# Regression Trees

Brendan Kitts

Datasage Inc.
19 Newcrossing Road,
Reading, MA. 01867. USA
Email: bj@datasage.com

*Regression Trees* are an axis parallel Decision Tree which can induce trees for predicting both categorical and real-valued targets, and hence they can be used for regression as well as classification. For regression problems, the induction algorithm uses variance to determine the split quality. We conjecture that the accuracy of trees induced under the variance metric, will tend to be higher than the same trees induced under Entropy or Gini, for problems with metric targets, *ceteris paribus*. This corrects a small aspect of the work done by Murthy in his thesis (Murthy, 1998) where he compared different split measures, and concluded variance was among the poorest. We replicate Murthy's work, and show that by avoiding a discrete coding he used, variance-based trees trees have much higher accuracy than the other measures. We also show how variable significance or "predictiveness" can be determined by summing that variable's contribution throughout the tree, and confirm this method by comparing these results to stepwise regression.

# 1.0 Introduction

This paper discusses using axis-parallel decision trees to predict real vectors. This represents a small but useful modification to the usual algorithm for these decision trees as described by Quinlain (1993). This paper describes the details of *Regression Tree*, including variable significance estimation, and concludes with a general discussion about how to get the most out of decision trees.

# 2.0 Variance splits

### 2.1.1 Categorical splits

A decision tree is a tree data-structure, with arbitrary number of nodes and branches at each node. On each branch, a test is performed, for example, *Is salary > 21000?* If the test is true, the user follows down that branch to the next node. After traversing all of the nodes to a terminal node, the category stored at the terminal node is reported. For more comprehensive introductions to Decision Tree induction, the reader is directed to excellent introductory articles by Payne and Preece (1980), Moret (1982), Breimann, (1984), or Quinlain (1993).

Decision trees are induced by finding *if-then* rules which operate on the independent variabels, and which divide the dependent cases so as to reduce the discrepancy between the classifications of those cases. "Discrepency" is measured using Entropy, the Gini index, the Twoing rule, Maximum difference measures, or similar (Murthy, 1995). For instance, if the Entropy metric is used, then the overall split quality is a weighted combination of the entropies of the individual groups we have isolated, where each weight is equal to the number of elements falling into each partition. This complete split measure is given by

$$Spit\_purity(n_i) = Gain(n_i) = \frac{|n_i|}{\sum_j |n_j|} Edrop(n_i)$$

where

$$Edrop(n_i) = E(parent(n_i)) - E(n_i)$$

and entropy is defined as

$$E(P) = -\sum_{i=1}^{n} p_i log(p_i)$$

This works very well for classification problems. For instance, if our task is to decide if an instance belongs to discrete and un-ordered class, for example, *RED, BLUE, YELLOW*. However, *regression problems* assume that the target is a real number.

If we try to apply Entropy to regression problems, we run into the following problems:

1. Entropy does not use any information about the relative closeness of one value to another in calculating how good a split is. Therefore, if one split resulted in "*a"*s and "*b"*s being isolated, and another resulted in "*a"*s and "*z"*s, then the split quality would be the same. Intuitively, if the first were "1"s and "2"s, and the latter were "1"s and "100"s, then we should conclude that the first split was a better partition than the second. Therefore, for regression problems, we need a way of taking into account how close values.

2. If the target space is continuous then Entropy will treat each continuous target value as a distinct category (eg. 12.5 and 0.432), one for every value! This results in a proliferation of trivial one-member categories. This has been addressed by discretizing data prior to running a decision tree, eg. if we have values 0.53, 9.4, 12.5 then group the values $(0..10) \rightarrow 1$ and $(11..20) \rightarrow 2$. However, this opens a Pandora's Box of representational issues. For metric problems it can be better to just leave the data as is, and find a split measure which is more naturally suited to metric data, and does not result in a multitude of discrete categories.

Variance is such a criterion. Variance gives the average of all the distances from the mean.

$$Var(x) = \frac{1}{n} \sum_{i=1}^{n} (x_i - x_i)$$

To apply variance to decision trees, we perform exactly the same greedy minimization, but use the variance of the values as the measure of "partition goodness". The lower the variance, the better the classification. For the leaves of the tree, the final category (the estimate of the tree) is equal to the mean of the categories of all the values isolated at that leaf. The degree of uncertainty of this category can be calculated using the standard error of the leaf. The lower the standard error, the more certain is the category estimate.

$$Spit\_purity(n_i) = \frac{|n_i|}{\sum_j |n_j|} Var(x) \cdot \forall x \in n_i$$

