Tree Structured Interpretable Regression

David Lubinsky
Department of Computer Science
University of The Witwatersrand
Johannesburg, South Africa
E-Mail: david@cs.wits.ac.za

November 9, 1994

Abstract

We describe a new method of regression closely related to the regression ideas CART which has the following potential advantages over traditional methods: the method can naturally be applied to very large datasets in which only a small proportion of the predictors are useful, the resulting regression rules are more easily interpreted and applied, and may be more accurate in application, since the rules are derived by means of a cross-validation technique which maximizes their predictive accuracy. The system is evaluated in an empirical study and compared to traditional regression and CART systems.

1 Introduction.

Broadly speaking, the problems of inducing rules from data are divided into two categories. When the response or predicted variable is continuous the problem is called a *regression* problem, while problems where the response is discrete (taking only one of a few possible, usually non-ordered) values are called *classification* problems.

There has been much work in building classifier systems that produce classification rules that can be easily understood (for surveys, see [1, 2] and [3]). The rules produced by these systems take the form of simple logical predicates or *if* ... *then* rules, rather than linear or non-linear combinations of variables.

In regression, recent work has attempted to increase the power of available models by taking advantage of cheaper computer horse-power, to fit more flexible models, to decrease the influence of outliers, or to fit models which are insensitive to the distributions of the variables but these methods have concentrated little on improving interpretability of the resulting models. Interpretability is especially important in two settings:

- when the aim of the analysis is not only to produce rules but also to get insight into the underlying process,
- when the rules are to be applied by people who need to understand all aspects of its operation. For example, when using rules for patient diagnosis, doctors must have rules that they can understand and agree with.

In this paper we introduce an approach to regression that produces sets of rules which can be easily interpreted and are also flexible in the forms of function that they can fit. The

rules are derived by maximizing their predictive ability and hence are robust and give good results in practical use.

We begin with a broad description of the proposed system. Then more details of how the system is implemented are given. The third section contains an experiment in which the system is evaluated by comparing it to traditional regression methods on a real-world dataset. The fourth section discusses a set of desiderata for an interpretable regression system and the extent to which these are met by the system. The appendix discusses some of algorithms used in organizing the computation efficiently.

2 Description of the TSIR (Tree Structured Interpretable Regression) system

2.1 The tree building process

The idea of the TSIR system is to build a model which is a tree, but some nodes in the tree have only one child. At these nodes the data are not split, but residuals are taken from a single variable regression.

At each stage, all possible regression and split steps are considered and the one with highest value is picked. The split or regressor is checked for *significance* and if it is not the execution terminates for the current branch and a leaf node is created.

If a regressor is chosen, then the residuals from the regressor are taken and the computation is continued recursively.

If a split is chosen, the data are split accordingly and the computation continues recursively on each split group.

At a leaf node the median of the response values of the observations that reached the leaf is used as the predictor.

Figure 1 is a simplified example of the output of the system in a heart-disease domain. At A the median of the response variable is given. B shows an example of a rule with a continuous regressor, C shows and example of a split, and D is an example of a discrete regressor. E is a leaf.

2.2 Comparison to CART

The system is a similar in concept to recursive partitioning systems such as CART [4] and uses predictive maximization techniques similar to those used in the PVM system[5]. The main differences between the CART system and the TSIR system are that in CART each case which arrives at a leaf of the tree is given the same predicted value. This throws away a lot of information about the case. In the TSIR system, all the relevant information is used to make the prediction. CART only has split steps, whereas the trees produced by the TSIR system have split steps, which split the data as in regression trees and regression steps, where each regression step adds one variable and its coefficient to an incrementally growing model. This means that each leaf of the TSIR tree corresponds to a multivariate linear regression. The final difference is that split and regression steps are chosen on the basis of their predictive power by cross validation rather than by maximizing a local criterion.

