An Introduction to Recursive Partitioning Using the RPART Routines

Terry M. Therneau
Elizabeth J. Atkinson
Mayo Foundation

January 11, 2011

Contents

1 Introduction

This document is an update of a technical report written several years ago at Stanford [6], and is intended to give a short overview of the methods found in the `rpart` routines, which implement many of the ideas found in the CART (Classification and Regression Trees) book and programs of Breiman, Friedman, Olshen and Stone [1]. Because CART is the trademarked name of a particular software implementation of these ideas, and *tree* has been used for the S-plus routines of Clark and Pregibon ∼[3] a different acronym — Recursive PARTitioning or rpart — was chosen.

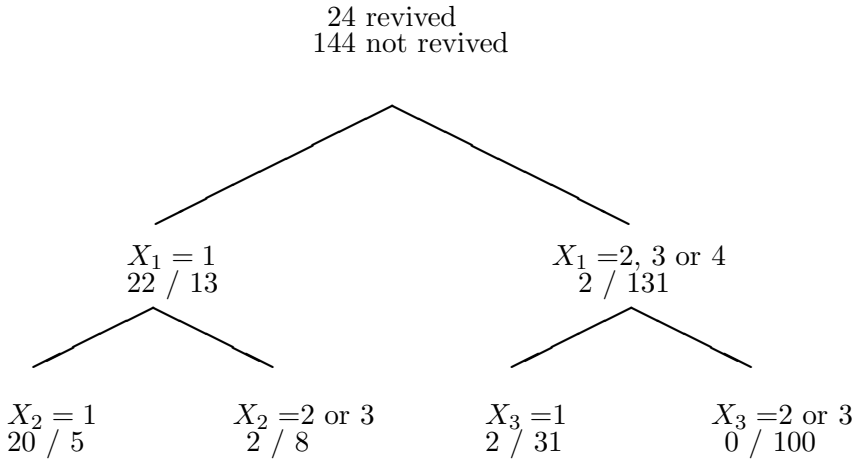


Figure 1: *Revival data*

The `rpart` programs build classification or regression models of a very general structure using a two stage procedure; the resulting models can be represented as binary trees. An example is some preliminary data gathered at Stanford on revival of cardiac arrest patients by paramedics. The goal is to predict which patients are revivable in the field on the basis of fourteen variables known at or near the time of paramedic arrival, e.g., sex, age, time from attack to first care, etc. Since some patients who are not revived on site are later successfully resuscitated at the hospital, early identification of these "recalcitrant" cases is of considerable clinical interest.

The resultant model separated the patients into four groups as shown in figure 1, where

$X1$ = initial heart rhythm
1= VF/VT 2=EMD 3=Asystole 4=Other

X_2 = initial response to defibrillation
1=Improved 2=No change 3=Worse

X_3 = initial response to drugs
1=Improved 2=No change 3=Worse

The other 11 variables did not appear in the final model. This procedure seems to work especially well for variables such as X_1 , where there is a definite ordering, but spacings are not necessarily equal.

The tree is built by the following process: first the single variable is found which best splits the data into two groups ('best' will be defined later). The data is

separated, and then this process is applied *separately* to each sub-group, and so on recursively until the subgroups either reach a minimum size (5 for this data) or until no improvement can be made.

The resultant model is, with a certainty, too complex, and the question arises as it does with all stepwise procedures of when to stop. The second stage of the procedure consists of using cross-validation to trim back the full tree. In the medical example above the full tree had ten terminal regions. A cross validated estimate of risk was computed for a nested set of subtrees; this final model was that subtree with the lowest estimate of risk.

2 Notation

The partitioning method can be applied to many different kinds of data. We will start by looking at the classification problem, which is one of the more instructive cases (but also has the most complex equations). The sample population consists of n observations from C classes. A given model will break these observations into k terminal groups; to each of these groups is assigned a predicted class. In an actual application, most parameters will be estimated from the data, such estimates are given by $\approx$ formulae.

π_i	$i = 1, 2, ..., C$	prior probabilities of each class
$L(i, j)$	$i = 1, 2, ..., C$	Loss matrix for incorrectly classifying an i as a j. $L(i, i) \equiv 0$
A		some node of the tree Note that A represents both a set of individuals in the sample data, and, via the tree that produced it, a classification rule for future data.
$\tau(x)$		true class of an observation x, where x is the vector of predictor variables
$\tau(A)$		the class assigned to A, if A were to be taken as a final node
n_i, n_A		number of observations in the sample that are class i, number of obs in node A
$P(A)$		probability of A (for future observations)

$= \sum_{i=1}^{C} \pi_i \, P\{x \in A \mid \tau(x) = i\}$
$\approx \sum_{i=1}^{C} \pi_i n_{iA}/n_i$

$p(i|A)$ $P\{\tau(x) = i \mid x \in A\}$ (for future observations)
$= \pi_i P\{x \in A \mid \tau(x) = i\}/P\{x \in A\}$
$\approx \pi_i(n_{iA}/n_i)/\sum \pi_i(n_{iA}/n_i\ )$

$R(A)$ risk of A
$= \sum_{i=1}^{C} p(i|A)L(i, \tau(A))$
where $\tau(A)$ is chosen to minimize this risk

$R(T)$ risk of a model (or tree) T
$= \sum_{j=1}^{k} P(A_j)R(A_j)$
where A_j are the terminal nodes of the tree

If $L(i, j) = 1$ for all $i \neq j$, and we set the prior probabilities π equal to the observed class frequencies in the sample then $p(i|A) = n_{iA}/n_A$ and $R(T)$ is the proportion misclassified.

3 Building the tree

3.1 Splitting criteria

If we split a node A into two sons A_L and A_R (left and right sons), we will have

$$P(A_L)r(A_L) + P(A_R)r(A_R) \leq P(A)r(A)$$

(this is proven in [1]). Using this, one obvious way to build a tree is to choose that split which maximizes Δr, the decrease in risk. There are defects with this, however, as the following example shows.

> Suppose losses are equal and that the data is 80% class 1's, and that some trial split results in A_L being 54% class 1's and A_R being 100% class 1's. Since the minimum risk prediction for both the left and right son is $\tau(A_L) = \tau(A_R) = 1$, this split will have $\Delta r = 0$, yet scientifically this is a very informative division of the sample. In real data with such a majority, the first few splits very often can do no better than this.

A more serious defect is that the risk reduction is essentially linear. If there were two competing splits, one separating the data into groups of 85% and 50% purity

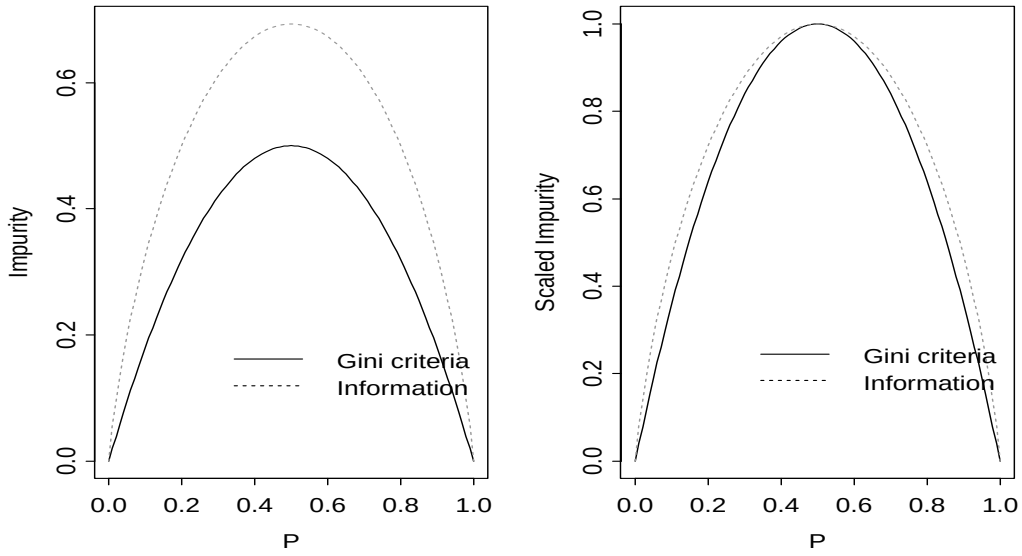


Figure 2: *Comparison of Gini and Information indices*

respectively, and the other into 70%-70%, we would usually prefer the former, if for no other reason than because it better sets things up for the next splits.

One way around both of these problems is to use lookahead rules; but these are computationally very expensive. Instead `rpart` uses one of several measures of impurity, or diversity, of a node. Let f be some impurity function and define the impurity of a node A as

$$I(A) = \sum_{i=1}^{C} f(p_{iA})$$

where p_{iA} is the proportion of those in A that belong to class i for future samples. Since we would like $I(A) = 0$ when A is pure, f must be concave with $f(0) = f(1) = 0$.

Two candidates for f are the information index $f(p) = -p\log(p)$ and the Gini index $f(p) = p(1-p)$. We then use that split with maximal impurity reduction

$$\Delta I = p(A)I(A) - p(A_L)I(A_L) - p(A_R)I(A_R)$$

The two impurity functions are plotted in figure (2), with the second plot scaled so that the maximum for both measures is at 1. For the two class problem the measures differ only slightly, and will nearly always choose the same split point.

Another convex criteria not quite of the above class is twoing for which

$$I(A) = \min_{C_1 C_2} [f(p_{C_1}) + f(p_{C_2})]$$

where C_1,C_2 is some partition of the C classes into two disjoint sets. If $C = 2$ twoing is equivalent to the usual impurity index for f. Surprisingly, twoing can be calculated almost as efficiently as the usual impurity index. One potential advantage of twoing is that the output may give the user additional insight concerning the structure of the data. It can be viewed as the partition of C into two superclasses which are in some sense the most dissimilar for those observations in A. For certain problems there may be a natural ordering of the response categories (e.g. level of education), in which case ordered twoing can be naturally defined, by restricting C_1 to be an interval $[1, 2, \ldots, k]$ of classes. Twoing is not part of `rpart`.

3.2 Incorporating losses

One salutatory aspect of the risk reduction criteria not found in the impurity measures is inclusion of the loss function. Two different ways of extending the impurity criteria to also include losses are implemented in CART, the generalized Gini index and altered priors. The `rpart` software implements only the altered priors method.

3.2.1 Generalized Gini index

The Gini index has the following interesting interpretation. Suppose an object is selected at random from one of C classes according to the probabilities $(p_1, p_2, ..., p_C)$ and is randomly assigned to a class using the same distribution. The probability of misclassification is

$$\sum_i \sum_{j \neq i} p_i p_j = \sum_i \sum_j p_i p_j - \sum_i p_i^2 = \sum_i 1 - p_i^2 = \text{Gini index for } p$$

Let $L(i, j)$ be the loss of assigning class j to an object which actually belongs to class i. The expected cost of misclassification is $\sum \sum L(i, j) p_i p_j$. This suggests defining a *generalized Gini* index of impurity by

$$G(p) = \sum_i \sum_j L(i, j) p_i p_j$$

The corresponding splitting criterion appears to be promising for applications involving variable misclassification costs. But there are several reasonable objections to it. First, $G(p)$ is not necessarily a concave function of p, which was the motivating

factor behind impurity measures. More seriously, G symmetrizes the loss matrix before using it. To see this note that

$$G(p) = (1/2) \sum \sum [L(i,j) + L(j,i)]\, p_i p_j$$

In particular, for two-class problems, G in effect ignores the loss matrix.

3.2.2 Altered priors

Remember the definition of $R(A)$

$$\begin{aligned} R(A) &\equiv \sum_{i=1}^{C} p_{iA} L(i, \tau(A)) \\ &= \sum_{i=1}^{C} \pi_i L(i, \tau(A)) (n_{iA}/n_i)(n/n_A) \end{aligned}$$

Assume there exists $\tilde{\pi}$ and $\tilde{L}$ be such that

$$\tilde{\pi}_i \tilde{L}(i,j) = \pi_i L(i,j) \qquad \forall i, j \in C$$

Then $R(A)$ is unchanged under the new losses and priors. If $\tilde{L}$ is proportional to the zero-one loss matrix then the priors $\tilde{\pi}$ should be used in the splitting criteria. This is possible only if L is of the form

$$L(i,j) = \begin{cases} L_i & i \neq j \\ 0 & i = j \end{cases}$$

in which case

$$\tilde{\pi}_i = \frac{\pi_i L_i}{\sum_j \pi_j L_j}$$

This is always possible when $C = 2$, and hence altered priors are exact for the two class problem. For arbitrary loss matrix of dimension $C > 2$, `rpart` uses the above formula with $L_i = \sum_j L(i,j)$.

A second justification for altered priors is this. An impurity index $I(A) = \sum f(p_i)$ has its maximum at $p_1 = p_2 = \ldots = p_C = 1/C$. If a problem had, for instance, a misclassification loss for class 1 which was twice the loss for a class 2 or 3 observation, one would wish I(A) to have its maximum at $p_1 = 1/5$, $p_2 = p_3 = 2/5$, since this is the worst possible set of proportions on which to decide a node's class. The altered priors technique does exactly this, by shifting the p_i.

Two final notes

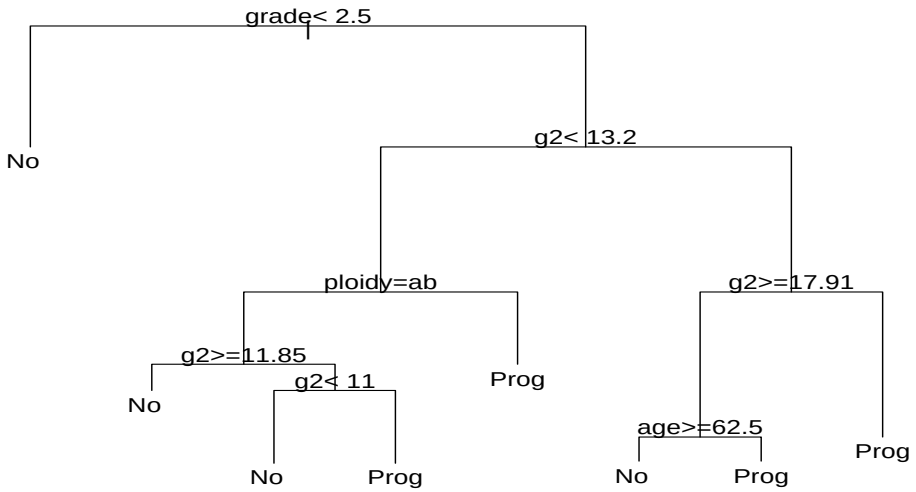


Figure 3: *Classification tree for the Stage C data*

- When altered priors are used, they affect only the choice of split. The ordinary losses and priors are used to compute the risk of the node. The altered priors simply help the impurity rule choose splits that are likely to be "good" in terms of the risk.
- The argument for altered priors is valid for both the gini and information splitting rules.

3.3 Example: Stage C prostate cancer (`class` method)

This first example is based on a data set of 146 stage C prostate cancer patients [5]. The main clinical endpoint of interest is whether the disease recurs after initial surgical removal of the prostate, and the time interval to that progression (if any). The endpoint in this example is `status`, which takes on the value 1 if the disease has progressed and 0 if not. Later we'll analyze the data using the `exp` method, which will take into account time to progression. A short description of each of the variables is listed below. The main predictor variable of interest in this study was DNA ploidy, as determined by flow cytometry. For diploid and tetraploid tumors, the flow cytometric method was also able to estimate the percent of tumor cells in a G_2 (growth) stage of their cell cycle; G_2% is systematically missing for most aneuploid tumors.

The variables in the data set are

pgtime	time to progression, or last follow-up free of progression
pgstat	status at last follow-up (1=progressed, 0=censored)
age	age at diagnosis
eet	early endocrine therapy (1=no, 0=yes)
ploidy	diploid/tetraploid/aneuploid DNA pattern
g2	% of cells in G_2 phase
grade	tumor grade (1-4)
gleason	Gleason grade (3-10)

The model is fit by using the `rpart` function. The first argument of the function is a model formula, with the $\sim$ symbol standing for "is modeled as". The `print` function gives an abbreviated output, as for other S models. The `plot` and `text` command plot the tree and then label the plot, the result is shown in figure 3.

```
> progstat <- factor(stagec$pgstat, levels=0:1, labels=c("No", "Prog"))
> cfit  <- rpart(progstat ~  age + eet + g2 + grade + gleason + ploidy,
               data=stagec, method='class')
> print(cfit)

node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 146 54 No ( 0.6301 0.3699 )
   2) grade<2.5 61  9 No ( 0.8525 0.1475 ) *
   3) grade>2.5 85 40 Prog ( 0.4706 0.5294 )
     6) g2<13.2 40 17 No ( 0.5750 0.4250 )
      12) ploidy:diploid,tetraploid 31 11 No ( 0.6452 0.3548 )
        24) g2>11.845 7  1 No ( 0.8571 0.1429 ) *
        25) g2<11.845 24 10 No ( 0.5833 0.4167 )
          50) g2<11.005 17  5 No ( 0.7059 0.2941 ) *
          51) g2>11.005 7  2 Prog ( 0.2857 0.7143 ) *
      13) ploidy:aneuploid 9  3 Prog ( 0.3333 0.6667 ) *
     7) g2>13.2 45 17 Prog ( 0.3778 0.6222 )
      14) g2>17.91 22  8 No ( 0.6364 0.3636 )
        28) age>62.5 15  4 No ( 0.7333 0.2667 ) *
        29) age<62.5 7  3 Prog ( 0.4286 0.5714 ) *
      15) g2<17.91 23  3 Prog ( 0.1304 0.8696 ) *

> plot(cfit)
> text(cfit)
```

- The creation of a labeled factor variable as the response improves the labeling of the printout.