### 2.1.2 Previous work with Variance-splits

Variance has been tried before as a decision tree split criterion. Murthy (1996) compared six split techniques including Variance (what he termed "Sum of Variances"), on four real-world problems. He concluded that Information Gain [Entropy], Gini Index, and the Twoing Rule perform better than the other measures, including Variance. (Murthy, 1996, pp. 57). However, Murthy's problem set consisted of all 1-0 categorical problems:

- Star Galaxy discrimination: predict star or galaxy.
- Breast cancer diagnosis: predict benign or malignant tumor.
- Diabetes: predict presence or absence of diabetes.
- Boston housing costs: predict the median value of owned homes. This is a continuous value, however Murthy actually discretized the category to a two-value category, 1 for value < $21,000 and 2 for value otherwise. He also tested the iris set, but excluded it because it had more than two categories.

Some of the problems were even originally continuous, but he *re-coded* them into two-value categories (!). This removed the rich data related to relative housing cost, and replaced it with a discrete two-value variable. Any "information" used by variance about closeness, would be misleading, since it would be related to the coding scheme[1]. Murthy compounded this problem with a complicated fix, uniformly shuffling category coding of this metric each split, so that coding scheme didn't bias the variance in a systematic way.

> To avoid this problem OC1 uniformly reassigns numbers according to the frequency of occurrence of each category at a node before computing the Sum of variances. (ibid., pp. 57)

Variance should have been applied to the *unaltered data*. On real-valued problems such as the Boston housing data, we might even expect variance to perform ***better*** than the categorical measures (Gini, Twoing, Entropy) for the reason that variance gives more detailed information about each partition.

To test this, we re-analysed the Boston Housing Problem. The Boston Housing problem consists of 12 variables including Nitric Oxide level, Charles river yes/no, age of house, and the problem is to estimate the yearly income of the household, a value between 0 and 660 (numbers in the thousands). To apply a decision tree to this problem, Murthy initially encoded all dependent values < $21K as 1 and >$21K as 2, and then ran his induction algorithms.

We ran *RegressionTree* using an identical 5-fold crossvalidation training test split used by Murthy, but using the original real coding with a variance split metric. We then post-coded *RegressionTree's* real-valued estimate for housing value, to a 1 or 2 for predictions < 21 or >21. Because *RegressionTree* creates axis-parallel splits, it should tend to be less accurate than oblique

[1] From Murthy's thesis (1998), "As this measure is computed using the actual class labels, it is easy to see that the impurity computed varies depending on how numbers are assigned to the classes. For instance, if T1 consists of 10 points of category 1 and 3 points of category2, and if T2 consists of 10 points of category 1 and 3 points of category 5, then the Sum Of Variances values are different for T1 and T2." (ibid., pp. 57)

decision trees, *ceteris paribus*. However, in the comparison chart below, *RegressionTree* has higher accuracy than all the other methods in Murthy's test suite. We conjecture that the reason why it performs better than both oblique methods and C4.5, is because the target variable in this problem is metric, and variance correctly captures more information about the quality of the split. As a result, the splits at each level are better quality, and the overall tree superior to the other methods forced to treat the set like a 0-1 category.

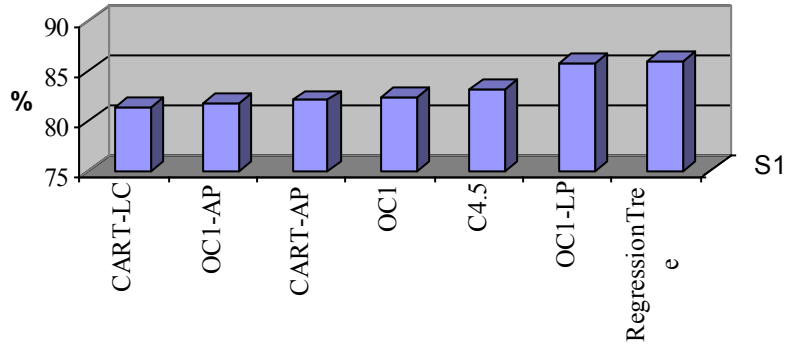


**Figure 1:** 5-fold crossvalidation results for different algorithms including RegressionTree. OC1 should be more accurate than RegressionTree since it can create oblique splits. However, RegressionTree is the most accurate method. We suggest the reason is that RegressionTree uses variance as a measure of quality, which is a more appropriate splitting criterion for this particular problem in which the target is real-valued and metric.

