

Local Learning Based On Recursive Covering

Jerome H. Friedman*
Department of Statistics
Stanford Linear Accelerator Center
Stanford University
jhf@playfair.stanford.edu

August 1, 1996

Abstract

Local learning methods approximate a global relationship between an output (response) variable and a set of input (predictor) variables by establishing a set of “local” regions that collectively cover the input space, and modeling a different (usually simple) input-output relationship in each one. Predictions are made by using the model associated with the particular region in which the prediction point is most centered. Two widely applied local learning procedures are K - nearest neighbor methods, and decision tree induction algorithms (CART, C4.5). The former induce a large number of highly overlapping regions based only on the distribution of training input values. By contrast, the latter partition the input space into a (relatively) small number of highly customized (disjoint) regions using the training output values as well. Recursive covering unifies these two approaches in an attempt to combine the strengths of both. A large number of highly customized overlapping regions are produced based on both the training input and output values. Moreover, the data structure representing this cover permits rapid search for the prediction region given a set of (future) input values.

1 Learning

In the learning problem one has a system that produces data in the form of simultaneous measurements on a set of quantities call variables. Some of these are regarded as “input” or “predictor” variables $\mathbf{x} = (x_1, \dots, x_n)$ and the others as “output” or “response” variables. Here we consider the case of only one output or response y . Assuming the statistical model

$$y = f(\mathbf{x}) + \varepsilon \quad (1)$$

where $f(\mathbf{x})$ is a single valued deterministic function of n arguments and ε is a random component, the goal is to approximate (estimate) $f(\mathbf{x})$ by another function $\hat{f}(\mathbf{x})$ over some range of $\mathbf{x} \in R^n$. The random component ε in (1) reflects the fact that knowing simultaneous values of all of the input variables \mathbf{x} (usually) does not specify a unique value of y , but rather a distribution of y -values characterized by the (usually unknown) distribution of ε , $\varepsilon \sim L(\varepsilon|\mathbf{x})$. To make $f(\mathbf{x})$ (1) identifiable one usually specifies

$$E(\varepsilon|\mathbf{x}) = 0 \quad (2)$$

for all values of $\mathbf{x} \in R^n$ so that the “target” function $f(\mathbf{x})$ is (uniquely) defined as

$$f(\mathbf{x}) = E(y|\mathbf{x}). \quad (3)$$

*Work supported in part by the Department of Energy under contract number DE-AC03-76SF00515 and by the National Science Foundation under grant number DMS-9403804

Learning procedures use a “training” sample

$$T = \{y_i, \mathbf{x}_i\}_1^N \quad (4)$$

of N previously solved cases for which both all of the inputs and output values have been obtained to attempt to “learn” the target function (3). That knowledge is incorporated in the approximating function $\hat{f}(\mathbf{x})$.

Learning has two purposes: prediction and interpretation. In prediction, it is presumed that (future) data will be available for which the input variables \mathbf{x} have been measured, but the value of the output y is unknown. One uses the approximator $\hat{f}(\mathbf{x})$ evaluated at \mathbf{x} as a prediction (estimate) \hat{y} for the unknown y value

$$\hat{y} = \hat{f}(\mathbf{x}). \quad (5)$$

In this application the goal is accuracy as defined by small values of

$$E_{\varepsilon, \mathbf{x}} L(y, \hat{f}(\mathbf{x})) \quad (6)$$

where $L(y, \hat{y})$ is some (user defined) loss function for predicting \hat{y} when the truth is y . When the output takes on (orderable) real values $y \in R^1$ (“regression”) commonly used loss functions are

$$L(y, \hat{y}) = |y - \hat{y}|^q \quad (7)$$

with $q = 2$ being the most popular because it leads to simpler minimization procedures. When y takes a discrete set of J (unordered) values (“classification”) the elements of the loss matrix $L(y, \hat{y}) \in R^{J \times J}$ are explicitly specified.

For interpretation the goal is to use the structural form of $\hat{f}(\mathbf{x})$ as a descriptive statistic for attempting to understand the predictive relationship between the inputs and output. This is largely a human engineering problem usually at tension with the prediction goal. Placing the additional constraint of “interpretability” on the approximator $\hat{f}(\mathbf{x})$ usually degrades its predictive ability.

2 Local learning

There is a large literature devoted to the learning problem describing many different types of procedures. One such class of learning methods is based on the concept of “local” learning. With this approach the space $\mathbf{x} \in R^n$ of input values is covered by a set of “local” regions $\{R_m\}_1^M$ where

$$R_m \subset R^n \text{ and } \cup_1^M R_m = R^n. \quad (8)$$

One then learns a separate (“simple”) approximator $\hat{f}_m(\mathbf{x})$ individually in each such region. These are usually taken to be q th order polynomials with $q = 0, 1, 2$ being the most popular choices. The notion of “local” is characterized by the “size” of the regions

$$size(R_m) = ave_{\mathbf{x}, \mathbf{x}' \in R_m} \|\mathbf{x} - \mathbf{x}'\| \quad (9)$$

where $\|\mathbf{x} - \mathbf{x}'\|$ is some measure of distance between the points \mathbf{x} and \mathbf{x}' , and the average in (9) is over all $\mathbf{x}, \mathbf{x}' \in R_m$.

With local learning, predictions (at \mathbf{x}) are made by first choosing one of the local regions $R_{m^*(\mathbf{x})}$ and using its corresponding approximator

$$\hat{y} = \hat{f}_{m^*(\mathbf{x})}(\mathbf{x}). \quad (10)$$

The region chosen is the one in which the prediction point is most “centered”

$$m^*(\mathbf{x}) = \arg \min_{1 \leq m \leq M} ave_{\mathbf{x}' \in R_m} \|\mathbf{x} - \mathbf{x}'\|. \quad (11)$$

Local learning is motivated by Taylor’s theorem which states that if a region is local enough any continuous function $f(\mathbf{x})$ can be well approximated by a low order polynomial within it.

3 Bias-variance trade-off

If one adopts a squared-error loss function [$q = 2$ in (7)] then its expected value (at \mathbf{x}) can be decomposed as

$$E[y - \hat{f}(\mathbf{x})]^2 = E[y - f(\mathbf{x})]^2 + [f(\mathbf{x}) - Ef(\mathbf{x})]^2 + E[\hat{f}(\mathbf{x}) - Ef(\mathbf{x})]^2. \quad (12)$$

The three terms on the right hand side of (12) are well known quantities. The first is the variance of the random component ε (1) and represents the irreducible prediction error owing to the random nature of the output y . The second term (“bias-squared”) reflects the ability of the approximation $\hat{f}(\mathbf{x})$ to represent the target $f(\mathbf{x})$. The third term (“variance”) measures the stability of the derived approximation to changes in the training data (4). Since the training data is usually a random sample, the variance measures the squared-error in $\hat{f}(\mathbf{x})$ induced by sampling fluctuations.

To reduce mean-squared error it is desirable to have small bias-squared and variance. However there is usually a tension between these goals. In the case of local learning the variance is largely determined by the number of training observations in the prediction region. More data in the region produces less variance in the estimates of the parameters of the low order polynomial and thereby less variance in the function estimate $\hat{f}_m(\mathbf{x})$. The bias-squared is mostly governed by the size (9) of the region. The smaller the region the more accurately a low order polynomial is likely to approximate the target $f(\mathbf{x})$ within it. Since there is a direct relation between the size of a region and the number of training points within it, there is a trade-off between bias-squared and variance leading to an optimal number of observations within any given region.