- We have explicitly directed the routine to treat `progstat` as a categorical variable by asking for `method='class'`. (Since `progstat` is a factor this would have been the default choice). Since no optional classification parameters are specified the routine will use the Gini rule for splitting, prior probabilities that are proportional to the observed data frequencies, and 0/1 losses.
- The child nodes of node x are always $2x$ and $2x+1$, to help in navigating the printout (compare the printout to figure 3).
- Other items in the list are the definition of the split used to create a node, the number of subjects at the node, the loss or error at the node (for this example, with proportional priors and unit losses this will be the number misclassified), and the predicted class for the node.
- * indicates that the node is terminal.
- Grades 1 and 2 go to the left, grades 3 and 4 go to the right. The tree is arranged so that the branches with the largest "average class" go to the right.

4 Pruning the tree

4.1 Definitions

We have built a complete tree, possibly quite large and/or complex, and must now decide how much of that model to retain. In stepwise regression, for instance, this issue is addressed sequentially and the fit is stopped when the F test fails to achieve some level α.

Let T_1, T_2,....,T_k be the terminal nodes of a tree T. Define

$|T|$ = number of terminal nodes
risk of $T = R(T) = \sum_{i=1}^{k} P(T_i)R(T_i)$

In comparison to regression, $|T|$ is analogous to the degrees of freedom and $R(T)$ to the residual sum of squares.

Now let α be some number between 0 and ∞ which measures the 'cost' of adding another variable to the model; α will be called a complexity parameter. Let $R(T_0)$ be the risk for the zero split tree. Define

$$R_\alpha(T) = R(T) + \alpha|T|$$

to be the cost for the tree, and define T_α to be that subtree of the full model which has minimal cost. Obviously T_0 = the full model and T_∞ = the model with no splits at all. The following results are shown in [1].

1. If T_1 and T_2 are subtrees of T with $R_\alpha(T_1) = R_\alpha(T_2)$, then either T_1 is a subtree of T_2 or T_2 is a subtree of T_1; hence either $|T_1| < |T_2|$ or $|T_2| < |T_1|$.

2. If $\alpha > \beta$ then either $T_\alpha = T_\beta$ or T_α is a strict subtree of T_β.

3. Given some set of numbers $\alpha_1, \alpha_2, \ldots, \alpha_m$; both $T_{\alpha_1}, \ldots, T_{\alpha_m}$ and $R(T_{\alpha_1}), \ldots, R(T_{\alpha_m})$ can be computed efficiently.

Using the first result, we can uniquely define T_α as the smallest tree T for which $R_\alpha(T)$ is minimized.

Since any sequence of nested trees based on T has at most $|T|$ members, result 2 implies that all possible values of α can be grouped into m intervals, $m \leq |T|$

$$\begin{aligned} I_1 &= [0, \alpha_1] \\ I_2 &= (\alpha_1, \alpha_2] \\ &\vdots \\ I_m &= (\alpha_{m-1}, \infty] \end{aligned}$$

where all $\alpha \in I_i$ share the same minimizing subtree.

4.2 Cross-validation

Cross-validation is used to choose a best value for α by the following steps:

1. Fit the full model on the data set
   compute $I_1, I_2, ..., I_m$
   set $\beta_1 = 0$
   $\beta_2 = \sqrt{\alpha_1 \alpha_2}$
   $\beta_3 = \sqrt{\alpha_2 \alpha_3}$
   $\vdots$
   $\beta_{m-1} = \sqrt{\alpha_{m-2} \alpha_{m-1}}$
   $\beta_m = \infty$
   each β_i is a 'typical value' for its I_i

2. Divide the data set into s groups $G_1, G_2, ..., G_s$ each of size s/n, and for each group separately:

   - fit a full model on the data set 'everyone except G_i' and determine $T_{\beta_1}, T_{\beta_2}, ..., T_{\beta_m}$ for this reduced data set,
   - compute the predicted class for each observation in G_i, under each of the models T_{β_j} for $1 \leq j \leq m$,

- from this compute the risk for each subject.

3. Sum over the G_i to get an estimate of risk for each β_j. For that β (complexity parameter) with smallest risk compute T_β for the full data set, this is chosen as the best trimmed tree.

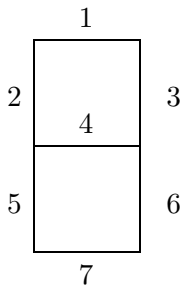
In actual practice, we may use instead the 1-SE rule. A plot of β versus risk often has an initial sharp drop followed by a relatively flat plateau and then a slow rise. The choice of β among those models on the plateau can be essentially random. To avoid this, both an estimate of the risk and its standard error are computed during the cross-validation. Any risk within one standard error of the achieved minimum is marked as being equivalent to the minimum (i.e. considered to be part of the flat plateau). Then the simplest model, among all those "tied" on the plateau, is chosen.

In the usual definition of cross-validation we would have taken $s = n$ above, i.e., each of the G_i would contain exactly one observation, but for moderate n this is computationally prohibitive. A value of $s = 10$ has been found to be sufficient, but users can vary this if they wish.

In Monte-Carlo trials, this method of pruning has proven very reliable for screening out 'pure noise' variables in the data set.

4.3 Example: The Stochastic Digit Recognition Problem

This example is found in section 2.6 of [1], and used as a running example throughout much of their book. Consider the segments of an unreliable digital readout



where each light is correct with probability 0.9, e.g., if the true digit is a 2, the lights 1, 3, 4, 5, and 7 are on with probability 0.9 and lights 2 and 6 are on with probability 0.1. Construct test data where $Y \in \{0, 1, ..., 9\}$, each with proportion 1/10 and the X_i, $i = 1, \ldots, 7$ are i.i.d. Bernoulli variables with parameter depending on Y.

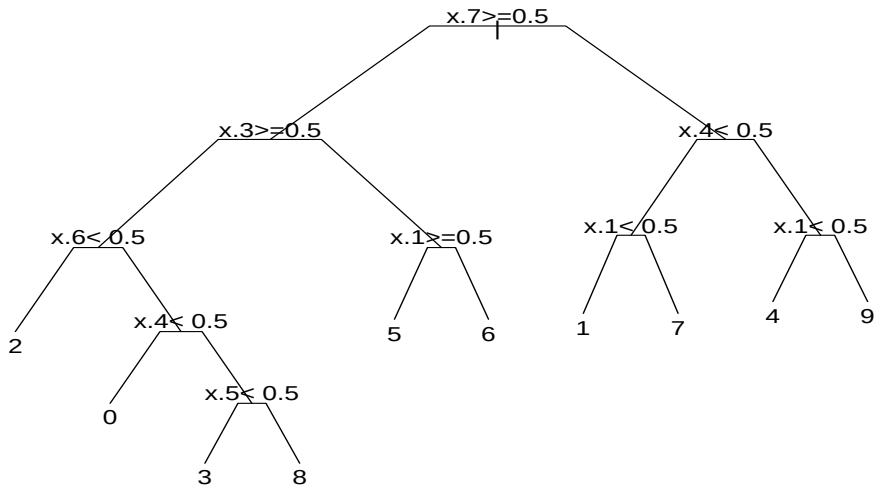


Figure 4: *Optimally pruned tree for the stochastic digit recognition data*

$X_8 - X_{24}$ are generated as i.i.d bernoulli $P\{X_i = 1\} = .5$, and are independent of Y. They correspond to imbedding the readout in a larger rectangle of random lights.

A sample of size 200 was generated accordingly and the procedure applied using the gini index (see 3.2.1) to build the tree. The S-plus code to compute the simulated data and the fit are shown below.

```
> n <- 200
> y <- rep(0:9, length=200)
> temp <- c(1,1,1,0,1,1,1,
            0,0,1,0,0,1,0,
            1,0,1,1,1,0,1,
            1,0,1,1,0,1,1,
            0,1,1,1,0,1,0,
            1,1,0,1,0,1,1,
            0,1,0,1,1,1,1,
            1,0,1,0,0,1,0,
            1,1,1,1,1,1,1,
            1,1,1,1,0,1,0)

> lights <- matrix(temp, 10, 7, byrow=T)    # The true light pattern 0-9
> temp1 <- matrix(rbinom(n*7, 1, .9), n, 7) # Noisy lights
> temp1 <- ifelse(lights[y+1, ]==1, temp1, 1-temp1)
```

```
> temp2 <- matrix(rbinom(n*17, 1, .5), n, 17) #Random lights
> x <- cbind(temp1, temp2)
```

The particular data set of this example can be replicated by setting `.Random.seed` to c(21, 14, 49, 32, 43, 1, 32, 22, 36, 23, 28, 3) before the call to rbinom. Now we fit the model:

```
> temp3 <- rpart.control(xval=10, minbucket=2, minsplit=4, cp=0)
> dfit <- rpart(y ~ x, method='class', control=temp3)
> printcp(dfit)

Classification tree:
rpart(formula = y ~ x, method = "class", control = temp3)

Variables actually used in tree construction:
 [1] x.1  x.10 x.12 x.13 x.15 x.19 x.2  x.20 x.22 x.3  x.4  x.5  x.6  x.7  x.8

Root node error: 180/200 = 0.9

          CP nsplit rel error  xerror      xstd
 1 0.1055556      0   1.00000 1.09444 0.0095501
 2 0.0888889      2   0.79444 1.01667 0.0219110
 3 0.0777778      3   0.70556 0.90556 0.0305075
 4 0.0666667      5   0.55556 0.75000 0.0367990
 5 0.0555556      8   0.36111 0.56111 0.0392817
 6 0.0166667      9   0.30556 0.36111 0.0367990
 7 0.0111111     11   0.27222 0.37778 0.0372181
 8 0.0083333     12   0.26111 0.36111 0.0367990
 9 0.0055556     16   0.22778 0.35556 0.0366498
10 0.0027778     27   0.16667 0.34444 0.0363369
11 0.0013889     31   0.15556 0.36667 0.0369434
12 0.0000000     35   0.15000 0.36667 0.0369434

> fit9 <- prune(dfit, cp=.02)
> plot(fit9, branch=.3, compress=T)
> text(fit9)
```

This table differs from that in section 3.5 of [1] in several ways, the last two of which are somewhat important.

- The actual values are different, of course, because of different random number generators in the two runs.
- The table is printed from the smallest tree (no splits) to the largest one (28

splits). We find it easier to compare one tree to another when they start at the same place.

- The number of splits is listed, rather than the number of nodes. The number of nodes is always 1 + the number of splits.
- For easier reading, the error columns have been scaled so that the first node has an error of 1. Since in this example the model with no splits must make 180/200 misclassifications, multiply columns 3-5 by 180 to get a result in terms of absolute error. (Computations are done on the absolute error scale, and printed on relative scale).
- The complexity parameter column has been similarly scaled.

Looking at the table, we see that the best tree has 10 terminal nodes (9 splits), based on cross-validation. This subtree is extracted with call to `prune` and saved in `fit9`. The pruned tree is shown in figure 4. Two options have been used in the plot. The `compress` option tries to narrow the printout by vertically overlapping portions of the plot. (It has only a small effect on this particular dendrogram). The `branch` option controls the shape of the branches that connect a node to its children. The section on plotting (9) will discuss this and other options in more detail.

The largest tree, with 35 terminal nodes, correctly classifies 170/200 = 85% of the observations, but uses several of the random predictors in doing so and seriously overfits the data. If the number of observations per terminal node (minbucket) had been set to 1 instead of 2, then every observation would be classified correctly in the final model, many in terminal nodes of size 1.

5 Missing data

5.1 Choosing the split

Missing values are one of the curses of statistical models and analysis. Most procedures deal with them by refusing to deal with them – incomplete observations are tossed out. `Rpart` is somewhat more ambitious. Any observation with values for the dependent variable and at least one independent variable will participate in the modeling.

The quantity to be maximized is still

$$\Delta I = p(A)I(A) - p(A_L)I(A_L) - p(A_R)I(A_R)$$

The leading term is the same for all variables and splits irrespective of missing data, but the right two terms are somewhat modified. Firstly, the impurity indices

$I(A_R)$ and $I(A_L)$ are calculated only over the observations which are not missing a particular predictor. Secondly, the two probabilities $p(A_L)$ and $p(A_R)$ are also calculated only over the relevant observations, but they are then adjusted so that they sum to $p(A)$. This entails some extra bookkeeping as the tree is built, but ensures that the terminal node probabilities sum to 1.

In the extreme case of a variable for which only 2 observations are non-missing, the impurity of the two sons will both be zero when splitting on that variable. Hence ΔI will be maximal, and this 'almost all missing' coordinate is guaranteed to be chosen as best; the method is certainly flawed in this extreme case. It is difficult to say whether this bias toward missing coordinates carries through to the non-extreme cases, however, since a more complete variable also affords for itself more possible values at which to split.

5.2 Surrogate variables

Once a splitting variable and a split point for it have been decided, what *is* to be done with observations missing that variable? One approach is to estimate the missing datum using the other independent variables; `rpart` uses a variation of this to define *surrogate* variables.

As an example, assume that the split (age <40, age ≥40) has been chosen. The surrogate variables are found by re-applying the partitioning algorithm (without recursion) to predict the two categories 'age <40' vs. 'age ≥40' using the other independent variables.

For each predictor an optimal split point and a misclassification error are computed. (Losses and priors do not enter in — none are defined for the age groups — so the risk is simply #misclassified / n.) Also evaluated is the blind rule 'go with the majority' which has misclassification error $\min(p, 1-p)$ where

$$p = (\#\text{ in A with age} < 40) \ / \ n_A.$$

The surrogates are ranked, and any variables which do no better than the blind rule are discarded from the list.

Assume that the majority of subjects have age ≤ 40 and that there is another variable x which is uncorrelated to age; however, the subject with the largest value of x is also over 40 years of age. Then the surrogate variable $x <$ max versus $x \geq$ max will have one less error that the blind rule, sending 1 subject to the right and $n-1$ to the left. A continuous variable that is completely unrelated to age has probability $1-p^2$ of generating such a trim-one-end surrogate by chance alone. For this reason the `rpart` routines impose one more constraint during the construction of the surrogates: a candidate split must send at least 2 observations to the left and at least 2 to the right.

Any observation which is missing the split variable is then classified using the first surrogate variable, or if missing that, the second surrogate is used, and etc. If an observation is missing all the surrogates the blind rule is used. Other strategies for these 'missing everything' observations can be convincingly argued, but there should be few or no observations of this type (we hope).

5.3 Example: Stage C prostate cancer (cont.)

Let us return to the stage C prostate cancer data of the earlier example. For a more detailed listing of the rpart object, we use the `summary` function. It includes the information from the CP table (not repeated below), plus information about each node. It is easy to print a subtree based on a different cp value using the `cp` option. Any value between 0.0555 and 0.1049 would produce the same result as is listed below, that is, the tree with 3 splits. Because the printout is long, the `file` option of `summary.rpart` is often useful.

```
> printcp(fit)

Classification tree:
rpart(formula = progstat ~ age + eet + g2 + grade + gleason + ploidy,
                data = stagec)

Variables actually used in tree construction:
[1] age    g2     grade  ploidy

Root node error: 54/146 = 0.36986

        CP nsplit rel error  xerror    xstd
1 0.104938      0   1.00000 1.0000 0.10802
2 0.055556      3   0.68519 1.1852 0.11103
3 0.027778      4   0.62963 1.0556 0.10916
4 0.018519      6   0.57407 1.0556 0.10916
5 0.010000      7   0.55556 1.0556 0.10916

> summary(cfit,cp=.06)

Node number 1: 146 observations,    complexity param=0.1049
  predicted class= No  expected loss= 0.3699
     class counts:  92 54
    probabilities:  0.6301 0.3699
  left son=2 (61 obs) right son=3 (85 obs)
  Primary splits:
      grade   < 2.5   to the left,  improve=10.360, (0 missing)
```

```
      gleason < 5.5   to the left,  improve= 8.400, (3 missing)
      ploidy  splits as  LRR, improve= 7.657, (0 missing)
      g2      < 13.2  to the left,  improve= 7.187, (7 missing)
      age     < 58.5  to the right, improve= 1.388, (0 missing)
  Surrogate splits:
      gleason < 5.5   to the left,  agree=0.8630, (0 split)
      ploidy  splits as  LRR, agree=0.6438, (0 split)
      g2      < 9.945 to the left,  agree=0.6301, (0 split)
      age     < 66.5  to the right, agree=0.5890, (0 split)

Node number 2: 61 observations
  predicted class= No  expected loss= 0.1475
     class counts:  52  9
    probabilities:  0.8525 0.1475

Node number 3: 85 observations,    complexity param=0.1049
  predicted class= Prog  expected loss= 0.4706
     class counts:  40 45
    probabilities:  0.4706 0.5294
  left son=6 (40 obs) right son=7 (45 obs)
  Primary splits:
      g2      < 13.2  to the left,  improve=2.1780, (6 missing)
      ploidy  splits as  LRR, improve=1.9830, (0 missing)
      age     < 56.5  to the right, improve=1.6600, (0 missing)
      gleason < 8.5   to the left,  improve=1.6390, (0 missing)
      eet     < 1.5   to the right, improve=0.1086, (1 missing)
  Surrogate splits:
      ploidy  splits as  LRL, agree=0.9620, (6 split)
      age     < 68.5  to the right, agree=0.6076, (0 split)
      gleason < 6.5   to the left,  agree=0.5823, (0 split)
   .
   .
   .
```

- There are 54 progressions (class 1) and 94 non-progressions, so the first node has an expected loss of $54/146 \approx 0.37$. (The computation is this simple only for the default priors and losses).
- Grades 1 and 2 go to the left, grades 3 and 4 to the right. The tree is arranged so that the "more severe" nodes go to the right.
- The improvement is n times the change in impurity index. In this instance, the largest improvement is for the variable `grade`, with an improvement of 10.36. The next best choice is Gleason score, with an improvement of 8.4.