2.3 The four types of steps

The four possible types of steps are:

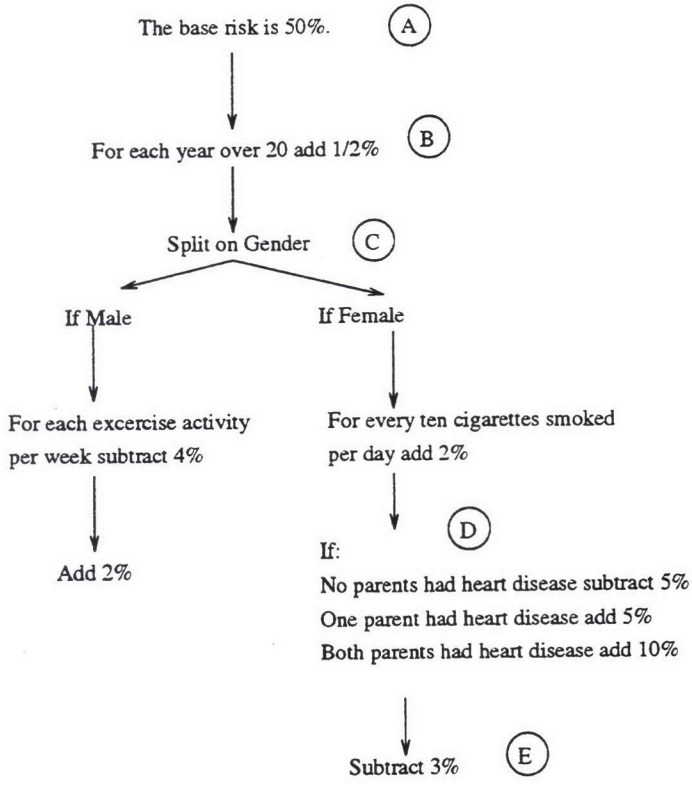

Figure 1: Example of a TSIR tree for probability of heart disease

1. **Numerical Split** The data are split into two sets on the basis of a threshold value for a numeric variable. The form of the split is $x <= a$.

2. **Discrete Split** The data are split into two sets using a discrete variable. The form of the split rules is $x \in A$ where A is a subset of the possible values of x.

3. **Numerical Residual** An L_1 line is found and the residuals of the response are taken from the line. The new values of the response are $y_i' = y_i - (a + bx_i)$. In this case there is no split.

4. **Discrete Residual** The medians of the response are found for each possible value of the predictor and these medians are subtracted from the response: $y_i' = y_i - medians(x_i)$.

3 The split and regression statistics

Four new statistics are introduced to choose between regressions and splits, each corresponding to one type of step. They are comparable, since all measure the reduction in cross-validated predicted variance.

3.1 The predictive correlation for regression steps

The data consists of a n cases, indexed from $1 \ldots n$. Each case consists of a set of measured variables and a continuous response variable. The measured variables may be discrete or continuous.

We will use capital letters, to indicate vectors of values, or all values of a particular variable, and small letters to indicate scalars. To indicate subsets of vectors, we use the notation $X_{<a}$ to indicate all values in X that are less than a, and similarly for other relational operators. If s is a set of indices, then X_s refers to the values in X corresponding to those indices. In either of the previous two cases, if we surround the subscript with ()'s then we mean all values of the vector except those that are referred to by the subscript. For example, $X_{\leq a} \equiv X_{(>a)}$.

Since the system uses an extensive search for good descriptors, it is important to guard against the possibility of spurious fitting of noise. The method used to do this to evaluate all potential descriptors in terms of their predictive ability by cross validation.

We define a cross-validation partition $V = \{v_1 \ldots v_k\}$, where each v_i is a set of indices, as a k partition of a random permutation of the indices $1 \ldots n$, such that $|v_i| = \lceil n/k \rceil$ for $1 \leq i < k$ and v_k contains the remaining indices.

The predictive correlation $\pi(X, Y)$ is a measure of the ability of variable X to predict Y. The variable X may be a continuous or discrete variable and Y is the continuous response variable. For each set in the cross-validation partition a predictor is calculated, based only on the cases not in the set, and the predictor is then used to predict the values, for the cases in the set. As an analog of traditional correlation, predictive correlation is defined as follows.

$$\pi(X,Y) = 1 - \frac{\sum_{v \in V} \sum_{i \in v} |R(X_{(v)}, Y_{(v)}, x_i) - y_i|}{\sum_{i=1}^{n} |y_i - median(Y)|} \quad \text{where}$$

$$R(X,Y,x) \Leftrightarrow \text{The predicted value at } x \text{ based on the predictor of } Y \text{ on } X$$

The value of $\pi(X, Y)$ is bounded above by one, but may be less than zero if R performs worse than the median as a predictor.

