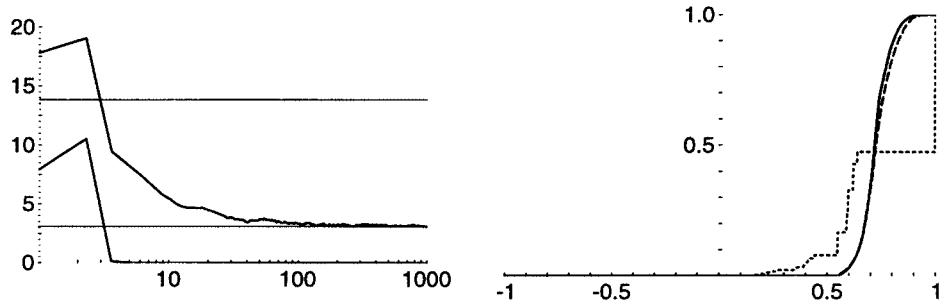

FIG. 1. *Error curves and the cumulative margin distribution graph for boosting C4.5 on the "letters" dataset.*

indicated in the figure as horizontal grid lines.) After just five trees have been combined, the training error of the combined hypothesis has already dropped to zero, but the test error continues to drop from 8.4% on round 5 down to 3.1% on round 1000.

As indicated above, our explanation for this phenomenon is based on the distribution of the margins of the training examples. We can visualize these margins by plotting their cumulative distribution (i.e., we can plot the fraction of examples whose margin is at most x as a function of $x \in [-1, 1]$). On the right side of Figure 1, we show the cumulative margin distributions that correspond to the experiment described above. The graphs show the margin distributions after 5, 100 and 1000 iterations, indicated by the short-dashed, long-dashed (mostly hidden) and solid curves, respectively.

Our main observation is that boosting tends to significantly increase the margins of the training examples, even after the training error reaches zero. In this case, although the training error remains unchanged (at zero) after round 5, the margin distribution changes quite significantly so that after 100 iterations all examples have a margin larger than 0.5. In comparison, on round 5, about 7.7% of the examples have margin below 0.5.

We present both experimental and theoretical evidence for the margin-based explanation for the effectiveness of boosting. Examining the margin distributions for a variety of problems and algorithms, we demonstrate a connection between the generalization error and the distribution of margins on the training set. We then back up these empirical observations with a theoretical explanation in two parts: first, we prove that, for sufficiently large training sets, there is a bound on the generalization error which is a function of the margin and which does *not* depend on the number of base hypotheses combined into the boosted hypothesis. Second, we prove that if the training errors of the base hypotheses are sufficiently small, then boosting is guaranteed to generate a combined hypothesis with large positive margins on all of the examples.

5. Some open problems. Breiman's work demonstrates the effectiveness of "perturb and combine" methods for reducing classification error. While a lot of understanding has been gained, many questions remain. Here are a few questions that seem particularly interesting to us.

1. What is the relation between the randomized effect of bagging and the deterministic effect of boosting? Can the two effects be separated in experiments?
2. Alternatively, is there a unified theory, based on provable theorems, which explains both boosting and bagging in a single framework?
3. Can we characterize the learning algorithms and/or the data generation processes which are most likely to benefit from boosting or from bagging or from their combination?

4. Is resampling the best way for randomly perturbing the training data? What about adding random noise to the label or to the features? How can we analyze these effects?
5. Quinlan [10] has reported that, in unusual cases, boosting can actually *increase* the generalization error of the base learning algorithm by a small amount. Can we characterize or predict when boosting will fail in this manner?
6. In Section 4, we discussed one explanation [13] for theoretically bounding the generalization error of voting methods like bagging and boosting. Are there other more practical and accurate methods for estimating the generalization error?

REFERENCES

[1] BREIMAN, L. (1996). Bagging predictors. *Machine Learning* **26** 123–140.
[2] BREIMAN, L. (1996). The heuristics of instability in model selection. *Ann. Statist.* **24** 2350–2383.
[3] DRUCKER, H. and CORTES, C. (1996). Boosting decision trees. *Advances in Neural Information Processing Systems* **8** 479–485.
[4] FLOYD, S. and WARMUTH, M. (1995). Sample compression, learnability, and the Vapnik–Chervonenkis dimension. *Machine Learning* **21** 269–304.
[5] FREUND, Y. (1995). Boosting a weak learning algorithm by majority. *Inform. and Comput.* **121** 256–285.
[6] FREUND, Y. and SCHAPIRE, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. System Sci.* **55** 119–139.
[7] KEARNS, M. and VALIANT, L. G. (1994). Cryptographic limitations on learning Boolean formulae and finite automata. *J. Assoc. Comput. Mach.* **4** 67–95.
[8] KOHAVI, R. and WOLPERT, D. H. (1996). Bias plus variance decomposition for zero-one loss functions. In *Machine Learning: Proceedings of the Thirteenth International Conference* (L. Saitta, ed.) 275–283. Morgan Kaufmann, San Francisco.
[9] KONG, E. B. and DIETTERICH, T. G. (1995). Error-correcting output coding corrects bias and variance. In *Proceedings of the Twelfth International Conference on Machine Learning* (A. Prieditis and S. Russell, eds.) 313–321. Morgan Kaufmann, San Francisco.
[10] QUINLAN, J. R. (1996). Bagging, boosting, and C4.5. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence* 725–730.
[11] QUINLAN, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Francisco.
[12] SCHAPIRE, R. E. (1990). The strength of weak learnability. *Machine Learning* **5** 197–227.
[13] SCHAPIRE, R. E., FREUND, Y., BARTLETT, P. and LEE, W. S. (1998). Boosting the margin: a new explanation for the effectiveness of voting methods. *Ann. Statist.* **26**. To appear.
[14] TIBSHIRANI, R. (1996). Bias, variance and prediction error for classification rules. Technical Report, Univ. Toronto.
[15] VALIANT, L. G. (1994). A theory of the learnable. *Communications of the ACM* **27** 1134–1142.
[16] VAPNIK, V. N. (1982). *Estimation of Dependences Based on Empirical Data*. Springer, New York.