The actual values of the improvement are not so important, but their relative size gives an indication of the comparative utility of the variables.

- Ploidy is a categorical variable, with values of diploid, tetraploid, and aneuploid, in that order. (To check the order, type `table(stagec$ploidy)`). All three possible splits were attempted: aneuploid+diploid vs. tetraploid, aneuploid+tetraploid vs. diploid, and aneuploid vs. diploid + tetraploid. The best split sends diploid to the right and the others to the left.

- The 2 by 2 table of diploid/non-diploid vs grade= 1-2/3-4 has 64% of the observations on the diagonal.

	1-2	3-4
diploid	38	29
tetraploid	22	46
aneuploid	1	10

- For node 3, the primary split variable is missing on 6 subjects. All 6 are split based on the first surrogate, ploidy. Diploid and aneuploid tumors are sent to the left, tetraploid to the right. As a surrogate, eet was no better than 45/85 (go with the majority), and was not retained.

	g2 < 13.2	g2 > 13.2
Diploid/aneuploid	71	1
Tetraploid	3	64

6 Further options

6.1 Program options

The central fitting function is `rpart`, whose main arguments are

- `formula`: the model formula, as in `lm` and other S model fitting functions. The right hand side may contain both continuous and categorical (factor) terms. If the outcome y has more than two levels, then categorical predictors must be fit by exhaustive enumeration, which can take a very long time.

- `data, weights, subset`: as for other S models. Weights are not yet supported, and will be ignored if present.

- `method`: the type of splitting rule to use. Options at this point are classification, anova, Poisson, and exponential.

- `parms`: a list of method specific optional parameters. For classification, the list can contain any of: the vector of prior probabilities (component `prior`), the loss matrix (component `loss`) or the splitting index (component `split`). The priors must be positive and sum to 1. The loss matrix must have zeros on the diagonal and positive off-diagonal elements. The splitting index can be 'gini' or 'information'.

- `na.action`: the action for missing values. The default action for rpart is `na.rpart`, this default is not overridden by the `options(na.action)` global option. The default action removes only those rows for which either the response y or *all* of the predictors are missing. This ability to retain partially missing observations is perhaps the single most useful feature of `rpart` models.

- `control`: a list of control parameters, usually the result of the `rpart.control` function. The list must contain

  - `minsplit`: The minimum number of observations in a node for which the routine will even try to compute a split. The default is 20. This parameter can save computation time, since smaller nodes are almost always pruned away by cross-validation.
  - `minbucket`: The minimum number of observations in a terminal node. This defaults to `minsplit/3`.
  - `maxcompete`: It is often useful in the printout to see not only the variable that gave the best split at a node, but also the second, third, etc best. This parameter controls the number that will be printed. It has no effect on computational time, and a small effect on the amount of memory used. The default is 4.
  - `xval`: The number of cross-validations to be done. Usually set to zero during exploratory phases of the analysis. A value of 10, for instance, increases the compute time to 11-fold over a value of 0.
  - `maxsurrogate`: The maximum number of surrogate variables to retain at each node. (No surrogate that does worse than "go with the majority" is printed or used). Setting this to zero will cut the computation time in half, and set `usesurrogate` to zero. The default is 5. Surrogates give different information than competitor splits. The competitor list asks "which other splits would have as many correct classifications", surrogates ask "which other splits would classify the same subjects in the same way", which is a harsher criteria.
  - `usesurrogate`: A value of usesurrogate=2, the default, splits subjects in the way described previously. This is similar to CART. If the value is 0,

then a subject who is missing the primary split variable does not progress further down the tree. A value of 1 is intermediate: all surrogate variables except "go with the majority" are used to send a case further down the tree.

- `cp`: The threshold complexity parameter.

The complexity parameter `cp` is, like `minsplit`, an advisory parameter, but is considerably more useful. It is specified according to the formula

$$R_{cp}(T) \equiv R(T) + cp * |T| * R(T_1)$$

where T_1 is the tree with no splits, $|T|$ is the number of splits for a tree, and R is the risk. This scaled version is much more user friendly than the original CART formula (4.1) since it is unitless. A value of `cp=1` will always result in a tree with no splits. For regression models (see next section) the scaled cp has a very direct interpretation: if any split does not increase the overall R^2 of the model by at least cp (where R^2 is the usual linear-models definition) then that split is decreed to be, a priori, not worth pursuing. The program does not split said branch any further, and saves considerable computational effort. The default value of .01 has been reasonably successful at 'pre-pruning' trees so that the cross-validation step need only remove 1 or 2 layers, but it sometimes overprunes, particularly for large data sets.

6.2 Example: Consumer Report Auto Data

A second example using the `class` method demonstrates the outcome for a response with multiple (> 2) categories. We also explore the difference between Gini and information splitting rules. The dataset cu.summary contains a collection of variables from the April, 1990 Consumer Reports summary on 117 cars. For our purposes, car Reliability will be treated as the response. The variables are:

Reliability	an ordered factor (contains NAs):
	Much worse < worse < average < better < Much Better
Price	numeric: list price in dollars, with standard equipment
Country	factor: country where car manufactured
Mileage	numeric: gas mileage in miles/gallon, contains NAs
Type	factor: Small, Sporty, Compact, Medium, Large, Van

In the analysis we are treating reliability as an unordered outcome. Nodes potentially can be classified as Much worse, worse, average, better, or Much better, though there are none that are labelled as just "better". The 32 cars with missing

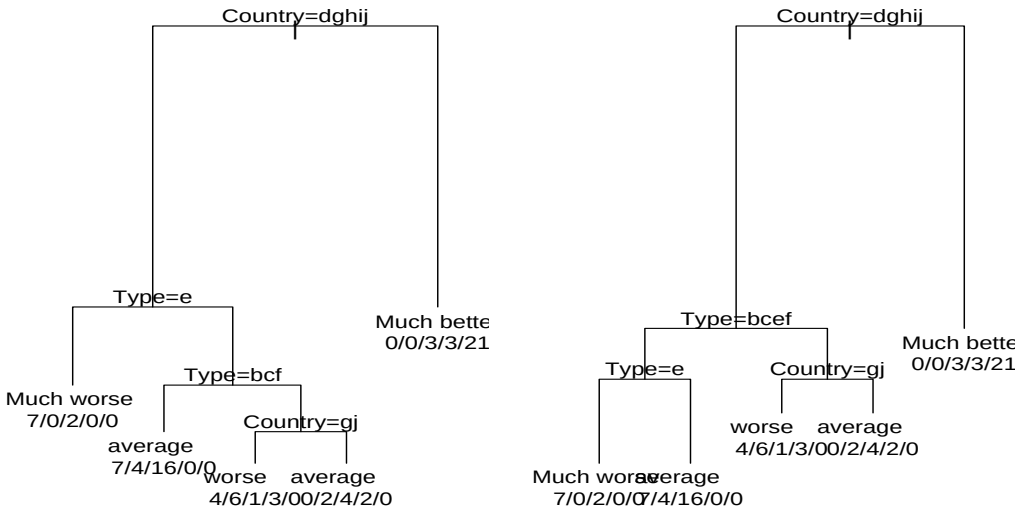


Figure 5: Displays the rpart-based model relating automobile Reliability to car type, price, and country of origin. The figure on the left uses the `gini` splitting index and the figure on the right uses the `information` splitting index.

response (listed as NA) were not used in the analysis. Two fits are made, one using the Gini index and the other the information index.

```
> fit1 <- rpart(Reliability ~ Price + Country + Mileage + Type,
                data=cu.summary, parms=list(split='gini'))
> fit2 <- rpart(Reliability ~ Price + Country + Mileage + Type,
                data=cu.summary, parms=list(split='information'))

> par(mfrow=c(1,2))
> plot(fit1);  text(fit1,use.n=T,cex=.9)
> plot(fit2);  text(fit2,use.n=T,cex=.9)
```

The first two nodes from the Gini tree are

```
Node number 1: 85 observations,    complexity param=0.3051
  predicted class= average  expected loss= 0.6941
     class counts:  18 12 26  8 21
    probabilities:  0.2118 0.1412 0.3059 0.0941 0.2471
  left son=2 (58 obs) right son=3 (27 obs)
  Primary splits:
```

```
      Country splits as  ---LRRLLLL, improve=15.220, (0 missing)
      Type    splits as  RLLRLL, improve= 4.288, (0 missing)
      Price   < 11970 to the right, improve= 3.200, (0 missing)
      Mileage < 24.5  to the left,  improve= 2.476, (36 missing)

Node number 2: 58 observations,    complexity param=0.08475
  predicted class= average  expected loss= 0.6034
     class counts:  18 12 23  5  0
    probabilities:  0.3103 0.2069 0.3966 0.0862 0.0000
  left son=4 (9 obs) right son=5 (49 obs)
  Primary splits:
      Type    splits as  RRRRLR, improve=3.187, (0 missing)
      Price   < 11230 to the left,  improve=2.564, (0 missing)
      Mileage < 24.5  to the left,  improve=1.802, (30 missing)
      Country splits as  ---L--RLRL, improve=1.329, (0 missing)
```

The fit for the information splitting rule is

```
Node number 1: 85 observations,    complexity param=0.3051
  predicted class= average  expected loss= 0.6941
     class counts:  18 12 26  8 21
    probabilities:  0.2118 0.1412 0.3059 0.0941 0.2471
  left son=2 (58 obs) right son=3 (27 obs)
  Primary splits:
      Country splits as  ---LRRLLLL, improve=38.540, (0 missing)
      Type    splits as  RLLRLL, improve=11.330, (0 missing)
      Price   < 11970 to the right, improve= 6.241, (0 missing)
      Mileage < 24.5  to the left,  improve= 5.548, (36 missing)

Node number 2: 58 observations,    complexity param=0.0678
  predicted class= average  expected loss= 0.6034
     class counts:  18 12 23  5  0
    probabilities:  0.3103 0.2069 0.3966 0.0862 0.0000
  left son=4 (36 obs) right son=5 (22 obs)
  Primary splits:
      Type    splits as  RLLRLL, improve=9.281, (0 missing)
      Price   < 11230 to the left,  improve=5.609, (0 missing)
      Mileage < 24.5  to the left,  improve=5.594, (30 missing)
      Country splits as  ---L--RRRL, improve=2.891, (0 missing)
  Surrogate splits:
      Price   < 10970 to the right, agree=0.8793, (0 split)
      Country splits as  ---R--RRRL, agree=0.7931, (0 split)
```

The first 3 countries (Brazil, England, France) had only one or two cars in the listing, all of which were missing the reliability variable. There are no entries

for these countries in the first node, leading to the − symbol for the rule. The information measure has larger "improvements", consistent with the difference in scaling between the information and Gini criteria shown in figure 2, but the relative merits of different splits are fairly stable.

The two rules do not choose the same primary split at node 2. The data at this point are

	Compact	Large	Medium	Small	Sporty	Van
Much worse	2	2	4	2	7	1
worse	5	0	4	3	0	0
average	3	5	8	2	2	3
better	2	0	0	3	0	0
Much better	0	0	0	0	0	0

Since there are 6 different categories, all $2^5 = 32$ different combinations were explored, and as it turns out there are several with a nearly identical improvement. The Gini and information criteria make different "random" choices from this set of near ties. For the Gini index, *Sporty vs others* and *Compact/Small vs others* have improvements of 3.19 and 3.12, respectively. For the information index, the improvements are 6.21 versus 9.28. Thus the Gini index picks the first rule and information the second. Interestingly, the two splitting criteria arrive at exactly the same final nodes, for the full tree, although by different paths. (Compare the class counts of the terminal nodes).

We have said that for a categorical predictor with m levels, all 2^{m-1} different possible splits are tested.. When there are a large number of categories for the predictor, the computational burden of evaluating all of these subsets can become large. For instance, the call `rpart(Reliability ~ ., data=car.all)` does not return for a *long,* long time: one of the predictors in that data set is a factor with 79 levels! Luckily, for any ordered outcome there is a computational shortcut that allows the program to find the best split using only $m - 1$ comparisons. This includes the classification method when there are only two categories, along with the anova and Poisson methods to be introduced later.

6.3 Example: Kyphosis data

A third `class` method example explores the parameters `prior` and `loss`. The dataset kyphosis has 81 rows representing data on 81 children who have had corrective spinal

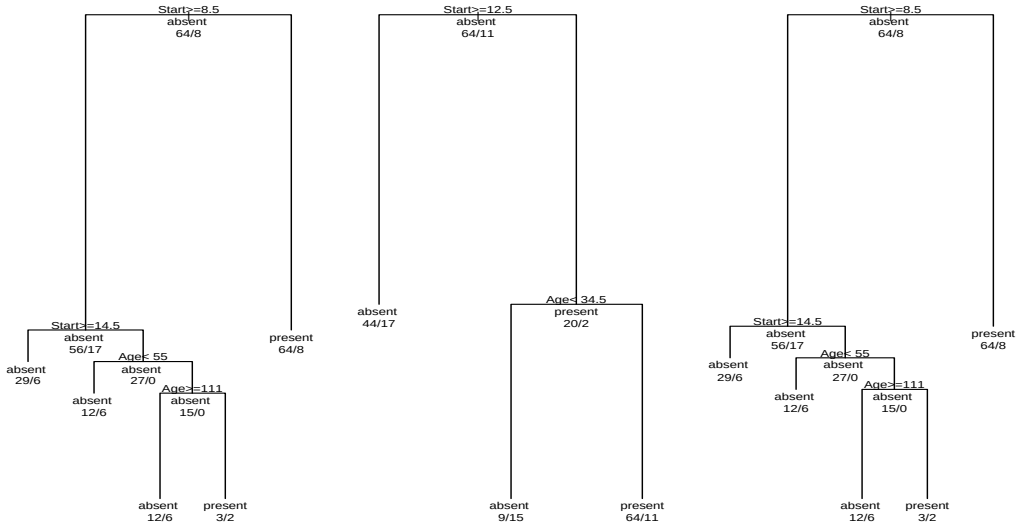


Figure 6: Displays the rpart-based models for the presence/absence of kyphosis. The figure on the left uses the default prior (0.79,0.21) and loss; the middle figure uses the user-defined prior (0.65,0.35) and default loss; and the third figure uses the default prior and the used-defined loss $L(1,2) = 3$, $L(2,1) = 2$).

surgery. The variables are:

Kyphosis	factor: postoperative deformity is present/absent
Age	numeric: age of child in months
Number	numeric: number of vertebrae involved in operation
Start	numeric: beginning of the range of vertebrae involved

```
> lmat <- matrix(c(0,4,3,0), nrow=2, ncol=2, byrow=F)
> fit1 <- rpart(Kyphosis ~  Age + Number + Start,data=kyphosis)

> fit2 <- rpart(Kyphosis ~  Age + Number + Start,data=kyphosis,
                parms=list(prior=c(.65,.35)))
> fit3 <- rpart(Kyphosis ~  Age + Number + Start,data=kyphosis,
                parms=list(loss=lmat))

> par(mfrow=c(1,3))
> plot(fit1);  text(fit1,use.n=T,all=T)
> plot(fit2);  text(fit2,use.n=T,all=T)
> plot(fit3);  text(fit3,use.n=T,all=T)
```

This example shows how even the initial split changes depending on the prior and loss that are specified. The first and third fits have the same initial split (Start < 8.5), but the improvement differs. The second fit splits Start at 12.5 which moves 46 people to the left instead of 62.

Looking at the leftmost tree, we see that the sequence of splits on the left hand branch yields only a single node classified as *present*. For any loss greater than 4 to 3, the routine will instead classify this node as *absent*, and the entire left side of the tree collapses, as seen in the right hand figure. This is not unusual — the most common effect of alternate loss matrices is to change the amount of pruning in the tree, more pruning in some branches and less in others, rather than to change the choice of splits.