# 3.0 Regression Tree Implementation Details

The following section describes Regression Tree's particular implementation of pruning, missing value handling, and significance testing. In most ways Regression Tree is similar to other decision trees proposed in the literature, including the ubiquitos C4.5 (Quinlain, 1993). The most important innovation with Regression Tree, is its use of variance minimization, and simple and neat significance testing after the fact. Significance testing can be used for diagnosis, to determine which variables in a problem are most useful. After describing these aspects of the implementation, the performance of the algorithm on regression problems is tested.

## 3.2 Significance of each variable

Regression Tree implements a novel technique for determining the most significant variables in a prediction problem. Each split in the decision tree corresponds to a discriminant being made on a particular dimension. To determine the usefulness of each dimension, all we need to do is to add up the individual variance (or entropy) decreases for each dimension, weighted by the number of datapoints which experienced the entropy-drop, ie.

$$S(i) = \frac{\sum_{n:\dim(n)=i} |n| \cdot E(parent(n)) - E(n)}{\sum_{n} |n| \cdot E(parent(n) - E(n)}$$

Using stepwize regression on the Abalone problem, variables 6 and 8 were found to be the most significant variables for estimating the dependent variable. Using RegressionTree, the most significant variables were also 6 and 8 (see figure 10).

## 3.3 Missing value handling

Decision trees are one of the few models which can elegantly cope with missing values. During execution, if an exemplar has a missing value in a dimension which is being tested for a split, then each arm of the split is traversed and weighted by the prior probability of falling into each partition (the incidence of all exemplars). The resulting categories are linearly combined by these weightings.

To describe this in a little more detail, let's formalize a tree using a Context Free Grammar:

| | | |
|---|---|---|
| *T* | → | *C* |
| *T* | → | *BranchSet* |
| *BranchSet* | → | *Branch Branchset* \| *Branch else Branchset* \| $\varepsilon$ |
| *Branch* | → | *if B then T* |
| *B* | → | *x<val* \| *x==val* |

*T* is a tree, *B* is a branch test, and *C* is the predicted category. Let $|B_i|$ be the number of elements which result from applying the $B_i$ partition (if $B_i$ then *N*) to the data. If we don't know the value *x*, which is being tested at branch $B_i$, then we must resort to calculating the probabilities of each branch. The probability of choosing $B_i$ by pure chance is

$$P(d \in B_i) = \frac{|B_i|}{\sum_{j=1}^{n} |B_j|}$$

If the value is unknown, we choose each branch, and weight the category probability which we find on the final branch according to the prior probability of choosing that initial branch. If after selecting one branch based on missing data, we then encounter another branch where our value is unknown, then we use the same probability rule again. Therefore, the resulting category probability on the leaf, P(Ci), is weighted by the accumulated uncertainties caused by missing values, to give an adjusted P(Ci)', and we can express this using our normal probability rule

$$P(C_j)' = P(C_j^T) \prod_i P(B_i)$$

where $P(C_j^T)$ is the probability of choosing class $C_j$ from the training set. The predicted category is then the category with the highest probability,

$$C_i \bullet \max P(C_i)'$$

Training can be handled in much the same way. If a value is missing, then that value is ignored during the entropy calculation, however, the case is then passed to the resulting branches carrying a probability based on the prior for falling into each path.

## 3.4 Pruning

Pruning is the standard way that decision trees are prevented from overfitting data. Pruning involves performing a complete induction on the data, splitting the nodes until they cannot be split any further. A pruning algorithm then tests the performance of the tree with and without its lower nodes, and those nodes are removed if the test set error improves.

*RegressionTree* implements hold-out set pruning. The algorithm first locates all parents which are penultimate nodes in the tree, that is, whose leaves are comprised only of terminals. If a parent has some leaves which are terminals, and others which are subtrees, then this parent cannot be used. Next, the leaves of these parents are removed, and the tree run using a holdout set. If error improves or stays the same on the smaller tree, then the leaves are permanently removed. The parent of this node is now considered as a possible penultimate node, and the removal process continues. Removal runs breadth-first, and stops after none of the parent leaf removals improve the error.

Many decision trees also implement pre-pruning. Pre-pruning automatically halts tree induction after a certain criterion is reached, and can be useful by forcing the tree to terminate within minutes rather than hours. In the case of C5.0, creation of a new child node is stopped if the following criteria are met:

a) a Chi square significance test for the entropy improvement of the child falls below a certain threshold
b) there are no longer two or more branches which have at least two cases in them (note that this seems like an awfully complex stopping critera!)

