Machine Learning Bias, Statistical Bias, and Statistical Variance of Decision Tree Algorithms

Thomas G. Dietterich
tgd@cs.orst.edu

Eun Bae Kong
ebkong@cs.orst.edu

Department of Computer Science
303 Dearborn Hall
Oregon State University
Corvallis, OR 97331-3202

Abstract

The term "bias" is widely used—and with different meanings—in the fields of machine learning and statistics. This paper clarifies the uses of this term and shows how to measure and visualize the statistical bias and variance of learning algorithms. Statistical bias and variance can be applied to diagnose problems with machine learning bias, and the paper shows four examples of this. Finally, the paper discusses methods of reducing bias and variance. Methods based on voting can reduce variance, and the paper compares Breiman's bagging method and our own tree randomization method for voting decision trees. Both methods uniformly improve performance on data sets from the Irvine repository. Tree randomization yields perfect performance on the Letter Recognition task. A weighted nearest neighbor algorithm based on the infinite bootstrap is also introduced. In general, decision tree algorithms have moderate-to-high variance, so an important implication of this work is that variance—rather than appropriate or inappropriate machine learning bias—is an important cause of poor performance for decision tree algorithms.

1 Introduction

In machine learning, the term "bias" was introduced by Mitchell (1980) to mean "any basis for choosing one generalization [hypothesis] over another, other than strict consistency with the observed training instances." Examples of such biases include *absolute biases* and *relative biases*. An absolute bias is an assumption by the learning algorithm that the target function to be learned is definitely a member of some designated set of functions (such as the set of linear discriminate functions or the set of boolean conjunctions). A relative bias is an assumption that the function to learned is more likely to be from one set of functions than from another. For example, the decision tree algorithms (e.g., C4.5, CART) consider small trees before they consider larger ones. If these algorithms find a small tree that can correctly classify the training data, then a larger one is not considered. The field of supervised learning has been described (Shavlik, J. and Dietterich, T.G., 1990) as the study of biases—their expressive power, their computational complexity, and their sample complexity (i.e., the number of examples required to produce accurate generalization).

Several authors have pointed out that every inductive learning algorithm must adopt a bias in order to generalize beyond the training data. Without a bias, all possible functions must be entertained as hypotheses, and, taken together, these functions predict that all possible future outcomes are equally likely, so they cannot provide a basis for generalization or prediction. The bias of a learning algorithm—if it can be formulated explicitly—provides a specification for the desired behavior of the algorithm and clarifies the design and implementation of machine learning algorithms. In the remainder of this paper, we will call this kind of bias, the *machine learning bias* or *ML bias*.

In statistics, the term "bias" is used in a more precise, but not entirely unrelated way. The bias of a learning algorithm (for a given learning problem and a fixed size m for training sets) is the persistent or systematic error that the learning algorithm is expected to make when trained on training sets of size m.

Consider a supervised learning algorithm A for learning real-valued functions. Let f be the unknown target function to be learned, where f maps from an input space $\mathbf{X}$ to the real numbers $\Re$. Let D be a probability distribution over $\mathbf{X}$ such that a random example, $x \in \mathbf{X}$, is drawn with probability $D(x)$. Let $S = \{(x, f(x) + \epsilon) | x \in \mathbf{X}\}$ be a training sample of size m drawn according to D and then labeled with the value of f, corrupted by noise ϵ. From this training sample, our algorithm A will output an hypothesis $A(S) = \hat{f}$. For a given test point x_0, the predicted value of f is $\hat{f}(x_0)$.

Statistical bias is defined as follows. Suppose we repeatedly draw training samples $S_1, \ldots, S_l$, each of size m, and apply our learning algorithm A to construct hypotheses $\hat{f}_{S_1}, \hat{f}_{S_2}, \ldots, \hat{f}_{S_l}$, where $\hat{f}_{S_i}$ denotes the hypothesis $A(S_i)$. We can combine all of these different hypotheses into an *averaged hypothesis*:

$$\overline{\hat{f}}(x) = \lim_{l \to \infty} \frac{1}{l} \sum_{i=1}^{l} \hat{f}_{S_i}(x).$$

The value $\overline{\hat{f}}(x)$ is the expected predicted value of $f(x)$, where the expectation is taken over all possible training samples of fixed size m.

The *statistical bias* of algorithm A (for sample size m at point x) is the error in this averaged hypothesis:

$$Bias(A, m, x) = \overline{\hat{f}}(x) - f(x).$$

Statistical bias captures the idea of a *systematic error* for a given sample size. For example, if the true function is a sine wave $f(x) = \sin(x)$ and the learning algorithm fits lines $\hat{f}(x) = ax + b$ as hypotheses, then there will be systematic errors at each "bump" in the sine wave.