The first node from the default tree is

```
Node number 1: 81 observations,    complexity param=0.1765
 predicted class= absent  expected loss= 0.2099
    class counts:  64 17
   probabilities:  0.7901 0.2099
 left son=2 (62 obs) right son=3 (19 obs)
 Primary splits:
     Start  < 8.5  to the right, improve=6.762, (0 missing)
     Number < 5.5  to the left,  improve=2.867, (0 missing)
     Age    < 39.5 to the left,  improve=2.250, (0 missing)
 Surrogate splits:
     Number < 6.5  to the left,  agree=0.8025, (0 split)
```

The fit using the prior (0.65,0.35) is

```
Node number 1: 81 observations,    complexity param=0.302
  predicted class= absent  expected loss= 0.35
     class counts:  64 17
    probabilities:  0.65 0.35
  left son=2 (46 obs) right son=3 (35 obs)
  Primary splits:
      Start  < 12.5 to the right, improve=10.900, (0 missing)
      Number < 4.5  to the left,  improve= 5.087, (0 missing)
      Age    < 39.5 to the left,  improve= 4.635, (0 missing)
  Surrogate splits:
      Number < 3.5  to the left,  agree=0.6667, (0 split)
```

And first split under 4/3 losses is

```
Node number 1: 81 observations,    complexity param=0.01961
  predicted class= absent  expected loss= 0.6296
     class counts:  64 17
    probabilities:  0.7901 0.2099
  left son=2 (62 obs) right son=3 (19 obs)
  Primary splits:
      Start  < 8.5  to the right, improve=5.077, (0 missing)
      Number < 5.5  to the left,  improve=2.165, (0 missing)
      Age    < 39.5 to the left,  improve=1.535, (0 missing)
  Surrogate splits:
      Number < 6.5  to the left,  agree=0.8025, (0 split)
```

7 Regression

7.1 Definition

Up to this point the classification problem has been used to define and motivate our formulae. However, the partitioning procedure is quite general and can be extended by specifying 5 "ingredients":

- A splitting criterion, which is used to decide which variable gives the best split. For classification this was either the Gini or log-likelihood function. In the `anova` method the splitting criteria is $SS_T - (SS_L + SS_R)$, where $SS_T = \sum(y_i - \bar{y})^2$ is the sum of squares for the node, and SS_R, SS_L are the sums of squares for the right and left son, respectively. This is equivalent to chosing the split to maximize the between-groups sum-of-squares in a simple analysis of variance. This rule is identical to the regression option for `tree`.

- A summary statistic or vector, which is used to describe a node. The first element of the vector is considered to be the fitted value. For the anova method this is the mean of the node; for classification the response is the predicted class followed by the vector of class probabilities.

- The error of a node. This will be the variance of y for anova, and the predicted loss for classification.

- The prediction error for a new observation, assigned to the node. For anova this is $(y_{new} - \bar{y})$.

- Any necessary initialization.

The `anova` method leads to regression trees; it is the default method if y a simple numeric vector, i.e., not a factor, matrix, or survival object.

7.2 Example: Consumer Report car data

The dataset `car.all` contains a collection of variables from the April, 1990 Consumer Reports; it has 36 variables on 111 cars. Documentation may be found in the S-Plus manual. We will work with a subset of 23 of the variables, created by the first two lines of the example below. We will use Price as the response. This data set is a good example of the usefulness of the missing value logic in rpart: most of the variables are missing on only 3-5 observations, but only 42/111 have a complete subset.

```
> cars <- car.all[, c(1:12, 15:17, 21, 28, 32:36)]
> cars$Eng.Rev <- as.numeric(as.character(car.all$Eng.Rev2))
> fit3 <- rpart(Price ~ ., data=cars)
> fit3
node), split, n, deviance, yval
      * denotes terminal node

 1) root 105 7118.00 15.810
   2) Disp.<156 70 1492.00 11.860
     4) Country:Brazil,Japan,Japan/USA,Korea,Mexico,USA 58  421.20 10.320
       8) Type:Small 21   50.31  7.629 *
       9) Type:Compact,Medium,Sporty,Van 37  132.80 11.840 *
     5) Country:France,Germany,Sweden 12  270.70 19.290 *
   3) Disp.>156 35 2351.00 23.700
     6) HP.revs<5550 24  980.30 20.390
      12) Disp.<267.5 16  396.00 17.820 *
      13) Disp.>267.5 8  267.60 25.530 *
     7) HP.revs>5550 11  531.60 30.940 *

> printcp(fit3)

Regression tree:
rpart(formula = Price ~ ., data = cars)

Variables actually used in tree construction:
[1] Country Disp.   HP.revs Type

Root node error: 7.1183e9/105 = 6.7793e7

        CP nsplit rel error  xerror    xstd
1 0.460146      0   1.00000 1.02413 0.16411
2 0.117905      1   0.53985 0.79225 0.11481
3 0.044491      3   0.30961 0.60042 0.10809
4 0.033449      4   0.26511 0.58892 0.10621
```

```
5 0.010000      5   0.23166 0.57062 0.11782
```

Only 4 of 22 predictors were actually used in the fit: engine displacement in cubic inches, country of origin, type of vehicle, and the revolutions for maximum horsepower (the "red line" on a tachometer).

- The relative error is $1 - R^2$, similar to linear regression. The xerror is related to the PRESS statistic. The first split appears to improve the fit the most. The last split adds little improvement to the apparent error, and increases the cross-validated error.
- The 1-SE rule would choose a tree with 3 splits.
- This is a case where the default cp value of .01 may have overpruned the tree, since the cross-validated error is barely at a minimum. A rerun with the cp threshold at .002 gave a maximum tree size of 8 splits, with a minimun cross-validated error for the 5 split model.
- For any CP value between 0.46015 and 0.11791 the best model has one split; for any CP value between 0.11791 and 0.04449 the best model is with 2 splits; and so on.

The `print` command also recognizes the `cp` option, which allows the user to see which splits are the most important.

```
> print(fit3,cp=.10)
node), split, n, deviance, yval
      * denotes terminal node

1) root 105 7.118e+09 15810
  2) Disp.<156 70 1.492e+09 11860
    4) Country:Brazil,Japan,Japan/USA,Korea,Mexico,USA 58 4.212e+08 10320 *
    5) Country:France,Germany,Sweden 12 2.707e+08 19290 *
  3) Disp.>156 35 2.351e+09 23700
    6) HP.revs<5550 24 9.803e+08 20390 *
    7) HP.revs>5550 11 5.316e+08 30940 *
```

The first split on displacement partitions the 105 observations into groups of 70 and 35 (nodes 2 and 3) with mean prices of 11,860 and 23,700. The deviance (corrected sum-of-squares) at these 2 nodes are 1.49×10^9 and 2.35×10^9, respectively. More detailed summarization of the splits is again obtained by using the function `summary.rpart`.

```
> summary(fit3, cp=.10)
Node number 1: 105 observations,    complexity param=0.4601
  mean=15810 , SS/n=67790000
  left son=2 (70 obs) right son=3 (35 obs)
  Primary splits:
      Disp.      < 156   to the left,  improve=0.4601, (0 missing)
      HP         < 154   to the left,  improve=0.4549, (0 missing)
      Tank       < 17.8  to the left,  improve=0.4431, (0 missing)
      Weight     < 2890  to the left,  improve=0.3912, (0 missing)
      Wheel.base < 104.5 to the left,  improve=0.3067, (0 missing)
  Surrogate splits:
      Weight     < 3095  to the left,  agree=0.9143, (0 split)
      HP         < 139   to the left,  agree=0.8952, (0 split)
      Tank       < 17.95 to the left,  agree=0.8952, (0 split)
      Wheel.base < 105.5 to the left,  agree=0.8571, (0 split)
      Length     < 185.5 to the left,  agree=0.8381, (0 split)

Node number 2: 70 observations,    complexity param=0.1123
  mean=11860 , SS/n=21310000
  left son=4 (58 obs) right son=5 (12 obs)
  Primary splits:
      Country splits as  L-RRLLLLRL, improve=0.5361, (0 missing)
      Tank    < 15.65 to the left,  improve=0.3805, (0 missing)
      Weight  < 2568  to the left,  improve=0.3691, (0 missing)
      Type    splits as  R-RLRR, improve=0.3650, (0 missing)
      HP      < 105.5 to the left,  improve=0.3578, (0 missing)
  Surrogate splits:
      Tank         < 17.8  to the left,  agree=0.8571, (0 split)
      Rear.Seating < 28.75 to the left,  agree=0.8429, (0 split)
        .
        .
        .
```

- The improvement listed is the percent change in SS for this split, i.e., $1 - (SS_{right} + SS_{left})/SS_{parent}$.
- The weight and displacement are very closely related, as shown by the surrogate split agreement of 91%.
- Not all types are represented in node 2, e.g., there are no representatives from England (the second category). This is indicated by a - in the list of split directions.

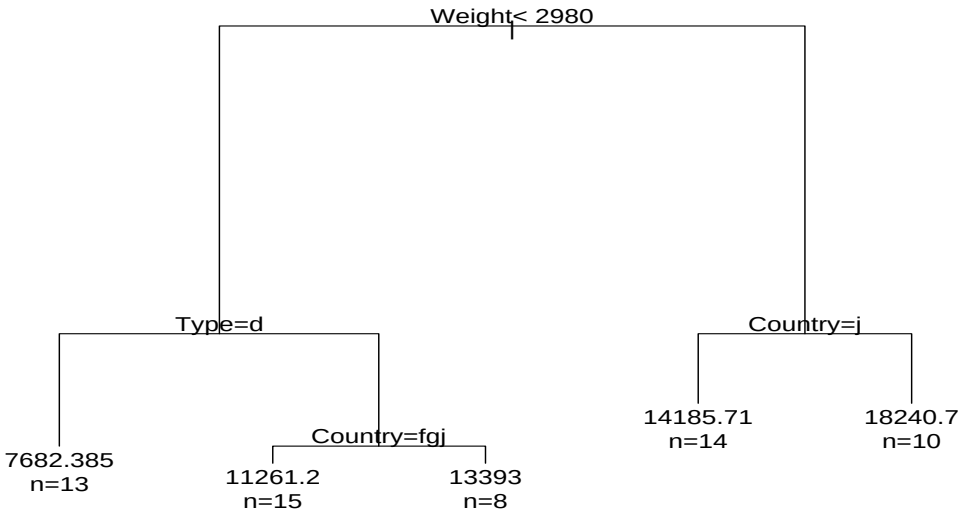


Figure 7: A `anova` tree for the car.test.frame dataset. The label of each node indicates the mean `Price` for the cars in that node.

```
> plot(fit3)
> text(fit3,use.n=T)
```

As always, a plot of the fit is useful for understanding the rpart object. In this plot, we use the option `use.n=T` to add the number of cars in each node. (The default is for only the mean of the response variable to appear). Each individual split is ordered to send the less expensive cars to the left.

Other plots can be used to help determine the best cp value for this model. The function `rsq.rpart` plots the jacknifed error versus the number of splits. Of interest is the smallest error, but any number of splits within the "error bars" (1-SE rule) are considered a reasonable number of splits (in this case, 1 or 2 splits seem to be sufficient). As is often true with modeling, simpler is often better. Another useful plot is the R^2 versus number of splits. The (1 - apparent error) and (1 - relative error) show how much is gained with additional splits. This plot highlights the differences between the R^2 values.

Finally, it is possible to look at the residuals from this model, just as with a regular linear regression fit, as shown in the following figure.

```
> plot(predict(fit3),resid(fit3))
> axis(3,at=fit3$frame$yval[fit3$frame$var=='<leaf>'],
```

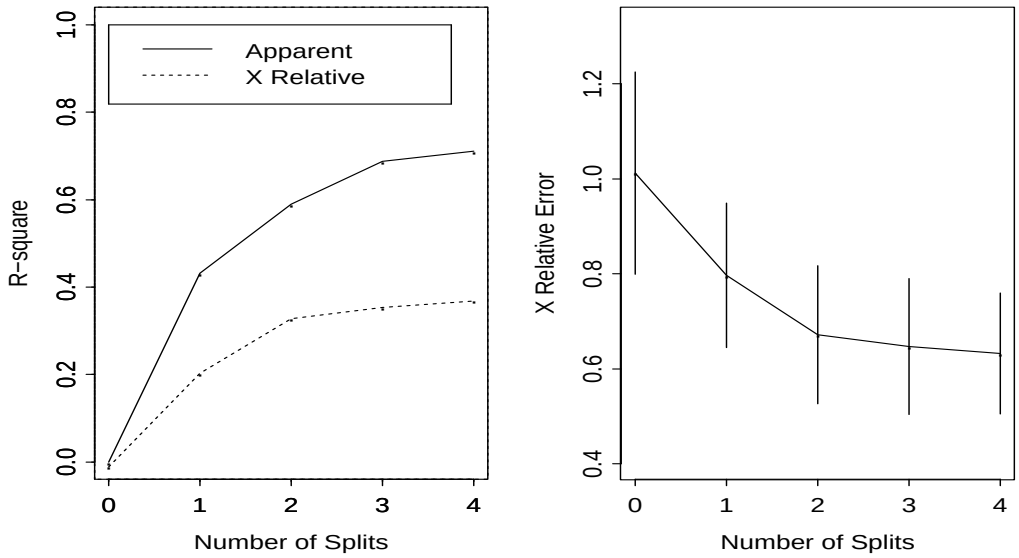


Figure 8: Both plots were obtained using the function `rsq.rpart(fit3)`. The figure on the left shows that the first split offers the most information. The figure on the right suggests that the tree should be pruned to include only 1 or 2 splits.

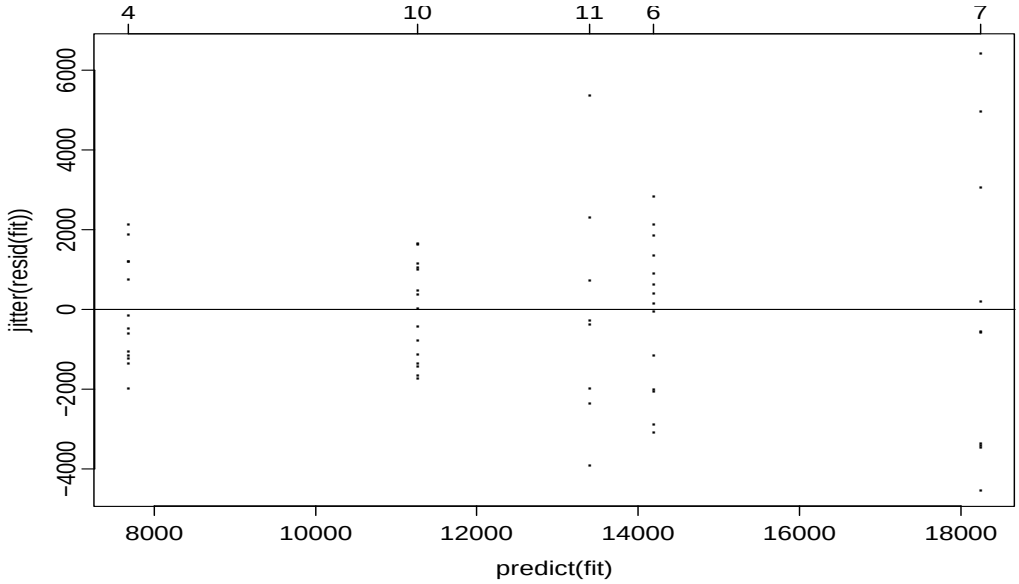


Figure 9: This plot shows the (observed-expected) cost of cars versus the predicted cost of cars based on the nodes/leaves in which the cars landed. There appears to be more variability in node 7 than in some of the other leaves.

```
       labels=row.names(fit3$frame)[fit3$frame$var=='<leaf>'])
> mtext('leaf number',side=3, line=3)
> abline(h=0)
```

7.3 Example: Stage C data (anova method)

The stage C prostate cancer data of the earlier section can also be fit using the anova method, by treating the status variable as though it were continuous.

```
> cfit2 <- rpart(pgstat ~ age + eet + g2 + grade + gleason + ploidy,
                 data=stagec)

> printcp(cfit2)
Regression tree:
rpart(formula = pgstat ~ age + eet + g2 + grade + gleason + ploidy, data =
        stagec)

Variables actually used in tree construction:
[1] age    g2     grade  ploidy

Root node error: 34.027/146 = 0.23306

        CP nsplit rel error  xerror     xstd
1 0.152195      0   1.00000 1.01527 0.045470
2 0.054395      1   0.84781 0.86670 0.063447
3 0.032487      3   0.73901 0.86524 0.075460
4 0.019932      4   0.70653 0.95702 0.085390
5 0.013027      8   0.63144 1.05606 0.092566
6 0.010000      9   0.61841 1.07727 0.094466

> print(cfit2, cp=.03)
node), split, n, deviance, yval
      * denotes terminal node

 1) root 146 34.030 0.3699
   2) grade<2.5 61  7.672 0.1475
     4) g2<13.19 40  1.900 0.0500 *
     5) g2>13.19 21  4.667 0.3333 *
   3) grade>2.5 85 21.180 0.5294
     6) g2<13.2 40  9.775 0.4250 *
     7) g2>13.2 45 10.580 0.6222
      14) g2>17.91 22  5.091 0.3636 *
      15) g2<17.91 23  2.609 0.8696 *
```

If this tree is compared to the earlier results, we see that it has chosen exactly the same variables and split points as before. The only addition is further splitting of node 2, the upper left "No" of figure 3. This is no accident, for the two class case the Gini splitting rule reduces to $2p(1-p)$, which is the variance of a node.

The two methods differ in their evaluation and pruning, however. Note that nodes 4 and 5, the two children of node 2, contain 2/40 and 7/21 progressions, respectively. For classification purposes both nodes have the same predicted value (No) and the split will be discarded since the error (# of misclassifications) with and without the split is identical. In the regression context the two predicted values of .05 and .33 *are* different — the split has identified a nearly pure subgroup of significant size.

This setup is known as *odds regression*, and may be a more sensible way to evaluate a split when the emphasis of the model is on understanding/explanation rather than on prediction error per se. Extension of this rule to the multiple class problem is appealing, but has not yet been implemented in `rpart`.

