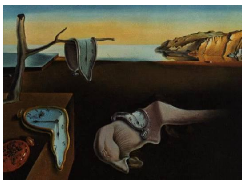

Journal of Statistical Software
April 2012, Volume 47, Issue 10. http://www.jstatsoft.org/

Classification Trees for Ordinal Responses in R: The rpartScore Package

Giuliano Galimberti
Università di Bologna

Gabriele Soffritti
Università di Bologna

Matteo Di Maso
Università di Bologna

Abstract

This paper introduces **rpartScore** (Galimberti, Soffritti, and Di Maso 2012), a new R package for building classification trees for ordinal responses, that can be employed whenever a set of scores is assigned to the ordered categories of the response. This package has been created to overcome some problems that produced unexpected results from the package **rpartOrdinal** (Archer 2010). Explanations for the causes of these unexpected results are provided. The main functionalities of **rpartScore** are described, and its use is illustrated through some examples.

Keywords: CART, generalized Gini impurity function, pruning, **rpart**.

1. Introduction

rpartOrdinal is a package implemented by Archer (2010) in R (R Development Core Team 2012), that contains some splitting functions for building a classification tree for predicting an ordinal response. These splitting functions are meant to be used in conjunction with the R package **rpart** (Therneau and Atkinson 2011), which was originally introduced as an implementation of the classification and regression trees (CART) methodology (Breiman, Friedman, Olshen, and Stone 1984). Three splitting functions are considered in **rpartOrdinal**. These three splitting functions are based on an ordinal impurity function (Piccarreta 2008), the generalized Gini impurity function (Breiman *et al.* 1984) and the ordered twoing method (Breiman *et al.* 1984).

While using this package on some real data (Galimberti and Soffritti 2012), we obtained some unexpected results. A careful inspection of the R functions contained in **rpartOrdinal** led us to discover the causes of these problems and to implement **rpartScore** (Galimberti *et al.* 2012), a new R package that overcomes these problems. In this paper, we provide theoretical and empirical explanations for the causes of the unexpected results obtained with **rpartOrdinal**

(Section 2). We next describe the main functionalities of **rpartScore** (Section 3). Finally, some directions for future research are highlighted in Section 4. All the examples shown in this paper were performed using the R release 2.14.2.

2. Unexpected results from rpartOrdinal

The reasons why **rpartOrdinal** may produce unexpected results depend on multiple issues. Some of these issues are related to the theoretical properties of the generalized Gini impurity function and to the way this function has been implemented in **rpartOrdinal**. The first aspect is detailed in Section 2.1, while Section 2.2 and Section 2.3 address the second aspect. Another issue concerns the measure of the predictive performance used in the pruning procedure (see Section 2.4).

2.1. Some properties of the generalized Gini impurity function

Using a notation similar to the one in Archer (2010), consider a data set $\{(y_i, x_{i1}, \ldots, x_{ip}); i = 1, \ldots, n\}$ that contains the values of a response Y with J ordered categories $\{\omega_1 < \omega_2 < \ldots < \omega_J\}$ and of p predictors $X_1, \ldots, X_p$ observed on n sample units. A classification tree can be derived by recursively partitioning the data set using splitting rules, which are logical rules defined according to the values of the predictors. Most statistical packages that can be used to implement classification trees rely on splitting rules obtained using one predictor at a time. According to the CART methodology, the optimal binary splitting rule for a given node t in a tree $\mathcal{T}$ is the one that results in the largest decrease in node impurity, as measured using a given impurity function. Let $p(\omega_j|t)$ be the proportion of units in node t belonging to the j-th category of Y, for $j = 1, \ldots, J$. The generalized Gini impurity function (Breiman *et al.* 1984) for node t is given by

$$i_{GG}(t) = \sum_{k=1}^{J} \sum_{l=1}^{J} C(\omega_k|\omega_l) p(\omega_k|t) p(\omega_l|t), \tag{1}$$

where $C(\omega_k|\omega_l)$ represents the misclassification cost of assigning category ω_k to a sample unit belonging to category ω_l. Clearly, $C(\omega_j|\omega_j) = 0$ for $j = 1, \ldots, J$. When $C(\omega_k|\omega_l) = C(\omega_l|\omega_k)\ \forall\ k \neq l$, these misclassification costs are symmetric and can also be interpreted as dissimilarities between pairs of categories of Y. The nominal Gini impurity function is obtained as a special case of Equation 1 by choosing $C(\omega_k|\omega_l) = 1\ \forall\ k \neq l$.

For any binary split s of node t, units assigned to node t are partitioned into two child nodes, t_R and t_L. According to Breiman *et al.* (1984), the decrease in node impurity induced by the binary split s of node t is given by

$$\Delta Imp_{GG}(t, s) = p(t) i_{GG}(t) - p(t_R) i_{GG}(t_R) - p(t_L) i_{GG}(t_L), \tag{2}$$

where $p(t)$, $p(t_R)$ and $p(t_L)$ are the proportions of units assigned to nodes t, t_R and t_L, respectively. At any step in the recursive partitioning procedure, an exhaustive search among all the admissible binary splits for a given node is performed, and the split showing the largest value of Equation 2 is selected.

Suppose that a set of increasing scores $\{s_1 < s_2 < \ldots < s_J\}$ is assigned to the ordered categories of the response Y. Variable misclassification costs can be defined by considering suitable transformations of the absolute differences between pairs of scores. For example, if one chooses $C(\omega_k|\omega_l) = |s_k - s_l|$, Equation 1 becomes

$$i_{GG1}(t) = \sum_{k=1}^{J}\sum_{l=1}^{J} |s_k - s_l| p(\omega_k|t)p(\omega_l|t). \tag{3}$$

Piccarreta (2001) showed that $i_{GG1}(t)$ can also be computed as follows:

$$i_{GG1}(t) = 2\sum_{j=1}^{J-1}(s_{j+1} - s_j)F(\omega_j|t)\left(1 - F(\omega_j|t)\right), \tag{4}$$

where $F(\omega_j|t) = \sum_{h=1}^{j} p(\omega_h|t)$, for $j = 1, \ldots, J$, are the cumulative proportions of Y in node t. By choosing any set of scores such that $(s_{j+1} - s_j) = 1$ for $j = 1, \ldots, J-1$, Equation 4 simplifies to

$$i^*_{GG1}(t) = 2\sum_{j=1}^{J-1} F(\omega_j|t)\left(1 - F(\omega_j|t)\right), \tag{5}$$

which is proportional to the ordinal impurity function $i_{OS}(t) = \sum_{j=1}^{J-1} F(\omega_j|t)\left(1 - F(\omega_j|t)\right)$ proposed by Piccarreta (2008).

By defining $C(\omega_k|\omega_l) = (s_k - s_l)^2$, Equation 1 results in

$$i_{GG2}(t) = \sum_{k=1}^{J}\sum_{l=1}^{J}(s_k - s_l)^2 p(\omega_k|t)p(\omega_l|t). \tag{6}$$

By adding and subtracting to each term $(s_k - s_l)$ the average score within node t, $\overline{s}(t) = \sum_{j=1}^{J} s_j p(\omega_j|t)$, it is easy to prove that

$$i_{GG2}(t) = 2\sum_{j=1}^{J}(s_j - \overline{s}(t))^2 p(\omega_j|t) = \frac{2}{n \cdot p(t)} SS(t), \tag{7}$$

that is, $i_{GG2}(t)$ is proportional to $SS(t)$, which is the deviance of the scores $s_1, \ldots, s_J$ within node t. The resulting splitting function is then equal to

$$\Delta Imp_{GG2}(t,s) = \frac{2}{n}\left[SS(t) - SS(t_R) - SS(t_L)\right]. \tag{8}$$

Thus, from a computational point of view, $\Delta Imp_{GG2}(t,s)$ corresponds (up to a multiplicative factor) to the ANOVA splitting function for a regression tree built using the scores $s_1, \ldots, s_J$ as numerical values of the ordered categories of the response Y. This equivalence also makes it possible to simplify the search for the best split when the admissible binary splits are defined according to a predictor X with M unordered categories. The split that maximizes Equation 8 can be found by treating X as an ordinal categorical predictor, after ordering its categories according to the M conditional mean scores of Y given X (see Breiman *et al.* 1984, for a detailed proof of this result). This makes it possible to examine only $M - 1$ splits of the $2^{M-1} - 1$ splits that can be defined according to all of the partitions of the M categories into two nonempty classes. Note that this computational simplification does not hold when the decrease in impurity is evaluated using the impurity function in Equation 5 (see Section 2.3 for an example).

2.2. Use of variable misclassification costs in rpartOrdinal

Archer (2010) proposes that the generalized Gini impurity function be implemented by specifying a matrix of variable misclassification costs as a `loss` parameter in the optional `parms` argument within the `rpart` call. The `method =` argument of the `loss.matrix` function in **rpartOrdinal** can be set to either `"linear"` or `"quadratic"` for creating linear or quadratic costs, respectively. According to Archer (2010), classification trees based on $i_{GG1}(t)$ and $i_{GG2}(t)$ should be derived in this way.