AT & T LABS
180 PARK AVENUE
FLORHAM PARK, NEW JERSEY 07932-0971
E-MAIL: yoav@research.att.com
schapire@research.att.com

DISCUSSION

YALI AMIT[1] AND DONALD GEMAN[2]

University of Chicago and University of Massachusetts

The subject of multiple decision trees, and, more generally, of aggregating classifiers, deserves attention from the statistics community and we thank Leo Breiman for the spotlight provided by his article. Aggregation should and does work, there now being evidence from diverse quarters, and we appreciate the broad view taken here, incorporating recent developments in machine learning, neural networks and pattern recognition. We have also reported good results with multiple decision trees, originally for handwritten digit recognition [1], [4] and more recently for face detection [3]; in addition, we attempted to provide some theoretical understanding in [2]. We have a somewhat different opinion about *why* aggregation works and a sharply different view on "off-the-shelf" classifiers. It might be best to first explain the context of our work, since it is not exactly standard tree induction à la CART.

1. Shape quantization. Our interest is shape recognition. Developing algorithms which rival human performance has long vexed researchers in this domain, although there are some success stories in machine vision, for example in optical character recognition. Still, no machine can read the courtesy numbers on a bank check. Face recognition is another difficult problem; face detection is simpler. To be concrete we shall focus on the problem discussed in the article—recognizing isolated handwritten digits. This avoids complications which arise (and are arguably at the heart of the matter) in working with real images, such as those due to "context," "clutter" and the "segmentation problem." In addition, all our remarks are about multiple *trees*. However, most of the discussion about aggregation extends to other classifiers.

The raw data are in image format—a huge matrix of integers which represent quantized intensity values. There is no obvious "feature vector" other than the catenated intensities, which may vary in number from image to image. The digit recognition problem is to assign label $Y \in \{0, 1, \ldots, 9\}$ to each image I. Our approach begins with an enormous set of "queries" or "splitting rules" $\mathbf{Q} = \{Q_1, \ldots, Q_M\}$; these are binary functions of the image data, each corresponding to the presence ($Q(I) = 1$) or absence ($Q(I) = 0$) of a *geometric arrangement* of labeled edge fragments. Note that $\mathbf{Q}$ contains all

[1]Supported in part by Army Research Office Grant DAAH04-96-1-0061 and Department of Defense Grant DAAH04-96-1-0445.

[2]Supported in part by NSF Grant DMS-92-17655, ONR Contract N00014-91-J-1021 and Army Research Office (MURI) Grant DAAH04-96-1-0445.

possible arrangements of anything between two to several tens of edge fragments and is therefore virtually infinite. The exact nature of the queries is not important for this discussion. Suffice it to say that the total information content is very high: two images which answer the same to every query must contain *very* similar shapes. In fact, we assume (and we believe) that $\mathbf{Q}$ determines Y; thus the theoretical Bayes error rate is zero. However, it is not evident how we might determine a priori which features are particularly informative for separating digits and thereby reduce M to order hundreds or even thousands. In contrast, in standard CART the queries are of the form $Q_m = I_{\{X_i < c\}}$, where X_i is one component of a fixed-length feature vector and c is a constant. One could of course use multivariate functions or "trans-generated features" [7].

Suppose we examine *some* of the queries by constructing a single binary tree $\mathcal{T}$ by the usual data-driven induction method: stepwise entropy reduction estimated from a training set $\mathcal{L}$. Since we cannot entertain all possible splits at each node, we exploit a natural partial ordering on the set $\mathbf{Q}$ and examine only a tiny fraction of them. Basically we incrementally grow the geometric arrangements as we proceed down the tree. The classifier based on $\mathcal{T}$ is then $C(\mathbf{Q}, \mathcal{L}) = \arg\max_j P(Y = j \mid \mathcal{T})$. If the depths of the leaves of $\mathcal{T}$ are far smaller than M, then evidently $C(\mathbf{Q}, \mathcal{L})$ is not the Bayes classifier. However, for depths on the order of hundreds or thousands we could expect that

$$P(Y = j \mid \mathcal{T}) \approx P(Y = j \mid \mathbf{Q}),$$

the difference (in some appropriate norm) being a kind of "approximation error." Of course, we cannot actually create or store a tree of such depth, and the best classification rate we obtained with a single tree (of average depth around ten) was about 90% on test sets similar to the one discussed by Leo Breiman.

2. Multiple trees. Our justification for considering multiple trees was the simple fact that the amount of information about Y in $\mathcal{T}$ was far less than in $\mathbf{Q}$. Given the superior performance of "bagging" and "arcing," the same must be true relative to the query set $\{I_{\{X_i < c\}}\}_{i,c}$ for a standard feature vector. So our goal was to efficiently produce and aggregate a set of trees. Hopefully we could examine enough features to control the approximation error, provided we could suitably aggregate the evidence. The aggregation method we choose (see below) required that we also control another source of error: regardless of the depth of $\mathcal{T}$, we do not know $P(Y = j \mid \mathcal{T})$; rather it must be estimated from $\mathcal{L}$ (by simply counting the number of training images of each class j which land at each terminal node of $\mathcal{T}$). Call this estimate $\hat{P}_{\mathcal{L}}(Y \mid \mathcal{T})$. If $\mathcal{T}$ is not too deep then

$$\hat{P}_{\mathcal{L}}(Y = j \mid \mathcal{T}) \approx P(Y = j \mid \mathcal{T}).$$

The difference is "estimation error." *Both approximation error and estimation error are controlled by building multiple trees $\mathcal{T}_1, \ldots, \mathcal{T}_N$ of modest depth.* Consequently, another departure from the standard case is that we do

not continue "quantizing" the shapes until the nodes are "pure" and then "prune back." Instead, we stop splitting early enough to keep the estimation error low. Clearly this error tradeoff is related to that between "bias" and "variance."

