BIAS, VARIANCE , AND ARCING CLASSIFIERS

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

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 sucessful 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.

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 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 most 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 developed in the context of boosting--combining classifiers so as to drive the training set error 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.

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. 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 5 our experiments show that neither bagging nor arcing has any effect on linear discriminant error rates.

Freund and Schapire refer to their algorithm as a boosting algorithm, where boosting is a term derived from PAC learning. Their 1995 paper gives a proof that arc-fs is boosting in the PAC sense. Section 6 notes that boosting has little to do with the test set accuracy of arc-fs. In fact, if one stops the arc-fs algorithm at the point where the training error drops to zero, the test set performance markedly deteriorates. Many methods of combining classifiers are boosting i.e. lead to zero training set error. Bagging and arc-x4 are two examples. But some boosting

methods lead to higher test set errors than others. These inequities are not explainable in terms of boosting or PAC learning. In fact, the corresponding boosting algorithm for regression given in the same Freund-Schapire paper works poorly on test sets.

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] 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 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 regression come from a well-known decomposition of prediction error. Given a training set $T = \{ (y_n,\mathbf{x}_n) \;\; n=1, \ldots ,N\}$ 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 $\mathbf{X},Y$ only leaving 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:

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

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

Then we get the **Fundamental Decomposition**

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

2.2 Bias and Variance in Classification

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 missclassification error is defined as:

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

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

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

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

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

with missclassification rate

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

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

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

and define the aggregated classifier as:

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

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

The bias of a classifier C is

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

and its variance is

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

This leads to the **Fundamental Decomposition**

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

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 $C(\mathbf{x},T)$ 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.

But while classifiers such as trees or neural nets tend to have low bias, stable classifiers can sometimes have high bias. We look at bias more closely by defining:

Definition 2.2

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

That is, C(**x**,T) is unbiased at **x** if, over the replications of T, C(**x**,T) picks the right class more often than any other class. A classifier that is unbiased at **x** is not necessarily an accurate classifier. For instance, suppose that in a two class problem $P(1|\mathbf{x}) = .9$, $P(2|\mathbf{x}) = .1$, and $Q(1|\mathbf{x}) = .6$, $Q(2|\mathbf{x}) = .4$. Then C is unbiased at **x** but the probabablilty of correct classification by C is .6 x .9 + .4 x .1 = .58. But the Bayes predictor C* has probability .9 of correct classification.

If C is unbiased at **x** then $C_A(\mathbf{x})$ is optimal. Let B be the set of all **x** at which C is biased, and let

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

Then

$$\text{Bias}(C) = \int_B [\, P(j^*(\mathbf{x})|\mathbf{x}) - P(j(\mathbf{x})|\mathbf{x})]\, P(d\mathbf{x}).$$

So the bias is small if either P(B) is small or if, on the set B, $P(j^*(\mathbf{x})|\mathbf{x}) \cong P(j(\mathbf{x})|\mathbf{x})$.

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. Also, when the vote is close, but wrong, then usually $P(j^*(\mathbf{x})|\mathbf{x}) \cong P(j(\mathbf{x})|\mathbf{x})$. 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 for a few examples. These all consist of artificially generated data,, since otherwise PE(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%.

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%.

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%

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

Table 1 Bias and Variance (%)

Data Set		LDA	CART
waveform			
	bias	.8	2.3
	var	5.7	13.0
twonorm			
	bias	.0	.6
	var	.3	19.3
threenorm			
	bias	6.6	2.6
	var	1.5	14.6
ringnorm			
	bias	36.4	3.4
	var	1.7	17.1

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.

In contrast, in all examples with the exception (somewhat) of waveform, LDA has low variance. But, as seen in the ringnorm data, it can have high bias. The waveform variance is unusually high for LDA. The reason is this: estimating the within-class covariance matrix requires the estimation of 294 parameters from the 300 cases in the training set. LDA uses the inverse of this matrix, and the inverse is sensitive to small changes in the data, therefore becoming a bit unstable. To stabilize it, a small ridge was added to the matrix before inversion. The resultant bias stayed about the same. The variance decreased from 5.7 to 2.4.

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 (by definition) the same bias as $C(\mathbf{x},T)$ but zero variance. 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 β 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	2.6	1.4	0.7
	var	13.0	5.2	3.9
twonorm				
	bias	0.6	0.2	0.1
	var	19.3	5.0	2.4
threenorm				
	bias	2.6	2.5	2.5
	var	19.6	7.5	5.9
ringnorm				
	bias	3.4	1.9	1.1
	var	17.1	7.7	4.4

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. But not because it reduces bias slightly more than bagging, but 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 wavefrom 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 sections 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 ε_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 ε_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

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

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 2.

