An Empirical Investigation of Brute Force to choose Features, Smoothers and Function Approximators

Andrew W. Moore Daniel J. Hill Michael P. Johnson
MIT Artificial Intelligence Laboratory
545 Technology Square
Cambridge, MA 02139

Abstract

The generalization error of a function approximator, feature set or smoother can be estimated directly by the *leave-one-out cross-validation error*. For memory-based methods, this *is* computationally feasible. We describe an initial version of a general memory-based learning system (GMBL): a large collection of learners brought into a widely applicable machine-learning family. We present ongoing investigations into search algorithms which, given a dataset, find the family members and features that generalize best. We also describe GMBL's application to two noisy, difficult problems—predicting car engine emissions from pressure waves, and controlling a robot billiards player with redundant state variables.

1 Introduction

The main engineering benefit of machine learning is its application to autonomous systems in which human decision making is minimized. Function approximation plays a large and successful role in this process. However, many other human decisions are needed even for simple supervised learning and particularly for learning control. These include (i) which function approximator to use, (ii) how to trade smoothness against goodness-of-fit and (iii) which problem features count as important or relevant inputs to the function approximator.

There has recently been much interest in the automation of these decisions. Unfortunately, there is no space here to provide an adequate review of the relevant literature. The main issue concerns how to minimize the expected *test set error*, which is a measure of how accurately the learner will predict new datapoints drawn from the same distribution as the learning dataset. Akaike, in [Akaike, 1974], introduced the *Final Prediction Error* (FPE) measure to estimate this value in linear approximation methods. [Moody, 1992] uses a value called the *Generalized Prediction Error* to extend FPE to non-linear systems. [MacKay, 1992] describes Bayesian methods for predicting the architecture and system parameters that minimize the expected test set error over a dataset.

In contrast, this paper investigates the technique of trying to minimize the test set error by means of leave-one-out cross-validation. We describe an initial version of a general memory-based learning system (GMBL). We present some ongoing investigations into efficient algorithms which, given a dataset, find function approximators and features which generalize best. We then describe the application of GMBL to two real, noisy, difficult problems.

2 Memory-based methods

Here, we merely summarize the essential aspects of memory-based function approximation. A more thorough review may be found in [Moore and Atkeson, 1992]. In memory-based learning, all experiences are explicitly remembered in a large memory of input-output vectors.

$$\{(x_1, y_1), (x_2, y_2) ... (x_n, y_n)\} \tag{1}$$

When a prediction for the output of a novel input x_{query} is required, the memory is searched to obtain experiences with inputs close to x_{query}. These neighbors are used to determine a locally consistent output for the query.

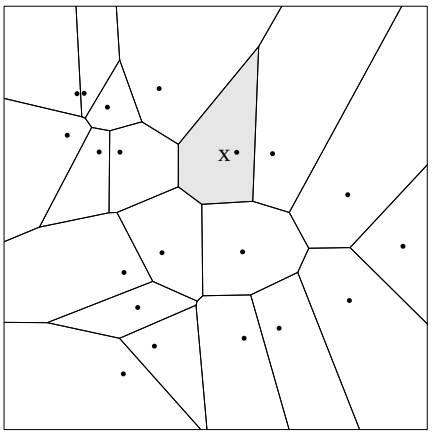


Figure 1: The nearest neighbor regions of a uniformly scaled input space.

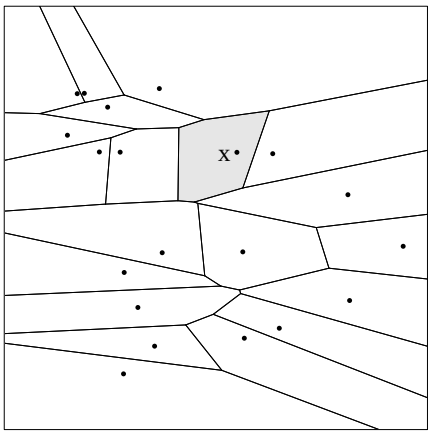


Figure 2: The same dataset with non-uniform scaling.

A simple memory-based method is **nearest neighbor**, which has a large literature. The seminal work [Fix and Hodges, 1951] is included in a recent collection of nearest neighbor papers [Dasarathy, 1991].

Given x_{query}, nearest neighbor searches the memory to obtain an input-output pair (x_i, y_i) which minimizes $\textbf{distance}(x_i, x_{\text{query}})$. The prediction is then y_i. We use the *scaled Euclidian metric:*

$$\textbf{distance}(x, x') = \sqrt{\sum_{k=1}^{D_{\text{in}}} {\sigma_k}^2 \left([x]_k - [x']_k\right)^2} \tag{2}$$

where $[x]_k$ is the kth component of the vector x. $\left\{\sigma_1, \sigma_2, \ldots, \sigma_{D_{\text{in}}}\right\}$ are *scaling parameters*, which greatly affect the prediction. Figure 1 shows the nearest neighbor regions for a set of two-dimensional input points with equal scaling $\sigma_1 = \sigma_2$. Figure 2 shows the considerable effect of non-uniform scaling. $\sigma_1 = 3\sigma_2$. Such scaling can play a crucial role in reducing the prediction error in cases where one input dimension has a greater effect on the output than another.