8 Poisson regression

8.1 Definition

The Poisson splitting method attempts to extend `rpart` models to event rate data. The model in this case is

$$\lambda = f(x)$$

where λ is an event rate and x is some set of predictors. As an example consider hip fracture rates. For each county in the United States we can obtain

- number of fractures in patients age 65 or greater (from Medicare files)
- population of the county (US census data)
- potential predictors such as
  - socio-economic indicators
  - number of days below freezing
  - ethnic mix
  - physicians/1000 population
  - etc.

Such data would usually be approached by using Poisson regression; can we find a tree based analogue? In adding criteria for rates regression to this ensemble, the

guiding principle was the following: the between groups sum-of-squares is not a very robust measure, yet tree based regression works very well. So do the simplest thing possible.

Let c_i be the observed event count for observation i, t_i be the observation time, and $x_{ij}, j = 1, \ldots, p$ be the predictors. The y variable for the program will be a 2 column matrix.

Splitting criterion: The likelihood ratio test for two Poisson groups

$$D_{\text{parent}} - \left(D_{\text{left son}} + D_{\text{right son}}\right)$$

Summary statistics: The observed event rate and the number of events.

$$\hat{\lambda} \quad = \quad \frac{\#\text{ events}}{\text{total time}} = \frac{\sum c_i}{\sum t_i}$$

Error of a node: The within node deviance.

$$D = \sum \left[c_i \log\left(\frac{c_i}{\hat{\lambda} t_i}\right) - (c_i - \hat{\lambda} t_i) \right]$$

Prediction error: The deviance contribution for a new observation, using $\hat{\lambda}$ of the node as the predicted rate.

8.2 Improving the method

There is a problem with the criterion just proposed, however: cross-validation of a model often produces an infinite value for the deviance. The simplest case where this occurs is easy to understand. Assume that some terminal node of the tree has 20 subjects, but only 1 of the 20 has experienced any events. The cross-validated error (deviance) estimate for that node will have one subset — the one where the subject with an event is left out — which has $\hat{\lambda} = 0$. When we use the prediction for the 10% of subjects who were set aside, the deviance contribution of the subject with an event is

$$\ldots + c_i \log(c_i/0) + \ldots$$

which is infinite since $c_i > 0$. The problem is that when $\hat{\lambda} = 0$ the occurrence of an event is infinitely improbable, and, using the deviance measure, the corresponding model is then infinitely bad.

One might expect this phenomenon to be fairly rare, but unfortunately it is not so. One given of tree-based modeling is that a right-sized model is arrived at by purposely over-fitting the data and then pruning back the branches. A program

that aborts due to a numeric exception during the first stage is uninformative to say the least. Of more concern is that this edge effect does not seem to be limited to the pathologic case detailed above. Any near approach to the boundary value $\lambda = 0$ leads to large values of the deviance, and the procedure tends to discourage any final node with a small number of events.

An ad hoc solution is to use the revised estimate

$$\hat{\hat{\lambda}} = \max\left(\hat{\lambda}, \frac{k}{\sum t_i}\right)$$

where k is 1/2 or 1/6. That is, pure nodes are given a partial event. (This is similar to the starting estimates used in the GLM program for a Poisson regression.) This is unsatisfying, however, and we propose instead using a shrinkage estimate.

Assume that the true rates λ_j for the leaves of the tree are random values from a Gamma(μ, σ) distribution. Set μ to the observed overall event rate $\sum c_i / \sum t_i$, and let the user choose as a prior the coefficient of variation $k = \sigma/\mu$. A value of $k = 0$ represents extreme pessimism ("the leaf nodes will all give the same result"), whereas $k = \infty$ represents extreme optimism. The Bayes estimate of the event rate for a node works out to be

$$\hat{\lambda}_k = \frac{\alpha + \sum c_i}{\beta + \sum t_i},$$

where $\alpha = 1/k^2$ and $\beta = \alpha/\hat{\lambda}$.

This estimate is scale invariant, has a simple interpretation, and shrinks least those nodes with a large amount of information. In practice, a value of $k = 10$ does essentially no shrinkage. For `method='poisson'`, the optional parameters list is the single number k, with a default value of 1.

8.3 Example: solder data

The solder data frame, as explained in the Splus help file, is a dataset with 900 observations which are the results of an experiment varying 5 factors relevant to the wave-soldering procedure for mounting components on printed circuit boards. The response variable, skips, is a count of how many solder skips appeared to a visual inspection. The other variables are listed below:

Opening	factor:	amount of clearance around the mounting pad (S < M < L)
Solder	factor:	amount of solder used (Thin < Thick)
Mask	factor:	Type of solder mask used (5 possible)
PadType	factor:	Mounting pad used (10 possible)
Panel	factor:	panel (1, 2 or 3) on board being counted

In this call, the `rpart.control` options are modified: `maxcompete = 2` means that only 2 other competing splits are listed (default is 4); `cp = .05` means that a smaller tree will be built initially (default is .01). The y variable for Poisson partitioning may be a two column matrix containing the observation time in column 1 and the number of events in column 2, or it may be a vector of event counts alone.

```
fit <- rpart(skips ~ Opening + Solder + Mask + PadType
             + Panel, data=solder, method='poisson',
             control=rpart.control(cp=.05, maxcompete=2))
```

The `print` command summarizes the tree, as in the previous examples.

```
node), split, n, deviance, yval
      * denotes terminal node

1) root 900 8788.0  5.530
  2) Opening:M,L 600 2957.0  2.553
    4) Mask:A1.5,A3,B3 420  874.4  1.033 *
    5) Mask:A6,B6 180  953.1  6.099 *
  3) Opening:S 300 3162.0 11.480
    6) Mask:A1.5,A3 150  652.6  4.535 *
    7) Mask:A6,B3,B6 150 1155.0 18.420 *
```

- The response value is the expected event rate (with a time variable), or in this case the expected number of skips. The values are shrunk towards the global estimate of 5.53 skips/observation.
- The deviance is the same as the null deviance (sometimes called the residual deviance) that you'd get when calculating a Poisson glm model for the given subset of data.

```
> summary(fit,cp=.10)
Call:
rpart(formula = skips ~ Opening + Solder + Mask + PadType + Panel, data =
        solder, method = "poisson", control = rpart.control(cp = 0.05,
        maxcompete = 2))

      CP nsplit rel error xerror    xstd
1 0.3038      0    1.0000 1.0030 0.05228
2 0.1541      1    0.6962 0.7005 0.03282
3 0.1285      2    0.5421 0.5775 0.02746
```

```
4 0.0500      3    0.4137 0.4200 0.01976

Node number 1: 900 observations,    complexity param=0.3038
  events=4977,  estimated rate=5.53 , deviance/n=9.765
  left son=2 (600 obs) right son=3 (300 obs)
  Primary splits:
      Opening splits as  RLL, improve=2670, (0 missing)
      Mask    splits as  LLRLR, improve=2162, (0 missing)
      Solder  splits as  RL, improve=1168, (0 missing)

Node number 2: 600 observations,    complexity param=0.1285
  events=1531,  estimated rate=2.553 , deviance/n=4.928
  left son=4 (420 obs) right son=5 (180 obs)
  Primary splits:
      Mask    splits as  LLRLR, improve=1129.0, (0 missing)
      Opening splits as  -RL, improve= 250.8, (0 missing)
      Solder  splits as  RL, improve= 219.8, (0 missing)

Node number 3: 300 observations,    complexity param=0.1541
  events=3446,  estimated rate=11.48 , deviance/n=10.54
  left son=6 (150 obs) right son=7 (150 obs)
  Primary splits:
      Mask    splits as  LLRRR, improve=1354.0, (0 missing)
      Solder  splits as  RL, improve= 976.9, (0 missing)
      PadType splits as  RRRRLRLRLL, improve= 313.2, (0 missing)
  Surrogate splits:
      Solder splits as  RL, agree=0.6, (0 split)

Node number 4: 420 observations
  events=433,  estimated rate=1.033 , deviance/n=2.082

Node number 5: 180 observations
  events=1098,  estimated rate=6.099 , deviance/n=5.295

Node number 6: 150 observations
  events=680,  estimated rate=4.535 , deviance/n=4.351

Node number 7: 150 observations
  events=2766,  estimated rate=18.42 , deviance/n=7.701
```

- The improvement is $\text{Deviance}_{\text{parent}} - (\text{Deviance}_{\text{left}} + \text{Deviance}_{\text{right}})$, which is the likelihood ratio test for comparing two Poisson samples.
- The cross-validated error has been found to be overly pessimistic when describing how much the error is improved by each split. This is likely an effect

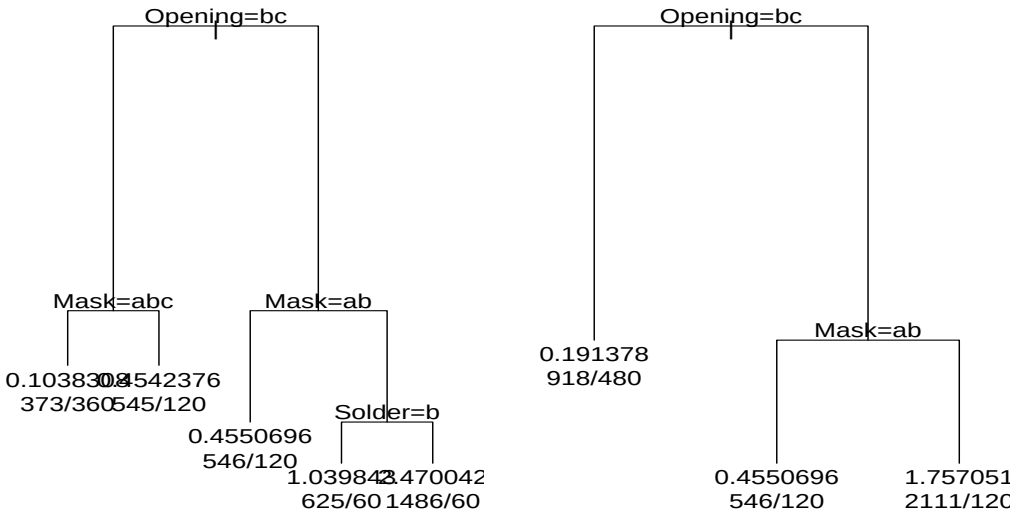


Figure 10: The first figure shows the solder data, fit with the `poisson` method, using a `cp` value of 0.05. The second figure shows the same fit, but with a `cp` value of 0.10. The function `prune.rpart` was used to produce the smaller tree.

of the boundary effect mentioned earlier, but more research is needed.

- The variation xstd is not as useful, given the bias of xerror.

```
> plot(fit)
> text(fit,use.n=T)

> fit.prune <- prune(fit,cp=.10)
> plot(fit.prune)
> text(fit.prune,use.n=T)
```

The `use.n=T` option specifies that number of events / total N should be listed along with the predicted rate (number of events/person-years). The function `prune` trims the tree `fit` to the `cp` value 0.10. The same tree could have been created by specifying `cp = .10` in the original call to `rpart`.

8.4 Example: Stage C Prostate cancer, survival method

One special case of the Poisson model is of particular interest for medical consulting (such as the authors do). Assume that we have survival data, i.e., each subject has

either 0 or 1 event. Further, assume that the time values have been pre-scaled so as to fit an exponential model. That is, stretch the time axis so that a Kaplan-Meier plot of the data will be a straight line when plotted on the logarithmic scale. An approximate way to do this is

```
temp <- coxph(Surv(time, status) ~1)
newtime <- predict(temp, type='expected')
```

and then do the analysis using the `newtime` variable. (This replaces each time value by $\Lambda(t)$, where Λ is the cumulative hazard function).

A slightly more sophisticated version of this which we will call *exponential scaling* gives a straight line curve for log(survival) under a parametric exponential model. The only difference from the approximate scaling above is that a subject who is censored between observed death times will receive "credit" for the intervening interval, i.e., we assume the baseline hazard to be linear between observed deaths. If the data is pre-scaled in this way, then the Poisson model above is equivalent to the *local full likelihood* tree model of LeBlanc and Crowley [4]. They show that this model is more efficient than the earlier suggestion of Therneau et. al. [7] to use the martingale residuals from a Cox model as input to a regression tree (anova method). Exponential scaling or method='exp' is the default if y is a `Surv` object.

Let us again return to the stage C cancer example. Besides the variables explained previously, we will use pgtime, which is time to tumor progression.

```
> fit <- rpart(Surv(pgtime, pgstat) ~ age + eet + g2 + grade +
                gleason + ploidy, data=stagec)
> print(fit)

node), split, n, deviance, yval
      * denotes terminal node

 1) root 146 195.30 1.0000
   2) grade<2.5 61  44.98 0.3617
     4) g2<11.36 33   9.13 0.1220 *
     5) g2>11.36 28  27.70 0.7341 *
   3) grade>2.5 85 125.10 1.6230
     6) age>56.5 75 104.00 1.4320
      12) gleason<7.5 50  66.49 1.1490
        24) g2<13.475 25  29.10 0.8817 *
        25) g2>13.475 25  36.05 1.4080
          50) g2>17.915 14  18.72 0.8795 *
          51) g2<17.915 11  13.70 2.1830 *
      13) gleason>7.5 25  34.13 2.0280
```

```
        26) g2>15.29 10  11.81 1.2140 *
        27) g2<15.29 15  19.36 2.7020 *
     7) age<56.5 10  15.52 3.1980 *

> plot(fit, uniform=T, branch=.4, compress=T)
> text(fit, use.n=T)
```

Note that the primary split on grade is the same as when status was used as a dichotomous endpoint, but that the splits thereafter differ.

```
> summary(fit,cp=.02)

Call:
rpart(formula = Surv(pgtime, pgstat) ~ age + eet + g2 + grade +
        gleason + ploidy, data = stagec)

       CP nsplit rel error xerror    xstd
1 0.12913      0    1.0000 1.0216 0.07501
2 0.04169      1    0.8709 0.9543 0.09139
3 0.02880      2    0.8292 0.9518 0.09383
4 0.01720      3    0.8004 0.9698 0.09529
5 0.01518      4    0.7832 0.9787 0.09351
6 0.01271      5    0.7680 0.9865 0.09340
7 0.01000      7    0.7426 0.9859 0.09412

Node number 1: 146 observations,    complexity param=0.1291
  events=54,  estimated rate=1 , deviance/n=1.338
  left son=2 (61 obs) right son=3 (85 obs)
  Primary splits:
      grade   < 2.5   to the left,  improve=25.270, (0 missing)
      gleason < 5.5   to the left,  improve=21.630, (3 missing)
      ploidy  splits as  LRR, improve=14.020, (0 missing)
      g2      < 13.2  to the left,  improve=12.580, (7 missing)
      age     < 58.5  to the right, improve= 2.796, (0 missing)
  Surrogate splits:
      gleason < 5.5   to the left,  agree=0.8630, (0 split)
      ploidy  splits as  LRR, agree=0.6438, (0 split)
      g2      < 9.945 to the left,  agree=0.6301, (0 split)
      age     < 66.5  to the right, agree=0.5890, (0 split)
  .
  .
  .
```

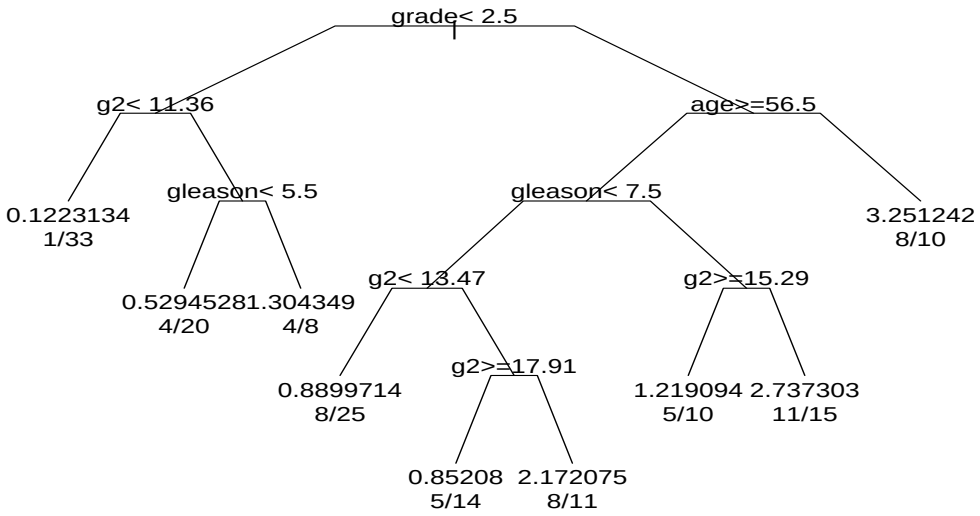


Figure 11: The prostate cancer data as a survival tree

Suppose that we wish to simplify this tree, so that only four terminal nodes remain. Looking at the table of complexity parameters, we see that `prune(fit, cp=.015)` would give the desired result. It is also possible to trim the figure interactively using `snip.rpart`. Point the mouse on a node and click with the left button to get some simple descriptives of the node. Double-clicking with the left button will 'remove' the sub-tree below, or one may click on another node. Multiple branches may be snipped off one by one; clicking with the right button will end interactive mode and return the pruned tree.

```
> plot(fit)
> text(fit,use.n=T)
> fit2 <- snip.rpart(fit)

node number: 6  n= 75
    response= 1.432013 ( 37 )
    Error (dev) =  103.9783

> plot(fit2)
> text(fit2,use.n=T)
```

For a final summary of the model, it can be helpful to plot the probability of survival based on the final bins in which the subjects landed. To create new variables

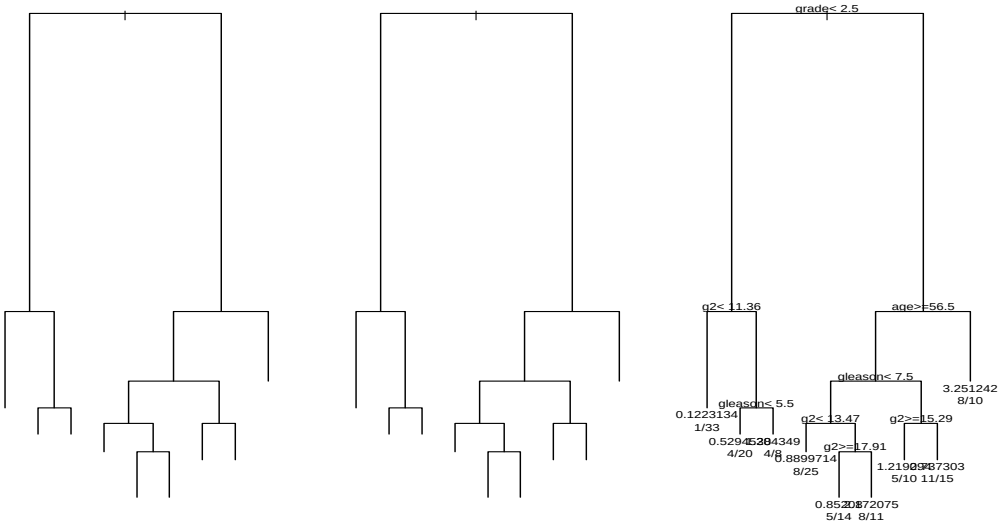


Figure 12: An illustration of how `snip.rpart` works. The full tree is plotted in the first panel. After selecting node 6 with the mouse (double clicking on left button), the subtree disappears from the plot (shown in the second panel). Finally, the new subtree is redrawn to use all available space and it is labelled.

