An overview of regression techniques for knowledge discovery

Article *in* The Knowledge Engineering Review · December 1999

DOI: 10.1017/S026988899900404X

2 authors:

İlhan Uysal
Deniz Harp Okulu - Tuzla Istanbul
9 PUBLICATIONS 168 CITATIONS
SEE PROFILE

Halil Altay Güvenir
Bilkent University
120 PUBLICATIONS 2,179 CITATIONS
SEE PROFILE

The Knowledge Engineering Review, Vol. 14:4, 1999, 319–340
Printed in the United Kingdom
Copyright © 1999, Cambridge University Press

An overview of regression techniques for knowledge discovery*

İLHAN UYSAL and H. ALTAY GÜVENIR

Department of Computer Engineering and Information Sciences, Bilkent University, 06533 Ankara, Turkey
(email: {uilhan,guvenir}@cs.bilkent.edu.tr)

Abstract

Predicting or learning numeric features is called *regression* in the statistical literature, and it is the subject of research in both machine learning and statistics. This paper reviews the important techniques and algorithms for regression developed by both communities. Regression is important for many applications, since lots of real life problems can be modeled as regression problems. The review includes Locally Weighted Regression (LWR), rule-based regression, Projection Pursuit Regression (PPR), instance-based regression, Multivariate Adaptive Regression Splines (MARS) and recursive partitioning regression methods that induce regression trees (CART, RETIS and M5).

1 Introduction

Predicting the values of numeric or continuous attributes is known as *regression* in the statistical literature, and it is a research area for many researchers in this field. Predicting real values is also an important topic for machine learning. Most of the problems that humans learn in real life, such as sporting abilities, are continuous. Dynamic control is one such problem which is the subject of research in machine learning. For example, learning to catch a ball, moving in a three-dimensional space, is an example of this problem which is studied in robotics. In such applications, machine learning algorithms are used to control robot motions, where the response to be predicted by the algorithm is a numeric or real-valued distance measure and direction. As an example of such a problem, Salzberg and Aha (1994) proposed an instance-based learning algorithm for a robot control task in order to improve a robot's physical abilities.

In machine learning, most research has been done for classification, where the single the predicted feature is nominal or discrete. Regression differs from classification in that the output or predicted feature in regression problems is continuous. Even though most of the research in machine learning is concentrated on classification, recently the focus of the machine learning community has moved strongly towards regression, since a large number of real-life problems can be modeled as regression problems. Various names are used for this problem in the literature, such as functional prediction, real value prediction, function approximation and continuous class learning. We prefer its historical name, *regression*, in this paper.

In designing expert systems, an important topic in knowledge engineering, induction techniques developed in machine learning and statistics have become important, especially for cases where a

* This research is supported, in part, by TUBITAK (Scientific and Technical Research Council of Turkey) under Grant 198E015.

domain expert is not available, or the knowledge of experts is tacit or implicit (Adeli, 1990; Peterson et al., 1990). These techniques are also important to discover new rules, even in cases where domain experts or formal domain knowledge is available (McTear & Anderson, 1990). Probably the most important advantage of induction techniques is that they enable us to extract knowledge automatically.

By the term "knowledge", we mean two types of information. One is the information used for prediction of a new case, given example cases; the other is the information used for extracting new rules about the domain which have not yet been discovered, by interpreting induced models. To help the reader to determine which regression techniques are suitable for each type of knowledge, a comparison is given in section 9 by including other important properties. The techniques covered in this paper can be employed in such systems by knowledge engineers, when the underlying problem is formalized as a prediction of a continuous target attribute.

The idea behind using induction for knowledge systems is particularly accepted by a newly emerged discipline, Knowledge Discovery in Databases (KDD), which incorporates researchers from various disciplines (Fayyad et al., 1996*a*, *b*; Weiss & Indurkhya, 1998). The main source of knowledge in this field is large databases. Since databases can store large amounts of data belonging to many different domains, the use of automatic methods such as induction for knowledge discovery is viable, because it is usually difficult to find an expert for each different domain or relation in databases. Today, database management systems enable only deductive querying. Incorporating an inductive component into such databases to discover knowledge from different domains automatically is the long-term expectation from this new field (Imielinski & Mannila, 1996). This particularly requires the cooperation of knowledge engineers and database experts. Such expectations make regression an important tool for the stand-alone or domain-specific KDD systems today, and Knowledge and Data Discovery Management Systems (Fayyad et al., 1996*a*; Weiss & Indurkhya, 1998) in the future.

In the paper, we review most current regression techniques developed in machine learning and statistics. After describing the main focus for the development of new techniques in the next section, we review instance-based regression, locally weighted regression, rule-based regression, projection pursuit regression, tree-based regression and multivariate adaptive regression splines, respectively, in sections 3–8. After a comparison of techniques described in section 9, for both important properties of these techniques and performance, we conclude by addressing future work in section 10.

2 Parametric vs. non-parametric regression

The most common approach in regression is to fit the data to a global parametric function. Simple regression in statistical analysis is an example of parametric learning. This model includes a dependent variable y and predictor (independent) variables (x's), and states that the value of y changes at a constant rate as the value of any independent variable changes. Thus, the functional relationship between y and x's is a straight line:

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} + \varepsilon_i \tag{1}$$

The subscript i denotes the observations or instances; the second subscript designates p independent variables. There are $p + 1$ parameters, $\beta_j, j = 0, \ldots, p$, to be estimated. As can be seen from the simple regression example, in the parametric model, the structure of the function is given, and the procedure estimates the parameters, β_j, according to a fitting criterion. This criterion is generally a minimization of an error function for all data points in a training set. Very often, this is a *least squares* criterion, which minimizes the sum of the squares of the prediction errors of the estimated linear function for all instances. The error term ε_i denotes the error of estimation for each instance i, and it is assumed to be normally distributed.

In machine learning, parametric learning methods have found wide application areas in the form of neural networks, and they have been very successful when the assumed structure of the function is sufficiently close to the function which generated the data to be modeled. However, the aim in machine learning is to find a general structure rich enough to model a large fraction of all possible functions. This idea leads us to non-parametric and nonlinear regression methods, where no assumption is made about the structure of the function or about the distribution of the error.

Most popular current non-parametric regression and learning techniques to predict a numeric feature are reviewed in the following sections.

In the rest of the paper, training set D is represented by the instance matrix $\mathbf{X}$, where rows represent instances and columns represent predictors, and a response vector $\mathbf{y}$ represents the continuous or numeric response to be predicted for all instances. Estimated values of y are shown with a column vector $\bar{\mathbf{y}}$, where $\bar{y}_i$ is a scalar of the vector. Coefficients in equation (1) are represented by a column vector $\boldsymbol{\beta}$. Any instance or any row in the instance matrix is represented by $\mathbf{x}_i$, where $i = 1, \ldots, n$ and n is the number of instances in the training set. Any column of $\mathbf{X}$ is represented by x_j, where $j = 1, \ldots, p$, and p is number of predictor features. x_{ij}, y_i and β_j represent scalars of $\mathbf{X}, \mathbf{y}$ and $\boldsymbol{\beta}$, respectively. For the operations where $\boldsymbol{\beta}$ is included, a column consisting only of constant 1 values is inserted into the instance matrix as the first row so as to enforce the first term in equation (1) $(j = 0, \ldots, p)$. The notations x_j and y are used as variables to represent predictor features and a response feature, respectively. To denote instance vectors $(\mathbf{x}_i)$ with a variable, $\mathbf{x}$ is used. To represent residuals, a column vector $\mathbf{r}$ is used, where r_i, $i = 1, \ldots, n$, is a scalar of it. To denote a query instance, a row vector $\mathbf{q}$ is used.

3 Instance-based regression

Instance-Based Learning (IBL) algorithms are very popular, since they are computationally simple during the training of instances (Aha et al., 1991; Dasarathy, 1991). In most applications, training is done simply by storing the instances. This section describes the application of this technique for regression (Kibler et al., 1989).