If there is only one input variable ($\mathbf{x} \in R^1$) then the regions of the input space are characterized by their (individual) locations and sizes. In the case of more than one input $\mathbf{x} \in R^{\geq 2}$ regions not only have location and size, they have “shape” as well. The shape of a region can be characterized by its size along all directions in the input space; that is, by the function

$$s_m(\boldsymbol{\alpha}) = \text{ave}_{\mathbf{x} \in R_m} |\boldsymbol{\alpha}^t (\mathbf{x} - \mathbf{u}_m)| \quad (13)$$

where $\boldsymbol{\alpha}$ is a unit vector representing a direction in R^n and \mathbf{u}_m is the location (center) of the (m th) region. For a given number of training points within a region (variance) the shape function (13) can be chosen to minimize bias-squared.

4 K - nearest neighbors

With K - nearest neighbor (“ K -NN”) local learning each region R_m is constructed by defining its location (center) \mathbf{u}_m and taking the region to be that part of the space occupied by the K - closest training points to \mathbf{u}_m . In terms of the points that lie within it the region is defined by

$$R_m(\mathbf{u}_m) = \{\mathbf{x}_i \mid \|\mathbf{x}_i - \mathbf{u}_m\| \leq d_m^{(K)}\} \quad (14)$$

where $d_m^{(K)}$ is the K th order statistic of $\{\|\mathbf{x}_i - \mathbf{u}_m\|\}_{i=1}^N$. As the center (location) \mathbf{u}_m ranges over all of R^n a finite number of distinct regions (14) are produced that cover the input space. For $K = 1$ this produces the Voronoy tessellation of the input space, making a partition of N disjoint regions. For $K > 1$ (and $n > 1$) a cover consisting of a (much) larger number of regions is produced.

In order to apply K -NN one must specify a value for K and define the distance $\|\mathbf{x} - \mathbf{u}\|$ used in (14). K is usually taken to be a meta-parameter of the procedure whose value is determined through model selection (i.e. cross-validation). The distance measure is a quadratic form in the input variables

$$\|\mathbf{x} - \mathbf{u}\|^2 = (\mathbf{x} - \mathbf{u})^t \mathbf{M}^{-1} (\mathbf{x} - \mathbf{u}) \quad (15)$$

characterized by a positive definite symmetric matrix $\mathbf{M} \in R^{n \times n}$. The matrix \mathbf{M} controls the shape (13) of the regions (under the constraint of elliptical symmetry). Common choices for \mathbf{M}

include \mathbf{I}_n the identity matrix producing spherically symmetric regions, or a diagonal matrix

$$\mathbf{M} = \text{diag}\{s_j^2\}_1^n \quad (16)$$

whose elements s_j^2 are squared scale estimates of each input x_j over the training data. This (16) gives equal influence to each input variable in defining the regions, but not to all directions in the input space. To achieve this \mathbf{M} can be set to the covariance matrix of the sample inputs.

K -NN local learning has been studied in the fields of statistics [Stone (1977) and Cleveland and Devlin (1988)], neural networks [Bottou and Vapnik (1992)], and machine learning [Atkeson, Moore, and Schaal(1996)]. It has been shown to be quite effective in low dimensional settings (few independent inputs). Unfortunately, as the input dimension grows the approach tends to rapidly loose power for regression ($y \in R^1$). [It often remains powerful for classification however. See Friedman (1996).]

The reason that K -NN procedures tend to loose effectiveness is that as the dimension of the input space increases the shape of the regions becomes more important. This is because in high dimensional spaces it is impossible to construct regions that have small size simultaneously in all directions and contain sufficient training data. This is a consequence of the so called ‘‘curse-of-dimensionality’’ [Bellman (1961)]. Suppose one chooses spherically symmetric regions [$s_m(\boldsymbol{\alpha})$ (13) independent of $\boldsymbol{\alpha}$], and let R_0 be a region containing all of the training data. Then one has (approximately) for any region R_m

$$\frac{\text{size}(R_m)}{\text{size}(R_0)} = \left(\frac{K}{N}\right)^{1/n} \quad (17)$$

where K is the number of training points in each region R_m and N is the total training sample size. Thus in high dimensions (large n) the size of any region is close to R_0 even for $K = 1$ (maximum variance). This results in high bias even for large variance and therefore large mean-squared error. In order to have a chance of overcoming the curse-of-dimensionality region shape (13) must be chosen judiciously in an attempt to minimize bias.

The optimal shape for any region R_m is governed by the properties of the target function $f(\mathbf{x})$ within it. Specifically, its size in any direction $\boldsymbol{\alpha}$ (13) should be inversely proportional to the rate of change of the (absolute) bias in that direction

$$s_m(\boldsymbol{\alpha}) \sim 1/\text{var}_{\mathbf{x} \in R_m}(\boldsymbol{\alpha}^t \frac{\partial}{\partial \mathbf{x}} |f(\mathbf{x}) - \hat{f}_m(\mathbf{x})|). \quad (18)$$

This clearly depends on the target function $f(\mathbf{x})$, the approximator $\hat{f}_m(\mathbf{x})$ used, and the location \mathbf{x} of the region in the input space. In most implementations of K -NN (14) (15) the shape of the regions is taken to be independent of all three. In some the matrix \mathbf{M} (15) is made a (meta) parameter of the procedure and (numerically) optimized to minimize lack-of-fit to the training data. A special case of this approach is variable subset selection in which \mathbf{M}^{-1} is restricted to be diagonal, with a subset of its elements set to zero. This generally results in improved performance since region shape is (globally) adjusted (under the constraint of elliptical symmetry) to account for the target function and the local approximator used. However, it still suffers from the limitation of using the same shape for all regions independently of their location in the input space.

Another limitation of K -NN is computational. The number of covering regions is usually quite large and in high dimensional spaces there exists no convenient data structure to delineate them or rapidly search for the one in which a prediction point \mathbf{x} is most centered. As a consequence one is generally forced to a ‘‘lazy’’ implementation. The prediction region $R_{m^*(\mathbf{x})}$ (11) is separately constructed for each prediction point \mathbf{x} by defining it as the center and searching for the K closest training points to it. This approach eliminates the need for preliminary training by delaying it until a prediction is needed, and then only doing enough computation to predict the given point. Lazy implementations are efficient if only a few predictions are needed, but are wasteful if there are many predictions or if they are time critical.

5 Recursive partitioning

Recursive partitioning “RP” [CART, Breiman et. al. (1984), C4.5, Quinlan (1993)] is a local learning paradigm intended to overcome the limitations of K -NN. As with any local learning method the input space is covered by a set of “local” subregions and a separate simple approximator (low order polynomial) is learned in each one. Unlike K -NN however, the shape (and cardinality) of each region is customized to the target function, local approximator, and its location in the input space. This is accomplished by the manner in which the regions are constructed.

As the name implies RP employs a top-down recursive splitting strategy to construct the subregions for local learning. It begins with a single region R_0 containing all of the training data. At each step every existing (“parent”) region is split into two (“daughter”) subregions. These replace each respective parent thereby increasing the number of regions. The splitting is then applied to each daughter. This recursive splitting is continued until a region meets a local (“terminal”) criterion and is not further split. When all regions have met the terminal criterion they represent the cover used for local learning.

The detailed nature (shape) of the terminal regions is governed by the splitting procedure. One defines a splitting function $g(\mathbf{x}, \boldsymbol{\alpha})$ of the input variables \mathbf{x} , characterized by a set of parameters $\boldsymbol{\alpha}$, and a real valued split point s . The form of the split function is usually taken to be linear

$$g(\mathbf{x}, \boldsymbol{\alpha}) = \boldsymbol{\alpha}^t \mathbf{x} \quad (19)$$

(CART) and even more often with restriction to the coordinate (input) axes (CART, C4.5)

$$\boldsymbol{\alpha} \in \{\mathbf{e}_1, \dots, \mathbf{e}_n\}. \quad (20)$$

Here \mathbf{e}_j is a unit vector parallel to the j th input coordinate. The splitting procedure is outlined in Algorithm 1. It takes as its input a (“parent”) region R and produces as its output the two daughters, R_l and R_r that replace it. First optimal parameter values $\boldsymbol{\alpha}^*$ and split point s^* are estimated using the training data contained in the parent region R , based on maximizing some goodness-of-split criterion *splitcri* defining optimality. In most “greedy” implementations this is taken to be the average goodness-of-fit of the local approximations to the training data separately in the two subsets defined by the split. Then the split is performed by thresholding the optimized splitting function at the optimized split point.