In recent years various authors have proposed methods for generating multiple trees which are sufficiently "different" from one another to improve classification; for example, Shlien [9] used different splitting criteria. Breiman describes three procedures, one based on simple resampling ("bagging" [5]) and two based on adaptive resampling. The best results are obtained when the misclassified data points from an existing set of trees, $\mathcal{T}_1, \ldots, \mathcal{T}_k$, are given larger weight in the training set used to produce the next tree $\mathcal{T}_{k+1}$—the method of Freund and Shapire [6]. These weights effect either the bootstrap resampling scheme or, in the non-random case, the probabilities calculated by the classifier.

Since our aim is to access the information in $\mathbf{Q}$ we produce the different trees by using different subsets of the queries. We experimented in [1] with several protocols for choosing these subsets and settled on *randomization*. Suppose we are at a node t and $\mathbf{Q}_t$ is the subset of queries which are allowed as splitting rules. Although $\mathbf{Q}_t$ is only a small subset of $\mathbf{Q}$ (due to the growth restrictions mentioned above) it is still very large, typically thousands or tens of thousands. We select a *random subset* from $\mathbf{Q}_t$ and choose the query in that subset which yields the greatest reduction in uncertainty. In the experiments reported in [10] and [4], the size of the random subset is 100. Of course this requires having a very large pool of splitting rules in the first place.

The different trees created in this manner provide different "points of view" on the shapes. This can be illustrated graphically by looking at the final arrangements of edge fragments at the leaves reached by an image, from tree to tree. We were not able to achieve the same degree of variability by randomizing only on the training sample.

FIG. 1. *The arrangement found on a sample "3" in three different trees. The lines connect pairs of points for which the relative orientation is defined in the arrangement.*

3. Aggregation. In "bagging" and "arcing", each tree votes for a class and the trees are subsequently aggregated by a plurality rule. This leads to substantial reductions in the variance component of the total error. We experimented with many protocols for aggregation and settled on simple averaging in the following sense. Recall that for each tree $\mathscr{T}$, we store (an estimate of) the distribution $\{P(Y = j \mid \mathscr{T} = t),\ j = 0, \ldots, 9\}$ at leaf t. Each tree does not vote. Rather we average these distributions by computing

$$\overline{\mu}(j) = \overline{\mu}(j; \mathscr{T}_1, \ldots, \mathscr{T}_N) = \frac{1}{N} \sum_{n=1}^{N} P(Y = j \mid \mathscr{T}_n)$$

and then classify the digit with the mode of $\overline{\mu}$: $C(\mathbf{Q}, \mathscr{L}) = \arg\max_j \overline{\mu}(j)$. Our very first experiments with multiple trees lifted our classification rate to about 96% and eventually over 99%.

Our rationale for randomization was to produce trees $\mathscr{T}_1, \ldots, \mathscr{T}_N$ providing different points of view. Statistically speaking, this means the trees are *weakly dependent conditional on* Y, and it appears to us that this is the underlying explanation for the variance reduction of the classifiers that Leo Breiman analyzes. Probably "bagging" produces more conditionally dependent trees than "arcing." Randomization is just one way to accomplish this (indeed nonrandomized "arcing" works as well).

In [2] we tried to estimate the error rate as a function of second-order properties of the set of trees. Assume the *mean* aggregated distribution is well on target:

$$E(\overline{\mu}(j) \mid Y = j) \gg E(\overline{\mu}(k) \mid Y = j), \qquad k \neq j.$$

(This was true in our case.) Then a simple argument based on Chebyshev's inequality yields a crude bound on the misclassification error $P(C(\mathbf{Q}, \mathscr{L}) \neq j \mid Y = j)$ in terms of $E(\overline{\mu} \mid Y = j)$ and the conditional covariances $\mathrm{Cov}(P(Y = k \mid \mathscr{T}_n), P(Y = k \mid \mathscr{T}_m) \mid Y = j)$, $1 \leq n, m \leq N$. Naturally, small covariances lead to small errors. More generally, if the trees are produced with some sampling mechanism from the *population of trees*, involving either resampling from $\mathscr{L}$ or random restrictions on the queries, then the quantities above can be analyzed by taking expectations relative to the space of trees. In regard to arcing, probably the deterministic reweighting on misclassified examples produces new trees which are quite different from the previous ones. Moreover, the errors induced on data points which were correctly classified by the existing trees are sufficiently randomized to avoid any systematic deterioration.

4. Off-the-shelf classifiers. In his Introduction, Leo Breiman says: "Furthermore, the arc-classifier is off-the-shelf. Its performance does not depend on any tuning or settings for particular problems. Just read in the data and press the start button."

Most of the science in the imaging problems we study is in finding powerful representations and meeting domain-specific constraints. For example, it is futile to think about object recognition without thinking in advance about computation and about invariance with respect to photometric and geometric transformations which leave the class label unchanged. Moreover, the raw data are rarely directly accessed in learning-based methods and in general the appropriate "features" are by no means apparent. Echoing the discussion in [8], no "off-the-shelf" classifier is going to learn about these things no matter how much training data is provided.

More specifically, we do not preprocess the image data and compute a vector of features. Rather, we jointly induce the features and the classifier; in effect, the tree is the representation of the data. It might well be that most of the success of neural nets and related methods is due to painstaking efforts to find good problem-specific features, rather than due to the type of classifier. In particular, Leo Breiman's relative success on handwritten digits is likely due to the clever and thoughtful preprocessing and standardization methods developed at AT & T, and perhaps many other "off-the-shelf" classifiers can take one this far. But something else is needed to go further and match human performance.

REFERENCES

[1] AMIT, Y. and GEMAN, D. (1994). Randomized inquiries about shape; an application to handwritten digit recognition. Technical Report 401, Univ. Chicago.

[2] AMIT, Y. and GEMAN, D. (1997). Shape quantization and recognition with randomized trees. *Neural Computation* **9** 1545–1588.

