Tutorial on How Simulating Right-Censored Data, Fitting Survival Trees, Evaluating Survival Trees Performance with R

Ambra Macis

In this document an example for simulating right-censored data, growing survival trees and evaluating their performance is reported, together with the related results.

1. Random generation of the set of covariates

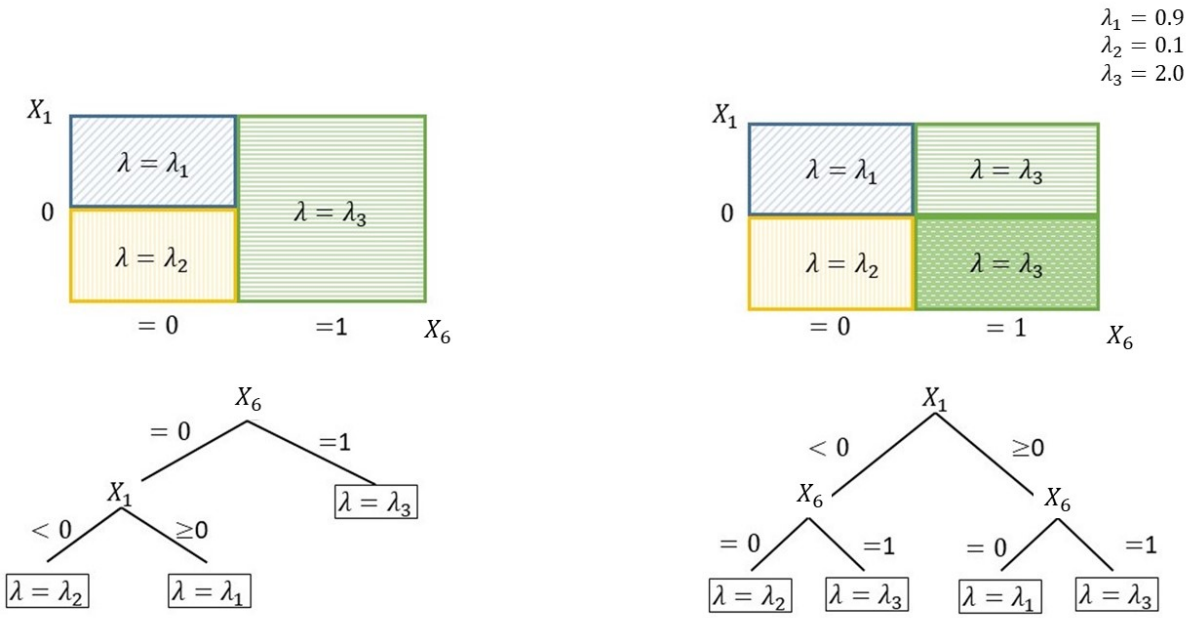
The code below randomly generates 10 covariates: the first half followed a Standard Normal distribution (X_1, X_2, X_3, X_4 and X_5) and the second half (X_6, X_7, X_8, X_9 and X_{10}) a Bernoulli distribution with different probability parameters p ranging respectively from 0.3 to 0.7 with steps of 0.1. For example, in a medical context the covariates could be clinical and socio-demographic information about the patient.

```
n_rv <- 10                      # Number of covariates
prob_rvar <- seq(0.3,0.7,0.1) # Parameters for the Bernoulli r.v.
N <- 1000                       # Sample size

library(mvtnorm)
set.seed(70522)
Norm_RV <- rmvnorm(n=N, mean=rep(0,n_rv/2), sigma=diag(n_rv/2))

library(MultiRNG)
set.seed(70522)
Ber_RV <- draw.correlated.binary(no.row=N, d=n_rv/2, prop.vec=prob_rvar,
                                 corr.mat=diag(n_rv/2))

cov <- as.data.frame(cbind(Norm_RV, Ber_RV))
colnames(cov) <- paste0("X", 1:n_rv)
```

2. Definition of the theoretical recursive partitioning model

In this example a Data Generating process with a tree structure was used. For building a recursive partitioning model, different distributions for survival times in the terminal nodes must be assumed (one for each leave of the tree). The tree was characterized by three terminal nodes with the following partitions and distribution of the survival times:

- First leave: $X6 = 1 : T_i \sim Exp(2)$;
- Second leave: $X6 = 0$ and $X1 \geq 0$: $T_i \sim Exp(0.9)$;
- Third leave: $X6 = 0$ and $X1 < 0 : T_i \sim Exp(0.1)$.



So, I hypothesized that the first node had the highest hazard (equal to $\lambda = 2$), the second one had an intermediate hazard, while the third one had the lowest risk of experiencing the event. In a medical context the survival outcome could be the relapse or the death of the patient. However, survival analysis can be carried out also in other settings; examples are the work setting (e.g. dismissal, failure of a firm) or the sport context (e.g. to evaluate the minutes needed to reach a given threshold of points).

```
node1 <- cov[,"X6"]==1
node2 <- cov[,"X6"]==0 & cov[,"X1"]>=0
node3 <- cov[,"X6"]==0 & cov[,"X1"]<0
```

Finally, we fixed the length of the follow-up equal to 3 years:

```
maxtime <- 3
tvec <- seq(1,maxtime,0.1)
```

During a simulation study many datasets of different sample sizes should be examined. Here, for simplicity, only one dataset has been generated.

