Hoeffding Races:
Model Selection for MRI Classification

by

Oded Maron

Sc.B., Brown University (1992)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1994

Author .....
Department of Electrical Engineering and Computer Science
May 1994

Certified by .....
Tomás Lozano-Pérez
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by .....
Frederic R. Morgenthaler
Chairman, Departmental Committee on Graduate Students

Hoeffding Races:

Model Selection for MRI Classification

by

Oded Maron

Submitted to the Department of Electrical Engineering and Computer Science
on May 1994, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science and Engineering

Abstract

Model selection can be thought of as meta-learning — it is the problem of finding the best model among a group of learning methods. To help avoid overfitting, this is normally done by cross validation. However, cross validation is computationally intensive, especially if the number of models or the number of training points is high. Methods such as gradient descent have problems such as local minima, or even worse, are not even applicable in some model spaces. In this thesis, I will develop a technique called Hoeffding Races for quickly discarding bad models and concentrating the computational effort at differentiating between the better ones, thereby reducing the number of cross validation queries. In addition, I will apply this technique to the problem of segmenting 3D magnetic resonance brain images and MS lesion detection. Despite the huge amount of data, I will show that learning algorithms such as memory based methods (variants of nearest-neighbor) can cope with problems that have required traditional vision techniques in the past.

Thesis Supervisor: Tomás Lozano-Pérez
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

First of all I have to thank my advisor, Tomás Lozano-Pérez, for picking me up when I was thrashing around looking for a home. In retrospect, his advice on finding a real application for Hoeffding Races helped this thesis immeasurably.

I also thank Andrew Moore and Chris Atkeson for taking me under their wings during my first year at MIT. I caught their enthusiasm for memory based methods and it has proved to be an area with much room for cool computational tricks.

Once I managed to convince Chris Atkeson, Ron Rivest, and Grace Wahba that Hoeffding Races are actually sound, I was a lot more confident about doing this work. Thanks for not shooting me down.

Many people helped me clean up some of the rough edges around this thesis: Greg Galperin, Marina Meila and Stephen Omohundro helped with the proof. Gil Ettinger and Sandy Wells helped with understanding the brains that I was swamped with. Carl de Marcken showed me the applications of this work to natural language understanding and discussions with Bud Baggett led me to believe that there is nothing here related to graphics. Michael de la Maza and Leslie Kaelbling provided many helpful and insightful comments. Thanks to Andy Gavin and Jason Rubin for making this work come alive by giving it some blood and guts.

The Surgery Planning Lab at Brigham and Women's Hospital provided the images and many computing resources. Special thanks go to Diane Doolin, who gracefully played the part of Igor when I needed "more brrrrains for my experrrriments". Without her, I would still be looking for data. Ron Kikinis also deserves thanks for trying to show me that there's more to care about than just error rates, even though I remain unconvinced.

Finally, thanks to all the people at the AI Lab who kept me sane during the past two years, gang, how-to-solve-AI, and most importantly, Holly Yanco.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. SUpport for the laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency

of the Department of Defense under Office of Naval Research contract N00014-91-J-4038.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Imagine that you are a world famous brain surgeon, flying across oceans to perform life saving operations and win the adorations (and money) of tearful and thankful relatives. Your schedule has become so crowded, however, that you are no longer able to spend appropriate time with every patient's records before the operation. What you need is an intern who will go over the patient's MRI (a 3-D picture of the brain) and tell you, as you rush from one operating room to another, where the deadly lesions and tumors are located. To find the best person for this job, you collect everyone in your hospital (even the cleaning staff — who knows who will make a good intern?) and teach them how to find lesions from pictures of slices of a brain. In order to find the best person of the bunch, you test them on a few example brains. Teaching only took a few hours, but who has time to grade the attempts of the entire hospital staff, some of which are truly feeble?

This thesis attempts to resolve the surgeon's problem. The solution will involve the rather harsh criterion of tossing out a candidate intern without grading their entire test if they are doing badly at the beginning of the test. This means that we have to spend considerably less time grading than if we were to check every answer of every candidate. On the other hand, using this method results in not knowing the exact grade of every candidate, and more seriously, has the chance of tossing out the best intern early if they happened to do badly at the beginning. By using Hoeffding's bound (a statistical formula which tells us how far we are from the average after n

independent samples) and formalizing the concepts of 'teaching', 'learning', and 'testing', I will show that we can bound the chance of failure of this method. Therefore, we can accelerate the selection process without sacrificing significant accuracy.

Of course, the above situation is merely an oversimplified analogy: in this thesis I instantiate the analogy by having many different computational learning mechanisms and choosing among them. The surgeon is instantiated by the researcher who has a classification or regression problem and needs to find the best learning mechanism for this task. I am therefore not so much concerned with 'learning', as I am with 'meta-learning' — learning how to learn. In the past, the problem of model selection has been dealt with in an ad hoc fashion. This thesis attempts to give an algorithm which can be analyzed formally, yet is efficient enough to make the problem computationally tractable.

1.1 Formalizing 'teaching' and 'learning'

Adults can do at least two wonderful things: they can tell chairs from objects which are not chairs and they can answer questions of the form "what is seven times nine?" as long as the numbers that need to be multiplied are less than ten. Assuming that these are not innate abilities, they must have been taught by an adult and learned by a child. The field of Machine Learning has a unifying formalism, called function approximation, for dealing with the apparently unrelated learning processes of distinguishing chairs and learning the multiplication table.

The function we are trying to learn maps attributes into outputs. If the output is continuous, then the approximation is called a regression; if the output is discrete, the task is called a classification. The attributes can be of any type, so for example, in the multiplication table example they are the two multiplicands; in the chair distinguishing example, finding appropriate attributes is less obvious. Possible candidates include 'number of legs','does it have a back', 'how much weight does it support', etc. The problem of choosing suitable attributes for any given function is a formidable one, and hence is given a specific name — the representation problem.

In this thesis, I will handle this problem in two ways: first, I will approach it from an optimistic perspective which says that we are stuck with what we are given. In other words, leave the attribute selection to the teacher, and assume that the examples given to us by the teacher have been processed in such a manner as to give us exactly the information we need. The other approach is more pessimistic yet more active. We still assume that all the information is located in the attributes, but some of the attributes might be irrelevant, and some might be more important than others. Finding that information is up to the learner.

I will discuss exactly what kind of learners I use in the next chapter. Before I go on, I would like to make sure that I do not give the mistaken impression that this formalism is what really goes on in children's brains when they learn. The multiplication and chair examples are only examples to show how almost any learning problem can be *formulated* as function approximation. However, that is where the analogy ends. There is no attempt made in this thesis to make the formalism biologically feasible. It is simply for purposes of analysis and computation that this model was chosen.

1.2 Formalizing 'testing'

Unfortunately, given any finite number of training points, there are an infinite number of functions which fit them. Figure 1.1 shows a few of the possible functions which fit the 5 training points perfectly. Clearly, it is a bad idea to use the same set of points for teaching as for testing. Problems such as overfitting (getting an overly optimistic error rate) lead to a choice of a learner which performs very badly on new queries. What is needed is a way to test not only rote memorization, but also the ability to generalize. There are two 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.

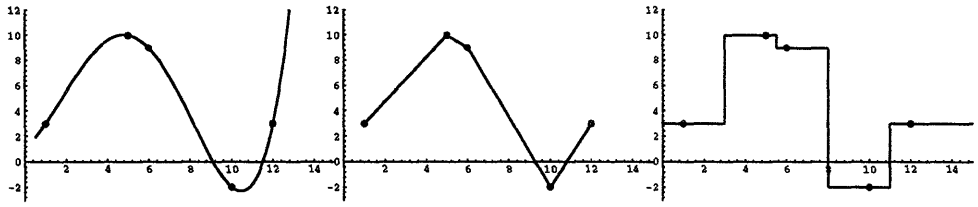


Figure 1-1: Different functions fitting the the same set of points

The second method of estimating generalization is called cross validation. In the case of leave-one-out cross validation, 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 which 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, as I will show in the next chapter, there are certain classes of learners where cross validation is just as cheap as finding the test-set error.

Cross validation turns out to be 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 Hoeffding Races at a glance

Hoeffding Races takes as input a set of models, and returns the ones with the lowest error. It tags each model with an estimated accuracy, which at the beginning is unknown, but as the model is tested on more and more points, becomes closer and closer to the true accuracy. When all the models are given the same point in parallel, and update their individual estimates, they are 'racing'. Hoeffding's bound is a statistical tool, similar to Chebyshev's and Chernoff's bound. By using that bound,

we can tell how close our estimated accuracy is to the true accuracy after seeing only a subsample of the test points. The more points we test on, the smaller this bound gets. The algorithm compares the various models' bounds and removes the models which with high probability have a worse estimated accuracy than all the other models. We can also put some guarantees on the chances of taking a model out of the race that could have won the race. Since the models which we are racing are memory based, it takes no time to train them; therefore, the algorithm optimizes the real cost of these models — the testing time.

1.4 Summary of thesis

This thesis discusses how to perform efficient model selection. I have developed a new technique called 'Hoeffding Races' and I apply it to the problem of finding the best classifier for segmenting MRI data. The thesis is organized as follows:

Chapter two will explore the class of models that I am using for this thesis. I explain the benefits and disadvantages of using memory based learners and their properties. I also discuss the nice computational properties of these models.

Chapter three gives the details of the Hoeffding Races algorithm. I also give a proof of correctness, given some confidence. I show results of running the algorithm on a few medium-sized problems and discuss its effectiveness.

Chapter four discusses the problem of brain segmentation and lesion detection. I compare the various training and testing options and show how the Hoeffding Races algorithm quickly found a good learner despite the size of the problem.

Chapter five talks about future work, related approaches to model selection and MRI segmentation, and summarizes the main ideas of this thesis.

Chapter 2

Memory based learning

Memory based learning is an encompassing name for a variety of statistical and machine learning methods such as nearest-neighbor and local weighted regression. The underlying principle of memory based learning is to simply remember all of the training examples. Variants of the nearest-neighbor method are then used for prediction. Every learning method must perform some sort of memorization (otherwise it would be ignoring its training); memory based learning differs in that the training does not modify any internal parameters with each new training point — it simply stores it in memory.