In RegressionTree tree induction for a node stops if:

a) the entropy of the new partition is lower than x% of the original entropy.
b) The number of datapoints in the new partition are lower than x% of the original data in the set

## 3.5 Confidence of Predictions

When working in regression mode (minimizing target variance), RegressionTree's category estimate is equal to the mean of the target values. We can also calculate the "uncertainty" of this estimate, by calculating its standard error. The standard error is a measure of how poorly mixed the training cases were in that node. The standard error is equal to

$$SE(x) = \sqrt{\frac{\sigma}{N}}$$

Given a 95% probability that the true category mean is within range(x,0.95) percent of the given category estimate, this range is calculated as:

$$range(x,0.95) = 100 \times \left| \frac{1.96 \times SE}{x} \right|$$

For example, the following might be reported: 12.5 +/- 50%. This means that there is a 95% chance that the actual value of the thing we are trying to predict is between 6 and 18.

In categorical mode, RegressionTree provides the (training set estimated) probability for every category. The probability of input $x_i$ belonging to category $y_i$ is calculated as

$$P(i \mid n) = \frac{card((x_j, y_j) \bullet y_j = i)}{card((x, y) \in n)}$$

For example,

```
category    2.000000     0.000000        1.000000
cat prob    0.058824     0.147059        0.794118
```

shows that there is a 6% chance that category 2 is correct, a 15% chance that category 0 is correct, and a 79% chance that category 1 is correct. Reporting the complete probability distribution could show, for example, that a "2" and a "3" are easily mistaken for each other (eg. P(2) = 0.42, P(3) = 0.43). Therefore we might be able to make the learning problem easier by these categories into the one category.

# 4.0 Performance

In order to test the performance of Regression Tree and see how it compared to other algorithms including other decision trees, a series of test problems were generated. The results of these tests are described below.

### 4.1 Nonlinear regression

The non-linear regression problem was to predict a number which was actually equal to the square of the first dimension of the input variables, plus some constant noise in the range E = [0..5]. Therefore,

$$y = x_1^2 + E$$

In figure 3 the significance for each variable on the continuous valued problem is reported. Only the first dimension was used, and so *RegressionTree* effectively picked out that dimension for discriminating, and ignored the other distractors. In figure 4, the jagged line is the prediction, and the smooth line the test function. This profile is very similar in spirit to a nearest neighbour estimator, since both provide a localized estimate which does not interpolate between regions

```
cat 151.419998 +/- 16.65%
 if (1<14.000000) then
     cat 68.907692 +/- 19.16%
     if (1<9.000000) then
         cat 28.315790 +/- 21.76%
     if (1>=9.000000) then
         cat 126.037041 +/-  9.40%
         if (1<12.000000) then
             cat 105.235291 +/-  7.89%
         if (1>=12.000000) then
             cat 161.399994 +/-  4.80%
 if (1>=14.000000) then
     cat 304.657135 +/-  8.19%
     if (1<18.000000) then
         cat 242.157898 +/-  5.47%
         if (1<16.000000) then
             cat 214.888885 +/-  4.58%
         if (1>=16.000000) then
             cat 266.700012 +/-  3.07%
     if (1>=18.000000) then
         cat 378.875000 +/-  4.56%
         if (1<20.000000) then
             cat 355.500000 +/-  2.95%
         if (1>=20.000000) then
             cat 417.833344 +/-  3.84%
```

**Figure 2:** *RegressionTree*'s solution to $y=x^2$ , $x \varepsilon [1..20]$

```
Contribution of each dimension:
Dimension 1 variance drop 98.15%
Dimension 2 variance drop  0.00%
Dimension 3 variance drop  0.00%
Training set variance accounted for 98.15%
```

**Figure 3:** Significance for each variable on the continuous valued problem. Only the first dimension was used, and so *RegressionTree* effectively picked out that dimension for discriminating, and ignored the other distractors.

