The Racing Algorithm: Model Selection for Lazy Learners

Oded Maron
MIT AI Lab
NE45-755, 545 Technology Square
Cambridge, MA 02139
oded@ai.mit.edu

Andrew W. Moore
Carnegie Mellon University
5000 Forbes Ave
Pittsburgh, PA 15213
awm@cs.cmu.edu

Abstract

Given a set of models and some training data, we would like to find the model that best describes the data. Finding the model with the lowest generalization error is a computationally expensive process, especially if the number of testing points is high or if the number of models is large. Optimization techniques such as hill climbing or genetic algorithms are helpful but can end up with a model that is arbitrarily worse than the best one or cannot be used because there is no distance metric on the space of discrete models. In this paper we develop a technique called "racing" that tests the set of models in parallel, quickly discards those models that are clearly inferior and concentrates the computational effort on differentiating among the better models. Racing is especially suitable for selecting among lazy learners since training requires negligible expense, and incremental testing using leave-one-out cross validation is efficient. We use racing to select among various lazy learning algorithms and to find relevant features in applications ranging from robot juggling to lesion detection in MRI scans.

1 Introduction

The problem of model selection can be thought of as trying to find the best student in a classroom. The teacher has the task of not only teaching the students some subject, but also of testing them to find out which student has best learned the subject. This problem, in different scaling, turns up repeatedly in the Machine Learning community: which neural net architecture gives the best generalization error? What should the value of k be in a k-nearest-neighbor algorithm? Should I use a decision tree, a neural net, a lazy learner, or a random guesser? Which of the 50 features are relevant for this problem? The answer to these questions changes depending on the training set. We would like to find an answer efficiently, given a training set and a collection of models.

There are a number of popular ways to determine a student's grade, or a model's generalization error [Weiss and Kulikowski, 1991]. In this paper we will use leave-one-out cross validation. The model is trained on all points but one, and is then queried for the value at that point. This is repeated for every point in the training set and the prediction errors are averaged to give an estimate of the generalization error. Other methods are discussed below.

The idea behind racing [Maron and Moore, 1994] is to test the various models in parallel, one test point at a time. This way, a running average can be maintained for each model's error. This is an estimate of the model's true error had it been tested on all of the test points. Using statistical bounds we can also determine how close the estimated error is to the true error of each model. After only a small number of test points, we can usually distinguish the best models (those with lowest error, or highest grade) from the very worst models (those with the highest error). The models which are significantly worse than the best ones are thrown out of the race and are not tested again. The more test points that are seen, the tighter the estimated error is to the true error; therefore, more models can be differentiated from each other and eliminated. This algorithm concentrates the computational effort on the best models, while discarding the inaccurate models and not testing them unnecessarily.

Unlike other machine learning methods, lazy learning algorithms do not force the training points into a decision tree, a neural net, symbolic rules, or any abstract representation. The points act as their own best representation. A direct implication of this fact is that very little work needs to be performed during training — we simply store the points in memory. Most of the computational expense comes when we use the learner for prediction. Racing uses laziness to its advantage, and also attempts to limit the number of predictions.

In this paper, we leverage the laziness in two ways. First, computing leave-one-out cross validation error is cheap because unlike non-lazy methods, there is no need to retrain on all of the points but one. We can simply 'cover up' that point in memory and that is equivalent to retraining. Therefore, we have a quick, reliable estimator for the error of a model by performing leave-one-out cross validation on all the points in the training set. Second, we can examine many different learners without worrying about the initial expense of training all of them. The only significant computation arises when they are tested.

This paper combines the authors' previous work on racing, [Maron and Moore, 1994],[Moore and Lee, 1994],[Maron, 1994], and adds additional experiments, discussion, and connections to lazy learning techniques.

1.1 Lazy learning algorithms

The collection of models to which we apply the racing algorithms in this paper are *lazy learning* methods. Lazy learning is an encompassing name for a variety of statistical and machine learning methods such as nearest-neighbor [Dasarathy, 1991] and local weighted regression [Cleveland *et al.*, 1988]. The underlying principle of lazy learning is to simply remember the training examples. Almost all of the computational work is performed not during training, but only when a query is made. Variants of the nearest-neighbor method are then used for prediction.

With a lazy learner, the data points, which are of the form (<input-vector> , <output-vector>) are kept in memory. The task is for the learner to predict output values when queried with inputs that might not be in memory. When the learner is asked to predict the value at $x = 5$, it looks in memory for a few points whose x-values are closest to 5. The output values at those points is averaged (or regressed [Moore *et al.*, 1992]) and returned as the predicted value. We assume in this paper that both the attributes (inputs) and predictions (outputs) are numeric, though many variations exist [Aha, 1990].

Within this general framework, there are still many choices for specifying a model:

- How do we measure *similarity*?
- How many neighbors should we look at?
- What method should be used for averaging the outputs of the neighbors?
- Should we try to perform local regression rather than local averaging?
- How much smoothing, or weighting, should we do over the regression or average?

Following are some possible answers to these questions:

1.1.1 k-nearest-neighbors

The motivation for k-nearest-neighbors is summarized by this quote:

> There seems to be a need for discrimination procedures whose validity does not require the amount of knowledge required by the normality assumption, the homoscedastic assumption, or any assumption of parametric form. ...can reasonable discrimination procedures be found which will work even if no parametric form can be assumed? [Fix and Hodges, 1951]

Fix and Hodges answered their own question by introducing nearest-neighbor classification. The principle is simple: when queried for the class of some attributes, return the class of the point with the most similiar attributes. Similarity (or distance) is usually measured by using the Manhattan or Euclidean metric[1], depending on the type of the attributes. The method generalizes easily to k-nearest-neighbors, where a majority rule is used to find the correct class, or averaging is used for continuous outputs. A larger value for k makes the rule more stable against outliers and noisy data. However, making k too large destroys the advantage of locality that a nearest-neighbor algorithm inherently owns.

1.1.2 Weighted distance metric

Finding an appropriate distance metric is often the toughest hurdle in attempting to use a lazy learning system. The distance metric is responsible for the representation of the memorized data. It decides which attributes are more important than others and aligns all the attributes into the same representation. To give an example, let us say that we are trying to predict hat size from height and weight. The data points are of the form $\langle$ height, weight, hat-size$\rangle$. Given a query q of the form $\langle$6ft,155LB$\rangle$, we look for the point p that minimizes the distance between p and q. However, it is probably not a good idea to use the standard Euclidean distance since differences in pounds are much less significant than differences in feet: someone who is $\langle$2ft,155LB$\rangle$ is closer to q than someone who is $\langle$6ft,165LB$\rangle$ using the plain Euclidean distance metric. Clearly, we need a weighted metric that forces differences in heights to be comparable to differences in weight. The initial representation (feet and pounds) should be changed to a more suitable one.

However, that is not the end of our problems. What if hat-size is predominantly affected by height, and only marginally by weight? In that case, we need to count the height attribute

[1] The Manhattan metric assumes that you can travel between two points only along a grid, so the distance between (x_1, y_1) and (x_2, y_2) is $\|x_1 - x_2\| + \|y_1 - y_2\|$. Using a Euclidean metric, the distance is $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.

much more than the weight attribute so that points with similar heights will be close together, even if they have very different weights.

Both problems can be solved with a priori knowledge of the importance and scale of the various attributes, but one of the main reasons for using nearest-neighbor methods is to remove any assumptions about the best representation for the data. The best weighting, like the best value of k, should be searched for. Other, non-symmetric distance measures are possible [Maron, 1994], but are not discussed in this paper.

1.1.3 Kernel regression

It is intuitive that some neighbors are more important than others. Specifically, neighbors that are closer to the queried point should count more toward its classification than neighbors that are farther away. *Kernel regression* performs weighted averaging (instead of a uniform average) over the space of neighbors. The weighting function which we use is taken from [Moore *et al.*, 1992]. The weight of the i^{th} point (x_i) with respect to the query point q is w_i.

$$w_i = \frac{1}{1 + 20 \cdot (distance(x_i, q)/K_{width})^2} \tag{1}$$

The parameter K_{width} determines the width of the weighting (or smoothing) range. The larger it is, the more weight is given to distant points. Again, it is usually not feasible to estimate this parameter before empirically testing various values of it.

1.1.4 Local regression and local weighted regression

The computationally intensive part of nearest-neighbor algorithms is usually to find the k neighbors. Once that is done, we can perform more complex operations than averaging to predict the queried value. For example, *local regression* involves finding a least-squares linear fit of the neighbors. *Local weighted regression* [Cleveland *et al.*, 1988] attempts to minimize the weighted error, where the weighting function is Equation 1.

1.2 Determining the accuracy of a model

To determine the accuracy of a model, we measure the difference between the predictions of the model and the true answers on a set of points where we have the true answers. However, we need to be careful not to use the training set for testing as well. For example, using a 1-nearest-neighbor method, we can get zero prediction error by testing on the training points. However, the accuracy of 1-nearest-neighbor on new points (the generalization error) can be much lower. What is needed is a way to test not only rote memorization, but also the ability to generalize.