3. Survival data simulation and definition of the training and test sets

I used the *simsurv* package to simulate survival data. This function requires the following arguments: (i) the distribution assumed for survival times; (ii) the corresponding parameters; (iii) a dataframe **x** with the set of covariates (if not - as in the recursive partitioning setting- only an *id* column is required); (iv) **maxt**, that corresponds to the length of the follow-up; and (v) the **seed** to use in order to get reproducible results.

```
simdata <- as.data.frame(matrix(data=NA, nrow=N, ncol=2))
colnames(simdata) <- c("eventtime","status")
library(simsurv)
# Node1
simdata[node1,1:2] <- simsurv(dist="exponential", lambda=2, maxt=maxtime,
                              x=as.data.frame(1:sum(node1)), seed=7052022)[,2:3]
# Node2
```

```
simdata[node2,1:2] <- simsurv(dist="exponential", lambda=0.9, maxt=maxtime,
                              x=as.data.frame(1:sum(node2)), seed=7052022)[,2:3]
# Node3
simdata[node3,1:2] <- simsurv(dist="exponential", lambda=0.1, maxt=maxtime,
                              x=as.data.frame(1:sum(node3)), seed=7052022)[,2:3]

data <- cbind(simdata,cov) # Dataset with survival data and covariates
data[,8:12] <- lapply(data[,8:12], as.factor)
head(simdata)
```

```
##   eventtime status
## 1 0.2534565      1
## 2 3.0000000      0
## 3 0.1797119      1
## 4 3.0000000      0
## 5 0.5632367      1
## 6 0.3993597      1
```

It can be seen that *simsurv* provides as outcome a dataset with three columns (*id*, *eventtime* and *status*). The *eventtime* column represents the time-to-event/censoring time, while the *status* column indicates the occurrence ($status = 1$) or not ($status = 0$) of the event.

```
str(data)
```

```
## 'data.frame':    1000 obs. of  12 variables:
##  $ eventtime: num  0.253 3 0.18 3 0.563 ...
##  $ status   : int  1 0 1 0 1 1 0 0 1 1 ...
##  $ X1       : num  -0.9326 -0.1384 0.0249 -0.2564 0.1795 ...
##  $ X2       : num  -1.322 -0.859 0.519 -0.833 0.719 ...
##  $ X3       : num  -2.44733 -1.37352 -0.00419 2.04342 -0.83023 ...
##  $ X4       : num  -0.71 1.142 0.369 0.633 -1.432 ...
##  $ X5       : num  -0.773 -0.328 0.214 -0.902 -0.242 ...
##  $ X6       : Factor w/ 2 levels "0","1": 2 1 2 1 1 1 1 1 2 1 ...
##  $ X7       : Factor w/ 2 levels "0","1": 2 2 1 2 1 1 1 2 1 1 ...
##  $ X8       : Factor w/ 2 levels "0","1": 2 2 2 2 1 1 2 1 1 1 ...
##  $ X9       : Factor w/ 2 levels "0","1": 1 2 2 1 2 1 2 1 2 2 ...
##  $ X10      : Factor w/ 2 levels "0","1": 2 2 2 1 2 1 2 1 2 1 ...
```

Once data has been simulated the training and test set can be randomly generated, taking a similar percentage of events in the two sets.

```
set.seed(70522)
library(caret)
trainset <- createDataPartition(y=data$status, p=0.5, list=F)
data_train <- data[trainset,]
data_test  <- data[-trainset,]
dim(data_train)
```

```
## [1] 500  12
```

```
dim(data_test)
```

```
## [1] 500  12
```

```
table(data_train$status)
```

```
##
##   0   1
```

```
## 143 357
```

```
table(data_test$status)
```

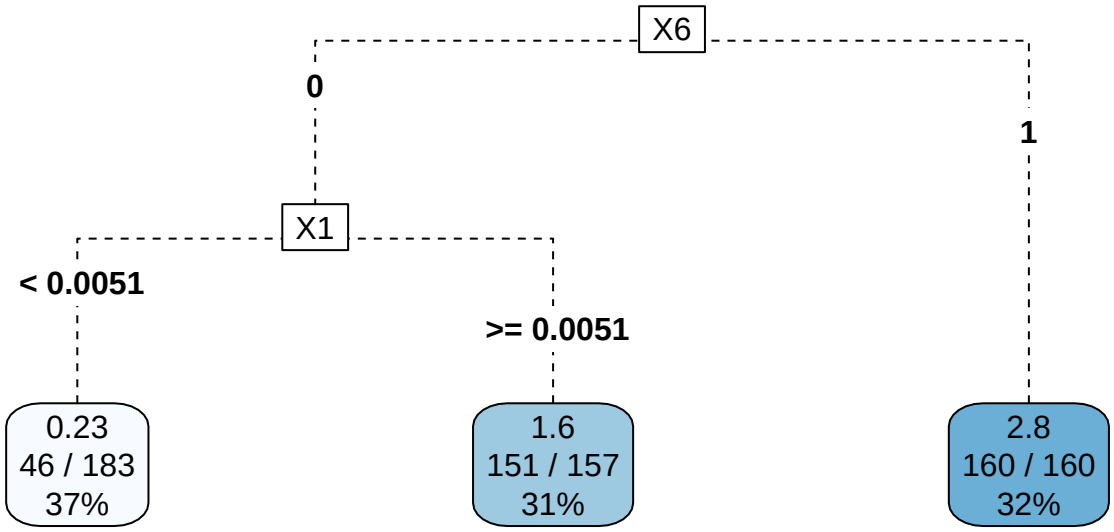
```
##
##   0   1
## 140 360
```

4. Fitting the model on the training set

Relative Risk Trees

The *rpart* function uses Poisson trees for building a Relative Risk Tree (RRT). The required arguments are: (i) a survival formula, i.e. a formula with a *Surv* object (in the *survival* package) as dependent variable; (ii) a training set; and (iii) usual control options (e.g. **minsplit**, minimum number of observations in a node; **xval**, number of cross-validations). The resulting tree can then be pruned with the usual cost-complexity algorithm used by CART (*prune* function) using a suitable value for the complexity parameter (**cp**), e.g. the one that leads to the lowest cross-validation error.

```
library(survival)
library(rpart)
RRT <- rpart(formula=Surv(eventtime, status)~., data=data_train,
             control=rpart.control(usesurrogate=2, minsplit=20, xval=10))
# Pruning RRT
minerr <- which.min(RRT$cptable[,"xerror"])
bestcp <- RRT$cptable[minerr,"CP"]
pruned_RRT <- prune(tree=RRT, cp=bestcp)
# Plotting RRT
library(rpart.plot)
rpart.plot(x=pruned_RRT, type=5, tweak=1.0, gap=0.02, branch.lty=2)
```



It can be seen that after pruning, the resulting tree is a tree with three leaves corresponding with the right tree theoretical partitions. In each leave the following information can be retrieved:

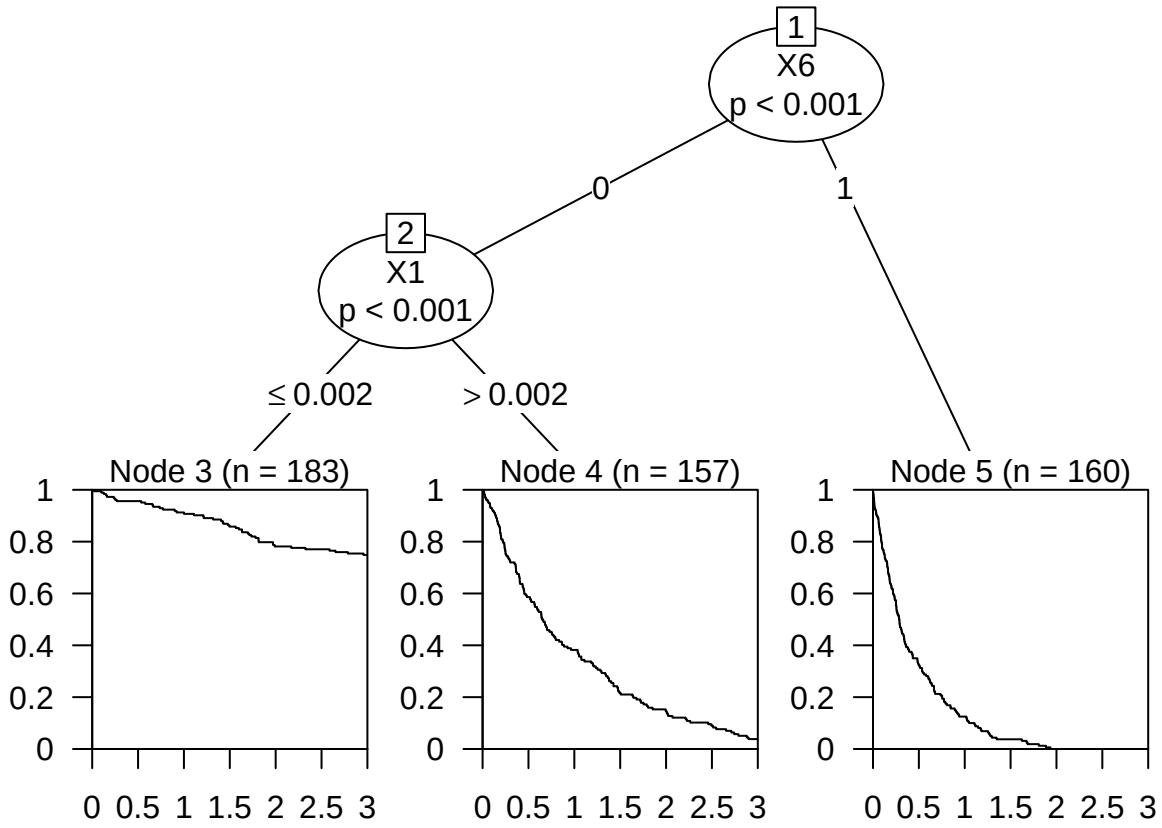
- the relative hazard (e.g. for the left terminal node equal to 0.23). i.e. the hazard of the node relative to that in the root node;
- the ratio between events and the number of observations included in that node (e.g. 46 and 183 respectively in the left terminal node);
- the percentage of observations included in that node (e.g. the left node contains the 37 of the entire training sample).

In particular, it can be seen that the estimated relative hazards in each node are coherent with the theoretical ones. So, for example, the observations which have a value of $X6$ equal to 0 and a value of $X1 < 0.0051$ have the lower hazard with respect to the overall sample.

Conditional Inference Trees with *ctree*

The *ctree* function in *partykit* allows to grow a Conditional Inference Tree through the *ctree* algorithm (CIT). It requires as arguments: (i) a survival formula, (ii) a training set and (iii) some control options as the test statistics for variable and splitpoint selection (**teststat** and **splitstat** respectively), the significance level for variable selection (**alpha**) and the minimum sum of weights in a node in order to be considered for splitting (**minsplit**). Many other options can be set, as shown in the help of the function.

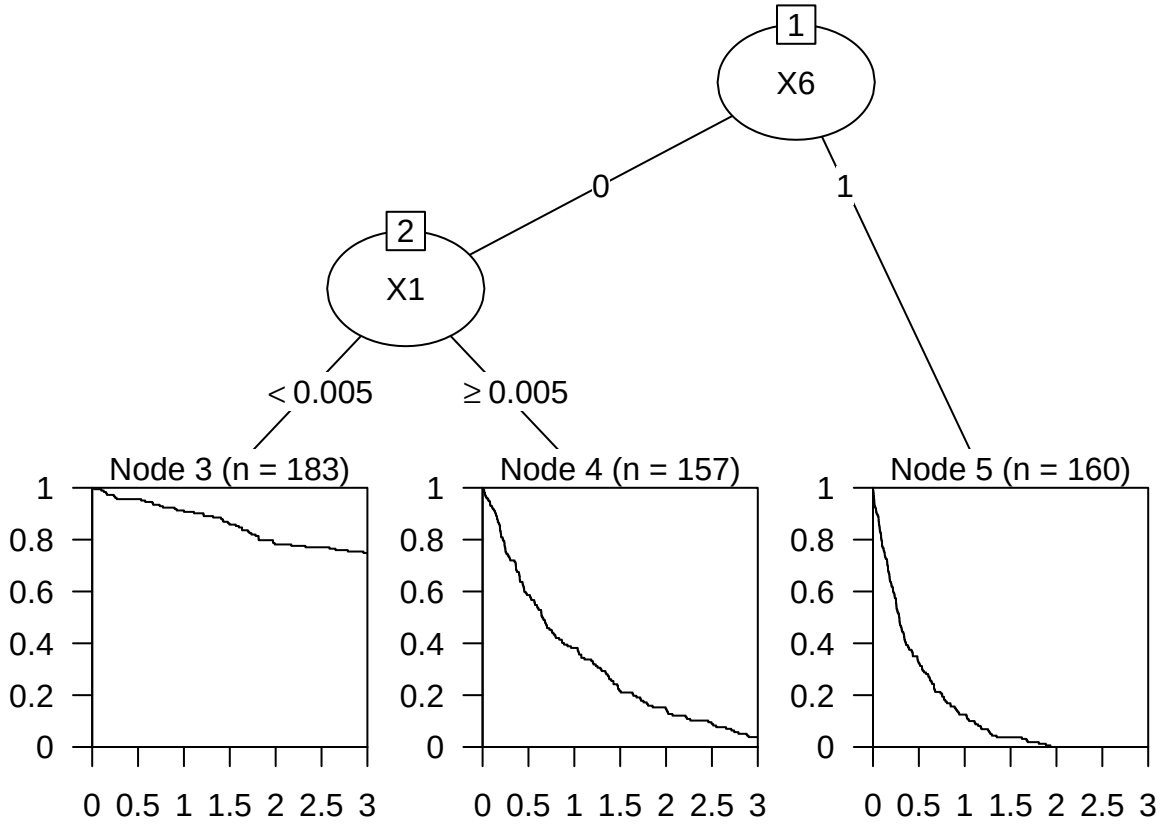
```
library(partykit)
CIT <- ctree(formula=Surv(eventtime, status)~., data=data_train,
             control=ctree_control(minsplit=20))
# Plotting CIT
plot(CIT)
```