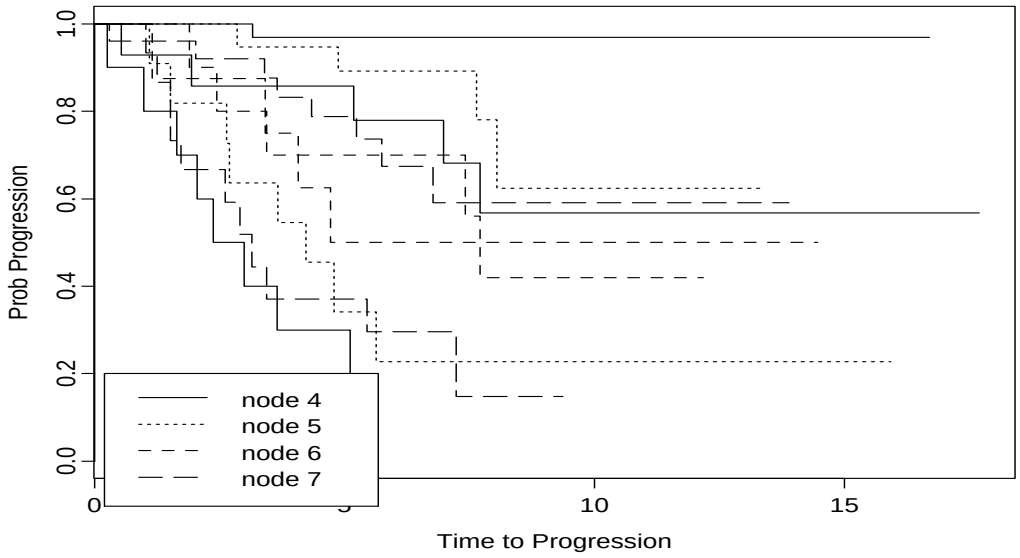


Figure 13: Survival plot based on snipped rpart object. The probability of tumor progression is greatest in node 8, which has patients who are older and have a higher initial tumor grade.

based on the rpart groupings, use `where`. The nodes of `fit2` above are shown in the right hand panel of figure 12: node 4 has 1 event, 33 subjects, grade = 1-2 and g2 $<$ 11.36; node 5 has 8 events, 28 subjects, grade = 1-2 and g2 $>$ 11.36; node 7 has 37 events, 75 subjects, grade = 3-4, age $>$ 56.5; node 8 has 8 events, 10 subjects, grade = 3-4, age $<$ 56.5. Patients who are older and have a higher initial grade tend to have more rapid progression of disease.

```
> newgrp <- fit2$where
> plot(survfit(Surv(pgtime,pgstat) ~ newgrp, data=stagec),
       mark.time=F, lty=1:4)
> title(xlab='Time to Progression',ylab='Prob Progression')
> legend(.2,.2, legend=paste('node',c(4,5,6,7)), lty=1:4)
```

8.5 Open issues

The default value of the shrinkage parameter k is 1. This corresponds to prior coefficient of variation of 1 for the estimated λ_j. We have not nearly enough experience to decide if this is a good value. (It does stop the log(0) message though).

Cross-validation does not work very well. The procedure gives very conservative results, and quite often declares the no-split tree to be the best. This may be another

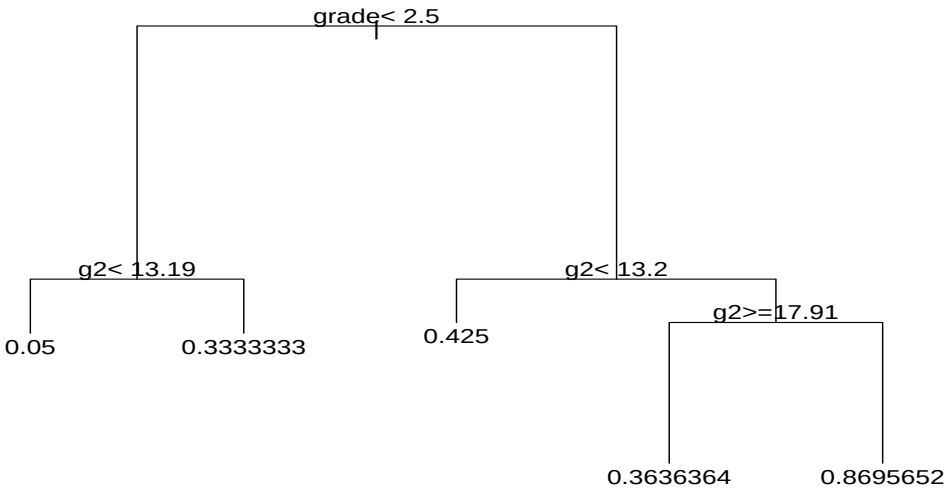


Figure 14: `plot(fit); text(fit)`

artifact of the edge effect.

9 Plotting options

This section examines the various options that are available when plotting an rpart object. For simplicity, the same model (data from Example 1) will be used throughout.

The simpliest labelled plot is called by using `plot` and `text` without changing any of the defaults. This is useful for a first look, but sometimes you'll want more information about each of the nodes.

```
> fit <- rpart(progstat ~  age + eet + g2 + grade + gleason + ploidy,
                stagec, control=rpart.control(cp=.025))

> plot(fit)
> text(fit)
```

The next plot has uniform stem lengths (`uniform=T`), has extra information (`use.n=T`) specifying number of subjects at each node, and has labels on all the nodes, not just the terminal nodes (`all=T`).

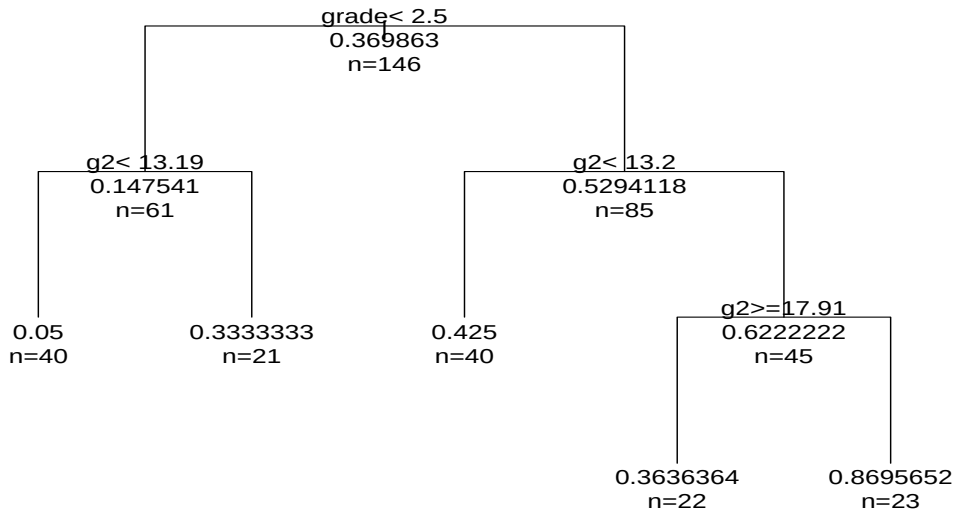


Figure 15: `plot(fit, uniform=T); text(fit,use.n=T,all=T)`

```
> plot(fit, uniform=T)
> text(fit, use.n=T, all=T)
```

Fancier plots can be created by modifying the `branch` option, which controls the shape of branches that connect a node to its children. The default for the plots is to have square shouldered trees (`branch = 1.0`). This can be taken to the other extreme with no shoulders at all (`branch=0`).

```
> plot(fit, branch=0)
> text(fit, use.n=T)
```

These options can be combined with others to create the plot that fits your particular needs. The default plot may be inefficient in it's use of space: the terminal nodes will always lie at x-coordinates of 1,2,.... The `compress` option attempts to improve this by overlapping some nodes. It has little effect on figure 17, but in figure 4 it allows the lowest branch to "tuck under" those above. If you want to play around with the spacing with compress, try using `nspace` which regulates the space between a terminal node and a split.

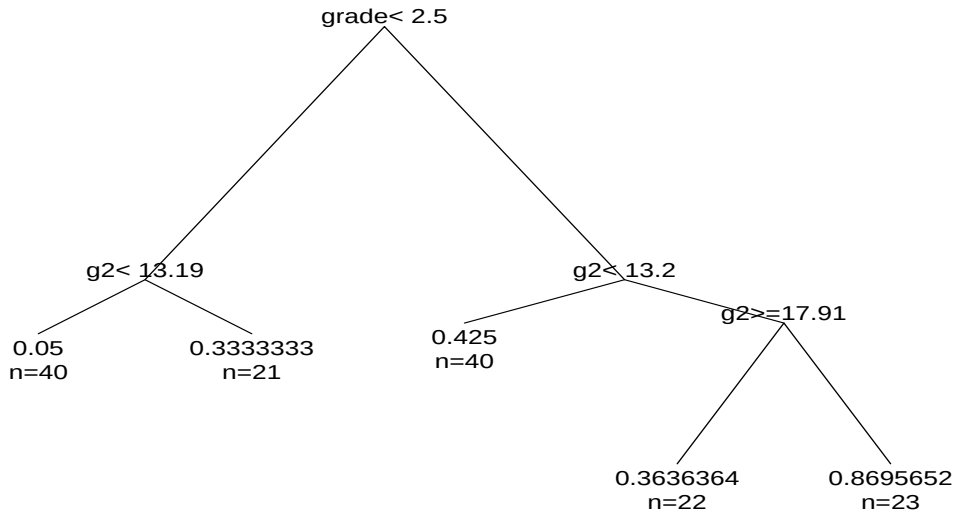


Figure 16: `plot(fit, branch=0); text(fit,use.n=T)`

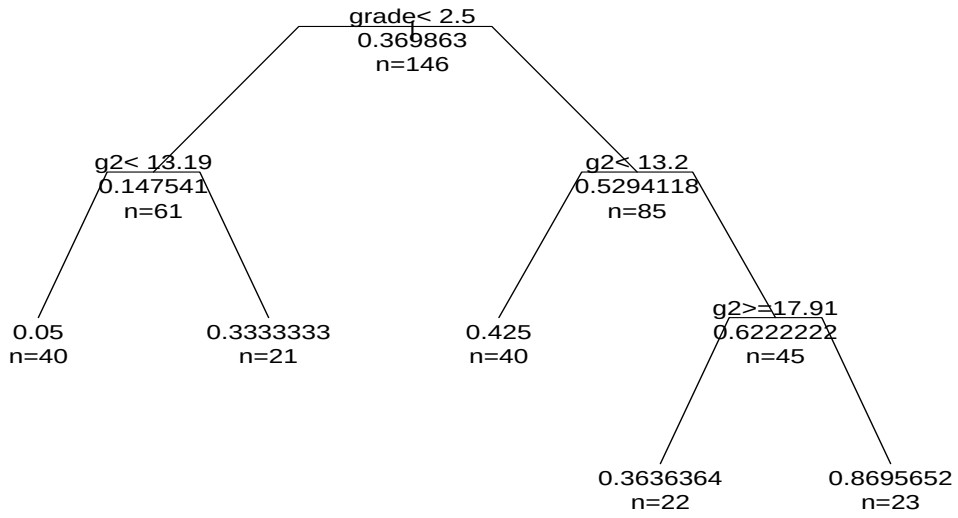


Figure 17: `plot(fit, branch=.4, uniform=T,compress=T)`

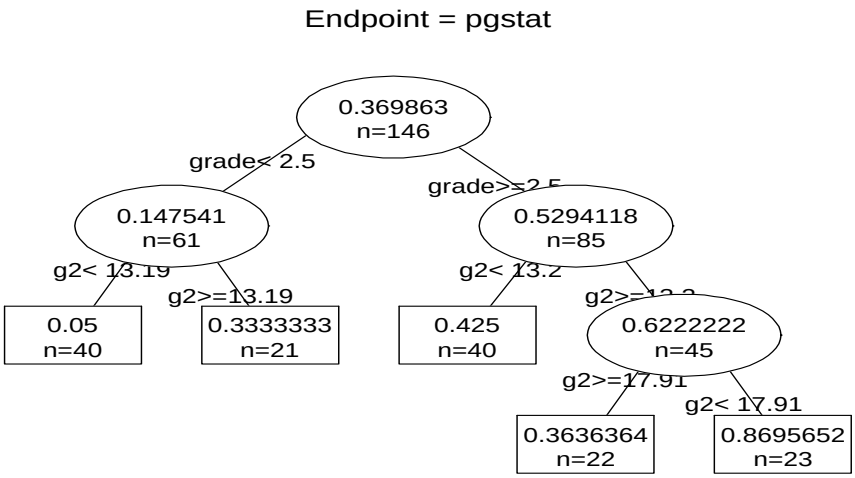


Figure 18: `post(fit)`

```
> plot(fit,branch=.4,uniform=T,compress=T)
> text(fit,all=T,use.n=T)
```

We have combined several of these options into a function called `post.rpart`. Results are shown in figure 18. The code is essentially

```
> plot(tree, uniform = T, branch = 0.2, compress = T, margin = 0.1)
> text(tree, all = T, use.n=T, fancy = T)
```

The `fancy` option of `text` creates the ellipses and rectangles, and moves the splitting rule to the midpoints of the branches. `Margin` shrinks the plotting region slightly so that the `text` boxes don't run over the edge of the plot. The `branch` option makes the lines exit the ellipse at a "good" angle. The call `post(fit)` will create a postcript file *fit.ps* in the current directory. The additional argument `file=''` will cause the plot to appear on the current active device. Note that `post.rpart` is just our choice of options to the functions `plot.rpart` and `text.rpart`. The reader is encouraged to try out other possibilities, as these specifications may not be the best choice in all situations.

10 Other functions

A more general approach to cross-validation can be gained using the `xpred.rpart` function. Given an rpart fit, a vector of k complexity parameters, and the desired number of cross-validation groups, this function returns an n by k matrix containing the predicted value $\hat{y}_{(-i)}$ for each subject, from the model that was fit without that subject. The *cp* vector defaults to the geometric mean of the *cp* sequence for the pruned tree on the full data set.

```
> fit <- rpart(pgtime ~ age + eet + g2 + grade + gleason + ploidy, stagec)
> fit$cptable
          CP nsplit rel error   xerror      xstd
1 0.07572983      0 1.0000000 1.012323 0.1321533
2 0.02825076      2 0.8485403 1.094811 0.1547652
3 0.01789441      7 0.7219625 1.216495 0.1603581
4 0.01295145      8 0.7040681 1.223120 0.1610315
5 0.01000000      9 0.6911166 1.227213 0.1660616

> temp <- xpred.rpart(fit)
> err  <- (stagec$pgtime - temp)^2
> sum.err <- apply(err,2,sum)
> sum.err / (fit$frame)$dev[1]
 0.27519053 0.04625392 0.02248401 0.01522362 0.01138044
   1.021901   1.038632    1.14714   1.179571   1.174196
```

The answer from `xpred.rpart` differs from the model computation shown earlier due to different random number seeds, which causes slightly different cross-validation groups.