The following example shows some of the unexpected results produced by **rpartOrdinal**. Consider the `lowbwt` data set, contained in **rpartOrdinal** and used in some illustrative applications presented by Archer (2010). The ordinal response `Category` is obtained by binning the variable `bwt` according to the following cutoffs: 2500, 3000, and 3500. This data set also contains several covariates that can be used to predict this ordinal response (see Archer 2010, for further details).

The linear costs corresponding to `Category` are obtained as follows:

```
R> library("rpartOrdinal")
R> data("lowbwt")
R> lowbwt$Category <- factor(ifelse(lowbwt$bwt <= 2500, 3,
+    ifelse(lowbwt$bwt <= 3000, 2, ifelse(lowbwt$bwt <= 3500, 1, 0))),
+    ordered = TRUE)
R> linear.loss <- loss.matrix(method = "linear", lowbwt$Category)
R> linear.loss
```

```
     [,1] [,2] [,3] [,4]
[1,]    0    1    2    3
[2,]    1    0    1    2
[3,]    2    1    0    1
[4,]    3    2    1    0
```

It is evident that these costs can be obtained by choosing any set of scores such that $(s_{j+1} - s_j) = 1$ for $j = 1, \ldots, J-1$ and by setting $C(\omega_k|\omega_l) = |s_k - s_l|$. Thus, according to the properties of the generalized Gini impurity function described in Section 2.1, the use of linear misclassification costs should lead to the same optimal splits selected by the ordinal impurity function $i_{OS}(t)$ because the two corresponding splitting functions are equal (up to a constant multiplicative factor). However, the corresponding classification trees obtained by analyzing the `lowbwt` data set using **rpartOrdinal** are different (see Archer 2010, pp. 7–8). This difference is due to the fact that, contrary to what is stated by Archer (2010), p. 6, the specification of a matrix of variable misclassification costs as a `loss` parameter in the optional `parms` argument within the `rpart` call does not lead to a classification tree based on the generalized Gini impurity function. This happens because the `rpart` function does not implement this impurity function (Therneau and Atkinson 1997, p. 7), but in fact resorts to the so-called altered prior method when supplied with variable misclassification costs (see Breiman *et al.* 1984; Therneau and Atkinson 1997, for a detailed description of this method and its rationale). Briefly, this method assumes that the misclassification costs have the following structure: $C(\omega_k|\omega_l) = C(\omega_l)\ \forall\ k \neq l$, that is, variable misclassification costs that vary only according to the true category of Y. The altered prior probabilities for the categories of Y are computed as follows:

$$\tilde{\pi}(\omega_j) = \frac{\pi(\omega_j)C(\omega_j)}{\sum_{j=1}^{J} \pi(\omega_j)C(\omega_j)}, \quad j = 1, \ldots, J, \tag{9}$$

where $\pi(\omega_j)$ is the prior probability of observing a unit belonging to category ω_j. By default, in the `rpart` function, these prior probabilities are set equal to the proportions of units in the data set belonging to the J categories of Y: $p(\omega_j)$ for $j = 1, \ldots, J$. Furthermore, when supplied with a matrix of variable misclassification costs, before computing the altered prior probabilities, the `rpart` function sets $C(\omega_j) = \sum_{k=1}^{J} C(\omega_k|\omega_j)$ for $j = 1, \ldots, J$. The altered prior probabilities are then used to adjust the proportions $p(\omega_j|t)$ according to the following formula:

$$\tilde{p}(\omega_j|t) = \frac{\tilde{\pi}(\omega_j)p(t|\omega_j)}{\sum_{j=1}^{J} \tilde{\pi}(\omega_j)p(t|\omega_j)}, \quad j = 1, \ldots, J, \tag{10}$$

where $p(t|\omega_j)$ is the proportion of units belonging to the j-th category of Y which are assigned to node t.

The altered proportions $\tilde{p}(\omega_j|t)$ are then used to compute node impurity according to the nominal Gini impurity function

$$i_{GG0}(t) = \sum_{k=1}^{J} \sum_{l \neq k} \tilde{p}(\omega_k|t)\tilde{p}(\omega_l|t) = 1 - \sum_{k=1}^{J} \tilde{p}(\omega_k|t)^2, \tag{11}$$

which is the impurity measure used by default in the `rpart` function for building classification trees when Y is an unordered response. Thus, the resulting classification trees do not take into account the ordered nature of the response.

In the `lowbwt` data set example, $p(\omega_j)$, $C(\omega_j)$ and $\tilde{\pi}(\omega_j)$, for $j = 1, \ldots, 4$ can be computed as follows:

```
R> pj <- table(lowbwt$Category) / sum(table(lowbwt$Category))
R> round(pj, 4)

     0      1      2      3
0.2434 0.2434 0.2011 0.3122

R> Cj.lin <- apply(linear.loss, 1, sum)
R> Cj.lin

[1] 6 4 4 6

R> pj.Cj.lin <- pj * Cj.lin
R> alt.pj.lin <- pj.Cj.lin / sum(pj.Cj.lin)
R> round(alt.pj.lin, 4)

     0      1      2      3
0.2857 0.1905 0.1573 0.3665
```

It is easy to see that using linear misclassification costs leads to increased prior probabilities for extreme categories of the response and to decreased prior probabilities for intermediate categories. Thus, when supplied with a linear misclassification cost matrix, the `rpart` function not only employs an impurity function that ignores the ordinal nature of the response but also simplifies the misclassification cost structure by giving more weight to misclassification errors for extreme categories of Y.

To provide empirical evidence to support the conclusion that specifying a matrix of linear misclassification costs as a `loss` parameter in the optional `parms` argument within the `rpart` function call does not lead to a classification tree based on $i_{GG1}(t)$, three trees are built on the `lowbwt` data set, using the same ordinal response and predictors as in Archer (2010): `ordinal.rpart` is obtained using the ordinal impurity criterion implemented in **rpartOrdinal**, `linear.loss.rpart` is obtained by supplying the linear misclassification cost matrix `linear.loss` as a `loss` parameter in the optional `parms` argument within the `rpart` call (as suggested by Archer 2010, p. 6) and `altered.priors.lin.rpart` is obtained by supplying altered prior probabilities `alt.pj.lin` (computed according to Equation 9) as a `prior` parameter in the optional `parms` argument within the `rpart` call.

The summary of each of these three R objects contains information about the selection of the optimal split for node 1 (illustrated below) as well as other information on the corresponding tree. For the best five candidate splits of node 1 (each defined by a different predictor), the summaries report the values of `improve`, which are computed as the decrease in node impurity for `ordinal.rpart` and as the decrease in node impurity multiplied by n for `linear.loss.rpart` and `altered.priors.lin.rpart`. A comparison of these values reveals that `linear.loss.rpart` does not coincide with `ordinal.rpart`. The values of `improve` in the summaries lead to a different ranking of the candidate splits; hence, these values are not proportional (as they should be, according to the theoretical properties described in Section 2.1). Furthermore, as expected, `linear.loss.rpart` gives the same values for `improve` as does `altered.priors.lin.rpart`.

```
R> ordinal.rpart <- rpart(Category ~ age + lwt + race + smoke + ptl + ht +
+    ui + ftv, data = lowbwt, method = ordinal)
R> summary(ordinal.rpart)

Call:
rpart(formula = Category ~ age + lwt + race + smoke + ptl + ht +
    ui + ftv, data = lowbwt, method = ordinal)
  n= 189
...
...
Node number 1: 189 observations,    complexity param=0.04758604
  predicted class= 3   expected loss= 1.354385
  left son=2 (177 obs) right son=3 (12 obs)
  Primary splits:
      ht    splits as  LR,        improve=0.32949840, (0 missing)
      ui    splits as  LR,        improve=0.25190870, (0 missing)
      smoke splits as  LR,        improve=0.13943480, (0 missing)
      race  splits as  LRR,       improve=0.10658910, (0 missing)
      lwt   < 109.5 to the left,  improve=0.03974253, (0 missing)
  Surrogate splits:
      lwt < 232   to the left,  agree=0.942, adj=0.083, (0 split)
...
...

R> linear.loss.rpart <- rpart(Category ~ age + lwt + race + smoke + ptl +
+    ht + ui + ftv, data = lowbwt, method = "class",
+    parms = list(loss = linear.loss))
R> summary(linear.loss.rpart)

Call:
rpart(formula = Category ~ age + lwt + race + smoke + ptl + ht +
    ui + ftv, data = lowbwt, method = "class",
        parms = list(loss = linear.loss))
  n= 189
...
...
Node number 1: 189 observations,    complexity param=0.0964467
  predicted class=2  expected loss=1.042328
    class counts:    46    46    38    59
   probabilities: 0.243 0.243 0.201 0.312
  left son=2 (147 obs) right son=3 (42 obs)
  Primary splits:
      lwt   < 109.5 to the right, improve=5.361524, (0 missing)
      ptl   < 0.5   to the left,  improve=4.925426, (0 missing)
      race  splits as  LRR,       improve=4.294291, (0 missing)
      ui    splits as  LR,        improve=3.047050, (0 missing)
      smoke splits as  LR,        improve=2.993956, (0 missing)
  Surrogate splits:
      age < 14.5  to the right, agree=0.783, adj=0.024, (0 split)
...
...

R> altered.priors.lin.rpart <- rpart(Category ~ age + lwt + race + smoke +
+    ptl + ht + ui + ftv, data = lowbwt, method = "class",
+    parms = list(prior = alt.pj.lin))
R> summary(altered.priors.lin.rpart)

Call:
rpart(formula = Category ~ age + lwt + race + smoke + ptl + ht +
    ui + ftv, data = lowbwt, method = "class",
        parms = list(prior = alt.pj.lin))
  n= 189
...
...
Node number 1: 189 observations,    complexity param=0.06862745
  predicted class=3  expected loss=0.6335404
```

```
    class counts:    46    46    38    59
   probabilities: 0.286 0.190 0.157 0.366
  left son=2 (147 obs) right son=3 (42 obs)
  Primary splits:
      lwt   < 109.5 to the right, improve=5.361524, (0 missing)
      ptl   < 0.5   to the left,  improve=4.925426, (0 missing)
      race  splits as  LRR,       improve=4.294291, (0 missing)
      ui    splits as  LR,        improve=3.047050, (0 missing)
      smoke splits as  LR,        improve=2.993956, (0 missing)
  Surrogate splits:
      age < 14.5  to the right, agree=0.783, adj=0.024, (0 split)
...
...
```