[3] AMIT, Y., GEMAN, D. and JEDYNAK, B. (1998). Efficient focusing and face detection. In *Face Recognition*: *From Theory to Applications* (H. Wechsler and J. Phillips, eds.) Springer, Berlin.

[4] AMIT, Y., GEMAN, D. and WILDER, K. (1997). Joint induction of shape features and tree classifiers. *IEEE Trans. PAMI* **19** 1300–1306.

[5] BREIMAN, L., FRIEDMAN, J., OLSHEN, R. and STONE, C. (1984). *Classification and Regression Trees*. Wadsworth, Belmont, CA.

[6] FREUND, Y. and SCHAPIRE, R. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. System Sci.* **55** 119–139.

[7] FRIEDMAN, J. H. (1973). A recursive partitioning decision rule for nonparametric classification. *IEEE Trans. Comput.* **26** 404–408.

[8] GEMAN, S., BIENENSTOCK, E. and DOURSAT, R. (1992). Neural networks and the bias/variance dilemma. *Neural Computation* **4** 1–58.

[9] SHLIEN, S. (1990). Multiple binary decision tree classifiers. *Pattern Recognition* **23** 757–763.

[10] WILDER, K. (1998). Decision tree algorithms for handwritten digit recognition. Ph.D. dissertation, Univ. Massachusetts, Amherst.

DEPARTMENT OF STATISTICS
UNIVERSITY OF CHICAGO
CHICAGO, ILLINOIS 60637
E-MAIL: amit@galton.uchicago.edu

DEPARTMENT OF MATHEMATICS
AND STATISTICS
UNIVERSITY OF MASSACHUSETTS
AMHERST, MASSACHUSETTS 01003
E-MAIL: geman@math.umass.edu

DISCUSSION

T. G. DIETTERICH

Oregon State University

1. Introduction. Leo Breiman's article makes several important observations and raises many interesting questions about bagging and boosting (or "arcing") classifiers. The most important observation in the paper confirms the earlier results of Freund and Schapire (1996): ensembles of decision trees constructed via the AdaBoost algorithm can achieve excellent classification accuracy.

I will leave to the other discussants the task of dissecting the details of AdaBoost and Arc-X4 and instead focus on what I take to be the most fundamental question raised by this work: What is the best way to construct an ensemble of classifiers?

The goal of an ensemble classification methods is to construct an accurate and diverse set of classifiers. If such a collection of classifiers can be constructed, then simple or weighted voting will yield a better classifier, because a diverse set of classifiers will "scatter" their errors across the input space, and an accurate set of classifiers will make relatively few errors. Hence, a majority vote of accurate, diverse classifiers will yield more accurate classification decisions.

2. Manipulating the training data. What do we know about methods for constructing accurate, diverse classifiers? The bagging and boosting methods that Breiman describes can all be characterized as ways of manipulating the training set to generate diverse classifiers. As Breiman points out, this strategy only works if the fitting algorithms are unstable.

3. Manipulating the input features. Many other strategies have been explored recently. The first strategy involves manipulating the input features (i.e., the predictor variables). In settings where there are many highly correlated inputs, the fitting algorithm can be given different subsets of these inputs in each run. For example, in analyzing images from Venus, Cherkauer (1996) applied several different preprocessing methods (including principle components analysis and Fourier analysis) to produce 119 candidate input features. He then trained 32 neural networks based on eight different subsets of the features and four different hidden layer sizes. The resulting ensemble classifier was able to match the performance of human experts in identifying volcanoes.

4. Manipulating the output targets. A second strategy involves manipulating the output targets. Adopting Breiman's notation, suppose our data

[1]Supported by NSF Grant Number 9626584-IRI.

consist of pairs $(y_n, \mathbf{x}_n)$, $n = 1, \ldots, N$, where $y_n \in \{1, \ldots, J\}$ is the class label. Dietterich and Bakiri (1995) describe a technique called error-correcting output coding. Suppose that the number of classes, J, is large. Then derived classification problems $l = 1, \ldots, L$ can be constructed by randomly partitioning the J classes into two subsets A_l and B_l. The input data can then be relabeled so that any of the original classes in set A_l are given the derived label 0 and the original classes in set B_l are given the derived label 1. This relabeled data is then given to the fitting algorithm, which constructs a classifier h_l. By repeating this process L times (generating different subsets A_l and B_l), we obtain an ensemble of L classifiers $h_1, \ldots, h_L$.

Now given a new data point $\mathbf{x}$, how should we classify it? The answer is to have each h_l classify $\mathbf{x}$. If $h_l(\mathbf{x}) = 0$, then each class in A_l receives a vote. If $h_l(\mathbf{x}) = 1$, then each class in B_l receives a vote. After each of the L classifiers has voted, the class with the highest number of votes is selected as the prediction of the ensemble.

An equivalent way of thinking about this method is that each class j is encoded as an L-bit codeword C_j, where bit l is 1 if and only if $j \in B_l$. The lth learned classifier attempts to predict bit l of these codewords. When the L classifiers are applied to classify a new point $\mathbf{x}$, their predictions are combined into an L-bit string. We then choose the class j whose codeword C_j is closest (in Hamming distance) to the L-bit output string. Methods for designing good error-correcting codes can be applied to choose the codewords C_j (or equivalently, subsets A_l and B_l). The technique can naturally be extended to work with classifiers that estimate the probability that $\mathbf{x}$ belongs to each of the classes (or sets of classes).

Dietterich and Bakiri report that this technique improves the performance of both Quinlan's (1993) C4.5 decision-tree method and the backpropagation neural network method on a variety of difficult classification problems. Dietterich and Kong (1995) present data suggesting that output coding methods only work with global classifiers such as decision trees and neural networks and not with local methods such as the nearest neighbor rule or related kernel methods.