Let us look at a learner which is not memory based. For example, if we assume that the function we are trying to learn is a quadratic, then we need to learn the three parameters (a,b, and c) in the equation $f(x) = ax^2 + bx + c$. When the program is asked to predict the value at $x = 5$, it plugs in the value of x into the quadratic with the trained parameter values. This is not a memory based learner since we do not keep track of all of the training points, only of the parameters. On the other hand, with a memory based learner, the training points, which are of the form $(x, f(x))$, are kept in memory. When the program 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 value of the function at those points is averaged and returned as the predicted value.

Clearly, we usually do not know that the function we are trying to learn is quadratic, or a polynomial, or piecewise linear, or has a normal distribution. There-

fore, nonparametric methods such as nearest-neighbor are more robust than methods that make assumptions about the parameters which characterize the space of functions. In addition, we can see that nearest-neighbor techniques also fall into a different *computational* category from parametric methods (or even very weakly parametric methods such as neural nets and decision trees). There is negligible computation involved in training (memorizing) and most of the effort goes into finding the neighbors of the queried point. However, there are a few unanswered questions such as:

- How do we measure 'nearness'?
- How does the choice of the number of neighbors influence prediction?
- What method should be used for averaging the outputs of the neighbors?
- How does memory based learning compare to other robust learning methods such as neural networks and decision trees, both in prediction accuracy and computational expense?

In this chapter, I will answer these questions, give a formal description of memory based models, and discuss the differences between these models and other popular machine learning techniques.

2.1 Nearest neighbor and variants

2.1.1 k-nearest-neighbors

> 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 is usually measured by using the Manhattan or Euclidean metric, 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. In this thesis, I present a method for picking the best value for k with respect to prediction error.

Cover and Hart [Cover and Hart, 1967] showed that 1-nearest-neighbor performs at most twice worse than the Bayes error (the error you would get if you knew the exact distribution of each class). As k gets larger, the competitiveness bound gets exponentially smaller. Since this thesis is concerned more with the practical and computational properties of nearest-neighbor, I refer the reader to Dasarathy's collection [Dasarathy, 1991] for more theoretical results.

Weighted distance metric

Despite the fact that I ignored it in the last section, finding an appropriate distance metric is often the toughest hurdle in attempting to use a memory based 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 (height, weight, hat-size). Given a query q of the form (6ft,155LB), we look for the point p which 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 (2ft,155LB) is closer to q than someone who is (6ft,165LB) 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 IQ is predominantly affected by height, and only marginally by weight? In that case, we need to count the height attribute 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 get away from making any assumptions about the best representation for the data. The best weighting, like the best value of k, must be learned.

2.1.2 Kernel regression

It is intuitive that some neighbors are more important than others. Specifically, neighbors which are closer to the queried point should count more toward its classification than neighbors which are farther away. 'Kernel regression' performs weighted averaging (instead of a uniform average) over the space of neighbors. The weighting function which I 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.

$$w_i = \frac{1}{1 + c \cdot (distance(x_i, q)/K_{width})^2} \tag{2.1}$$

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 trying out various values of it. In this thesis, we use the value of $c = 20$.

2.1.3 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 in order 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 the one shown in Equation 2.1.

2.1.4 Preprocessing

In order to reduce the cost of finding a nearest neighbor, researchers have used two types of preprocessing methods. The two methods are organizing the data into a fast-access structure (such as a k-d tree) or eliminating redundant training points (also known as editing). By processing the data before the queries, a speedup in prediction is achieved at the expense of non-negligible training time.

A k-d tree [Preparata and Shamos, 1985] divides the space of points into hyper-ranges which span the space and allow nearest-neighbor queries of almost any size to be completed in $O(\log n)$, where n is the number of training points. This is an improvement over the $O(n)$ behavior of the obvious brute force algorithm. However, there is an $O(n \log n)$ preprocessing cost and O(n) of additional memory is needed to maintain the tree. In addition, k-d trees tend to perform badly if the data is non-uniformly distributed across many dimensions [Moore and Atkeson, 1992].

Editing methods approach the problem by attempting to reduce n, the number of points that need to be searched. There are various approaches, which include throwing out points which are classified correctly, merging points which are similar, and iteratively condensing the data set. For more detailed discussion of these methods, see [Dasarathy, 1991]. In this thesis, I do not do any formal editing; when the training set is too large (hundreds of thousands of points) I simply take a randomly selected subset.

2.2 Example

To give an intuitive notion of how the various memory based learning methods fit a function, I trained some of them on 75 random points from the function $f(x) = 0.01x^3 - 3x^2 - 5x + 4$ in the interval $[-300, 400]$. I also added 10% random noise to all of the training points, which are shown in Figure 2-1. I then took 1000 random points in the interval $[-350, 450]$ and the predictions made by 1-nearest-neighbor,

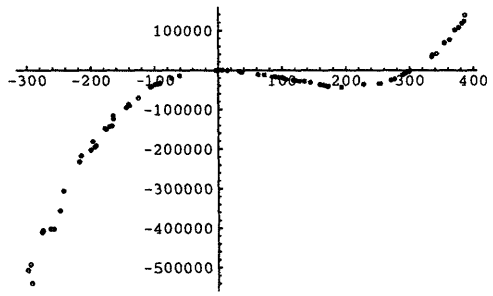


Figure 2-1: 75 random training points with noise

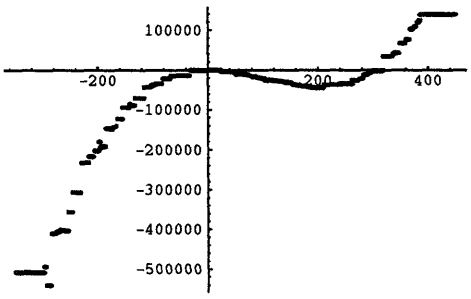
5-nearest-neighbor, kernel regression with $K_{width} = \frac{1}{16}$ and $K_{width} = \frac{1}{4}$, and local weighted regression with $K_{width} = \frac{1}{16}$ and $K_{width} = 1$ are shown in Figure 2-2.

As can be seen, having a kernel width which is too large results in too much smoothing, while using only a few nearest neighbors results in too much prediction noise. Efficient decision-making of which of these functions is best is the topic of this thesis.

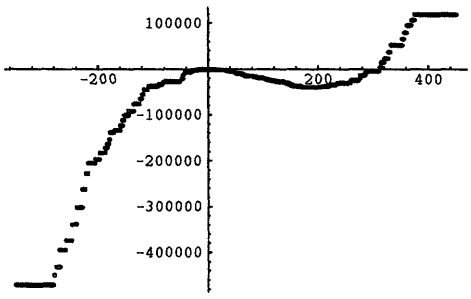
2.3 Pros and cons

Before discussing the benefits of memory based learning methods, I would like to comment on some of the arguments against them. Specifically, in the CART book [Breiman *et al.*, 1984], the authors state that:

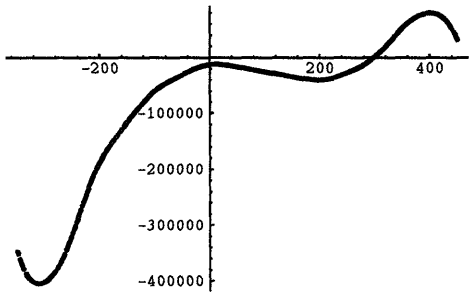
1. They are sensitive to the choice of the metric $\|\mathbf{x}\|$, and there is usually no intrinsically preferred definition.
2. There is no natural or simple way to handle categorical variables and missing data.
3. They are computationally expensive as classifiers; $\mathcal{L}$ must be stored, the interpoint distances and $d(\mathbf{x})$ recomputed for each new point $\mathbf{x}$.



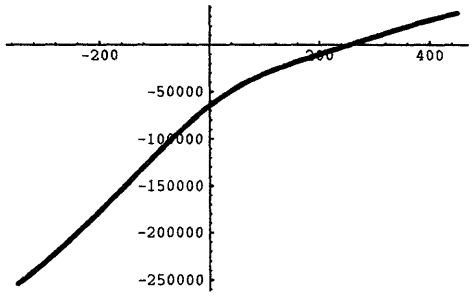
(a) one nearest neighbor



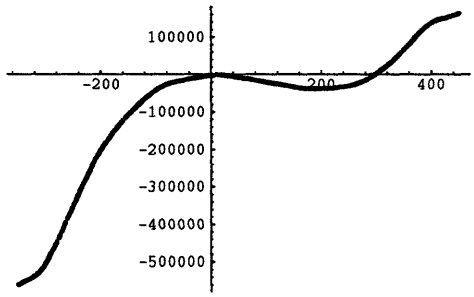
(b) five nearest neighbors



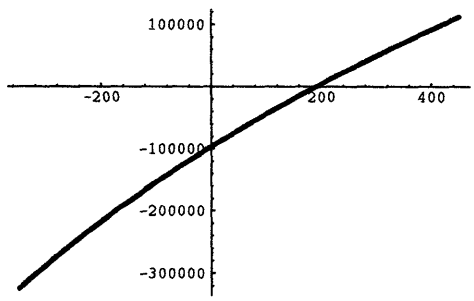
(c) Kernel regression, $K_{width} = \frac{1}{16}$



(d)Kernel regression, $K_{width} = \frac{1}{4}$



(e) Local weighted regression with $K_{width} = \frac{1}{16}$



(f) Local weighted regression with $K_{width} = 1$

Figure 2-2: Testing results on various memory based learning algorithms

4. Most serious, they give very little usable information regarding the structure of the data.

For the first point, I think that there is an intrinsically preferred definition: pick the metric which makes the algorithm have the least generalization error. In the next chapter, I discuss ways of accomplishing that efficiently. For the second point, categorical variables simply have a non-standard (i.e. non-Euclidean) distance function between them. For example, it is simple to assign a specific distance function to the categorical attributes — one which returns a distance of 1 between any two values which are not equivalent, and a distance of 0 between values which are identical. In his thesis, Aha [Aha, 1990] demonstrated this and showed how to handle missing data. As for the third point, computers have become a lot more powerful in the past decade. Memory is cheap, and storing all of the points in a k-d tree results in fairly impressive runtime performance. The computational benefits (discussed below), are in my opinion equal to the cost at prediction time.