A concept closely related to statistical bias is *variance*. Formally, the variance of an algorithm is defined as the expected value of the squared difference between any particular hypothesis $\hat{f}_S$ and the averaged hypothesis $\overline{\hat{f}}$:

$$Variance(A, m, x) = E\left[\left(\hat{f}_S - \overline{\hat{f}}(x)\right)^2\right].$$

The expectation is taken with respect to all training samples S of size m. The variance captures random variation in the algorithm from one training set S to another. This variation can result from variation in the training sample, from random noise (ϵ) in the training data, or from random behavior in the learning algorithm itself, such as the random initial weights often used in backpropagation.

It can be shown (e.g., Geman, Bienenstock, & Doursat, 1992) that the average error of learning algorithm A at point x is equal to the squared bias plus the variance:

$$Error(A, m, x) = Bias(A, m, x)^2 + Variance(A, m, x).$$

Hence, an important goal in algorithm design is to minimize statistical bias and variance and thereby minimize error.

The goal of this paper is to apply the statistical ideas of bias and variance to diagnose and repair problems in the machine learning bias of algorithms, particularly for C4.5. We begin by discussing the relationship between ML bias and statistical bias and variance. Then we develop definitions of statistical bias and variance for classification algorithms (rather than the regression algorithms discussed above). We then show one technique for measuring the statistical bias and variance of an algorithm for simulated training data. We apply this technique to measure the bias and variance of C4.5 and the nearest neighbor algorithm on various artificial problems and show examples where the error is caused primarily by bias, by variance, or by both. These examples demonstrate how statistical bias and variance can diagnose errors in the ML bias. Finally, we discuss one method—voting—for reducing the variance of C4.5 (and randomized C4.5). We evaluate voting on five domains from the UC Irvine collection.

2 The Relationship Between ML Bias and Statistical Bias and Variance

Machine learning bias, as we mentioned above, can be described in terms of absolute bias (certain hypotheses are entirely eliminated from the hypothesis space) and relative bias (certain hypotheses are preferred over others). On any particular problem, an absolute

Table 1: Relationship between ML bias and statistical bias and variance

ML Bias		Statistical	
Absolute	Relative	Bias	Variance
appropriate	too strong	high	low
appropriate	ok	low	low
appropriate	too weak	low	high
inappropriate	too strong	high	low
inappropriate	ok	high	moderate
inappropriate	too weak	high	high

bias can be characterized as appropriate or inappropriate. The hypothesis space of an inappropriate absolute bias does not contain any good approximations to the target function. An appropriate bias does contain good approximations.

A relative bias can be described as being too strong or too weak. A bias that is too strong is one that, though it may not rule out good approximations to the target function, prefers other, poorer hypotheses instead. A bias that is too weak does not focus the learning algorithm on the appropriate hypotheses but instead allows it to consider too many hypotheses.

What are the consequences of these problems in ML bias for statistical bias and variance? Table 1 shows the relationship between various properties of ML-bias and the corresponding properties of statistical bias. In general, if the relative bias of an algorithm is very strong, then the algorithm will have low variance, and if it is too weak, the algorithm will have high variance. If the ML bias is inappropriate, then the algorithm will have high statistical bias. However, the statistical bias can be high even in cases where the bias is appropriate—if the relative bias is too strong. To simultaneously achieve low bias and low variance, the algorithm must have an appropriate absolute bias and the right level of strength for the relative bias.

3 Statistical Bias and Variance for Classification Algorithms

The definitions given for statistical bias and variance in the introduction were for regression tasks. We now extend them by analogy to cover classification algorithms. Suppose that f is a function that maps from the input space $\mathbf{X}$ to a finite set of class labels $\{c_1, \ldots, c_k\}$. Given a set S of training examples, algorithm A outputs an hypothesis $A(S) = \hat{f}_S$.

It is convenient to define $\hat{p}_S(x)$ to be the probability that $\hat{f}_S$ misclassifies test point x. This probability is 1 if $\hat{f}_S$ misclassifies x and 0 otherwise.

$$\hat{p}_S(x) = \begin{cases} 1 & \text{if } \hat{f}_S(x) \neq f(x) \\ 0 & \text{if } \hat{f}_S(x) = f(x) \end{cases}$$

Now suppose that, as before, we draw a sequence of training sets $S_1, \ldots, S_l$, each of size m, and apply our learning algorithm A to construct hypotheses $\hat{f}_{S_1}, \hat{f}_{S_2}, \ldots, \hat{f}_{S_l}$. We define the *averaged probability of error* to be the average of these $\hat{p}$'s, where the average is taken over all possible training sets S:

$$\overline{\hat{p}}(A, m, x) = \lim_{l \to \infty} \frac{1}{l} \sum_{i=1}^{l} \hat{p}_{S_i}(x).$$

Intuitively, $\overline{\hat{p}}(A, m, x)$ is the probability that a hypothesis produced by algorithm A from a training set of size m will misclassify test point x. Another way of saying this is that the expected error rate of A for test point x is

$$Error(A, m, x) = \overline{\hat{p}}(A, m, x).$$

Now consider any point x such that $\overline{\hat{p}}(A, m, x) > 0.5$. For any given hypothesis $\hat{f}_S$, we expect on the average that x will be misclassified. Hence, x is a systematic error. We will therefore define the bias of any algorithm A trained on training sets of size m to be

$$Bias(A, m, x) = \begin{cases} 0 & \text{if } \overline{\hat{p}}(A, m, x) \leq 0.5 \\ 1 & \text{if } \overline{\hat{p}}(A, m, x) > 0.5 \end{cases}$$

We will define the variance of A at point x to be the difference between the error rate and the bias:

$$Variance(A, m, x) = \begin{cases} \overline{\hat{p}}(x) & \text{if } \overline{\hat{p}}(x) \leq 0.5 \\ \overline{\hat{p}}(x) - 1 & \text{if } \overline{\hat{p}}(x) > 0.5 \end{cases}$$

The variance is the increase in the error rate at x relative to the bias.

