TECHNICAL REPORT 460
April 1996

STATISTICS DEPARTMENT

UNIVERSITY OF CALIFORNIA

Berkeley, CA. 94720

# BIAS, VARIANCE , AND ARCING CLASSIFIERS

Leo Breiman*
Statistics Department
University of California
Berkeley, CA 94720
leo@stat.berkeley.edu

## ABSTRACT

Recent work has shown that combining multiple versions of unstable classifiers such as trees or neural nets results in reduced test set error. To study this, the concepts of bias and variance of a classifier are defined. Unstable classifiers can have universally low bias. Their problem is high variance. Combining multiple versions is a variance reducing device. One of the most effective is bagging (Breiman [1996a] ) 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 [1995,1996] propose an algorithm the basis of which is to **a**daptively **r**esample and **c**ombine (hence the acronym--arcing) so that the weights in the resampling are increased for those cases most often missclassified and the combining is done by weighted voting. Arcing is more successful than bagging in variance reduction. We explore two arcing algorithms, compare them to each other and to bagging, and try to understand how arcing works.

* Partially supported by NSF Grant 1-444063-21445

## 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--by generating multiple versions of the predictor by perturbing the training set or construction method and then combining these 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 generic of P&C (perturb and combine) to designate this group of methods.

One of the more effective of the 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, ..., N. Put equal probabilities p(n) = 1/N on each case, and using these probabilities, 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 recently [1995], [1996] proposed a P&C algorithm which was designed to drive the training set error rapidly to zero. But 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 missclassifed. 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 Appendix 2.

To better understand stability and instability, and what bagging and arcing do, in Section 2 we define the concepts of bias and variance for classifiers (Appendix 1 discusses some althernative definitions). The difference between the test set missclassification 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 classifers like linear discriminant analysis have low variance, but can have high bias. This is illustrated on several excamples 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. Arcing seems to usually do better at this than bagging . Arc-fs does complex things and its behavior is puzzling. But 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 ad hoc 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 get results more favorable to arcing. These are given in Section 4. 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 US 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 to always 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 and Breiman[1996a] noted that bagging does not improve nearest neighbor classification. Linear discriminant analysis is also relatively relatively stable (low variance) and in Section 6 our experiments show that neither bagging nor arcing has any effect on linear discriminant error rates.

Section 7 contains remarks--mainly aimed at understanding 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's possible that other arcing algorithms intermediate between acrc-fs and arc-x4 will give even better performance. The experiments here, in Freund-Shapire [1995] and in Drucker-Cortes[1995], and in Quinlan[1996] indicate that arcing decision trees may lead to fast and universally 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 order to understand how the methods studied in this article function, its helpful to define the bias and variance of a classifier. Since these terms originate in predicting numerical outputs, we first look at how they are defined in regression.

### 2.1 Bias and Variance in Regression

The terms bias and variance in regression come from a well-known decomposition of prediction error. Given a training set T = { ($y_n$,$\mathbf{x}_n$) n=1, ... ,N} where the $y_n$ are numerical outputs and the $\mathbf{x}_n$ are multidimensional input vectors, some method (neural nets, regression trees, linear regression, etc.) is applied to this data set to construct a predictor f($\mathbf{x}$,T) of future y-values. Assume that the training set T consists of iid samples from the distribution of Y,$\mathbf{X}$ and that future samples will be drawn from the same distribution. Define the squared error of f as

$$PE( f( ,T)) = E_{\mathbf{X},Y} (Y - f(\mathbf{X},T))^2$$

where the subscripts indicate expectation with respect to $\mathbf{X}$,Y holding T fixed. Let PE(f) be the expectation of PE( f( ,T)) over T. We can always decompose Y as:

$$Y = f^*(\mathbf{X}) + \varepsilon$$

where $E(\varepsilon \mid \mathbf{X})=0$. Let $f_A(x) = E_T f(\mathbf{x},T)$. Define the bias and variance as:

$$\text{Bias}(f) = E_{\mathbf{X}}(f^*(\mathbf{X}) - f_A(\mathbf{X}))^2$$

$$\text{Var}(f) = E_{T,\mathbf{X}}(f(\mathbf{X},T) - f_A(\mathbf{X}))^2$$

Then we get the **<u>Fundamental Decomposition</u>**

$$PE(f) = E\varepsilon^2 + \text{Bias}(f) + \text{Var}(f)$$

At each point $\mathbf{x}$ the contribution to the error at $\mathbf{x}$ from bias is $(f^*(\mathbf{x}) - f_A(\mathbf{x}))^2$ and that from variance is $E_T(f(\mathbf{x},T) - f_A(\mathbf{x}))^2$ . At some points bias predominates, at others the variance. But generally, at each point $\mathbf{x}$ both contributions are positive.

This decomposition is useful in understanding the properties of predictors. Usually some family $\mathfrak{I}$ of functions is defined and f is selected as the function in $\mathfrak{I}$ having minimum squared error over the training set. If $\mathfrak{I}$ is small, for instance, if $\mathfrak{I}$ is the set of linear functions, and f* is fairly nonlinear, then the bias will be large. But because we are only selecting from a small set of functions, i.e. estimating a small number of parameters, the variance will be low. But if $\mathfrak{I}$ is a large family of functions, i.e. the set of functions represented by a large neural net or by binary decision trees, then the bias is usually small, but the variance large. An illuminating discussion of this problem in the context of neural networks is given in Geman, Bienenstock, and Doursat[1992].