Finally, on the point that they deem most serious, these statisticians fall into the same trap as neural-net researchers who create 'Hinton diagrams'. CART creates trees which attempt to minimize generalization error, not maximize understandability. The same goes for the backpropagation algorithm. By looking at the resulting structure, whether in the form of a tree or activation of neurons, we only learn what the algorithm did with the data, not the underlying structure (if there is such a thing). To put it another way, we can think of our a priori guess at the underlying structure as a theory. Theories, by definition, are things which can be disproved. Unless we are willing to toss out the model that CART produces if it does not agree with the theory (no matter how good its predictive accuracy is), or to toss out our theory (no matter how much believe in it), then we cannot honestly attempt to understand the structure of the data. One might be able to achieve both predictive accuracy and understandibility in physics, but in Machine Learning, understandibility comes at the expense of computational tractability and generalization performance.

This thesis is concerned mostly with the computational properties of memory based algorithms, but it is important to note that they hold their own in empirical

comparisons of accuracy to more complicated models such as neural networks and decision trees. Examples of this can be seen in [Aha, 1990] and [Holte, 1993].

The three main advantages of memory based learners is that they are simple, have an obvious representation, and are lazy. At the heart of even the most complex local weighted regression models lies a simple engine of finding the nearest neighbor. There are not many algorithms which can be described in only two words. In addition, unlike other learning methods, there is not an attempt to force the training points into a decision tree, a neural net, symbolic rules, or any formal representation. The points act as their own best representation. A direct implication of this fact is that no work needs to be performed during training — simply store the points in memory. Therefore, memory based learners are lazy; they only do work when queried for prediction of a new point.

In this thesis, I leverage the laziness in two ways. First, computing the leave-one-out cross validation error is just as cheap as computing the test-set error. That is because there is no need to retrain on all of the points but one. We can just 'cover up' that point in memory and that is equivalent to retraining. Therefore, we have a quick, reliable estimator for the error of a memory based model by performing leave-one-out cross validation on all the points in the training set. Second, we can look at many different learners without worrying about the initial expense of training all of them. The only significant computation arises when they are tested. In fact, even if a k-d tree is built, we can use the same tree for learners with different k, different K_{width}, and different regression techniques. The only time when different k-d trees need to be built for different learners is if the distance metric varies.

To conclude, I borrow an analogy from John Pratt. We can think of memory based learners as interpreters and think of neural networks as compilers. Interpreters are simple, give a quick answer, but perform a lot of computation during runtime. Compilers are complex and have a significant start up cost, but their product is more compact and there is less work during runtime.

Chapter 3

Hoeffding Races

This chapter describes a new algorithm called Hoeffding Races that guarantees to find the best model from a set (within a confidence parameter), yet does not require the computational expense of testing all of the models on all of the possible test points. The idea behind the algorithm is that you do not really need many testing queries to find out who the really good models are for a given set of data. Since all we care about is determining the best model, we can stop testing the worst ones after only a few queries and concentrate the computational effort on distinguishing among the better models. We use a statistical tool — Hoeffding's Bound — to decide when a model is significantly bad, to bound how wrong we can be about its performance, and to guarantee with some confidence that this model is indeed bad.

In this chapter, I will first describe two alternative methods of model selection. I will then present the Hoeffding Races algorithm and give a proof of its correctness. Then I procceed to give various refinements to the algorithm and give some results of running it on a number of relatively small problems. I also discuss the relation between Hoeffding Races and descent methods, and speculate on when it is appropriate to race, and when to climb.

3.1 Alternative model selection techniques

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. Hoeffding Races manages to lessen the evils of these techniques, while retaining their benefits.

3.1.1 Brute force

Brute force attacks the problem in the simplest possible way. Given m trained learners and a test set which 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 $O(n \cdot m)$.

3.1.2 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 this surface, and proceeds to 'descend' in the direction which has less error. 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 which 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.

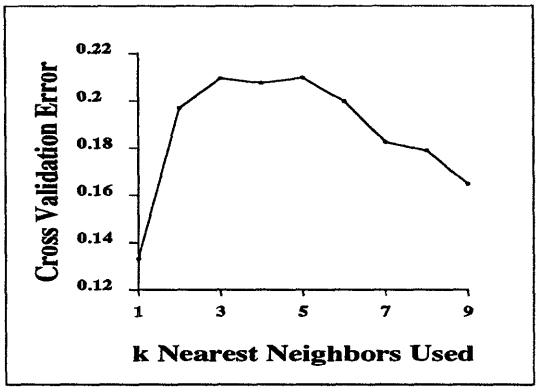


Figure 3-1: A space of models consisting of local-weighted-regression models with different numbers of nearest neighbors used. The global minimum is at one-nearest-neighbor, but a gradient descent algorithm would get stuck in local minima unless it happened to start in in a model where $k < 4$.

To give an example of it in the realm of model selection, let us say that we need to select among nine k-nearest-neighbor learners. They differ in their k (i.e., the number of neighbors they look at in order to make a prediction). The error of each learner is plotted in Figure 3-1. The points are connected so that we can pretend that the surface is continuous. Gradient descent starts at one of the points on the surface and moves toward a learner which has lower error. This method works fine if the initial point is $k = 3, 2$, or 1. However, for any other starting point, the algorithm will get stuck at a local minimum either at $k = 4$ or $k = 9$. 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 over in the last example — namely, there are many instances when it is not even applicable to model selection. Let us go back to the analogy of trying to find the best student in the class, and attempt to use gradient descent for this task. The class is shown in Figure 3-2. We start with Bill and compute his grade; we then look at Bill's neighbors and find that Shaniquah has the best grade out of that group. We compare Shaniquah's grade and the grades of her neighbors and move on to Juan, etc. Clearly, this is a nonsensical method. There is no reason to presume that just because two students happen to sit next to each other in class means that their grades are related (unless they are cheating). More formally, at every step of the gradient descent

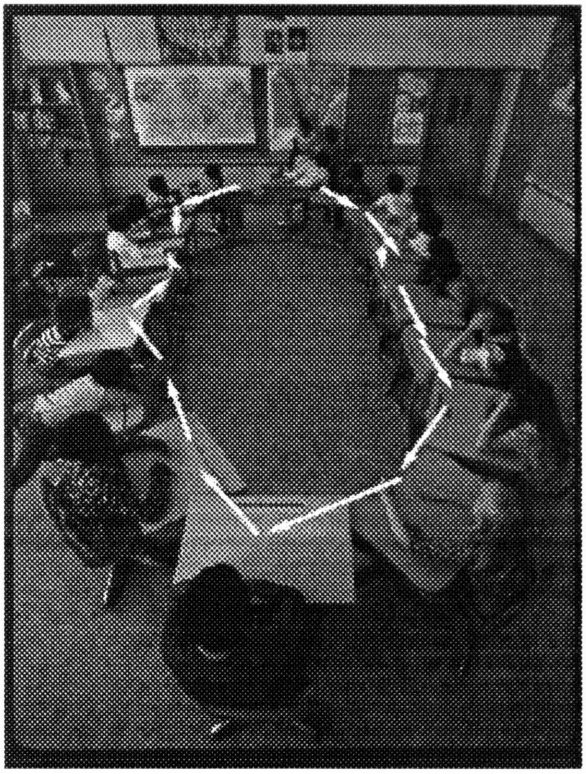


Figure 3-2: An example of gradient descent search for the best student in a classroom

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, as shown dramatically in the classroom example, less obviously in the k-nearest-neighbor example, and even more sneakily in the cases where models are ordered according to simplicity. (See [Schaffer, 1993a] and [Wolpert, 1993] for a more thorough discussion of the misuse of occam's razor). However, just because we can impose a metric 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. Conjugate gradient is just as susceptible to these problem as its less-capable relative.

3.1.3 Genetic algorithms and simulated annealing

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

3.2 Racing

The algorithm derives its name from Hoeffding's formula [Hoeffding, 1963], and from the idea that we want to 'race' all models in parallel and drop the ones which are clearly losing the race for the least error.

3.2.1 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 which 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

$$\Pr(|E_{true} - E_{est}| > \epsilon) < 2e^{-2n\epsilon^2/B^2} \tag{3.1}$$

where B bounds the greatest possible error that a model can make.

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 3.1 gives us an expression for ϵ in terms of n, B, and δ.

$$\epsilon(n) = \sqrt{\frac{B^2 \log(2/\delta)}{2n}} \tag{3.2}$$

Equation 3.2 tells us how close the estimated mean is to the true mean after n points with confidence $1 - \delta$. I will discuss how to obtain a value for B in a later section.

3.2.2 The algorithm

The 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 are not looking at how complex or time-consuming each prediction is, just at 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 points it has been tested upon so far. The algorithm also starts with a test set of size N. For leave-one-out cross validation, we can perform N queries on a training set of size N.

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

- compute the 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 3.2 to calculate how close the current

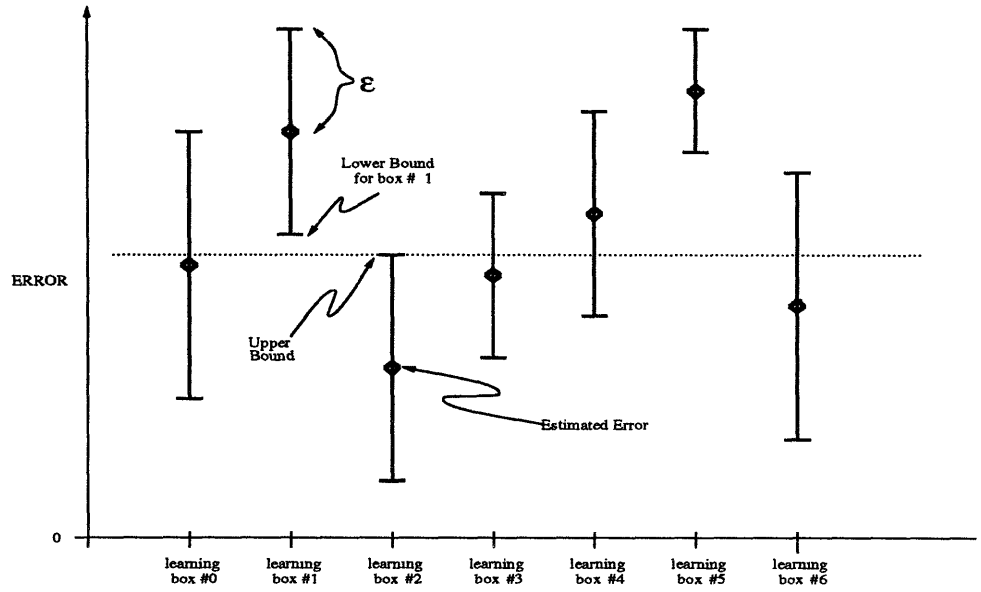


Figure 3-3: An example where the best upper bound of learning box #2 eliminates learning boxes #1 and #5. The size of ϵ varies since each learning box has its own upper bound on its error range, B.