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 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. There is a small but peculiar aberration in arc-fs behavior. In the diabetes data set it gives higher error rate than a single run of CART. This also happened in two of the data sets in the Freund-Schapire[1996] experiments. This behavior does not appear in bagging.

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 were 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 was 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 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 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 such 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 are 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.

7. Arcing and boosting are not the same.

The purpose of arcing is to reduce test set error. The purpose of boosting is to reduce training set error to zero. The two have different purposes and what is good for one will not necessarily be good for the other. Boosting is a generic term for procedures that convert predictors with large training set error rates into predictors having arbitrarily small error rates. Freud and Schapire [1995] prove that for a sequence of classifiers $C_1, \ldots, C_K$ and training set T , if:

i) the kth step probabilities are {p(n)}.

ii) if $\varepsilon_k = \Sigma_n\, p(n)d(n)$, where d(n) = 1 if C_k classifies the nth case in T incorrectly and 0 otherwise and $\varepsilon_k < 1/2$.

iii) if the (k+1)st step probabilities are defined by

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

where $\beta_k = (1 - \varepsilon_k)/\varepsilon_k$

Then, the training set error is bounded above by

$$2^K \prod [\varepsilon_k (1 - \varepsilon_k)]^{1/2}$$

The Freund-Schapire proof specifies that the sequence of classifiers $C_1, \ldots, C_K$ are the arc-fs classifiers gotten by using the training sets resampled from T using the probabilities {p(n)}. But their proof holds for any sequence of classifiers, and is not specific to the arc-fs algorithm. So the justification is more general than their paper implies. In fact, many algorithms may satisfy the conditions of their proof and qualify as boosting. For instance, bagging usually does (see Table 5 below).

Freud and Schapire [1995] appeal to VC-dimension to argue that if training error is low, so is test set error. If their argument is valid, then it seems reasonable that for the same training set error, combining fewer classifiers results in a simpler classifier that should have lower VC-dimension and give lower test set error than results from combining more.

To check on the above reasoning, we run arc-fs but exiting the construction loop when the training set error became zero. The test set errors and average number of combinations to exit the loop are given in Table 5 and compared to the stop at k=50 results from Table 2. To verify our claim that bagging also is a boosting algorithm, we ran it 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 6 (soybean was not used because of restarting problems).

Table 8 Test Error(%) and Exit Times for Arc-fs as a Boosting Algorithm

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
digit			

Table 9 Test Error(%)and Exit Times for Bagging as a Boosting Algorithm

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 boosting and better test set accuracy. Arc-fs is by far the most effective booster, reaching zero training set error 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.

Clearly, if the goal is reduction of test set error, then arcing has to be run far past the point where the training error is zero. As pointed out in the bagging paper, test set error is lowered when "variance" is reduced by using the combination of a large numbers of unstable classifiers grown on resampled training sets. In fact, Table 4 shows that test set error keeps getting smaller as increasing numbers of classifiers are combined--even out to as many as 250 classifiers.

Another piece of evidence that boosting has little to do with test set error is in the regression domain. Freund and Schapire [1995] extend their proof and derive a complex boosting algorithm for regression. We programmed this algorithm for CART regression and applied it to the two real regression sets used as examples in the bagging paper, using the same procedure as for bagging (combinations of 25 regressors and 100 repetitions). The results are given in Table 7.

Table 10 Test Set Mean Squared Error

Data Set	F-S Algorithm	Bagging	CART
Ozone	20.3	18.8	23.9
Boston Housing	21.1	11.6	20.0

Here, the fact that the regression version of arc-fs is a boosting algorithm does not connect to test set error.

In one of its two faces, arc-fs is a boosting algorithm. It reduces training error quickly to zero. The operation of arc-fs as an algorithm that reduces test set error is its second face. Confusing

the two faces by referring to arc-fs as a boosting algorithm may misdirect research. For instance, boosting and the PAC theory give no insights as to why continuing arc iterations far past the point of zero training error leads to low test set error; or why other boosting algorithms, bagging for instance, have higher test set error rates than arc-fs, or why the boosting algorithm regression analogue of arc-fs does not give significant reductions in test set mean squared error. Furthermore, arc-x4, which works about as well as arc-fs, is a simple implementation of adaptive resampling and has no connection with a boosting background.