4 Measuring Bias and Variance

We can measure the bias and variance by directly simulating the definitions. Figure 1 shows the target function for a simple learning problem. There are two features, x_1 and x_2, two classes c_0 and c_1, and a simple linear decision boundary such that points above the line are in class c_0 and points below the line are in class c_1. We define the distribution D over this two-dimensional input space to be the uniform distribution on the square $0 \leq x_1 \leq 15$ and $0 \leq x_2 \leq 15$.

To measure the bias of the C4.5 algorithm, we drew 200 training sets $S_1, \ldots, S_{200}$ each containing 200 examples labeled according to this target function (call it f). We also drew a complete test set of 22,801 examples (every possible data point on a grid of points separated by 0.1 along each axis). We then approximated $\overline{\hat{p}}(A, m, x)$ by computing the probability (averaged over all 200 training sets) that each test point x will be misclassified. This is equivalent to having each of the 200 hypotheses $\hat{f}_{S_1}, \ldots, \hat{f}_{S_{200}}$ vote on the classification of each test point.

Figure 2 plots all of the test data points whose estimated value of $\overline{\hat{p}}(A, m, x) > 0.5$. These are the systematic errors. As one would expect, these errors result from C4.5's attempt to approximate the sloped decision boundary by vertical splits on feature x_1.

We can compute the mean error rate over any T-element test set by the following:

$$\frac{1}{T} \sum_{j=1}^{T} \overline{\hat{p}}(A, m, x_j),$$

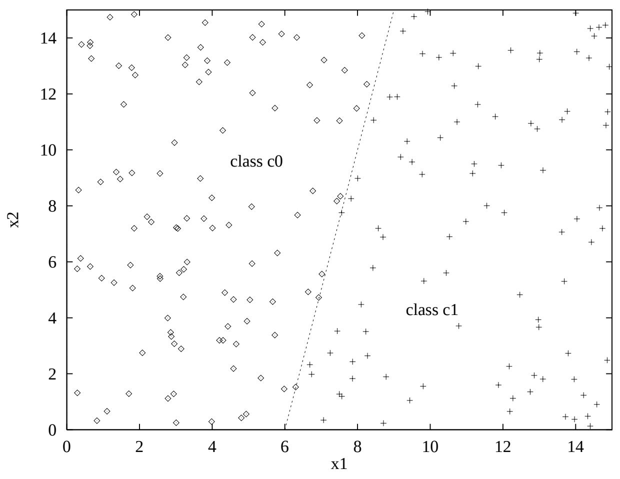

Figure 1: A two-class problem with 200 training examples.

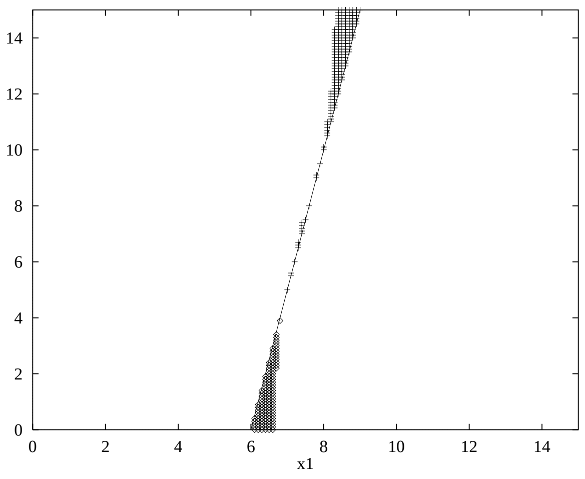

Figure 2: Bias errors of C4.5 on the problem from Figure 1.

where x_j is the j-th element in the test set. This is the average error of each of the 200 trees. We can also compute the bias as

$$\frac{1}{T}\sum_{j=1}^{T} Bias(A, m, x_j)$$

and the variance by the difference between these two. The results show that the mean error rate is 536 errors (out of the 22,801 test examples), of which 297 are the result of bias and the remaining 239 are the result of variance. This is interesting, because it shows that variance accounts for a significant portion of the errors from C4.5 on this problem.

Figure 3 shows another example. Here, there is a single linear decision-boundary whose slope is 1. Surprisingly, C4.5 has very low bias for this problem. The bias is restricted to the very ends of the line. This is a consequence of the uniform distribution over the input space: splits on x_1 and x_2 are equally likely and, when averaged over all 200 training sets, they compensate for each other to give a good approximation to the diagonal decision boundary. The mean error for this problem is 821 errors, but the bias is only 177, so most of the errors result from the variance of 644 errors.