A common extension of nearest neighbor is to use the mean output value of a small number $k > 1$ of the nearest neighbors. The advantage is protection against noisy data, but a potential disadvantage is a slower learning rate: bias is traded against variance.

Another extension, called **kernel regression**, finds the weighted average of experiences in the memory.

$$y_{\text{predict}} = \frac{\sum w_i y_i}{\sum w_i} \tag{3}$$

where w_i is the weight of the ith experience. This weight depends on its distance to the query point, and is constructed so that the closest points have the greatest effect. A common weighting function is the quadratic hyperbola:

$$w_i = \frac{1}{1 + 20(\textbf{distance}(x_i, x_{\text{query}})/K_{\text{width}})^2} \tag{4}$$

Conventionally, the weighting function is parameterized by a *smoothing kernel width*, K_{width}, which determines the distance, in the Euclidian metric, over which the smoothing function has a significant weight.

Instead of using local averages, **local regression** can be applied. In the simplest case, each time a query arrives, the k nearest neighbors are obtained, where k is a system parameter. A least-squares linear fit is then performed on these neighbors—this is done by direct solution of a $D_{\text{in}} \times D_{\text{in}}$ system of linear equations, where D_{in} is the number of input variables. The resulting linear map is then applied to the query point.

Local weighted regression (LWR) [Cleveland and Delvin, 1988, Grosse, 1989, Atkeson and Reinkensmeyer, 1989] uses the same kind of weighting function as kernel regression. Instead of using ordinary least squares regression, it finds the linear map $y = \mathbf{a}^T x + c$ to minimize the sum of locally weighted squares of residuals:

$$\sum_i {w_i}^2 (y_i - \mathbf{a}^T x_i - c)^2 \tag{5}$$

This minimization can also be solved by means of a $D_{\text{in}} \times D_{\text{in}}$ system of linear equations.

3 Leave-one-out cross validation

Cross-validation is commonly used by statisticians for manually evaluating the application of different function approximators to a particular dataset. However, it can also be used as part of an automatic system to optimize among a large space of possible learning boxes.

In cross-validation, a learner is judged by how well it predicts datapoints which are not in the dataset—simply minimizing the errors for existing datapoints can easily lead to overfitting. The *leave-one-out cross validation* error of the ith point is called e_i^{xve}, and is computed by first removing the ith experience from the memory, then using the remaining data to predict its value, and then observing the discrepancy. Let $y_i^\star$ be the output predicted for input x_i using the memory in which (x_i, y_i) has been removed. The ith leave-one-out cross validation error is $e_i^{\text{xve}} = |y_i^\star - y_i|$. For most kinds of function approximator it is very expensive to perform leave-one-out cross validation because the training stage must be repeated with the new datapoint missing. The essential advantage of memory-based methods for our purpose now comes to light:

> With the memory-based formalism all work takes place at prediction time and there is no training phase. Thus, there is no extra expense in temporarily ignoring a piece of data and pretending we had never seen it.

The value $e_{\text{total}}^{\text{xve}} = \text{RMS}\{e_i^{\text{xve}}\} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} e_i^{\text{xve}2}}$, provides a measure of how well the current representation fits the data. By optimizing this value, we can automatically decide which is the better of a number of alternative learners. For example, consider three LWR learners, each identical except for different K_{width} smoothing parameters $K_{\text{width}} \in \{1/64, 1/4, 1\}$. These three learners applied to the dataset in Figure 3 exemplify a commonly observed trade-off. A overfits, C oversmooths and B, which trades off these extremes, is the option preferred by cross-validation.

We are interested in seeing how hard we can push cross validation to choose a good learner from a combinatorial space of a wide variety of memory-based learners. To do that, we will first describe the space of learners and then describe some optimization methods to search over the space.

4 Generalized Memory-Based Learning

General memory-based learning consists of a wide family of memory-based learners. All family members can be described by a simple syntax called a `gmbl-spec`. An example is shown in Figure 4.