The Freund-Schapire discovery of adaptive resampling as embodied in arc-fs is a creative idea which can lead to interesting research. But its two faces need to be kept distinct and understood in their own contexts.

6. Acknowledgments

I am indebted to Yoav Freund for forwarding to me the draft papers referred to in this article and for some informative email interchanges: To Trevor Hastie for making available the preprocessed US Postal Service data: And 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.

References

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

Breiman, L. [1996a] Bagging predictors, in press, Machine Learning

Breiman, L. [1996b] The heuristics of instability in model selection, in press, Annals of Statistics

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

Drucker, H. and Cortes, C. [1995] Boosting decision trees, unpublished manuscript

Freund, Y. and Schapire, R. [1995] A decision-theoretic generalization of on-line learning and an application to boosting. unpublished manuscript

Freund, Y. and Schapire, R. [1996] Experiments with a new boosting algorithm, unpublished manuscript

Hastie, T. and Tibshirani, R. [1994] Handwritten digit recognition via deformable prototypes, unpublished manuscript

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

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

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

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

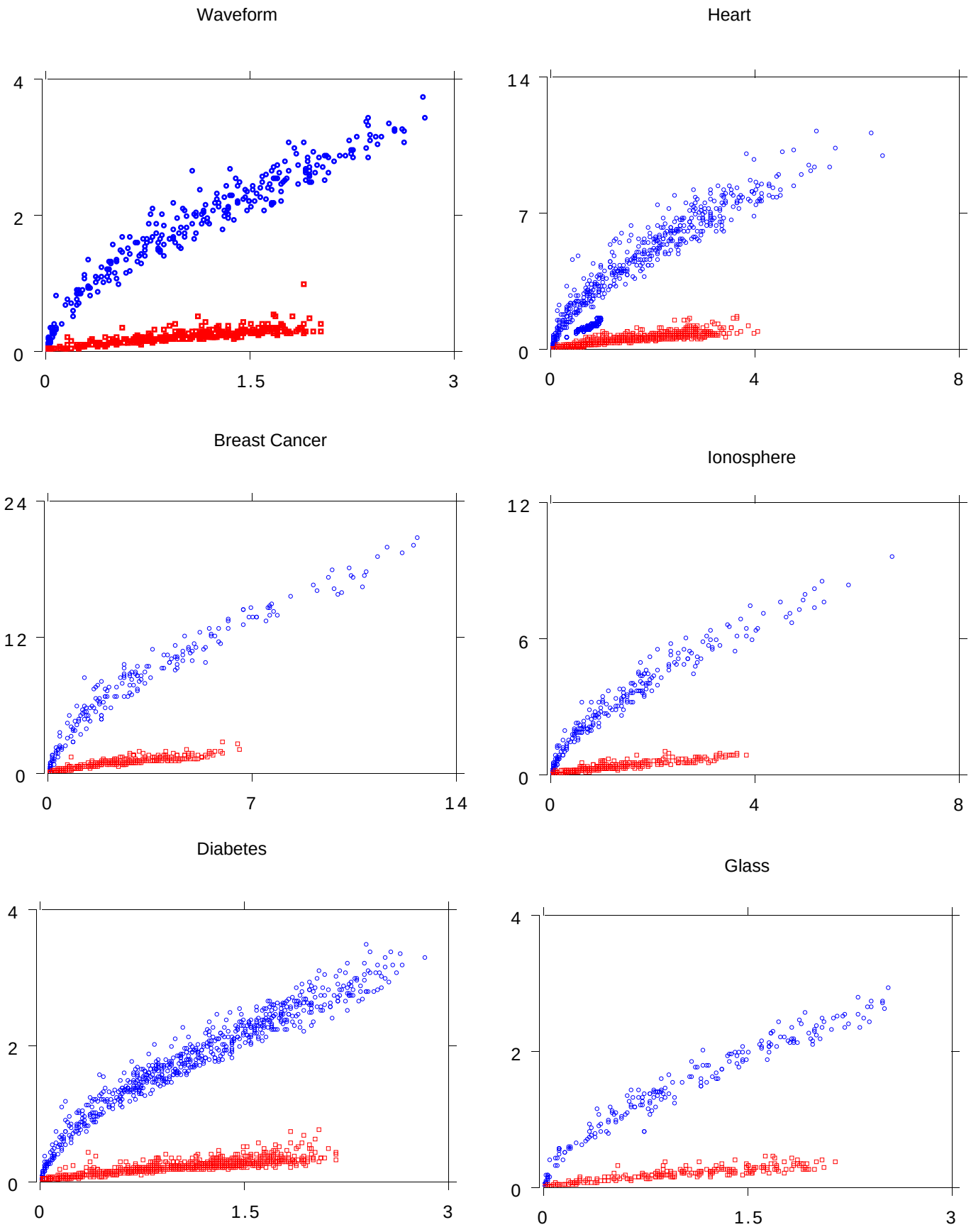

FIGURE 2 No. of Missclassifications vs. No. Times in training Set

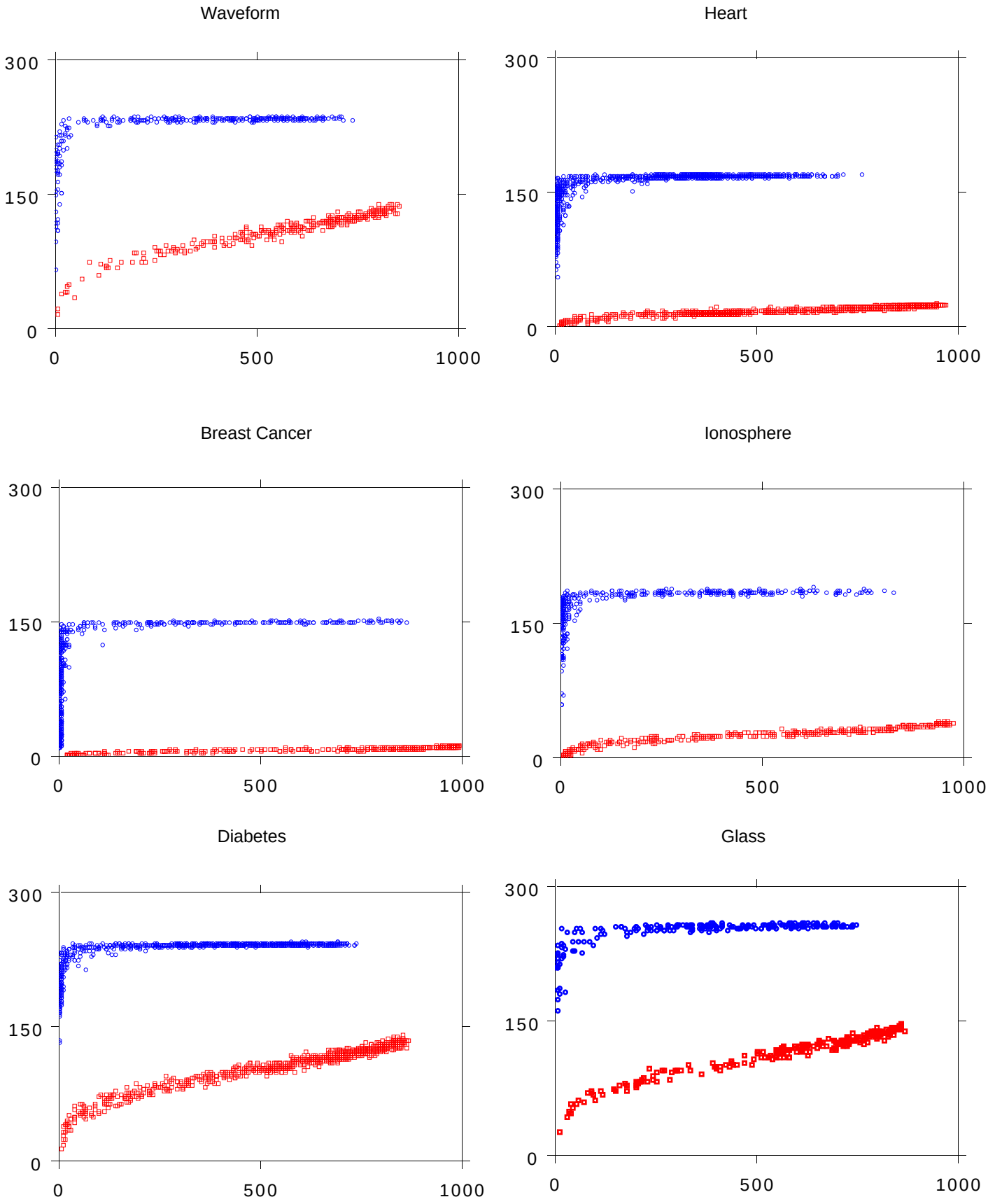

FIGURE 3 Proportion of Times Missclassified for Two Cases

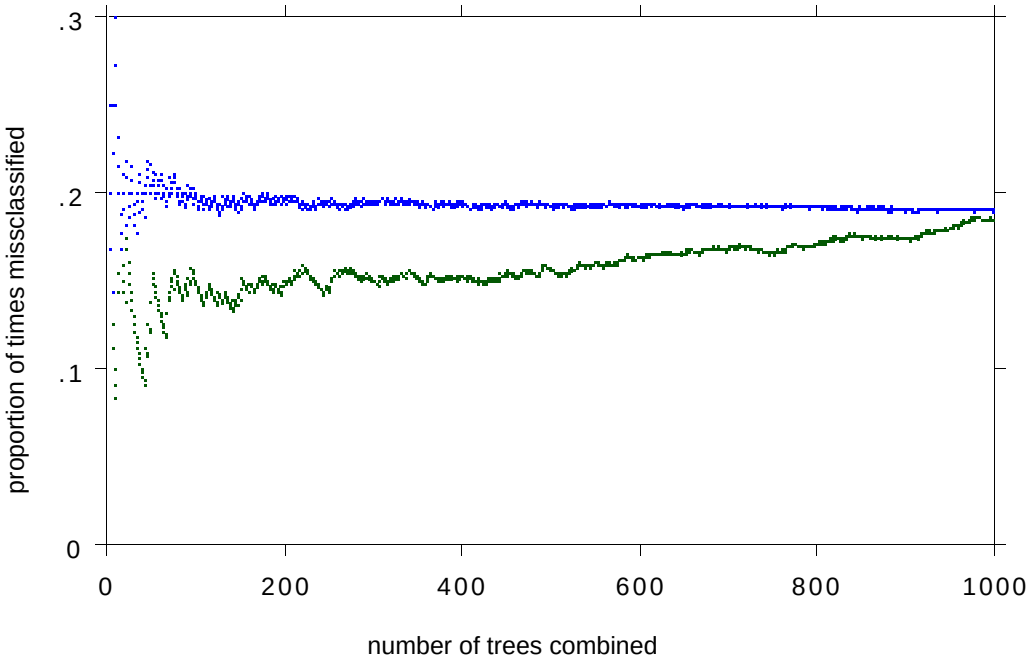

FIGURE 4 Percentile Plot--Proportion of Training Sets that Cases are In

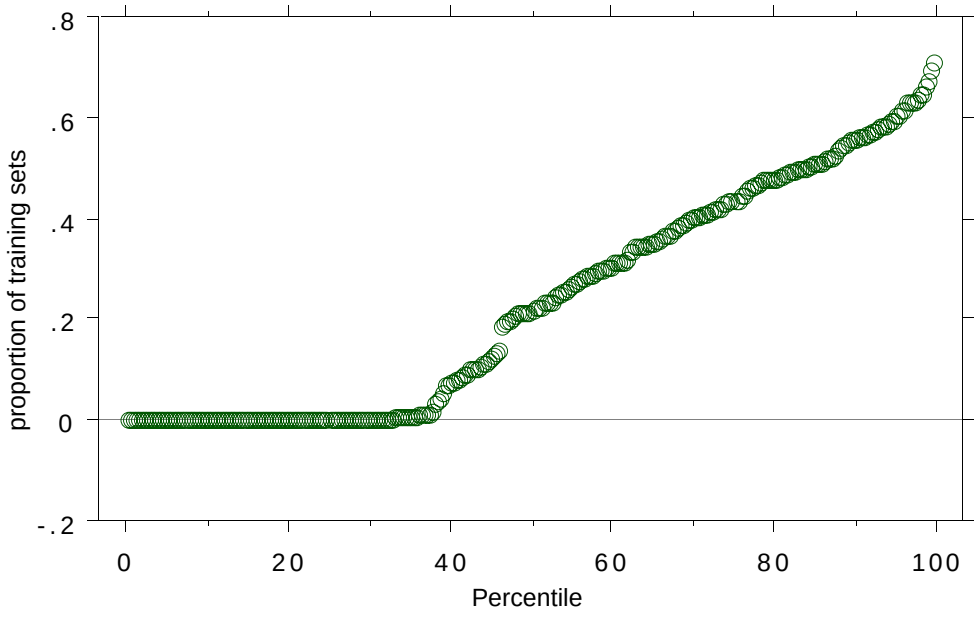

FIGURE 5 Average p(n) vs. |(**x,1**)|