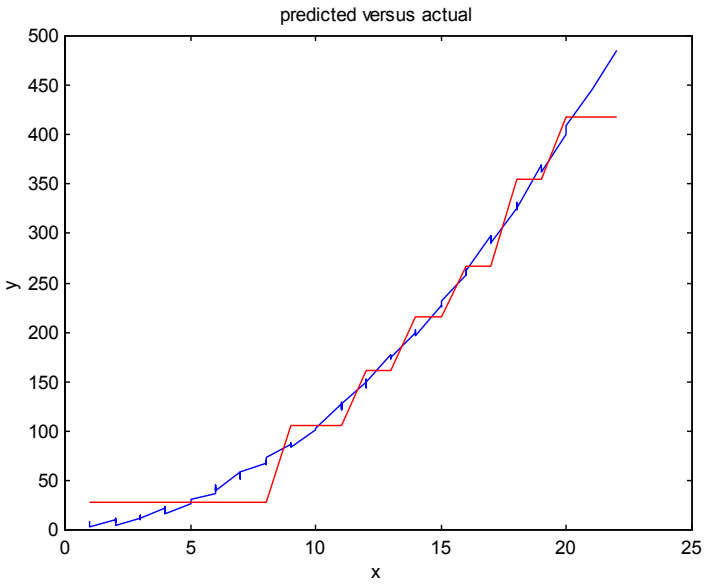


**Figure 4:** Quadratic regression test problem, $y = x^2$. The jagged line is the prediction. This profile is very similar in spirit to a nearest neighbour estimator, since both provide a localized estimate which does not interpolate between regions.

## 4.2 Noisy category identity map

This problem was to categorize data as either a 1,2,or 3. The targets were identical to the value of the first dimension. Thus, the other dimensions were distractors. There was also a 20% chance of having the target category flipped to a random category. On this problem, RegressionTree built the following tree:

```
cat 2.000000 err  64%
 if (1==1.000000) then
     cat 1.000000 err  21%
     category 2.000000  0.000000        1.000000
     cat prob 0.058824  0.147059        0.794118
 if (1==2.000000) then
     cat 2.000000 err   6%
     category 2.000000  0.000000        1.000000
     cat prob 0.939394  0.000000        0.060606
 if (1==0.000000) then
     cat 0.000000 err  15%
     category 2.000000  0.000000        1.000000
     cat prob 0.090909  0.848485        0.060606
```

**Figure 5:** Induced tree for the 1-2-3 classification task. Note that probabilities for each category are given.

Thus if dimension1 equals 1 then predict 1 (probability 79%), if dimension1 equals 2, then predict 2 (probability 94%) and if dimension1 equals 0 then predict 0 (probability 85%), which is the desired relationship. The significance chart also shows that dimension 1 was the only useful dimension, which is the correct inference.

```
contribution of each dimension:
Dimension 1 entropy drop 63.10%
Dimension 2 entropy drop  0.00%
Dimension 3 entropy drop  0.00%

Training set entropy accounted for 63.10%
```

**Figure 6:** Significance for each variable on the 1-2-3 categorization task. This shows that RegressionTree used the first dimension only for making splits. Therefore, the algorithm effectively ignored the "distractor dimensions".

### 4.3 Two gaussians

The "Two Gaussians" function is as follows:

$$y = x_1 \cdot e^{(-x_1^2 - x_2^2)}$$

A plot of this function is shown in figure 7. *RegressionTree* was run on this problem and stopped after entropy or datapoints reduced to 20% of the value at the original node. Therefore, only the major surfaces were induced. Figure 8 depicts the resulting surface. This demonstrates nicely that *RegressionTree* approximates by boxing in regions with combinations of hyperplanes.