Using a similar comparison, it is also possible to show that the use of a quadratic misclassification cost matrix (as proposed by Archer 2010) leads to decreases in node impurity that are not proportional to the ones obtained by fitting a regression tree to the scores of Y (as they should be, according to the properties described in the Section 2.1) but are instead equal to the decreases obtained by a classification tree built using the nominal Gini impurity function and the following altered priors:

```
R> quadratic.loss <- loss.matrix(method = "quadratic", lowbwt$Category)
R> Cj.quad <- apply(quadratic.loss, 1, sum)
R> Cj.quad

[1] 14 6 6 14

R> pj <- table(lowbwt$Category) / sum(table(lowbwt$Category))
R> pj.Cj.quad <- pj * Cj.quad
R> alt.pj.quad <- pj.Cj.quad / sum(pj.Cj.quad)
R> round(alt.pj.quad,4)

     0      1      2      3
0.3262 0.1398 0.1155 0.4184
```

Note that these latter comparisons are meaningful only with respect to the decreases in node impurity: predicted values from the regression tree fitted to the scores of Y should not be used, because they are equal to the average score within each terminal node and hence are meaningless for predicting a categorical response.

2.3. Use of categorical predictors in rpartOrdinal

Other unexpected results may arise when using **rpartOrdinal** with unordered categorical predictors. As stated in Section 2.1, when a predictor with M unordered categories is considered, $2^{M-1} - 1$ admissible splits can be defined, according to all of the partitions of the M categories into two nonempty classes. However, if the impurity function $i_{GG2}(t)$ is used to compute the decrease in node impurity, only $M - 1$ splits can be examined to find the best one. These

$M - 1$ splits can be identified by treating the predictor as an ordinal one, after ordering its categories according to the M conditional mean scores of Y, given the predictor.

The following example shows that this computational simplification does not hold when the decrease in impurity is defined according to the ordinal impurity function $i_{OS}(t)$ (nor when $i_{GG1}(t)$ is considered). Let Y be an ordered response with scores $\{0, 1, 2, 3\}$ and X be a categorical predictor with $M = 4$ unordered categories $\{A, B, C, D\}$. Furthermore, consider the cross-classification of $n = 400$ sample units, based on Y and X, obtained as follows:

```
R> y.labs <- 0:3
R> x.labs <- c("A", "B", "C", "D")
R> yx.labs <- expand.grid(y.labs, x.labs)
R> wt.1 <- c(6, 35, 45, 14, 36, 10, 10, 44, 51, 0, 0, 49, 0, 54, 46, 0)
R> yx.labs <- yx.labs[rep(1:nrow(yx.labs), wt.1), ]
R> y <- yx.labs[, 1]
R> x <- yx.labs[, 2]
R> cont.table <- table(y, x)
R> cont.table
```

```
   x
y    A  B  C  D
  0  6 36 51  0
  1 35 10  0 54
  2 45 10  0 46
  3 14 44 49  0
```

The conditional mean scores of Y given the four categories of X are equal to

```
R> wt <- rep(1, length(y))
R> wtsum <- tapply(wt, x, sum)
R> ysum  <- tapply(y * wt, x, sum)
R> means <- ysum / wtsum
R> means
```

```
   A    B    C    D
1.67 1.62 1.47 1.46
```

Thus, in this example, the categories of X can be ordered according to increasing values for the conditional mean scores of Y as follows: D, C, B, A.

The following instructions make it possible to compute the decrease in node impurity using $i_{OS}(t)$ for all the $2^3 - 1 = 7$ admissible splits:

```
R> no.classes <- length(y.labs)
R> node <- xtabs(wt ~ y)/sum(wt)
R> cum.node<-cumsum(node)
R> root<-0
R> for(j in 1:(no.classes - 1)) root <- root +
+    cum.node[j] * (1 - cum.node[j])
```

```
R> num.cat <- length(x.labs)
R> splits <- expand.grid(rep(list(c(0, 1)), (num.cat)))
R> splits <- splits[1:(nrow(splits) / 2), ]
R> splits <- splits[-1, ]
R> colnames(splits) <- x.labs
R> rownames(splits) <- 1:nrow(splits)
R> node.l <- function(a) apply(as.table(cont.table[, a == 1]), 1, sum)
R> left.1 <- apply(splits, 1, node.l)
R> right.1 <- as.numeric(xtabs(wt ~ y)) - left.1
R> col.left.1 <- apply(left.1, 2, sum)
R> col.right.1 <- apply(right.1, 2, sum)
R> freq.cum.l <- apply(left.1, 2, cumsum)
R> freq.cum.r <- apply(right.1, 2, cumsum)
R> cum.matrix.l.1 <- t(freq.cum.l) / col.left.1
R> cum.matrix.r.1 <-  t(freq.cum.r) / col.right.1
R> left.imp.1 <- 0
R> right.imp.1 <- 0
R> for(j in 1:(no.classes - 1)) {
+    left.imp.1 <- left.imp.1 +
+      cum.matrix.l.1[, j] * (1 - cum.matrix.l.1[, j])
+    right.imp.1 <- right.imp.1 +
+      cum.matrix.r.1[, j] * (1 - cum.matrix.r.1[, j])
+ }
R> impure.1 <- (root - col.left.1 / sum(wt) * left.imp.1 -
+    col.right.1 / sum(wt) * right.imp.1)
R> cbind(splits, t(left.1), t(right.1), impure.1)

  A B C D  0  1  2   3  0  1   2  3   impure.1
1 1 0 0 0  6 35 45  14 87 64  56 93 0.01697083
2 0 1 0 0 36 10 10  44 57 89  91 63 0.01547083
3 1 1 0 0 42 45 55  58 51 54  46 49 0.00303750
4 0 0 1 0 51  0  0  49 42 99 101 58 0.04247083
5 1 0 1 0 57 35 45  63 36 64  56 44 0.00541250
6 0 1 1 0 87 10 10  93  6 89  91 14 0.08003750
7 1 1 1 0 93 45 55 107  0 54  46  0 0.04307083
```

Note that the rows of `splits` make it possible to identify the categories of X belonging to one of the two sets that define each binary partition of the M categories of X. Without loss of generality, for any admissible split, it is assumed that each row of `splits` defines the categories of X associated with the left child node. Furthermore, for any admissible split in `splits`, there are corresponding columns in matrices `left.1` and `right.1` that contain the conditional frequency distributions of Y in the left and right child node, respectively.

According to $i_{OS}(t)$, the maximum decrease in node impurity (0.0800) can be achieved by considering the following binary partition of the categories of X: $\{(A, D), (B, C)\}$. Note that this partition cannot be examined if the predictor is treated as an ordinal one, after ordering its categories according to the conditional mean scores of Y.

To obtain classification trees based on the ordinal impurity function $i_{OS}(t)$, Archer (2010) defines the list `ordinal`, to be passed to `method =` option in `rpart` function. This list consists of three functions named `eval`, `split` and `init`. In particular, the `split` function makes it possible to compute, for any given node t, the decreases in node impurity induced by the binary splits defined according to any predictor. When this function examines an unordered categorical predictor X with M categories, it returns a list containing two objects: `direction` and `goodness`. The `direction` object is a vector of length M containing the category labels of X, ordered according to some criterion. In **rpart**, it is assumed that the best split can be found by treating X as an ordinal predictor, where the ordering of its categories is the one reported in `direction`. The `goodness` object contains the decreases in impurity for the corresponding $M - 1$ splits.