There are a number of popular methods of estimating the learner's ability to generalize. One is partitioning the entire set of the teacher's examples into a training set and a testing set. The learner is trained on the training set, but its performance is determined on its average prediction error for points in the testing set. A prediction query involves giving the learner the attributes, but withholding the correct outputs. The learner's guess at the outputs is then compared to the true outputs.

The second method of estimating generalization is called *leave-one-out cross validation.* Here, we can use the same set of points for both training and testing by performing the following trick: train on all points except one; perform a prediction query on the one point that was held out; repeat this process for all points and return the average prediction error of all the iterations. This method might seem more computationally expensive than the test-set method, but it does have certain advantages, namely in cases where we cannot afford to partition the few training points that we have (i.e., there are not enough points) or when we need a good estimate of the error distribution over the entire space we are trying to learn. Also, for lazy learners, leave-one-out cross validation is efficient.

Another popular method is multifold cross validation, where instead leaving just one point out, a different set of points is held out during each iteration and is then used as the test set. The average of the errors gives an estimation of the generalization error.

Cross validation is a special case of a number of statistical reuse techniques such as jackknife and bootstrapping. For a readable overview of these techniques, see [Efron and Tibshirani, 1991].

1.3 Summary

The paper is organized as follows: we discuss previous approaches to model selection such as brute force, gradient descent and genetic algorithms. We show that they are either slow or inappropriate. We then describe the racing algorithm, and talk about two bounds that can be used to differentiate the various models' errors. We show that using Hoeffing's bound is general and correct, but it is not a very tight bound. We give a Bayesian bound which turns out to be faster in practice. We also discuss various extensions and heuristics that can be made to help the race.

We then describe a set of experiments that show some empirical results of running the various types of races on real datasets. We then discuss a different usage of racing; namely, to find the relevant attributes for prediction. Finally, we conclude with some related work and future directions.

2 Previous approaches

Traditionally, there have been a number of popular ways to search through a large collection of models. Brute force was always applicable and gave the desired result, but at a high computational price. Descent methods such as hill climbing and conjugate gradient were much faster, yet did not guarantee to return the right result and, even worse, was not applicable in many cases. Other techniques such as simulated annealing and genetic algorithms fall into similar traps, since provably convergent versions of simulated annealing and genetic algorithms are too slow to be used in practice. We discuss these approaches in the following subsections. Racing [Maron and Moore, 1994] manages to lessen the evils of these techniques, while retaining their benefits.

2.1 Brute force

Brute force attacks the problem in the simplest possible way. Given m trained learners and a test set that consists of n points, it performs a prediction query on every point for every single learner. It then computes the mean squared error of the predictions of each learner and selects the learner with the lowest error. This algorithm runs in time $O(n \cdot m)$, but always returns the best learner.

2.2 Descent methods

Gradient descent, or hill climbing [Press *et al.*, 1992], treats the collection of models and their prediction error as a continuous and differentiable surface. It starts at some point on

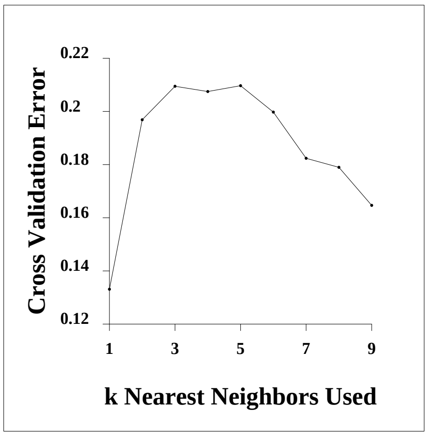

Figure 1: The cross-validation error of a space of models consisting of local-weighted-regression models with different numbers of nearest neighbors used. The global minimum is at one-nearest-neighbor, but a gradient descent algorithm would get stuck in local minima unless it happened to start in in a model where $k < 4$.

this surface, and proceeds to 'descend' in the direction that has less error. This approach to optimizing the lazy learner has been taken by [Atkeson, 1990, Lowe, 1995]

The algorithm stops when it reaches a local minimum. In other words, when all of the neighboring models have higher error, the algorithm returns the current model. This algorithm is much faster than brute force since it does not need to find the error of every single model. It only needs to compute the error for the learners that are on the path to the optimal learner. However, there are two major problems: local minima and applicability.

Local minimum is a well documented and analyzed problem with gradient descent. An example of it in the realm of model selection is given in Figure 1. In general, the error at the local minimum can be many times larger than the error at the global minimum, and in addition, the chance of starting at a place which will lead to a global minimum can be arbitrarily small.

There is an even more serious problem with gradient descent that was ignored in the last example — namely, there are many instances when it is not even applicable to model selection. At every step of the gradient descent algorithm, we need to find a collection of models which are 'near' the current model. However, the collection of models may not have a distance metric (and therefore no concept of 'near') defined on it. For example, how far is a neural network from a nearest-neighbor model? How far is a decision tree from a local weighted regressor?

It is possible to *impose* a distance metric on any collection of objects, for example in the k-nearest-neighbor example, or somewhat deviously by ordering the models according to simplicity. However, although we can impose a metric, it does not mean that it is correct, or even useful. Just because there is a parameter to tune (e.g., k in k-nearest-neighbor) does not mean that there is a predictable relation between a change in the parameter and the performance of the algorithm. In the cases where there is no clear distance metric among the various models, all of them must be raced. This is a problem with all minimization methods. Gradient descent, steepest descent, Newton's method [Press *et al.*, 1992] and conjugate gradient are as susceptible to this problem as the plain version of gradient descent.

2.3 Genetic algorithms and simulated annealing

The same problems described with descent methods also apply to these techniques, only less explicitly. In a genetic algorithm [Goldberg, 1989], we start with a population of different models, where each member is represented by a description which can be mutated and crossed with other descriptions. There is a certain probability that some bit in the representation mutates. Therefore, there is a distance imposed between models that is proportional to the average number of generations it takes to get between two representations. Again, this distance metric is arbitrary and is based solely on the representation of the models, which is as meaningless as having it based on their position in a room.

Simulated annealing [Kirkpatrick *et al.*, 1983] at high temperature performs as slowly as brute force. At low temperatures, it is restricted by the same problems described above for gradient descent. There are cooling schedules which guarantee convergence at a global minimum, but they are too slow to be practical.

3 Racing: Hoeffding and Bayes

In this section we formalize the notion of "racing" by defining what it means for one model to be winning the race over another. We use two different statistical bounds to determine with a certain confidence when one model's average error is significantly smaller than another's. Hoeffding's bound is a general bound that only relies on the assumption that the error queries are drawn independently. Using Bayesian bounds gives tighter results, at the expense of assuming that the errors are drawn from a normal distribution.

3.1 Hoeffding races

3.1.1 Hoeffding's Bound

Let us say that we have N points with which to test a given model. If we were to test a model on all of them, then we would have an average error that we will call E_{true}. However, if we only tested the model on 10 points, then we only have an estimate of the true average error. We call the average after only n points ($n < N$) E_{est} since it is an estimate of E_{true}. The more points we test on (the bigger n gets), the closer our estimate gets to the true error. How close is E_{est} to E_{true} after n points? Hoeffding's bound lets us answer that question when the n points are picked with an identical independent distribution from the set of N original test points. In this case, we can say that the probability of E_{est} being more than ϵ away from E_{true} is

$$\Pr(|E_{true} - E_{est}| > \epsilon) < 2e^{-2n\epsilon^2/B^2} \tag{2}$$

where B bounds the greatest possible error that a model can make [Hoeffding, 1963]. This bound does not make any assumptions other than the independence of the samples.

We would like to say that "we are 99% confident that our estimate of the average error is within ϵ of the true average error", or in other words, $\Pr(|E_{true} - E_{est}| > \epsilon) < 0.01$. We denote the confidence parameter with δ. Equating δ with the right-hand side of Equation 2 gives us an expression for ϵ in terms of n, B, and δ.

$$\epsilon(n) = \sqrt{\frac{B^2 \log(2/\delta)}{2n}} \tag{3}$$

Equation 3 tells us how close the estimated mean is to the true mean after n points with confidence $1 - \delta$. We will discuss how to obtain a value for B in section 4.1. We can also determine the number of samples n necessary to obtain a certain accuracy ϵ and confidence δ:

$$n > \frac{B^2 \log(2/\delta)}{2\epsilon^2} \tag{4}$$

The number of samples increases quickly as we attempt to increase the accuracy. This is a problem with using Hoeffding's bound, and we try to resolve it by using Bayesian bounds which are less general but tighter.