Algorithm 1

Splitting procedure used in recursive partitioning.

Split(R) \rightarrow (R_l, R_r)

$$(\boldsymbol{\alpha}^*, s^*) = \arg \max_{\boldsymbol{\alpha}, s} \textit{splitcri}(\{\mathbf{x}_i, y_i \mid \mathbf{x}_i \in R\}, \boldsymbol{\alpha}, s)$$

For all $\mathbf{x} \in R$ do:

$$\text{if } g(\mathbf{x}, \boldsymbol{\alpha}^*) \leq s^* \text{ then } \mathbf{x} \in R_l$$

$$\text{if } g(\mathbf{x}, \boldsymbol{\alpha}^*) > s^* \text{ then } \mathbf{x} \in R_r$$

End for

End split

The recursive partitioning strategy clearly overcomes the major limitations of the K -NN approach. By choice of an appropriate splitting function $g(\mathbf{x}, \boldsymbol{\alpha})$ and split criterion *splitcri*, the shapes of the resulting (“terminal”) regions are locally adapted to the target function and local approximator. Each split chooses the direction $\boldsymbol{\alpha}^*$ (19) (20) that gives the most improvement by the local approximator $\hat{f}_m(\mathbf{x})$ in each daughter. Therefore directions of high bias are preferentially split. Since the regions become more local as the partitioning proceeds this adaptation

is more and more local. In addition an appropriate terminal criterion can use different numbers of training observations in each terminal region, providing further customization.

Another advantage of recursive partitioning is that it lends itself to a simple “eager” (as opposed to lazy) implementation. Owing to the recursive nature of the (binary) splitting it can be parsimoniously represented by a binary tree. The internal nodes of the tree represent the splits, the associated optimal splitting parameters α^* , s^* being stored at each one. The terminal nodes represent the local regions, the parameter values of the local approximator being stored in each one. To make a prediction at a point \mathbf{x} one simply traverses the tree to the (unique) local region containing \mathbf{x} . Starting at the root node one evaluates the value of the split function at \mathbf{x} and then descends to either the left or right daughter depending on whether the value is either less than or equal to, or greater than, the split point. This is done recursively at each daughter visited until a terminal node is reached. The local approximator associated with the corresponding terminal region is then used to make the prediction. Such a traversal can be performed very rapidly even for a large number of terminal regions.

Probably the most attractive feature of RP is its interpretability. The binary tree representation provides a powerful graphic for visualizing the input-output relationship especially when simple axis oriented splits (20) are employed. With the tree one can easily identify the combinations of input variable values leading to various values of the (predicted) output.

Although the intuition leading to RP seems compelling, it suffers from some severe limitations with respect to K -NN that have only been recently recognized. Most of these limitations stem from the fact that it produces a *partition* of the input space. That is the set of terminal regions that cover the space are disjoint rather than overlapping as in K -NN. As a consequence, for similar (average) number of training points per region (variance), RP produces many fewer regions. The result is a much more discontinuous approximation leading to large errors (bias) near region boundaries. Each prediction point is contained in only one region and it can be very far from the region center and training points in that region. This can induce high bias that may overcome the bias reduction achieved by customizing the shape of the region.

Another potential problem with RP is data fragmentation. Each split reduces the training sample in each resulting daughter to an average of one half that of the parent. Thus the average number of splits defining a terminal node is fairly small even for large training samples. In order to reduce the size (13) of a region along a direction α , a split must be performed (along that direction). Since the number of splits leading to most regions is small, its size can be reduced in at most a few directions. This can be an advantage if the bias associated with the local approximations varies only in a small number of directions (locally). But if it varies in more than a few directions this becomes a serious limitation. K -NN on the other hand limits (at least a little) the extent of its regions in all directions.

Finally, the approximations produced by RP are highly unstable with respect to minor perturbations of the training data. This leads to high variance predictions, induced by sampling fluctuations associated with the random nature of the mechanism producing the (training) data. This instability is a consequence of the top-down (greedy) partitioning strategy. Each split is conditioned on its (“ancestor”) splits that precede it. Thus any variability in split parameter estimates is propagated to its descendent splits in a multiplicative (rather than additive) manner. Minor changes in an early split can have a major impact on later splits producing very different terminal regions. On the other hand the regions produced by K -NN are highly stable with respect to perturbations of the training data. Several approaches have been proposed to mitigate this instability aspect of RP. These include “bagging” [Breiman (1994)], “bumping” [Tibshirani and Knight (1995)], and Bayesian CART [George, Chipman, and McCulloch (1996) and Smith, Denison, and Mallick (1996)].

6 Recursive covering

As discussed in Sections 4 and 5 both the K -NN and RP approaches to local learning have complementary strengths and limitations. Recursive covering (“RC”) is a hybrid approach that

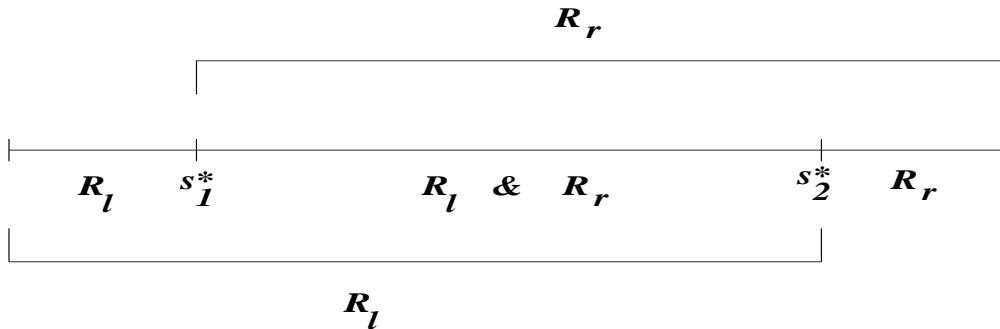


Figure 1: Splitting strategy for recursive covering.

attempts to combine the attractive features of both while mitigating their respective limitations. Following the K -NN approach a large number of highly overlapping regions are produced and a low order polynomial is learned in each one. The region in which a prediction point \mathbf{x} is most centered is used to obtain its output estimate. Like RP the shape of each region is separately customized to the target function $f(\mathbf{x})$, local approximator $\hat{f}_m(\mathbf{x})$ used, and its location in the input space. In addition a data structure exists (“cover tree”) that permits rapid search for the predictor region given the location \mathbf{x} of the prediction point.

A simple modification of the RP splitting paradigm (Algorithm 1) leads to RC. As with RP a splitting function $g(\mathbf{x}, \boldsymbol{\alpha})$ is defined. However with RC there are *two* associated split points $s_1 < s_2$. The splitting procedure is outlined in Algorithm 2. It takes as its input a parent region R , and produces as output its two daughter regions R_l, R_r that replace it. First optimal values of the split parameters are obtained based on the training data in the parent region R by maximizing a (goodness-of-) split criterion. Using the optimized split parameters the split of R is performed. All points in R for which the (optimized) split function has a value less than (or equal to) the upper split point are assigned to the left daughter R_l . All points that have a value greater than the lower split point are assigned to the right daughter R_r .

Algorithm 2

Splitting procedure used in recursive covering.