The following instructions are taken from the `ordinal$split` function contained in **rpartOrdinal** and have been applied to the data summarized in `cont.table`. These instructions make it possible to obtain the information that `ordinal$split` passes to the `rpart` function to select the best split:

```
R> class.labs <- names(table((y)))
R> ux <- sort(unique(x))
R> ord <- order(means)
R> n <- length(ord)
R> y <- y[order(x)]
R> x <- sort(x)
R> ymat <- matrix(rep(0, length(x) * no.classes), ncol = no.classes)
R> for(j in 1:no.classes) {
+    ymat[which(y == class.labs[j]), j] <- cumsum(
+      wt[which(y == class.labs[j])])
+ }
R> keep <- matrix(logical(), nrow = dim(ymat)[1], ncol = no.classes)
R> for(i in 2:(dim(ymat)[1] - 1)) {
+    for(j in 1:no.classes) {
+      keep[i, j] <- ifelse((ymat[i, j] == (ymat[i - 1, j] + 1) &
+        ymat[i + 1, j] == 0) | (ymat[i, j] != 0 & ymat[i - 1, j] == 0 &
+        ymat[i + 1, j] == 0), TRUE, FALSE)
+    }
+ }
R> keep[1, ] <- keep[dim(ymat)[1], ] <- rep(TRUE, no.classes)
R> keep.row <- apply(keep, 1, sum)
R> ymat <- ymat[keep.row > 0, ]
R> rx <- x[keep.row > 0]
R> left <- matrix(nrow = dim(ymat)[1], ncol = dim(ymat)[2])
R> left[1, ] <- ymat[1, ]
R> for(i in 2:nrow(left)) {
+    for(j in 1:ncol(left)) {
+            left[i, j] <- ifelse(ymat[i, j] == 0, max(ymat[1:i, j]), ymat[i, j])
+    }
+ }
R> col.left <- apply(left, 1, sum)
```

```
R> impure.l <- left / col.left
R> row.total <- apply(ymat, 2, max)
R> right <- matrix(rep(row.total, dim(left)[1]), ncol = no.classes,
+    byrow = TRUE) - left
R> col.right <- apply(right, 1, sum)
R> impure.r <- right / col.right
R> cum.matrix.l <- t(apply(impure.l, 1, cumsum))
R> cum.matrix.r <- t(apply(impure.r, 1, cumsum))
R> left.imp <- 0
R> right.imp <- 0
R> for(j in 1:(no.classes - 1)) {
+    left.imp <- left.imp + cum.matrix.l[, j] * (1 - cum.matrix.l[, j])
+    right.imp <- right.imp + cum.matrix.r[, j] * (1 - cum.matrix.r[, j])
+  }
R> impure <- (root - col.left / length(y) * left.imp -
+    col.right / length(y) * right.imp)
R> summary.impure <- aggregate(impure, by = list(rx),
+    function(x) max(x, na.rm = TRUE))
R> names(summary.impure) <- c("x", "goodness")
R> list(goodness = summary.impure$goodness[-n], direction = ux[ord])

$goodness
[1] 0.03990128 0.02639743 0.04307083

$direction
[1] D C B A
Levels: A B C D
```

According to these results, the binary partition $\{(A),(B,C,D)\}$ of the categories of X should be used to define the best split, leading to a decrease in node impurity equal to 0.0431.

This example shows that the `ordinal$split` function proposed by Archer (2010) to implement the ordinal impurity function wrongly assumes that the best split can be found after ordering the categories of X by increasing values of the conditional mean scores of Y. Thus, the split selected using `ordinal$split` may be different from the optimal one (as in the example provided).

Furthermore, the reported values for the decrease in node impurity are wrongly computed. According to the values contained in `impure.1`, the decreases in impurity associated with the splits $\{(A,B,C),(D)\}$, $\{(A,B),(C,D)\}$, and $\{(A),(B,C,D)\}$ (which correspond to rows 7, 3 and 1 of the `splits` object) are 0.0431, 0.0030, and 0.0170, respectively.

These latter errors can be explained by examining the `left` and `right` objects, whose rows contain the conditional distributions of Y in 13 different partitions of the sample units into two classes that are examined by `ordinal$split`. The `impure` object provides the corresponding decreases in impurity.

```
R> cbind(left, right, impure)
```

```
                                              impure
 [1,]  1  0   0   0 92 99 101 107 0.002333365
 [2,]  6  0   0   0 87 99 101 107 0.014177855
 [3,]  6 35   0   0 87 64 101 107 0.039901285
 [4,]  6 35  45   0 87 64  56 107 0.026854112
 [5,]  6 35  45  14 87 64  56  93 0.016970833
 [6,] 42 35  45  14 51 64  56  93 0.020776716
 [7,] 42 45  45  14 51 54  56  93 0.026397434
 [8,] 42 45  55  14 51 54  46  93 0.024923292
 [9,] 42 45  55  58 51 54  46  49 0.003037500
[10,] 93 45  55  58  0 54  46  49 0.042531579
[11,] 93 45  55 107  0 54  46   0 0.043070833
[12,] 93 99  55 107  0  0  46   0 0.046261511
[13,] 93 99 101 107  0  0   0   0          NA
```

By carefully inspecting these objects, it is possible to note that some partitions of the sample units examined by the function `ordinal$split` do not correspond to any admissible split defined by X.

Furthermore, the conditional distributions of Y and the decreases in node impurity corresponding to splits defined according to the ordering given by `direction` are contained in rows 11, 9 and 5, while the values contained in `goodness` correspond to rows 3, 7 and 11.

2.4. Optimal tree size selection

As many authors have pointed out (see, for example, Breiman *et al.* 1984; Murthy 1998; Hastie, Tibshirani, and Friedman 2009), one of the main issues in recursive partitioning methods is the selection of the optimal tree size. This issue is particularly crucial when the aim of the analysis is to obtain a prediction rule for future observations. Several techniques have been suggested for obtaining correctly sized trees (see, for example, Niblett and Bratko 1986; Mingers 1987; Quinlan 1993; Cappelli, Mola, and Siciliano 2002; Zhong, Georgiopoulos, and Anagnostopoulos 2008). A comparison among some of these pruning procedures can be found in Esposito, Malerba, and Semeraro (1997). One of the most popular techniques is the pruning procedure proposed by Breiman *et al.* (1984). This procedure is based on the cost-complexity measure

$$R_\alpha(\mathcal{T}) = R(\mathcal{T}) + \alpha \cdot card(\mathcal{T}), \tag{12}$$

where $R(\mathcal{T})$ is usually a measure of the predictive performance of the tree $\mathcal{T}$, $card(\mathcal{T})$ is the complexity of $\mathcal{T}$ (measured by its number of leaves) and α is a tuning parameter ($\alpha > 0$) that governs the trade-off between predictive performance and tree complexity. The choice of the functional form for $R(\mathcal{T})$ depends on the nature of Y.

The `method =` option in `rpart` function allows users to specify not only their own split functions but also their own predictive performance measures. In particular, the function `eval` is used to compute $\hat{s}(t)$ and $R(t)$, which are the predicted value of Y and the predictive performance measure within node t, respectively. $R(\mathcal{T})$ is then obtained as $\sum_{t \in \tilde{\mathcal{T}}} R(t)$, where $\tilde{\mathcal{T}}$ is the set of the terminal nodes of the tree $\mathcal{T}$. The `eval` function associated with the `ordinal` method implemented by Archer (2010) is given by

```
R> ordinal$eval
```

```
function (y, wt, parms)
{
    labels <- names(table(y))
    freq <- tapply(wt, y, sum)
    id <- labels[which(freq == max(freq))]
    newid <- ifelse(length(id) > 1, id[sample(1:length(id), size = 1)], id)
    wmean <- sum(y * wt)/sum(wt)
    rss <- sum(wt * (y - wmean)^2)
    list(label = newid, deviance = rss)
}
<environment: namespace:rpartOrdinal>
```

Note that this function sets $\hat{s}(t)$ equal to the modal category of Y within node t. However, it computes $R(t)$ as the within-node deviance, which is inconsistent with the categorical nature of the response, as it implicitly exploits the average score as predicted value for Y. The same `eval` function is also associated with the `twoing` method implemented by Archer (2010).