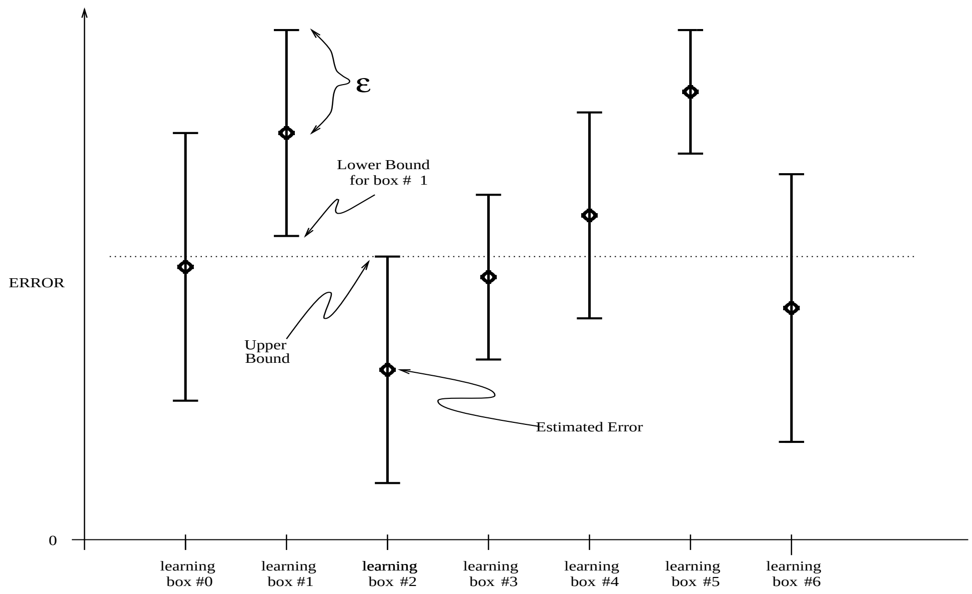

Figure 2: An example where the best upper bound of learning box #2 eliminates learning boxes #1 and #5. The size of ϵ varies since each learning box has its own upper bound on its error range, B.

3.1.2 Hoeffding races

The racing algorithm starts with a collection of learning boxes. We call each model a learning box since we are treating the models as if they were black boxes. We do not consider how complex or time-consuming each prediction is, but only the input and output of the box. Associated with each learning box are two pieces of information: a current estimate of its average error and the number of test points it has processed so far. Assume the dataset has N datapoints.

At each iteration of the algorithm, we randomly select a point from the test set. Then for each learning box:

- compute the leave-one-out error at the point by using that learning box.
- update the learning box's estimate of its own average error rate.
- use Hoeffding's bound and Equation 3 to calculate how close the current estimate is to the true error for each learning box.

Each learning box now has a bound within which its true average error lies. We can eliminate those learning boxes whose best possible error (their lower bound) is still greater than the worst error of the best learning box (its upper bound); see Figure 2. The intervals get smaller as more points are tested (since ϵ gets smaller as n gets larger), thereby "racing" the good learning boxes and eliminating the bad ones.

3.1.3 Ending the race

The algorithm iterates, repeatedly picking test points until one of three conditions occurs:

1. All but one of the learning boxes have been eliminated.
2. A sufficient number (e.g., N) of test points have been picked.
3. Alternatively, the algorithm can be stopped once ϵ has reached a certain threshold.

In any case, the algorithm returns a set of learning boxes whose errors are indistinguishable to within $2 \cdot \epsilon(n)$.

3.1.4 Proof of correctness

Notice that $1 - \delta$ is the confidence in Hoeffding's bound for *one* learning box during *one* iteration of the algorithm. What we need is to prove that the *entire* algorithm has some confidence $1 - \Delta$ of returning the best learning box.

For the sake of a simpler proof, let us make the requirement of a correct algorithm more stringent. We will say that the algorithm is correct if every learning box is within ϵ of its true error at every iteration of the algorithm. This requirement encompasses the weaker requirement that we do not eliminate the best learning box. An algorithm is correct with confidence $1 - \Delta$ if

$$\Pr\{ \textit{ all learning boxes are within } \epsilon(n) \textit{ on all iterations up to } n\} \geq 1 - \Delta \tag{5}$$

What we would like to do is show the relationship between δ (the chance of being wrong on one learning box in one iteration) and Δ (the chance of being wrong on the whole algorithm), so that when the user wants the algorithm to work with probability 0.999, we can translate that into the confidence that we need for each learning box at each iteration. We will be relying on the disjunctive probability inequality, which states that for any events A and B, $Pr\{A \vee B\} \leq Pr\{A\} + Pr\{B\}$.

Assume that there are n iterations, and m learning boxes $(LB_1, \cdots, LB_m)$. We start with the fact that:

$$Pr\{ \textit{ a particular LB is within } \epsilon(i) \textit{ on a particular iteration } i\} \geq 1 - \delta \tag{6}$$

Which is equivalent to:

$$Pr\{\ a\ particular\ LB\ is\ wrong\ on\ a\ particular\ iteration\} < \delta \tag{7}$$

Using the disjunctive inequality,

$$\begin{aligned} Pr\{ \quad & a\ particular\ LB\ is\ wrong\ on\ iteration\ 1\ \vee \\ & a\ particular\ LB\ is\ wrong\ on\ iteration\ 2\ \vee \\ & \dots \\ & a\ particular\ LB\ is\ wrong\ on\ iteration\ n\} \leq \delta \cdot n \end{aligned} \tag{8}$$

Rewrite this as:

$$Pr\{\ a\ particular\ LB\ is\ wrong\ on\ any\ iteration\ up\ to\ n\} \leq \delta \cdot n \tag{9}$$

And likewise for all learning boxes:

$$\begin{aligned} Pr\{ \quad & LB_1\ is\ wrong\ on\ any\ iteration\ \vee \\ & LB_2\ is\ wrong\ on\ any\ iteration\ \vee \\ & \dots \\ & LB_m\ is\ wrong\ on\ any\ iteration\} \leq \delta \cdot n \cdot m \end{aligned} \tag{10}$$

or in other words:

$$Pr\{\ some\ LB\ is\ wrong\ in\ some\ iteration\} \leq \delta \cdot n \cdot m \tag{11}$$

which is equivalent to:

$$Pr\{\ all\ LBs\ are\ within\ \epsilon(n)\ on\ all\ iterations\} \geq 1 - \delta \cdot n \cdot m \tag{12}$$

Clearly, Equation 12 is the same as Equation 5 and we can therefore conclude that $\delta = \frac{\Delta}{n \cdot m}$. When we plug this into Equation 3 (our expression for ϵ from the previous section), we ensure the correctness of this algorithm with confidence Δ. The new ϵ is expressed as:

$$\epsilon(n) = \sqrt{\frac{B^2(\log(2nm) - \log(\Delta))}{n}} \tag{13}$$

This is an extremely pessimistic bound on Δ and tighter proofs are possible [Omohundro, 1993]. It is pessimistic in two regards: first, it assumes that all learning boxes are completely independent of each other. Second, it assumes that the error of a learning box after seeing n points is completely independent of its error after seeing $n+1$ points. This is clearly a worst case assumption, and most PAC bounds are made tighter by leveraging this point [Haussler, 1992].

3.2 Bayesian races

Hoeffding's bound is as tight as we can get without making additional assumptions about the distribution of the errors we get from a model. If we assume that the errors are normally distributed, then we can use Bayesian statistics to achieve a tighter bound on the probability that one model is better than another.

The algorithm first randomizes the order of the datapoints. For the jth model the distribution of leave one out errors is Gaussian. Let us say it has unknown mean E^j_{true} and unknown variance V^j_{true}.

As evidence accumulates, the uncertainty of E^j_{true} decreases, since E^j_{true} refers to the leave-one-out errors, not the out of sample errors. Let $\hat{\mu}_{nj}$ and $\hat{\sigma}^2_{nj}$ be the sample mean and variance of model j's errors up to the nth iteration, when all surviving models have been evaluated on n datapoints.

$$\hat{\mu}_{nj} = \frac{1}{n}\sum_{i=1}^{n} e_j(i) \qquad \hat{\sigma}^2_{nj} = \frac{1}{n-1}\sum_{i=1}^{n}(e_j(i) - \hat{\mu}_{nj})^2 \tag{14}$$

defining $e_j(i)$ as the leave-one-out error when the jth model is used to predict the ith datapoint. These statistics can be updated incrementally efficiently: $\hat{\mu}_{(n+1)j}$ and $\hat{\sigma}^2_{(n+1)j}$ can be defined as a closed form of n, $\hat{\mu}_{nj}$, $\hat{\sigma}^2_{nj}$, and $e_j(n+1)$.