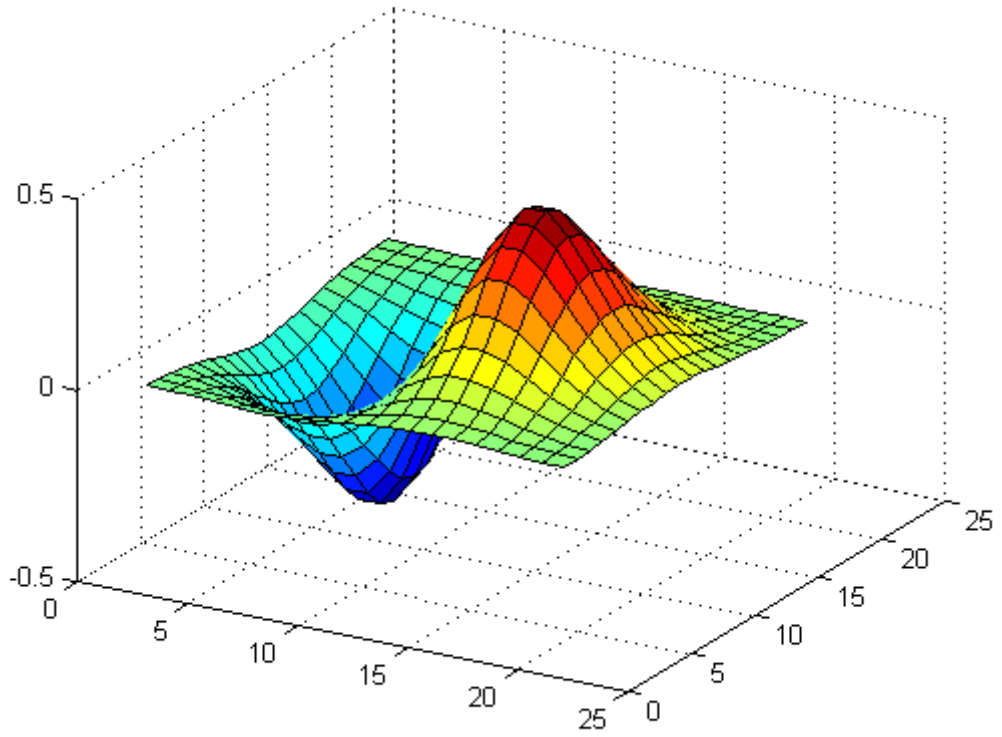


**Figure 7:** Graph of the Two Gaussians problem

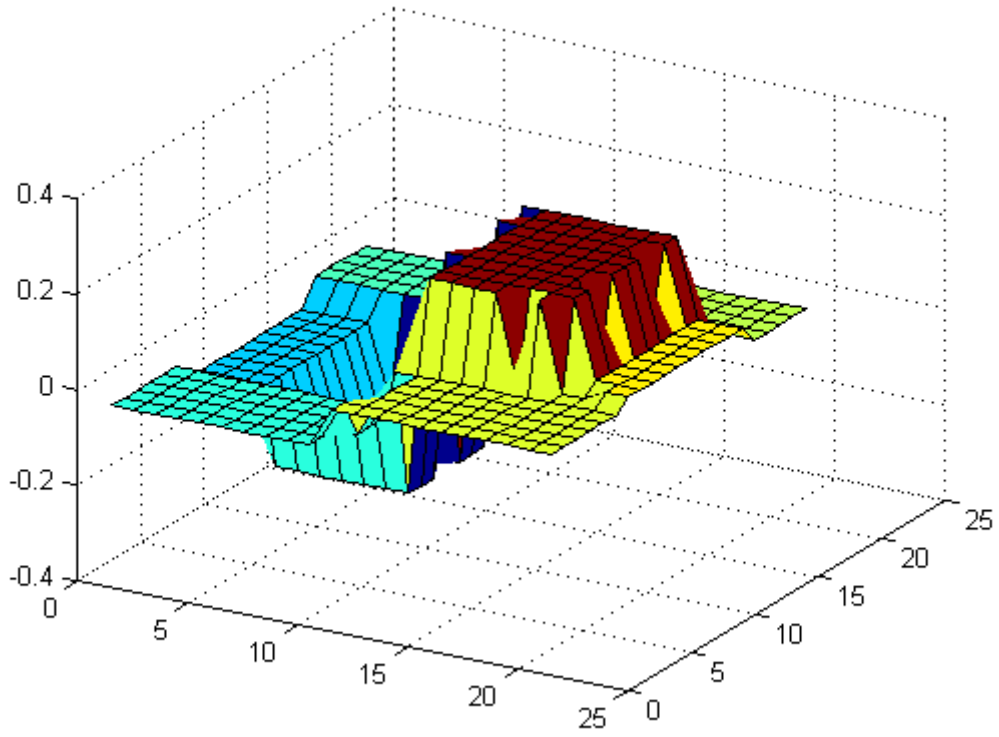


**Figure 8:** *RegressionTree*'s estimation for the two-gaussians problem.

## 4.4 Boston Housing

As discussed in Section 3, the Boston Housing problem was also tested using this algorithm. *RegressionTree* achieved a "percent correct" of 86%, which was better than Murthy (1998). We also applied RT using continuous values, and achieved a median and mean absolute error of 6.01K and 19.69K respectively, and a correlation between actual and predicted of R=0.9478, and $R^2$= 0.8983, which signifies a very good fit.

## 4.5 Abalone problem

The Abalone problem is another UCI Machine Learning Repository problem (Merz and Murphy, 1998). The Abalone database was compiled by the Department of Primary Industry and Fisheries, Australia, and used by Waugh (1995) in his thesis on the Cascade Correlation algorithm. Its is a difficult problem, and not linearly separable without additional information. This problem is to estimate the number of growth rings in an Abalone by looking at external characteristics.