Two more suitable predictive performance measures to be used when Y is an ordinal response are the total number of misclassifications $R_{mr}(\mathcal{T})$ (Breiman *et al.* 1984) and the total misclassification cost $R_{mc}(\mathcal{T})$ (Piccarreta 2008). When a set of increasing scores $\{s_1 < s_2 < \ldots < s_J\}$ is assigned to the ordered categories of Y, these measures can be computed as follows:

$$R_{mr}(\mathcal{T}) = \sum_{i=1}^{n} \left[1 - I_{\{s_i\}}(\hat{s}_{i,\mathcal{T}})\right], \tag{13}$$

$$R_{mc}(\mathcal{T}) = \sum_{i=1}^{n} |s_i - \hat{s}_{i,\mathcal{T}}|, \tag{14}$$

where s_i is the observed score for unit i, $\hat{s}_{i,\mathcal{T}}$ is the predicted score for unit i according to the tree $\mathcal{T}$, and $I_{\{s_i\}}(\hat{s}_{i,\mathcal{T}}) = 1$ if $s_i = \hat{s}_{i,\mathcal{T}}$ and 0 otherwise. Note that $\hat{s}_{i,\mathcal{T}} = \hat{s}(t)$ for all units i that are assigned to the terminal node t of $\mathcal{T}$, according to the splitting rules in the tree $\mathcal{T}$.

The predictive performance measures in Equation 13 and Equation 14 can also be exploited to select $\hat{s}(t)$, $\forall t \in \tilde{\mathcal{T}}$, by minimizing them within each node. In particular, $\hat{s}(t)$ is given by the within-node modal score when $R_{mr}(\mathcal{T})$ is used and by the within-node median score when $R_{mc}(\mathcal{T})$ is used.

3. A new package for classification trees with ordinal response

To overcome the problems with **rpartOrdinal** highlighted in Section 2, we have implemented **rpartScore**, a new R package that makes it possible to build classification trees for responses with ordered categories. This package can be employed whenever a set of increasing scores $\{s_1 < s_2 < \ldots < s_J\}$ is assigned to the categories of the response. The simplest choice for these scores is given by $s_j = j$ for $j = 1, \ldots, J$, but this package can be used with any other choice for the scores. General guidelines for choosing scores to be assigned to a categorical variable can be found, for example, in Agresti (2002).

This package uses the general CART framework. In particular, it allows the user to grow classification trees using splitting functions based on the impurity functions $i_{GG1}(t)$ and $i_{GG2}(t)$. Furthermore, it performs the pruning procedure using either $R_{mr}(\mathcal{T})$ or $R_{mc}(\mathcal{T})$.

rpartScore requires the **rpart** package. The main function in **rpartScore** is `rpartScore`, which uses the function `rpart` internally. In particular, `rpartScore` creates a model frame, a list containing parameters that control aspects of the `rpart` fit, and a list of three functions named `eval`, `split` and `init`. These three objects are passed to the function `rpart` through the arguments `model`, `control` and `method`, respectively. Because the function `rpart` does not perform cross-validation when the argument `method` is set equal to a list, the function `rpartScore` also uses the function `xpred.rpart` to internally compute the cross-validated values for the predictive performance measure, used to select the optimal tree size, and their estimated standard errors. Note that this can require long computational times, especially when the pruning procedure requires the evaluation of many subtrees. In this situation, it may be advisable to use either a small number of cross-validation groups or an independent test set.

The functionalities of `rpartScore` are almost the same as those of `rpart`. The main difference is the presence of two arguments (`split` and `prune`) instead of the `method` argument.

The argument `split` controls the impurity function used to grow the classification tree by setting it equal to $i_{GG1}(t)$ (`"abs"` - is the default option) or to $i_{GG2}(t)$ (`"quad"`). Note that when `split = "abs"` and $(s_{j+1} - s_j) = 1$ for $j = 1, \ldots, J-1$, `rpartScore` builds a tree based on the ordinal impurity function $i_{OS}(t)$ or, equivalently, based on the generalized Gini impurity function with linear costs as proposed by Archer (2010) but without producing the unexpected results that are obtained using **rpartOrdinal** (see Section 2.2 and Section 2.3). In particular, the internal call to `rpart` in `rpartScore` uses the function `splitAbs`, which, in contrast to the function `ordinal$split` in **rpartOrdinal**, examines all of the admissible splits for unordered categorical predictors as well. Furthermore, the objects `direction` and `goodness` returned by `splitAbs` are devised such that the internal call to `rpart` selects the optimal split (for further details, see the internal code of `splitAbs`).

Analogously, classification trees based on the generalized Gini impurity function with quadratic costs can be obtained by setting `split = "quad"` and choosing scores such that $(s_{j+1} - s_j) = 1$ for $j = 1, \ldots, J-1$.

The argument `prune` makes it possible to select the predictive performance measure used to prune the classification tree and can take two values: `"mr"` ($R_{mr}(\mathcal{T})$) or `"mc"` ($R_{mc}(\mathcal{T})$ - is the default option). In this way, `rpartScore` overcomes the problem highlighted in Section 2.4, using predictive performance measures that are consistent with the categorical nature of the response.

The function `rpartScore` returns an object belonging to the class `rpart`. Thus, all of the functions contained in the **rpart** package for summarizing, plotting and pruning trees, as well as for making predictions, can also be applied to the objects obtained using `rpartScore`. These objects can also be displayed, summarized and plotted using the functionalities of the **partykit** package (Hothorn and Zeileis 2012). However, since objects created using `rpartScore` contain as response a numeric variable whose values correspond to the scores $\{s_1 < s_2 < \ldots < s_J\}$, the default options in **partykit** functions interpret the response as a quantitative variable. This may lead to summary measures and graphical representations of the dependent variable that are not consistent with its original nature. In order to avoid this problem, suitable options have to be chosen when using **partykit** functions.

3.1. Illustrative example: Low birth weight data set

The use of the `rpartScore` function is illustrated by analyzing the `lowbwt` data set described in Section 2.2. First, numerical scores have to be assigned to the categories of `Category`. The following set of scores is considered: $\{0, 1, 2, 3\}$:

```
R> library("rpartScore")
R> lowbwt$Category.s <- ifelse(lowbwt$bwt <= 2500, 3,
+    ifelse(lowbwt$bwt <= 3000, 2, ifelse(lowbwt$bwt <= 3500, 1, 0)))
```

The following instructions are meant to allow reproducibility of the 10-fold cross-validation results:

```
R> set.seed(16112011)
R> xgroups <- sample(rep(1:10, length = nrow(lowbwt)), nrow(lowbwt),
+    replace = FALSE)
```

As a first example, `rpartScore` is applied using the default options ($i_{GG1}(t)$ as the impurity function and $R_{mc}(\mathcal{T})$ as the predictive performance measure):

```
R> T.abs.mc <- rpartScore(Category.s ~ age + lwt + race + smoke + ptl + ht +
+    ui + ftv, data = lowbwt, xval = xgroups)
```

The object `T.abs.mc` contains information that makes it possible to select the optimal tree size as any object of the class `rpart`. This information can be displayed using the `printcp` and `plotcp` functions:

```
R> plotcp(T.abs.mc)
R> printcp(T.abs.mc)
```

```
rpartScore(formula = Category.s ~ age + lwt + race + smoke +
    ptl + ht + ui + ftv, data = lowbwt, xval = xgroups)

Variables actually used in tree construction:
[1] age   lwt   race  smoke ui

Root node error: 197/189 = 1.0423

n= 189

        CP nsplit rel error  xerror     xstd
1 0.096447      0   1.00000 1.06091 0.042135
2 0.060914      1   0.90355 1.06599 0.036605
3 0.030457      2   0.84264 0.98985 0.055722
4 0.027919      3   0.81218 1.00508 0.045854
5 0.016920      5   0.75635 0.97970 0.029642
6 0.015228      8   0.70558 0.96447 0.030457
7 0.010000      9   0.69036 0.94924 0.031333
```

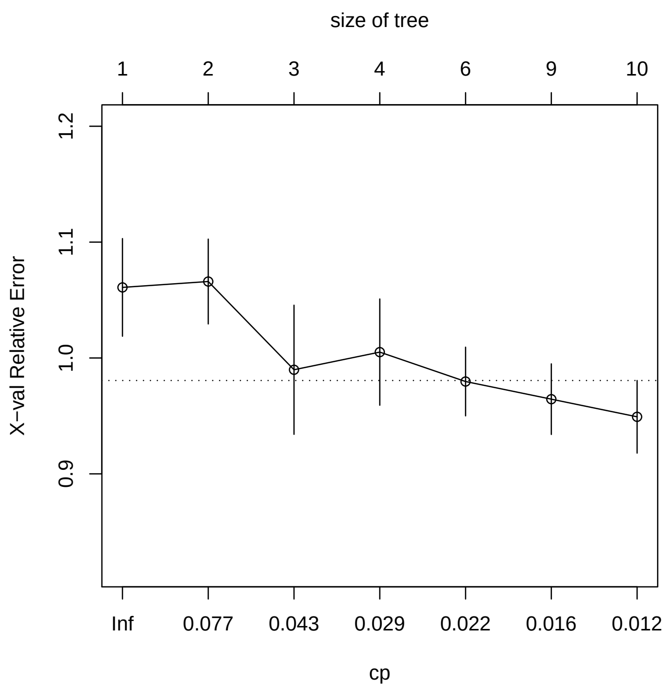

Figure 1: Relative cross-validated total misclassification costs for `T.abs.mc` and the 1-se rule threshold.