For each model we use Bayesian statistics, along with the values $\hat{\mu}_n j$, $\hat{\sigma}^2_{nj}$, and n, to put a probability distribution on E^j_{true}. This is a relatively elementary process. Assume that a priori we know nothing about the distribution except that it is normal. The true mean E^j_{true} might be any value between $-\infty$ and $+\infty$. The variance V^j_{true} might be any value between 0 and $+\infty$. This ignorance can be turned into the uninformative priors:

$$p(E^j_{true}) = \text{Constant} \qquad p(V^j_{true}) = \text{Constant}/\sigma \tag{15}$$

where $p(.)$ denotes a probability density function.[2] The marginal posterior distribution of the mean E^j_{true} is a student distribution with mean $\hat{\mu}_{nj}$, variance $\hat{\sigma}^2_{nj}/n$, and $n-1$ degrees of freedom.

Figure 3 gives an example with four models racing. We eliminate the jth model if there exists some other model j' for which

$$\text{Prob}(E^j_{true} < E^{j'}_{true} \mid e_j(1), \ldots e_j(n), e_{j'}(1), \ldots e_{j'}(n)) < \delta \tag{16}$$

[2] Neither of these priors is a legitimate probability density function. Such an approximation is harmless (see [Schmitt, 1969] for more details).

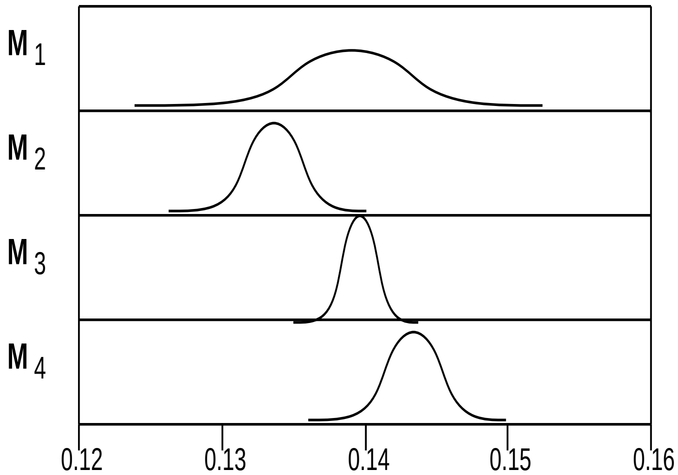

Figure 3: The posterior distributions on the errors of four models involved in a race. The lower the error the better, so it seems very unlikely that model 4 will turn out to be better than model 2 at the end of the race. Thus model 4 is eliminated at this point.

δ again denotes the confidence that is required. Statistics are gathered and models are eliminated until only one model remains, or we run out of datapoints, whereupon we select the model with the lowest $\hat{\mu}_{nj}$. In Figure 3 it would be very probable that E^2_{true} is lower than E^4_{true}, but only marginally probable that E^1_{true} is lower than E^4_{true}.

We thus need to compute the probability that the means of one unknown Gaussian distribution is less than the mean of another unknown Gaussian distribution, where in each case the posterior distributions of each mean can be computed with Bayesian statistics. This is achieved by the Welch approximation to the Behrens-Fisher problem [Welch, 1937].

Given two samples with the same assumptions as before, let the first sample have size n_1, sample mean $\bar{x}_1$ and sample variance s_1^2. The corresponding values for the second sample are n_2, $\bar{x}_2$ and s_2^2. Let

$$u_1 = s_1^2/n_1, \quad u_2 = s_2^2/n_2, \quad b = u_1/(u_1 + u_2). \tag{17}$$

Then $\mu_1 - \mu_2$ (the signed difference between the population means) has, approximately, a student distribution with mean $\bar{x}_2 - \bar{x}_1$, variance $u_1 + u_2$, and degrees of freedom

$$\left(\frac{b^2}{n_1 - 1} + \frac{(1-b)^2}{n_2 - 1} \right)^{-1}. \tag{18}$$

By computing the cumulative probability distribution, we get the probability that $\mu_1 < \mu_2$.

4 Extensions

This section describes several extensions and expansions of the basic racing algorithm. We describe how to determine a bound on B — the maximum error of a model on any test query, to be used in Hoeffding races. We then describe a method for tightening the confidence intervals for each model and for eliminating redundant models in the set (models that are statistically equivalent). We also talk about a statistical method known as *blocking* [Box *et al.*, 1978] that helps with the problems of large variances and dependent models.

4.1 Bounding errors

The most obvious obstacle to implementing Hoeffding races is finding a value for B, the maximum error of a learning box. For classification problems, there is no difficulty — B is simply 1. That is because the worst mistake that the algorithm can make is a misclassification which has an error of 1. For regression problems the solution is less straightforward. If we know something about the learner and something about the data, then we can try to put some finite bound on B. If that is not possible, then we can attempt to estimate B by adding a few standard variances to the average error of this learner. Since the average error and variance are updated after every new point, the value of B also gets modified at each iteration. The value of B tends to fluctuate wildly during the first few points until the average and variance settle down. To avoid that, we only start racing after collecting errors from about 30 points from each learning box.

4.2 Shrinking the intervals

There is a simple heuristic to tighten the bounds around the estimated error for each learning box. First we need to name a few important components of a learning box. Let us call the estimated error for the k^{th} learning box at the i^{th} iteration $E^k_{est}(i)$. We will call the lower bound of that learning box $lower^k(i)$ and likewise the upper bound will be called $upper^k(i)$. These can be calculated by $E^k_{est}(i) - \epsilon(i)$ and $E^k_{est}(i) + \epsilon(i)$ respectively. From now on, we will be dropping the superscript, since we will be talking about one learning box, but applying the ideas to all of them.

During the course of running the race, all three of these components tend to fluctuate. $E_{est}(i)$ moves around with every new point, trying to get closer to the true error. $lower(i)$ and $upper(i)$ move around for two reasons: the first is that ϵ gets smaller at each iteration; the second is that $E_{est}(i)$ changes after almost every new point. However, despite all of this movement, we are guaranteed that with confidence $1 - \Delta$, $E_{est}(i)$ will stay between the lower

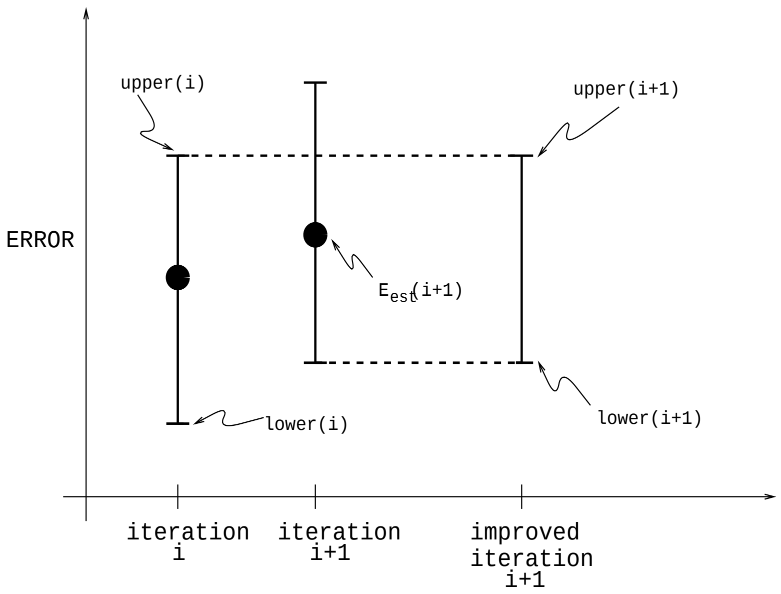

Figure 4: Example of how to shrink the interval bounds

and upper bounds. What if at iteration $i+1$ we get a point upon which the learning box performs very badly? In this case, $E_{est}(i+1)$ is larger than $E_{est}(i)$. The bounds have become tighter because of the decrease in ϵ, but they have been transformed by the increase in E_{est}. However, $E_{est}(i+1)$ is guaranteed to stay not only within the bounds at iteration $i+1$, but also within the bounds at all iterations until now. Therefore, the new upper bound should not be $E_{est}(i+1)+\epsilon(i+1)$, but instead the tighter

$$upper(i+1) = Min(E_{est}(i+1)+\epsilon(i+1), E_{est}(i)+\epsilon(i)). \tag{19}$$

Likewise, a tighter lower bound at iteration $i+1$ is

$$lower(i+1) = Max(E_{est}(i+1)-\epsilon(i+1), E_{est}(i)-\epsilon(i)) \tag{20}$$

An example of shrinking the intervals is shown in Figure 4. The bound of iteration $i+1$ can be improved based on previous bounds.

4.3 Eliminating indistinguishable models

So far we have only discussed elimination of models that are considerably worse than the top model. We can also try to speed up the race by eliminating models that seem to be identical in predictive power to other models. In other words, we wish to eliminate any model that we are confident is worse than some other model; we also wish to stop a race between two models that we believe with high confidence to be extremely similar. The latter can be achieved by defining a threshold γ (a small positive number) and eliminating any model that we are confident has an estimated error within γ of another model.