5. Randomizing the fitting procedure. A third strategy for constructing accurate and diverse ensembles is to randomize the fitting procedure. In neural network training, for example, the initial weights are typically set to small random values. Several different networks can be fit to the data by starting with different random weights. Perrone and Cooper (1993) and Hashem (1993) report good results on regression problems by taking a weighted vote of multiple networks trained in this fashion. Many people have experimented with randomizing tree fitting algorithms. Breiman mentions the work of Ali and Pazzani (1996), who employ a Boltzmann-like distribution to choose among the best candidate splits at each node in the tree. Kong and Dietterich (1995) experimented with a variant of C4.5 that chooses randomly (with equal probability) among the top 20 best candidate splits. They report

that this procedure gives performance comparable to (and sometimes much better than) bagging on several large classification tasks.

Perhaps the most complex methods for randomizing fitting algorithms are the Markov chain Monte Carlo (MCMC) techniques [Neal (1993)]. The basic idea of MCMC is to construct a Markov chain that generates an infinite sequence of hypotheses h_l. In a Bayesian setting, the goal is to generate an hypothesis h_l with probability $P(h_l|T)$, where T is the training sample. To apply MCMC, we define a set of operators that convert one h_l into another. For a neural network, such an operator might adjust one of the weights in the network. In a decision tree, the operator might interchange a parent and a child node in the tree or replace one node with another. The MCMC process works by maintaining a current hypothesis h_l. At each step, it selects an operator, applies it (to obtain h_{l+1}), and then computes the likelihood of the resulting classifier on the training data. It then decides stochastically, based on the computed likelihood, whether to keep h_{l+1} or discard it and go back to h_l. Under various technical conditions, it is possible to prove that a process of this kind will eventually converge to a stationary probability distribution in which the h_l's are generated in proportion to their posterior probabilities. In practice, it can be difficult to determine when this stationary distribution is reached. A standard approach is to run the Markov process for a long period (discarding all generated classifiers) and then collect a set of L classifiers from the Markov process. Markov chain Monte Carlo methods have been applied to neural networks by MacKay (1992) and Neal (1993) with impressive results, especially on small data sets.

6. Concluding remarks. This brief review shows that the bagging and boosting methods described by Breiman are just one of many different approaches to generating ensembles. An important question for future research is to understand how these different methods are related. For example, how is the boosting method related to the MCMC techniques? Both methods construct a complex stochastic process and then combine the classifiers generated by that process.

Another important question is to determine whether these different methods can be combined. One step in this direction is the recent work of Schapire (1997), which shows that AdaBoost can be combined with error-correcting output coding to yield an excellent ensemble classification method that he calls AdaBoost.OC. The performance of the method is superior to the ECOC method alone and to bagging alone. AdaBoost.OC gives essentially the same performance as another (quite complex) algorithm, called AdaBoost.M2. Hence, the main advantage of AdaBoost.OC is implementation simplicity: it can work with any fitting algorithm for solving two-class problems.

REFERENCES

ALI, K. M. and PAZZANI, M. J. (1996). Error reduction through learning multiple descriptions. *Machine Learning* **24** 173–202.

CHERKAUER, K. J. (1996). Human expert-level performance on a scientific image analysis task by a system using combined artificial neural networks. In *Working Notes of the AAAI Workshop on Integrating Multiple Learned Models* (P. Chan, ed.) 15–21. AAAI Press, Menlo Park, CA.

DIETTERICH, T. G. and BAKIRI, G. (1995). Solving multiclass learning problems via error-correcting output codes. *J. Artificial Intelligence Res.* **2** 263–286.

DIETTERICH, T. G. and KONG, E. B. (1995). Machine learning bias, statistical bias, and statistical variance of decision tree algorithms. Technical Report, Dept. Computer Science, Oregon State Univ., Corvallis, Oregon. Available from ftp://ftp.cs.orst.edu/pub/tgd/papers/tr-bias.ps/gz.

FREUND, Y. and SCHAPIRE, R. E. (1996). Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference on Machine Learning* (L. Saitta, ed.) 148–156. Morgan Kaufmann, San Francisco.

HASHEM, S. (1993). Optimal linear combinations of neural networks. Ph.D. dissertation, School of Industrial Engineering, Purdue Univ., Lafayette, IN.

KONG, E. B. and DIETTERICH, T. G. (1995). Error-correcting output coding corrects bias and variance. In *Twelfth International Conference on Machine Learning* (A. Prieditis and S. Russell, eds.) 313–321. Morgan Kaufmann, San Francisco.

MACKAY, D. (1992). A practical bayesian framework for backpropagation networks. *Neural Computation* **4** 448–472.

NEAL, R. (1993). Probabilistic inference using Markov chain Monte Carlo methods. Technical Report CRG-TR-93-1, Dept. Computer Science, Univ. Toronto.

PERRONE, M. P. and COOPER, L. N. (1993). When networks disagree: ensemble methods for hybrid neural networks. In *Neural Networks for Speech and Image Processing* (R. J. Mammone, ed.) 126–142. Chapman and Hall, London.

QUINLAN, J. R. (1993). *C*4.5: *Programs for Empirical Learning*. Morgan Kaufmann, San Francisco.

SCHAPIRE, R. E. (1997). Using output codes to boost multiclass learning problems. In *Proceedings of the Fourteenth International Conference on Machine Learning* 313–321. Morgan Kaufmann, San Francisco.

DEPARTMENT OF COMPUTER SCIENCE
OREGON STATE UNIVERSITY
303 DEARBORN HALL
CORVALLIS, OREGON 97331-3202
E-MAIL: tgd@cs.orst.edu

REJOINDER

LEO BREIMAN

University of California, Berkeley

First, I thank the discussants. All have made significant contributions to the general theme raised in my paper, that is, the idea that getting a collection of classifiers by perturbations of a single algorithm and having them vote for the class membership of a case can raise fairly mediocre classifiers into world-beater status.

As in Rashomon, we have different views of a phenomenon. But, unlike Rashomon, the diverse views add to the richness of the subject. I'll respond individually first and then make some general remarks. Before I begin, note

that terminology in the machine learning and neural network literature sometimes differs from that in statistics. For instance, what Dietterich calls a "hypothesis" we would call a "classifier." Freund and Schapire use the word "instance" instead of our "case."