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 3-3. 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.

3.2.3 Ending the race

We iterate and keep picking test points until one of three conditions occur:

1. All but one of the learning boxes have been eliminated. The algorithm simply returns it as the best one.

2. The algorithm can also be stopped once we have picked more than N test points. However, since we are picking the testing points in an independent indentical distribution, that is not a strict requirement.

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 $2 \cdot \epsilon(n)$.

3.3 Proof of correctness

The rash reader would ask why a proof of correctness is necessary since we used a proven statistical tool which should make the algorithm proven with confidence δ as well. However, the careful reader would have noticed that the confidence $1 - \delta$ given in the previous section is not correct for the entire algorithm. $1 - \delta$ 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

$$\Pr\{ \textit{all learning boxes are within } \epsilon(n) \textit{ on all iterations}\} \geq 1 - \Delta \qquad (3.3)$$

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 $Pr\{A \vee B\} \leq Pr\{A\} + Pr\{B\}$.

Let us assume that we have n iterations (we have n points in our test set), and

that we have m learning boxes $(LB_1, \cdots, LB_m)$. We start with the fact that:

$$Pr\{\ a\ particular\ LB\ is\ within\ \epsilon(i)\ on\ a\ particular\ iteration\ i\} \geq 1 - \delta \tag{3.4}$$

Flipping that around we get:

$$Pr\{\ a\ particular\ LB\ is\ wrong\ on\ a\ particular\ iteration\} < \delta \tag{3.5}$$

Using the disjunctive inequality we can say

$$\begin{aligned} Pr\{ \quad & a\ particular\ LB\ is\ wrong\ on\ iteration\ 1\ \vee \\ & a\ particular\ LB\ is\ wrong\ on\ iteration\ 2\ \vee \\ & \cdots \\ & a\ particular\ LB\ is\ wrong\ on\ iteration\ n\} \leq \delta \cdot n \end{aligned} \tag{3.6}$$

Let us rewrite this as:

$$Pr\{\ a\ particular\ LB\ is\ wrong\ on\ any\ iteration\ up\ to\ n\} \leq \delta \cdot n \tag{3.7}$$

Now we do the same thing for all learning boxes:

$$\begin{aligned} Pr\{ \quad & LB_1\ is\ wrong\ on\ any\ iteration\ \vee \\ & LB_2\ is\ wrong\ on\ any\ iteration\ \vee \\ & \cdots \\ & LB_m\ is\ wrong\ on\ any\ iteration\} \leq \delta \cdot n \cdot m \end{aligned} \tag{3.8}$$

or in other words:

$$Pr\{\ some\ LB\ is\ wrong\ in\ some\ iteration\} \leq \delta \cdot n \cdot m \tag{3.9}$$

We flip this to get:

$$Pr\{\ all\ LBs\ are\ within\ \epsilon(n)\ on\ all\ iterations\} \geq 1 - \delta \cdot n \cdot m \tag{3.10}$$

Clearly, Equation 3.10 is the same as Equation 3.3 and we can therefore conclude that $\delta = \frac{\Delta}{n \cdot m}$. When we plug this into Equation 3.2 (our expression for ϵ from the previous section), we pump up ϵ, and thereby ensure the correctness of this algorithm with confidence Δ. The new ϵ is expressed as:

$$\epsilon(n) = \sqrt{\frac{B^2(\log(2nm) - \log(\Delta))}{n}} \tag{3.11}$$

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.

3.4 Refinements

This section presents some speed-ups and patches for the bare-bones algorithm presented above.

3.4.1 Bounding errors

The most obvious obstacle to implementing the algorithm 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

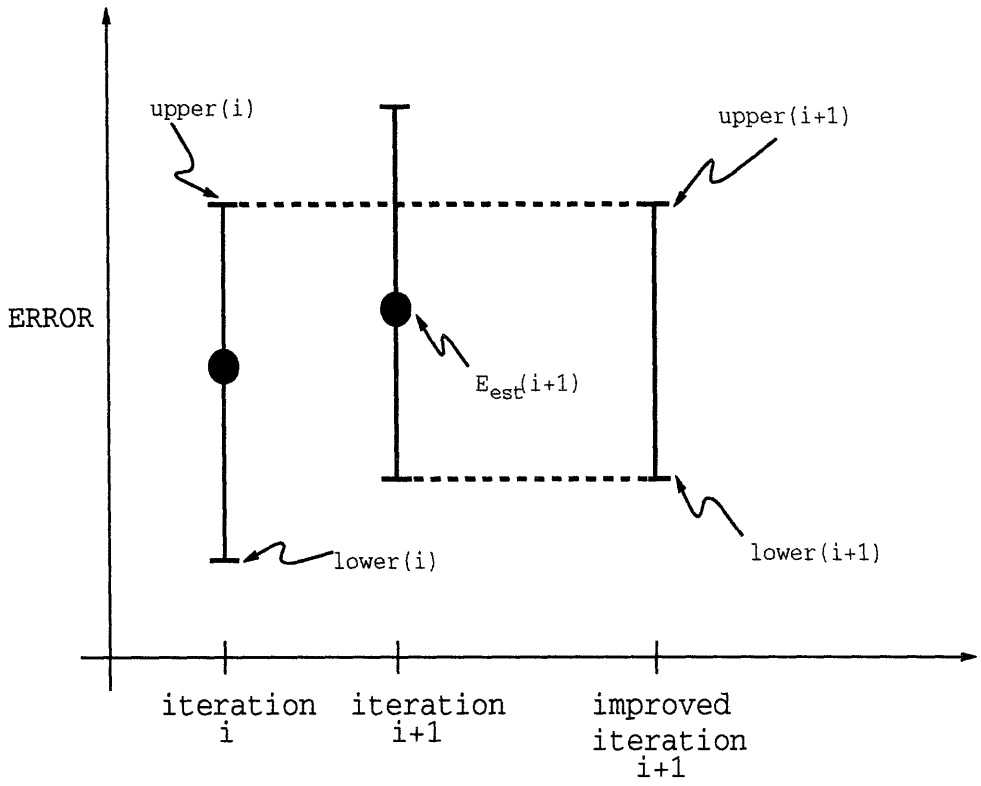


Figure 3-4: Example of how to shrink the interval bounds

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, I only start racing after collecting errors from about 30 points from each learning box. However, using this heuristic invalidates the proof.

3.4.2 Shrinking the intervals

With a little effort, we can actually 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, I will be dropping the superscript, since I 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 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

$$upper(i + 1) = Min(E_{est}(i + 1) + \epsilon(i + 1), E_{est}(i) + \epsilon(i)). \tag{3.12}$$

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

$$lower(i + 1) = Max(E_{est}(i + 1) - \epsilon(i + 1), E_{est}(i) - \epsilon(i)) \tag{3.13}$$

An example of shrinking the intervals is shown in Figure 3-4. The bound of iteration $i + 1$ can be improved based on previous bounds.

3.4.3 Reducing variance

Recently, [Moore and Lee, 1994] added an extension known as *blocking* to the races. Blocking attempts to solve two problems: first, learning boxes with very high variance of error tend to survive the race despite the fact that their mean error could be very high. That is because their variance translates into B, thereby increasing their ϵ and making it impossible to eliminate them. Second, learning boxes with very similar average error but different variances are very hard to distinguish, resulting in the fact that neither one can eliminate the other.

By using blocking, we race not the average errors of each learning box, but differences in errors between learning boxes. Therefore, with m models, we have $\binom{m}{2}$ differences to race. Once the difference between learning box A and learning box B is less than 0 with confidence δ, we can eliminate learning box A. This results in eliminating the effect of error variance on the removal of racers. The reason that this method works is that learners are usually not independent, and when one of them performs badly on a particular point, most of them likely to perform badly as well.

3.5 Examples and results

We ran Hoeffding Races on a wide variety of classification and regression problems which are described below. These results were also presented in [Maron, 1994]. The data files are available from the author.

ROBOT 10 input attributes, 5 outputs. Given an initial and a final description of a robot arm, learn the control needed in order to make the robot perform devil-sticking [Schaal and Atkeson, 1993].

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

ENERGY Given solar radiation sensing, predict the cooling load for a building. This is taken from the Building Energy Predictor Shootout.

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 artificially constructed set of points with many discontinuities. Local models should outperform global ones.

Problem	# points	Initial # learning boxes	# queries with Brute Force
ROBOT	972	95	92340
PROTEIN	4965	95	471675
ENERGY	2444	189	461916
POWER	210	95	19950
POOL	259	95	24605
DISCONT	500	95	47500

Problem	# queries with Hoeffding Races	# learning boxes left	speedup factor
ROBOT	15637	6	5.91
PROTEIN	349405	60	1.35
ENERGY	121400	40	3.80
POWER	13119	48	1.52
POOL	22095	75	1.11
DISCONT	25144	29	1.89

Table 3.1: Results of Brute Force vs. Hoeffding Races.

I have run Hoeffding Races on a collection of memory based 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. The distance metric was not varied. Chapter two describes these learning boxes in more detail. All of the experiments were run using $\delta = 0.01$. We compare the algorithms relative to the number of queries made, where a query is one learning box finding its error at one point. The results are summarized in Table 3.1.

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

- Hoeffding Races never performs more queries than brute force, and its overhead is negligible.

- In all the cases we have tested, the learning box chosen by brute force is also contained by the set returned from Hoeffding Races. Therefore, there is no loss of performance accuracy.

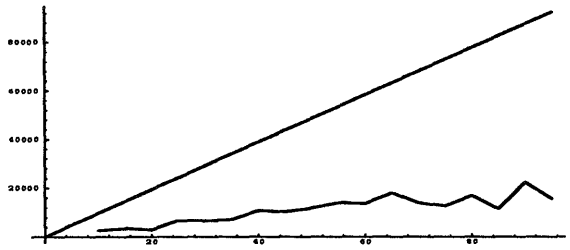


Figure 3-5: The bottom line shows the number of queries taken by Hoeffding Races for the ROBOT problem as the size of the initial set of learning boxes is increased. The top line shows the performance by brute force. At each point, the set of learning boxes was chosen randomly.