According to the 1-se rule (Breiman *et al.* 1984), the optimal tree size is equal to 6 (see Figure 1). The function `prune` makes it possible to extract the corresponding optimal tree, as shown in Figure 2. For each node in the tree, units that satisfy the corresponding splitting rule are assigned to the left child node, while units that do not satisfy that rule are assigned to the right child node.

```
R> T.abs.mc.min.pos <- which.min(T.abs.mc$cptable[, 4])
R> th.1std.rule.mc <- T.abs.mc$cptable[T.abs.mc.min.pos, 4] +
+    T.abs.mc$cptable[T.abs.mc.min.pos, 5]
R> best.1std.rule.mc <- which.max(T.abs.mc$cptable[, 4] < th.1std.rule.mc)
R> best.1std.rule.mc.cp <- T.abs.mc$cptable[best.1std.rule.mc, 1]
R> T.abs.mc.opt <- prune(T.abs.mc, cp = best.1std.rule.mc.cp)
R> plot(T.abs.mc.opt, margin = 0.05)
R> text(T.abs.mc.opt, pretty = TRUE)
```

The following instructions make it possible to perform a similar analysis using $R_{mr}(\mathcal{T})$ instead of $R_{mc}(\mathcal{T})$ as the predictive performance measure:

```
R> T.abs.mr <- rpartScore(Category.s ~ age + lwt + race + smoke +
+    ptl + ht + ui + ftv, data = lowbwt, prune = "mr", xval = xgroups)
R> T.abs.mr.min.pos <- which.min(T.abs.mr$cptable[, 4])
R> th.1std.rule.mr <- T.abs.mr$cptable[T.abs.mr.min.pos, 4] +
+    T.abs.mr$cptable[T.abs.mr.min.pos, 5]
R> best.1std.rule.mr <- which.max(T.abs.mr$cptable[, 4] < th.1std.rule.mr)
```

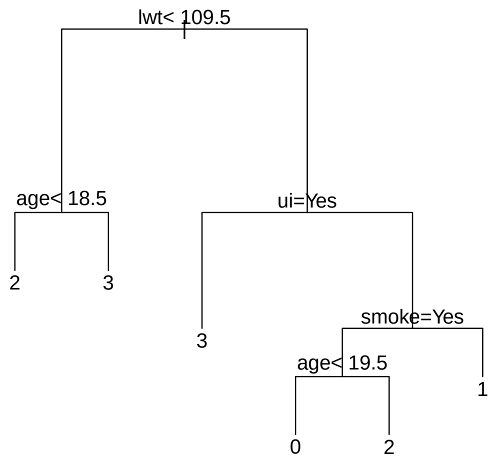

Figure 2: Optimal classification tree using $i_{GG1}(t)$ and $R_{mc}(\mathcal{T})$.

```
R> best.1std.rule.mr.cp <- T.abs.mr$cptable[best.1std.rule.mr, 1]
R> T.abs.mr.opt <- prune(T.abs.mr, cp = best.1std.rule.mr.cp)
R> par(mfrow = c(1, 2))
R> plotcp(T.abs.mr)
R> plot(T.abs.mr.opt, margin = 0.05)
R> text(T.abs.mr.opt, pretty = TRUE)
R> par(mfrow = c(1, 1))
```

As shown in Figure 3, the change in the predictive performance measure leads to the selection of an optimal tree size equal to 4 (thus, to a simpler classification tree). The optimal splitting rules of `T.abs.mr.opt` are present also in `T.abs.mc.opt`, and this is due to the use of the same splitting function for building the two trees. Note that 3 out of 4 terminal nodes of `T.abs.mr.opt` are characterized by the same predicted score.

The classification trees based on the impurity function $i_{GG2}(t)$ and pruned according to either $R_{mc}(\mathcal{T})$ or $R_{mr}(\mathcal{T})$ are displayed in Figure 4 and in Figure 5 and can be obtained as follows:

```
R> T.quad.mc <- rpartScore(Category.s ~ age + lwt + race + smoke +
+    ptl + ht + ui + ftv, split = "quad", data = lowbwt, xval = xgroups)
R> T.quad.mc.min.pos <- which.min(T.quad.mc$cptable[, 4])
R> th.1std.rule.mc <- T.quad.mc$cptable[T.quad.mc.min.pos, 4] +
+    T.quad.mc$cptable[T.quad.mc.min.pos, 5]
R> best.1std.rule.mc <- which.max(T.quad.mc$cptable[, 4] < th.1std.rule.mc)
R> best.1std.rule.mc.cp <- T.quad.mc$cptable[best.1std.rule.mc, 1]
R> T.quad.mc.opt <- prune(T.quad.mc, cp = best.1std.rule.mc.cp)
R> par(mfrow = c(1, 2))
R> plotcp(T.quad.mc)
R> plot(T.quad.mc.opt, margin = 0.05)
R> text(T.quad.mc.opt, pretty = TRUE)
R> par(mfrow = c(1, 1))
R> T.quad.mr <- rpartScore(Category.s ~ age + lwt + race + smoke + ptl + ht +
+    ui + ftv, split = "quad", prune = "mr", data = lowbwt, xval = xgroups)
```

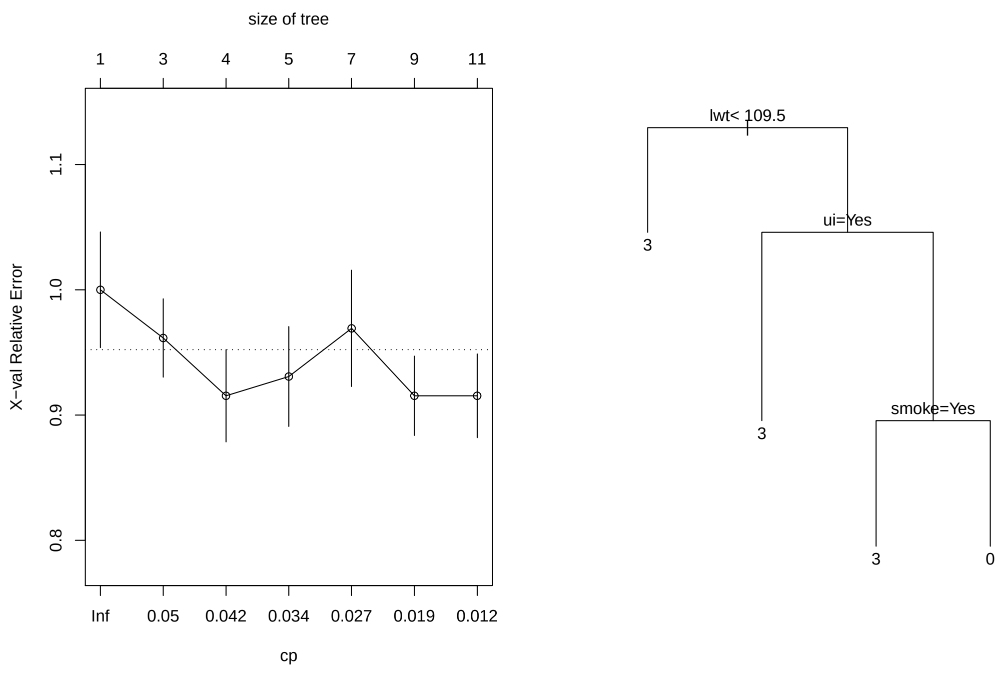

Figure 3: Results using $i_{GG1}(t)$ and $R_{mr}(\mathcal{T})$. Left panel: Relative cross-validated total number of misclassifications for `T.abs.mr` and the 1-se rule threshold. Right panel: Optimal classification tree.