Dietterich.

Generating the ensemble. I agree with Tom's assessment that the most crucial element in making voting methods work is how the ensemble of classifiers is generated. There is a secondary question of how to assign a vote to each one (Adaboost uses weighted voting; Amit and Geman average the class probabilities). Numbers of methods have been successfully used. Just among the discussants and myself, we have:

1. Alter the training set (Freund and Schapire–Adaboost, Breiman–bagging and arcing);
2. Randomize the choice of splits in the tree (Kong and Dietterich);
3. Randomize the set of features that can be split on (Amit and Geman).

In their 1995 paper, Dietterich and Kong generate an ensemble of trees by a random selection among the 20 best splits at every node and combine using unweighted voting. They analyze the performance in terms of a bias-variance framework, although with a slightly different definition than I use. But the idea is similar—variance is that component of error that can be reduced by ensemble voting. Bias cannot. Their method of generating an ensemble gives test set performance close to that of bagging. In particular, the variance reductions were similar.

Not all methods of generating ensembles are equal. For instance, in fitting neural nets by gradient descent, the initial weights (parameters) are taken as small random numbers. Starting from a different random selection of initial weights usually gives a different final neural net. So a seemingly natural way of generating an ensemble of neural nets is by using different random selections of initial weights. However, a number of studies have shown that this gives results decidedly inferior to bagging. The reason may be that changing the initial weights simply does not perturb enough. The bias-variance context says that the optimum reduction in variance is given when the ensemble of classifiers are grown on independent copies of the training set.

Any method of perturbation that does variance reduction succeeds by generating a similar ensemble. Bagging clearly mimics the independent copies. The random split selection succeeds by generating a similar ensemble, even though it does this by perturbing the classifier construction and not the training set. An interesting question raised by this line of reasoning is how to introduce enough randomness into the construction of a neural net so that the ensemble has the "independent copies" property.

(In a paper just received, Dietterich (1998) reports on a more extensive comparison of bagging, randomizing the choice of splits, and Adaboost. In these results, some systematic differences appear between bagging and ran-

domization, with randomization sometimes producing more accurate results than bagging. This deepens the mystery.)

Perturbing the data is more universal. An advantage of bagging or arcing is that they produce an ensemble of classifiers when applied to any algorithm. Thus, there is no need to tinker with the innards of the algorithm in order to perturb it. The perturbation is done to the data. Furthermore, the resulting accuracy can be surprisingly high even for very simple algorithms. For example, Ji and Ma (1996), apply a form of arcing to the two-class algorithm that passes a hyperplane through the space of the x-vectors and classifies all the points on one side as being in class 1 (2), on the other as class 2 (1). Using this simple classifier, they show that high test set accuracy can be gotten on large and complex data sets. Freund and Schapire run Adaboost on an algorithm called the stump, which is a tree restricted to do only one split. They show that even the stump can perform with high accuracy when voting is taken over the Adaboost ensemble (see Table 1 in their Discussion).

Coding classes. The two-class coding idea for multiple class problems impressed me when I first heard Tom talk about it some years ago, and I spent some time mulling it over. A nice feature is that, like perturbing training sets, it does not depend on the structure of the classification algorithm, and it has been applied both to neural nets and trees. Dietterich and Kong (1995) assert that it works by reducing bias. I'm not sure that is the right explanation, and I'm waiting for one that really hits the target for me. Overall, I think it is one of the better ideas to come up in the last few years.

Bayes. Dietterich is optimistic about the use of Bayesian methods and MCMC to generate the ensemble of classifiers. I am less so. The work in Neal (1993) uses an awesome amount of machinery and cpu cycles to get results on several small data sets. There has not yet been any testing on a suite of data sets comparable to the ones used in my paper. Seeing is believing!

Freund and Schapire.

Are increased margins the answer? The interaction between Freund and Schapire and me has been a creative one, leading to new insights. A number of empirical studies show that Adaboost generally works better than bagging. Furthermore, the empirical results presented in their comments show that while bagging is a variance reducing method, Adaboost works by reducing both bias and variance. So the question is, "Why?"

Freund and Schapire advance a plausible hypothesis—that Adaboost is more effective than bagging in increasing the margins. Their basic premise is that the higher the margins, the lower the generalization error (= infinite test set error). Recent work of mine [Breiman (1997)] shows that this is not the case.

In this latter paper, I formulate both regression and classification as sum–zero two-person games. For instance, a simplified version of the classification game is this: there is a finite collection H of classifiers $[h_m, m = 1, \ldots, M]$. Player II selects an $(y, \mathbf{x})$ from the training set. Player I, not knowing II's selection, chooses a classifier h from H. If $h(\mathbf{x}) \neq y$, then II wins one unit. A mixed strategy on I's part is equivalent to assigning a nonnegative vote to each classifier in H such that the votes total to one. A mixed strategy for II is a probability distribution on the training set.

The mini-max theorem then gives an upper bound for the minimum value of the margin over the training set. An arcing algorithm called arc-gv is found which converges to this upper bound. Arc-gv consistently produces higher values of the margins than Adaboost. For instance, on the ionosphere data, arc-gv produced margins that were larger than the corresponding Adaboost margin for every case in the training set, yet the test set errors for the combinations produced by arc-gv were generally higher over a variety of data sets than those produced by Adaboost. Trying to get the margins too high resulted in overfitting the data and increased test set error. So if the Freund–Schapire explanation is only a partial answer, then where are we?

Arc-gv goes to the bottom of the valley (highest margins) and then takes very small steps. Adaboost does not go to the bottom but circulates around the inner rim of the valley taking steps that are about the same size and generating votes that are about the same size. What I think is happening is that Adaboost and other arcing algorithms such as arc-x4 mark out a certain part of the training set population consisting of cases that are hard to classify and then circulate among subsets of this "difficult" subpopulation in a manner similar to bagging.