The cure for bias is known to every linear regression practitioner--enlarge the size of the family $\mathfrak{I}$. Add quadratic and interaction terms, maybe some cubics, etc. But in doing this, while the bias is decreased, the variance goes up. But there may be some partial cures for high variance. Consider the aggregated predictor $f_A(x)$. By definition, $f_A(x)$ has the same bias as $f(\mathbf{x})$ but has <u>zero variance.</u> If we could approximate $f_A(x)$, then we get a predictor with reduced variance. As we will see, this simple idea carries over into classification.

## 2.2 <u>Bias and Variance in Classification</u>

In classification, the output variable $y \; \varepsilon \; \{1, \ldots ,J\}$ is a class label. The training set T is of the form $T = \{ (y_n,x_n) \; n=1, \ldots ,N\}$ 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 missclassification error is defined as:

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

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

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

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

$$C^*(\mathbf{x}) = \text{argmax}_j \, P(j \mid \mathbf{x})$$

with missclassification rate

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

In defining bias and variance in regression, the key ingredient was the definition of the aggregated predictor $f_A(\mathbf{X})$. A different definition is useful in classification. Let

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

and define the aggregated classifier as:

$$C_A(\mathbf{x}) = \text{argmax}_j \, Q(j \mid \mathbf{x}).$$

This is aggregation by voting. Consider many independent replicas $T_1$, $T_2$, ... ; construct the classifiers $C(\mathbf{x},T_1)$, $C(\mathbf{x},T_2)$, ... ; 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**

$C(\mathbf{x},T)$ *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 $P(1 \mid \mathbf{x}) = .9$, $P(2 \mid \mathbf{x}) = .1$, and $Q(1 \mid \mathbf{x}) = .6$, $Q(2 \mid \mathbf{x}) = .4$. Then C is unbiased at $\mathbf{x}$ but the probabablilty of correct classification by C is $.6 \times .9 + .4 \times .1 = .58$. But the Bayes predictor C* has probability .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. The complement of U is called the bias set and denoted by B. Define

**Definition 2.2**

*The bias of a classifier C is*

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

*and its variance is*

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

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 these properties:

a) Bias and variance are always non-negative.
b) The variance of $C_A$ is zero.
c) If C is deterministic, i.e, does not depend on T, then its variance is zero.
d) The bias of C* is zero.

The proofs of a)-d) are immediate from the definitions. The variance of C can be expressed as

$$\mathrm{Var}(C) = \int_U [\max_j P(j\,|\,\mathbf{x}) - \Sigma_j Q(j\,|\,\mathbf{x})\, P(j\,|\,\mathbf{x})\,]\; P(d\mathbf{x})\,.$$

The bias of C is a similar integral over B. Clearly, both bias and variance are non-negative. 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, its clear that C* has zero bias.

In distinction to the defintion of bias and variance for regression, in classification each point **x** is either in the bias set or in the variance set. If it is in the bias set, then the variance at **x** is zero. Converseley, if it is in the variance set, the bias at **x** is zero. This reflects the difference between classification and regression. In classification, you either get it right or wrong. In regression. the error is continuous. See the Appendix for further remarks about the definition of bias and variance.

### 2.3 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(**x**,T) can differ markedly from each other and from the aggregated classifier $C_A$(**x**). Stable classifiers do not change much over replicates of T, so C(**x**,T) and $C_A$(**x**) will tend to be the same and the variance will be small.

Procedures like trees 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.

### 2.3 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.

i) *waveform:* This is 21 dimension, 3 class data. It is described in the CART book (Breiman et.al [1984]) and code for generating the data is in the UCI repository. PE(C*) = 13.2%

ii) *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, ... ,a) and class #2 has mean (-a,-a, ... ,-a). PE(C*) = 2.3%. $a=2/(20)^{1/2}$

iii) *threenorm:* This is 20 dimension, 2 class data. Class #1 is drawn with equal probability from a unit multivariate normal with mean (a,a, ... ,a) and from a unit multivariate normal with mean (-a,-a, ... ,-a). Class #2 is drawn from a unit multivariate normal with mean at (a,-a,a,-a, ... .a). PE(C*) = 10.5%. $a=2/(20)^{1/2}$

iv) *ringnorm:* This is 20 dimension, 2 class data Class #1 is multivariate normal with mean zero and covariance matrix 4 times the identity. Class #2 has unit covariance matrix and mean (a,a, ...,a). PE(C*) = 1.3%. $a=1/(20)^{1/2}$

Monte Carlo techniques were used to compute bias and variance. The results are in Table 1.

Table 1 Bias, Variance and Error of CART (%)

| Data Set | Bias | Variance | Error |
|---|---|---|---|
| waveform | 1.7 | 14.1 | 29.0 |
| twonorm | .1 | 19.6 | 22.1 |
| threenorm | 1.4 | 20.9 | 32.8 |
| ringnorm | 1.5 | 18.5 | 21.4 |

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 peformance of CART on these data sets.

## 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 $C_A(\mathbf{x})\cdot$ This classifier has zero variance and low bias. Specifically, on the four problems above its bias is 2.9, .4, 2.6, 3.4. Thus,it is nearly optimal. Recall that it is based on generating independent replicates of T, constructing multiple classifiers using these 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. T' is more familiarly called a bootstrap sample from T.