In instance-based regression, each instance is usually represented as a set of attribute value pairs, where the values are either nominal or numeric, and the value to be predicted is continuous. The problem to be solved is, for a given query instance, to predict the target value as a function of other instances whose target values are known. The *nearest neighbour* is the most popular instance-based algorithm. The target values of the most similar neighbours are used in this task. Here the similarity is the negation of the Euclidean distance between instances. Formally, if we let real numbers, R, be a numeric domain, and $\mathbf{X}$ be an instance space with p attributes, we can then describe the approximation function, F, for predicting numeric values as follows:

$$F(x_1, \ldots, x_p) = \bar{y}_i \quad \text{where } \bar{y}_i \in R \tag{2}$$

There is a variety of instance-based algorithms in the literature. Here, the simplest one, the *proximity algorithm*, is described in Figure 1. The proximity algorithm simply saves all training instances in the training set. The normalization algorithm maps each attribute value into the continuous range (0–1). The estimate $\bar{y}_t$ for test instance $\mathbf{x}_t$ is defined in terms of a weighted

```
[1]    $\forall \mathbf{x}_i \in$ Training Set
[2]        $normalize(\mathbf{x}_i)$
[3]    $\forall \mathbf{x}_t$Test Set
[4]        $normalize(\mathbf{x}_t)$
[5]        $\forall \mathbf{x}_i\{\mathbf{x}_i \neq \mathbf{x}_t\}$: Calculate $Similarity(\mathbf{x}_t, \mathbf{x}_i)$
[6]        Let $Similars$ be set of $N$ most similar instances to $\mathbf{x}_t$ in Training Set
[7]        Let $Sum = \sum_{\mathbf{x}_i \in Similars} Similarity(\mathbf{x}_t, \mathbf{x}_i)$
[8]        Then $\bar{y}_t = \sum_{\mathbf{x}_i \in Similars} \frac{Similarity(\mathbf{x}_t, \mathbf{x}_i)}{Sum} F(\mathbf{x}_i)$
```

Figure 1 The proximity algorithm.

similarity function of $\mathbf{x}_t$'s nearest neighbours in the training set. The similarity of two normalized instances is described by equation (3):

$$Similarity(\mathbf{x}_t, \mathbf{x}_i) = \sum_{j=1}^{p} Sim(x_{tj}, x_{ij}) \tag{3}$$

where $Sim(x, y) = 1.0 - |x - y|$, and t_i.

The assumption in this approach is that the function is locally linear. For sufficiently large sample sizes, this technique yields a good approximation for continuous functions. Another important property of instance-based regression is its incremental learning behaviour. By default, instance-based regression assumes that all the features are equivalently relevant. However, the prediction accuracy of this technique can be improved by attaching weights to the attributes. To reduce the storage requirements for large training sets, averaging techniques for the instances can be employed (Aha et al., 1991). The most important drawback of instance-based algorithms is that they do not yield abstractions or models that enable interpretation of the training sets (Mitchell, 1997).

4 Locally weighted regression

Locally Weighted Regression (LWR) is similar to the nearest neighbour approach described in the previous section, especially for three main properties. First, the training phases of both algorithms include just storing the training data, and the main work is done during prediction. Such methods are also known as *lazy learning* methods. Secondly, they predict query instances by strongly influencing nearby or similar training instances. Thirdly, they represent instances as real-valued points in p-dimensional Euclidean space. The main difference between IBL andLWR is that, while the former predicts instances by averaging the nearby instances, the latter makes predictions by forming an averaging model at the location of query instance. This local model is generally a linear or nonlinear parametric function. After a prediction for query instance is done, this model is deleted, and for every new query a new local model is formed, according to the location of the query instance. In such local models, nearby instances of the query have large weights on the model, and distant instances have fewer or no weights. For a detailed overview of the locally weighted methods, see Atkenson et al. (1997), from where the following subsections are summarized.

4.1 Nonlinear local models

Nonlinear local models can be constructed by modifying global parametric models. A general global model can be trained to minimize the following training criterion:

$$C = \sum_i L(f(\mathbf{x}_i, \boldsymbol{\beta}), y_i) \tag{4}$$

where y_i is the response value corresponding to the input vector $\mathbf{x}_i$, $\boldsymbol{\beta}$ is the parameter vector for the nonlinear model $\bar{y}_i = f(x_i, \beta)$, and L is the general loss function for predicting y_i. If this model is a neural net, then the $\boldsymbol{\beta}$ will be a vector of the synaptic weights. If we use least squares for the loss function L, the training criterion will be

$$C = \sum_i (f(\mathbf{x}_i, \boldsymbol{\beta}) - y_i)^2 \tag{5}$$

To ensure points nearby to the query have more influence in the regression, we can add a weighting factor to the criterion:

$$C(q) = \sum_i [L(f(\mathbf{x}_i, \boldsymbol{\beta}), y_i))K(d(\mathbf{x}_i, \mathbf{q}))] \tag{6}$$

where K is the weighting or *kernel* function, and $d(\mathbf{x}_i, \mathbf{q})$ is the distance between the data point $\mathbf{x}_i$ and the query $\mathbf{q}$. Using this training criterion, f becomes a local model, and can have a different set of parameters for each query point.

4.2 Linear local models

The well-known linear global model for regression is simple regression (1), where least squares approximation is used as the training criterion. Such linear models can be expressed as

$$\mathbf{x}_i\boldsymbol{\beta} = y_i \tag{7}$$

where $\boldsymbol{\beta}$ is the parameter vector. Whole training data can be defined with the following matrix equation:

$$\mathbf{X}\boldsymbol{\beta} = \mathbf{y} \tag{8}$$

where $\mathbf{X}$ is the training matrix whose ith row is $\mathbf{x}_i$ and $\mathbf{y}$ is a vector whose ith element is y_i. Estimating the parameters $\boldsymbol{\beta}$ using the least squares criterion minimizes the following criterion:

$$C = \sum_i (\mathbf{x}_i\boldsymbol{\beta} - y_i)^2 \tag{9}$$

We can use this global linear parametric model, where all the training instances have equal weights; for locally weighted regression, by giving nearby instances to the query point higher weights. This can be done using the following weighted training criterion:

$$C = \sum_i [(f(\mathbf{x}_i, \boldsymbol{\beta}) - y_i)^2 K(d(\mathbf{x}_i, \mathbf{q}))] \tag{10}$$

Various distance (d) and weighting (K) functions for local models are described by Atkenson et al. (1997). Different linear and nonlinear locally weighted regression models can be estimated with those functions.

4.3 Implementation

In LWR, as stated before, the computational cost of training is at a minimum, since training includes only storing new data points in the memory. However, the lookup procedure for prediction is more expensive than in other instance-based learning methods, since a new model is constructed for each query. Here, the use of a *k-d tree* data structure to speed up this process is described briefly (Atkenson et al., 1997).

The difficulty in the table lookup procedure is to find the nearest neighbours, if only nearby instances are included in LWR. If there are n instances in the database, for a naïve implementation we need n distance computations. For an efficient implementation, a *k-d tree* can be employed.

A *k-d tree* is a binary data structure that recursively splits a d-dimensional space into smaller subregions, and those subregions are the branches or leaves of the tree data structure. The search for the nearest neighbours starts from the nearby branches in the tree. For a given distance threshold, there is no need to search further branches by implementing this data structure. Figure 2 illustrates a two-dimensional region.

5 Regression by rule induction

Inducing rules from a given training set is a much studied topic in machine learning. Weiss and Indurkhya (1993*b*, 1995) employed rule induction for a regression problem, and reported significant results. In this section, we first review the rule-based classification algorithm (Weiss & Indurkhya, 1993*a*), Swap-1, which learns decision rules in Disjunctive Normal Form (DNF), and later on describe its adaptation for regression.