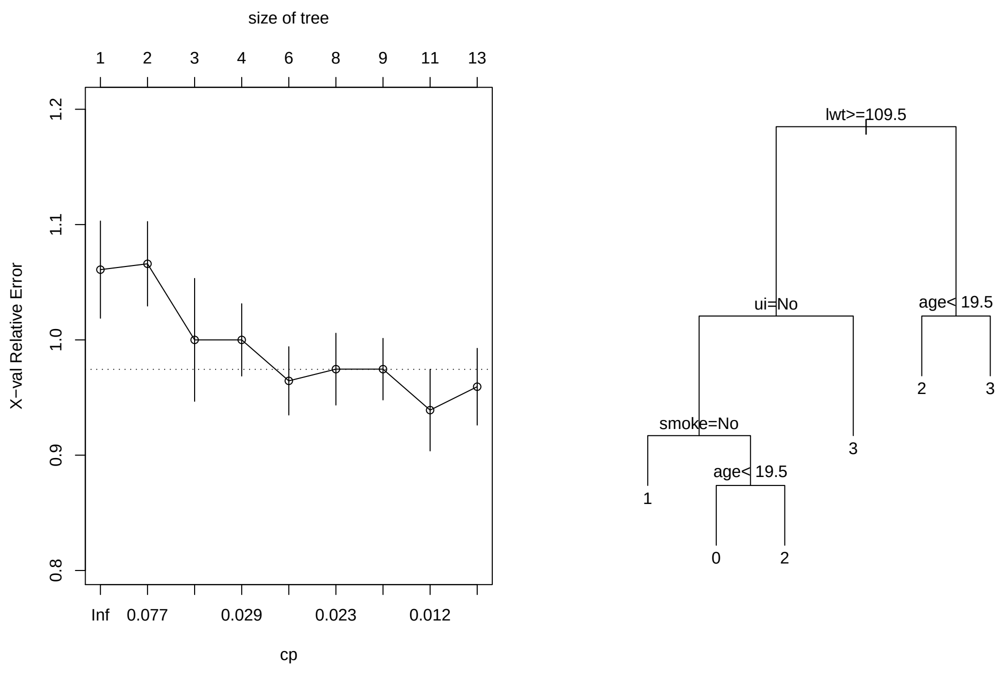

Figure 4: Results using $i_{GG2}(t)$ and $R_{mc}(\mathcal{T})$. Left panel: Relative cross-validated total misclassification costs for `T.quad.mc` and the 1-se rule threshold. Right panel: Optimal classification tree.

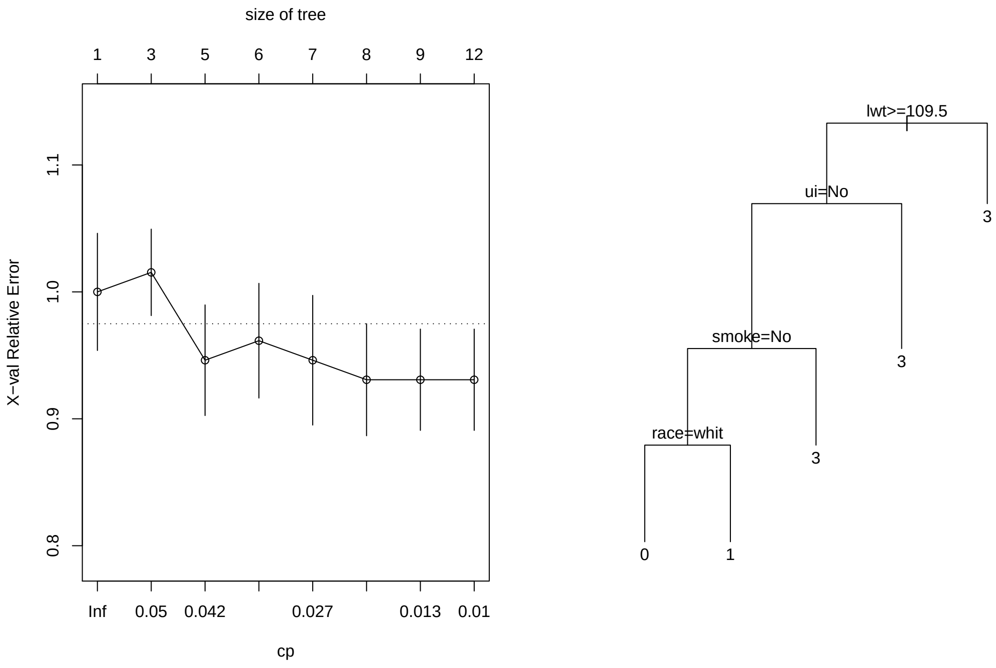

Figure 5: Results using $i_{GG2}(t)$ and $R_{mr}(\mathcal{T})$. Left panel: Relative cross-validated total number of misclassifications for `T.quad.mr` and the 1-se rule threshold. Right panel: Optimal classification tree.

```
R> T.quad.mr.min.pos <- which.min(T.quad.mr$cptable[, 4])
R> th.1std.rule.mr <- T.quad.mr$cptable[T.quad.mr.min.pos, 4] +
+    T.quad.mr$cptable[T.quad.mr.min.pos, 5]
R> best.1std.rule.mr <- which.max(T.quad.mr$cptable[, 4] < th.1std.rule.mr)
R> best.1std.rule.mr.cp <- T.quad.mr$cptable[best.1std.rule.mr, 1]
R> T.quad.mr.opt <- prune(T.quad.mr, cp = best.1std.rule.mr.cp)
R> par(mfrow = c(1, 2))
R> plotcp(T.quad.mr)
R> plot(T.quad.mr.opt, margin = 0.05)
R> text(T.quad.mr.opt,pretty = TRUE)
R> par(mfrow = c(1, 1))
```

In this example, the choice of the splitting function has little effect on the tree topology. The two trees `T.abs.mc.opt` and `T.quad.mc.opt`, pruned using $R_{mc}(\mathcal{T})$, differ only in the thresholds used in one splitting rule: `age<18.5` and `age<19.5`, respectively (see Figure 2 and Figure 4). Similarly, `T.quad.mr.opt` is identical to `T.abs.mr.opt`, except for the presence of an additional splitting rule and, thus, an additional terminal node (see Figure 3 and Figure 5).

3.2. Choosing splitting function and predictive performance measure

The function `rpartScore` allows the user to obtain $K = 4$ different trees, according to the choice of the splitting function and the predictive performance measure. In the illustrative example shown in Section 3.1, the main differences among these trees can be attributed to the choice of the predictive performance measure. The identification of the best combination of the splitting function and the predictive performance measure is beyond the scope of this paper. However, by exploiting the theoretical framework described in Hothorn, Leisch, Zeileis, and Hornik (2005), a simulation experiment with a dependent $K = 4$ samples design has been performed on the `lowbwt` data set, to evaluate whether significant differences exist among the $K = 4$ solutions.

Differences have been measured in terms of ordinal association between predicted and observed scores, using Somers' d measure (Agresti 2002). This is an asymmetric measure of association, which in this experiment has been computed as the difference between the proportions of concordant and discordant pairs of units with respect to the observed and predicted scores out of those pairs that are untied with respect to the observed scores. The choice of this measure instead of a symmetric one, such as Goodman and Kruskal's γ or Kendall's τ_b, is due to the fact that these symmetric measures are not defined when the predicted score is constant for all of the units (which happens whenever the optimal tree size is 1). The function `somersXY`, which computes values of Somers' d measure for a set of observed (argument `x`) and predicted (argument `y`) scores, is implemented as follows:

```
R> somersXY <- function(x, y) {
+    out <- table(x, y)
+    r <- nrow(out)
+    c <- ncol(out)
+    p <- matrix(rep(0, r * c), ncol = c)
+    q <- matrix(rep(0, r * c), ncol = c)
+    somers.num <- 0
+    if(c > 1) {
+      for(i in 2:r) {
+        for(j in 2:c) {
+          p[i, j] <- sum(out[i, j] * sum(out[1:(i - 1), 1:(j - 1)]))
+        }
+      }
+      for(i in 2:r) {
+        for(j in 1:(c - 1)) {
+          q[i, j] <- sum(out[i, j] * sum(out[1:(i - 1), (j + 1):c]))
+        }
+      }
+      somers.num <- (sum(p) - sum(q))
+    }
+    n <- length(x)
+    out.x <- table(x)
+    somers.den <- 0.5 * (n * (n - 1) - sum(out.x * (out.x - 1)))
+    somers.num / somers.den
+  }
```