Denote the distribution on T given by {p(n)} as $P^{(B)}$. T' is iid from $P^{(B)}$. Repeat this sampling procedure, getting a sequence of independent bootstrap training sets. Form classifiers based on these training sets and have them vote for the classes. Now $C_A(\mathbf{x})$ really depends on the underlying probability P that the training sets are drawn from i.e. $C_A(\mathbf{x}) = C_A(\mathbf{x}, P)$. The bagged classifier is $C_A(\mathbf{x}, P^{(B)})$. The hope is that this is a good enough approximation to $C_A(\mathbf{x}, P)$ 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. Then the probabilities are updated depending on how the cases in T are classified by C(**x**,T'). A factor $\beta > 1$ is defined which depends on the missclassification rate--the smaller it is, the larger $\beta$ is. If the nth case in T is missclassified by C(**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 missclassified. 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. The results are given in Table 2 and compared with the CART results.

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 |

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 4. Standard errors average about 0.1%

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 | 6.9 | 6.9 | 7.0 | 6.6 |

Arc-fs error rates decrease significantly out to 250 combination, reaching rates close to the Bayes minimums (13.2% for waveform and 2.3% for twonorm). Bagging error rates do not decrease markedly. One standard of comparison is linear discriminant analysis, which should be almost optimal for twonorm. It has an error rate of 2.8%, averaged over 100 repetitions.

## 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 missclassifications 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 [1995]. Arc-x4 is an ad hoc invention.

arc-fs specifics:

i) At the kth step, using the current probabilities{p(n)}, sample with replacement from T to get the training set $T^{(k)}$ and construct classifier $C_k$ using $T^{(k)}$.

ii) Run T down the classifier $C_k$ and let d(n)=1 if the nth case is classified incorrectly, otherwise zero.

iii) Define

$$\varepsilon_k = \Sigma_n p(n)d(n), \quad \beta_k = (1 - \varepsilon_k)/\varepsilon_k$$

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

$$p(n) = p(n)\beta_k{}^{d(n)} / \Sigma p(n)\beta_k{}^{d(n)}$$

After K steps, the $C_1, \ldots, C_K$ are combined using weighted voting with $C_k$ having weight $\log(\beta_k)$. Two revisions to this algorithm are necessary. If $\varepsilon_k$ becomes equal to or great than 1/2, then the original Feund and Schapire algorithm exits from the construction loop. We found that better results were gotten by setting all {p(n)} equal and restarting. This happened frequently on the soybean data set. If $\varepsilon_k$ to equals zero, making the subsequent step undefined, we again set the probabilities equal and restart.

arc-x4 specifics:

i) Same as for arc-fs

ii) Run T down the classifier $C_k$ and let m(n) be the number of missclassifications of the nth case by $C_1, \ldots, C_k$.

iii) The updated k+1 step probabilities are defined by

$$p(n) = (1+ m(n)^4) / \Sigma(1+ m(n)^4)$$

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 missclassified. 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 6 moderate sized data sets and 4 larger ones used in the bagging paper (Breiman [1996a] plus a handwritten digit data set. The data sets are summarized in Table 4.

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 |

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

i) Select at random 10% of the training set and set it aside as a test set.

ii) Run arc-fs and arc-x4 on the remaining 90% of the data, generating 50 classifiers with each.

iii) Combine the 50 classifiers and get error rates on the 10% test set.

The error rates computed in iii) are averaged over the 100 iterations to get the final numbers shown in Table 5.

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 2.

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 | .007 | .021 | .014 | .062 |
| DNA | 4.2 | 4.8 | 5.0 | 6.2 |
| digit | 6.2 | 7.5 | 10.5 | 27.1 |

The first four of the larger data sets were used in the Statlog Project (Michie et.al. 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 US Postal Service data set as preprocessed by Le Cun et. al [1990] to result in 16x16 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 used deformable prototypes [1994] and get to 5.5% error. Using a very smart metric and nearest neighbors gives the lowest error rate to date--2.7% (P. Simard et. al [1993]). All of these classifiers were specifically tailored for this data.

The interesting SV 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-fs gives accuracy competitive with the hand-crafted classifiers. It is also relatively fast. The 100 trees constructed in arc-fs took about 4 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 runs on each data set--generating sequences of 1000 trees. 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 construction 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 3 gives the average percent of the data used by the arc algorithms in constructing each classifier in a sequence of 100. 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 | av. 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 missclassification 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 run 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 six the 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.

Figure 1

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 quit small and only increase slowly with average p(n). Further, the range of p(n) for arc-fs is 2-3 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 each run of 1000, we also kept track of the number of times the nth case appeared in a training set and the number of times it was missclassified. For both algorithms, the more frequently a point is missclassified, the more its probability increases, and the more fequently it will be used in a training set. This seems intuitively obvious, so we were mystified by the graphs of figure 2.

Figure 2

For each data set, number of times missclassified 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 missclassification 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 $\beta_k$ is constant equal to $\beta$ (in our experiments, the values of $\beta_k$ had moderate sd/mean values for K large). For each n, let r(n) be the proportion of times that the nth case was missclassified. Then

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

Let $r^* = \max_n r(n)$, L the set of indices such that $r(n) > r^* - \varepsilon$ and $|L|$ the cardinality of L . If $|L|$ is too small, then there will be an increasing numbers of missclassifications for those cases not in L that are not accurately classified by training sets drawn from L. Thus, their missclassification rates will increase until they get close to r*. To illustrate this, Figure 3 shows the missclassification 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.