The main advantage of inducing rules in DNF is its explanatory capability. It is comparable to decision trees, since trees can also be converted into DNF models. The most important difference

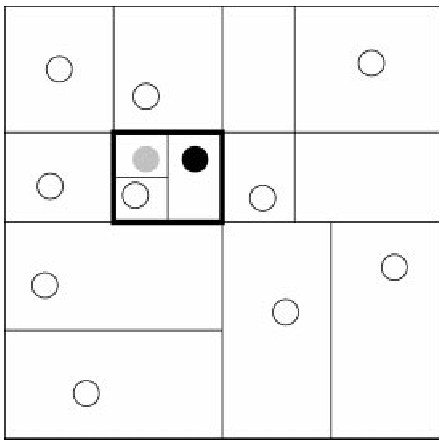


Figure 2 The black dot is the query point, and the shaded dot is the nearest neighbour. Outside the black box does not need to be searched to find the nearest neighbour.

between them is that the rules are not mutually exclusive, as in decision trees. In decision trees, for each instance there is exactly one rule, a path from a root to a leaf, that is satisfied. Because of this restriction, decision tree models may not produce compact models. However, because of this property of rule-based models, a problem emerges such that, for a single instance, two or more classes may be satisfied. The solution found for this problem is to assign priorities or an ordering to the rules according to their extraction order, according to this ordering. According to this ordering, the first rule that satisfies the query instance, determines the class of a query. The Swap-1 rule induction algorithm (Weiss & Indurkhya, 1993*a*) and its sample output are shown in Figures 3 and 4, respectively.

```
[1]   Input: $D$, a set of training cases
[2]   Initialize $R_1 \leftarrow$ empty set, $k \leftarrow 1$, and $C_1 \leftarrow D$
[3]   repeat
[4]       create a rule $B$ with a randomly chosen attribute as its left-hand side
[5]       while ($B$ is not 100-percent predictive) do
[6]           make single best swap for any component of $B$, including
                  deletion of the component, using cases in $C_k$
[7]           If no swap is found, add the single best component to $B$
[8]       endwhile
[9]       $P_k \leftarrow$ rule $B$ that is now 100-percent predictive
[10]      $E_k \leftarrow$ cases in $C$ that satisfy the single-best-rule $P_k$
[11]      $R_{k+1} \leftarrow R_k \cup \{P_k\}$
[12]      $C_{k+1} \leftarrow C_k - \{E_k\}$
[13]      $k \leftarrow k + 1$
[14]  until ($C_k$ is empty)
[15]  find rule $r$ in $R_k$ that can be deleted without affecting performance on cases
          in training set D
[16]  while ($r$ can be found)
[17]      $R_{k+1} \leftarrow R_k - \{r\}$
[18]      $k \leftarrow k + 1$
[19]  endwhile
[20]  output $R_k$ and halt.
```

Figure 3 Swap-1 algorithm.

$$\begin{array}{lll} CA > 0.5 \text{ and } CP > 3.5 & \leftarrow & Class = 2 \\ THAL > 6.5 & \leftarrow & Class = 2 \\ [True] & \leftarrow & Class = 1 \end{array}$$

Figure 4 A solution induced from heart-disease data.

Table 1 Example of swapping rule components

Step	Predictive value (%)	Rule
1	31	p3
2	36	p6
3	48	p6 & p1
4	49	p4 & p1
5	69	p4 & p1 & p2
6	80	p4 & p1 & p2 & p5
7	100	p3 & p1 & p2 & p5

While constructing a rule, the Swap-1 algorithm searches all the conjunctive components it has already formed, and swaps them with all possible components it will build. This search also includes the deletion of some components from the rule. If no improvement is established from these swaps and deletions, then the best component is added to the rule. To find the best component to be added, the predictive value of a component, as a percentage of correct decisions, is evaluated. If the predictive values of them are equal, maximum instance coverage isused as the second criterion. These swappings and additions end when the rule reaches 100% prediction accuracy.

Table 1 illustrates a sample rule induction. After forming a new rule for the model, all instances that the rule covers are removed from the instance set, and the remaining instances are considered for the following steps. When a class is covered, the remaining classes are considered, in turn. This process iterates until the instance set becomes empty, that is, all instances are covered.

After formation of the rule set, if the removal of any rule does not change the performance of the training set, such rules are removed from the model. Furthermore, to reach an optimum rule set, an optimization procedure is used (Weiss & Indurkhya, 1993*a*).

The rule induction algorithms for classification, such as Swap-1, can also be applied to regression problems. Since these algorithms are designed for the prediction of nominal attributes, using a preprocessing procedure, the numeric attribute in regression to be predicted is transformed into a nominal one.

For this transformation, the P-class algorithm, shown in Figure 5, is used by Weiss and Indurkhya (1995). This transformation is in fact a one-dimensional clustering of training instances on response variable y, in order to form classes. The purpose is to make y values within one class similar, and across classes dissimilar. The assignment of these values to classes is done in such a way that the distance between each y_i and its class mean must be minimum.

The P-Class algorithm does the following. First, it sorts the y values, then assigns an approximately equal number of contiguous sorted y_i to each class. Finally, it moves a y_i to a contiguous class if it reduces the distance of it to the mean of that class.

This procedure is a variation of the *KMEANS* clustering algorithm (Duda & Hart, 1973; Kaufman & Rousseeuw, 1990). Given the number of initial clusters, on randomly decomposed clusters, the *KMEANS* algorithm swaps the instances between the clusters if it increases a clustering measure or criterion that employs inter- and intra-cluster distances. Given the number

```
[1]   Input:$\{y\}$ a set of output values
[2]   Initialize $n$ = number of cases, $k$ = number of classes

[3]   repeat for each $Class_i$
[4]       $Class_i$ = next $n/k$ cases from list of sorted $y$ values
[5]   end

[6]   repeat for each $Class_i$ (until no change for any class)
[7]       repeat for each case $j$ in $Class_i$
[8]           1. Move $Case_{ij}$ to $Class_{i-1}$, compute $Err_{new}$
[9]              If $Err_{new} > Err_{old}$ return $Case_{ij} to C_i$
[10]          2. Move $Case_{ij}$ to $Class_{i+1}$ , compute $Err_{new}$
[11]             If $Err_{new} > Err_{old}$ return $Case_{ij}$ to $C_i$
[12]      next $Case_j$ in $Class_i$
[13]  Next $Class_i$

[14]  repeat for each $Class_i$ (until no change for any class)
[15]      If Mean($Class_i$) = Mean($Class_j$) then
[16]      Combine $\mathrm{Class_i}$ and $\mathrm{Class_j}$
[17]  end
```

Figure 5 Composing Pseudo-Classes (P-Class).

of classes, P-Class is a quick and precise procedure. However, no idea is stated in the literature about an efficient way to determine the number of classes.

After the formation of classes (pseudo-classes) and the application of a rule induction algorithm such as Swap-1 to these classes, in order to produce an optimum set of regression rules, a pruning and optimization procedure can be applied to these rules, as described by Weiss and Indurkhya (1993*a*, 1995). An overview of the procedure for the induction of regression rules is shown in Figure 6.

The naïve way to predict the response for a query instance is to assign the average of responses. The average may be a median or mean of that class. However, different approaches also can be considered by applying a parametric or non-parametric model for that specific class. For example, the nearest-neighbour approach is used for this purpose, and significant improvements of this combination against the naïve approach are reported by Weiss and Indurkhya (1995).

1. Generate a set of Pseudo-classes using the P-Class algorithm.
2. Generate a covering rule-set for the transformed classification problem using a rule induction method such as Swap-1.
3. Initialize the current rule set to be the covering rule set and save it.
4. If the current rule set can be pruned, iteratively do the following:
   (a) Prune the current rule set.
   (b) Optimize the pruned rule set and save it.
   (c) Make this pruned rule set the new current rule set.
5. Use test instances or cross-validation to pick the best of the saved rule sets.

Figure 6 Overview of method for learning regression rules.

6 Projection pursuit regression

One problem with most local averaging techniques, such as the nearest-neighbour technique, is the *curse of dimensionality*. If a given amount of data is distributed in space, then the distance between adjacent data points increases with the increasing number of dimensions (Hall, 1989). In Friedman and Stuetzle (1981), a numeric example is given for this problem. Projection Pursuit Regression (PPR) forms the estimation model by reflecting the training set onto lower dimensional projections as a solution for high dimensional data sets.

Another important characteristic of PPR is its *successive refinement* property. At each step of model construction, the best approximation of the data is selected and added to the model, while removing the well described portion of the instance space. The search on the data set continues for the remaining part, and this process iterates by increasing the complexity of the model at each step. The successive refinement concept is applied to regression in a different way here, by subtracting the *smooth* from residuals. A *smooth* is a function formed by averaging responses (y). An example of *smooth* is shown in section 6.2.