The plot of a CIT provides for each terminal node the number of subjects included in the model and the estimated Kaplan-Meier survival curve. It can be seen that even in this case the model identifies the right partitions. In particular, the observations in Node 3 have the lower hazard, followed by the observations in Node 4 who have an intermediate level of risk and the observations in Node 5 who have the highest risk of experiencing the event (and consequently the lowest survival probability).

The same graphical representation of CITs can be obtained for RRTs coercing the *rpart* object into a *party* one:

```
library(partykit)
plot(as.party(pruned_RRT))
```



Conditional Inference Trees with *SurvCART*

SCTs can be grown using the *SurvCART* function in *LongCART* package. It requires that the training data includes an *"id"* column and that categorical variables are numerically coded. Once the training dataset has been formatted following these requirements, trees can be grown. The input arguments are: (i) the data on which training the algorithm (**data**); (ii) the name of the *"id"* variable (**patid**); (iii) the name of the time variable (**timevar**); (iii) the name of the status variable (**censorvar**); (iv) a vector including the names of the partitioning variables to use (**gvars**) and (v) a vector indicating the type of partitioning variable (**tgvars**). This argument assumes value 1 if the corresponding variable in **gvars** is continuous and 0 if the variable is categorical. Moreover, it is necessary to specify (vi) the assumed distribution for survival times (**time.dist**). By default censoring is considered homogeneous (as in this hypothesized setting) and the related argument, **censdist**, is set equal to NA. However, it is also possible to set a particular distribution for censoring when the interest is also in censoring heterogeneity. Possible assumptions for both the time-to-event and censoring distributions are *"exponential"*, *"weibull"*, *"lognormal"* and *"normal"*. Finally, control options can be set, as the significance level of the parameter instability test (**alpha**), and the minimum number of observations in a node (**minsplit**).

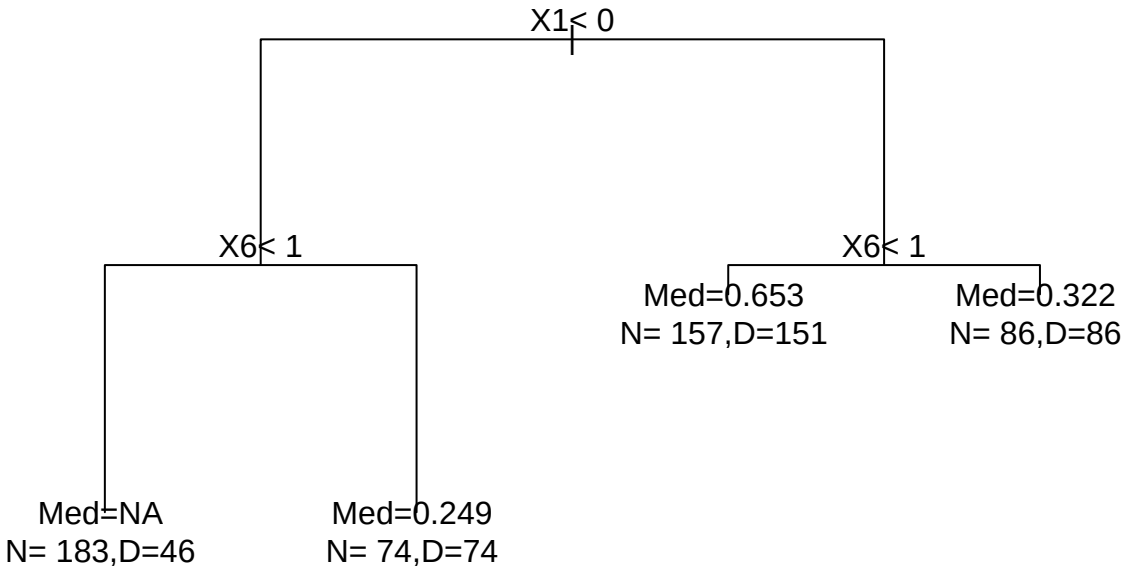
```
# Data pre-processing
data_train_SCT <- data_train
id <- 1:nrow(data_train_SCT)
data_train_SCT <- cbind(data_train_SCT,id)
data_train_SCT[,paste0("X",6:10)] <-
                        lapply(data_train_SCT[,paste0("X",6:10)], as.numeric)
data_train_SCT[, paste0("X",6:10)] <- data_train_SCT[, paste0("X",6:10)]-1

# Model Fitting
```

```
library(LongCART)
SCT <- SurvCART(data=data_train_SCT, patid="id", timevar="eventtime",
                censorvar="status", gvars=paste0("X",1:10),
                tgvars=c(1,1,1,1,1,0,0,0,0,0), time.dist="exponential",
                cens.dist="NA", minsplit=20)
```

```
##   ID   n   D median.T median.C loglik    AIC var index p (Instability) improve
## 1  1 500 357    0.928        3 -592.9 1187.8  X1     0           0.000   117.9
## 2  2 257 120       NA        3 -291.8  585.7  X6     1           0.000   129.1
## 3  4 183  46       NA        3 -153.0  308.1 X10    NA           0.068      NA
## 4  5  74  74    0.249       NA   -9.7   21.5  X2    NA           0.073      NA
## 5  3 243 237    0.488        3 -183.2  368.4  X6     1           0.000    15.3
## 6  6 157 151    0.653        3 -149.8  301.6  X1    NA           0.930      NA
## 7  7  86  86    0.322       NA  -18.1   38.3  X9    NA           0.837      NA
##   Terminal
## 1    FALSE
## 2    FALSE
## 3     TRUE
## 4     TRUE
## 5    FALSE
## 6     TRUE
## 7     TRUE
```