The `gtype` of a `gmbl-spec` denotes the type of memory-based learner, details about its local weighting function and the number of nearby neighbors which are used. The encoding is shown in Table 1. The `gfeatures`

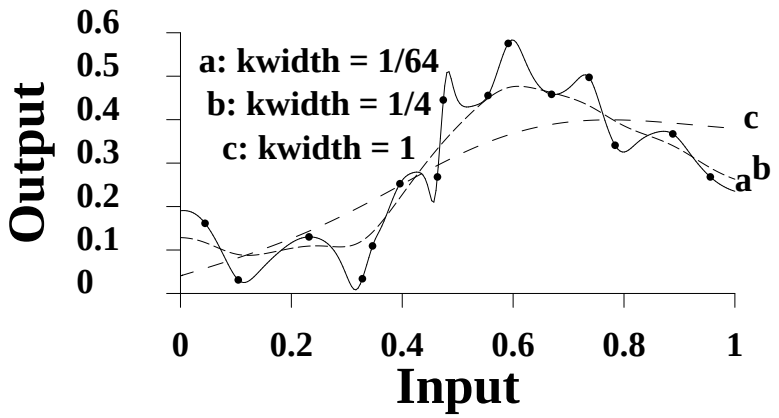


Figure 3: LWR and three alternative K_{width} values.

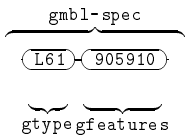


Figure 4: A `gmbl-spec`.

First field: **Degree**	A	Denotes learners which use locally constant or locally averaging predictions, such as k-nearest neighbor or kernel regression
	L	Denotes learners which use locally linear (possibly weighted) regression
Second field: **Kernel width**	0	Denotes learners which use no local weighting
	$1 \leq x \leq 8$	Denotes learners which use the weighting function of Equation 4 with $K_{\text{width}} = 2^{x-7}$ relative to scaled input metric
	9	Denotes learners which use full weighting. For the C degree, this reduces to computing the global average and for the L degree it reduces to global linear regression
Third field: **Number neighbors**	0	Means we rely entirely on local weighting
	$1 \leq k \leq 9$	For the C-degree learner this denotes k-nearest neighbors, possibly in addition to any local weighting. For the L-degree learner this denotes local regression using the $N_{\text{features}} + k$ nearest neighbors, where N_{features} is the number of inputs which have non-zero scaling.

Table 1: Details of the `gtype`.

contains details of how the input variables are scaled. Before proceeding we must deal with the question of differing spreads of values in different features of the dataset. Compensation for this is by means of an initial basic scaling of the different input variables before the `gfeatures` are applied. A robust measure of the underlying spread of each input variable is taken. For the ith feature, this is the distance between the 10th percentile and 90th percentile of all values of the ith input variable in the dataset. Call the spread estimate for the ith feature z_i.

The `gfeatures` provokes additional scaling. There is one field, f_i, for each input variable, which takes a value $0 \leq f_i \leq 9$. 9 denotes the strongest scaling, 1 denotes very weak scaling, with intermediate scales distributed geometrically. 0 means that the input variable is ignored entirely.

$$s_i = \begin{cases} 0 & \text{if } f_i = 0 \\ 2^{f_i} & \text{if } 0 < f_i \leq 9 \end{cases} \tag{6}$$

The combined, scaled, Euclidian metric has

$$\sigma_i = \frac{s_i}{z_i \sqrt{s_1^2 + s_2^2 + \cdots + s_{D_{\text{in}}}^2}} \tag{7}$$

A01 990590	Simple 1-nearest neighbor, with the 1st, 2nd and 5th spread-normalized inputs equally scaled, the 4th input scaled at $\frac{1}{16}$th the strength and inputs 3 and 6 ignored.
L60 999999	Local weighted regression using a K_{width} smoothing parameter of $\frac{1}{4}$. All inputs equal.
L34 019090	Local weighted regression using a K_{width} value of $\frac{1}{32}$, but giving full weighting to the N_{features} + `k-near-spec` $= 3 + 4 = 7$ nearest neighbors, whatever their distance.

Table 2: Three `gmbl-spec`s.

Table 2 shows three examples from our space of memory-based learners.

5 Searching for the best `gmbl-spec`

Although the space of learners is discrete and finite, it is generally too large for exhaustive search for the `gmbl-spec` that minimizes $e^{\text{xve}}_{\text{total}}$. This problem is partially addressed by only considering a small subset S_{restrict} of the space of possible `gtype`s. The members of S_{restrict} are listed in Figure 3. The search for the best complete `gmbl-spec` proceeds by performing the top-level search over the space of `gfeature`s, and for each `gfeature`, exhaustively evaluating each `gtype` in S_{restrict}. The final question concerns the search among feature-strings.

Hill-climbing
This simple algorithm is used:

> Given the current best feature string, try all one-feature-change adjustments to it until no improvement is obtained.

All possible alterations are tried in turn, and the current best feature string is updated immediately, whenever an improvement occurs. The optimization terminates when $9D_{\text{in}}$ alterations (the number of possible one-feature changes) in succession make no improvement.