At any rate, the theoretical explanation is still unclear, but the empirical results are fascinating.

VC-type bounds may not tell the whole story. Another aspect of the interchange between us is the issue of VC-type bounds on the generalization error. These bounds [Vapnik (1995)] have received much attention in the machine learning and neural nets communities. Although they are generally too large to be useful estimates, they are considered to provide accurate theoretical guidance to the classification mechanism.

In their initial paper on Adaboost, Freund and Schapire (1996) used the concept of VC-dimension to explain the success of Adaboost in producing low generalization error. I pointed out that, as more and more classifiers were combined, the VC-dimension of the combined classifiers increased while the empirical estimates of the generalization error consistently decreased.

Freund and Schapire, together with Bartlett and Lee (1997) then found an elegant derivation showing that the generalization error of a convex combination of classifiers could be bounded in terms of the margin distribution and the VC-dimension of each classifier, no matter how many were combined. As their Discussion notes, this formed the theoretical guidance for their idea that the larger the margins, the lower the generalization error.

In my paper [Breiman (1997)] I derived a tighter VC-type bound based on the minimum value of the margin over the training set. Both bounds were still quite loose, but the important point is that these bounds do not tell the complete theoretical theory. On the surface, their implication is that if the margins are raised, the generalization error decreases, but the empirical evidence does not support this.

Open problems. Freund and Schapire list a number of interesting open problems. The first one concerns the difference between forming a new training set by resampling from the original set (weighted as in arcing or unweighted as in bagging) as contrasted to forming a new training set by altering the weights on each case.

My empirical results show that there is little difference between the two as far as Adaboost is concerned. But there is a large difference in implementation. I've found (and so has Dietterich) that when using trees, the best results are gotten by *not doing any pruning of the trees grown on the resampled data*. In general, each terminal node of the tree contains only the resampled duplicates of a single training case. Thus, very large trees are being combined, such that the terminal nodes of each represents only one training sample, yet overfitting does not occur. The fact that at each iteration the classifier works on a training set that excludes a large fraction of the original training set seems to regularize it. Furthermore, the combination process averages over a large number of these noisy trees. In the bias-variance framework, one could explain this by saying, "Grow large tree to reduce bias, then combine to reduce variance."

In reweighting, the size of the tree has to be limited. Assuming all weights nonnegative, the largest tree will contain one training case in each terminal node and have training error zero. Then Adaboost will stop. In my runs, I limited the size of the tree by not splitting any node of size 10 or less. This was a purely ad hoc rule but seemed to work fairly well. Even with this limitation, the trees grown are much larger than the optimally pruned trees.

In resampling and reweighting, an important ingredient is that the trees grown are large but still misclassify some of the cases. Then the weights on the points misclassified are increased to grow the next tree. The more curious of the two methods is resampling. Here, each individual training point, in the trees in which it occurs, is surrounded by a small rectangular region which contains no other distinct training point. Each test point is classified on the votes of these regions that it falls into. Thus, the method has the flavor of a nonmetric based k-nearest neighbor classification scheme.

Another question that Freund and Schapire raise is whether resampling or reweighting is the best way of perturbing the data and suggest the possibility of perturbing the class labels or features. As to the former, in Breiman (1996b) I got excellent results by averaging subset-selection regressions, each one grown by adding noise to the response variable only. Whether this can be extended to classification is so far unknown. An interesting sideline is that, in his 1998 study, Dietterich did some random perturbations of the class labels,

and found that bagging was more robust to this kind of noise than randomization, with Adaboost the least so.

As to perturbing the features, Dietterich, in his Discussion, points to a successful application by Cherkauer that grows neural nets on randomly selected features. In their work on images, Amit and Geman (1997) grow trees on randomly selected subsets of features. So we have at least two compelling pieces of evidence saying that growing classifiers on different sets of features and combining can give excellent accuracy. But to the best of my knowledge, there has been no systematic general investigation or attempt to construct a theoretical framework. Most of the other questions Freund and Schapire raise concern the need for theory to understand what is going on. I agree. We know what is happening in the laboratory; we watch the apple fall. But why?

Amit and Geman. The Amit and Geman article (1997) on shape recognition is a fascinating study and I recommend it to readers who are interested in pattern recognition. My take on their major contributions relevant to this discussion is this:

1. Don't worry about how many features you define in a complex problem. Define enough to distinguish the classes, the more, the merrier.
2. Grow many trees by presenting a random selection of features available for splitting at each node. Grow trees to a medium depth and average estimated class probabilities to combine the trees.

Get better solutions by increasing the dimensionality. For a long time, the prevalent wisdom in pattern recognition was to try to extract a small number of features that contained all of the relevant information, that is, reducing the dimensionality of the problem. Recent work has shown that the opposite approach may be more promising, that is, increasing the dimensionality drastically by increasing the number of input features. The trick, then, is to squeeze the relevant information out of this multitude of features without overfitting the data. For instance, the idea of support vector machines [Vapnik (1995)] for two-class problems is to increase the dimensionality of the input space of features until there is a hyperplane in the high-dimensional space that separates the classes. The Amit–Geman approach is another successful instance.

In the mid-1980s we had to keep telling new CART users not to worry about how many features they fed in. The tree structure will pick out those having information and the pruning will eliminate the noninformation splits. The tree structure does try to focus on the informative variables, but in the applications that Amit and Geman are focused on—with many thousands of features, each with a small piece of information—growing a single deep tree using splits on a single feature is not an efficient way of extracting the information. Instead, at each node, a large class of admissible features is

defined, a small subset of these is randomly selected and the best split on this subset found. With this, another interesting avenue is opened on growing multiple classifiers and combining them. But it also raises the question, "Why does it work; how can we understand it?" Amit and Geman combine trees by averaging class probabilities, thus putting it into a regression-like context. Why bagging works in regression has a clear and sensible explanation in terms of bias and variance, but an explanation for the Amit–Geman procedure seems further off.