Waugh's results were: 24.86% Cascade-Correlation (no hidden nodes), 26.25% Cascade-Correlation (5 hidden nodes), 21.5% C4.5, 0.0% Linear Discriminate Analysis, 3.57% k=5 Nearest Neighbour, (Problem encoded as a classification task). Using the same training-test split (3000,1000), RegressionTree obtained 26% Using continuous targets (which couldn't be compared against Waugh's findings) we obtained an $R^2$ model fit of 0.36.

```
cat 9.939500 +/-  1.46%
 if (8<0.195000) then
     cat 7.812349 +/-  2.02%
 if (8>=0.195000) then
     cat 11.436116 +/-  1.55%
     if (8<0.410000) then
         cat 11.009463 +/-  1.62%
         if (6<0.250500) then
             cat 13.958333 +/-  7.86%
         if (6>=0.250500) then
             cat 10.852714 +/-  1.59%
             if (8<0.255000) then
                 cat 9.687023 +/-  2.28%
             if (8>=0.255000) then
                 cat 11.329173 +/-  1.90%
                 if (6<0.403500) then
                     cat 12.917127 +/-  3.83%
                 if (6>=0.403500) then
                     cat 10.704348 +/-  1.88%
     if (8>=0.410000) then
         cat 13.255606 +/-  3.54%
```

**Figure 9:** Output from RegressionTree on the Abalone problem (continuous targets). This also shows that the only variables found to be useful by the algorithm were 8 and 6.

```
Contribution of each dimension:
Dimension 1 variance drop  0.00%
Dimension 2 variance drop  0.00%
Dimension 3 variance drop  0.00%
Dimension 4 variance drop  0.00%
Dimension 5 variance drop  0.00%
Dimension 6 variance drop  4.88%
Dimension 7 variance drop  0.00%
Dimension 8 variance drop 35.29%
Training set variance accounted for 40.17%
```

**Figure 10:** Significant variables in the Abalone problem

### 4.6 Summary

A sample of test cases has been presented which show that RegressionTree seems to be performing well, or at least as well as other implementations of C4.5. These cases also show nicely how Regression Tree (and other Decision Trees) approximate by building hyperplanes. In specific cases such as the Boston Housing problem, the use of the variance metric also appears to give the algorithm a distinct advantage, and allow it to take advantage of more information about its data.

# 5.0 Getting the most out of Decision Trees

This final section will discusses some helpful hints for practicioners wanting to get the most out of decision trees. The general points in this section should be true for most popoular decision trees including *RegressionTree*, C4.5, CHAID, and CART.

## 5.1 Brief Background of Decision Trees

20 years ago expert systems became the first major AI technique to find commercial viability. These systems were deployed on a large scale for a variety of problems, from mineral exploration, to copier machine configuration.

However, to build these systems, one needed access to a human expert who could supply the rules. The knowledge elicitation process was a painstaking, time consuming, and expensive process, and involved taking the best people in a department and diverting them into an intensive knowledge capture process. This situation was made more difficult by work in Cognitive Psychology which showed that much of human expertise is not explicit, and therefore not directly accessable by experts (Dreyfus, 1971).

Decision Trees were conceived as an answer to the Knowledge Acquisition Bottleneck. Decision trees automatically induce rules (or trees) from data, and therefore work without access to a human expert.

It was later discovered that decision trees have specific advantages over other machine learning algorithms that are worthy of study in their own right. Specifically, decision trees are more interpretable than neural networks, and they generate human-readable, human-understandable rules. This virtue has become one of this model's main attractions in the late 1990s. Today the emphasis is on data *understanding* – making sense of the vast bytes of data which are being generated by automated data collection hardware such as EPOS and credit transactions. The ability to automatically induce rules presents immense opportunity for automatically distilling and interpreting this data.

## 5.2 Using Decision Trees

RegressionTree, like other decision trees, **tests every split**, when inducing its tree. Therefore, used naively, decision tree algorithms can spend days inducing their tree. Because of this reason, practicioners must take some steps to ensure that the algorithm terminates within a reasonable amount of time. The purpose of this section is to provide a general guide as to what should and should not work with RegressionTree, but by no means will hold for every domain, as has been well established by researchers in several domains (Rasmussen, Neal, et. al., 1986).

## 5.3 Interpretability versus accuracy

One of the most important features of decision trees is their explanatory power. Because the tree generates a set of if-then rules of the form "if x>num then ", their classifications can be intuitively understood by human beings. At best they can provide some insight about why a particular classification has been made. For example, if the tree classifies someone as a high spender, we might look at the rules and observe that income==high, house==expensive, meaning that customers who are high spenders tend to have high income and own houses.