Genetic Optimization
The basic genetic optimization algorithm described in [Goldberg, 1989] is used. A population of 50–200 feature strings is maintained along with their

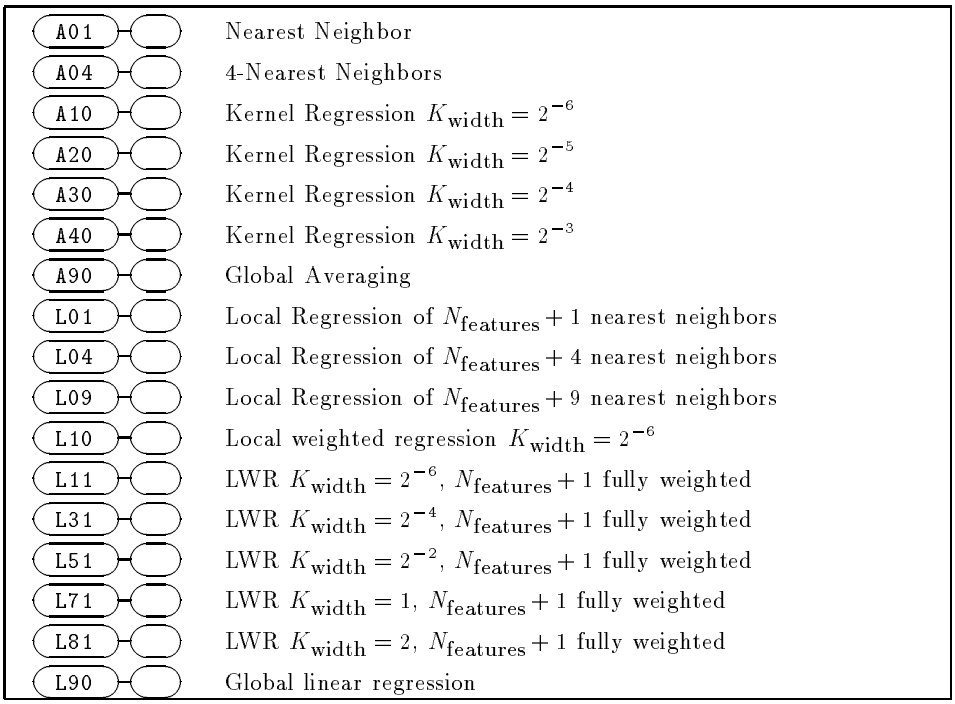


Table 3: The restricted types

cross validation errors. Pairs of substrings are selected as parents for the next generation according to a biased random procedure which favors strings with a relatively low $e^{\text{xve}}_{\text{total}}$. Two parents produce two children by randomly swapping subcomponents.

Feature Success Statistics
A `gfeatures` is *binary* if it contains only zeros and nines—every input is either fully scaled or completely ignored. This method uses a statistical heuristic to find a good binary feature-string with relatively few $e^{\text{xve}}_{\text{total}}$ computations. A sample of fifty binary strings is generated at random and then evaluated. Each individual feature component f_i is then judged by how frequently it takes part in a relatively good `gfeatures`. Feature i is scored by

$$\frac{\text{No. of strings in which } f_i = 9 \textbf{ and } e^{\text{xve}}_{\text{total}} \text{ was below the sample median}}{\text{No. of strings in which } f_i = 9} \tag{8}$$

The features are ranked according to this score, and then the following sequence of `gfeatures` are generated:

$$\begin{aligned} \text{fs}_1 &= \text{Feature ranked 1st is set to 9, all others to 0} \\ \text{fs}_2 &= \text{Features ranked 1st and 2nd are set to 9, all others to 0} \\ \text{fs}_3 &= \text{Features ranked 1st, 2nd and 3rd are set to 9, all others to 0} \\ &\vdots \end{aligned} \tag{9}$$

This sequence is used to seed the previous techniques. Hill-climbing takes the best fs_i, and the genetic algorithm includes the complete set in its initial population.

Exhaustive search over k-feature binary features
An initial seed is obtained by exhaustive search over all binary strings which contain k or less '9's.

6 Empirical results

6.1 Simulated datasets

The first test was performed using the feature set in Figure 5. 100 datapoints were generated with random $\theta_1, \theta_2, \theta_3, \theta_4$ ($-\pi/2 < \theta_i < \pi/2$), and

Experiment	gmbl-spec	$ne^{\text{xve}}_{\text{total}}$	Time	nt_{total}
All feature scales equal	L10 999999999999	0.325	2 secs	0.348
Best from hill climbing	L11 000000800090	0.054	21 mins	0.063
Best from all $k = 3$ binaries then hill	L11 000000800090	0.054	24 mins	0.063
Best from GA	L31 009000900990	0.102	2 hours	0.129
Best from GA with feature-stats	L31 009000900990	0.102	1 hour	0.129