These two rules can be combined into one formulation: given two distinct learning boxes LB_i and LB_j, with an estimated error of E^i_{est} and E^j_{est} respectively, we eliminate LB_j from the race if

$$\Pr\{E^i_{est} < E^j_{est} - \gamma\} < \delta. \tag{21}$$

4.4 Blocking

There are two common problems that empirically seem to slow down the races. One problem occurs when learning boxes have a vary large variance in error over the test set. In that case, we cannot eliminate them or use them to eliminate other learning boxes since their confidence are so large. Another problem occurs when two or more learning boxes have some degree of dependence, so that they perform badly on the same points. In that case, we need many test points to distinguish between them.

We deal with both problems by using a statistical method known as *blocking* [Box *et al.*, 1978]. Instead of racing to get a tight estimate of a model's error, we race for a tight estimate of the difference in error between two models. Rather than maintaining an average error for each learning box, we maintain an average error difference (denoted $ED^{i,j}_{est}$) for every pair of learning boxes (i, j). If we let $e_i(l)$ be the error of the i^{th} learning box on the l^{th} testing point, then our estimate of the error difference after seeing k test points is

$$ED^{i,j}_{est} = \frac{1}{k}\sum_{l=1}^{k}(e_i(l) - e_j(l)) \tag{22}$$

As before, we can use either Hoeffding's Bound or a Bayesian bound in order to determine when the error difference between two models becomes significant or, in other words, when

$$\Pr\{ED^{i,j}_{est} < 0\} < \delta. \tag{23}$$

We check this condition for every pair (i, j) of learning boxes and eliminate LB_j if it is true.

Indistinguishable models can be eliminated if $\Pr\{ED^{i,j}_{est} < -\gamma\} < \delta$. The simplest example of blocking having a beneficial effect is the case where two models in the race are identical — the original racing algorithm would have to race for a long time. The racers would have the same mean at each step of the race, but the race would only end when they both had so many samples that the confidence intervals on its measures of the mean error of each model were both very close to γ (its indifference parameter). In contrast, the blocking race algorithm would maintain, at each step, the difference between the leave-one-out errors on each datapoint. Because the models are the same, this difference would always be zero and it would only require a very small number of statistics (perhaps less than ten, depending on the parameters) before one of the models was eliminated. This example of identical models is extreme, but in the more common case of near-identical models there can also be a large reduction in the time to elimination.

5 Empirical results of Hoeffding and Bayesian racing

We ran races on a wide variety of classification and regression problems that are described below. Some of these results were also presented in [Maron and Moore, 1994]. The data files are available from the UCI repository [Murphy and Aha, 1995]. The problems were chosen arbitrarily, but turned out to have widely different optimal models: from 3-nearest-neighbor to local weighted regression with a large smoothing kernel.

ROBOT Given initial and final description of a robot arm, learn the control needed for juggling [Schaal and Atkeson, 1993]. 10 input attributes, 5 outputs.

PROTEIN 3 inputs, and output is a classification into one of three classes. The famous protein secondary structure database, with some preprocessing [Zhang *et al.*, 1992].

ENERGY Given solar radiation sensing, predict cooling load for a building. This is taken from the Building Energy Predictor Shootout [Kreider and Haberl, 1993].

POWER Market data for electricity generation pricing period class for the new United Kingdom Power Market.

POOL The visually perceived mapping from pool table configurations to shot outcome for two-ball collisions [Moore, 1992].

DISCONT An artificial dataset with many discontinuities. Local models should outperform global ones.

We ran the races on a collection of lazy learning algorithms. The learning boxes varied in the number of nearest neighbors that they looked at ($k = 1, 3, 5, 7, 9$), in the degree of smoothing performed ($K_{width} = 4, 2, 1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{32}, \frac{1}{64}$), and whether the function was locally constant or performed local weighted regression (see Section 1.1. These combinations give us either 95 or 189 different models, depending on the permutations used. The distance metric was not varied, and all attributes were held to be equally important. All of the experiments were run using $\delta = 0.01$. We used the techniques mentioned earlier for estimating B and for shrinking the intervals.

In Table 1, we compare the brute force approach to Hoeffding races, Bayesian races, and Bayesian races with blocking. Each of the columns is the results for one of the datasets described above. The first two rows of the table describe how many points are in each dataset and how many learning boxes (LBs) are in the initial set of models. BRUTE is the number of leave-one-out cross validation (**LOOCV**) queries made by the brute force algorithm. Likewise, HOEFFDING, BAYES and BLOCK-BAYES show the average number of **LOOCV** queries made by each of those algorithms over 5 runs. We also show the speedup in terms of the number of queries for each of the algorithms with respect to BRUTE. Hoeffding races usually returns a set of LBs that cannot be distinguished, and for these datasets it always contained the model that BRUTE chose. However, the Bayes-based algorithms returned just one model. We measured its quality by "regret", which is the difference between the BRUTE's top model and the model returned, normalized by the standard deviance of the output.

There are a few observations to be made from this table:

- Racing never performs more queries than brute force, and its overhead is negligible, except in the case of the blocking race, which has quadratic cost in the number of racers. In addition, the accuracy of the model (or models) selected is very close to the accuracy of the best model in the initial set.

- Bayesian races always improved on Hoeffding races, and Bayes with blocking in turn improved upon Bayesian. The improvement over brute force tends to be greater for larger datasets since a larger dataset means the exhaustive method has to perform proportionally many extra **LOOCV** evaluations, whereas the racing methods can stop before all datapoints are evaluated.

- The Bayesian methods are faster than Hoeffding, but they do not carry a formal correctness proof. Specifically, they will sometimes throw out good models early in the

Table 1: Results of various races.

	ROBOT	PROTEIN	ENERGY	POWER	POOL	DISCONT
points	972	4,965	2,444	210	259	500
initial learning boxes	95	95	189	95	95	95
BRUTE	92,340	471,675	461,916	19,950	24,605	47,500
HOEFFDING	15,637	349,405	121,400	13,119	22,095	25,144
LBs left	6	60	40	48	75	29
speedup	**5.91**	**1.35**	**3.80**	**1.52**	**1.11**	**1.89**
BAYES	15,482.8	204,466	47,689.2	9,318.2	11,138.4	1,158.4
regret	0.004	0.016	0	0.0001	0	0
speedup	**5.96**	**2.31**	**9.69**	**2.14**	**2.21**	**41.00**
BLOCK-BAYES	1,839.4	4,430.8	19,444.6	5,421.8	1,409.4	906.2
regret	0.005	0.05	0	0.002	0	0
speedup	**50.20**	**106.45**	**23.76**	**3.68**	**17.46**	**52.41**

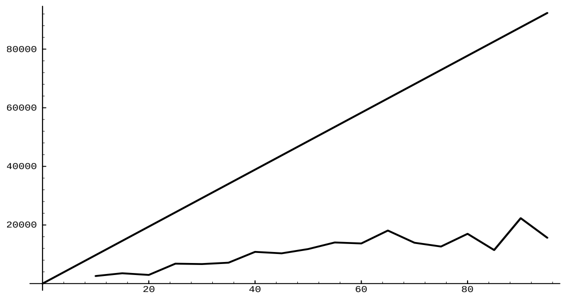

Figure 5: The bottom line shows the number of queries taken by Hoeffding races for the ROBOT problem as the size of the initial set of learning boxes is increased. The top line shows the performance by brute force. At each point, the set of learning boxes was chosen randomly.

race, especially if the errors generated by those good models do not fit the priors (i.e., the errors are not Gaussian).

- Hoeffding races (and racing in general) is least effective when a large percentage of the original learning boxes are left at the end. For example, in the POOL problem, where there were 75 learning boxes left at the end of the race, the number of queries is only slightly smaller for Hoeffding races than for brute force. In the ROBOT problem, where only 6 learning boxes were left, a significant reduction in the number of queries can be seen.

- The obvious conclusion from this observation is that racing is most effective when there exists a small subset of clear winners within the initial set of models. In fact, the larger the size of the initial set of models, the more effective racing is in comparison to brute force.

To test this conclusion, we created random subsets of increasing sizes from the 95 learning boxes used for the ROBOT experiment. We ran Hoeffding races on each one of the subsets, and tabulated the results in Figure 5. As can be seen, we can search over a large set of models without much concern about the computational expense of a large initial set. In other words, if we have very little knowledge of the problem, we should not preclude any possible solution. Racing lets us do that without much computational expense.

We do not think that there is a general asymptotic relation between races and brute force as the number of models grow. As shown by Table 1, it is problem dependent. However, it is clear that the Racing algorithms work badly on 'boring' problems, where any arbitrary model does as well as anything else, and works very well on 'interesting' problems, where only a few models really fit the domain. Using the classroom analogy again, the boring problems can be thought of as picking the best gym student (everybody gets A's in gym), and the interesting problems can be thought of as picking the best history student (where the grade distribution is such that there are only a few excellent students).