Let us now consider four cases that illustrate the ability of statistical bias and variance to diagnose problems with ML bias. The four cases will illustrate all four combinations where the statistical bias is either high or low and the variance is either high or low.

We begin with a situation where the statistical bias and variance are both high. Figure 4 shows an example six-class problem on which we have performed extensive experiments. The figure shows the persistent errors of C4.5 when training on training sets of size 200. The statistical bias of this algorithm is 1788 and the variance is 1046. As predicted in Table 1, the ML bias of C4.5 is inappropriate and too weak. It is inappropriate, because none of the decision boundaries are axis-parallel, and it is too weak because the only good approximations to the decision boundaries are very large decision trees, and the ML bias provides little guidance for preferring one of these over another.

To check the robustness of these observations, we computed a learning curve for C4.5 on this problem which shows the mean error, statistical bias, and variance of C4.5 as a function of the size of the training sample (see Figure 5). Note that as the training sample gets larger,

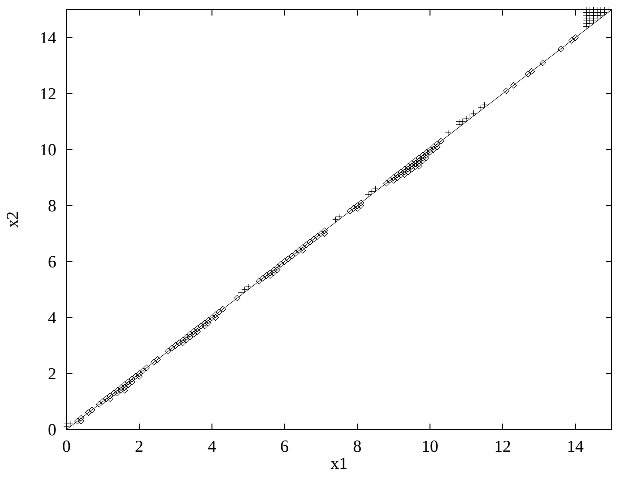

Figure 3: Bias errors of C4.5 on a diagonal decision boundary problem.

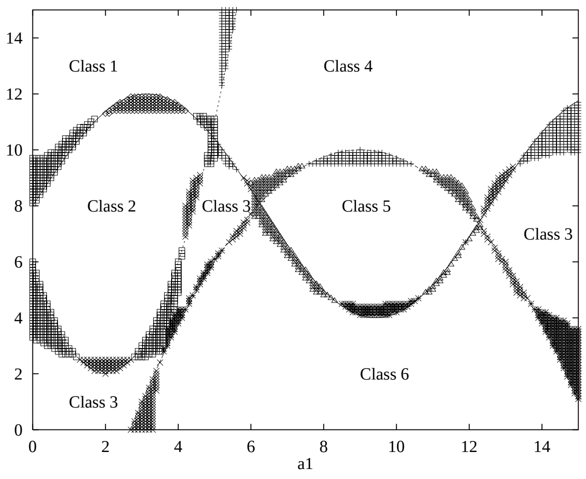

Figure 4: Bias errors of C4.5 for a six-class problem.

the bias of C4.5 decreases but the variance remains large. This results from the fact that as the sample expands, C4.5 is willing to grow larger and larger decision trees (which have smaller statistical bias). However, there are still a very large number of alternative trees to consider, so the variance does not decrease.

The second case we will consider arises when we apply C4.5 to this same problem but we require it to place at least 25 training examples in each leaf—that is, we prevent it from growing large decision trees, since it is permitted to partition the training sample into at most eight regions of size 25 or more. Under these conditions, the mean error jumps to 4476, the statistical bias to 4281, and the variance drops to a mere 195 points. Table 1 predicts that a high bias/low variance case is caused by an ML bias that is too strong (regardless of whether it is appropriate or inappropriate). In this particular case, the ML bias is inappropriate and much too strong.

The third case we will consider is a low bias/high variance case. A learning algorithm with low bias is the nearest neighbor algorithm. When we applied it to the problem from Figure 4, its mean error rate was measured as 2074 of which only 377 errors were due to bias and the remaining 1697 to variance. Table 1 predicts from these observations that the algorithm has an appropriate but very weak ML bias. The ML bias of the nearest neighbor algorithm is appropriate—it can easily represent decision boundaries of any shape—but it is weak. The weakness is caused by the highly local nature of the algorithm. A slight change in the position of a single training example can cause a shift in the position of the decision boundary. Slight changes throughout the training set can work combinatorially to generate a huge number of slightly different alternative hypotheses.

It is interesting to compare the learning curve of nearest neighbor (shown in Figure 6) with the learning curve of C4.5 (Figure 5). As the training sample enlarges, the statistical bias of the nearest neighbor algorithm approaches zero. The variance decreases as well, but it accounts for an even larger fraction of the total error.