Table 4. Redundant Kinematics experiment.

the x_i's and y_i's were computed according to the problem kinematics. The output variable, to be learned, was the distance between the origin and the third joint (x_3, y_3). Uniform random noise of maximum magnitude $\frac{1}{10}$ link-length was added. The problem is interesting because the dataset is highly redundant. Table 4 describes how the search methods fared. The $e^{\text{xve}}_{\text{total}}$ error is *normalized* in this, and subsequent tables—divided by the standard deviation of the output variable which is being predicted. For example, the simple learner which always predicted the global mean of the outputs would achieve a $ne^{\text{xve}}_{\text{total}}$ score of approximately 1. Also shown is nt_{total}, the normalized RMS error of an independent test set of 100 further points.

It is clearly beneficial to not use all features. Local regression with a small kernel was preferred in all cases—this is consistent with the relatively small amount of noise in the dataset. The hill climbers chose what appear to be particularly suitable features, x_3 and y_3, which are indeed sufficient to determine the output.

The next experiment used the opposite kind of problem—one in which all features play an equally significant part. Each dataset entry had twenty independent random input variables $x_1, x_2 \cdots x_{20}$ (each x_i chosen uniformly in the range $0 < x_i < 1$), with the output defined as $\sum_{i=1}^{20} x_i^2$. There were 100 datapoints, and the results are in Table 5.

All searches converged to equal weighting. Global linear regression was universally preferred—a reflection of the relatively small dataset size in comparison with the number of dimensions.

6.2 Predicting Engine Emissions from Pressure Waves

Figure 6 shows 100 successive pressure waves from one cylinder of a car engine. We have thirty pressure-wave series, each obtained with the en-

Experiment	gmbl-spec	$ne^{\text{xve}}_{\text{total}}$	Time	nt_{total}
All feature scales equal	L90 9999999999999999999	0.250	30 secs	0.250
Best from hill climbing	L90 9999999999999999999	0.250	10 hours	0.250
Best from all $k = 3$ binaries then hill	L90 9999999999999999999	0.250	10 hours	0.250
Best from GA	L90 9999999999999999999	0.250	3 hours	0.250

Table 5. Experiment with equally important features.

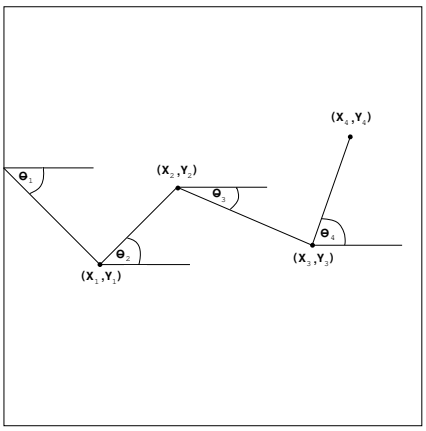


Figure 5: Twelve features of a four-link kinematics problem. The goal is to learn the value $\sqrt{x_3^2 + y_3^2}$.

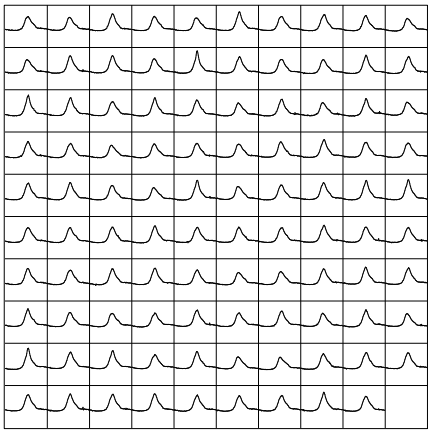


Figure 6: 99 Pressure waves.

gine running under various *loads* and *air-fuel ratios*. Along with each series there is a reading from an oxygen sensor at the exhaust. Can the oxygen be predicted by the pressure waves?. If so, what are good pressure-wave features?

Programs were written to extract a large number of simple features from each pressure wave (listed in Table 6), No good univariate fit is available (the best, over all `gtypes` has $ne^{\text{xve}}_{\text{total}} = 0.52$). The GMBL cross-validation algorithms were each run on the dataset, with the results shown in Table 7. Due to the shortage of data, no independent test set validation was performed.

This search space was peppered with local minima, hence the wide range of differing solutions. It is interesting that nearest neighbor was universally preferred over regression for this problem.

mean	sdev-of-mean	sdev	sdev-of-sdev
max-value	sdev-of-max-value	min-value	sdev-of-min-value
max-index	sdev-of-max-index	imep	sdev-of-imep
burn-start	sdev-of-burn-start	burn-rate	sdev-of-burn-rate
moment-1	sdev-of-moment-1	moment-2	sdev-of-moment-2
moment-3	sdev-of-moment-3		

Table 6: Pressure wave feature names.