- It 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 Hoeffding Races is most effective when there exists a small subset of clear winners within the initial set of models. In fact, it becomes more and more effective (in comparison to brute force) the larger the size of the initial set of models.

In order to test this conclusion, I created random subsets of increasing sizes from the 95 learning boxes used for the ROBOT experiment. I ran Hoeffding Races on each one of the subsets, and tabulated the results in Figure 3-5. 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 — Hoeffding Races lets us do that without much computational expense.

I do not think that there is a general asymptotic relation between Hoeffding Races and brute force as the number of models grow. As shown by Table 3.1, it is problem dependent. However, it is clear that Hoeffding Races works badly on 'boring' prob-

lems, 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. Running the classroom analogy completely into the ground, 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).

I did not compare Hoeffding Races to gradient descent for the reasons given at the beginning of this chapter — descent methods are simply not applicable for selection among these models. Gradient descent might have been applicable if I was trying to find suitable weights (i.e. a good distance metric) for the attributes of a particular model (see [Atkeson, 1990]). Even then, a Hoeffding Race needs to be run between the current weights and the neighboring weights. This technique was used successfully in [Greiner and Jurisica, 1992].

Chapter 4

Segmenting MR data

Now that we have a method for efficiently choosing a classification or regression model, let us try to apply it to a problem in which its efficiency and reliability are really necessary. Segmenting Magnetic Resonance images by tissue type is such a task. I will look at the problem of determining, for every voxel in the image, whether it is gray matter, white matter, white-matter lesion, or cerebral fluid. This chapter is meant to serve mostly as an extensive example of model selection. As with most real world problems, the data needs to be massaged into a form which is appropriate for learning. However, I have tried to maintain syntactic, rather than semantic, transformations of the data. Expecting a learner to normalize and align the data on its own is too much to ask for with today's technology.

The segmentation problem is important in several medical imaging applications. For example, Multiple Sclerosis (MS) causes brain lesions, whose volume needs to be tracked over time in order 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 add them up 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. This chapter describes an attempt to find an automated system that has been trained on expertly segmented brains (i.e. given examples of correct identification of lesions, gray and white matter, and fluid) and

gives good segmentations on new brains.

This is a difficult problem for several reasons. First, it is a real-world problem with noise, human errors, and very little a priori knowledge about which features are important. Second, it is one of the bigger problems (in terms of the number of data points) that has been attempted using these techniques. In fact, understanding images has usually been relegated to the Computer Vision domain, where image specific techniques such as filtering, edge detection, and object recognition have been used. Here I will show how a general technique can be applied in their place. Of course, the main reason that I can do this is that the images are very constrained. I am not attempting to solve the general vision problem using Machine Learning; I am attempting to use general learning techniques to solve a highly constrained vision-like problem. Finally, this is a hard problem because it has yet to be solved. No one has come up with a perfect segmentation algorithm, and I think that is because it is hard to formulate the solution in terms of a simple program or a few simple rules. Therefore, the segmentation problem falls into the class of problems which are more efficiently solved by learning from examples, than by rote programming.

A moderate goal of this project was to be able to segment future scans of the same patient after training on that patient's images. A more ambitious goal was to segment arbitrary patient's images after training on various patients. The attempts are described in the next few sections.

4.1 The training data

Thanks to the MS study headed by Ferenc Jolesz and Ron Kikinis at Brigham and Women's Hospital in Boston, I have MR images from 8 different patients. For each patient, I have two to four scans taken at different times, weeks or months apart. A Magnetic Resonance Image (MRI) is a three dimensional picture of the brain. Each point in the picture contains five pieces of information, and if the image has been segmented, a classification as well. Three of the attributes are the point's (x, y, z) coordinates. The other two attributes are known as the two channels. The first

CLASS	PERCENTAGE
WHITE MATTER	44.7 ± 9.3
LESION	0.8 ± 0.6
FLUID	11.7 ± 4.2
GRAY MATTER	41.9 ± 9.1

Table 4.1: Average percentage of each class in a brain

channel measures the proton density at the point, and the second channel measures the viscosity of the tissue at at that location in the brain. These two attributes are the standard output of the MR machine for Multiple Sclerosis patients at Brigham and Women's hospital. They were determined a priori to be key attributes for tissue classification. The point can fall in one of four classes: gray matter, white matter, MS lesion or cerebral fluid.

The brain images have been processed so that all matter outside of the cranial cavity (such as the skull and the skin) has been removed. The images have been segmented by a semi-automated process [Gerig *et al.*, 1992], in which an expert operator selects 15 to 20 points and classifies them. A program finishes the process by classifying the rest of the points. Clearly this is not the ideal training set. A much better one would have contained brains whose every pixel was hand classified. Unfortunately, those are very expensive to produce since there are over 500,000 pixels in a typical image.

Since the function we are attempting to approximate is six-dimensional (5 attributes and an output classification), it cannot be easily displayed. The standard way of displaying this information is to show horizontal slices through the brain, displaying the two channels and the classification information at each point. A typical slice is shown in Figure 4.1, parts (a), (b), and (c). Each brain consists of 52 to 56 slices. There are about 3.5 million points per brain, but only about 500,000 of them are actually part of the brain. The rest are empty space. The average distribution of each class is shown in Table 4.1.

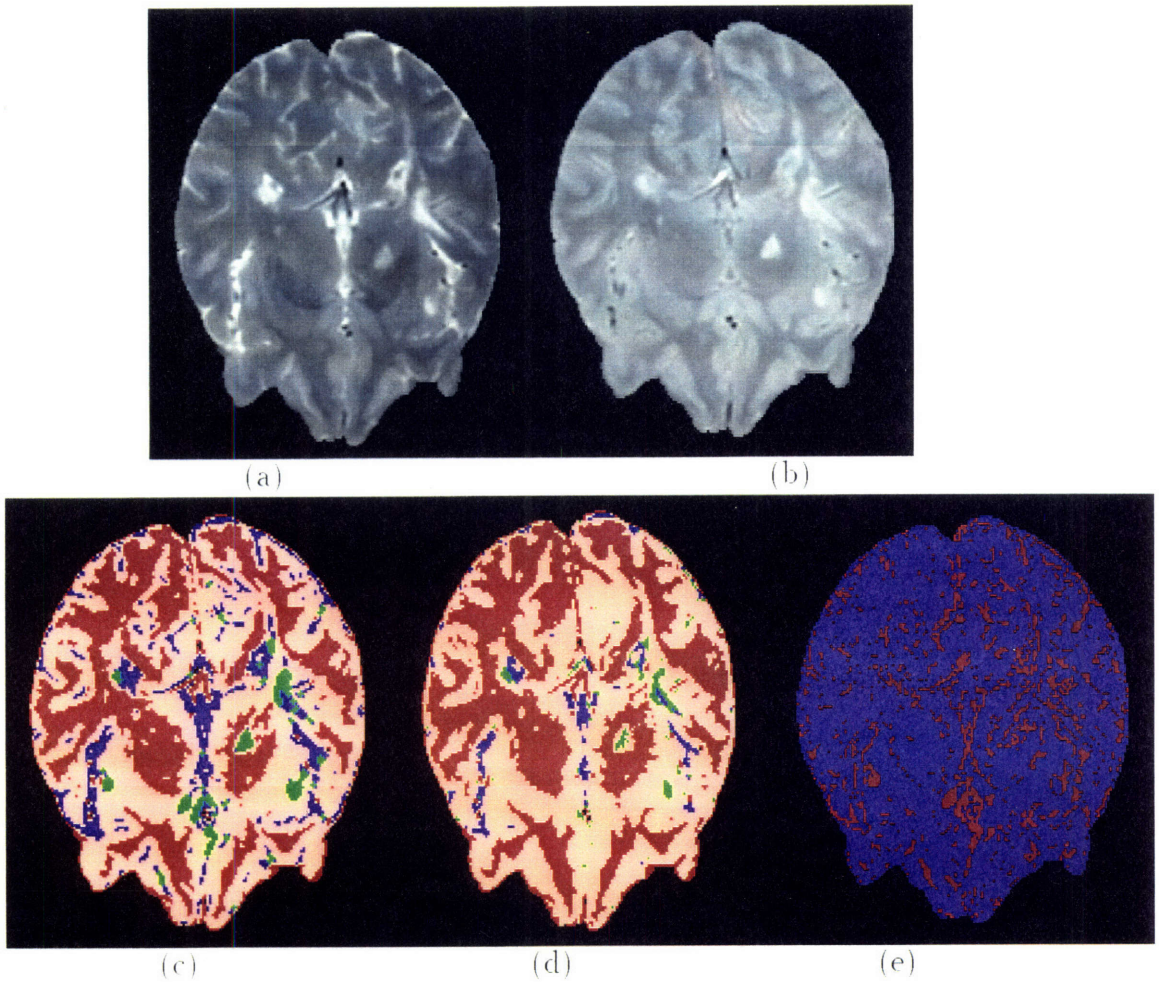


Figure 4-1: (a) and (b) show intensities for the two channels. (c) is the correct classification. Lesions are green, fluid is blue, and white and gray matter is shown in red and salmon. (d) shows the classification performed by a learner which was trained on a different scan from the same person. The learning box uses only intensity information. (e) shows the errors in red.

4.2 Intensity based classification

I was interested in finding out how good a classifier can be found that looks only at the two channels, and ignores the position of each pixel. By using only intensity information, I avoid the problem that the brains need to be aligned (or *registered*) in order to use position information from one brain to predict classifications on another. If brains always have fluid in their center, then it is necessary to align all the brains to be centered on the same coordinates. In addition, patients' heads are tilted at various angles and alignment needs to consider rotations as well as translations. Finally, different patients have brains of various sizes and they therefore need to be scaled.

In order to avoid these problems, we look only at the data from the two channels, but unfortunately, there are normalization problems even here. Due to changes and drifts in the hardware of the MR machine it is possible for the intensity values to vary wildly from image to image, and unfortunately, even within an image. An example of this can be seen in the intensity histograms in Figure 4-2, where the distribution of values for each of the two channels is shown for images of the same patient taken at two different times. The reason for the change in distribution is not the growth of a lesion, but a reconfiguration of the hardware. Attempting to train on the first image and to test on the second image will not work, for the same reason that teaching the multiplication table and asking about chairs will not work. Memory based learning, like most learning models, works under the assumption that the distribution stays constant through both training and prediction.