5.1 Segmenting Magnetic Resonance Images

In addition to the small datasets described above, we used the combination of Hoeffding races and lazy learning in a larger application. The input is a 3-D image of the brain, generated in slices from a Magnetic Resonance machine. Each point in the image becomes a point in the training set. A point is described by its (x, y, z) coordinates in the image and by two descriptive attributes: the proton density and the viscosity of the tissue. This quintuple is mapped into a classification (segmentation) of either gray matter, white matter, fluid, or lesion. The entire data set is quite large: 10 patients, two to four images across time for each patient, and about 0.5 million points per image.

The segmentation problem is important in several medical imaging applications. For example, Multiple Sclerosis causes brain lesions, whose volume needs to be tracked over time to determine if the treatment is working, or the rate of deterioration. If this were to be done by hand, a specialist would need to look through over 50 slices of the brain, determine the location and size of the lesions in each slice, and sum them to get a piece of information for one person during one time point. This becomes horribly time consuming when there are many patients getting scanned on a weekly basis.

We would like to automate the process by training on images from a single patient, taken at different times, and then segmenting future images. An even more ambitious goal is to train on multiple patients and predict on unseen patients' images.

A detailed description of the various experiments and results can be found in [Maron, 1994]. In summary, using Hoeffding races allowed us to make only 69% of the queries and saved hundreds of hours of CPU time. The best learning box incorrectly classified 18% of points from new patients, and more importantly, produced results that looked reasonable to medical experts.

6 Racing for relevant attributes

A particularly promising use of cross validation is to automatically choose relevant inputs from a wider set of possible inputs. An obvious benefit is in accelerating the learning rate of algorithms which suffer in the face of irrelevant inputs. Other benefits include helping select relevant visual features for visually controlled robot tasks [Moore *et al.*, 1992], and selecting sets of time windows in time series predictions. This problem, known as "subset selection" or "feature selection" is a well-known problem in statistics, surveyed thoroughly in [Miller, 1990] and is rapidly gaining attention in the Machine Learning community [Caruana and Freitag, 1994, John *et al.*, 1994, Skalak, 1994].

Given D inputs there are 2^D possible input sets, and so performing an exhaustive cross validation search over all of them soon becomes impractical as D rises, even assuming adequate data support to justify searching so many models. Hill climbing is a sensible alternative since the space we are climbing over has a reasonable "neighbor" concept. In this section we provide several hill climbing versions of the racing algorithm; these aim to both speed up the computation and also to reduce the danger of becoming trapped at local maxima.

Sets of inputs can be represented as binary strings. Given four possible inputs, 0101 would denote "ignore inputs 1 and 3, use inputs 2 and 4." The standard non-racing hill climbing algorithm, *forward sequential selection*, begins with a start string (e.g., 0000) then makes all possible 1-feature changes to it (1000, 0100, 0010, 0001) and exhaustively finds which minimizes the leave-one-out cross validation (**LOOCV**) error. It then uses this best string (say 0100) as a new base point, generates all its 1-feature successors (1100, 0000, 0110, 0101), and determines the best. It continues in this way until no single-feature change improves it. The special case of starting with all zeroes is termed forward sequential selection (FOR-SEL), and that of starting with all ones is termed backward elimination (BACK-SEL). Forward sequential selection is better if only a few features are expected to be relevant and backward elimination is better if only a few features are expected to be irrelevant. Unfortunately, such prior knowledge may not be available at the start of the search.

The racing counterparts to these algorithms are straightforward: from the base string generate all 1-feature changes and race them. Proceed until the winner of a race does not improve on its base. In the experiments described later two versions are tested, FOR-RACE and BACK-RACE, that start at all zeroes and all ones respectively.

However, there is an objection to this simple application of racing. Imagine that inputs 4, 5, and 6 are all relevant and independently provide a reduction in the LOOCV error. If we start at string 0000000 and successors 0000100, 0000010, and 0000001 are all good, it will be a shame to run through three separate hill climbing iterations to switch them all on. This motivates the next algorithm, a Gauss-Seidel version of hill climbing:

- Begin with a predefined start string (e.g., 00000)
- Race between the current string and the current string with the first bit flipped (00000 versus 10000)
- Select the winner of that race as the new current string (e.g., 10000)
- Now race between the current string and the current string with the second bit flipped (10000 versus 11000)

$\vdots$

...until all bits have been raced. Then return to the first bit and proceed until an entire pass through the current string fails to produce an improvement.

Versions of this algorithm, FOR-GS-RACE and BACK-GS-RACE, are tested below. On some occasions they do indeed help, but on others their performance is poor. A further new algorithm, *schemata search* does Gauss-Seidel's job better, and also solves another problem.

6.1 Schemata search

Schemata search is a new algorithm described in [Moore and Lee, 1994] which aims to solve the same problem that the Gauss-Seidel method addresses—the problem of forward sequential selection taking a long time if many features are relevant, or similarly backward elimination taking a long time if many features are irrelevant. It will also help with a second problem. Suppose there is a family of three features which must all be on simultaneously for any reduction in the **LOOCV** error. If any family member is ignored then the **LOOCV** error is as bad as if all family members were ignored. This can happen quite easily, for example if the features are distributed between -1 and 1 and the function being learned is their product. Forward sequential selection would be very likely to miss this family and to converge on something suboptimal. Backward elimination would not have this problem, but if many features are irrelevant then it can become stuck as well (because in the early stages of hill climbing the removal of one irrelevant attribute among many does not improve the **LOOCV** error).

Schemata search searches over the space of schemata strings, which have 0's, 1's and $\star$'s in them. A $\star$ denotes a fifty percent chance of the attribute being ignored, and a fifty percent chance of it being used. The **LOOCV** error of such a string is the expected **LOOCV** error of a binary string generated from the schemata string according to these random rules, for example

$$\mathbf{LOOCVE}(101\star\star) = \frac{1}{4}\left(\mathbf{LOOCVE}(10100) + \mathbf{LOOCVE}(10101) + \mathbf{LOOCVE}(10110) + \mathbf{LOOCVE}(10111)\right)$$

Now a simple algorithm is to begin with all stars (e.g., $\star\star\star\star\star$) , and then to find out (by racing) whether it is better to have the first field as a 1 or a 0 (i.e., we would race $1\star\star\star\star$ and $0\star\star\star\star$). Having finished the first race, we could determine the second field by another race, and so on until the entire string is filled with 1's and 0's.

In practice, we can do better than this. Instead of beginning by racing the first field, we can race all fields against each other in parallel:

$1\star\star\star\star$	races against	$0\star\star\star\star$
$\star1\star\star\star$	races against	$\star0\star\star\star$
$\star\star1\star\star$	races against	$\star\star0\star\star$
$\star\star\star1\star$	races against	$\star\star\star0\star$
$\star\star\star\star1$	races against	$\star\star\star\star0$

Thus, given D inputs, we have D races occurring in parallel, and we stop all races when any one race produces a winner (to confidence level δ). On each step of the race a random binary string is generated, and then the **LOOCV** error of one randomly chosen datapoint is computed using that binary string. This statistic is added to the statistics of all the strings in the above sets of races which match the binary string. This continues until one of the pairs of racers becomes significant (i.e., when we believe with probability $1-\delta$ that one member of the significant pair beats its competitor). Then the next iteration of the race begins with a new set of racers which all have the winning field of the previous race switched on. If, in the above race, $\star1\star\star\star$ became significantly better than $\star0\star\star\star$ the next iteration would have

$11\star\star\star$	races against	$01\star\star\star$
$\star11\star\star$	races against	$\star10\star\star$
$\star1\star1\star$	races against	$\star1\star0\star$
$\star1\star\star1$	races against	$\star1\star\star0$

This may be preferable to our other hill climbing racers for three reasons:

- If any feature is outstandingly good, it will be detected quickly without having to wait for an entire iteration of hill climbing to take place.
- If several features are independently good then one of them will be quickly selected, without having to wait to determine which precisely is the best, which is a weakness of FOR-RACE and BACK-RACE.
- Small, mutually dependent, families of features that would be missed by the other hill climbers may be found. If features 1, 2 and 3 must all be on to gain any benefit, then schemata string $1\star\star\star\star\star$ will eventually win a race against $0\star\star\star\star\star$ because 25% of the strings generated from the former have features 1, 2, and 3 all on, whereas 0% of the latter do.

6.2 Experiments

We have run all these algorithms on fifty-six randomly generated synthetic datasets. The task was to find the set of features that minimized the leave-one-out cross-validation error of a 1-nearest-neighbor function approximator. The datasets all had between 4 and 12 inputs and one real-valued output which was a noisy multivariate function of a random subset of the inputs. All inputs were randomly generated uniformly in the range $-1 \leq x_i \leq 1$. The multivariate function was generated from the syntax in Table 2.