Experiment	gmbl-spec	$ne^{\text{xve}}_{\text{total}}$	Time
All feature scales equal	A04 99999999999999999999	0.3455	1 sec
Best from hill climbing	A04 0090000009088000000007	0.2244	40 mins
Best from all $k = 3$ binaries then hill	A01 0500008090000000090089	0.0949	4 hours
Best from GA	A04 9090900009090999909990	0.2218	5 hours
Best from GA with feature-stats	A04 8095529502098882095055	0.1999	6 hours

Table 7. Pressure waves experiment.

7 Learning to play Billiards with Redundant Features

Some experiments were performed with the real billiards robot shown in Figure 7. All sensing is visual: one camera looks along the cue stick and the other looks down at the table. The cue stick swivels around the cue ball which, in these experiments, has to start each shot at the same position. A shot proceeds as follows:

1. The object ball (the ball which we will try to sink in a pocket) is placed at a random position in the half of the table opposite the cue stick. The position is selected by the computer in order to avoid human bias.

2. The camera above the table looks at the object ball. Simple image processing extracts eight visual features of the board position, shown in Figure 10. All features are in image coordinates: no attempt is made to map them on to real world coordinates, or to compensate for camera non-linearities.

3. The controller then selects a *cue-action*, specified by what we wish the view from the cue-camera to be just prior to shooting. Figure 8 shows a view from the cue camera during this process. The cue swivels until the centroid of the object ball's image (shown by the vertical line) coincides with the chosen cue-action $x^{\text{cue}}_{\text{object}}$ (shown by the cross). The cue-action is chosen by prediction: the function

$$\underbrace{\text{Table features}}_{\text{Eight values}} \times \underbrace{\text{Cue-action}}_{\text{One value}} \rightarrow \underbrace{\text{Outcome}}_{\text{One value}} \tag{10}$$

is approximated from all previous shots. A search is performed for a cue-action that has an outcome predicted as sinking the ball. If the search fails, a random action is chosen.

4. The shot is then performed and observed by the overhead camera. The image after a shot, overlayed with the tracking of both balls, is shown in Figure 9. The *outcome* is defined as the cushion and position on the cushion with which the object ball first collides. In Figure 9 it is the point b.

Figure 7: The billiards robot. In the foreground is the cue stick which attempts to sink balls in the far pockets.

As time progresses, the database of experiences increases, hopefully converging to expertise in the manifold of state-space corresponding to sinking balls placed in arbitrary positions. Several experiments were performed, each starting with an empty database and consisting of 120 shots. Results

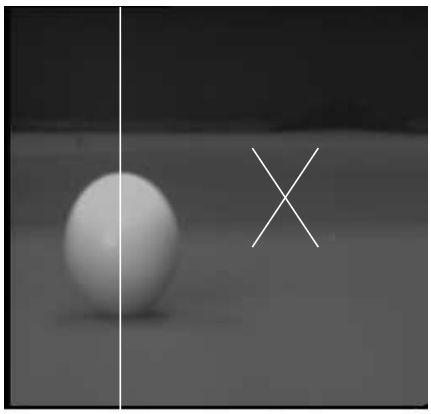


Figure 8: The view from the cue camera during aiming

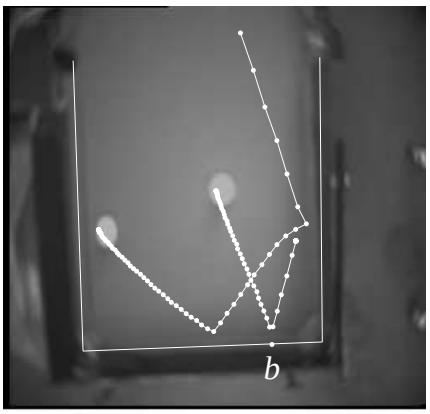


Figure 9: Tracking the shot with the overhead camera

Experiment	gmbl-spec	$ne^{\text{xve}}_{\text{total}}$	Time
Best from all $k = 3$ binaries then hill	L71 — 000300339	0.141	20 mins

Table 8. Pool robot: the best gmbl-spec.

are in Table 9. The first used all the features, and both generalization and performance suffered. The second run used the basic features—the X and Y coordinates of the image of the ball. This restriction of features to the bare essentials improved performance. After 100 shots a success rate above 70% was achieved. The third experiment interleaved cross-validation with learning control. After every five shots, the robot was paused, and the search for the best gmbl-spec was undertaken. Again, the results were better than using all features, though on several occasions poor gmbl-specs were chosen, causing strings of misses.

The final best gmbl-spec after this series of shots was L71 — 000300339, as shown in Table 8.