Figure 3

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 missclassification 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 train sets. The rest have a uniform distribution across the proportion of training sets.

Figure 4

### 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 **x** depends only on the value of $|(\mathbf{x},\mathbf{1})|$ where **1** is the vector whose coordinates are all one. The smaller $|(\mathbf{x},\mathbf{1})|$ is, the closer the ratio of the two densities to one, and the more difficult the point **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.

Figure 5

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 Isn't 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 4.

Table 7 Linear Discriminant Test Set Error(%).

| Data Set | LDA | LDA: bag | LDA: arc | Restart Freq. |
|---|---|---|---|---|
| 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 |

Recall that for arc-fs, if $\varepsilon_k \geq .5$, then the construction was restarted with equal {p(n)}. The last column of Table 4 indicates how often restarting occurred. For instance, in the heart data, on the average, it occurred about once every 9 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 disciminant analysis. If the procedure is stable, the same cases tend to be missclassified even with the changing training sets. Then their weights increase and so does the weighted training set error.

These results illustrate 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. The problem is bias--when it is wrong, it is consistently wrong, and with a simple model there is no hope of generally low bias.

## 7. **Remarks**

### 7.1 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 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 9.

Table 9 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 |

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

### 7.2 Arcing

Arcing is much less transparent than bagging. Freund and Schapire [1995] 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 average number of combinations to exit the loop are given in Table 10 and compared to the stop at k=50 results from Table 2. 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 combinations to exit and the test set error. These numbers are given in Table 10 (soybean was not used because of restarting problems).

Table 10 Test Error(%) and Exit Times for Arc-fs

| Data Set | stop: k=50 | stop: error=0 | exit time |
|---|---|---|---|
| heart | 1.1 | 5.3 | 3 |
| breast cancer | 3.2 | 4.9 | 3 |
| ionosphere | 6.4 | 9.1 | 3 |
| diabetes | 26.6 | 28.6 | 5 |
| glass | 22.0 | 28.1 | 5 |
| letters | 3.4 | 7.9 | 5 |
| satellite | 8.8 | 12.6 | 5 |
| shuttle | .007 | .014 | 3 |
| DNA | 4.2 | 6.4 | 5 |

Table 11 Test Error(%)and Exit Times for Bagging

| Data Set | stop: error=0 | exit time |
|---|---|---|
| heart | 3.0 | 15 |
| breast cancer | 4.1 | 55 |
| ionosphere | 9.2 | 38 |
| diabetes | 24.7 | 45 |
| glass | 25.0 | 22 |

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 5 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.

The arcing classifier is not expressible as 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 and probably have a stationary distribution $\pi(d\mathbf{p})$. Let $Q(j \mid \mathbf{x},\mathbf{p}) = P_T(C(\mathbf{x},T)=j)$, 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 $\int Q(j \mid \mathbf{x},\mathbf{p})\, \pi(d\mathbf{p})$.

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 abilty to drive the training set error to zero in a few iterations is unknown.

What we do know is that arcing derives most of its power from the ability of adaptive resampling to reduce variance. This is illustrated by arc-x4--a simple algorithm made up expressly to show that the thing that makes arcing work is not the explicit form of arc-fs but the general idea of adaptive resampling--the really nice idea of focusing on those cases that are harder to classify. When arc-fs does better than bagging, its because its votes are right more often. We surmise that this is because it votes the right way on some of the hard-to-classify points that bagging votes the wrong way on.

Another complex aspect of arcing is illustrated in the experiments done to date. In the diabetes data set it 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 5 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. Its 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 missclssified, so that its probability of being sampled will continue to increase az 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 classifers.

### 7.3 Future Work

Arc-fs and other arcing algorithms function to reduce test set error on a wide variety of data sets and to improve the classification accuracy of methods like CART to the point where they are the best available off-the-shelf classifiers. 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 structure to their variance 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. Better understanding of how arcing functions will lead to further improvements.

## 8. **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 email interchanges and help in understanding the boosting context: To Trevor Hastie for making available the preprocessed US 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 .

## **References**

Because much of the work in this area is recent, some of the relevant papers are not yet published. Addresses are given where they can be obtained electronically.

Ali, K. [1995] Learning Probablistic Relational Concept Descriptions, Thesis, Computer Science, University of California, Irvine

Breiman, L. [1996a] Bagging predictors, in press, Machine Learning, ftp stat.berkeley.edu/users/pub/breiman

Breiman, L. [1996b] The heuristics of instability in model selection, in press, Annals of Statistics, ftp stat.berkeley.edu/users/pub/breiman

Breiman, L., Friedman, J., Olshen, R., and Stone, C. [1984] Classification and Regression Trees, Chapman and Hall

Dietterich, T.G. and Kong, E. B[1995] Error-Correcting Output Coding Corrects Bias and Variance, Proceedings of the 12th International Conference on Machine Learning pp. 313-321 Morgan Kaufmann. ftp://ftp.cs.orst.edu/~tgd/papers/ml95-why.ps.gz

Drucker, H. and Cortes, C. [1995] Boosting decision trees, to appear, Neural Information Processing 8, Morgan-Kaufmann, 1996 , ftp ftp.monmouth.edu /pub/drucker/nips-paper.ps.Z

Freund, Y. and Schapire, R. [1995] A decision-theoretic generalization of on-line learning and an application to boosting. http://www.research.att.com/orgs/ssr/people/yoav or http://www.research.att.com/orgs/ssr/people/schapire

Freund, Y. and Schapire, R. [1996] Experiments with a new boosting algorithm, to appear "Machine Learning: Proceedings of the Thirteenth International Conference," July, 1996.