A first output of the tree is the *Treeout* matrix that provides summary information of tree fitting for each node (terminal and non-terminal ones). It contains the identity number of each node (*ID*), the number of observations in each node (n), the number of events observed in the node (D), the median survival and the median censoring time at each node (*median.T*, *median.C*). It also includes the log-likelihood at each node (*loglik*), the AIC at the node (*AIC*), the splitting variable (*var*), the cut-off value of each splitting variable (*index*), the p-vaue for the parameter instability test (*p(Instability)*), the improvement in deviance given by the split (*improve*) and an indicator variable of terminal node (*Terminal*).

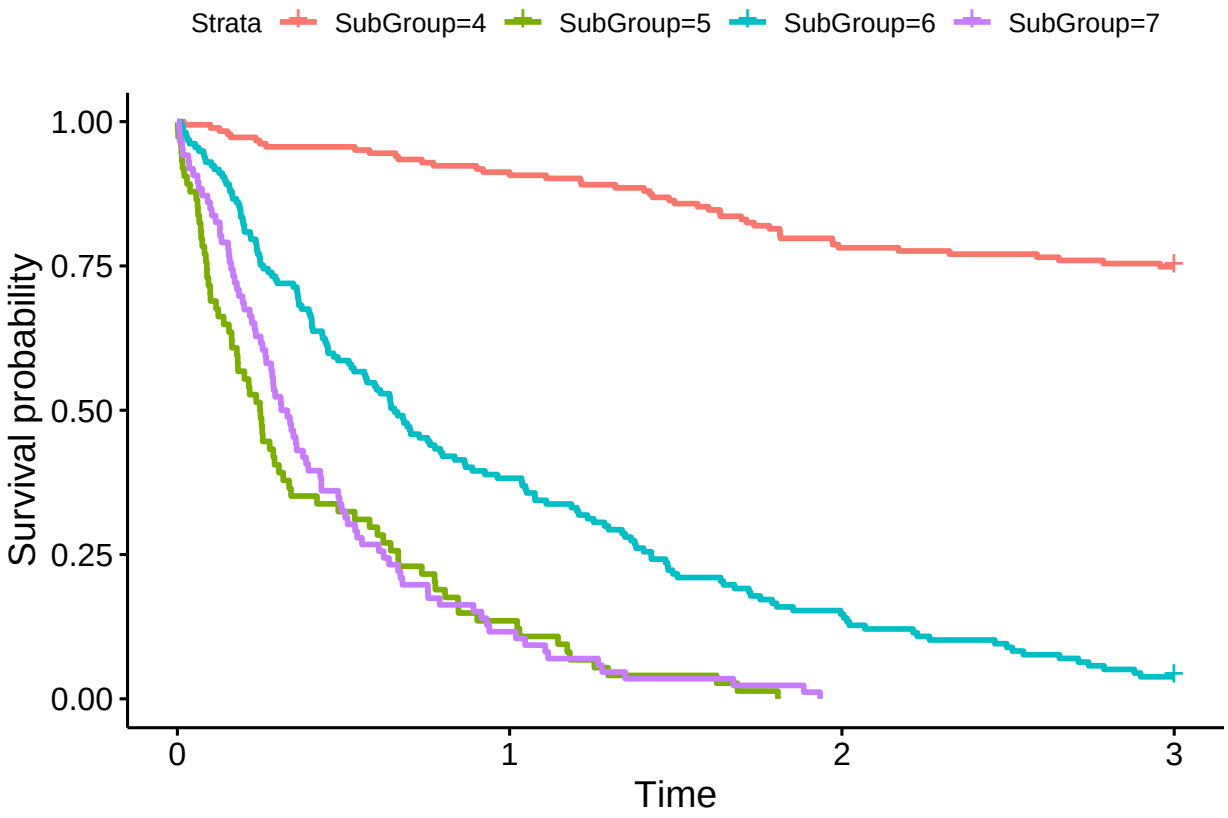
```
# Plotting SCT
par(xpd=TRUE)
plot(SCT, compress=TRUE)
text(SCT, use.n=TRUE)
```



In this case the plot represents a structure that is equal to the equivalent theoretical structure (see the figure of the theoretical structure above). In particular left splits represent the condition written in the node (e.g. the left leave represents observations with a value of $X1 < 0$ and a value of $X6 < 1$, corresponding to $X6 = 0$). For each node the corresponding number of observations and events, together with the estimated median survival time, is shown. It can be seen that for the left node a median survival time equal to NA is estimated. This because the median survival time in that node is greater than the follow-up time (see Subgroup 4 in the figure below). Finally, it is intesting to see that the estimated median survival times for the nodes corresponding to $X1 < 0$ and $X6 = 1$ and $X1 \geq 0$ and $X6 = 1$ are very similar (as expected in the equivalent theoretical structure).