This was used for the fourth experiment, which produced (in, admittedly, a statistical sample of one) our best learning curve to date. The robot became adept very early and averaged approximately 80% success later on in life.

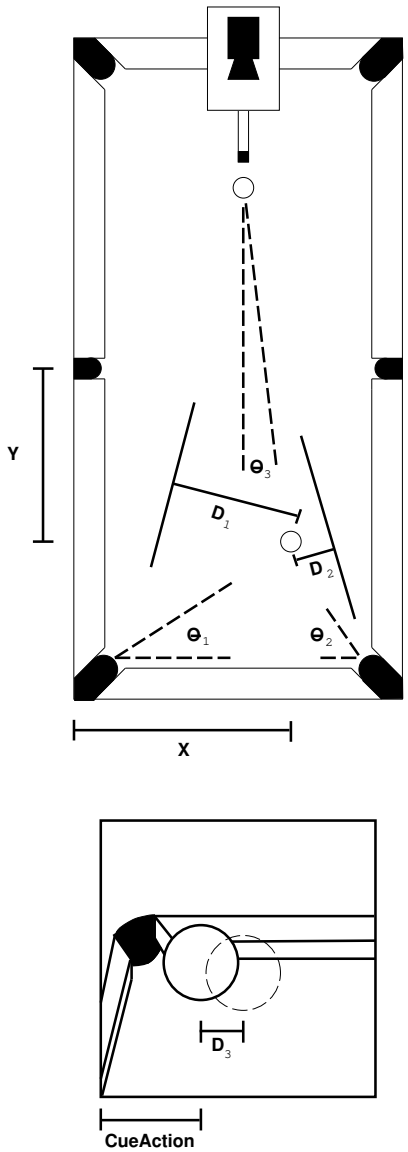
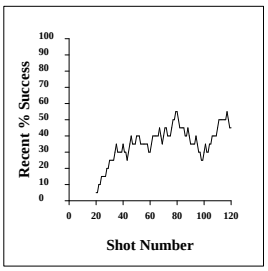


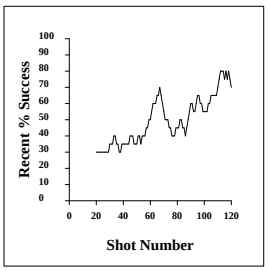
Figure 10: The pool robot and its features.

X	X-coord of ball centroid in pixels.
Y	Y-coord of ball centroid in pixels.
θ_1	Angle between image of back cushion and ball at left pocket.
θ_2	Angle between image of back cushion and ball at right pocket.
θ_3	Angle between the camera vertical and the line from cue ball to object ball.
D_1	Projection, on the perpendicular to the cue $\leftrightarrow$ left pocket line, of the distance between object and the pocket. (This is an approximation of a feature we conjecture that humans use).
D_2	The same, for the right pocket.
D_3	The "naive physics" estimate of the X-coord of the cue contact position which would pocket the ball.

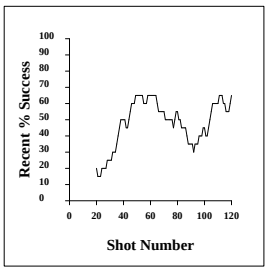
Using all features



Using X Y and cue-action only



XVE every five shots to choose `gmbl-spec`



Using final best `gmbl-spec` from previous test

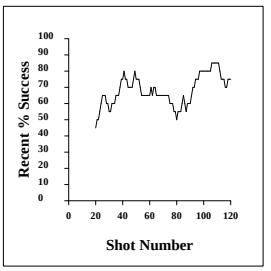


Table 9: Recent percent of balls sunk against time. The graph shows what percentage of the twenty most recent shots were successful.

8 Discussion

Even though cross-validation estimates the generalization error, rather than goodness of fit, the danger of overfitting a dataset remains. The large space of potential learning boxes raises the possibility that for a particular dataset the search will stumble upon a learning box with a deceptively low cross-validation error, which does not reflect the expected error on future datapoints. Keeping aside a further test set of datapoints for a higher level of cross validation can be used to detect such occurrences.

Finding good features has always been considered a desirable but difficult aspect of machine learning. We have investigated the brutal method of inventing a large set of redundant features and then searching for the function approximator and feature-set which have the best predictive accuracy.

Killing irrelevant input features has received attention in several fields. [Aha, 1990], in particular, describes an incremental approach for some memory based learners, as well as surveying some earlier approaches to the problem. A measure of which attributes have the most information is also estimated for decision tree learners [Breiman *et al.*, 1984]. Projection pursuit learning [Zhao, 1992] searches the space of directions in input space to find "interesting" projections.