Friedman, J. H. [1996] On Bias, Variance, 0/1-loss, and the Curse of Dimensionality

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, 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, Harvard University Aiken Computation Laboratory

Kearns, M. and Valiant, L.G.[1989] Cryptograohic Limitations on Learning Boolean Formulae and Finite Automata. Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing , ACM Press, 433-444.

Kohavi, R. and Wolpert, D.H.[1996] Bias Plus Variance Decomposition for Zero-One Loss Functions, ftp starry.stanford.edu/pub/ronnyk/biasVar.ps

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, in D. Touretzky, ed. Advances in Neural Information Processing Systems, Vol.2, Morgan Kaufman

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, to appear in the Proceedings of AAAI'96 National Conference, on Artificial Intelligence, http://www.cs.su.oz.au/~quinlan

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, in Advances in Neural Information Processing Systems, Morgan Kaufman

Tibshirani, R [1996] Bias, Variance, and Prediction Error for Classification Rules, ftp utstat.toronto.edu/pub/tibs/biasvar.ps

Vapnik, V. [1995] The Nature of Statistical Learning Theory, Springer

## Appendix on 1 Bias and Variance Definitions

In the latter part of 1995 and early 1996 there was a flurry of activity concerned with definitions of bias and variance for classifiers, some of it stimulated by the circulation of the first draft of this paper. That draft used a different definition of bias and variance which I call Definition 0.

**Definitionn 0:**

*The bias of a classifier C is*

$$\text{Bias (C)} = \text{PE}(C_A) - \text{PE}(C^*)$$

*and its variance is*

$$\text{Var (C)} = \text{PE(C)} - \text{PE}(C_A)$$

The same definition of variance was proposed earlier by Dietterich and Kong [1995]. They defined Bias(C) as PE($C_A$), thus arriving at a different decomposition of PE(C) than the one I work with. Their paper notes that the variance, as defined, could be negative. Kohavi and Wolpert [1996] criticized Definition 0, not only for the possibility that the variance could be negative, but also on the grounds that it did not assign zero variance to deterministic classifiers. They give a different definition of bias and variance. But in their definition, the bias of C* is generally positive. Tibshirani [1996] defined bias the same way as definition 0 but defined variance as P($C \neq C_A$) and explored methods for estimation of the bias and variance terms.

After considering the various suggestions and criticisms and exploring the cases in which the variance, as defined in Definition 0, was negative, I formulated the definition in Section 2. It gives the correct intuitive meaning to bias and variance and does not have the drawback of negative variance. Some additional support for it comes from Friedman[1996]. This ms. contains 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 **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 that I have defined as the variance set.

## Appendix 2 The Boosting Context of Arc-fs

Freund and Schapire [1995} 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 [1995] devised arc-fs in the context of boosting theory (see Schapire[1990]) and named it Adaboost. We follow Freund[1995] in setting out the definitions: Assume that there is an input space of vectors **x** and an unknown function Co(**x**)$\in$ {0,1} defined on the space of input vectors **x** that assigns a class label to each input vector. The problem is to "learn" Co.

A classifying method is called a <u>weaklearner</u> 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(**dx**) on input space together with the corresponding $j_n$=Co($\mathbf{x}_n$), n=1, ... ,N and the classifier C(**x**,T) constructed, then the probability of a T such that P(C(**X**,T)$\neq$Co(**X**)|T)<.5-$\varepsilon$ is greater than $\delta$, where **X** is a random vector having distribution P(**dx**).