How small a tree for accurate estimation? In each terminal node t of a tree, the estimate for $P(j|t)$ is the proportion of class j cases in node t. Amit and Geman want to keep the estimation error small, so they use trees with enough samples in each terminal node to make the estimation error small. Actually t is a big rectangle in multidimensional space. What is really required is not that the estimation error in $P(j|t)$ be small, but that the estimation error in $P(j|\mathbf{x})$ be small where $\mathbf{x}$ is any vector of features in t.

In Breiman (1996c) I show that on a variety of synthetic data sets the major part of the error in estimating $P(j|\mathbf{x})$ comes from the within-mode variability, that is, the error in approximating $P(j|\mathbf{x})$ by $P(j|t)$. The smaller the tree, the larger the within-node variability. Thus, the rationale used by Amit and Geman to justify smaller sized trees does not square with my empirical results. Also, both Dietterich and I and a number of others have found that, when combining, one should use very large trees. So there are two sets of empirical results pointing in directions that appear opposite.

Off-the-shelf-classifiers. Amit and Geman take umbrage at my statement that arcing CART provides an excellent off-the-shelf-classifier. But this is a simple statement of fact. I think our differences are due to our differing perspectives. There are many thousands of data analysts over the world, statisticians, economists, business analysts, biologists, and so on, who are trying to make sense out of their data. Often they are trying to form predictors of future outcomes to understand which structures are most predictive. They do not have the time or the technical skills to construct a state-of-the-art prediction method tailored to the precise problem on hand. So I believe it is an obligation of the statistical community to provide them with the best off-the-shelf tools available. As a result of the research on bagging and arcing, both are offered in the CART distribution.

From another perspective I am in complete agreement with them. On complex problems, to get past the point where off-the-shelf predictors can go, one needs to incorporate knowledge about the problem. For instance, the most accurate classifier to date on the AT & T digit recognition problem is a nearest neighbor classifier that uses a distance measure based on the nature of the problem, that is, locally invariant to transformations such as rotations, translations, thinning and so on. However, an important point here is that the off-the-shelf Euclidean distance nearest neighbor algorithm did pretty

well on this data to begin with. The strategy that seems to have worked the best on these complex problems is that one begins with a good off-the-shelf classifier and then, by incorporating problem-specific information, pushes up its accuracy.

Vapnik's off-the-shelf support vector machine gets a test set error of 1.1% on a NIST data base of 60,000 handwritten characters with a test set of 10,000. The lowest error to date is 0.7% resulting from a method of training a large neural net that is problem specific. On a different subset of the NIST data, the Amit–Geman trees get an error rate of 0.7%. Vapnik (1995) remarks, concerning closing the gap between 1.1% and 0.7%, "Probably one has to incorporate some a priori knowledge about the problem."

So I agree that to close the small gap between the best off-the-shelf methods and what is the best attainable by a machine algorithm, problem-specific knowledge will have to be incorporated, but it has often been accomplished by taking a good off-the-shelf method and tweaking it in the right directions.

General. The differing approaches presented by the discussants and my paper reflect the rich texture of this problem. The fact that in a wide range of problems, we can, by growing an ensemble of B-classifiers and combining, get an A+ classifier, is barely short of magical. As usual with magic, one wonders how it was done. The more empirical data we get, the more questions appear that are mystifying and urgent.

Crossing over. There has been great burst of work in the last decade in the machine learning and neural net community focused on hard problems in classification and regression. Many other areas have also come under fresh and enthusiastic scrutiny: unsupervised learning aka clustering, nonlinear principle components and factor analysis, hidden Markov chains, reinforced learning aka dynamic programming with large state spaces, graphical models, nonlinear control systems and so on and on. An idea of the breadth of this research can be gotten by looking at one of the Neural Information Processing Systems conference proceedings. The applications are generally to large and complex problems, of which speech and character recognition are the more recognizable, but they are also to things such as playing world-class backgammon using a neural net or controlling a Tokomak reactor.

Cross-overs between our field and theirs would encourage and strengthen research in important applied areas. The shared interests between fields is shown by discussants of this paper. Tom Dietterich is Editor of the *Journal of Machine Learning*. Yoav Freund and Robert Schapire have made important contributions to machine learning theory. Amit and Geman belong to statistics and mathematics, but have crossed over in the sense that their recent article on shape recognition appears in the *Journal of Neural Computing*. For the crossing over in this article and its discussion, I am grateful to the adventuresome spirit of John Rice, Coeditor of *The Annals of Statistics*.

REFERENCES

AMIT, Y. and GEMAN, D. (1997). Shape quantization and recognition with randomized trees. *Neural Computation* **9** 1545–1588.

BREIMAN, L. (1996c). Out-of-bag estimation. Available at `ftp.stat/users/breiman` as OOBestimation.

BREIMAN, L. (1997). Prediction games and arcing algorithms. Technical Report 504, Dept. Statistics, Univ. California, Berkeley. Available at `www.stat.berkeley.edu.`

DIETTERICH, T. (1998). An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting and randomization. *Machine Learning* 1–22.

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.

JI, C. and MA. S. (1997). Combinations of weak classifiers. *IEEE Trans. Neural Networks* **8** 32–42.

KONG, E. B. and DIETTERICH, T. G. (1995). Error-correcting output coding corrects bias and variance. In *Proceedings of the Twelfth International Conference on Machine Learning* (A. Prieditis and S. Russell, eds.) 313–321. Morgan Kaufmann, San Francisco.

SCHAPIRE, R., FREUND, Y., BARTLETT, P. and LEE, W. S. (1998). Boosting the margin: a new explanation for the effectiveness of voting methods. *Ann. Statist*. **26**. To appear.

VAPNIK, V. N. (1995). *The Nature of Statistical Learning Theory*. Springer, New York.

DEPARTMENT OF STATISTICS
UNIVERSITY OF CALIFORNIA
367 EVANS HALL
BERKELEY , CALIFORNIA 94720-3860
E-MAIL: leo@stat.berkeley.edu