The model approximated by the PPR algorithm is the sum of the smooth functions S of the linear projections, determined in each iteration:

$$\varphi(x) = \sum_{m=1}^{M} S_{\boldsymbol{\beta}_\mathbf{m}}(\boldsymbol{\beta}_\mathbf{m}.\mathbf{X}) \tag{11}$$

where $\boldsymbol{\beta}_\mathbf{m}$ is the parameter vector (projection), $\mathbf{X}$ is the training set against predictor variables, $S_{\boldsymbol{\beta}_\mathbf{m}}$ is the smooth function, and M is the number of terms or "smoothes" in the model.

6.1 Projection pursuit regression algorithm

At each iteration of the PPR algorithm, a new term, m in equation (11), is added to the regression surface φ. The critical part of the algorithm is the search for the coefficient vector $\boldsymbol{\beta}$ or projection of the next term. After finding a coefficient vector at each iteration, the smooth of the estimated response values resulting from the inner product ($\boldsymbol{\beta}_\mathbf{m}.\mathbf{X}$) is added to the model as a new term, where the term is a function of all features. The linear sum of these functions (11) forms the model, which is employed for the prediction task.

The search for the coefficient vector for each term is done according to a fitting criterion (figure of merit), such that the average sum of the squared differences between residuals and the smooth is the minimum. For this purpose, $I(\boldsymbol{\beta})$, the fraction of unexplained variance that is explained by smooth S_β, is used as an optimality criterion, or figure of merit. $I(\boldsymbol{\beta})$ is computed as

$$I(\boldsymbol{\beta}) = 1 - \sum_{i=1}^{n}(r_i - S_{\boldsymbol{\beta}}(\boldsymbol{\beta}.\mathbf{x}_i))^2 \Big/ \sum_{i=1}^{n} r_i^2 \tag{12}$$

where r_i is a residual which takes the value of y_i in the first step of the algorithm. The coefficient vector $\boldsymbol{\beta}$ that maximizes $I(\boldsymbol{\beta})$ is the optimal solution.

In the first line of the algorithm, current residuals and the term counter are initialized. In the second step, the coefficient vector that results in the best smooth close to the residuals according to fitting criterion I is found. A smooth is found for each $\boldsymbol{\beta}$ vector, in ascending order of the linear combination ($\boldsymbol{\beta}.\mathbf{X}$). If the criterion value found is below a given threshold, the iteration of the algorithm is continued by the new residual vector, which is found by subtracting the smooth from the current residuals at Step 4. With this subtraction operation, the algorithm gains successive refinement characteristics.

For searching of the coefficient vector that maximizes the fitting criterion, a modification of the Rosenbrock method (Rosenbrock, 1960) is chosen by Friedman and Stuetzle (1981), and as a smoothing procedure, a method is described in the next subsection.

Some models approximate the regression as a sum of the functions of individual predictors (standard additive models), and because of that, they cannot deal with interactions between

```
[1]    $r_i \leftarrow y_i$, $M \leftarrow 0$, $i = 1, \ldots, n$
[2]    Search for the coefficient vector $\beta_M$, that maximize fitting criterion $I(\boldsymbol{\beta})$
       by using Equation (12)
[3]    If $I(\boldsymbol{\beta})$ is greater than the given threshold
[4]        $r_i \leftarrow r_i - S_{\beta_{M+1}}(\boldsymbol{\beta}_{M+1}.\mathbf{x}_i)$, $i = 1, \ldots, n$
[5]        $M \leftarrow M + 1$
[6]        go to *Step* 2
[7]    Otherwise stop, by excluding last term $M$.
```

Figure 7 Projection pursuit regression algorithm.

predictors. In such models, the projections are done onto individual predictors rather than onto a projection vector, which is the linear sum of the predictors, as in PPR. These projection vectors, instead of individual predictors, allow PPR to deal with interactions, which is the third main property of PPR.

6.2 Smoothing algorithm

The traditional smoothing procedures assume that the observed variation, response y_i, is generated by a function which has a normally distributed error component. The smooth constitutes an estimation for that function. As an example, in simple linear regression, this function is a linear combination of predictors. As stated above, PPR tries to explain this variation with not just one smooth, but with a sum of smoothes over linear combinations of predictors.

Generally, the smooth functions employed here are not expressions, rather, they are a local averaging of the responses or residuals. Taking the averages of responses in neighbourhood regions forms this smooth function. The boundaries of the neighbourhood region where the averages are taken are called *bandwidth*. For example, in the *k-nearest neighbour* algorithm, *k* is used for the constant bandwidth. In Friedman and Stuetzle (1981), a variable bandwidth algorithm is employed, where larger bandwidths are used in regions of high local variability of response. To clarify the concept of smoothing, we describe the constant bandwidth smoothing algorithm of Tukey (1977), called "running medians".

Running medians is a simple procedure that averages the response by taking the median of the neighbour region. *Running medians of three* algorithms described by Tukey (1977) are shown with a simple example in Figure 8. The smooth of each response is found by the median of three values in the sequence. One of them is the response itself, and other two are neighbours.

Friedman and Stuetzle (1981) employ *running medians of three* in their variable bandwidth smoothing algorithm, which is shown in Figure 9.

In Step 1, a smooth for the response is formed. In Step 2, for each smoothed response value, we find the variance of the neighbours in the interval determined by a given constant bandwidth. In Step 3, these variances are smoothed by a given constant bandwidth. Finally, by employing these smoothed variance values as a bandwidth for each smoothed response determined in Step 1, we obtain a variable bandwidth smooth.

Given	:	4	7	9	3	4	11	12	1304	10	15	12	13	17
Smooth	:	?	7	7	4	4	11	12	12	15	12	13	13	?

Figure 8 Running medians of three.

```
[1]    Running medians of three;
[2]    Estimating the response variability at each point by the average squared residual
       of a locally linear fit with constant bandwidth;
[3]    Smoothing this variance estimates by a fixed bandwidth moving average;
[4]    Smoothing the sequence obtained by pass (1) by locally linear fits with bandwidths
       determined by the smoothed local variance estimates obtained in pass (3).
```

Figure 9 Variable bandwidth smoothing algorithm.

7 Regression by tree induction

Tree induction algorithms construct the model by partitioning the data set. The task of constructing a tree is accomplished by employing a search to select an attribute to be used for partitioning the data at each node of the tree. The explanation capability of regression trees and their use to determine key features from a large feature set are major advantages for these applications. In terms of performance and accuracy, regression tree applications are comparable to other models. Regression trees are also noted to be strong when there are higher order dependencies among the predictors.

The characteristic common to all regression tree methods is that they partition the training set into disjoint regions recursively, where the final partition is determined by the leaf nodes of the regression tree. To avoid overfitting and form simpler models, pruning strategies are employed in all regression tree methods.

In the following subsections, three different regression tree methods are described: CART, RETIS and M5. They share the common properties described above, but show significant differences in some of the measures and traits they demonstrate.

7.1 CART

Using trees as regression models was first applied in the CART (Classification and Regression Trees) program, developed by the statistical research community (Breiman et al., 1984). This program induces both regression and classification trees.

In the first step, we start with the whole training set represented by the root node to construct the tree. A search is done on the features to construct the remaining part of the tree recursively. We find the best feature and feature value at which to split the training set represented by the root node. This splitting forms two leaf nodes that represent two disjoint regions in the training set. In the second step one of these regions is selected for further splitting. This splitting is again done according to a selected feature value. This splitting process continue at a selected leaf node recursively to construct the regression tree.

In the CART system, a constant function (a constant response value) is used for estimation of a query or test instance that falls into the regions represented by leaf nodes. Generally, this function is the average of the response values of instances. Each disjoint region has its own estimated value that is assigned to any query instance located in this region.