Split(R) \rightarrow (R_l, R_r)

$$(\boldsymbol{\alpha}^*, s_1^*, s_2^*) = \arg \min_{\boldsymbol{\alpha}, s_1, s_2} \text{splitcri}(\{\mathbf{x}_i, y_i \mid \mathbf{x}_i \in R\}, \boldsymbol{\alpha}, s_1, s_2)$$

For all $\mathbf{x} \in R$ do:

if $g(\mathbf{x}, \boldsymbol{\alpha}^*) \leq s_2^*$ then $\mathbf{x} \in R_l$

if $g(\mathbf{x}, \boldsymbol{\alpha}^*) > s_1^*$ then $\mathbf{x} \in R_r$

End for

End split

Figure 1 illustrates the splitting. The (center) horizontal line represents values of $g(\mathbf{x}, \boldsymbol{\alpha}^*)$, and s_1^*, s_2^* are the two (optimized) split points. All points $\mathbf{x} \in R$ for which $g(\mathbf{x}, \boldsymbol{\alpha}^*) \leq s_1^*$ are assigned to R_l only. Those for which $g(\mathbf{x}, \boldsymbol{\alpha}^*) > s_2^*$ are assigned to R_r only. Those in the interval $s_1^* < g(\mathbf{x}, \boldsymbol{\alpha}^*) \leq s_2^*$ are assigned to *both* daughters R_l and R_r .

For the implementation of the RC strategy presented here (“DART”) the splitting function is taken to be linear (19) with the axis oriented restriction (20) as an option. The split points are (nonadaptive) functions of the split direction $\boldsymbol{\alpha}$. Specifically, $s_1(\boldsymbol{\alpha})$ and $s_2(\boldsymbol{\alpha})$ are taken to be respectively the δ and $(1 - \delta)$ quantiles of $\{\boldsymbol{\alpha}^t \mathbf{x}_i \mid \mathbf{x}_i \in R\}$. Thus, the splitting criterion

$splitcri(\boldsymbol{\alpha})$ (Algorithm 2) is a function of the split direction $\boldsymbol{\alpha}$ only. The “trimming factor” $\delta \leq 1/2$ is a procedural (“meta”) parameter. Generally performance is inversely related to the value chosen for δ (the smaller the better). However, as seen below, the (training) computation increases rapidly as its value approaches zero. The terminal criterion used here is the same as K -NN; a region is not further split when the number of training observations within it is less than a threshold K . The value for K is chosen through model selection (cross-validation). Finally, the splitting criterion $splitcri(\boldsymbol{\alpha})$ used in this implementation is discussed in Section 10.

The recursive covering is accomplished by substituting its splitting procedure (Algorithm 2) in place of that for RP (Algorithm 1) in the RP strategy (Section 5). Beginning with an initial region R_0 , containing all the training data, it is recursively applied to its outputs (daughters) until the terminal criterion is met. When all regions meet the terminal criterion they represent the final region set for local learning. If the trimming factor δ is set to its largest value ($\delta = 1/2$) then one has an RP strategy with median splits at each step. This produces a partition of the input space with N/K disjoint regions. For $\delta < 1/2$, a cover is produced in which the terminal regions overlap. The number of terminal regions depends on the training sample size N , the number of points K in each terminal region, and the value chosen for the trimming factor δ . The number of splits L leading to each terminal region is determined by the terminal criterion

$$(1 - \delta)^L N = K \quad (21)$$

so that

$$L = \log_{(1-\delta)}(K/N). \quad (22)$$

Since each recursive split produces two regions the total number of regions M is

$$M = 2^L = (N/K)^{\log 2 / \log(1-\delta)^{-1}}. \quad (23)$$

Because the regions are overlapping each point in the input space resides in one or more terminal regions. The average number of regions containing each point is given by

$$ave \#\{R_m | \mathbf{x} \in R_m\} = (K/N) \cdot M. \quad (24)$$

Table 1 shows values of M (23) for various combinations of values for δ (columns) and K/N (rows). One sees that the number of regions grows rapidly as both δ and K/N become small. Variance considerations generally restrain K/N from very small values. However, as noted above, smaller values of δ generally improve performance. Thus there is a performance - computation trade-off associated with choosing a value for δ . This is discussed in more detail in Section 8. In terms of (RAM) memory requirements $M = 10^5$ regions are easily accommodated on most lap-top computers, and $M = 10^6$ on scientific work stations. Many more regions can be stored by arranging them in a hierarchical manner on secondary (disk) storage and only reading in those that are necessary to predict each (future) point \mathbf{x} .

Table 1
Number of regions produced by recursive covering.

$\frac{\delta}{K/N}$	1/3	1/4	1/5	1/10
1/10	32	256	1024	2.1×10^6
1/20	128	1024	8192	2.7×10^8
1/100	2048	65536	1.1×10^6	8.8×10^{12}

7 Prediction

The prediction rule for RC is the same as that for K -NN (10) (11). Using exhaustive search to find the region $R_{m^*(\mathbf{x})}$ (11) in which the prediction point is most centered is very slow owing to the very large number of regions (23). One can reduce this computation by arranging the

terminal regions in a binary (“cover”) tree in direct analogy with the partition tree used to represent recursive partitions (Section 5). Each internal node represents a split and stores the corresponding split direction α^* and split points s_1^*, s_2^* . The terminal nodes represent each of the regions of the cover and store the parameters of its corresponding polynomial. Note that the RC splitting strategy used here results in a (nearly perfectly) balanced binary tree of depth L (22). Given a prediction point \mathbf{x} this tree can be recursively traversed in a straight forward manner to visit only those terminal nodes representing regions containing \mathbf{x} . This requires computation proportional to the number of such regions, which on average (24) is much less than the total number of regions (23). Even so, this search time can still be large enough to represent a problem for many applications.

The computation required to find the prediction region $R_{m^*(\mathbf{x})}$ can be further reduced to insignificance, regardless of the size of the cover tree, by a simple trick. The binary cover tree described above is interpreted as a binary *partition* tree. At each internal node the corresponding split direction α^* is stored (as with the cover tree). The (single) split point s^* associated with the node is the midpoint between the corresponding split points for that node in the cover tree

$$s^* = (s_1^* + s_2^*)/2. \quad (25)$$

Interpreting the tree in this manner induces a *partition* on the input space. The number of (disjoint) regions of this partition is the same as that of the associated cover (23). In fact, there is a one-to-one relationship between the regions of this partition and those of the cover; namely, a point that lies (uniquely) in a region of the partition is most centered in its corresponding region of the cover.

As discussed in Section 5 searching a binary partition tree is very fast since only one terminal node is visited (no back tracking). Because the tree here is perfectly balanced the search time is proportional to its depth L (22), and $L = \log_2 M$ (23). This computation is likely to be insignificant for any conceivable number of regions M in the cover.

8 Training

Although the prediction time for recursive covering can be reduced to insignificance the training time cannot. As a function of the parameters of the problem the computation required to construct the entire cover (tree) is given by

$$time(eager) = cN \sum_{l=0}^L [2(1 - \delta)]^l \quad (26)$$

where c is constant depending on the number of inputs n and the computer used, N is the training sample size, L is the tree depth (22), and $\delta \leq 1/2$ is the trimming factor. For $\delta < 1/2$ there is an exponential explosion in computation with increasing L , and L increases with decreasing δ (and K) (22). This is illustrated in Table 2. Shown is the depth of the tree L (column 2) and the computation (26) in arbitrary units (column 3) for several values of δ (first column), with $K/N = 0.05$. The exponential growth is clearly evident. Thus, for any given problem available computing limits how small δ can be.

Table 2

Computation required by eager, lazy, and prudent implementations of RC for $K/N = 0.05$.