The fourth case we will consider is the low bias/low variance case shown in Figure 7. The decision boundaries in this problem can be exactly represented by an axis-parallel decision tree. The mean error on this problem is only 17, with all 17 errors resulting from variance in the algorithm; the bias is zero. According to Table 1, the low bias/low variance case can

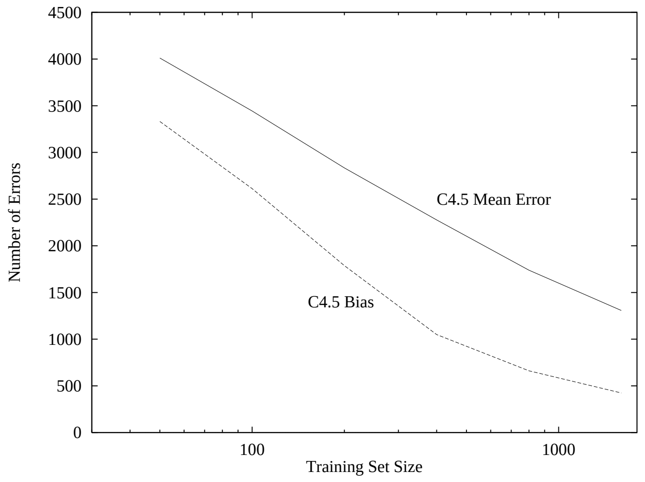

Figure 5: Learning curve for C4.5 on the problem from Figure 4. The upper curve is the average performance of C4.5; the lower curve shows the persistent errors of C4.5. The difference between these two curves is the variance. Note that the horizontal scale is logarithmic.

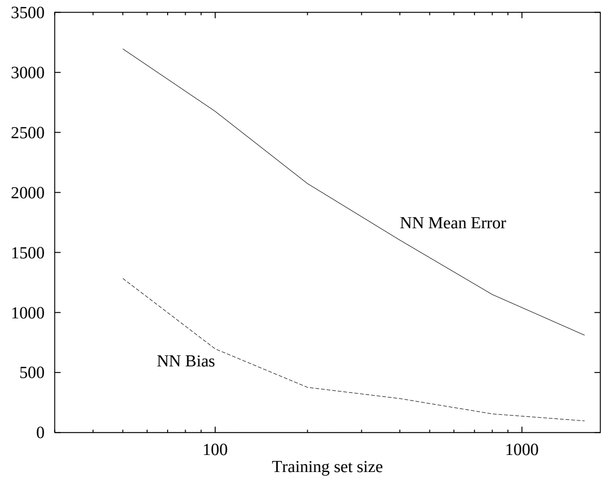

Figure 6: Learning curve for the nearest neighbor algorithm on the program from Figure 4.

only be caused by an appropriate absolute bias and a moderate preference bias. This is indeed the correct diagnosis for this problem.

5 Reducing Bias and Variance

Because statistical bias and variance result from the basic design of a learning algorithm, any change to the algorithm can change the bias and variance. Any change that increases the representational power of an algorithm can reduce its statistical (and ML) bias. Any change that expands the set of available alternatives for an algorithm or makes them depend on a smaller fraction of the training data can increase the variance of the algorithm.

In particular, one important source of variance in C4.5 is the fact that the algorithm must choose a single split at each node in the tree. With most splitting criteria (e.g., information gain, gain ratio, or GINI index), the choice of split can be altered by the addition or removal of a single training example. Furthermore, the subsequent splits at the descendent nodes in a tree are influenced by the split at the root, so that a change of one training example can produce a cascade of changes in subsequent splits and alter the entire tree. This is a plausible cause of the variance of C4.5.

This analysis suggests that the variance of C4.5 could be reduced by making the choice of splits more robust to slight changes in the training set or by making the splits "softer." The C4.5 system has a facility for softening splits such that test examples near the split threshold of a continuous feature are routed down both sides of the split. The class probability estimates resulting from those subtrees are then combined by weighted sum (where the weights depend on how close the example was to the split threshold). We ran this configuration of C4.5 on the program from Figure 4, and it resulted in a decrease in the mean error (from 2834 errors to 2740), a slight increase in the statistical bias (from 1788 to 1851) and a sig-

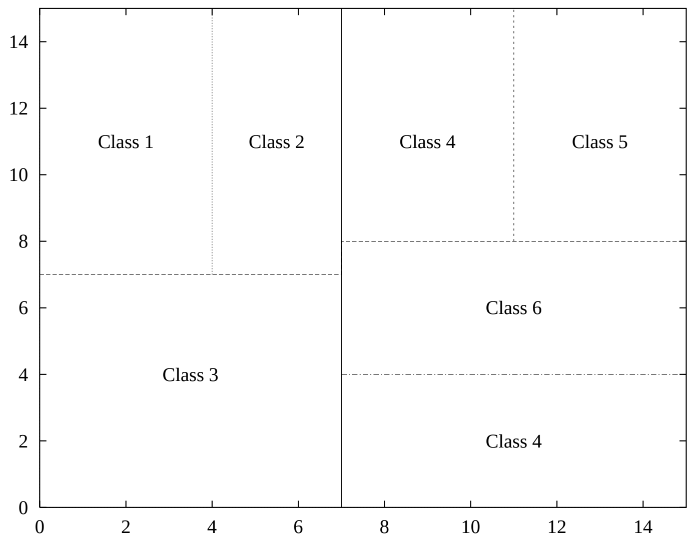

Figure 7: A simple learning problem for which the correct function is a decision tree with axis-parallel splits.