A general architecture of a rich class of inductive concept learners is described in [Rendell *et al.*, 1987], which uses meta-level classification on

dataset features to try to predict, for a new dataset, which member of the class will perform best. Real-time until convergence is used as the performance criterion. Another possibility is that even better accuracy could be obtained by partitioning input space into different zones, in which different `gmbl-spec`s are allowed to operate. For example, the architecture of [Jordan and Jacobs, 1992] uses gradient descent to recursively partition a family of different neural nets.

Memory-based learning is an ideal kind of approximator with which to perform leave-one-out cross validation because it is no more expensive to predict a value with one datapoint removed than with it included.

Further investigations will evaluate the extent to which minimizing the test set error coincides with optimizing the desired *ultimate* performance-measure of a learning task. For example, the billiards robot uses learning primarily in order to sink balls, and we have assumed that this coincides well with good test set generalization. It would perhaps be desirable, though computationally difficult, to use Bayesian methods to choose, from the class of learners, that member which is best suited to the ultimate task at hand.

The brute force approach is computationally expensive—some of the searches described in this paper took several hours to perform on a Sun-4. We are developing a new technique called *Hoeffding Racing* [Maron and Moore, 1993] which cuts down computation time considerably by detecting early on in a cross-validation computation that there is little probability that the final mean error will be a low as that of the best `gmbl-spec` to date, and so can cut off that computation early. There will be many other opportunties, both in hardware and software, to greatly increase the search rate.

Acknowledgements

Some of the work discussed in this paper is being performed in collaboration with Chris Atkeson. The robot cue stick was designed and built by Wes Huang with help from Gerrit van Zyl. Andrew Moore is supported by a Postdoctoral Fellowship from SERC/NATO. Support was provided under Air Force Office of Scientific Research grant AFOSR-89-0500 and a National Science Foundation Presidential Young Investigator Award to Christopher G. Atkeson.

References

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

[Akaike, 1974] H. Akaike. A new look at Statistical Model Identification. *IEEE Transactions on Automatic Control*, 19, 1974.

[Atkeson and Reinkensmeyer, 1989] C. G. Atkeson and D. J. Reinkensmeyer. Using Associative Content-Addressable Memories to Control Robots. In W. T. Miller, R. S. Sutton, and P. J. Werbos, editors, *Neural Networks for Control*. MIT Press, 1989.

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

[Cleveland and Delvin, 1988] W. S. Cleveland and S. J. Delvin. Locally Weighted Regression: An Approach to Regression Analysis by Local Fitting. *Journal of the American Statistical Association*, 83(403):596–610, September 1988.

[Dasarathy, 1991] B. V. Dasarathy. *Nearest Neighbor Pattern Classification Techniques*. IEEE Computer Society Press, 1991.

[Fix and Hodges, 1951] E. Fix and J. L. Hodges. Discriminatory analysis, Nonparametric regression: consistency properties. Technical report, USAF School of Aviation Medicine, 1951.

[Goldberg, 1989] D. E. Goldberg. *Genetic Algorithms*. Addison Wesley Co., 1989.

[Grosse, 1989] E. Grosse. LOESS: Multivariate Smoothing by Moving Least Squares. In L. L. Schumaker C. K. Chul and J. D. Ward, editors, *Approximation Theory VI*. Academic Press, 1989.

[Jordan and Jacobs, 1992] M. I. Jordan and R. A. Jacobs. Hierarchies of Adaptive Experts. In J. E. Moody, S. J. Hanson, and R. P. Lippman, editors, *Advances in Neural Information Processing Systems 4*. Morgan Kaufmann, April 1992.

[MacKay, 1992] D. J. C. MacKay. Bayesian Model Comparison and Backprop Nets. In J. E. Moody, S. J. Hanson, and R. P. Lippman, editors, *Advances in Neural Information Processing Systems 4*. Morgan Kaufmann, April 1992.

[Maron and Moore, 1993] O. Maron and A. Moore. Hoeffding Races: Accelerating Model Selection Search for Classification and Function Approximation. In *Advances in Neural Information Processing Systems 6*. Morgan Kaufmann, December 1993.

[Moody, 1992] J. E. Moody. The Effective Number of Parameters: An Analysis of Generalization and Regularization in Nonlinear Learning Systems. In J. E. Moody, S. J. Hanson, and R. P. Lippman, editors, *Advances in Neural Information Processing Systems 4*. Morgan Kaufmann, April 1992.

[Moore and Atkeson, 1992] A. W. Moore and C. G. Atkeson. Memory-based Function Approximators for Learning Control. In preparation, 1992.

[Rendell *et al.*, 1987] L. Rendell, R. Seshu, and D. Tcheng. Layered Concept-learning and Dynamically-variable bias Management. In *Proceedings of the ninth IJCAI, Milan*. Morgan Kaufmann, 1987.

[Zhao, 1992] Y. Zhao. On Projection Pursuit Learning. PhD. Thesis, MIT Artificial Intelligence Laboratory, 1992.