To construct optimum disjoint regions, an error criterion is employed. The optimum value of this criterion produces a decomposition at any step of the tree induction process described above, so that the correct region, feature, feature value (splitting surface) and estimates for each region are selected. To determine the predicted target values in these regions, averaging methods such as mean and median are used. As a fitting criterion, the variances of the regions are used (14):

$$Error(Variance) = \sum_{i=1}^{n} (y_i - \bar{y})^2 \qquad (13)$$

where n is the number of instances in the region,

$$Splitting\ Error = \frac{1}{n}\left\{ \sum_{\mathbf{x}_i \in \mathbf{X}_{left}} (y_i - \bar{y}_{left})^2 + \sum_{\mathbf{x}_j \in \mathbf{X}_{right}} (y_j - \bar{y}_{right})^2 \right\} \tag{14}$$

After computing the splitting error for all possible splits of a particular predictor, the splitting that maximizes the following criterion is selected:

$$C = Variance - Splitting\ Error \tag{15}$$

The node and predictor that reach the maximum criterion C are selected for splitting. An example regression tree is shown in Figure 10 . The construction process is illustrated in Figure 11.

Formally, the resulting model can be defined in the following form (Breiman et al., 1984; Friedman, 1991):

$$If\quad \mathbf{x} \in R_m, \quad \text{then} \quad f(\mathbf{x}) = g_m(\mathbf{x}\{a_j\}_1^P) \tag{16}$$

where $\{R_m\}_1^P$ are disjoint subregions representing p partitions of the training set. The functions g are generally in simple parametric form. The most common parametric form is a constant function (17), which is illustrated with the example given in Figure 10.

$$g_m(\mathbf{x}|a_m) = a_m \tag{17}$$

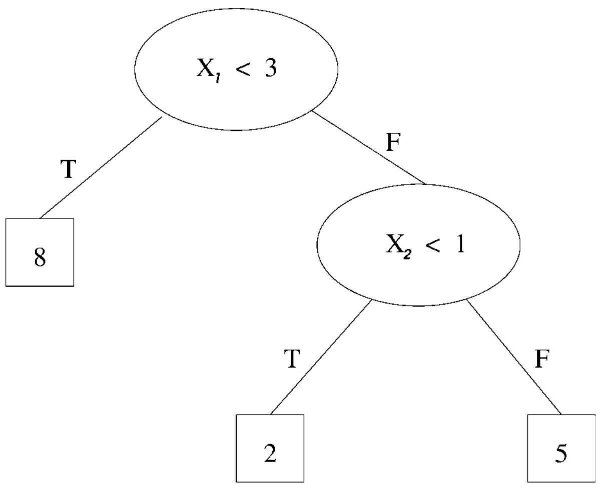


Figure 10 Example of regression tree.

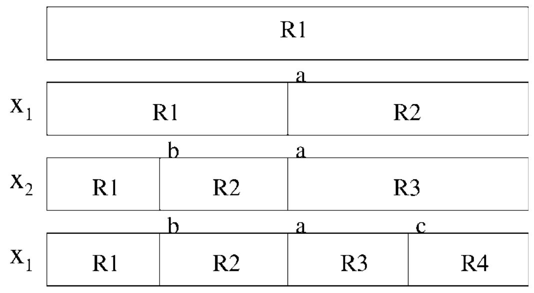


Figure 11 An example of the tree construction process. Four regions are determined by predictors x_1 and x_2.

The constant values of leaves or partitions are generally determined by averaging. More formally, the model can be denoted by using basis functions:

$$\bar{f}(\mathbf{x}) = \sum_{m=1}^{M} a_m B_m(\mathbf{x}) \tag{18}$$

The basis functions $B_m(x)$ take the form

$$B_m(\mathbf{x}) = I(\mathbf{x} \in R_m) \tag{19}$$

where I is an indicator function having the value one if its argument is true, and zero otherwise. Let $H[\eta]$ be a step function, indicating a positive argument:

$$H[\eta] = \begin{cases} 1 & \text{if } \eta > 0 \\ 0 & \text{otherwise} \end{cases} \tag{20}$$

and let LOF(g) be a procedure that computes the lack of fit of an estimation function g to the data. The recursive partitioning algorithm is given in Figure 12.

The first line of the algorithm assigns the whole training set as the initial region. The first loop iterates the splitting until reaching a maximum number of regions. The next three loops select the optimum basis function B_{m^*} (intuitively, the optimum region), predictor x_{v^*} and split point t^*. At lines 12 and 13, the selected region for splitting, B_{m^*}, is replaced with its two partitions. This is done by adding a factor to its product; with $H[-(x_{v^*} - t^*)]$ for the negative portion of the region at line 12 by creating a new basis function; and with $H[+(x_{v^*} - t^*)]$ for the positive portion of the region at line 13, by modifying or removing the previous basis function. Finally, the basis functions formed by the algorithm will take the following form:

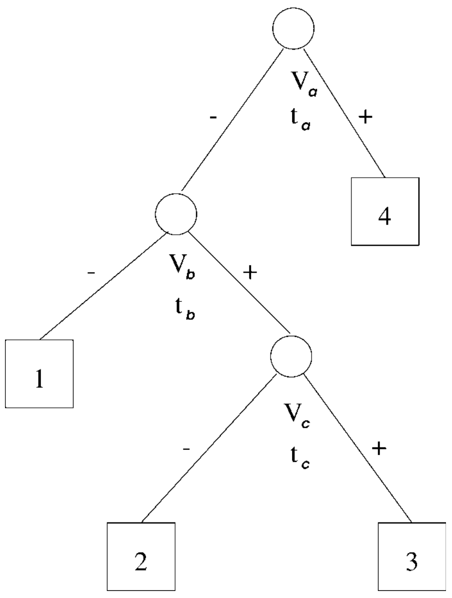
$$B_m(\mathbf{x}) = \prod_{k=1}^{K_m} H[s_{km}.(x_{v(k,m)} - t_{km})] \tag{29}$$

where the quantity K_m is the number of splits that gave rise to B_m, and the arguments of the step functions contain the parameters associated with each of these splits. The quantity s_{km} takes $(+/-)1$ values, indicating the right/left portions, $v(k,m)$ labels the predictor variables, and t_{km} represents values on the corresponding variables. A possible output of the algorithm is shown in Figure 13.

The partition may lead to very small regions with a large tree. This situation may cause overfitting with unreliable estimates. Stopping the process early may also not produce good results. The solution to this problem is to employ a pruning strategy.

```
[1]   $B_1(\mathbf{x}) \leftarrow 1$
[2]   For $M = 2$ to $M_{max}$ do : $lof^* \leftarrow \infty$
[3]     For $m = 1$ to $M - 1$ do :
[4]       For $v = 1$ to $n$ do :
[5]         For $t \in \{x_{vj} | B_m(\mathbf{x}_j) > 0\}$
[6]           $g \leftarrow \sum_{i \neq m} a_i B_i(\mathbf{x}) + a_m B_m(\mathbf{x}) H[+(x_v - t)] + a_M B_m(\mathbf{x}) H[-(x_v - t)]$
[7]           $lof \leftarrow min_{a_1 \ldots a_M} LOF(g)$
[8]           if $lof < lof^*$, then $lof^* \leftarrow lof; m^* \leftarrow m; v^* \leftarrow v; t^* \leftarrow t$ end if
[9]         end for
[10]      end for
[11]    end for
[12]    $B_M(\mathbf{x}) \leftarrow B_{m^*}(\mathbf{x}) H[-(x_{v^*} - t^*)]$
[13]    $B_{m^*}(\mathbf{x}) \leftarrow B_{m^*}(\mathbf{x}) H[+(x_{v^*} - t^*)]$
[14]  end for
```

Figure 12 Recursive partitioning algorithm.



$$B_1 = H[-(x_{va} - t_a)]H[-(x_{vb} - t_b)]$$
$$B_2 = H[-(x_{va} - t_a)]H[+(x_{vb} - t_b)]H[-(x_{vc} - t_c)]$$
$$B_3 = H[-(x_{va} - t_a)]H[+(x_{vb} - t_b)]H[+(x_{vc} - t_c)]$$
$$B_4 = H[+(x_{va} - t_a)]$$

Figure 13 A binary tree representing a recursive partitioning regression model with associated basis functions.

Pruning the regression tree by removing leaves will leave holes, which is an important problem, since we will not be able to give an answer to queries that fall into these regions or holes. That is why the removal of regions is done pairwise, with siblings, by merging them into a single (parent) region. This pruning strategy is described by Breiman et al. (1984).