The form of the function R depends on whether X is a numeric or discrete variable. If X is numeric then R is the prediction based on the L_1 regression[1] of Y on X. If a and b are the coefficients of this regression then $R(X, Y, x) = a + bx$. If X is discrete then let $X(x')$ be the set of indices of X whose value is x', then we define

$$R(X, Y, x') = median(Y_{X(x')}).$$

3.2 The predictive correlation for split steps

We also define split correlation $\sigma(X, Y)$. The definition of $\sigma(X, Y)$ where the response variable is Y again depends on the type of X.

If X is continuous, then let U_x be the set of values at midpoints between adjacent values of X. We define the *value* of the split on the continuous variable X as

$$V(X,Y) = \min_{z \in U_x} \sum_{v \in V} \left[\sum_{y \in (Y_v)_{\leq z}} |y - median((Y_{(v)})_{\leq z})| + \sum_{y \in (Y_v)_{>z}} |y - median((Y_{(v)})_{>z})| \right]$$

If X is discrete, then let S_X be the set of possible values that X can take on. Now for a subset $s \subset S_X$ let $X(s)$ be the set of indices of values of X in s. Then the split value is

$$V(X,Y) = \min_{s \subset S_X} \sum_{v \in V} \left[\sum_{y \in (Y_v)_{X(s)}} |y - median((Y_{(v)})_{X(s)})| + \sum_{y \in (Y_v)_{(X(s))}} |y - median((Y_{(v)})_{(X(s))})| \right]$$

[1] L_1 regression minimizes the sum of the absolute deviations of the points from the fitted line.

Using the appropriate version of V(X,Y) we define

$$\sigma(X,Y) = 1 - \frac{V(X,Y)}{\sum_{i=1}^{n} |y_i - median(Y)|}$$

At each step, we pick the best of the four possible steps by picking the step that maximizes the predictive correlation. The procedure terminates when no step yields a *significant* correlation. The various forms of the predictive correlation are nor amenable to analytic derivation of their distributions so a large simulation study was run to determine the thresholds of the predictive correlations under various scenarios.

3.3 Description of the experiment

Four experiments were run to evaluate the distribution of the various forms of the predictive correlation. In each experiment the predictive correlation was evaluated under a wide range of parameters where there was in fact no relationship between the predictors and the response. These values were then used to get a null distribution of the predictive correlation and the 95'th percentile of this distribution is used in the system to determine whether a predictive correlation value is significant or not.

To evaluate the predictive correlation for numeric splits and numeric regression steps, random datasets were generated with sizes from 100 to 10000 and at each size 1000 datasets were generated. Also, three different distributions (normal, uniform and exponential) were used at each size. There was no difference in the distributions of the predictive correlation for the three distribution, but the value of the 95'th percentile does decrease as the size of the dataset increases.

For discrete splits and regression steps, the number of categories is introduced as a further parameter. Datasets were generated with two five and ten categories and varying the proportion of cases in the dominant category from 0.1 to 0.9, and again taking sizes between 100 and 10000. The number of datasets at each size was also 1000. The distributions of the 95'th percentile were again remarkably similar and also similar to the distributions for numeric splits and regression.

To simplify the system, it was decided to use a single distribution function which was similar to all the distributions derived in the experiments. The 95'th percentiles of this function are shown on a log-log scale in figure 2.

4 Empirical Evaluation

Figure 3 shows an actual TSIR tree built from the Boston housing dataset [6]. This display format allows for more interpretation and diagnosis than the simpler format used in Figure 1. The response variable is the median cost of homes in areas around Boston. There are thirteen predictors. This dataset does contain one categorical variable but is does not appear in the final tree. The tree contains nine nodes including four leaves. The root node is a linear regression step in which a clear downward trend based on LSTAT (the percentage of lower status of the population). The next node is a split on the number of room in the house. Areas were the mean is fewer than seven rooms go left. The large majority of areas fit into this category and no further steps were found to be significant. Of those that went right another significant split is found at about 7.5 rooms. The areas with the larger houses then have no further steps. Those in the intermediate category have another numeric regression step (on property tax) followed by a final split on pupil-teacher ratio.