nificant decrease in the variance (from 1046 to 889). Further improvements can probably be made here. We predict that the Markov Tree models of Jordan et al (1991, 1994), which are soft, stochastic versions of decision trees, would have much lower variance than C4.5. We also predict that other learning systems, such as RL (Clearwater & Provost, 1990), which learn collections of independent rules, will have lower variance also, because they reduce the cascading of decisions that results from the top-down construction of decision trees.

Many authors have suggested that an important source of variance in decision tree algorithms is the choice of splits and classifications for the leaves of the tree. These choices are based on a very small number of training examples, so one would expect them to have high variance. The conventional remedy for this is to prune the tree to remove high-variance leaves. We tested this remedy using the pessimistic pruning procedure of C4.5 (with pruning confidence level 0.10) on the problem from Figure 4. We found that this had almost no effect on the variance (reducing it from 1046 errors to 1015), but that it increased the bias (from 1788 to 1927) and, hence, increased the mean error from 2834 to 2942. From this and many other test problems, we conclude that pruning does not necessarily work the way its advocates claim. In our experience, it rarely produces improvement in classification accuracy (Dietterich & Bakiri, 1995).

A very general technique for reducing variance is to construct a set of hypotheses and then have them vote on the classification of test cases. For example, if a relative ML bias does not have a strong preference for one hypothesis over another, the two hypotheses could both be generated and then voted. There are many different ways of producing and voting the hypotheses, and this is a very active topic of research, particularly in the neural network community (Perrone, 1993; Perrone & Cooper, 1993; Perrone, 1994). There are strong Bayesian justifications for voting as well (Buntine, 1990).

We explored two methods for generating multiple hypotheses. The first is bootstrapping (Efron & Tibshirani, 1993; Breiman, 1994). Many equally-plausible decision trees can be constructed by the following procedure. Let S be the available training set of size m. We can draw a *bootstrap replicate training set* by drawing a set of examples S^1 of size m by sampling at random with replacement from S. We then apply C4.5 to this bootstrap replicate data set to obtain a bootstrap replicate hypothesis, $\hat{f}^1$. In our experiments, we constructed 200 bootstrap replicate decision trees. To classify a new example x, we voted these trees by summing their class probability vectors component-wise and then choosing the class with the highest probability. This relies on the ability of C4.5 to estimate the probability that x belongs to each of the k classes, $c_1, \ldots, c_k$. (This is in fact how we compute our estimates of $\overline{\hat{p}}$ throughout this paper.)

Breiman (1994) calls this procedure *bagging* (for bootstrap aggregating). He shows that is produces very significant improvements in performance for the CART algorithm applied to several real-world data sets. When we followed this procedure with C4.5 on the problem from Figure 4, the mean error rate dropped from 2834 to 2648, the bias increased quite a bit from 1788 to 2067, but the variance was reduced by almost half from 1046 down to 581. Hence, in this problem at least, bootstrap aggregation trades a slight increase in bias for a major decrease in variance to yield a significant improvement in performance.

The second procedure that we explored for generating alternative hypotheses was randomization, which was first introduced in a simple form by Kwok and Carter (1990). We modified C4.5 so that at each node in the decision tree, it computes the 20 best splits according to its gain ratio criterion and then chooses randomly among them to select the attribute and value to split on. As with bootstrapping, we constructed 200 trees in this fashion and voted them by summing their class probability vectors component-wise. The result was that the mean error rate dropped from 2834 to 2627, which is almost exactly the same reduction obtained from bootstrapping (2648). However, unlike bootstrapping, with randomized trees, the bias remains virtually unchanged (1788 for a single tree versus 1772 for 200 random trees), while the variance decreases from 1046 to 855.

Randomization is paradoxical, because at first glance it seems to increase variance by deliberately introducing variation into the splits in the decision tree. However, it can also be seen as a way of smoothing or averaging out the effects of several equally-good splits by sampling them all and then voting them.

We tested these two voting methods (bootstrap and randomization) on five interesting problems: the vowel, soybean, letter recognition, and NETtalk tasks from the Irvine repository, and a part-of-speech task which was made available to us by C. Cardie (personal communication). The results are summarized in Table 2. Bootstrap voting improves performance on all five problems. The measured improvement in the accuracy of the hypotheses produced by the algorithms is significant for the bootstrap in the NETtalk and letter recognition task and for randomized C4.5 in the vowel, NETtalk, and letter recognition tasks. The letter recognition task is particularly astonishing. Bootstrap voting reduces the error from more than 20% to less than 1%, and randomized voting classifies the test set with perfect accuracy!