The number of terms in the dataset was also randomly decided, and varied between 5 and 30. Thus some datasets were trivial, such as output $= \frac{1}{2}(x_2 + x_7)$, and others complex, such as

output = max(corrupt(product(mean(g(x4) , g(corrupt(x2))), product(x5 , corrupt(x5)))) , g(corrupt(g(g(g(g(corrupt(max(x5 , corrupt(x3)))))))))))

Table 2: Syntax Of the Multivariate Functions In Experiments

expr::=	X_i	the ith input
	$\mathbf{expr} \times \mathbf{expr}$	the product of the subexpressions
	$\mathrm{mean}(\mathbf{expr}, \mathbf{expr})$	the mean of the subexpressions
	$\max(\mathbf{expr}, \mathbf{expr})$	the maximum of the subexpressions
	$\mathrm{corrupt}(\mathbf{expr})$	Gaussian random noise of ± 0.1 is added to the value of the subexpression
	$g(\mathbf{expr})$	where g is a non-differentiable function $g(x) = -x^2$ if $x > 0$ and $g(x) = -x^2 - 2x$ if $x \leq 0$.

It is interesting to note that all the searchers managed to identify the precise set of relevant inputs for this complex dataset, which had 950 datapoints[3]. In all the experiments, $\delta = \gamma = 0.001$.

Figure 6 shows the performance of the forward searchers and schemata on the 56 random datasets. There are two measures of performance.

- **Accuracy.** How often do the searchers end at suboptimal solutions? This is shown by the columns
  - P_{IP} = the percentage of datasets for which the searcher produced an "imperfect" result. A result is imperfect if any other search produced a feature-set with a lower **LOOCV** error.
  - P_{FW} = the percentage of datasets for which the result was fairly wrong, i.e. had a **LOOCV** error more than 0.001 greater than the minimum found by any other search. To give this number some meaning, the minimum **LOOCV** errors found were typically in the range $0.01 - 0.2$ depending on the dataset, with a similar magnitude of variation.
  - P_{VW} = the percentage of datasets for which the result was very wrong, i.e. had a **LOOCV** error more than 0.01 greater than the minimum found by any other search.

[3]The slowest method was FOR-SEL, needing 19000 evaluations, and the fastest was SCHEMATA+, described shortly, which needed 2215 evaluations

- **Search time.** This is given by the number of individual evaluations of **LOOCV** errors. The mean figure is shown, but this is dominated by the few hard datasets that required tens of thousands of evaluations for all methods. Many other datasets required only thousands, or in some cases, hundreds of evaluations. For this reason, also shown (by scatterplots) are the distributions of the ratio of number of samples needed compared with the number of samples needed by the conventional forward sequential selection method. As can be seen, this distribution is highly skewed, especially for the schemata searches. 50% of the schemata searches took less than a quarter of the time of the conventional search. 12 of the 56 schemata searches took over twice as long, an issue that we discuss below.

Figure 7 is a similar table comparing the various backward methods and the same schemata searches (which have no forward or backward biases).

The forward and backward racing methods were usually faster than the conventional methods with little loss of accuracy. The Gauss-Seidel races were similar in performance. The schemata search was also roughly equal in accuracy, except that in this set of experiments it achieved the distinction of no "very wrong" errors.

Of the twelve schemata searches that were twice as long as the conventional forward method, seven were due to the conventional method quickly becoming stuck with an inferior solution—after considerably more computation schemata search found a better result. Of the other five, four eventually found equally good solutions and one found a slightly inferior solution. Interestingly the schemata searches frequently found all the relevant features very fast, often in a tenth of the total time of their search. This produced strings with only 1's and $\star$'s in them. Unfortunately, they would then spend a very long time convincing themselves that they were justified in putting 0's elsewhere.

As an initial and ugly attempt to address this, we tried an additional algorithm, SCHEMATA+, which would give up and replace $\star$'s with 0's very eagerly. If 2000 iterations of one of its races produced no significant winners, it forced one of the stars to zero (using the race statistics to choose the input least likely to be relevant). SCHEMATA+ was the fastest algorithm at converging but was less reliable than SCHEMATA.

Method	Mean # Evals	P_{IP}	P_{FW}	P_{VW}	Mean relative evals	Median relative evals
FOR-SEL	17378	17.9	16.1	7.1	1	1
FOR-RACE	6658	19.6	17.9	5.4	0.51	0.43
FOR-GS-RACE	7960	17.9	14.3	5.4	0.67	0.53
SCHEMATA	7779	16.1	8.9	0	1.7	0.22
SCHEMATA+	4159	19.6	14.3	5.4	0.9	0.13

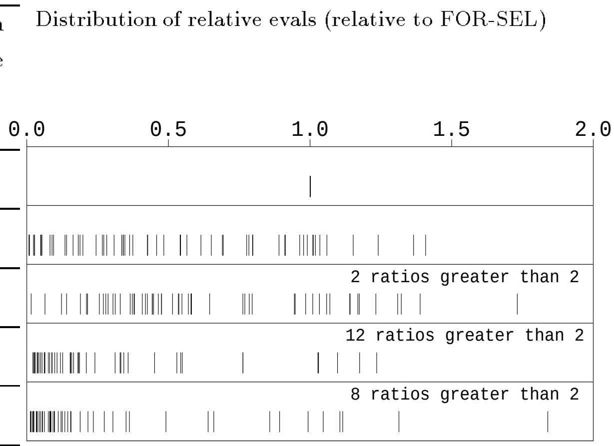

Figure 6: Comparing the conventional forward sequential selection algorithm against its racing counterparts, and against schemata search (which has no forwards-backwards bias).

7 Conclusion

In this paper, we presented the racing algorithm and some extensions to it, specifically in the framework of model selection. However, racing can be thought of as another searching technique such as A* or hill climbing. It can therefore be used in the many of the applications of search in Machine Learning and Artificial Intelligence. It seems to be especially useful for search in spaces that are riddled with local minima, or spaces in which the collection of models is discretized. The speedup over exhaustive search, and the algorithm's ability to quantify the confidence in its result make it a viable alternative to existing heuristic search techniques.

7.1 Related Work

We were motivated to find efficient techniques for model selection by working on Moore's GMBL system [Moore *et al.*, 1992]. Haussler's work on generalizations of the PAC model [Haussler, 1992] provided a treasure of ideas that, combined with Kaelbling's confidence bounds [Kaelbling, 1990], generated the Hoeffding races algorithm. The idea of parallel decision making has been used repeatedly in AI, but the combination of racing with lazy learners gives us the ability to make decisions efficiently, soundly, and incrementally. The notion of racing shows up in related forms in multi-armed bandit problems. To give one example, Rivest and

Method	Mean # Evals	P_{IP}	P_{FW}	P_{VW}	Mean relative evals	Median relative evals
BACK-EL	26358	16.1	14.3	7.1	1	1
BACK-RACE	13943	16.1	12.5	7.1	0.62	0.57
BACK-GS-RACE	5611	17.9	16.1	5.4	0.46	0.23
SCHEMATA	7779	16.1	8.9	0	2.1	0.18
SCHEMATA+	4159	19.6	14.3	5.4	1.0	0.11

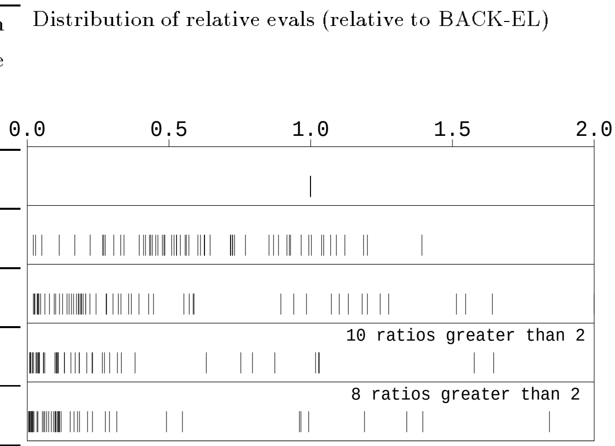

Figure 7: Comparing the conventional backward elimination algorithm against its racing counterparts, and against schemata search (which has no forwards-backwards bias).

Yin [Rivest and Yin, 1993] give a heuristic for picking which arm to pull next according to the probability distributions of the payment of each arm.

An extremely thorough survey and investigation of how the k-armed bandit problem and its solution can be used to minimize wasted computation can be found in [Gratch, 1994]. In earlier work, [Gratch *et al.*, 1993] uses a related method for choosing appropriate search rules in very large scheduling domains.

Similar questions can be found in the earlier statistics literature. This work is chiefly concerned with detecting an effect in a new drug or industrial process with as few experiments as possible. [Box *et al.*, 1978] is an excellent introduction to this field.