Half a million training points is a lot of probably redundant information and will make prediction and search for the best model intolerably slow. Therefore, I randomly selected a subset of about 10,000 (non duplicated) points out of each image.

Finally, we can start racing. I start with a collection of 55 models which vary in the number of nearest-neighbors (1,3,5,7 or 9), the value of K_{width} ($\frac{1}{64}$,$\frac{1}{16}$,$\frac{1}{4}$, or 1), and in their choice of weighted regression, kernel regression, local weighted regression, global weighted regression, and k-nearest-neighbors. I used cross-validation to test each model, and I repeated the race for every brain in my collection images. If I

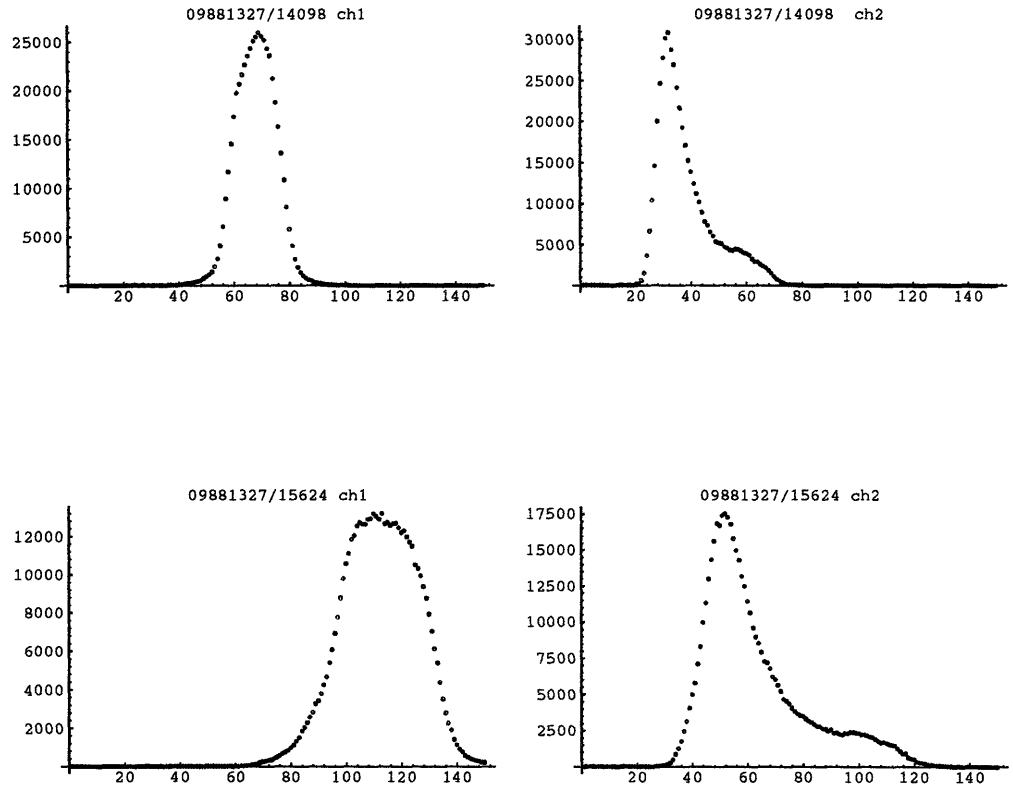


Figure 4-2: Histograms of intensity values of the two channels for the same patient. The top two histograms are for an image before recalibration of the MR machine, and the bottom two are for an image taken after recalibration. The left histogram shows value distribution for the first channel and the right histogram for the second.

PATIENT#–BRAIN#	QUERIES MADE / POSSIBLE QUERIES	LBs LEFT / INITIAL LBs	BEST BOX AND ERROR
9-1	410165 / 617210	30/55	$k = 5$, 2.3%
9-2	413665 / 616825	30/55	$k = 5$, 3.1%
7-1	391860 / 613085	30/55	$k = 7$, 2.7%
7-2	397845 / 600875	30/55	$k = 7, 9$, 5.0%
11-1	377700 / 492250	35/55	$k = 7$, 3.8%
11-2	348285 / 491755	35/55	$k = 7$, 4.1%
3-1	321625 / 523600	30/55	$k = 7$, 2.8%
3-2	380095 / 561385	30/55	$k = 3, 5$, 1.8%
4-1	385300 / 550880	35/55	$k = 1, 3, 5, 7, 9$, 1.4%
4-2	368130 / 529430	35/55	$k = 7$, 2.9%
6-1	359715 / 522500	30/55	$k = 7$, 3.0%
6-2	348905 / 528055	30/55	$k = 5, K_{width} = \frac{1}{64}$, 3.3%
10-1	393800 / 546150	35/55	$k = 7$, 2.5%
10-2	375850 / 555500	35/55	$k = 7, 9$, 2.2%
8-1	356355 / 491040	35/55	$k = 7$, 2.5%
8-2	365440 / 499345	35/55	$k = 9$, 3.2%

Table 4.2: Results of racing by cross-validation on individual images

were to use brute force, I would need to make about 550,000 queries to find the best model for each brain (the number of points per brain subsample varies). However, by using Hoeffding Races I only made 68.8% $\pm$ 3.8% of these queries. At the end of each race, I ended up with 30 or 35 of my original learning boxes. Even though all of these are equivalent to within ϵ, the best learning box of the remaining ones was usually the one which used the 7-nearest-neighbor model, and its average error rate was 2.9% $\pm$ 0.9%. In general, the more global a model was, the worst it performed — the learning boxes left at the end of the race were rather local. In terms of compute time, finding the best model for each brain takes over 4 hours on a SPARC ELC. By using Hoeffding Races, I saved over 32 hours of computer time for this experiment alone. Detailed results are shown in Table 4.2.

The low error rate was verified by out-of-sample testing to make sure that cross validation worked. However, it turns out that testing on different points from the same brain that was used for training is not an accurate measure of the task we are trying to perform. When we attempt to train on one image, and then make predictions about another image, the error rate jumps to over 17%. The following

results show this: I trained on one image, then tested on

- images from different patients (testing across patients)
- the same patient at a different time (testing across time)
- different points from the same image

This was repeated, using a different training image each time. If the test set was from the same image, the error rate was $2.76\% \pm 0.64\%$ (which confirms the original result). If the test set was from an image of the same patient at a different time, the error rate was $17.4\% \pm 6.34\%$. If the test set was from an image of a different patient, the error rate was $18.2\% \pm 7.16\%$. Detailed error rates are provided in Table 4.2, and are summarized in Table 4.4. Error rates for testing on the training image are shown in boldface, and error rates for testing across time are shown in italics.

There are a couple of unexpected results from this experiment. It is surprising that testing across time gives the same error as testing across patients. The results are also very consistent; in other words, there is not one particular image which stands out as being especially good (or bad) to train over.

The key question, though, is why the error rates jumped up when the test points were not taken from the training image? Is it because the two channels simply do not contain enough information to generalize well to new images, or is it because we mined the data for patterns that are not really there by cross-validating to find the best classifier? (a pitfall suggested in [Schaffer, 1993b]) To find out, I raced to find the best classifier again. However, instead of racing for the best cross-validated error, I raced for the best error rate on a different test image. I repeated the race with different training images and 15 different test images. I got the same results (to within half a percentage point) as above on 14 of the test sets. Therefore, the learning boxes which the race converged on are the indeed the best ones, but they can only generalize so well.

One last attempt was to train not on a single image, but on a conglomeration of images. The training set was made up of a combination of three brains from different patients. Again, 55 learning boxes were raced and the winner was a local weighted

TRAINING IMAGE	TESTING IMAGE								
	3-1	3-2	3-3	3-4	4-1	4-2	6-1	6-2	7-1
3-1	**2.63**	*13.71*	*27.81*	*17.35*	10.02	17.71	12.66	25.78	30.03
3-2	*14.73*	**3.48**	*20.43*	*17.57*	11.70	13.21	12.65	20.24	21.38
3-3	*28.23*	*21.25*	**2.15**	*23.29*	26.40	21.51	24.32	6.94	7.46
3-4	*16.90*	*16.16*	*23.86*	**2.84**	16.22	20.75	15.60	19.91	24.77
4-1	11.76	12.18	29.14	19.38	**2.61**	*15.43*	15.26	28.98	29.37
4-2	20.82	15.70	24.94	23.95	*15.57*	**1.15**	19.17	27.35	24.00
6-1	12.05	10.32	21.85	13.84	11.00	12.53	**3.74**	*20.89*	23.75
6-2	24.67	17.96	5.97	19.80	22.84	19.84	*21.38*	**3.06**	8.22
7-1	27.72	19.77	6.86	23.28	24.72	19.95	24.77	8.71	**2.30**
7-2	19.99	16.61	12.54	15.06	19.12	16.45	13.62	10.95	*12.65*
8-1	34.44	24.32	9.33	29.49	28.50	19.01	33.00	14.97	9.05
8-2	24.63	18.11	16.10	21.36	19.67	9.04	23.61	17.56	15.42
9-1	41.35	31.02	14.46	36.40	36.89	30.68	38.05	19.75	15.32
9-2	14.02	8.81	19.57	14.47	11.59	9.45	13.38	18.17	18.97
10-1	32.23	23.40	10.62	27.78	26.47	14.76	30.75	13.92	9.76
10-2	29.43	20.36	8.29	24.73	24.35	17.28	28.50	11.22	9.90
11-1	25.25	19.37	20.45	22.23	19.80	6.98	25.73	21.73	19.07
11-2	16.35	11.41	18.94	16.16	13.52	7.70	16.02	18.86	18.63