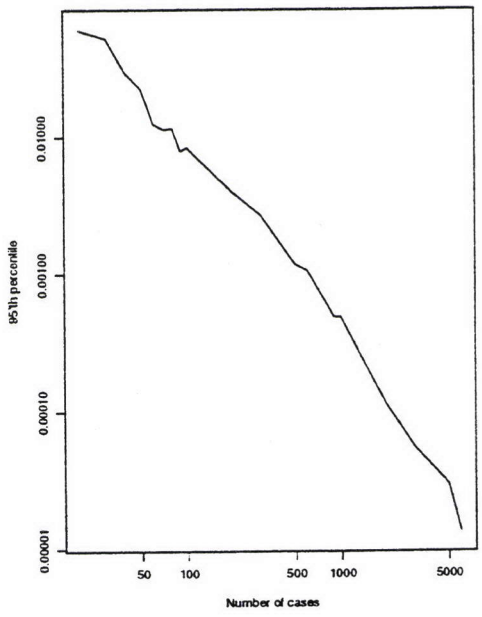

Figure 2: The function used to check the significance of steps.

The TSIR method was also evaluated in a small test where 415 of the observation were used in a training set and 91 were held back as a test set. These data were also used to build a linear model and a CART model. In the linear model ten of the predictors were significant and the total sum of absolute residuals on the hold-back set of 278.643 The CART model contained 43 nodes but achieved a sum of absolute deviation of 277.21. The TSIR model with only nine node achieved a sum of absolute residuals of 276.13, marginally beating the other two, in terms of absolute residual, with a more interpretable model.

5 Desiderata for an interpretable regression system

In this section, we give a set of desirable attributes for an interpretable regression system and discuss to what extent each one is met by TSIR.

An interpretable regression system should:

- maximize performance in terms of predictive ability. We attempt to do this by choosing each step to maximize performance in terms of prediction rather than explanation.
- give all rules on untransformed variables. This aids users of systems to interpret the results.
- give indication of variable importance. The TSIR system can output the best predictive correlation for each variable at each step, giving an indication of the variables importance.
- give rules that can be interpreted piecewise (or locally). The power of tree models, like all hierarchies, is that they allow the mind to concentrate on small parts of the model