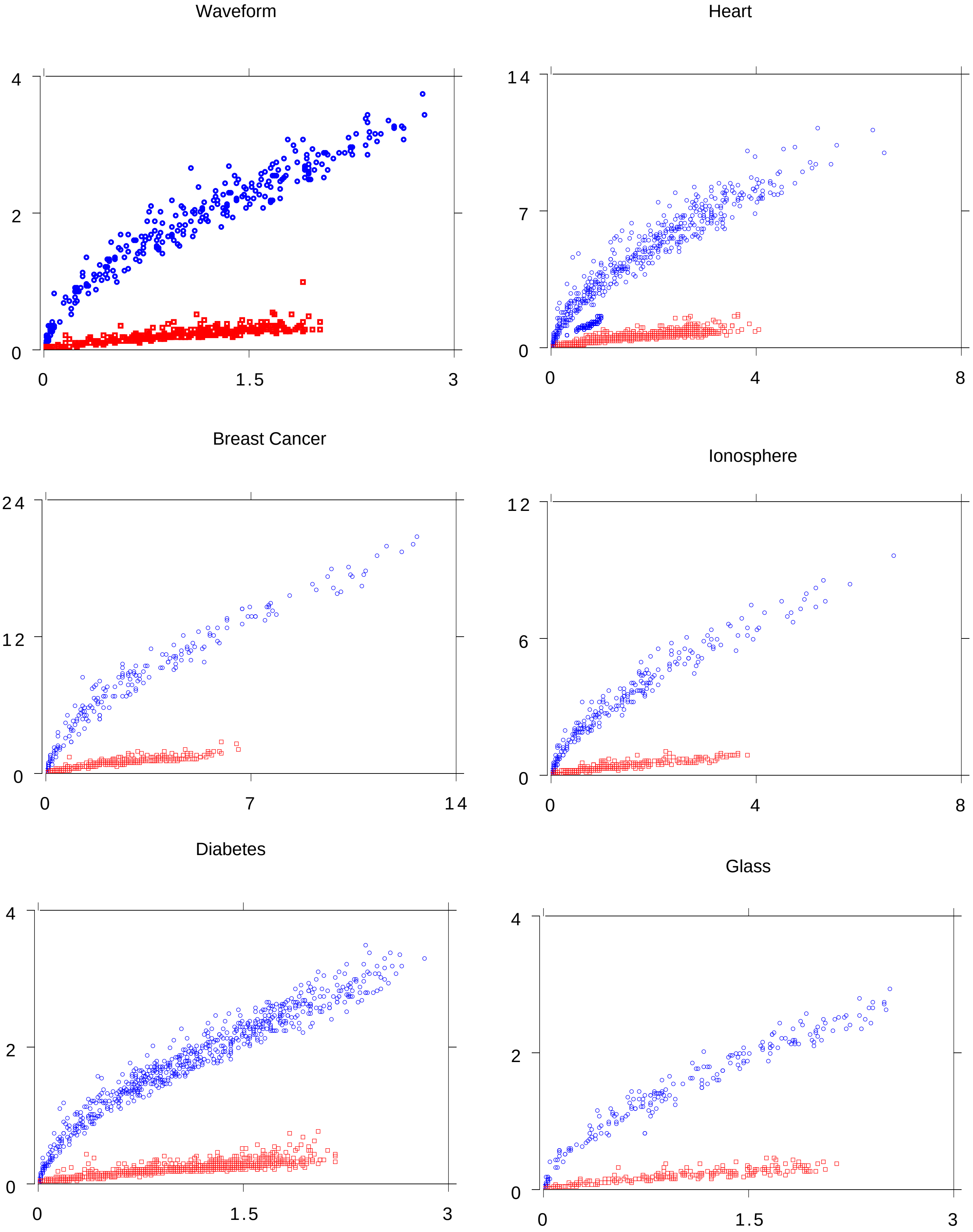
A classifying method is called a <u>stronglearner</u> if for any $\varepsilon>0$, $\delta>0$ there is an integer N such that if it is given a training set T consisting of $x_1, x_2, \ldots x_N$ drawn at random from any distribution P(**dx**) on input space together with the corresponding $j_n$=Co($\mathbf{x}_n$),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

Note that a stronglearner has low error over the whole input space, not just the training set--i.e. 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 learnabilty are equivalent. The question was termed the *boosting problem* since equivalence 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 <u>boosting algorithm</u> is a method that takes a weaklearner and converts it into a stronglearner. Freund [1995] proved that an algorithm similar to arc-fs is boosting. Freud and Schapire [1995] apply the results in Freund[1995] and conclude 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{1995} 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 zer o and their steady-state test set reduction, it is not rooted in the boosting context.