This interpretability is also one of its weaknesses. Rules which are of the form "if x>num then y" make a split parallel to variable axes. Thus, the type of approximation decision trees can do is limited to hyperplanes and "boxing in" categories. Thus, decision trees may be more interpretable than other models, but may not be as accurate since their decision surfaces are restricted to planes.

Therefore, when using Decision trees, your model accuracy may not be as high as other models, however, it may be useful for understanding your domain better.

## 5.4 Best Split < or ==

As we have seen, RegressionTree implements two kinds of splits, discrete splits which test equal to each arm of the split, and continuous splits, which split if dim<value.

In speed terms, < can result in faster and smaller trees than ==. In the best case, < can result in a binary partitioning of the data, resulting in lookup speed of O(log N). In the worst case, < must be used to test every possible value, and so functions the same as ==. Therefore, for building fast trees, < is preferred.

It has also been noted that < results in better quality partitions than == (REF). The reason is that decision tree induction is a greedy process, and < does not make as much of a committment as == does. < only results in a single partition, where-as == causes the adoption of n partitions. Thus, < is a more conservative strategy, which gives the algorithm more opportunities to look again at the data, and adjust the split accordingly.

The only time when == should be preferred is if a dimension doesn't have a metric ordering (eg. gree, yellow, blue). In this case < makes "no sense" given the semantics of the data. If < is used, it could result in artifacts which won't hold on a test set.

In summary, < results in smaller, faster and even more accurate trees, unless the variables are categorical.

## 5.5 Dealing with continuous input variables

Continuous variables are the bane of decision trees. They result in performance analogous to wading through thick porridge. Decision trees scale quadratically with the number of values in each dimension, and test every possible split. Therefore, as the values on a dimension increase, decision trees rapidly move from passable performance into intractable slowness.

The best way to approach this problem is to discretize your continuous variables prior to reaching the decision tree algorithm. This can be as simple as dividing by 10, casting as an int (rounding), and then multiplying by 10. The precision loss is almost negligable, and will speed up tree induction, allowing better analysis to be performed.

## 5.6 Target variables as categories or metric values

If the target variable is metric (eg. a number, as in salary or net present value), it should generally not be treated as a category. Categorical splitting is not as informative as splits based on variance (which take into account similarity of each category). Therefore the splits will be better if they exploit the similarity information.

There are rare cases where one might want to treat metric data as categories. If discontinuities or singularities in the category space exist, then quantizing the category values and treating them like categories will provide a more robust solution (since the mean will then be meaningless).

## 5.7 Summary

Summarizing the above, analysts should try to do the following:

1. discretize your input variables to speed up the induction
2. use variance unless the target is purely categorical with no metric.
3. unless the input data is categorical, set continuous_thresh=2
   (ie. always use < splits)
4. use early stopping when testing the algorithm, to prevent the algorithm
   from taking too long.

# References


Breiman, L., Friedman, J., et. al., (1984), *Classification and Regression Trees*, Wadsworth International Group.

Heath, D., Kasif, S., and Salzberg, S., (1993), Learning Oblique decision trees, *International Joint Conference on Artificial Intelligence*, ed: Bajcsy, R., Vol. 221, pp. 1002-1007.

Merz, C.J., & Murphy, P.M. (1998). UCI Repository of machine learning databases, http://www.ics.uci.edu/~mlearn/MLRepository.html Irvine, CA: University of California, Department of Information and Computer Science.

Moret, B. (1982), Decision Trees and Diagrams, *Computing Surveys*, Vol. 14, pp. 593-623.

Murthy, S. (1995), Growing Better Decision Trees from Data, PhD Thesis, John Hopkins University, http://www.cs.jhu.edu/~murthy/thesis/

Payne, R. and Preece, D. (1980), Identification Keys and Diagnostic tables: A review, Journal of the Royal Statistical Society, Series A, Vol. 143, pp. 253.

Rasmussen, C., Neal, R., Hinton, G. (1996), The DELVE Manual, University of Toronto, http://www.cs.utoronto.ca/~delve

Quinlan, J. (1993), C4.5: Programs for Machine Learning, San Mateo: Morgan Kaufmann.

Quinlan, J. (1996), Improved use of continuous attributes in C4.5, Journal of Artificial Intelligence Research, Vol. 4, pp. 77-90.

Waugh, S. (1995), Extending and benchmarking Cascade-Correlation, PhD thesis, Computer Science Department, University of Tasmania.

Wnek, J., Michalski, R. (1994), "Hypothesis-driven constructive induction in AQ17-HCI: A Method and Experiments", *Machine Learning*, Vol. 14, pp. 139, Kluwer Academic Publishers, Boston.