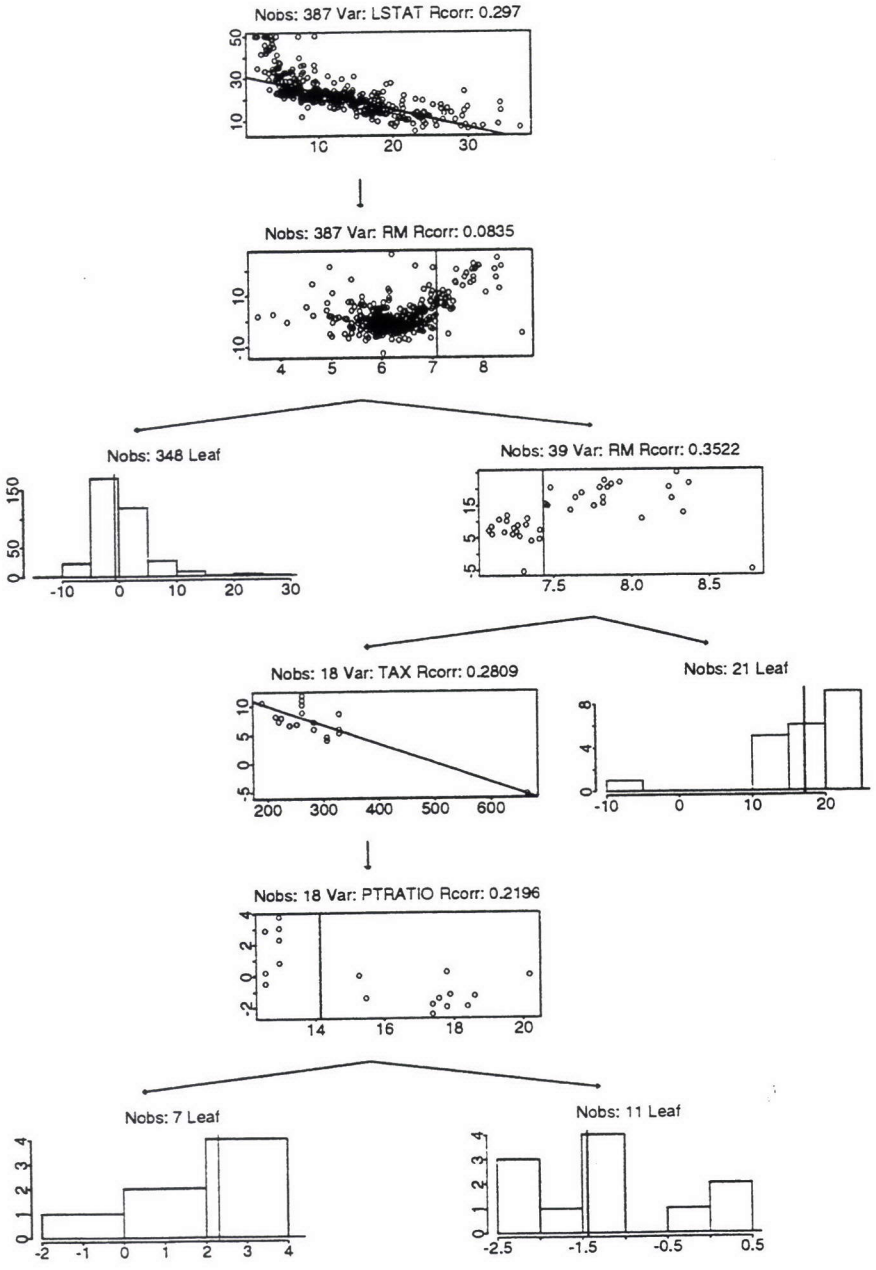

Figure 3: Output for the Boston dataset

without too much though for what happens in other parts of the tree. This allows them to be interpreted more easily.

- cope with large datasets. Since each step takes $O(mn \log n)$ time, where n is the number of cases and m the number of variables, the TSIR system is quite efficient and can be used on large datasets.
- cope with missing values. The fact that each variable is dealt with individually means that only those cases which are missing on a the variable need be omitted.
- cope with redundant variables. Since each variable is dealt with individually redundant variables are no problem except for in the increased computational cost.
- cope with variables that are linearly related. The numeric regression steps add power to the original CART formulation.

6 Further Directions

There is a need for more empirical evaluation. The current method of using a stopping rule rather than building a large tree and then pruning back as is done in CART needs to be re-evaluated.

7 Acknowledgements

The *boston* dataset was copied from the Statlib archive. The reimplimentation of the regression part of CART is by Terry Therneau and is also available from *lib.stat.cmu.edu*.

A Algorithmic speedups

A.1 Updating medians

For each categorical variable X, we need to maintain the set of medians of the response variable for each element of S_X, the set of possible values for X. If the values associated with each element of S_X are stored in a balanced binary tree then the median can be obtained in $O(\log n)$ time and each insert and delete can also be achieved in $O(\log n)$ time. This means that if we consider a k-way cross-validation, the initial trees can be set up in $O(n \log n)$ time and each subsequent partition cost $O((n \log n)/k)$ giving a total cost of $O(n \log n)$, compared with $O(kn \log n)$ for a naive implementation.

A.2 Saving sorting information

When evaluating potential splits on numeric variables, it is necessary that the variable be sorted first. This need only be done once in the beginning, and the sorting information can be saved across splits as follows. Assume the sort order for variable X is O_X, i.e. the i'th element in sorted order of X is $X[O_X[i]]$. Now assume we are splitting on another variable Z, and the set of indeces which *go left* is I_L and the indeces which go right is I_R, then we have two new subsets of the X's $X_L = X[I_L]$ and $X_R = X[I_R]$. To get sort order of X_L, we need a function that for each element x of X_L returns its position in X_L. This function can be computed in linear time. Let this function which returns the index of x in X_L be $I_L(x)$,

```
seq = 0
for i = 1 to n
  if O_X[i] ∈ L
    seq = seq + 1
    O_L[seq] = I_L(x_i)
```

Figure 4: Finding the sort order after a split

then we apply the following algorithm: After this algorithm has completed, O_L will contain the correct sort order for X_L. The algorithm clearly runs in $O(n)$ time so the sort orders can be maintained with minimal cost.