TRAINING IMAGE	TESTING IMAGE								
	7-2	8-1	8-2	9-1	9-2	10-1	10-2	11-1	11-2
3-1	21.62	31.14	22.62	37.58	14.24	30.47	28.74	22.89	16.03
3-2	18.41	22.22	17.28	28.02	8.96	22.31	20.77	18.87	12.36
3-3	13.52	7.18	14.81	12.81	19.40	9.10	08.19	20.16	19.72
3-4	15.81	29.81	21.70	33.58	14.13	27.15	25.36	21.60	16.81
4-1	23.39	29.68	21.73	36.62	12.82	28.82	27.83	21.30	15.11
4-2	20.65	18.80	9.56	29.14	11.78	17.02	20.50	8.63	9.86
6-1	12.79	25.34	17.59	31.49	10.21	25.43	25.30	19.83	13.49
6-2	10.93	10.19	13.23	16.23	16.46	9.81	8.59	17.94	17.40
7-1	*13.55*	6.76	13.92	13.19	17.99	7.79	8.70	18.63	18.36
7-2	**3.05**	16.00	10.77	22.16	14.13	15.50	15.96	16.05	15.42
8-1	20.27	**2.57**	*12.51*	12.29	21.55	6.57	7.51	16.40	20.03
8-2	13.27	*11.43*	**2.98**	21.07	12.91	10.89	12.54	6.57	10.40
9-1	26.94	14.03	23.95	**2.17**	*28.57*	17.93	18.29	28.65	28.72
9-2	15.29	20.71	12.37	*26.38*	**3.38**	20.44	19.09	13.04	6.36
10-1	18.32	5.89	10.57	15.73	20.66	**2.15**	*6.32*	13.00	17.53
10-2	17.65	6.63	11.16	14.71	17.72	*5.72*	**2.44**	14.21	15.34
11-1	16.64	14.71	5.38	24.22	13.40	13.14	15.28	**3.30**	*10.41*
11-2	15.29	18.13	9.55	24.58	6.17	17.46	16.22	*10.38*	**3.75**

Table 4.3: Error rates for training on every image and testing on every image

TEST SET	AVERAGE ERROR
same image	$2.76\% \pm 0.64\%$
across time	$17.4\% \pm 6.34\%$
across patient	$18.2\% \pm 7.16\%$

Table 4.4: Summary of errors in testing various images

regression model with $K_{width} = \frac{1}{4}$. This learning box was tested on all the images and the resulting average error rate was $15.95\% \pm 6.2$.

4.2.1 Accuracy for each class and MS lesion detection

In addition to looking at the overall performance of a learning algorithm, it is sometimes informative to look at its performance on each class. Table 4.5 shows the average percentage error on each of the classes. I averaged over 4 types of test sets: all of them, those which were taken from the training image, those from a different time but from the same patient, and those from different patients.

As can be seen, it is very difficult to detect lesions, as over half of them are incorrectly classified. However, since lesions account for less than one percent of the data, it does not greatly effect the overall error rate. What really influences the overall error rate is how well white matter and gray matter are classified, and 10% of all white matter and about 15% of gray matter is classified incorrectly.

There are some applications where lesion detection is much more important than distinguishing between white and gray matter. In all of the above experiments, we have implicitly ignored the lesion detection problem since lesions are such a tiny fraction of both the training and testing set. Therefore, learning boxes which perform badly on lesions are not penalized as much as learning boxes that do well on lesions but badly on white matter.

The other possible cause for performing badly on the lesion class is that there are simply not enough training points or attribute information to make good predictions. There is a simple way to make a certain class more important than other classes without modifying the natural distribution of the data — change the *output distance*

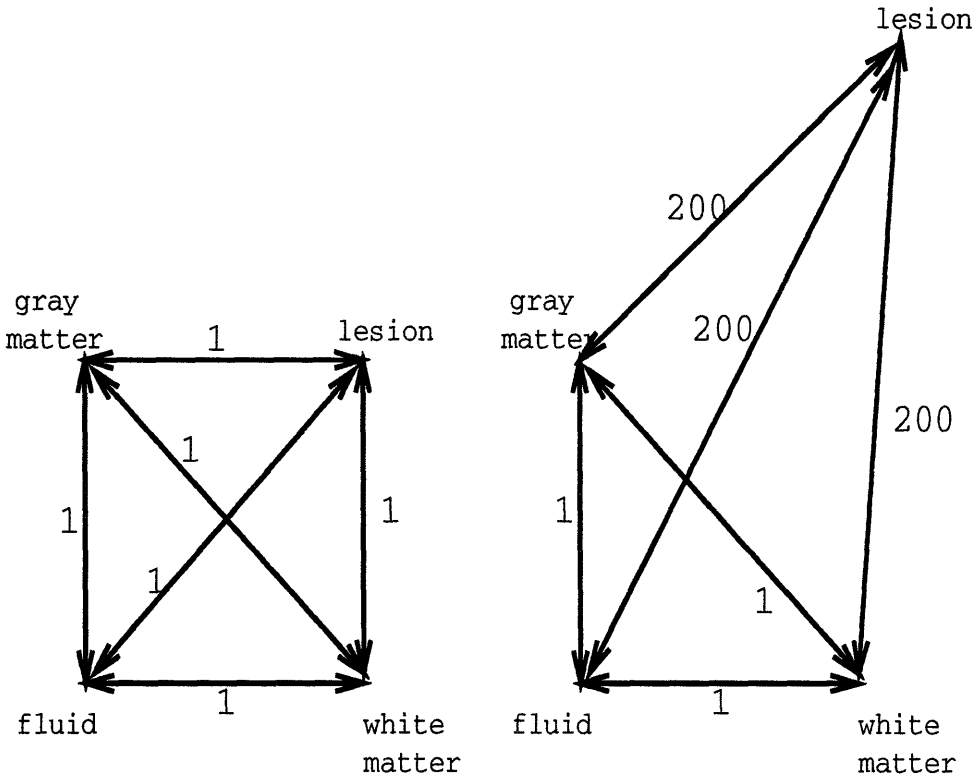


Figure 4-3: Two different output distance metrics

metric. Normally in classification tasks, the distance between any two different outputs is 1. In other words, if the predicted class is correct then the error is 0, and if the predicted class is incorrect then the error is 1. However, we can easily change that to be

$$dist(x,y) = \begin{cases} 0 & \text{if } x = y \\ 200 & \text{if } (x = \textit{lesion} \text{ or } y = \textit{lesion}) \text{ and } x \neq y \\ 1 & \text{otherwise} \end{cases}$$

The two metrics are shown in Figure 4-3.

A side-effect of giving high errors to lesions is that the races tend to be slower. The maximum error for each box (B) is now 200 instead of 1, resulting in a larger ϵ. In fact, the bounds were so large that no learning box was eliminated when I ran Hoeffding Races with the new output distance metric. However, the top models remained the same ones as before.

TEST SETS	WHITE MATTER	MS LESION	CEREBRAL FLUID	GRAY MATTER
all images	$10.5\% \pm 13.5$	$53.04\% \pm 27.45$	$29.32\% \pm 18.18$	$15.26\% \pm 14.26$
same image	$0.959\% \pm 0.528$	$39.86\% \pm 22.08$	$14.90\% \pm 6.76$	$0.756\% \pm 0.243$
across time	$11.01\% \pm 14.02$	$52.00\% \pm 25.0$	$25.04\% \pm 14.89$	$16.40\% \pm 14.52$
across patient	$11.16\% \pm 13.69$	$54.06\% \pm 27.71$	$30.66\% \pm 18.46$	$16.21\% \pm 14.19$

Table 4.5: Errors on each class, given various testing sets

Another possibility is to convert the four classes into just two: lesion or not-lesion. Unfortunately, this performs no better than the previous strategy. Therefore, the problem is that there simply is not enough information to be able to detect lesions effectively, given our collection of learning boxes.

4.3 Combining intensity and location

Due to intensity variations across time, across patients, and even across the same image, we would like to find a model which does not rely solely on intensity. If we can get all of the images to be invariant with respect to translation, rotation, and scaling, then we can use the (x, y, z) information of a pixel to inform us of its classification. As a first stab, I used the registration system developed in [Ettinger *et al.*, 1994]. This system registers brain images by aligning their surfaces. However, problems such as scaling (different patients have brains of varying sizes) and changes in brain structure across time lead to difficulties in using a system which aligns globally, as opposed to a system which locally aligns various key internal structures.

Using this registration system as a first-order approximation for perfectly aligned brain images, we were not able to improve on the error rate of learners which used only intensity information. An example of a predicted image using both intensity and location is shown in Figure 4.3. (a) and (b) are slices from the test brain which was registered to the training image. (c) is the correct classification, and (d) is the classification generated by a model which uses local weighted regression with $K_{width} = \frac{1}{64}$ and a distance metric which counts differences in channel values 16 times