A training set (containing 131 of the 189 units in the `lowbwt` data set, selected using a stratified random scheme) has been employed to grow and prune four classification trees (one for each combination of the splitting function and the predictive performance measure) using 10-fold cross-validation and the 1-se rule. These trees have been used to predict the scores of the remaining 58 units (the test set). The value of Somers' d has been computed for the test set for each of the four trees. To account for sampling variability, $B = 100$ different pairs of training and test sets have been considered, resulting in a total of 400 values for Somers' d. Stratified sampling has been performed using the R package **sampling** (Tillé and Matei 2012).

```
R> set.seed(16112011)
R> b.rep <- 100
R> splitting <- c("abs", "quad")
R> pruning <- c("mc", "mr")
R> sp.comb <- expand.grid(splitting = splitting, pruning = pruning)
R> sp.comb.rep <- sp.comb[rep(1:4, b.rep), ]
R> somers.B <- c()
R> library("sampling")
R> size.cat <- as.numeric(trunc(table(lowbwt$Category.s) * 0.7))
R> size.cat.ord <- size.cat[order(0:3, decreasing = TRUE)]
R> n.train <- sum(size.cat)
R> B <- nrow(sp.comb.rep)
R> for(b in 1:B) {
+    if (b %% 4 == 1) train <- strata(lowbwt, "Category.s",
+      size = size.cat.ord, method = "srswor")
+    lowbwt.train <- lowbwt[c(train$ID_unit), ]
+    lowbwt.test <- lowbwt[-c(train$ID_unit), ]
+    if (b %% 4 == 1) xgroups <- sample(rep(1:10,
+      length = nrow(lowbwt.train)), nrow(lowbwt.train), replace = FALSE)
+    fit <- rpartScore(Category.s ~ age + lwt + race + smoke + ptl + ht +
+      ui + ftv, data = lowbwt.train,
+      split = as.character(sp.comb.rep[b, 1]),
+      prune = as.character(sp.comb.rep[b, 2]), xval = xgroups)
+    xerror.min.pos <- which.min(fit$cptable[, 4])
+    th.1std.rule <- fit$cptable[xerror.min.pos, 4] +
+      fit$cptable[xerror.min.pos, 5]
+    best.1std.rule <- which.max(fit$cptable[, 4] < th.1std.rule)
+    best.1std.rule.cp <- fit$cptable[best.1std.rule, 1]
+    fit.best <- prune(fit, cp = best.1std.rule.cp)
+    test.pred <- predict(fit.best, newdata = lowbwt.test)
+    somers.B[b] <- somersXY(lowbwt.test$Category, test.pred)
+  }
```

The global hypothesis of equality of the four solutions has been tested using Friedman's nonparametric rank test for repeated measurements in a randomized complete block design (see, for example, Hollander and Wolfe 1999, p. 272), treating each of the 100 training sets as a block. The test statistic and the corresponding asymptotic p value have been computed using the `friedman_test` function in the **coin** package (Hothorn, Hornik, van de Wiel, and Zeileis 2008).

```
R> settings <- factor(rep(1:4, b.rep),
+    labels = c("abs+mc", "quad+mc", "abs+mr", "quad+mr"))
```

```
R> dataset <- factor(rep(1:b.rep, each = 4))
R> results <- data.frame(somers.B)
R> results$settings <- settings
R> results$dataset <- dataset
R> library("coin")
R> friedman_test(somers.B ~ settings | dataset, data = results)

        Asymptotic Friedman Test

data:  somers.B by
         settings (abs+mc, quad+mc, abs+mr, quad+mr)
         stratified by dataset
chi-squared = 4.7503, df = 3, p-value = 0.191
```

According to these results, the global equality hypothesis is not rejected at a 0.05 significance level (or lower than 0.05). Thus, for the `lowbwt` data set, there are not significant differences in the results obtained using different combinations of the splitting function and the predictive performance measure.

4. Concluding remarks

This paper has introduced **rpartScore**, a new R package that makes it possible to build classification trees for ordinal responses within the general CART framework. This package has been implemented to overcome some unexpected results that may arise using the package **rpartOrdinal**. Classification trees for ordinal responses can also be obtained through the **party** package, which recursively partitions data using a two-step procedure, based on association tests between the response and each predictor (Hothorn, Hornik, and Zeileis 2006).

Because the function `rpartScore` allows the user to obtain four different trees, it may be useful to establish general guidelines for choosing the best combination of the splitting function and the predictive performance measure to be used. This remains an open issue that will be addressed in our future research. The study described in Section 3.2 will be extended to other artificial and real data sets, with particular attention to situations in which the categories of the response are unbalanced. Furthermore, along with Somers's d, other measures able to evaluate the global quality of a tree will be considered.

Finally, it is worth noting that this package could also be employed to build regression trees, when the response Y is a quantitative variable, by treating the observed values of Y as scores. In this case, it is advisable to choose $R_{mc}(\mathcal{T})$ as the predictive performance measure. It may be interesting to evaluate the properties of the resulting regression trees, with a particular focus on their robustness in the presence of outlying values for Y, to establish whether they could represent a valid alternative to standard least squares regression trees and to regression trees based on M-estimators (Galimberti, Pillati, and Soffritti 2007, 2011).

References

Agresti A (2002). *Categorical Data Analysis*. 2nd edition. John Wiley & Sons, Hoboken.

Archer KJ (2010). "**rpartOrdinal**: An R Package for Deriving a Classification Tree for Predicting an Ordinal Response." *Journal of Statistical Software*, **34**(7), 1–17. URL http://www.jstatsoft.org/v34/i07/.

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

Cappelli C, Mola F, Siciliano R (2002). "A Statistical Approach to Growing a Reliable Honest Tree." *Computational Statistics & Data Analysis*, **38**, 285–299.

Esposito F, Malerba D, Semeraro G (1997). "A Comparative Analysis of Methods for Pruning Decision Trees." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **19**, 476–491.

Galimberti G, Pillati M, Soffritti G (2007). "Robust Regression Trees Based on *M*-Estimators." *Statistica*, **LXVII**, 173–190.

Galimberti G, Pillati M, Soffritti G (2011). "Notes on the Robustness of Regression Trees Against Skewed and Contaminated Errors." In S Ingrassia, R Rocci, M Vichi (eds.), *New Perspectives in Statistical Modeling and Data Analysis*, Studies in Classification, Data Analysis, and Knowledge Organization, pp. 255–263. Springer-Verlag, Berlin.

Galimberti G, Soffritti G (2012). "Tree-Based Methods and Decision Trees." In RS Kenett, S Salini (eds.), *Modern Analysis of Customer Surveys: With Applications Using R*, pp. 283–308. John Wiley & Sons, New York.

Galimberti G, Soffritti G, Di Maso M (2012). ***rpartScore**: Classification Trees for Ordinal Responses*. R package version 0.1.1, URL http://CRAN.R-project.org/package=rpartScore.

Hastie T, Tibshirani R, Friedman JH (2009). *The Elements of Statistical Learning*. 2nd edition. Springer-Verlag, New York.

Hollander M, Wolfe DA (1999). *Nonparametric Statistical Methods. 2nd Edition*. John Wiley & Sons, New York.

Hothorn T, Hornik K, van de Wiel M, Zeileis A (2008). "Implementing a Class of Permutation Tests: The **coin** Package." *Journal of Statistical Software*, **28**(8), 1–23. URL http://www.jstatsoft.org/v28/i08/.

Hothorn T, Hornik K, Zeileis A (2006). "Unbiased Recursive Partitioning: A Conditional Inference Framework." *Journal of Computational and Graphical Statistics*, **15**(3), 651–674.

Hothorn T, Leisch F, Zeileis A, Hornik K (2005). "The Design and Analysis of Benchmark Experiments." *Journal of Computational and Graphical Statistics*, **14**(3), 675–699.

Hothorn T, Zeileis A (2012). ***partykit**: A Toolkit for Recursive Partytioning*. R package version 0.1-3, URL http://CRAN.R-project.org/package=partykit.

Mingers J (1987). "Expert Systems – Rule Induction with Statistical Data." *Journal of the Operational Research Society*, **28**, 39–47.

Murthy SK (1998). "Automatic Construction of Decision Trees from Data: A Multi-Disciplinary Survey." *Data Mining and Knowledge Discovery*, **2**, 345–389.

Niblett T, Bratko I (1986). "Learning Decision Rules in Noisy Domains." In *Proceedings in Expert Systems 86*, pp. 25–34. Cambridge University Press.

Piccarreta R (2001). "A New Measure of Nominal-Ordinal Association." *Journal of Applied Statistics*, **28**, 107–120.

Piccarreta R (2008). "Classification Trees for Ordinal Variables." *Computational Statistics*, **23**, 407–427.

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

R Development Core Team (2012). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/.

Therneau TM, Atkinson EJ (1997). "An Introduction to Recursive Partitioning Using **rpart** Routines." *Technical Report 61*, Section of Biostatistics, Mayo Clinic, Rochester. URL http://www.mayo.edu/hsr/techrpt/61.pdf.

Therneau TM, Atkinson EJ (2011). ***rpart**: Recursive Partitioning*. R package version 3.1-51, URL http://CRAN.R-project.org/package=rpart.

Tillé Y, Matei A (2012). ***sampling**: Survey Sampling*. R package version 2.5, URL http://CRAN.R-project.org/package=sampling.

Zhong M, Georgiopoulos M, Anagnostopoulos GC (2008). "A K-Norm Pruning Algorithm for Decision Tree Classifiers Based on Error Rate Estimation." *Machine Learning*, **71**, 55–88.

Affiliation:

Giuliano Galimberti, Gabriele Soffritti
Dipartimento di Scienze Statistiche
Università di Bologna
40126 Bologna, Italia
E-mail: giuliano.galimberti@unibo.it, gabriele.soffritti@unibo.it

Matteo Di Maso
Facoltà di Scienze Statistiche
Università di Bologna
40126 Bologna, Italia
E-mail: matteo.dimaso@yahoo.it

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/

Volume 47, Issue 10 *Submitted:* 2010-11-23
April 2012 *Accepted:* 2012-04-13