FIGURE 1  S.D. vs. Av for Resampling Probabilities



Waveform

Heart

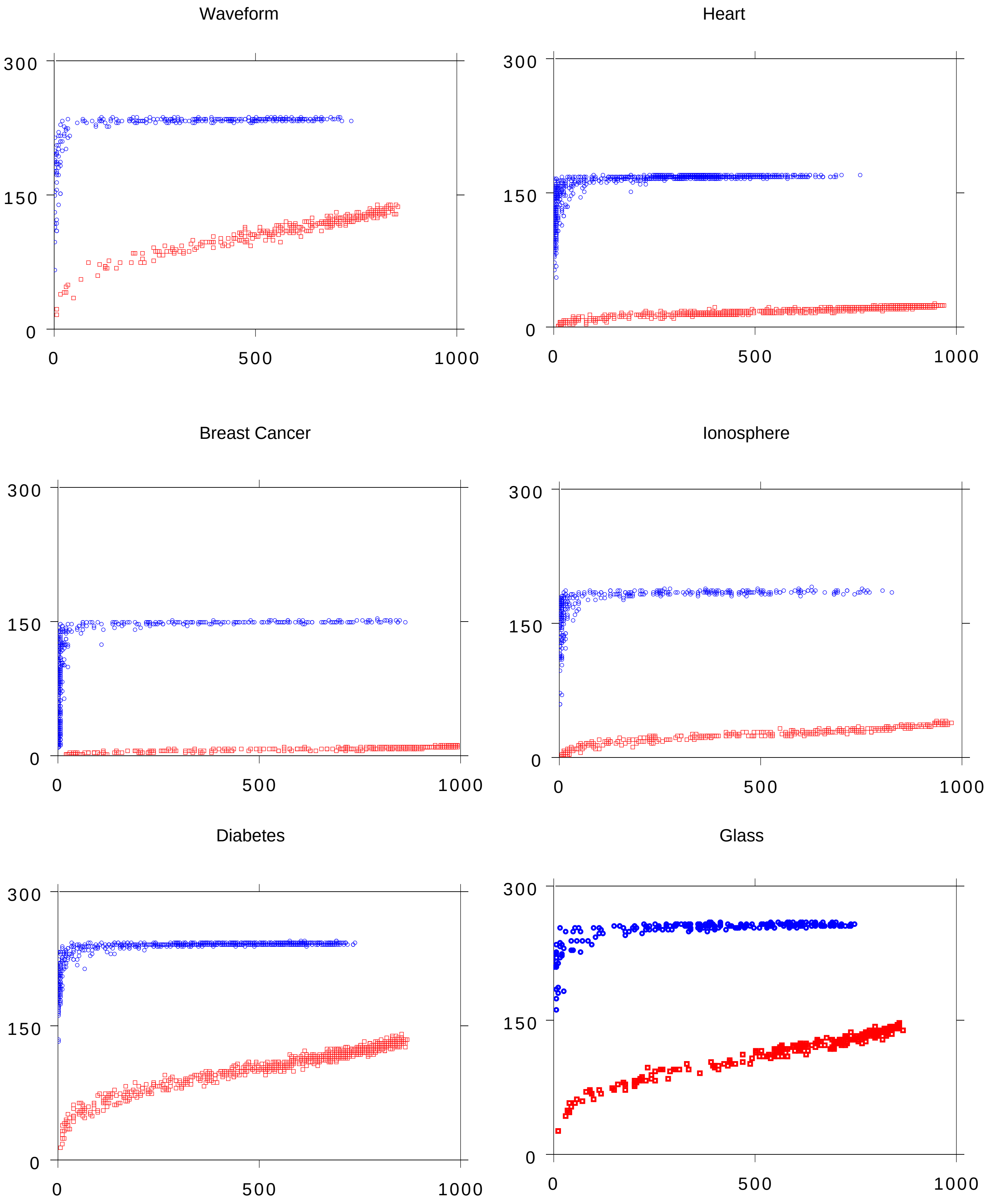
Breast Cancer

Ionosphere

Diabetes

Glass

FIGURE 2   No. of Missclassifications vs. No. Times in training Set



Waveform

Heart

Breast Cancer

Ionosphere

Diabetes

Glass

FIGURE 3  Proportion of Times Missclassified for Two Cases