Recursive partitioning regression is an adaptive method (one that dynamically adjusts its strategy to take into account the behaviour of a particular problem to be solved (Friedman, 1991)). For example, recursive partitioning has the ability to exploit low local dimensionality of functions. In local regions, the dependence of the response may be strong on a few of the predictors, and these few variables may be different in different regions. Another property of recursive partitioning regression is that they allow interpretations, especially when a constant estimation is done on the leaves.

On the other hand, it has some drawbacks and limitations, the most important one being that the estimation is discontinuous. The model cannot approximate even simple continuous functions such as linear functions, which limits the accuracy of the model. As a consequence of this limitation, one cannot extract from the representation of the model the structure of the function (e.g. linear or additive), or whether it involves a complex interaction among the variables.

7.2 RETIS

In the basic CART algorithm described above, the estimated response value, $\bar{y}$ on the leaves of the regression tree was a constant function (17). On the other hand, RETIS (Regression Tree Induction System) (Karalic, 1992*a*, *b*), a different system used to construct regression trees developed by the machine learning community, is an extension of CART that employs a function on the leaves. This is a linear function of continuous predictors. The use of linear regression at the leaves of a regression tree is called as *local linear regression* (Karalic, 1992*a*). RETIS can also be categorized as a LWR system (section 4).

RETIS is not just a modification of CART at the leaf nodes. The employment of linear regression

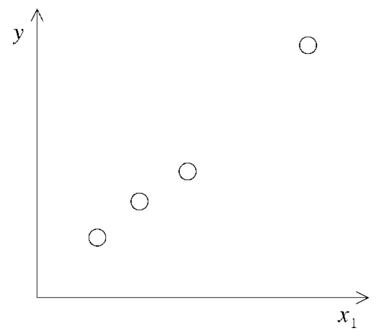


Figure 14 An example region, with large variance, which is inappropriate for splitting.

enforces modifications in the construction of the regression tree. In the process of tree construction, the CART system forms subtrees to minimize the expected variance (21). However, when applying local linear regression to the regression tree, the variance is not an appropriate measure as an optimality criterion. If the relationship between the predictors and response is linear, this region may not be appropriate for splitting even if the variance is very large. This situation is illustrated with an example in Karalic (1992*a*). Suppose we have a region with four instances described with only one predictor, as shown in Figure 14. Although for cases where the variance is large, the error committed when using linear regression is almost zero. Such regions are not appropriate for further splitting. That is why an alternative splitting criterion is employed in RETIS, given in equation (23).

$$I(\mathbf{X}) = \sum_{i=1}^{n} (y_i - g(\mathbf{x}_i))^2 \tag{22}$$

where n is the number of instances, g is the linear function that best fits the instances of the region, and I is called the *impurity measure*. Consequently, the figure of merit (the splitting criterion) is defined as in equation (23):

$$C = \frac{1}{n}[n_{left}I_{left} + n_{right}I_{right}] \tag{23}$$

The use of equation (22) instead of equation (14) in computing the figure of merit is the main difference between CART and RETIS. When estimating a response value for a query, the value that results from the linear function on the leaf node on which the query falls is used.

After construction of a regression tree, a pruning strategy is employed, as in most other tree induction models. See Niblett and Bratko (1986) for an in-depth explanation of pruning. The strategy used in RETIS computes two different error measures: *static error* and *backed-up error*. The static error is computed at a node, supposing it is a leaf, and backed-up error is computed at the same node for the case in which the subtree is not pruned. If the static error is less than or equal to the backed-up error, then the subtree is pruned at that node, and the tree node is converted into a leaf node.

7.3 M5

M5 is another system (Quinlan, 1992) that builds tree-based models for the regression task, similar to CART and RETIS. The advantage of M5 over CART is that the trees are generally much smaller than regression trees. On the other hand, the tree construction in M5 is similar to CART. Standard deviation is employed as the error criterion in M5, instead of variance as used in CART. The reduction on the error (33) on subregions after splitting a region is the measure used to decide on splitting.

$$error = \sigma(\mathbf{X}) - \sum_{i} \frac{|\mathbf{X}_i|}{|\mathbf{X}|} \sigma(\mathbf{X}_i) \tag{24}$$

where σ is standard deviation and i is the number of subregions of a region whose instances are denoted by **X**. After examining all possible splits, M5 chooses the one that maximizes the expected error reduction (24).

M5 is also similar to RETIS in that it employs a linear regression model on the nodes to estimate responses by using standard linear regression techniques (Press et al., 1988). These linear models are constructed on all the nodes, starting from the root down to the leaves. However, instead of using all the attributes or predictors, a model at a node is restricted to the attributes referenced by linear models in the subtree of that node.

After constructing the tree and forming linear models at the nodes as described above, each model is simplified by eliminating parameters to maximize its accuracy. The elimination of parameters generally causes an increase in the average residual. To obtain linear models with fewer parameters, the value is multiplied by $(n + p)(n - p)$, where n is the number of instances and p is the number of parameters in the model. The effect is to increase the estimated error of models with many parameters and with a small number of instances or training cases. M5 uses a greedy search to remove variables that contribute little to the model. In some cases, M5 removes all of the variables, leaving only a constant (Quinlan, 1992).

The pruning process is the same as RETIS. To prune the constructed tree, each non-leaf node is examined, starting near the bottom. If the estimated error at a node is smaller than its subtree, then that node is pruned.

A smoothing process is employed in M5 for estimation of the response variable. The smoothing process described by Quinlan (1992) is as follows:

1. The predicted value at the leaf is the value computed by the model at that leaf.
2. If the instance follows branch S_i of subtree S, let n_i be the number of training cases at S_i, $PV(S_i)$ the predicted value at S_i, and $M(S)$ the value given by the model at S. The predicted value at S is given by recursive equation (25):

$$PV(S) = \frac{n_i PV(S_i) + kM(S)}{n_i + k} \tag{25}$$

where k is the smoothing constant.

The accuracy of the model is enhanced by the smoothing process. Improvements in accuracy and model simplification are obtained by M5 over CART; some applications with different training sets are experimented on and reported by Quinlan (1992).

8 Multivariate adaptive regression splines

As stated in the previous section, a fundamental drawback of recursive partitioning regression (CART) is the lack of continuity, which affects the accuracy. Another problem with that method is its inability to provide good approximations to some functions, even to the most simple linear ones. Multivariate Adaptive Regression Splines (MARS) addresses these two problems of recursive partitioning regression to increase accuracy (Friedman, 1991).

8.1 Piecewise parametric fitting paradigm and splines

There are different paradigms for global parametric modelling to generalize low dimensional data. One of them is piecewise parametric fitting. The basic idea is to approximate a function by several simple parametric functions (usually low order polynomials), each defined over different subregions of the training set. The constraint for the formation of polynomial fitting is that it must be continuous at every point.

The most popular piecewise polynomial fitting procedures are based on splines, where the parametric functions are polynomials of degree q. The procedure is implemented by constructing a set of globally defined basis functions. These functions span the space of the qth order spline approximations, and fit the coefficients of the basis function to the data using the least squares technique. The spline basis functions are denoted by

$$\{(x - t_k)_+^q\}_1^K \tag{26}$$

where $\{t_k\}_1^K$ is the set of split (knot) locations. The subscript $+$ indicates a value of zero for negative values of the argument. This is known as a *truncated power basis* in the mathematical literature. A general review of splines is given by De Boor (1978).

8.2 MARS algorithm

The MARS algorithm is a modified recursive partitioning algorithm which addresses the problems stated above. The reason that recursive partitioning algorithms are discontinuous, because of the use of the step function. If the step function were everywhere replaced by a continuous function, where it appears in that algorithm (lines 6, 12 and 13), it could produce a continuous model. The step function employed in that algorithm can be considered as a special case of a spline basis function, where $q = 0$.

The one-sided truncated power basis functions for representing qth order splines are

$$b_q(x - t) = (x - t)_+^q \tag{27}$$

where t is the knot location, q is the order of the spline and the subscript indicates the positive part of the argument. For $q > 0$, the spline approximation is continuous. A two-sided truncated power basis is of the form

$$b_q^{\pm}(x - t) = [\pm(x - t)]_+^q \tag{28}$$

The step functions that appear in recursive partitioning algorithms are seen to be two-sided truncated power basis functions for $q = 0$ splines. The solution for discontinuity is solved by employing spline functions, of the order of $q > 0$, instead of step functions in the algorithm.