δ	depth	eager	lazy	prudent(10^3)	prudent(10^4)
.05	58	3.1×10^{17}	19.0	11.0	8.7
.10	28	3.2×10^8	9.5	3.0	1.8
.15	18	3.4×10^5	6.4	1.0	0.4
.20	13	1200	4.8	0.3	0
.30	8	49.2	3.2	0	0
.40	6	12.9	2.4	0	0
.50	4	5.0	1.9	0	0

If a smaller value of δ is required than can be accommodated with reasonable computation one can adopt a lazy strategy in analogy to the K -NN approach. In this case one waits until a prediction (at \mathbf{x}) is required and then constructs only that part of the cover tree containing the path (from the root) to the region in which \mathbf{x} is most centered. The local polynomial is then fit to the training data in that region and a prediction is made. The computation required for each such prediction is

$$time(lazy) = cN \sum_{l=0}^L (1 - \delta)^l. \quad (27)$$

Here c is the same constant appearing in (26). The fourth column of Table 2 shows this (27) (in the same units as column 3) as a function of δ . One sees that for small to moderate values of δ this strategy is much faster (for an individual prediction) than building the entire tree. Of course, this computation must be repeated for every prediction, whereas with the tree predictions are (relatively) instantaneous. If only a few predictions are to be made, and they are not time critical, a lazy strategy can avoid the massive training computation associated with small values of the trimming factor δ . Owing to its operational similarity to (and in honor of) LOESS [Cleveland and Devlin (1988)] we refer to this lazy strategy as “HYESS” indicating a generalization to higher dimensions.

Comparing the third and fourth columns of Table 2 one sees that the rate of growth of computation with increasing tree depth is much faster for eager training (DART) than for lazy prediction (HYESS). This suggests a hybrid (“prudent”) strategy. Eager training is used to build a cover tree with $\bar{K} > K$ observations in each terminal region. The value of \bar{K} is governed by available computing. For each prediction (at \mathbf{x}) the tree is searched to find the region of this cover in which \mathbf{x} is most centered. HYESS is then applied to the training data in that region for further refinement. Column 5 of Table 2 shows the corresponding prediction time (again in the same units) using this strategy with a tree representing 1000 terminal regions (nodes). The training time to build this tree was 89 units. Column 6 shows the corresponding time for a tree with 10000 regions. Training time for this latter tree was 1200 units. The hybrid strategy clearly improves prediction time while mitigating the exponential growth in training. For $\delta = 0.15$ use of the 1000 node tree reduces prediction time by over a factor of six, while the 10000 node tree reduces it by a factor of 16. For smaller values of δ the relative savings (using these trees) are less but still substantial.

The trade-off between how much computation to invest in training as opposed to savings in prediction depends on how many predictions are to be made and how time critical they are. If there are to be N_p predictions and the goal is to minimize total computation (training plus all predictions) then (26) (27) imply that \bar{K} should be chosen so as to produce a cover tree with $M = N_p$ terminal regions (23). However this is usually not the goal. Training is seldom time critical so that one can afford to commit large resources to it, in order to reduce that of later (perhaps time critical) prediction. However in the early exploratory stages of an analysis one may wish to gauge the potential success of the approach without a large computational

investment. In this case one can employ a purely lazy strategy, or a hybrid one with a small tree, for testing purposes. Then if results appear promising one can commit larger resources to training in order to speed-up future prediction.

9 Model selection

In the implementation of RC presented here there is one problem dependent (meta) parameter. It is the number of training observations K in each terminal region of the final cover. An optimal value for each situation is estimated through model selection. For this implementation cross-validation [Stone (1974)] is employed. An observation is removed from the training sample and its output value y is predicted (\hat{y}) using the remaining observations in the sample. The corresponding error $|y - \hat{y}|$, as a function of K , is averaged over repeated applications of this procedure. The value of K that minimizes this average error is then used in subsequent prediction.

Since with cross-validation only one observation is being predicted each time ($N_p = 1$) it is clear that a purely lazy strategy is appropriate for this purpose. In addition, only one parameter is being estimated, and its precise value is not critical. Thus, the average cross-validated errors (as a function of K) need not be evaluated to high precision. Little (if any) performance is sacrificed by cross-validating a (random) subset of the training observations to obtain an estimate for the optimal value of K , thereby saving computation especially in early exploratory work.

10 Splitting criterion

In all RP implementations the splitting criterion (*splitcri* - Algorithm 1) is heuristically motivated, rather than being derived from first principles. The goal is to find a criterion that when combined with the greedy top-down optimization strategy produces a good solution to the original optimization problem. This is often not the quantity that defines optimality in the original problem itself. With RC (as with RP) the goal is to derive a set of terminal regions such that within each the local approximator $\hat{f}_m(\mathbf{x})$ represents the target $f(\mathbf{x})$ as closely as possible.

From the point of view of a point \mathbf{x} being predicted, RC can be regarded as a multidimensional “peeling” procedure. At each step (region R) a direction $\boldsymbol{\alpha}^*$ in the input space is selected and the points $\mathbf{x}' \in R$ farthest away from \mathbf{x} in that direction $|\boldsymbol{\alpha}^{*t}(\mathbf{x} - \mathbf{x}')|$ are removed to define the next (smaller) region containing \mathbf{x} . Deleting the farthest away points each time keeps \mathbf{x} as centered as possible in subsequent regions used to predict it. If the direction $\boldsymbol{\alpha}^*$ were chosen to maximize distance

$$splitcri(\boldsymbol{\alpha}) = ave_{\mathbf{x}' \in R} |\boldsymbol{\alpha}^t(\mathbf{x} - \mathbf{x}')| \quad (28)$$

one would reproduce the (linear) K -NN procedure in which neither the output values y nor those of the local approximator $\hat{f}_m(\mathbf{x})$ are used to customize the regions. The goal with RC is to choose the direction $\boldsymbol{\alpha}^*$ such that when the (far away) points are removed, the approximator provides the best fits in the resulting daughter regions. Although such a strategy could be directly implemented it might prove to be too computationally intense for some (nonconstant) local approximators. The splitting criterion suggested here (29) (30) is motivated by (18) and is intended to be a computationally feasible approximation to this strategy.

Let

$$\{r_i = y_i - \hat{f}_R(\mathbf{x}_i) \mid \mathbf{x}_i \in R\} \quad (29)$$

be the residuals from the local approximator $\hat{f}_R(\mathbf{x})$ fit to the data in the region R being split. The criterion (to be maximized) for selecting the splitting direction $\boldsymbol{\alpha}^*$ (Algorithm 2) is

$$splitcri(\boldsymbol{\alpha}) = |ave_{\mathbf{x}_i \in R} \{r_i \mid \mathbf{x}_i \leq s_1(\boldsymbol{\alpha})\}| + |ave_{\mathbf{x}_i \in R} \{r_i \mid \mathbf{x}_i > s_2(\boldsymbol{\alpha})\}|. \quad (30)$$

Here $s_1(\boldsymbol{\alpha})$ and $s_2(\boldsymbol{\alpha})$ are respectively the lower and upper split points (δ and $1 - \delta$ quantiles of $\{\boldsymbol{\alpha}^t \mathbf{x}_i \mid \mathbf{x}_i \in R\}$). Maximizing (30) finds the direction $\boldsymbol{\alpha}^*$ in which the points to be removed

(defining the next two daughters) most *systematically* deviate from the local approximation in the region being split. It can be feasibly optimized since the local approximator $\hat{f}_R(\mathbf{x})$ need only be computed once and can be rapidly obtained from that of its parent region through least-squares updating formulae. Also, only a subset of the data in each trial direction α need be considered, namely the $2 \cdot \delta$ points at the extremes. The set of directions over which (30) is optimized are taken to be the original coordinate axes (20) and the direction $\hat{\alpha}_R$ defined by a linear least squares fit of the training outputs to the inputs in R . This direction can be rapidly obtained from that of the parent region through updating formulae.