Bootstrap voting can also be applied to the nearest neighbor algorithm. As with many interesting variations of the nearest neighbor procedure, bootstrap voting has an efficient closed-form solution. Instead of repeatedly drawing 200 training sets and applying the nearest neighbor algorithm to each of those sets, we can compute the probability that the k-th nearest neighbor of a test point will be the point that "survives" the bootstrap subsampling

Table 2: Results on Five Domains (best error rate in **boldface**)

Task	Test set size	C4.5	200-fold bootstrap C4.5	200-fold random C4.5
Vowel	462	0.5758	0.5152	**0.4870***
Soybean	376	0.1090	**0.0984**	0.1090
Part-of-Speech	3060	0.0827	**0.0765**	0.0788[a]
NETtalk	7242	0.3000	0.2670***	**0.2500*****
Letter Recognition	4000	0.2010	0.0038***	**0.0000*****

Difference from C4.5 significant at $p < 0.05^{*}$, 0.001^{***}. [a]256-fold random.

to be the nearest neighbor of that test point. Suppose the training set contains m elements. When a bootstrap sample is drawn by random sampling with replacement, a given point will be missing from the bootstrap sample with probability $(1-\frac{1}{m})^m$, since it has a probability of $\frac{1}{m}$ of being selected on each draw. Hence, for a given test point x, its first nearest neighbor in the full data set will also be its first nearest neighbor in the bootstrap data set with probability $1-(1-\frac{1}{m})^m$. As m gets large, this is approaches $1-e^{-1}$. In general, the k-th nearest neighbor of x in the full training set will be the first nearest neighbor in the bootstrap data set with probability $e^{-(k-1)} - e^{-k}$. Hence, we can implement ideal bootstrapping as a weighted k-nearest neighbor classifier where we compute the seven nearest neighbors and then weighting their classes appropriately (the weight of the seventh nearest neighbor will be 0.0016 according to this formula, so no additional neighbors are needed).

For the problem in Figure 4, this modification of the nearest neighbor algorithm has the desired effect: the variance has decreased (from 1697 errors to 1538). However, it turns out the bias has increased by exactly the same amount (from 377 to 536), so the mean error remains unchanged.

Now that we have considered voting as a general technique for variance reduction, we briefly describe one technique for bias reduction: error-correcting output coding (ECOC). For a k-class problem where $k > 5$, it is possible to construct a large number of 2-class problems whose solution can be converted into a solution to the k-class problem. The construction is based on error-correcting codes. Each class is assigned a codeword in an error-correcting code, and each bit position of the codeword then defines a 2-class learning problem, which can be solved by applying such algorithms as C4.5. To classify a new example, each of the "bit-position hypotheses" is evaluated to produce a binary string of 2-class decisions. This string is then mapped to the nearest legal codeword (in Hamming distance) to classify the example. See Dietterich and Bakiri (1991, 1995) for more details.

A companion paper (Kong & Dietterich, 1995) shows that the bias errors made in each of the bit-position hypotheses can be substantially uncorrelated, so that the error-correction procedure can correct for bias errors in the algorithm. We applied this procedure to the 6-class learning problem from Figure 4 with the following results. Compared to standard C4.5, the bias is reduced from 1788 errors to 1669. The variance is also slightly reduced from 1046 to 977. So that the overall mean error is reduced from 2834 to 2646. This is essentially the same improvement that was obtained from the two variance reduction methods, but it was accomplished through bias reduction instead.

This suggests that we should combine bias and variance reduction. We did this by using

Table 3: Results of Experiments on the Problem From Figure 4.

Configuration	Mean Error	Bias	Variance
C4.5 multiclass	2834	1788	1046
C4.5 multiclass pruned	2942	1927	1015
C4.5 multiclass softened	2740	1851	889
C4.5 multiclass `-m 25`	4476	4281	195
C4.5 200-fold bootstrap	2648	2067	581
C4.5 200-fold random	2627	1772	855
C4.5 32-bit ECOC	2646	1669	977
C4.5 32-bit ECOC + 200-fold bootstrap	2407	1562	845
NN	2074	377	1697
NN infinite bootstrap	2074	536	1538

the 200-fold bootstrap to learn each bit-position hypothesis in the error-correcting code. The result was simultaneous and substantial decreases in bias (from 1788 to 1562) and variance (from 1046 to 845) so that the mean error rate dropped from 2834 to 2407.

The results of all of our experiments are summarized in Table 3.

6 Concluding Remarks

This paper has attempted to clarify the relationship between machine learning bias and statistical bias (and variance). We began with a formal definition of statistical bias and variance for classification algorithms. Then, we developed an experimental procedure for measuring bias and variance and applied it to several synthetic learning tasks. We showed that statistical bias and variance can diagnose problems in machine learning bias (such as overly strong, overly weak, or inappropriate biases). Finally, we discussed some algorithm-specific and algorithm-independent methods for reducing variance and bias. The variance-reduction technique of voting was tested in five real-word domains and shown to improve performance uniformly—sometimes modestly and sometimes dramatically. Finally, we showed that variance-reduction and bias reduction could be combined to produce excellent error reductions.