11 Relation to other programs

11.1 CART

Almost all of the definitions in rpart are equivalent to those used in CART, and the output should usually be very similar. The printout given by `summary.rpart` was also strongly influenced by some early `cart` output. Some known differences are

- Surrogate splits: `cart` uses the percentage agreement between the surrogate and the primary split, and `rpart` uses the total number of agreements. When one of the surrogate variables has missing values this can lead to a different ordering. For instance, assume that the best surrogate based on x_1 has 45/50 = 90% agreement (with 10 missing), and the best based on x_2 has 46/60 (none

missing). Then `rpart` will pick x_2. This is only a serious issue when there are a large number of missing values for one variable, and indeed the change was motivated by examples where a nearly-100best surrogate due to perfect concordance with the primary split.

- : Computation: Some versions of the `cart` code have been optimized for very large data problems, and include such features as subsampling from the larger nodes. Large data sets can be a problem in S-plus.

11.2 Tree

The user interface to `rpart` is almost identical to that of the `tree` functions. In fact, the `rpart` object was designed to be a simple superset of the `tree` class, and to inherit most methods from it (saving a lot of code writing). However, this close connection had to be abandoned. The rpart object is still very similiar to tree objects, differing in 3 respects

- Addition of a `method` component. This was the single largest reason for divergence. In `tree`, splitting of a categorical variable results in a `yprob` element in the data structure, but regression does not. Most of the "downstream" functions then contain the code fragment "if the obeject contains a yprob component, then do A, else do B". `rpart` has more than two methods, and this simple approach does not work. Rather, the method used is itself retained in the output.

- Additional components to describe the tree. This includes the `yval2` component, which contains further response information beyond the primary value. (For the gini method, for instance, the primary response value is the predicted class for a node, and the additional value is the complete vector of class counts. The predicted probability vector `yprob` is a function of these, the priors, and the tree topology.) Other additional components store the competitor and surrogate split information.

- The `xlevels` component in `rpart` is a list containing, for each factor variable, the list of levels for that factor. In tree, the list also contains `NULL` values for the non-factor predictors. In one problem with a very large (4096) number of predictors we found that the processing of this list consumed nearly as much time and memory as the problem itself. (The inefficiency in S-plus that caused this may have since been corrected).

Although the `rpart` structure does not inherit from class `tree`, some of the `tree` functions are used by the `rpart` methods, e.g. tree.depth. `Tree` methods that have

y	1	2	3	1	2	3	1	2
x1	1	2	3	4	5	6	7	8
x2	1	2	3	4	5	6	1	2
x3	NA	22	38	12	NA	48	14	32

y	3	1	2	3	1	2	1
x1	9	10	11	12	13	14	15
x2	3	4	5	6	1	2	3
x3	40	NA	30	46	28	34	48

Table 1: *Data set for the classification test case*

not been implemented as of yet include burl.tree, cv.tree, hist.tree, rug.tree, and tile.tree.

Not all of the plotting functions available for `tree` objects have been implemented for `rpart` objects. However, many can be accessed by using the `as.tree` function, which reformats the components of an `rpart` object into a `tree` object. The resultant value may not work for all operations, however. The `tree` functions ignore the `method` component of the result, instead they *assume* that

- if y is a factor, then classification was performed based on the information index,
- otherwise, anova splitting (regression) was done.

Thus the result of `as.tree` from a Poisson fit will work reasonably for some plotting functions, e.g., `hist.tree`, but would not make sense for functions that do further computations such as `burl.tree`.

12 Test Cases

12.1 Classification

The definitions for classification trees can get the most complex, especially with respect to priors and loss matrices. In this section we lay out a simple example, in great detail. (This was done to debug the S functions.)

Let $n = 15$, and the data be as given in table 1. The loss matrix is defined as

$$L = \begin{matrix} 0 & 2 & 2 \\ 2 & 0 & 6 \\ 1 & 1 & 0 \end{matrix}$$

where rows represent the true class and columns the assigned class. Thus the error in mistakenly calling a class 2 observation a 3 is quite large in this data set. The prior probabilities for the study are assumed to be $\pi = .2, .3, .5$, so class 1 is most prevalent in the input data (n_i=6, 5, and 4 observations, respectively), but class 3 the most prevalent in the external population.

Splits are chosen using the Gini index with altered priors, as defined in equation (4.15) of Breiman et al [1].

$$\begin{aligned}
\tilde{\pi}_1 &= \pi_1(0+2+2)/\sum \tilde{\pi}_i = 4/21 \\
\tilde{\pi}_2 &= \pi_2(2+0+6)/\sum \tilde{\pi}_i = 12/21 \\
\tilde{\pi}_3 &= \pi_3(1+1+0)/\sum \tilde{\pi}_i = 5/21
\end{aligned}$$

For a given node T, the Gini impurity will be $\sum_j p(j|T)[1 - p(j|T)] = 1 - \sum p(j|T)^2$, where $p(j|T)$ is the expected proportion of class j in the node:

$$p(j|T) = \tilde{\pi}_j[n_i(T)/n_i]/\sum p(i|T)$$

.

Starting with the top node, for each possible predicted class we have the following loss

predicted class	E(loss)
1	.2*0 + .3*2 + .5*1 = 1.1
2	.2*2 + .3*0 + .5*1 = 0.9
3	.2*2 + .3*6 + .5*0 = 2.2

The best choice is class 2, with an expected loss of 0.9. The Gini impurity for the node, using altered priors, is $G = 1 - (16 + 144 + 25)/21^2 = 256/441 \approx .5805$.

Assume that variable $x1$ is split at 12.5, which is, as it turns out, the optimal split point for that variable under the constraint that at least 2 observations are in each terminal node. Then the right node will have class counts of (4,4,4) and the left node of (2,1,0). For the right node (node 3 in the tree)

$$\begin{aligned}
P(R) &= .2(4/6) + .3(4/5) + .5(4/4) = 131/150 \\
&= \text{probability of the node (in the population)} \\
p(i|R) &= (.2(4/6),\ .3(4/5),\ .5(4/4))/P(R) \\
\tilde{P}(R) &= (4/21)(4/6) + (12/21)(4/5) + (5/21)(4/4) = 259/315 = 37/45 \\
\tilde{p}(i|R) &= [(4/21)(4/6),\ (12/21)(4/5),\ (5/21)(4/4)]\ (45/37) \\
G(R) &= 1 - \sum \tilde{p}(i|R)^2 \approx .5832
\end{aligned}$$

For the left node (node 2)

$$\begin{aligned} P(L) &= .2(2/6) + .3(1/5) + .5(0/4) = 19/150 \\ p(i|L) &= (.4/3,\ .3/5,\ 0)/\ P(L) = (10/19,\ 9/19,\ 0) \\ \widetilde{P}(L) &= 1 - \widetilde{P}(R) = 8/45 \\ \tilde{p}(i|L) &= [(4/21)(2/6),\ (12/21)(1/5),\ (5/21)(0/4)]/\ \widetilde{P}(L) \\ G(L) &= 1 - \sum \tilde{p}(i|L)^2 \approx .459 \end{aligned}$$

The total improvement for the split involves the change in impurity between the parent and the two child nodes

$$n(G - [\widetilde{P}(L) * G(L) + \widetilde{P}(R) * G(R)]) \approx .2905$$

where $n = 15$ is the total sample size.

For variable $x2$ the best split occurs at 5.5, splitting the data into left and right nodes with class counts of (2,2,1) and (4,3,3), respectively. Computations just exactly like the above give an improvement of 1.912.

For variable $x3$ there are 3 missing values, and the computations are similar to what would be found for a node further down the tree with only 12/15 observations. The best split point is at 3.6, giving class counts of (3,4,0) and (1,0,4) in the left and right nodes, respectively.

For the right node (node 3 in the tree)

$$\begin{aligned} \widetilde{P}(R) &= (4/21)(3/6) + (12/21)(4/5) + (5/21)(0/4) = 174/315 \\ \tilde{p}(i|R) &= [(4/21)(3/6),\ (12/21)(4/5),\ (5/21)(0/4)]\ (315/174) \\ &= (5/29,\ 24/29,\ 0) \\ G(R) &= 1 - (25 + 576)/29^2 = 240/841 approx .2854 \end{aligned}$$

For the left node (node 2)

$$\begin{aligned} \widetilde{P}(L) &= (4/21)(1/6) + (12/21)(0/5) + (5/21)(4/4) = 85/315 \\ \tilde{p}(i|L) &= [(4/21)(1/6),\ (12/21)(0/5),\ (5/21)(4/4)]\ (315/85) \\ &= (2/17,\ 0,\ 15/17) \\ G(L) &= 1 - (4 + 225)/17^2 = 60/289 \approx .2076 \end{aligned}$$

The overall impurity for the node involves only 12 of the 15 observations, giving the following values for the top node:

$$\begin{aligned} \widetilde{P}(T) &= 174/315 + 85/315 = 259/315 \\ \tilde{p}(i|T) &= [(4/21)(4/6),\ (12/21)(4/5),\ (5/21)(4/4)]\ (315/259) \\ &= (40/259,\ 144/259,\ 75/259) \\ G(T) &= 1 - (40^2 + 144^2 + 75^2)/259^2 = 39120/67081 \end{aligned}$$

The total improvement for the split involves the impurity G of all three nodes, weighted by the probabilities of the nodes under the alternate priors.

$$15 * \{(259/315)(39120/67081) - [(174/315)(240/841) + (85/315)(60/289)]\} \approx 3.9876$$

As is true in general with the rpart routines, variables with missing values are penalized with respect to choosing a split – a factor of 259/315 or about 12/15 in

Cutpoint	$P(L)$	$P(R)$	$G(L)$	$G(R)$	ΔI
1.3	0.03	0.97	0.00	0.56	0.55
1.8	0.06	0.94	0.00	0.53	1.14
2.5	0.18	0.82	0.46	0.57	0.45
2.9	0.21	0.79	0.50	0.53	0.73
3.1	0.32	0.68	0.42	0.56	1.01
3.3	0.44	0.56	0.34	0.52	1.96
3.6	0.55	0.45	0.29	0.21	3.99
3.9	0.61	0.39	0.41	0.26	2.64
4.3	0.67	0.33	0.48	0.33	1.56
4.7	0.73	0.27	0.53	0.45	0.74
4.8	0.79	0.21	0.56	0.00	0.55

Table 2: *Cut points and statistics for variable $x3$, top node*

the case of $x3$ at the top node. Table 2 shows the statistics for all of the cutpoints of $x3$.

Because $x3$ has missing values, the next step is choice of a surrogate split. Priors and losses currently play no role in the computations for selecting surrogates. For all the prior computations, the effect of priors is identical to that of adding case weights to the observations such that the apparent frequencies are equal to the chosen priors; since surrogate computations do account for case weights, one could argue that they should also then make use of priors. The argument has not yet been found compelling enough to add this to the code.

Note to me: the cp is not correct for usesurrogate=0. The error after a split is not (left error + right error) – it also needs to have a term (parent error for those obs that weren't split).

13 User written rules

The later versions of rpart have the ability to use a set of S functions to define user written splitting rules. This was originally written as a way to pilot new ideas in `rpart`, before adding them to the underlying C code. As it turns out, many of these extensions have been useful as-is, and we encourage others to try out the feature.

If you were to examine the underlying C-code for `rpart`, you would discover that the greatest portion of it is concerned with a number of simple, but tedious, bookkeeping tasks. These include the list of which observatations are in each node, which split was the best (primary) and which were secondary, pruning rules, cross-validation, surrogate splits, and missing values. The primary design goal for user-

written splits was to hide all of this complexity from the user. A user-written split rule consists of a set of five functions: an initialization function that sets certain parameters and checks the input data for consistency, an evaluation function that labels each node and controls cost-complexity pruning, and, most important, a splitting function. Two others functions define the labels for printed and plotted output. We will explore these using 4 examples.

13.1 Anova

The anova rule is one of the simplest rules to implement. It is already built into rpart, so it is also easy to test out the accuracy of our code.

Let us start with the init function.

```
temp.init <- function(y, offset, parms, wt) {
    if (!is.null(offset)) y <- y-offset
    if (is.matrix(y)) stop("response must be a vector")

    list(y=y, parms=0, numy=1, numresp=1)
    }
```

The arguments of init are always the y data, an optional offset, any user supplied control parameters, and a vector of weights.

In the case of anova (regression), we expect y to be a vector, not a matrix. If there is an offset, it is just subtracted from y before doing the computations, that is, we compute on the residuals after subtracting the offset. There are no optional parameters, so that argument is ignored, as are the weights.

The important task of the routine is to define a list of returned elements:

1. The (possibly transformed) response vector y.
2. The vector of optional control parameters. These are passed forward to the splitting and evaluation functions, but not otherwise used.
3. The number of columns of y.
4. The number of responses. This is the length of the labeling vector that will be produced by the evaluation function. In this case, the label will be the mean of the node, so is of length 1.

Next comes the evaluation function:

```
temp.eval <- function(y, wt, parms) {
    wmean <- sum(y*wt)/sum(wt)
    rss <- sum(wt*(y-wmean)^2)
```

```
list(label= wmean, deviance=rss)
}
```

This function will be passed the y vector (or matrix) *for a single node*. It needs to create a label (which will be used in the printout), along with a measure of the impurity of the node. The label must be a numeric vector of length `numresp`, the value returned by the init function, and can contain anything that you might find useful in the final printout. The measure of impurity does not have to be a deviance, although that is the name of the argument and is a common choice for the impurity. This impurity is used to drive the cost-complexity pruning, by comparing the sum of impurities for two child nodes to the impurity of their parent. (If the impurity does not get smaller, then the split has, by definition, not accomplished anything. We discuss this in more detail later.)

The real work is done by the splitting function, which is always called with the following arguments

```
temp.split <- function(y, wt, x, parms, continuous)
```

Here `y`, `wt`, and `x` are the response vector (or matrix), the vector of weights, and the vector of values x for a single predictor at a *single* node. Any missing values have been eliminated, and the data has already been sorted according to the values of x. The last argument indicates whether the x vector corresponds to a continuous variable (T) or to a factor (F). Factors have already been converted into integers $1, 2, \ldots$. (Although not all levels might be present in this node!) The `parms` vector has been passed forward from the `init` function.

For the case of x continuous, the splitting function needs to return two vectors, `goodness` and `direction`. If the y vector is of length n, then each of these needs to be of length $n - 1$. The first element of the goodness vector gives a measure of how 'good' a split of y_1 vs $y_2, \ldots, y_n$ would be, the second the goodness of y_1, y_2 vs $y_3, \ldots, y_n$, and so on. This vector of values is used by the routine to choose the best split on this particular x variable, and then uses that value to choose which variable gives the best split overall.

In the simple example below we use the lung cancer data set, which has 7 continuous predictors and and 228 observations. The final tree has 25 nodes (internal and terminal), which means that the split routine was called 7*25= 175 times (without cross-validation). If at all possible, we would like to calculate all of the split statistics *without* using a `for` loop in the code. For anova splitting, the goodness criteria we will use is to maximize the difference in sums of squares $\sum(y - \bar{y})^2$ between the parent node and the sum of its two children nodes. Looking back in our old textbooks at the formula for one-way ANOVA, we see that this is equivalent to

$$n_L(\bar{y}_L - \bar{y})^2 + n_R(\bar{y}_R - \bar{y})^2$$

where L and R refer to the left and right child respectively. This leads to an efficient algorithm for the continuous case

```
temp.split <- function(y, wt, x, parms, continuous) {
    # Center y
    n <- length(y)
    y <- y- sum(y*wt)/sum(wt)

    if (continuous) {
        temp <- cumsum(y*wt)[-n]
        left.wt  <- cumsum(wt)[-n]
        right.wt <- sum(wt) - left.wt
        lmean <- temp/left.wt
        rmean <- -temp/right.wt

        goodness <- (left.wt*lmean^2 + right.wt*rmean^2)/sum(wt*y^2)
        list(goodness= goodness, direction=sign(lmean))
        }
```

We have made one further simplification to the formula by first centering y. The direction vector contains values of ± 1, and is used in an advisory capactity by `rpart` in organizing the printout. As used here, it causes the smaller node, in terms of $\bar{y}$, to be plotted as the left child and the larger one as the right child. This can make the overall tree easier to read.

The code does not need to worry about ties in the x variable. If, for instance x 6 and 7 were tied, then a split between positions 6 and 7 is not really possible. The parent routine takes care of this.

The above code has also made another important but not necessarily obvious choice. It has returned the percentage decrease in SS rather than the absolute decrease. One obvious effect of this is that the 'goodness' values on the printout are an R^2 between 0 and 1. A second concerns missing values. Suppose that one particular covariate was missing on 1/2 of the subjects, but on the others it was perfectly predictive. In the above code, it will have an R^2 of 1 and be chosen as the best split. If instead we had returned the absolute change in SS (without dividing by $\sum w_i y_i^2$), the variable might easily have been only second or third best, beaten out by one that had a less perfect R^2 but a smaller number of missing values. If there were no missing values, then choosing the split based on absolute decrease in the SS, relative decrease in the SS, or the F-statistic for the decrease are equivalent rules; but not so if some variables have missing values. None of these three choices is more 'right' than the other, and which is more useful depends on the particular problem at hand. (There are proponents of each who would strongly disagree with this statement). Sometimes, for instance, a variable might be missing in the training

data set, but if chosen could always be made available in the actual application of the rule. Then the split based on R^2, which does not penalize missing values, seems sensible. If, on the other hand, there were a variable that will always be missing on 50% of the observations, use of the absolute SS would appropriately discourage its choice.