The validity of any splitting criterion is established by how well it produces the desired goal. Although the one adopted here (29) (30) appears to lead to good performance (Section 12) it is unlikely to be the “last word”. Other (better) criteria may be developed in the future that lead to superior performance, at least in some situations.

11 “Admissibility” example

In this section results are presented of applying this RC implementation (DART/HYESS) to a target function for which it might be expected to perform well. The target is taken to be

$$f(\mathbf{x}) = \frac{10^6 \prod_{j=1}^{10} x_j^2}{1 + 100 \prod_{j=1}^{10} x_j^2}. \quad (31)$$

There are thus $n = 10$ input variables. They are randomly generated from a uniform distribution $\mathbf{x} \sim U^{10}[0, 1]$. Each corresponding output y is generated from (1) with no error ($\varepsilon = 0$). The training sample size was taken to be $N = 500$. Shown in Table 3 are the percentiles (columns) of the distribution of absolute error $|y - \hat{f}(\mathbf{x})|$ on an independent validation data set of 5000 observations, averaged over five (random) training samples. The first row of Table 3 shows this distribution for DART/HYESS ($\delta = 0.1$), and the remaining rows the corresponding values for three competitors. For K -NN, the value of K was chosen by cross-validation and the metric (15) (16) was defined by the interquartile range on each input over the training data. MARS [Friedman (1991)] is an extension of RP (CART) intended to produce continuous approximations. The last row (“linear”) is a global linear least squares fit.

Table 3
Percentiles of the distribution of $|y - \hat{f}(\mathbf{x})|$ for several methods applied to (31).

method	50	75	90	95
DART/HYESS	.015	.385	6.31	26.8
K -NN	.244	2.23	14.0	42.5
MARS	2.59	8.00	64.0	200.3
linear	29.4	50.8	91.9	135.0

One would not expect a (global) linear model to do well on such a complex function (31). Also, this target is symmetric in the input variables, thereby making them all equally relevant. Thus, procedures based on subset selection, such as MARS, might not be highly successful, whereas K -NN which treats the inputs in a (globally) symmetric manner ought to be better suited to such a target. This is verified in Table 3. Although the target is symmetric, the respective inputs have high differential relevance at different locations of the input space. RC exploits this by customizing the shapes of individual regions to improve performance, here dramatically.

12 Random function generator

The target function (31) was deliberately selected to illustrate the power achievable with RC in some situations. The relative success of any method depends on the details of the problem to

which it is applied, most notably the particular target function (3) encountered. Every method has particular targets for which it is most appropriate and others for which it is not. In order to gain a (somewhat) broader perspective on the relative merits of the RC approach it is applied to a variety of randomly generated targets. Each one takes the form

$$f(\mathbf{x}) = \sum_{l=1}^L a_l h(\mathbf{x}, \mathbf{z}_l, \mathbf{V}_l) \quad (32)$$

with

$$h(\mathbf{x}, \mathbf{z}, \mathbf{V}) = \frac{1}{|\mathbf{V}|} \exp \left[-\frac{1}{2} (\mathbf{x} - \mathbf{z})^t \mathbf{V}^{-1} (\mathbf{x} - \mathbf{z}) \right]. \quad (33)$$

This is a linear combination of Gaussian functions. The coefficients (32) are randomly generated from a uniform distribution $\{a_l \sim U[-1, 1]\}_1^L$. The parameters of the Gaussians are also randomly selected. The locations are generated from a uniform distribution $\{\mathbf{z}_l \sim U^n[0, 1]\}_1^L$. The eigenvectors of each \mathbf{V}_l are randomly generated uniformly on the unit n -sphere subject to an orthogonality constraint. The eigenvalues are generated from a uniform distribution $U[0.525\sqrt{n}, 0.025\sqrt{n}]$. Depending on the actual (random) values generated for each set of parameters, a wide variety of different target functions, in terms of the geometric shapes of their contours, can be realized.

The simulation studies presented here consist of applying several versions of RC (DART/HYESS) and a set of competing methods to 150 such randomly generated functions (32) (33) with $n = 10$ input variables. Each target was learned using a training data set of $N = 500$ observations. The input points for each were randomly generated from a uniform distribution $\{\mathbf{x}_i \sim U^{10}[0, 1]\}_1^{500}$. The corresponding output values were generated from (1) with $f(\mathbf{x})$ scaled so that

$$E |f(\mathbf{x}) - \bar{f}| = 1. \quad (34)$$

The random (error) component is normally distributed $\{\varepsilon_i \sim N[0, \sigma^2]\}_1^{500}$. Its variance σ^2 was chosen so that $E |\varepsilon_i| = 0.2$. This introduces an irreducible prediction error of 20%, or about a 5/1 signal-to-noise ratio.

For each target function the median absolute ‘‘model’’ error for each method j

$$e_j = \text{med} |f(\mathbf{x}) - \hat{f}_j(\mathbf{x})| \quad (35)$$

was computed over 5000 independently generated ‘‘test’’ observations. In order to compare various methods over problems of varying difficulty the actual quantity used for comparison is

$$r_j = e_j / \min_k e_k. \quad (36)$$

That is the model error for each method is normalized by that of the best method (being compared) for each target. Thus the best (lowest e_j) method j^* receives a value (36) of $r_{j^*} = 1$ and all others have larger values of r_j for each target.

The total simulation study of 150 target functions is divided into three parts of 50 targets (32) (33) each. The first 50 are unimodal targets ($L = 1$), the next 50 trimodal ($L = 3$), and the last 50 are decimodal ($L = 10$). Relative performance for each method j is provided by the distribution of its r_j (36) values over the 50 targets in each study. Note that if a particular method was best for all 50 target functions its resulting distribution would be a point mass at the value 1.0.

Figure 2 summarizes these distributions with boxplots, for seven methods, on the unimodal ($L = 1$) targets. The dark area of each boxplot shows the interquartile range of the distribution with the enclosed white bar being the median. The outer hinges represent the points closest to (plus/minus) 1.5 interquartile range units from the (upper/lower) quartiles. The isolated narrow (dark) bars represent individual points outside this range (outliers). The bars at the value 10

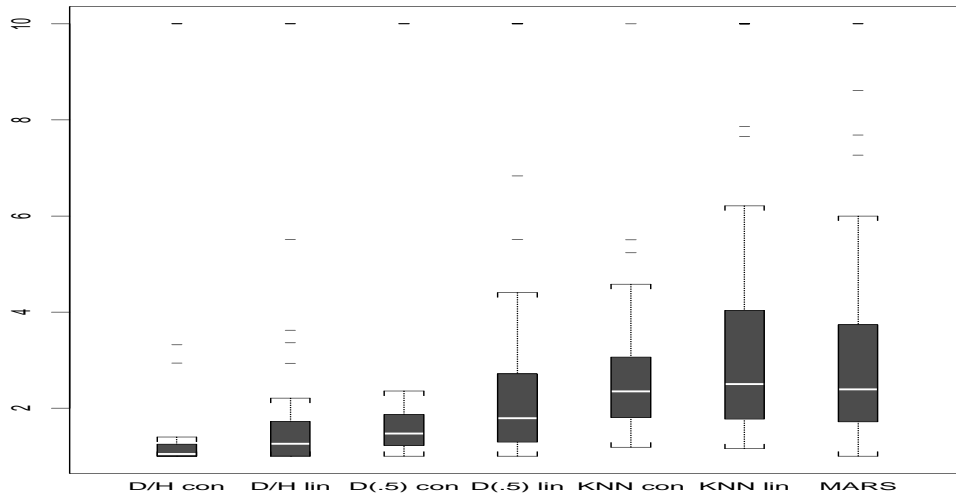


Figure 2: Distribution of relative errors for the unimodal ($L = 1$) target functions.

indicate the existence of extreme outliers with values $r_j > 10$. There were typically two to three such points for each method.