The second modification is related to the second problem, the inability of the algorithm to provide good approximations to certain functions. After the first modification, the algorithm tends to involve functions with more than a few variables (higher order interactions). At each split, one such function is removed, and two new functions are produced with one more variable. This causes a one level increase in the interaction order. With such complex functions, having high level orders, it becomes difficult to approximate simple functions like linear ones.

The solution for this problem is not to delete the lower order parent after splitting. With this modification, all basis functions now become eligible for further splitting. The new model involves either high or low order interactions, or both.

A third problem emerges after the employment of splines in the algorithm. Since the algorithm allows multiple splits on the same predictor, along a single path of the binary tree, final basis functions may include several factors, involving the same variable in their product. For $q > 0$, higher orders than q may be produced on a single predictor.

After the second modification, not deleting the parents after splits, a restriction on the basis function can be applied to involve distinct predictors. Since we do not remove the parent after splitting, many such splits can be done on the same parent. By employing another split to that parent instead of splitting a child, MARS does not increase the depth or add a new factor to the product.

One remaining problem, which is not solved with MARS, is the value of q. The general idea is to use $q = 1$. A discussion of this problem is given by Friedman (1991).

In summary, the following modifications are done to the recursive partitioning algorithm: (a) replacing the step function $H[\pm(x - t)]$ by a truncated power basis function $[\pm(x - t)]_+^q$; (b) not removing the parent basis function B_{m^*} after its split, thereby making it and both its daughters eligible for further splitting; (c) restricting the product associated with each basis function to factors involving distinct predictor variables.

```
[1]   $B_1(\mathbf{x}) \leftarrow 1; M = 2$
[2]   Loop until $M > M_{max}$ : $lof^* \leftarrow \infty$
[3]     For $m = 1$ to $M - 1$ do :
[4]       For $v \notin \{v(k,m) | 1 \leq k \leq K_m\}$
[5]         For $t \in \{x_{vj} | B_m(x_j) > 0\}$
[6]           $g \leftarrow \sum_{i=1}^{m-1} a_i B_i(\mathbf{x}) + a_m B_m(\mathbf{x}) H[+(x_v - t)]_+ + a_M B_m(\mathbf{x}) H[-(x_v - t)]_+$
[7]           $lof \leftarrow min_{a_1 \ldots a_{M-1}} LOF(g)$
[8]           if $lof < lof^*$ , then $lof^* \leftarrow lof; m^* \leftarrow m; v^* \leftarrow v; t^* \leftarrow t$ end if
[9]         end for
[10]      end for
[11]    end for
[12]    $B_M(\mathbf{x}) \leftarrow B_{m^*}(\mathbf{x}) H[+(x_{v^*} - t^*)]_-$
[13]    $B_{M+1}(\mathbf{x}) \leftarrow B_{m^*}(\mathbf{x}) H[-(x_{v^*} - t^*)]_-$
[14]    $M \leftarrow M + 2$
[15]  end loop
[16]  end algorithm
```

Figure 15 MARS algorithm.

After using two-sided truncated power basis functions instead of a step function, the MARS algorithm (shown in Figure 15) now produces multivariate spline basis functions of the following form:

$$B_m^{(q)}(\mathbf{x}) = \prod_{k=1}^{K_m} H[s_{km} \cdot (x_{v(k,m)} - t_{km})]_+^q \tag{29}$$

For pruning of the resulting model after the MARS algorithm, it is now no longer necessary to employ the two-at-a-time deletion strategy used in the previous algorithm. Because the parents are not deleted, there will be no holes left after any deletion. Any pruning algorithm can thus be employed for the MARS procedure.

In the algorithm above, truncated power basis functions ($q = 1$) are substituted for step functions in lines 6, 12 and 13. The parent basis function is included in the modified model in line 6, and remains in the model through lines 12–14. Basis function products are constrained to contain factors involving distinct variables by the control loop in line 4. Figure 16 illustrates the regions after constructing the model. Note that the split regions are not deleted from the model, as in CART, and another splitting for the same region can be applied with the same or a different predictor.

9 Discussion

We have reviewed six different regression techniques, each having different characteristics when compared to others. Three of them (instance-based regression, locally weighted regression, and rule-based regression) have been developed mainly by the machine learning community, and others (projection pursuit regression, regression tree induction and multivariate adaptive regression splines) mainly by the statistical community. The common property of all these methods is that all of them are non-parametric, and they are the most popular among current regression methods.

In instance-based learning, a lazy approach is employed, where no model is constructed in the training phase. The model is the training set itself. The whole computational complexity of this method is in its prediction, especially the determination of neighbour instances. The prediction is based on the location of the query, and it is computed according to the target values of neighbour instances. The criterion used to detect neighbour instances is the similarity measure based on distance.

Locally weighted regression is another lazy or memory-based approach, where the instances are

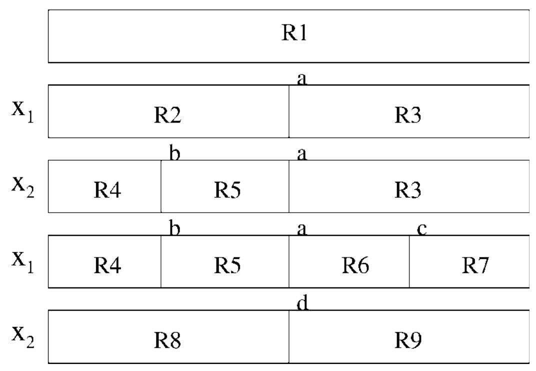


Figure 16 An example for regions of the MARS algorithm.

simply stored in memory during the training phase. The difference between locally weighted regression and instance-based methods is in the prediction phase, where a local parametric model is constructed for each query instance by using the neighbour instances. Since, at each query instance, a new local model is constructed, it is more complex than the previous approach.

The projection pursuit regression method has the ability to reduce dimensionality by projecting instances to lower dimensional (one or two) vectors or surfaces. The idea of projection is also used in exploratory data analysis to determine clusters on projections (Friedman & Tukey, 1974). The same idea is adapted to regression. A successive refinement technique is also applied in the projection pursuit regression, which shows significant improvements for most applications.

All the remaining methods reviewed in this paper estimate models by partitioning the training set into regions. Rule-based regression techniques accomplish this by partitioning the data using the rule induction techniques of machine learning. On the other hand, in the other partitioning methods (CART, RETIS, M5 and MARS), this is done by splicing the features recursively into two regions, by constructing a binary regression tree. The main difference between these methods and MARS is that MARS is continuous at the borders of the partitioned regions, while others are discrete. CART simply uses the averages of the regions for prediction; RETIS and M5 make prediction by constructing linear models. On the other hand, since MARS produces a large number of overlapping regions, its computational complexity is larger than other partitioning methods.

The properties of regression methods are summarized in Table 2. Five different properties are used to compare the algorithms. The main characteristic of *memory-based* models is storing the instances and delaying processing to the prediction phase. The model constructed is in fact the training set itself. *Recursive partitioning* algorithms construct the models by partitioning the data into regions. *Interpretability* is one of the main concerns for most knowledge acquisition and knowledge engineering applications, in order to extract information that can be verified by experts. The algorithms covered in this paper that induce models have this property. If the local positions of

Table 2 Properties of regression algorithms (the names of programs developed with those methods are shown in parentheses)

Properties	Instance based regression (KNN)	Locally weighted regression (LOESS)	Proj. pursuit regression (PPR)	Rule based regression (Rule)	Tree-based regression (CART)	Adaptive regression splines (MARS)
Memory-based (lazy)	✓	✓				
Partiitioning				✓	✓	✓
Interpretable			✓	✓	✓	✓
Adaptive		✓	✓	✓	✓	✓
Incremental	✓	✓				

the test or query instances affect the model, prediction and contribution of variables in the regression task, such algorithms are called *adaptive*. Another important property given in the table is *incrementality* of the algorithm. This is the opposite of batch processing. For large training sets, or databases, particularly, processing can be done without loading all of the data set into memory if this property is satisfied. The order of the training instances is ignored when constructing any such model.