Moreover, the Kaplan-Meier survival curves can be plotted for each terminal node as follows. Each subgroup represents a terminal node; in detail, each subgroup is named as the ID of the corresponding node (ID reported in the *Treeout* matrix).

```
KMPlot.SurvCART(x=SCT, type=1)
```



5. Performance evaluation in the test set

Once all the trees are fitted their performance can be evaluated. Beyond the above-mentioned performance measures (C-index, time-dependent AUC, BS and IBS), the size and the structure of fitted trees can be compared to those of the theoretical one (considering also equivalent structures). For doing that it is necessary to extract the needed information from the resulting objects. This means, for example, working with the *frame* matrix of *rpart*, or with the *Treeout* matrix of *SurvCART* (see Supplementary Materials File 2).

Then, for evaluating classical performance measures in the test set, the following functions can be used. Only examples for RRTs and CITs are shown below (due to the impossibility of using *predict*, *predictSurvProb* and *pec* functions for SCTs).

Harrel's C-index

Harrel's C-index can be evaluated through *rcorr.cens* and *Cindex* functions in *Hmisc* and *SurvMetrics* packages respectively.

The first function requires as first object a vector of survival predictions (one for each observation) and a *Surv* object. So, as a first thing the survival predictions must be obtained. If it is not possible to retrieve them (but only risk estimates can be obtained) a possible solution is to replace risk predictions with a decreasing function of these values. This is due to the fact that concordance index is a rank measure and so it is based only on the ordering of predicted values.

To obtain an overall measure of concordance for RRTs one can use the *predict* function. It returns for each observation the corresponding node outcome, i.e. a measure of risk.

```
pred_risk_RRT <- predict(object=pruned_RRT, newdata=data_test)
head(pred_risk_RRT)
```

```
##         2         3         4         8         9        10
## 0.2260154 2.8183587 0.2260154 0.2260154 2.8183587 1.6012796
```

The concordance index can then be evaluated as shown below:

```
# Survival outcome in the test set
ySurv_test <- Surv(data_test$eventtime, data_test$status)

library(Hmisc)
C_ind_RRT <- rcorr.cens(x= -pred_risk_RRT, S=ySurv_test)
```

```
##        C Index            Dxy           S.D.              n        missing
##   7.622413e-01   5.244827e-01   2.174412e-02   5.000000e+02   0.000000e+00
##     uncensored Relevant Pairs     Concordant      Uncertain
##   3.600000e+02   2.300400e+05   1.753460e+05   1.946000e+04
```

The function returns a vector containing the C-index value and Somer's *d Dxy*. It also provides other information like the number of comparable (*"relevant"*), concordant (*"concordant"*) and uncertain (*"uncertain"*) pairs. It can be seen that RRT has a good performance, with a C-index equal to 0.76.

For an overall C-index for CITs it is possible to estimate the median survival time for each observation in the test set and to provide it to the function. If Inf values are returned, a possible solution is to artificially replace them with a value greater than all the other observed median survival times (of course this trick can be used only for evaluating ranking measures).

```
pred_surv_CIT <- predict(object=CIT, newdata=data_test, type="response")
head(pred_surv_CIT)
```

```
##         2         3         4         8         9        10
##       Inf 0.2880490       Inf       Inf 0.2880490 0.6534732
```

```
# Replacement of Inf values with a value greater than all the others:
pred_surv_CIT[which(pred_surv_CIT==Inf)] <-
                           max(pred_surv_CIT[-which(pred_surv_CIT==Inf)])+3
C_ind_CIT <- rcorr.cens(x=pred_surv_CIT, S=ySurv_test)
```

```
##        C Index            Dxy           S.D.              n        missing
##   7.622413e-01   5.244827e-01   2.174412e-02   5.000000e+02   0.000000e+00
##     uncensored Relevant Pairs     Concordant      Uncertain
##   3.600000e+02   2.300400e+05   1.753460e+05   1.946000e+04
```

The fitted CIT performs as well as the RRT (this is an expected result since the two algorithms return the same tree).

The *SurvMetrics::Cindex* requires the same arguments of *rcorr.cens*, i.e. a *Surv* object and a vector of predicted survival probabilities or survival times:

```
library(SurvMetrics)
C_ind_RRT <- Cindex(object=ySurv_test, predicted= -pred_risk_RRT)
```

```
##   C index
## 0.7622413
```

```
C_ind_CIT <- Cindex(object=ySurv_test, predicted=pred_surv_CIT)
```

```
##        C Index            Dxy           S.D.              n        missing
##   7.622413e-01   5.244827e-01   2.174412e-02   5.000000e+02   0.000000e+00
```

```
##     uncensored Relevant Pairs     Concordant      Uncertain
##   3.600000e+02   2.300400e+05   1.753460e+05   1.946000e+04
```

If, instead, interest is on the C-index at given time points, a truncated C-index can be evaluated. The first thing to do is to evaluate a matrix of predicted survival probabilities at the different time points. These probabilities can be obtained with the *pec:::predictSurvProb* function. This last function is compatible only with a *pec* object; so, the fitted RRT and CIT must be transformed through the *pecRpart* and *pecCtree* functions. Please note that *pecCtree* uses the old *party* package and not *partykit*.

```
library(pec)
RRT_pec <- pecRpart(formula=Surv(eventtime, status)~., data=data_train,
                    cp=bestcp)

CIT_pec <- pecCtree(formula=Surv(eventtime, status)~., data=data_train)
```