The first two methods (left to right) shown in Fig. 2 represent RC, namely DART/HYESS with trimming factor $\delta = 0.1$. The first uses a local constant (zero-degree polynomial) approximator and the second a local linear one. The next two methods represent RP, namely DART with $\delta = 0.5$, again using local constant and linear approximators. Comparison of these two ($\delta = 0.5$) with the first two ($\delta = 0.1$) allows one to gauge the relative effectiveness of the RC strategy with respect to that of RP, since all other aspects of the procedures (split function (19), splitting criterion (30), and terminal criterion) are the same. The next two methods represent the K -NN approach using respectively local constant and linear approximators. The last method being compared here is MARS [Friedman (1991)]. For DART/HYESS and K -NN the terminal region size K was estimated by cross-validation separately for each target. For K -NN the metric (15) (16) was defined by the interquartile ranges of the inputs.

The first (and perhaps most important) thing to observe from Fig. 2 is that none of the methods was best for all 50 target functions. Each had targets for which it did well and others on which it did very badly (relative to the best). Except for the two K -NN procedures each method out performed all others on at least one of the targets. The respective number of times each was best (Fig. 2 - left to right) was 23, 13, 4, 3, 0, 0, and 6. For the local learning methods, local constant fitting seems to give better performance than local linear on average, but by no means every time. Comparing the first four methods one sees that RC tends to out perform RP typically producing a 60% reduction in median absolute error (35) over these targets. RC tends to provide fairly dramatic improvement over the K -NN procedures typically reducing this error by a factor of 2.2. The relatively poor performance of MARS on these targets is not a surprise. It is an axis oriented procedure and these target functions (32) (33) have no preferred orientation with respect to the input coordinate axes. In other simulation studies (not shown) in which the covariance matrices in (33) were restricted to being diagonal, the relative performance of the first six methods was quite similar to that shown in Fig. 2. However, that for MARS was much better yielding a distribution quite similar to that for DART/HYESS (local constant) with $\delta = 0.1$.

Figure 3 shows the respective distributions of the seven methods for the 50 trimodal ($L = 3$)

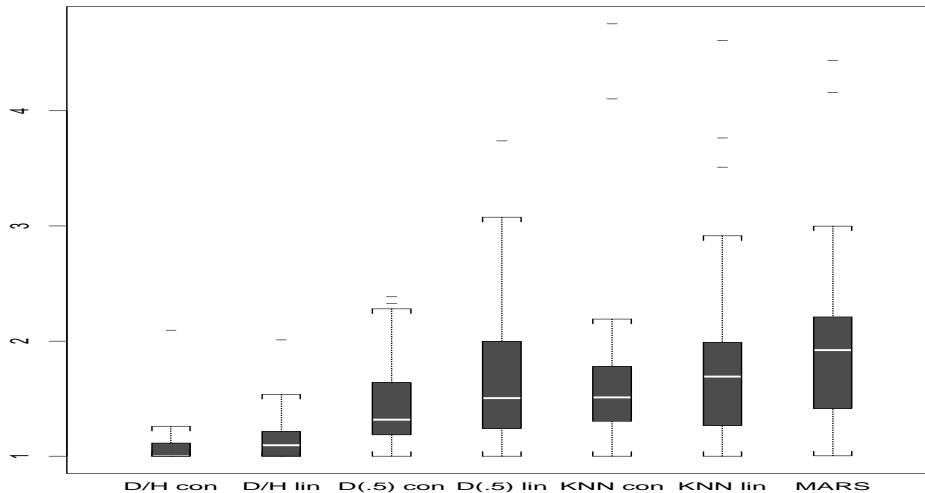


Figure 3: Distribution of relative errors for the trimodal ($L = 3$) target functions

functions (32) (33), and Fig. 4 shows them for the 50 decimodal ($L = 10$) ones. One sees generally similar (relative) results. For Fig. 3 the number of times each respective method (left to right) was best was 28, 15, 2, 1, 3, 1, and 0. For Fig. 4 these numbers were 34, 7, 0, 0, 8, 1, and 0. As seen RC tends to more consistently out perform the others on these more complex functions, but the magnitude of its gain is less. (Note the changing vertical scale.) All methods have increased difficulty with these more complicated functions and there is generally an upper limit on bad performance, namely that of a globally constant fit.

13 Discussion

Although this simulation study based on 150 target functions is more extensive than testing on only a few targets (either simulated or from real data) it is far from being totally comprehensive. The aim was to produce unknown difficult targets that do not necessarily conform to any obvious special structure. Such targets should be generally more suited to nonparametric local learning methods than other approaches that seek to exploit such special structure. For example, if the target can be closely represented by a additive function

$$f(\mathbf{x}) \simeq \sum_{j=1}^n f_j(x_j) \quad (37)$$

then additive modeling [Hastie and Tibshirani (1990)] should provide better performance. If it is close to being additive in a few linear combinations of the inputs

$$f(\mathbf{x}) \simeq \sum_{m=1}^M f_m(\boldsymbol{\alpha}_m^t \mathbf{x}) \quad (38)$$

then projection pursuit [Friedman and Stuetzle (1981)] or neural networks [Rumelhart, Hinton, and Williams (1986)] will likely work better. Finally, if the target can be well approximated by a sum of functions each of which dominately involve a small number of (unknown) input variables

$$f(\mathbf{x}) \simeq \sum_{m=1}^M f_m(\text{few } x_j) \quad (39)$$

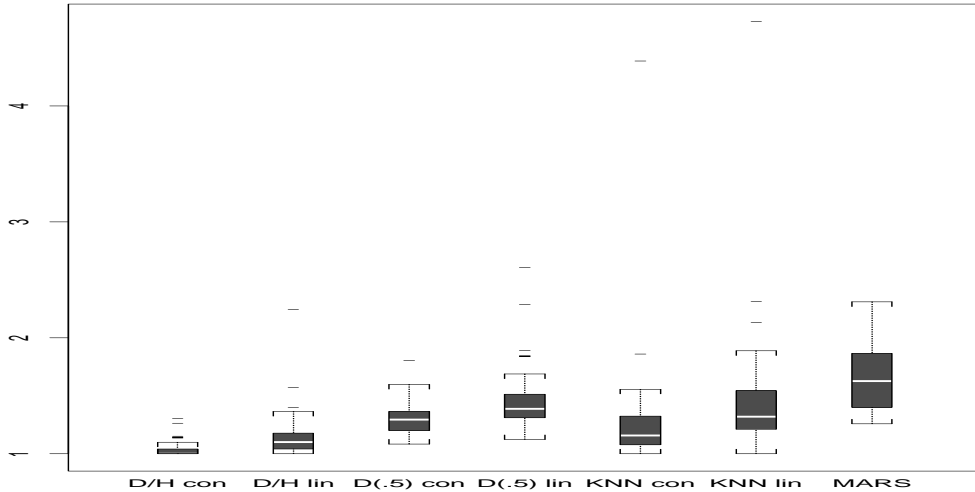


Figure 4: Distribution of relative errors for the decimodal ($L = 10$) target functions.

then tensor product methods such as MARS [Friedman (1991)] should dominate. The reason why local learning methods do not necessarily perform well on such targets with these special properties is that local learning explicitly constructs *convex* equivalent kernels [Hastie and Tibshirani (1990)] whereas targets like (37) (38) (39) are best approximated by procedures involving (implicit) equivalent kernels with highly *concave* shapes.

The relative performance among any group of methods strongly depends on the specific properties of the problems over which they are compared. The results of such comparisons should not be extrapolated beyond that particular problem set. The results of the study presented here suggest that for classes of target functions for which local learning methods are appropriate, RC may provide an advantage over some other popular local learning methods such as K -NN and RP.