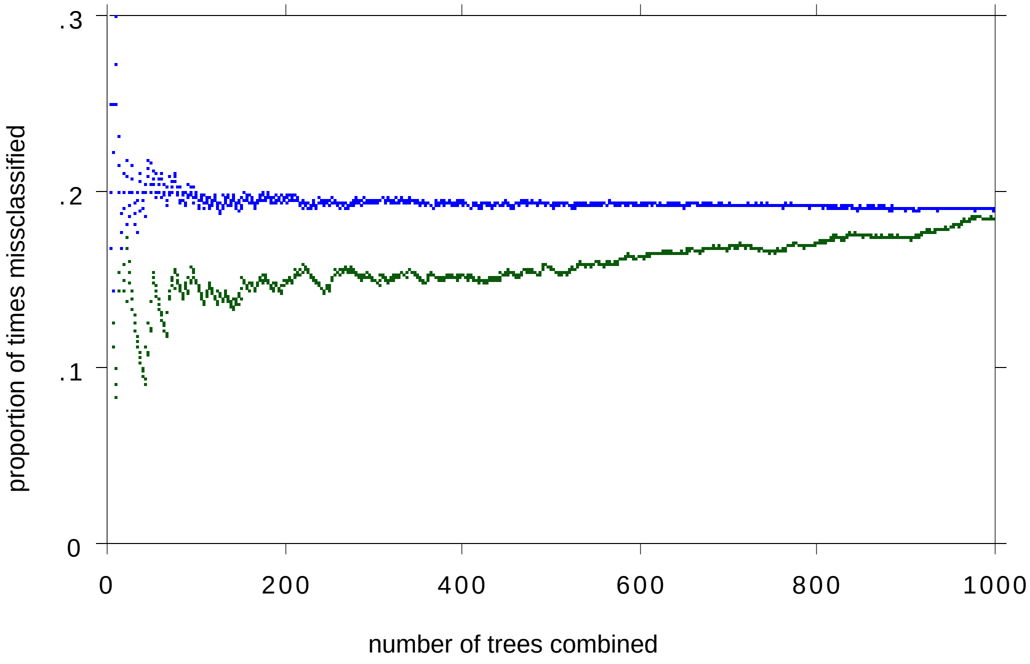


FIGURE 4  Percentile Plot--Proportion of Training Sets that Cases are In

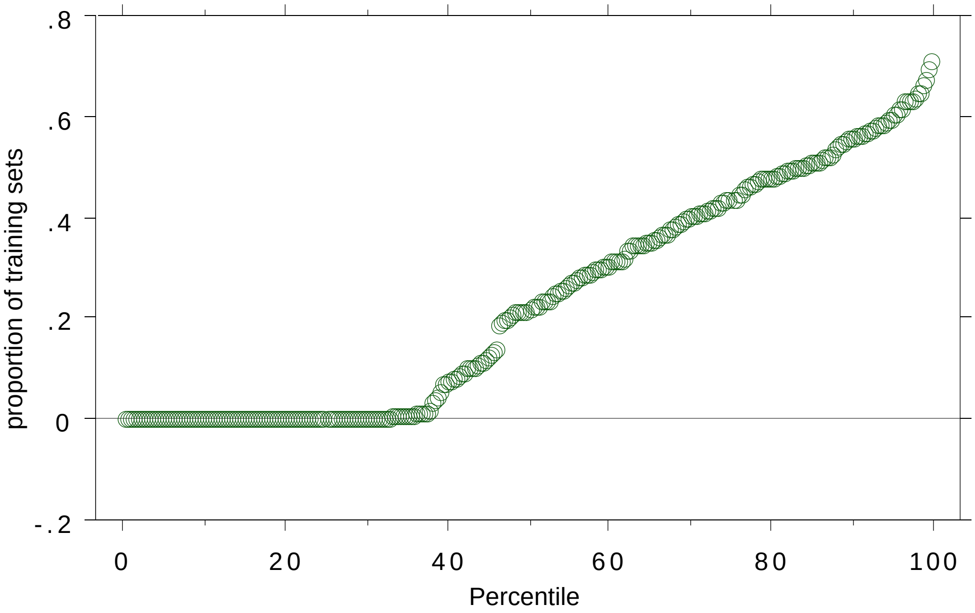
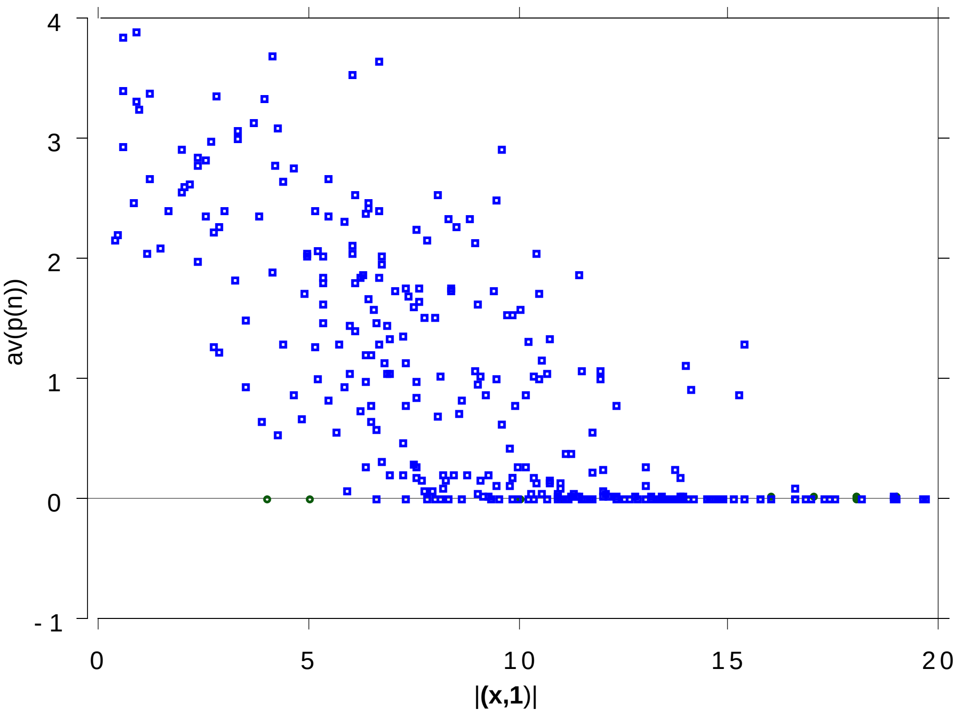
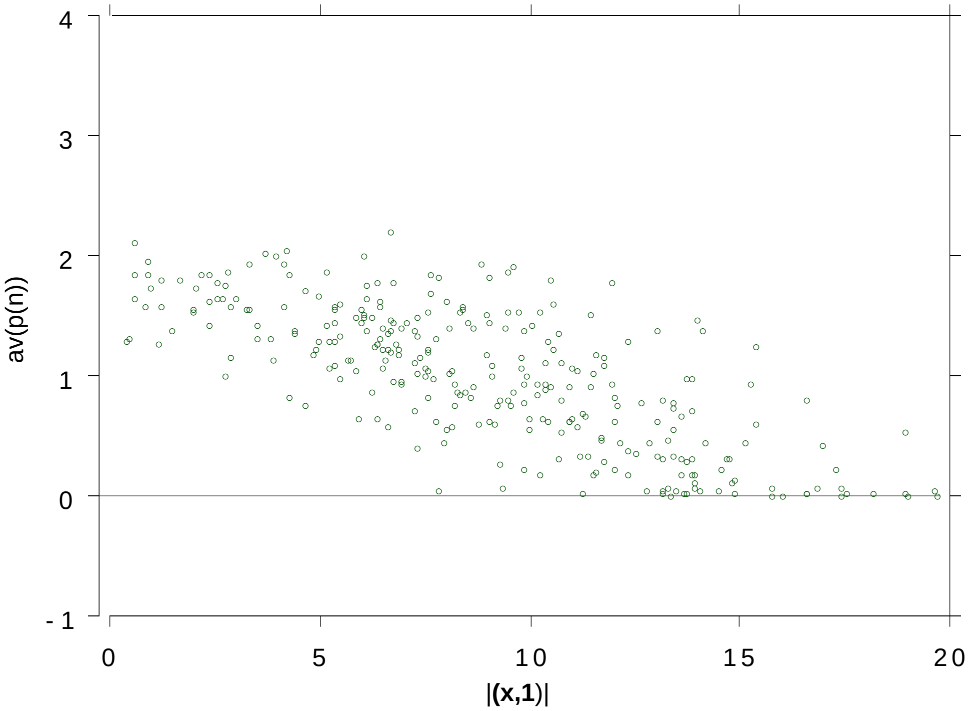


FIGURE 5  Average p(n) vs. |(**x,1**)|

ARC-FS



ARC-X4



4
3
2
1
0
- 1
av(p(n))
0
5
10
15
20
|(x,1)|