The *predictSurvProb* function returns a matrix with a number of rows equal to the number of subjects in the test set and a number of columns equal to the length of *times* vector (*tvec*):

```
pred_RRT_pec <- predictSurvProb(object=RRT_pec, newdata=data_test,
                                times=tvec)
colnames(pred_RRT_pec) <- paste0("time=", tvec)
head(pred_RRT_pec[,c(1,6,11,16,21)])
```

```
##         time=1  time=1.5    time=2   time=2.5     time=3
## [1,] 0.9125683 0.8579235 0.7814208 0.77049180 0.74863388
## [2,] 0.1250000 0.0375000        NA         NA         NA
## [3,] 0.9125683 0.8579235 0.7814208 0.77049180 0.74863388
## [4,] 0.9125683 0.8579235 0.7814208 0.77049180 0.74863388
## [5,] 0.1250000 0.0375000        NA         NA         NA
## [6,] 0.3821656 0.2165605 0.1464968 0.08917197 0.03821656
```

```
pred_CIT_pec <- predictSurvProb(object=CIT_pec, newdata=data_test,
                                times=tvec)
colnames(pred_CIT_pec) <- paste0("time=", tvec)
head(pred_CIT_pec[,c(1,6,11,16,21)])
```

```
##         time=1  time=1.5    time=2   time=2.5     time=3
## [1,] 0.9125683 0.8579235 0.7814208 0.77049180 0.74863388
## [2,] 0.1250000 0.0375000 0.0062500 0.00625000 0.00625000
## [3,] 0.9125683 0.8579235 0.7814208 0.77049180 0.74863388
## [4,] 0.9125683 0.8579235 0.7814208 0.77049180 0.74863388
## [5,] 0.1250000 0.0375000 0.0062500 0.00625000 0.00625000
## [6,] 0.3821656 0.2165605 0.1464968 0.08917197 0.03821656
```

We can see that for RRT there are some NA values. This will imply problems with the evaluation of the performance indices. Going into more details we can see that, for example, the second observation belongs to the node with relative risk estimate equal to 2.8, thus to the right node (as can be seen in the graphical representation of the RRT). It can be noticed that, despite all observations experienced the event, their survival is estimated equal to NA starting from time=2.

```
pred_RRT_node<-predict(object=pruned_RRT, newdata=data_test[2,])
```

```
##        3
## 2.818359
```

This is an error because when the event occurs for all subjects in a group their survival is equal to 0 after the last observed timepoint. Indeed, if we estimate the survival probability in that node in the training set, the resulting curve is the following:

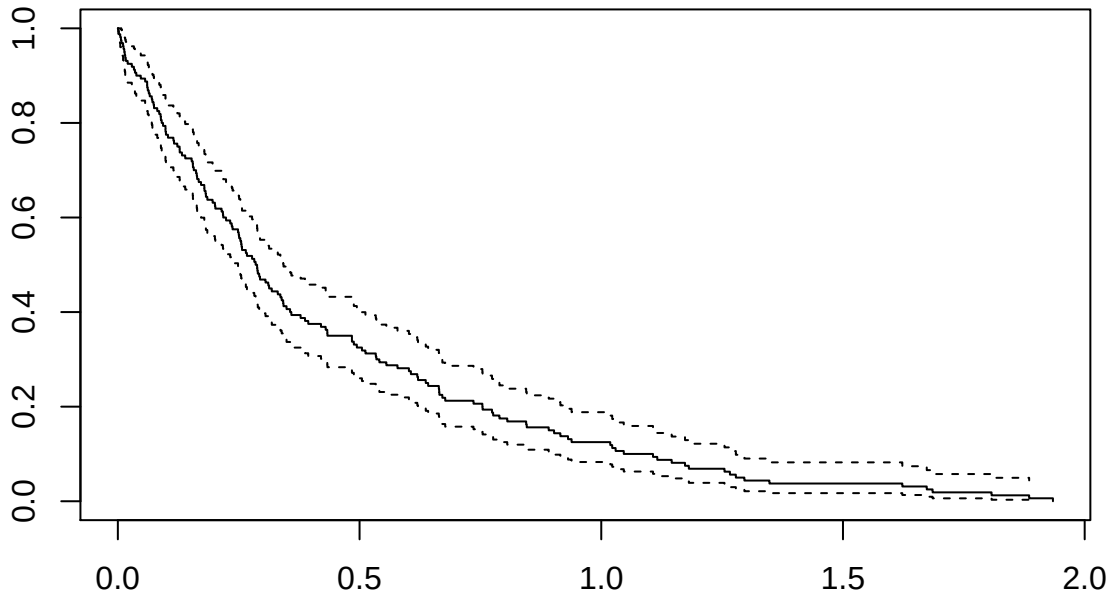
```r
# Risk predictions obtained with RRT in the training set
pred_risk_RRT_train <- predict(object=pruned_RRT, newdata=data_train)
# Subset of observations in the node with highest risk (node in which all
# subjects experience the event)
sub_node <- data_train[which(pred_risk_RRT_train==pred_RRT_node),]

surv_node <- survfit(formula=Surv(eventtime, status)~1, data=sub_node)
surv_node_summary <- cbind(surv_node$time, surv_node$surv)
colnames(surv_node_summary)<- c("Time", "Surv. Prob.")
head(surv_node_summary)
```

```
##             Time Surv. Prob.
## [1,] 0.001000609     0.99375
## [2,] 0.002167734     0.98750
## [3,] 0.004988887     0.98125
## [4,] 0.006337641     0.97500
## [5,] 0.007549165     0.96875
## [6,] 0.010812549     0.96250
```

```r
tail(surv_node_summary)
```

```
##            Time Surv. Prob.
## [155,] 1.622786     0.03125
## [156,] 1.673325     0.02500
## [157,] 1.685413     0.01875
## [158,] 1.807326     0.01250
## [159,] 1.885141     0.00625
## [160,] 1.934352     0.00000
```

```r
plot(surv_node)
```




```
max(sub_node$eventtime)
```

```
## [1] 1.934352
```

It can be seen that after the last observed timepoint, i.e. 1.934351784, the survival probability is equal to 0. So, a possible solution to solve the issue of NA values is to replace them with some consistent values. In this case, for example, after time 1.9 the survival probability can be set equal to 0.

```
pred_surv_RRT_new <- pred_RRT_pec
pred_surv_RRT_new[which(is.na(pred_surv_RRT_new))]<-0
head(pred_surv_RRT_new[,c(1,6,11,16,21)])
```

```
##         time=1  time=1.5    time=2   time=2.5     time=3
## [1,] 0.9125683 0.8579235 0.7814208 0.77049180 0.74863388
## [2,] 0.1250000 0.0375000 0.0000000 0.00000000 0.00000000
## [3,] 0.9125683 0.8579235 0.7814208 0.77049180 0.74863388
## [4,] 0.9125683 0.8579235 0.7814208 0.77049180 0.74863388
## [5,] 0.1250000 0.0375000 0.0000000 0.00000000 0.00000000
## [6,] 0.3821656 0.2165605 0.1464968 0.08917197 0.03821656
```