Greiner has independently developed a PALO (Probably Approximately Locally Optimal) algorithm [Greiner and Jurisica, 1992] that also uses Hoeffding's bound to decide when one point is better than another. The main differences between our work is that he uses his method for gradient descent, and that he is not trying to select among models, but among Horn clauses and default rules. It is possible to use a descent method in a discretized space by picking among a finite number of discrete gradients. At each iteration, the PALO algorithm compares the estimated error of stopping against the estimated error of each one of the neighboring possibilities. The algorithm then goes in the direction of the least error. The comparison is made faster by using a racing technique.

7.2 Future Work

There are many ways to extend this research. We describe a few of them.

Hoeffding's bound was chosen since it is a tight, distribution-free statistical bound. However, it might not be the best bound to use for classification problems. Other tests, such as the f-test, can be used in its place. For a given value of δ (defining the confidence level at which we are prepared to cut off a competitor from the race) the Bayesian approach cuts off far earlier than the Hoeffding approach. The Bayesian approach achieves its superiority by making stronger assumptions about the distribution of the errors, and so might be expected to be less robust than the almost assumptionless Hoeffding approach. In the experiments we have performed to date the Bayesian method does not seem to converge to the wrong model more easily than the Hoeffding approach, but this is an empirical observation that may not be true for all datasets.

Another issue concerns our choice to assume a normal distribution for the errors. In future work it might be more sensible to use a different distribution, such as an exponential or chi-squared. The normal distribution does not take into account the possibility of occasional highly aberrant datapoints (outliers) in the dataset that might have errors very many times greater than the root-mean-square error.

An exciting offshoot of this research is a search for an 'ideal' collection of models. There is not a single model that is a solution to every problem in the world, but it is possible that there is a group of models which will cover a large portion of problems, and only a small number of which are suitable for each problem. This set of models needs to stretch across the space of possible problems without much overlap. This can be thought of as a basis set of vectors that are orthogonal and span the space. Once we have a basis set of models, then racing is the perfect tool for finding the best model out of this idealized collection since, for any given problem, only a few of the models will perform well on it. That is the ideal condition for racing.

Finally, there is the danger of overfitting by searching too many models. If one looks at data long enough, one will begin to see patterns even if it is completely random (this overly optimistic estimate of the quality of the model is known as *data mining*). If racing is used, it would be statistically wise to use yet another test set at the end of the race to make sure that the chosen model really has its purported error. In addition, users of Hoeffding or Bayesian races should keep in mind Bonferroni's advice that the more models you throw at the data, the better you should expect to perform. If that is not the case, you are probably mining.

References

[Aha, 1990] D. W. Aha. A Study of Instance-Based Algorithms for Supervised Learning Tasks: Mathematical, Empirical and Psychological Evaluations. PhD. Thesis; Technical Report No. 90-42, University of California, Irvine, November 1990.

[Atkeson, 1990] C. G. Atkeson. Memory-Based Approaches to Approximating Continuous Functions. In *1990 Workshop on Nonlinear Modeling and Forecasting.* Adison-Wesley, 1990.

[Box *et al.*, 1978] G. E. P. Box, W. G. Hunter, and J. S. Hunter. *Statistics for Experimenters.* Wiley, 1978.

[Caruana and Freitag, 1994] R. A. Caruana and D. Freitag. Greedy Attribute Selection. In W. W. Cohen and H. Hirsh, editors, *Machine Learning: Proceedings of the Eleventh International Conference.* Morgan Kaufmann, 1994.

[Cleveland *et al.*, 1988] W. S. Cleveland, S. J. Devlin, and E. Grosse. Regression by local fitting: Methods, properties, and computational algorithms. *Journal of Econometrics*, 37:87–114, 1988.

[Dasarathy, 1991] B. V. Dasarathy. *Nearest Neighbor Norms: NN Patern Classifaction Techniques.* IEEE Computer Society Press, 1991.

[Efron and Tibshirani, 1991] B. Efron and R. Tibshirani. Statistical Data Analysis in the Computer Age. *Science*, 253:390–395, 1991.

[Fix and Hodges, 1951] E. Fix and J.L. Hodges. Discriminatory Analysis: Nonparametric Discrimination: Consistency Properties. Project 21-49-004, Report Number 4., USAF School of Aviation Medicine, 1951.

[Goldberg, 1989] D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, Reading, MA, 1989.

[Gratch *et al.*, 1993] J. Gratch, S. Chien, and G. DeJong. Learning Search Control Knowledge for Deep Space Network Scheduling. In *Proceedings of the 10th International Conference on Machine Learning.* Morgan Kaufmann, June 1993.

[Gratch, 1994] J. Gratch. An effective method for correlated selection problems. Department of Computer Science Technical Report Num. 1893, University of Illinois at Urbana-Champaign, 1994.

[Greiner and Jurisica, 1992] R. Greiner and I. Jurisica. A statistical approach to solving the EBL utility problem. In *Proceedings of the Tenth International conference on Artificial Intelligence (AAAI-92).* MIT Press, 1992.

[Haussler, 1992] D. Haussler. "decision theoretic generalizations of the pac model for neural net and other learning applications". *Information and Computation*, 100:78–150, 1992.

[Hoeffding, 1963] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58:13–30, 1963.

[John *et al.*, 1994] G. H. John, R. Kohavi, and K. Pfleger. Irrelevant features and the Subset Selection Problem. In W. W. Cohen and H. Hirsh, editors, *Machine Learning: Proceedings of the Eleventh International Conference*. Morgan Kaufmann, 1994.

[Kaelbling, 1990] L. P. Kaelbling. Learning in Embedded Systems. PhD. Thesis; Technical Report No. TR-90-04, Stanford University, Department of Computer Science, June 1990.

[Kirkpatrick *et al.*, 1983] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[Kreider and Haberl, 1993] J. F. Kreider and J. S. Haberl. The great energy predictor shootout, June 1993.

[Lowe, 1995] D. G. Lowe. Similarity metric learning for a variable-kernel classifier. *Neural Computation*, 7:72–85, 1995.

[Maron and Moore, 1994] O. Maron and A. W. Moore. Hoeffding Races: Accelerating model selection search for classification and function approximation. In J. D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems 6*. Morgan Kaufmann, April 1994.

[Maron, 1994] O. Maron. Hoeffding Races: Model Selection for MRI Classification. Masters Thesis, Dept. of Electrical Engeineering and Computer Science, M.I.T., May 1994.

[Miller, 1990] A. J. Miller. *Subset Selection in Regression*. Chapman and Hall, 1990.

[Moore and Lee, 1994] A. W. Moore and M. S. Lee. Efficient Algorithms for Minimizing Cross Validation Error. In William W. Cohen and Haym Hirsh, editors, *Machine Learning 11*. Morgan Kauffman, 1994.

[Moore *et al.*, 1992] A. W. Moore, D. J. Hill, and M. P. Johnson. An empirical investigation of brute force to choose features, smoothers and function approximators. In S. Hanson, S. Judd, and T. Petsche, editors, *Computational Learning Theory and Natural Learning Systems, Volume 3*. MIT Press, 1992.

[Moore, 1992] A. W. Moore. Fast, robust adaptive control by learning only forward models. In J. E. Moody, S. J. Hanson, and R. P. Lippman, editors, *Advances in Neural Information Processing Systems 4*. Morgan Kaufmann, April 1992.

[Murphy and Aha, 1995] P. M. Murphy and D. W. Aha. UCI repository of machine learning databases. For more information contact ml-repository@ics.uci.edu, 1995.

[Omohundro, 1993] S. Omohundro. Private communication, 1993.

[Press *et al.*, 1992] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C: the art of scientific computing*. Cambridge University Press, New York, second edition, 1992.

[Rivest and Yin, 1993] R. L. Rivest and Y. Yin. Simulation Results for a new two-armed bandit heuristic. Technical report, Laboratory for Computer Science, M.I.T., February 1993.

[Schaal and Atkeson, 1993] S. Schaal and C. G. Atkeson. Open loop stable control strategies for robot juggling. In *Proceedings of IEEE conference on Robotics and Automation*, May 1993.

[Schmitt, 1969] S. A. Schmitt. *Measuring Uncertainty: An elementary introduction to Bayesian Statistics.* Addison-Wesley, 1969.

[Skalak, 1994] D. B. Skalak. Prototype and Feature Selection by Sampling and Random Mutation Hill Climbing Algorithms. In W. W. Cohen and H. Hirsh, editors, *Machine Learning: Proceedings of the Eleventh International Conference.* Morgan Kaufmann, 1994.

[Weiss and Kulikowski, 1991] S. M. Weiss and C. A. Kulikowski. *Computer systems that learn: Classification and prediction methods from statistics, neural nets, machine learning, and expert systems.* Morgan-Kaufmann, San Mateo, CA, 1991.

[Welch, 1937] B. L. Welch. The significance of the difference between two means when the population variances are unequal. *Biometrika*, 29, 1937.

[Zhang *et al.*, 1992] X. Zhang, J.P. Mesirov, and D.L. Waltz. Hybrid system for protein secondary structure prediction. *Journal of Molecular Biology*, 225:1049–1063, 1992.