There are many additional implications of the bias/variance perspective on learning algorithms. For example, virtually none of the work in computational learning theory has addressed the bias/variance tradeoff and the nature of bias and variance in predicting the effectiveness of algorithms. For example, the boosting (Schapire, 1990) algorithm appears to be a variance-reduction algorithm, while the weighted majority algorithm (Littlestone & Warmuth, 1989) could provide both variance and bias reduction. Can we develop a theory of variance reduction that can predict when it will succeed? Can we develop a theory of bias reduction and understand its capabilities and limitations?

Another consequence of this perspective concerns the statistical evaluation of machine learning algorithms. If an algorithm has high variance, then it is essential to run that algorithm multiple times so that the variance can be measured. Traditional tests based on the t statistic do not directly measure the variance (as it is defined in this paper), nor do they take into consideration the size of the test set. Much work remains to be done.

A final practical issue with variance reduction and bias reduction is that large scale voting and error-correcting codes require huge amounts of memory and CPU time to classify each test example. For instance, with 200-fold bootstrap replications on each of 32 bits in the error-correcting code for the 6-class problem of Figure 4, we must evaluate 6,400 decision trees to classify each example. To measure the variance and bias of this procedure, we replicated it 200 times, for a total of 1,280,000 decision trees. Some method is needed for converting a combination of trees (or other complex hypotheses) into a smaller, equivalent hypothesis. These trees are very redundant; how can we remove this redundancy while still reducing bias and variance?

7 Acknowledgements

The authors gratefully acknowledge the support of the NSF under grants IRI-9204129 and CDA-9216172. The authors also thank Leo Breiman and David Cohn for very helpful discussions.

References

Breiman, L. (1994). Bagging predictors. Tech. rep. 421, Department of Statistics, University of California, Berkeley, CA.

Buntine, W. L. (1990). *A Theory of Learning Classification Rules*. Ph.D. thesis, University of Technology, Sydney.

Clearwater, S., & Provost, F. (1990). RL4: A tool for knowledge-based induction. In *Proceedings of the Second International IEEE Conference on Tools for Artificial Intelligence*, pp. 24–30. IEEE Computer Society Press.

Dietterich, T. G., & Bakiri, G. (1991). Error-correcting output codes: A general method for improving multiclass inductive learning programs. In *Proc. of the Ninth National Conference on Artificial Intelligence*, pp. 572–577. AAAI Press/MIT Press.

Dietterich, T. G., & Bakiri, G. (1995). Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, *2*, 263–286.

Efron, B., & Tibshirani, R. J. (1993). *An Introduction to the Bootstrap*. Chapman and Hall, New York, NY.

Geman, S., Bienenstock, E., & Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural Computation*, *4*(1), 1–58.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., & Hinton, G. E. (1991). Adaptive mixtures of local experts. *Neural Computation*, *3*(1), 79–87.

Jordan, M. I. (1994). A statistical approach to decision tree modeling. In *Proc. 7th Annu. ACM Workshop on Comput. Learning Theory*, pp. 13–20. ACM Press, New York, NY.

Kong, E. B., & Dietterich, T. G. (1995). Error-correcting output coding works by correcting bias and variance. In *Submitted to the International Conference on Machine Learning.*

Kwok, S. W., & Carter, C. (1990). Multiple decision trees. In Schachter, R. D., Levitt, T. S., Kannal, L. N., & Lemmer, J. F. (Eds.), *Uncertainty in Artificial Intelligence 4*, pp. 327–335. Elsevier Science, Amsterdam.

Littlestone, N., & Warmuth, M. K. (1989). The weighted majority algorithm. In *Proc. 30th Annu. IEEE Sympos. Found. Comput. Sci.*, pp. 256–261. IEEE Computer Society Press, Los Alamitos, CA.

Mitchell, T. M. (1980). The need for biases in learning generalizations. Tech. rep. CBM-TR-117, Rutgers University, New Brunswick, NJ.

Perrone, M. P. (1993). *Improving regression estimation: Averaging methods for variance reduction with extensions to general convex measure optimization.* Ph.D. thesis, Brown University, Institute for Brain and Neural Systems.

Perrone, M. P. (1994). Putting it all together: Methods for combining neural networks. In Cowan, J. D., Tesauro, G., & Alspector, J. (Eds.), *Advances in Neural Information Processing Systems*, Vol. 6, pp. 1188–1189. Morgan Kaufmann, San Francisco, CA.

Perrone, M. P., & Cooper, L. N. (1993). When networks disagree: Ensemble methods for hybrid neural networks. In Mammone, R. J. (Ed.), *Neural networks for speech and image processing.* Chapman and Hall.

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning*, *5*(2), 197–227.

Shavlik, J. and Dietterich, T.G. (1990). *Readings in Machine Learning.* Morgan Kaufmann, San Mateo, CA.