The last part of the split routine deals with the case of a predictor that is a factor. If the factor has k distinct levels, then the return vector of goodness values is expected to be of length $k-1$. The direction vector will be of length k, and it gives an ordering to the k levels. The first element of `goodness` corresponds to a split between the ordered levels 1 and $2-k$, the second to one that splits the first two ordered levels from the others, and so on.

In the case of ANOVA splitting, it is fairly easy to prove that if one first orders the levels of a categorical predictor according to the average y value for that category, then the best possible split, among all 2^{k-1} possible ways of breaking the k levels into two subsets, is one of the $k-1$ choices obtained by considering only the ordered splits. A similar result can be shown for almost all applications to date of rpart, although each is a separte proof. For simplicity of the code, the user-split routines *assume* that such a result holds. (An example where this is not true is a classification tree with more than 2 categories for y.)

```
else {
    # Categorical X variable
    ux <- sort(unique(x))
    wtsum <- tapply(wt, x, sum)
    ysum  <- tapply(y*wt, x, sum)
    means <- ysum/wtsum

    # order the categories by their means
    #  then use the same code as for a non-categorical
    ord <- order(means)
    n <- length(ord)
    temp <- cumsum(ysum[ord])[-n]
    left.wt  <- cumsum(wtsum[ord])[-n]
    right.wt <- sum(wt) - left.wt
    lmean <- temp/left.wt
    rmean <- -temp/right.wt
    list(goodness= (left.wt*lmean^2 + right.wt*rmean^2)/sum(wt*y^2),
         direction = ux[ord])
    }
}
```

The last two functions control the printout.

```
temp.summary<- function(yval, dev, wt, ylevel, digits ) {
    paste("  mean=", format(signif(yval, digits)),
          ", MSE=" , format(signif(dev/wt, digits)),
                        sep='')
            }
temp.text <- function(yval, dev, wt, ylevel, digits, n, use.n ) {
    if(use.n) paste(formatg(yval,digits),"
nn=", n,sep="")
    else      paste(formatg(yval,digits))
    }
```

Our summary function has been designed to create a single line containing the mean and mean squared error for the node. This label is used in the printout of `summary.rpart`. The text function is used to label the nodes in a plot, and will usually be designed to create a much shorter label. The `ylevel` argument applies when the response is a factor, and contains the text lablels for each level of the response; it is not applicable in this case. The total number of observations is a default part of the summary printout but an optional part of text labels, controled by the `use.n` argument of `text.rpart`.

To use these functions in rpart is quite simple. We bundle the functions into a list, and then use that as the splitting criteria.

```
alist <- list(eval=temp.eval, split=temp.split, init=temp.init,
              summary=temp.summary, text=temp.text)

fit0 <- rpart(income ~population +illiteracy  + murder + hs.grad,
             mystate, control=rpart.control(minsplit=10, xval=0),
             method=alist)
```

13.2 Smoothed anova

The anova example above is uninteresting in one way, in that it only duplicates an ability already in the rpart code. (In fact, the test suite for `rpart` uses this fact to validate the code for user-written functions). We now consider an extension of the anova code based on smoothed splits.

Figure 19 shows the goodness of split versus the split point for age, in the simple model `rpart(time` $\sim$ `age, data=lung)`. The roughness of the split criteria is not uncommon, in this case a cut between ages 75 and 76 has the second best goodness, and is the split chosen by rpart with the default minimum node size of 5, but a split between 74 and 75, or 76 and 77, is not nearly as useful. Superimposed on the plot are the results of the `smooth` function, which is much better behaved and has its maximum for a split between ages 72 and 73.

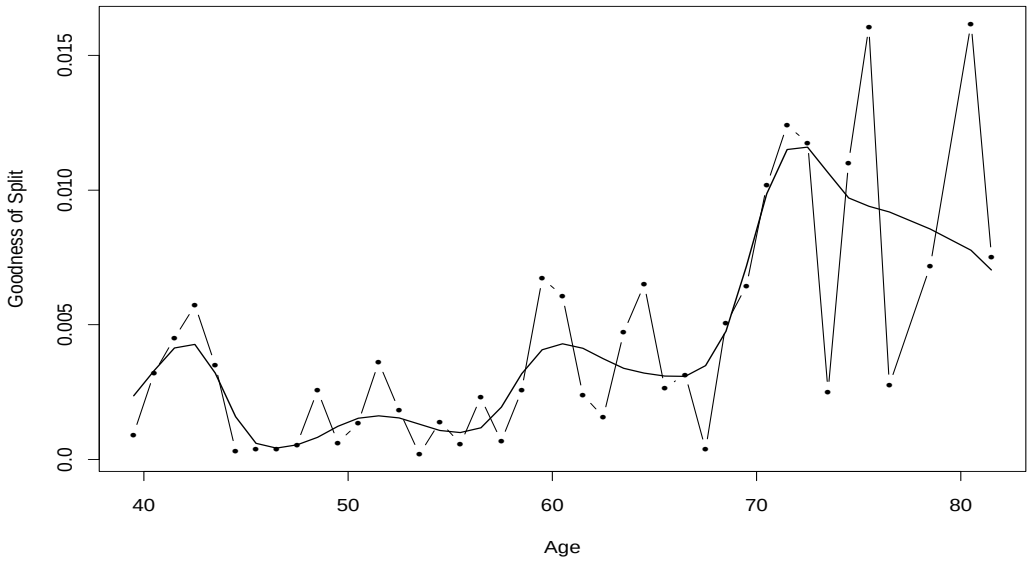


Figure 19: *Cutpoints versus goodness for the age variable, lung cancer data.*

Implementing a smoothed split in the user written function is quite easy. One important constraint, however, is that we need to ignore ties. Even though $n-1$ results are returned in the goodness vector, the parent rpart routine ignores all but those for which `x[i] != x[i+1]`, i.e., it will not split observations that have the same x value; the remaining elements of the goodness vector are really just intermediate sums. Our call to the `smooth` function also needs to have these intermediate values suppressed, to get a valid smoother We need to add only 3 lines to the split function defined in the last section (for the continuous case)

```
...
goodness <- (left.wt*lmean^2 + right.wt*rmean^2)/sum(wt*y^2)

indx <- which(diff(x)!=0)                  #  new code
if (length(indx) >4)
     goodness[indx] <- smooth(goodness[indx])   # new code

list(goodness= goodness, direction=sign(lmean))
}
```

(The `smooth` function requires an argument of length at least 4.) There are of course many different smoothers that could be used, some with stronger justification than the original Tukey suggestions implemented by `smooth`. An advantage of user-written split rules is the ease with which different suggestions can be explored.

13.3 Classification with an offset

This example comes by way of an interaction with Jinbo Chen, who had a very interesting problem. The outcome variable is 0/1, so we want to do classifiaction, but the input variables can be divided into two groups

1. a large number of genetic variables, for which a tree based model is appealing,
2. a set of clinical variables for which an ordinary logistic regression model is felt to be superior

How might we blend these two tasks?

The first question is how the two predictions should interact. That is, if we knew the right model for the clinical variables, how would we want to use that in the `rpart` model, and vice versa. After some discussion, we decided to use a logistic link, so that the final predictor would be

$$\begin{aligned} Pr(y=1) &= \frac{e^\eta}{1+e^\eta} \\ \eta_i &= X_i\beta + \gamma_{(i)} \end{aligned}$$

where X_i are the clinical variables for observation i, β the coefficients for those variables, and γ a vector of predictors from the rpart model, one value per terminal node, $\gamma_{(i)}$ stands for the terminal node into which observation i falls.

To fit this, we need to use alternating `glm` and `rpart` models, each of which includes the result of the other as an offset. We will ignore here the issues of which goes first, termination of the iteration, etc., and focus only on how to fit the `rpart` portion of the data. For details see [2].

Here are the init and summary routines.

```
jinit <- function(y, offset, parms, wt) {
    if (is.null(offset)) offset <- 0
    if (any(y!=0 & y!=1)) stop ('response must be 0/1')

    list(y=cbind(y, offset), parms=0, numresp=2, numy=2)
    }
jsumm <- function(yval, dev, wt, ylevel, digits ) {
               paste("events=",  round(yval[,1]),
                     ", coef= ", format(signif(yval[,2], digits)),
                     ", deviance=" , format(signif(dev, digits)),
                       sep='')
            }
```

We need to make the offset variable available to the splitting rule, so it is added as a second column of the response. (This trick of tacking ancillary information

needed at every split onto the 'y' variable was first suggested by Dan Schaid, and has been useful in a number of cases). At this point, we are not supporting priors or losses, so there are no optional parameters. The printed label will include the number of events along with the coefficient γ for the node. We do not supply a text routine; the default action of rpart is to use the first element of vector returned by our eval function, which is sufficient in this case.

For the splitting rule, we make direct use of the glm routine:

```
jsplit <- function(y, wt, x, parms, continuous) {
    if (continuous) {
        # do all the logistic regressions
        n <- nrow(y)
        goodness <- direction <- double(n-1)  # allocate 0 vector
        temp <- rep(0, n)    # this will be the 'x' variable in glm
        for (i in 1:(n-1)) { # for each potential cut point
            temp[i] <- 1     # update the left/right indicator
            if (x[i] != x[i+1]) {
                tfit <- glm(y[,1] ~ temp + offset(y[,2]),
                            binomial, weight=wt)
                goodness[i] <- tfit$null.deviance - tfit$deviance
                direction[i] <- sign(tfit$coef[2])
                }
            }
        }
    else {        # Categorical X variable
        # First, find out what order to put the categories in, which
        #  will be the order of the coefficients in this model
        tfit <- glm(y[,1] ~ factor(x) + offset(y[,2]) -1, binomial,
                              weight=wt)
        ngrp <- length(tfit$coef)
        direction <- (1:ngrp)[order(tfit$coef)]

        xx <- direction[match(x, sort(unique(x)))] #relabel
        goodness <- double(length(direction) -1)
        for (i in 1:length(goodness)) {
            tfit <- glm(y[,1] ~ I(xx>i) + offset(y[,2]),
                                    binomial, weight=wt)
            goodness[i] <- tfit$null.deviance - tfit$deviance
            }
        }
    list(goodness=goodness, direction=direction)
    }
```

Consider the continuous case first. For each possible splitpoint, we create a dummy variable `temp` which divides the observations into left and right sons, and then call

glm to do the fit. When running this example, the first thing you will notice is that it is astoundingly slow, due to the large number of calls to the glm function. We have already taken the first step in speeding up the function by only doing the productive calls, those where $x_i \neq x_{i+1}$, since the parent routine will ignore all the other values of the goodness vector anyway. The next step in speed would be to optimize the glm call itself. For instance, since we know that the x, y, offset, and weight vectors have already been checked — all the same length with no missing values — the second level routine glm.fit could be used directly, saving some overhead.

The algorithm for the categorical case is only a bit more complex, although the necessary indexing looks more formidable. Our starting point is a theorem found in Brieman et al [1] that for the 2 class case, one can first order the categories defined by x according to $\hat{p}$, the fraction of observations in that category with $y = 1$; the optimal split must then be one of those between the ordered categories. We assume (without any proof) that the theorem still holds in the case of offsets.

The program uses a first glm to find the ordered coefficients for the categories, and then reorders them. Suppose that $x = 1, 2, 1, 3, 3, 5, 3, 1, 2, 5$, and the coefficients of the first glm fit were -.3, .4, -.6, and .2. (Remember that in a node further down the tree, not all categories of x might be present). Then we want to set direction= 3,1,4,2, showing that the third group in our list should be first in line, then group 1, group 5 (the 4th largest in our list), and finally group 2. For easier computation, we create a dummy variable xx = 2,4,2,1,1,3,1,2,4,3 which has relabeled the categories from first to last (so 3 becomes a 1, reflecting its rank, 1 becomes 2, etc.)

The evaluation rule also makes use of a call to glm

```
jeval <- function(y, wt, parms) {
    tfit <- glm(y[,1] ~ offset(y[,2]), binomial, weight=wt)
    list(label= c(sum(y[,1]), tfit$coef), deviance=tfit$deviance)
    }
```

Here is a trivial example of using the functions. Remember that the status variable is coded as 1/2 in the lung data set.

```
jlist <- list(eval=jeval, split=jsplit, init=jinit, summary=jsumm)
temp <- predict(glm((status-1) ~ age, binomial, data=lung))
rpart(status-1 ~ factor(ph.ecog) +sex + meal.cal +
                            ph.karno + offset(temp),
               data=lung, method=jlist)

  n= 228

          CP nsplit rel error
1 0.04437011      0 1.0000000
```

```
2 0.03567130      1 0.9556299
3 0.02907407      3 0.8842873
4 0.01339915      4 0.8552132
5 0.01303822      7 0.8150158
6 0.01272237      8 0.8019775
7 0.01000000      9 0.7892552

Node number 1: 228 observations,    complexity param=0.04437011
events=165, coef=  9.703813e-12, deviance=2.637128e+02
  left son=2 (90 obs) right son=3 (138 obs)
  Primary splits:
      sex             < 1.5    to the right, improve=11.700970, (0 missing)
      ph.karno        < 75     to the right, improve= 7.853151, (1 missing)
      factor(ph.ecog) splits as  LLRR,       improve= 6.896152, (1 missing)
      meal.cal        < 1087.5 to the left,  improve= 2.376010, (47 missing)

Node number 2: 90 observations,    complexity param=0.0356713
events=53, coef= -5.683397e-01, deviance=1.201381e+02
  left son=4 (27 obs) right son=5 (63 obs)
  Primary splits:
      factor(ph.ecog) splits as  LRR-,       improve=8.989182, (0 missing)
      meal.cal        < 1067.5 to the left,  improve=8.845852, (23 missing)
      ph.karno        < 95     to the right, improve=2.954572, (0 missing)
  Surrogate splits:
      ph.karno < 95     to the right, agree=0.833, adj=0.444, (0 split)

...
```

In this data set of advanced lung cancer patients, the effect of age is not very highly confounded with that for the other variables. A model without the offset has the same split points, in the first node, for 3 of the 4 primary splits listed. Notice that the overall coefficient for the first node is 0, which is expected since the logistic model using age has already incorporated an overall intercept.

13.4 Cost-complexity pruning

The evaluation function for a node returns both a label for the node, which is an arbitrary choice of the designer, along with a 'deviance'. This latter is intended to be some measure of the impurity of a node, which need not necessarily be an actual deviance, but it does have some restrictions. It is nice if the value has an intrinsic meaning to the user, since it is printed as part of the summary output; it is also used, however, in cost complexity pruning.

Let D be the deviance for a node. For a given value c of the complexity param-

eter, a given split is evaluated using a recursive rule

$$D_{parent} - \sum [D_{child} + c]$$

This is evaluated, using an efficient recursive algorithm, over all possible subtrees of the parent node; the right hand sum is over all the terminal nodes of said subtree. The value c acts somewhat like the per-variable penalty in an AIC or BIC calculation. If there is at least one larger tree, i.e., set of splits below the parent, for which this overall score is > 0, then a split of this parent node is "worth keeping" at threshold c.

- Note that since $c \geq 0$ and $D \geq 0$, any node which has an impurity $D = 0$ will not be split further. The node has been marked as perfectly pure, so there is nothing left to do.
- In all of the examples above the impurity measure D and the goodness-of-split measure were closely related. This is not a necessary constraint.
- An overly vigorous or a poorly chosen measure D will lead to the program declaring that no splits are worth keeping. For instance $D = n$, the node size, always shows that a split had no real gain. Setting $D = n^2$ would show that any split gives a gain in impurity (at $c = 0$).

Choosing a good impurity measure is often one of the harder parts of creating a new splitting method. However, the cost-complexity measure is what orders the final tree into a set of nested models. This aids in printing and plotting, and is essential for cross-validation.

14 Source

The software exists in two forms: a stand-alone version, which can be found in statlib in the 'general' section, and an S version, also on stalib, but in the 'S' section of the library. All of the splitting rules and other computations exist in both versions, but the S version has been enhanced with several graphics options, most of which are modeled (copied actually) on the `tree` functions.

References

[1] L. Breiman, J.H. Friedman, R.A. Olshen, , and C.J Stone. *Classification and Regression Trees*. Wadsworth, Belmont, Ca, 1983.

[2] J. Chen, K. Yu, A. Hsing, and T. Therneau. A partially lineear tree-based regression model for assessing complex joint gene-gene and gene-environment effects. *Genetic Epidemiology*, 31:228–251, 2007.

[3] L.A. Clark and D. Pregibon. Tree-based models. In J.M. Chambers and T.J. Hastie, editors, *Statistical Models in S*, chapter 9. Wadsworth and Brooks/Cole, Pacific Grove, Ca, 1992.

[4] M. LeBlanc and J Crowley. Relative risk trees for censored survival data. *Biometrics*, 48:411–425, 1992.

[5] O. Nativ, Y. Raz, H.Z. Winkler, Y. Hosaka, E.T. Boyle, T.M. Therneau, G.M. Farrow, R.P. Meyers, H. Zincke, and M.M Lieber. Prognostic value of flow cytometric nuclear DNA analysis in stage C prostate carcinoma. *Surgical Forum*, pages 685–687, 1988.

[6] T.M. Therneau. A short introduction to recursive partitioning. Orion Technical Report 21, Stanford University, Department of Statistics, 1983.

[7] T.M. Therneau, Grambsch P.M., and T.R. Fleming. Martingale based residuals for survival models. *Biometrika*, 77:147–160, 1990.