14 Previous work

Modifications to RP (other than RC) have been proposed to mitigate its limitations. These are mostly based on the notion of “soft” partitioning (“SP”) [O’Sullivan (1991), Quinlan (1993), and Jordan and Jacobs (1994)]. With these methods the splitting function (19) and split point s are replaced by a “mass partition” function $0 \leq m(\boldsymbol{\alpha}^t \mathbf{x} - s) \leq 1$. A common choice is the logistic function

$$m(z) = 1/(1 + e^{-z}). \quad (40)$$

For each split the mass associated with each input point \mathbf{x} is divided into two parts based on its value of the partition function; $m(\boldsymbol{\alpha}^t \mathbf{x} - s)$ of its mass is assigned to the left daughter and its remaining mass $1 - m(\boldsymbol{\alpha}^t \mathbf{x} - s)$ assigned to the right daughter. As with RP the parameters $\boldsymbol{\alpha}$ and s are chosen to optimize some splitting criterion. The result of recursive application of this approach can be viewed as a partition of the input space in which each point \mathbf{x} is assigned to all of the regions, with differing mass (weight) in each one, depending on its location. Predictions are made by taking the corresponding weighted average of the local approximators over all regions.

SP is an attractive approach qualitatively sharing many of the attributes of the RC strategy. The principal advantage of the RC approach is computational. Since (whole) training points are removed at each step updating formulae can be employed to rapidly compute the splitting

criterion and local approximations in each region from those of its parent. This permits the construction of much larger trees. Prediction is also very much faster (even with much larger trees) using the trick outlined in Section 7. In addition, there is no clear analog of the lazy or hybrid strategies (Section 8) in the context of SP for employing even larger region sets with reasonable computation. With SP the entire tree must be built, and traversed so as to visit all terminal nodes (regions) in which the prediction point has substantive (relative) mass.

The computational advantage of RC translates into statistical advantage in those situations (target functions) for which a large number of regions is advantageous. Among these are target functions where the fragmentation aspect of RP (Section 5) represents a problem. As seen in Table 2 smaller values of δ allow for much deeper trees (either explicitly, or implicitly through the lazy/hybrid strategy) thereby allowing for many more splits directions to define each region.

From the other direction techniques have been proposed to mitigate the limitations of K -NN [Friedman (1994), Hastie and Tibshirani (1995), and Friedman, Kohavi, and Yun (1995)]. Like RC they induce a large number of overlapping regions and attempt to customize the shape of each to account for the local properties of the target function. As with K -NN they employ a purely lazy strategy. Again the primary advantage of RC over these methods is computational owing to the prediction trick (Section 7) and its eager/hybrid implementation (Section 8). Also, the detailed strategy for customizing the regions is quite different with this implementation of RC. Whether this results in improved performance will depend on the particular situation (target function) encountered.

15 Future work

The basic concept of RC may have application beyond that developed here. In the learning context an obvious extension is to classification where the output variable y assumes J (unordered) categorical values. Here the goal is to learn J target functions $\{f_j(\mathbf{x}) = \Pr(y = j | \mathbf{x})\}_1^J$ which are the probabilities (at \mathbf{x}) that the output realizes each of its respective values. These estimates are then used in a decision rule to make a class assignment for input vectors of unknown class. The RC implementation presented here may work well for this purpose. However, as discussed in Friedman (1996), the bias-variance trade-off associated with classification is very different than that for regression (12). Hence the bias reduction strategy inherent in this implementation of RC may not be the most effective for classification, and alternative strategies may provide superior performance. Also, the RC approach presented here is oriented towards real valued inputs. Inputs with categorical values are incorporated by transforming them to 0/1-valued “dummy” variables. Although this can be effective, analogs of more sophisticated strategies for handling categorical variables [Friedman, Kohavi, and Yun (1995)] may work better in this context.

In addition to the learning problem, RP (“divide and conquer”) strategies have been applied to a variety of search problems in computational geometry and data base management [Bentley and Friedman (1979)]. There may be applications among these for which the substitution of an RC strategy could lead to improved performance. For example, in closest point problems [Friedman, Bentley, and Finkle (1977)] use of RC with a splitting criterion like (28) may lead to faster search algorithms than those based on RP (“k-d trees”). This remains to be seen and is left to future work. RP has been (and is) a powerful paradigm for solving many different types of problems. Substituting the corresponding RC strategy may lead to improvement in some cases.

References

- [1] Atkeson, C., Moore, A., and Schaal, S. (1996). Locally weighted learning. *Artificial Intelligence Review* (to appear).
- [2] Bellman, R. E. (1961). *Adaptive Control Processes*. Princeton University Press.

- [3] Bentley, J. and Friedman, J. H. (1979). A survey of algorithms and data structures for range queries. *ACM Computing Surveys* **11**, 397.
- [4] Bottou, L. and Vapnik, V. (1992). Local learning algorithms. *Neural Comp.* **4**, 888.
- [5] Breiman L. (1994). Bagging predictors. Technical Report, Dept. of Statistics, University of California, Berkeley.
- [6] Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth.
- [7] Cleveland, W. S. and Devlin, S. J. (1988). Locally weighted regression: An approach to regression analysis by local fitting. *J. Amer. Statist. Assoc.* **83**, 596.
- [8] Friedman, J. H. (1991). Multivariate adaptive regression splines (with discussion). *Ann. Statist.* **15**, 1.
- [9] Friedman, J. H. (1994). Flexible metric nearest neighbor classification. Technical Report, Dept. of Statistics, Stanford University.
- [10] Friedman, J. H. (1996). On bias, variance, 0/1 - loss, and the curse-of-dimensionality. Technical Report, Dept. of Statistics, Stanford University.
- [11] Friedman, J. H., Bentley, J., and Finkle, R. (1977). An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Software* **3**, 209.
- [12] Friedman, J. H. and Stuetzle, W. (1981). Projection pursuit regression. *J. Amer. Statist. Assoc.* **76**, 817.
- [13] Friedman, J. H., Kohavi, R. and Yun, Y. (1995). Lazy decision trees. Technical Report, Dept. of Computer Science, Stanford University.
- [14] George, E. I., Chipman, H. and McCulloch, R. E. (1996). Bayesian CART. Proceedings: *Computer Science and Statistics 28th Symposium on the Interface*. Sydney, Australia.
- [15] Hastie, T. J. and Tibshirani R. J. (1990). *Generalized Additive Models*. Chapman & Hall.
- [16] Hastie, T. J. and Tibshirani, R. J. (1995). Discriminant adaptive nearest neighbor classification. Technical Report, Dept. of Statistics, Stanford University.
- [17] Jordan, M. I. and Jacobs, R. A. (1994). Hierarchical mixtures of experts and the EM algorithm. *Neural Comp.* **6**, 181.
- [18] O'Sullivan, F. (1991). Discussion: Multivariate adaptive regression splines. *Ann. Statist.* **15**, 99.
- [19] Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Kaufmann Publishers, Inc.
- [20] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1, Foundations, pp 318. MIT Press.
- [21] Smith, A. F. M., Denison, D. G. T., and Mallick, B. K. (1996). Bayesian curves and CARTs. Proceedings: *Computer Science and Statistics 28th Symposium on the Interface*. Sydney, Australia.
- [22] Stone, C. J. (1977). Nonparametric regression and its applications (with discussion). *Ann. Statist.* **5**, 595.
- [23] Stone, M. (1974). Cross-validatory choice and assessment of statistical predictors (with discussion). *J. Roy. Statist. Soc. Ser. B* **36**, 111.

- [24] Tibshirani, R. J. and Knight, K. (1995). Model search and inference by bootstrap “bumping”. Technical report Dept. of Statistics, University of Toronto.