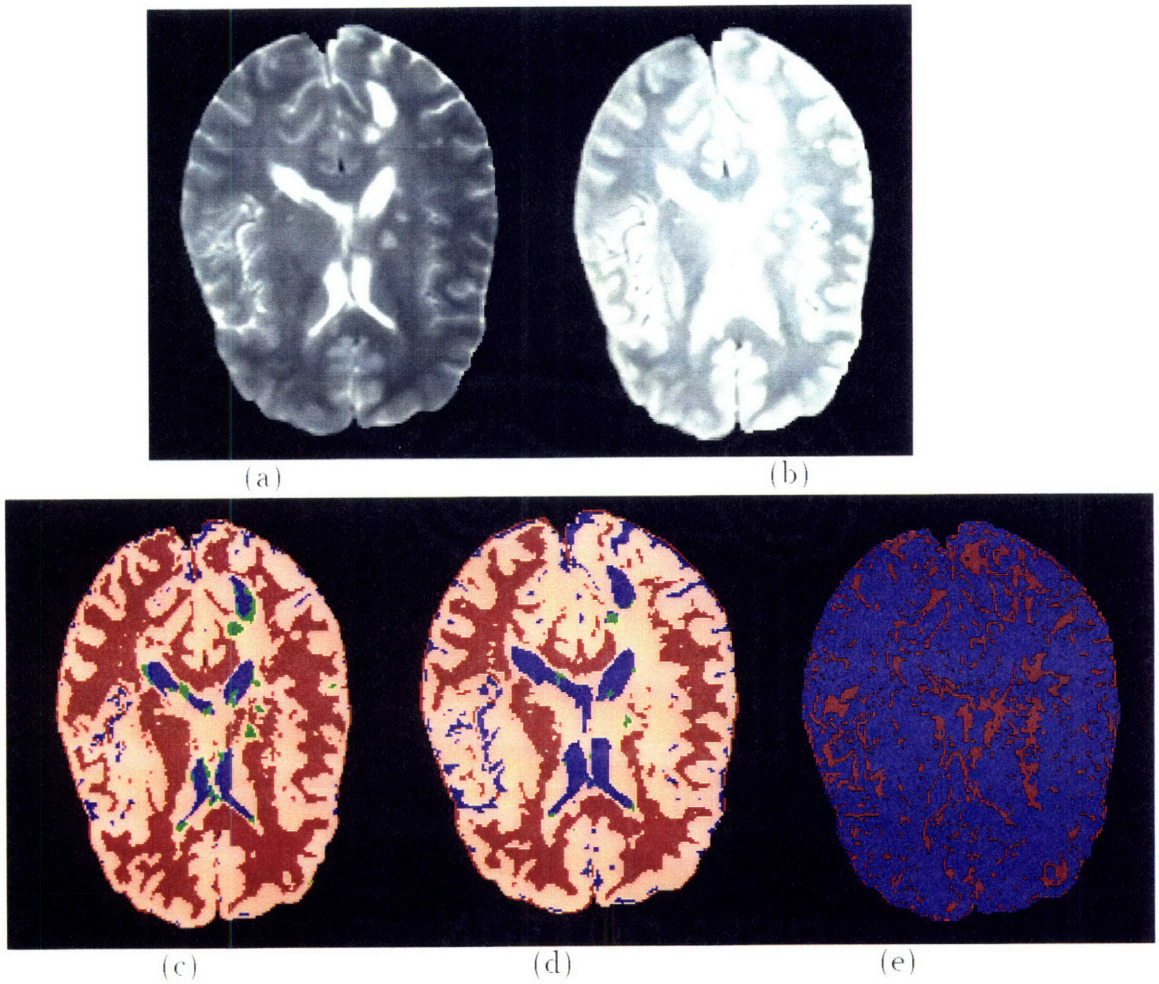


Figure 4-4: (a) and (b) show intensities for the two channels. (c) is the correct classification. Lesions are green. fluid is blue. and white and gray matter is shown in red and salmon. (d) shows the classification performed by a learner which was trained on a different scan from the same person. The learning box uses both intensity and position information. (e) shows the errors in red.

as much as differences in position. The model was trained on an image from the same person taken at an earlier time. (e) shows the error in red for this particular slice. The error rate for the entire image is 19%.

4.4 Future directions in automating MRI segmentation

There are still several crucial obstacles to overcome in order to make automatic segmentation of MR images a mainstay of radiology. From the machine learning perspective, the most important obstacle is the lack of a large standardized corpus of example segmentations. The training images used in this thesis were generated by a semi-automated process. However, we would like to learn the bias of experts, not the bias of a different computerized system. There exists one computerized atlas of the brain which gives tissue and functional segmentation. However, that brain is healthy, and even worse, there is only one of it. We need a collection of brains, both healthy and diseased, where every point on every brain has been correctly segmented by a group of human experts. This is clearly a very costly and time-consuming project. However, it is necessary for a *scientific* comparison of the performance of various learning algorithms. Simply eyeballing an output segmentation of an algorithm does not give an accurate estimation of how well the algorithm works. The corpus of segmented images can be used both for training and testing various approaches so that they can be compared.

Another problem is the intensity deviations of the MR machines, both across time and across the image itself. Wells [Wells *et al.*, 1994] goes a long way toward fixing the shadowing problem (nonlinear intensity changes) by uses bayesian techniques.

As explained in an earlier section, it is necessary to align all of the brains in order to use location information such as the (x, y, z) coordinates of a point. The registration methods that have been developed so far work well when images taken from the same patient need to be registered. However, using a global transform for registration across patients is a first-order approximation at best. More specific

knowledge about tissues and their properties and locations need to be put into the system before brains from different people can be aligned.

Finally, using model selection along with supervised learning methods is not necessarily the best way to approach the problem. It is possible that clustering methods such as the ones used in [Gerig *et al.*, 1992] will produce more robust results.

Chapter 5

Conclusion

5.1 Related Work

I was 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 which, combined with Kaelbling's confidence bounds [Kaelbling, 1990], generated the Hoeffding Races algorithm. The notion of racing shows up in related forms in multi-armed bandit problems. To give one example, Rivest and 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.

Greiner has independently developed a PALO (Probably Approximately Locally Optimal) algorithm [Greiner and Jurisica, 1992] which also uses Hoeffding's bound in order to decide when one point is better than another. The main differences between our work is that he uses his method for gradient descent (as described at the end of Chapter three), 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-like technique.

Another method based on the racing principle called 'schemata races' is described in [Moore and Lee, 1994]. Here, combinations of attributes are raced to find the most effective combination.

The use of ϵ, δ, and "probably correct" all make use of PAC-like terminology. That is no accident, since the basis for all PAC algorithms is a distribution-free bound on distance from a mean. That bound is normally Chernoff's bound, which is a slightly weaker form of Hoeffding's bound.

5.2 Future Work

- This work was applied to memory based learning algorithms because of their nice computational properties (discussed in Chapter two). Since training takes no time, and Hoeffding Races decreases the testing time, the ratio of brute force to Hoeffding Races represents exactly the savings brought about by racing. The algorithm is easily applicable to any collection of models, including neural networks and decision trees. However, the cost of training all of the models initially may then become the overriding computational factor. An interesting question is whether it is possible to race the training stage, or maybe race partially trained models.

- The only criterion used in this thesis to compare and evaluate a learner has been its accuracy. It is possible for other factors to influence the selection of a learner, but then we can no longer treat it as a learning *box*. We need to look inside each learner and consider issues such as response speed, simplicity, or other a priori biases. These issues can be formulated in a bayesian manner as a priori probabilities and incorporated into a racing scheme by making it harder for a favored learner to lose the race.

- This algorithm should be easy to parallelize. Simply assign one processor per model. If there are less models than processors, then it is probably not even worth racing them. However, there are usually going to be more possible models

than processors, in which case whenever a model gets thrown out of the race, its processor gets a new model.

- 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.

- Haussler's paper on generalizations of the PAC model [Haussler, 1992] shows a way of getting rid of the square root in the expression for ϵ (see Equation 3.2). This is good if there are lots of test points and B is relatively small, but might not be a good idea otherwise. It would be interesting to see tighter bounds so that ϵ can decrease faster with respect to n, the number of test points seen so far.

- Despite the fact that this thesis concentrates on racing learning models, this technique is applicable in many minimization problems which have a discretized space. For example, there has been a successful application of Hoeffding Races to finding the best pinball player among a variety of computerized players. The criterion along which the race takes place is the average score for each player [Johnson, 1993].

- In my opinion, the most exciting (and least likely) offshoot of this research is a search for an 'ideal' collection of models. There is not a single model which is a solution to every problem in the world, but it is possible that a relatively small group of models will cover a large portion of problems. 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 which are orthogonal and span the space. Once we have a basis set of models, then Hoeffding Races 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 cross validation and data mining which I ignored

for the most part. The problem of data mining is that if you look at data long enough, you will begin to see patterns even if it is completely random. It is likely that by throwing too many models at the data, I am actually mining it. If Hoeffding Races 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 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.

5.3 Conclusion

In this thesis, I have given an algorithm for efficiently selecting a classification or regression model. The memory-based learning models that I used are particularly suited to this method because of their computational properties. I have tried to maintain a zero-knowledge approach, and let the various models be judged only according to how well they do, rather than tailor them to a particular domain. I applied these ideas to the problem of segmenting a MR image into gray matter, white matter, lesions, and cerebral fluid. Despite the fact that this is a huge problem, normally relegated to domain-specific or vision-based techniques, memory-based models performed competitively and were made computationally tractable by racing them. I am therefore confident that this technique can be applied to almost any data set with reasonable and fast results.

Bibliography

[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.

[Breiman *et al.*, 1984] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees.* Wadsworth, 1984.

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

[Cover and Hart, 1967] T.M. Cover and P.E. Hart. Nearest Neighbor Pattern Classification. In *IEEE Transactions on Information Theory*, volume IT-13. IEEE, January 1967.

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

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

[Ettinger *et al.*, 1994] G. J. Ettinger, W. E. L. Grimson, and T. Lozano-Pérez. Automatic 3D Image Registration for Medical Change Detection Applications. In *AAAI*

Spring Symposium Series: Applications of Computer Vision in Medical Image Processing, March 1994.

[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.

[Gerig *et al.*, 1992] Guido Gerig, John Martin, Ron Kikinis, Olaf Kubler, Martha Shenton, and Ferenc A. Jolesz. Unsupervised tissue type segmentation of 3D dual-echo MR head data. *Image and Vision Computing*, 10, 1992.

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

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

[Holte, 1993] Robert C. Holte. Very simple classification rules perform well on most commonly-used datasets. *Machine Learning*, 11:63–91, 1993.

[Johnson, 1993] Michael P. Johnson. Reinforcement Learning of a Pinball Controller Using Directed Random Search. Bachelor's Thesis, Massacusetts Institute of Technology, May 1993.

[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.

[Maron, 1994] Oded Maron. 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.

[Moore and Atkeson, 1992] Andrew W. Moore and Chris G. Atkeson. An Investigation of Memory-based Function Approximation for Learning Control. Technical report, Artificial Intelligence Laboratory, M.I.T., July 1992.

[Moore and Lee, 1994] A. W. Moore and Mary 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.

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

[Preparata and Shamos, 1985] Franco P. Preparata and Michael Ian Shamos. *Computational geometry: an introduction.* Springer-Verlag, 1985.

[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] Ronald L. Rivest and Yiqun 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.

[Schaffer, 1993a] Cullen Schaffer. Overfitting avoidance as bias. *Machine Learning*, 10:153–178, 1993.

[Schaffer, 1993b] Cullen Schaffer. Selecting a Classification Method by Croos-Validation. *Machine Learning*, 13:135–143, 1993.

[Wells *et al.*, 1994] W. M. Wells, W. E. L. Grimson, R. Kikinis, and F. A. Jolesz. In-Vivo Intensity Correction and Segmentation of Magnetic Resonance Image Data. In *AAAI Spring Symposium Series: Applications of Computer Vision in Medical Image Processing*, March 1994.

[Wolpert, 1993] David H. Wolpert. On overfitting avoidance as bias. Technical Report No. 92-03-5001, The Santa Fe Institute, 1993.

[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.