A.3 Calculating the optimal LAD splits

To calculate the optimal LAD split we must find the index such the following is minimized:

$$\min_s \sum_{i=1}^{s} |y_i - median(y_1 \ldots y_s)| + \sum_{i=s+1}^{n} |y_i - median(y_{s+1} \ldots y_n)|$$

We would like to avoid taking the n median explicitly, since the naive method of doing this would cost $O(n^2 \log n)$ or even using the linear median algorithm, $O(n^2)$.

We need to be able to update the running median as each new point is added. The basic idea is to maintain two sets of points, those above the current median and those below, then there is a little bit of book keeping when points migrate from one set to the other. This can be achieved by keeping the two sets in balanced binary trees, then the inserts, deletions of smallest or largest can all be done in $O(\log n)$ time leading to a total time of $O(n \log n)$. The following algorithm implements this idea and makes sure that all book keeping is done correctly. The main idea of the algorithm is to make sure that the sizes of the *above* and *below* sets never differ by more than one. The values of ra and rb maintain the sum of residuals above and below the current median. These are also updated as each point is inserted in constant time, so that the total cost of finding the running medians as well as the sum of absolute residuals is still $O(n \log n)$. The details of the algorithm are given in Figure 5.

This function gives the running medians in the forward direction. The medians in the backward direction can be obtained simply by reversing the data.

The function $current\_med(above, below)$ returns the current median as either the smallest of the above group, the largest of the below group, or the midpoint between them depending on the relative sizes of the two groups.

A.4 Updating LAD line fits

We use LAD line-fitting algorithm presented on page 543 of [7]. This method starts by finding a least squares fit to use as an initial guess then iteratively improves on these estimates to find the LAD line. This process can be improved by using the fit from the previous dataset. Since each consecutive pair of datasets shares 8/9'ths of their data, the initial guess will be a more useful starting point than the least-squares fit.

References

[1] R.O. Duda and P.E. Hart. *Pattern classification and scene analysis.* Wiley, 1973.

[2] Sholom Weiss and Casimir Kulikowski. *Computer Systems that Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems.* Morgan Kaufmann, 1990.

[3] S.C. Choi, editor. *Statistical Methods of Discrimination and Classification.* Pergamon Press, 1986.

[4] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and Regression Trees.* Wadsworth, Belmont, California, 1984.

[5] Sholom M. Weiss, Robert S. Galen, and Prassad V. Tadepalli. Maximizing the predictive value of production rules. *Artificial Intelligence*, 45:47–71, 1990.

[6] D. Harrison and D.L Rubinfeld. The boston house-price data, 1988. From lib.stat.cmu.edu.

[7] William H. Press, Brian P. Falnnery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes The ART of Scientific Computing.* Cambridge, 1988.

```
below.insert(y[0]) assuming y[0] < y[1]
above.insert(y[1])
ra = (y[1]-y[0])/2
rb = ra
for i = 2 to n
  old = current
  switched = 0
  n_above = above.size()
  n_below = below.size()
  v = y[i]
  min_above = minval(above)
  max_below = maxval(below)
  if v ≥ min_above
    above.insert(v)
    if n_above > n_below
      above.del_min()
      below.insert(min_above)
      switched = 1

    current = current_med(above.below)
    ra = ra + (n_above-switched) * (old - current)
    rb = rb + n_below * (current - old)
    ra = ra + v - current
    if switched
      ra = ra - (min_above - old)
      rb = rb + (current - min_above)

  else if v ≤ max_below
    Do the same as above reversing all and inserting in below
  else new value is between.
    if n_above < n_below insert in above
      above.insert(v)
      current = current_med(above.below)
      ra = ra + n_above * (old - current)
      rb = rb + n_below * (current - old)
      ra = ra + v - current
    else
      Do the same as above reversing all and inserting in below
```

Figure 5: Finding the running median and sums of absolute deviations