Besides these properties, a comparison of the accuracy between these methods is required. In classical statistical literature, the comparisons are done based on the fitting of the model on the training instances, by means of computing variance explained by the constructed model. If we construct a model for exploratory data analysis, this approach is valid since the model is mainly used for interpretation. However, from the perspective of machine learning and knowledge discovery, the predictive performance of the method is very important. For that purpose, the accuracy is measured on the test cases. The general method used here is to employ *cross-validation*. Cross-validation is applied in order to get very accurate results about the algorithms. The data is partitioned into k partitions. At each step, one of these partitions is used as the test data, and the remaining part is used for training. After k iterations, the average number of accuracies found at each step is used as the final accuracy result.

To measure the performance of a regression method, a comparison is done between the predicted and sample values of an instance. A classical measure is the variance of the error computed from these values, the average squared distance between estimated and sample values. Another is the Mean Absolute Distance (MAD), which is employed in least absolute deviation regression:

$$Variance = \frac{1}{n}(y_i - \bar{y}_i)^2 \tag{30}$$

$$MAD = \frac{1}{n}\sum_{i=1}^{n}|y_i - \bar{y}_i| \tag{31}$$

A comparison of the performance of these regression methods, excluding projection pursuit regression and locally weighted regression, is given by Weiss and Indurkhya (1998), on the datasets provided and described by Quinlan (1993). Table 3 summarizes the key characteristics of the datasets. In this comparison, the accuracy is reported by employing 10-fold cross-validation on test instances, with MAD and relative error. The relative error is the estimated MAD (measured by cross validation), normalized by the initial mean absolute distance from the median. The results reported by Weiss and Indurkhya (1998) are shown in Table 4.

According to this comparison, two partitioning methods, rule-based regression and MARS, outperform the other two methods in terms of performance. Another important result from these comparisons shows that constructing local models for the partitions of data, like splines in the partitions of MARS model, may significantly improve performance.

Table 3 Dataset characteristics

Dataset	Instances	Predictors
Price	159	16
Servo	167	19
Cpu	209	6
Mpg	392	13
Peptide	431	128
Housing	506	13
Pole	15 000	48

Table 4 Comparison of regression methods using published results

	5-nn		Rule		CART		MARS	
Dataset	MAD	Error	MAD	Error	MAD	Error	MAD	Error
Price	1643.00	0.40	1335.00	0.32	1660.00	0.40	1559.00	0.38
Servo	0.58	0.63	0.24	0.25	0.20	0.21	0.21	0.23
CPU	29.40	0.38	27.62	0.35	30.50	0.39	27.29	0.35
Mpg	2.14	0.33	2.17	0.33	2.28	0.35	1.94	0.30
Peptide	0.95	0.45	0.86	0.40	0.97	0.46	0.98	0.46
Housing	2.77	0.42	2.51	0.38	2.74	0.42	2.24	0.34
Pole	5.91	0.20	3.76	0.13	4.10	0.14	7.41	0.25

10 Future work

Even though much research has been done on regression, two important aims seem to have directed these studies. One is the fitness of the estimation or accuracy; the other is the interpretability of the constructed model. New research can be conducted in these traditional directions. Additionally, research can also be directed to increase the efficiency, both in computational complexity and storage. Also, constructing systems that deal with outliers, noise, missing values and irrelevant features is very important, since databases today have a very large number of records and attributes. Models that enables the detection and interpretation of interactions between attributes can also be researched in the future.

References

Adeli, H, 1990, *Knowledge Engineering*, Vol. 1, McGraw Hill.

Aha, D, Kibler, D and Albert, M, 1991, "Instance-based learning algorithms" *Machine Learning*, **6** 37–66.

Aha, DW and Salzberg, SL, 1994, "Learning to catch: applying nearest neighbor algorithms to dynamic control tasks" *Selecting Models from Data: Artificial Intelligence and Statistics IV* Springer-Verlag.

Atkenson, GC, Moore, AW and Schaal, S, 1997, "Locally weighted learning" *Artificial Intelligence Review* **11**(1/5) 11–73.

Breiman, L, Friedman, JH, Olshen, RA and Stone, CJ, 1984, *Classification and Regression Trees* Wadsworth.

Dasarathy, BV (ed), 1991, *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques* IEEE Press.

De Boor, C, 1978, *A Practical Guide to Splines* Springer-Verlag.

Duda, R and Hart, PE, 1973, *Pattern Classification and Scene Analysis* Wiley.

Fayyad, UM, Piatetsky-Shapiro, G, Smyth, P and Uthurusamy, R, 1996*a*, *Advances in Knowledge Discovery and Data Mining* AAAI Press.

Fayyad, UM, Piatetsky-Shapiro, G and Smyth, P, 1996*b*, "The KDD process for extracting useful knowledge from volumes of data" *Communications of the ACM*, **39**(11) 27–34.

Friedman, JH, 1991, "Multivariate adaptive regression splines" *Annals of Statistics* **19**(1) 1–141.

Friedman, JH and Stuetzle, W, 1981, "Projection pursuit regression" *J. Amer. Statist. Assoc.* **76** 817–823.

Friedman, JH and Tukey, JW, 1974, "A projection pursuit algorithm for exploratory data analysis" *J. Amer. Statist. Assoc.* **C-23**(9) 881–890.

Hall, P, 1989, "On projection pursuit regression" *Annals of Statistics* **17**(2) 573–588.

Imielinski, T and Mannila, H, 1998, "A database perspective on knowledge discovery" *Communications of the ACM* **39**(11) 58–64.

Karalic, A, 1992*a*, "Employing linear regression in regression tree leaves, *Proceedings of ECAI'92 (European Congress on Artificial Intelligence)* Vienna, Austria, pp. 440–441.

Karalic, A, 1992*b*, "Linear regression in regression tree leaves" *Proceedings of ISSEK'92 (International School for Synthesis of Expert Knowledge) Workshop* Bled, Slovenia.

Kaufman, L and Rousseeuw, PJ, 1990, *Finding Groups in Data – An Introduction to Cluster Analysis* Wiley.

Kibler, D, Aha, DW and Albert, MK, 1989, "Instance-based prediction of real-valued attributes" *Comput. Intell.* **5** 51–57.

McTear, MF and Anderson, TJ, 1990, *Understanding Knowledge Engineering* Ellis Horwood.

Mitchell, TM, 1997, *Machine Learning* McGraw Hill.

Niblett, T and Bratko, I, 1986, "Learning decision rules in noisy domains" *Developments in Expert Systems* Cambridge University Press.

Peterson, M, Lundberg, D and Ek, A, 1990, *Knowledge Engineering System Design in Diffuse Domains* Studentlitteratur, Chartwell-Bratt.

Press, WH, Flannery, BP, Teukolsky, SA and Vetterling, WT, 1988, *Numerical Recipes in C* Cambridge University Press.

Quinlan, JR, 1992, "Learning with continuous classes" *In Proceedings AI 92* Singapore, pp. 343–348.

Quinlan, JR, 1993, "Combining instance-based and model-based learning" *International Conference on Machine Learning* pp. 236–243.

Rawlings, JO, 1988, *Applied Regression Analysis: A Research Tool* Wadsworth.

Rosenbrock, HH, 1960, "An automatic method for finding the greatest or least value of a function" *Computer Journal* **3** 175–184.

Tukey, JW, *Exploratory Data Analysis* Addison-Wesley.

Weiss, S and Indurkhya, N, 1993*a*, "Optimized rule induction" *IEEE Expert* **8**(6) 61–69.

Weiss, S. and Indurkhya, N, 1993*b*, "Rule-based regression" *In Proceedings of the 13th International Joint Conference on Artificial Intelligence* pp. 1072–1078.

Weiss, S. and Indurkhya, N, 1995, "Rule-based machine learning methods for functional prediction" *Journal of Artificial Intelligence Research* **3** 383–403.

Weiss, S. and Indurkhya, N, 1998, *Predictive Data Mining: A Practical Guide* Morgan Kaufmann.