Once the survival probabilities have been estimated, a single column of the matrix (survival probabilities at a specific timepoint) has to be provided to the function (as **x** argument for *rcorr.cens* and *predicted* in *Cindex*). The idea is to evaluate an index similar to the one used by *pec* (shown hereafter). In particular, for each time point the status indicator δ_i is set equal to 1 only for those subjects who experienced the event before the time point of interest. Then, the time interval is truncated; thus all times (*eventtime*) greater than the given time point (tvec[i]) are considered equal to this last one, used as upper bound of the time interval:

```
# Truncated Concordance Index for RRT
C_ind_RRT_vec <- C_ind_RRT_vec1 <- rep(NA, length(tvec))
names(C_ind_RRT_vec) <- names(C_ind_RRT_vec1) <- tvec
for(i in 1:length(tvec))
   {
    eventtime <- data_test$eventtime; status <- data_test$status
    status[eventtime>tvec[i]] <- 0
    eventtime[eventtime>tvec[i]] <- tvec[i]
    ySurv <- Surv(eventtime, status)
    C_ind_RRT_vec[i] <- rcorr.cens(x=pred_RRT_pec[,i], S=ySurv)
   }
```

```
##         1       1.1       1.2       1.3       1.4       1.5       1.6       1.7
## 0.7588490 0.7615150 0.7640076 0.7632779 0.7636416 0.7649862 0.7652261 0.7643473
##       1.8       1.9         2       2.1       2.2       2.3       2.4       2.5
## 0.7638402 0.7633837 0.7473524 0.7475462 0.7448584 0.7423143 0.7436200 0.7433438
##       2.6       2.7       2.8       2.9         3
## 0.7427685 0.7446914 0.7439127 0.7443319 0.7449650
```

```
for(i in 1:length(tvec))
   {
    eventtime <- data_test$eventtime; status <- data_test$status
    status[eventtime>tvec[i]] <- 0
    eventtime[eventtime>tvec[i]] <- tvec[i]
    ySurv <- Surv(eventtime, status)
    C_ind_RRT_vec1[i] <- Cindex(object=ySurv, predicted=pred_RRT_pec[,i])
   }
```

```
## Error in Cindex(object = ySurv, predicted = pred_RRT_pec[, i]): The input vector cannot have NA
```

```
##         1       1.1       1.2       1.3       1.4       1.5       1.6       1.7
## 0.7588490 0.7615150 0.7640076 0.7632779 0.7636416 0.7649862 0.7652261 0.7643473
##       1.8       1.9         2       2.1       2.2       2.3       2.4       2.5
## 0.7638402 0.7633837        NA        NA        NA        NA        NA        NA
##       2.6       2.7       2.8       2.9         3
##        NA        NA        NA        NA        NA
```

It can be seen that, starting from time 2 the truncated C-index cannot be evaluated with the *Cindex* function in the *SurvMetrics* package, due to the presence of NA in the matrix of survival predictions.

```
# Truncated Concordance Index for CIT
C_ind_CIT_vec <- C_ind_CIT_vec1 <- rep(NA, length(tvec))
names(C_ind_CIT_vec) <- names(C_ind_CIT_vec1) <- tvec
for(i in 1:length(tvec))
  {
   eventtime <- data_test$eventtime; status <- data_test$status
   status[eventtime>tvec[i]] <- 0
   eventtime[eventtime>tvec[i]] <- tvec[i]
   ySurv <- Surv(eventtime, status)
   C_ind_CIT_vec[i] <- rcorr.cens(x=pred_CIT_pec[,i], S=ySurv)
   C_ind_CIT_vec1[i] <- Cindex(object=ySurv, predicted=pred_CIT_pec[,i])
  }
```

```
C_ind_CIT_vec
```

```
##         1       1.1       1.2       1.3       1.4       1.5       1.6       1.7
## 0.7588490 0.7615150 0.7640076 0.7632779 0.7636416 0.7649862 0.7652261 0.7643473
```

```
##       1.8       1.9         2       2.1       2.2       2.3       2.4       2.5
## 0.7638402 0.7633837 0.7643987 0.7643804 0.7630886 0.7616710 0.7621788 0.7619918
##       2.6       2.7       2.8       2.9         3
## 0.7615452 0.7623159 0.7619367 0.7619821 0.7622413
```

```
C_ind_CIT_vec1
```

```
##         1       1.1       1.2       1.3       1.4       1.5       1.6       1.7
## 0.7588490 0.7615150 0.7640076 0.7632779 0.7636416 0.7649862 0.7652261 0.7643473
##       1.8       1.9         2       2.1       2.2       2.3       2.4       2.5
## 0.7638402 0.7633837 0.7643987 0.7643804 0.7630886 0.7616710 0.7621788 0.7619918
##       2.6       2.7       2.8       2.9         3
## 0.7615452 0.7623159 0.7619367 0.7619821 0.7622413
```

Uno's C-index

Uno's C-index can be evaluated through the *UnoC* function in the *survAUC* package. This function requires three arguments: (i) a *Surv* object containing the outcome of the training set; (ii) a *Surv* object containing the outcome of the test set and (iii) a vector of predicted risk values for the test set.

```
### Survival outcome in the training set
ySurv_train <- Surv(data_train$eventtime, data_train$status)

library(survAUC)
UnoC_RRT <- UnoC(Surv.rsp=ySurv_train, Surv.rsp.new=ySurv_test,
                 lpnew=pred_risk_RRT)
```

```
## [1] 0.61984
```

```
UnoC_CIT <- UnoC(Surv.rsp=ySurv_train, Surv.rsp.new=ySurv_test,
                 lpnew= -pred_surv_CIT)
```

```
## [1] 0.61984
```

As expected and as already seen, the two fitted trees (RRT and CIT) have the same performance. The obtained values indicate a moderate performance (lower than that evaluated with Harrel's C-index).

Time-truncated C-index

Finally, an alternative function for evaluating the concordance index is *cindex* in the *pec* package, that evaluates the time-truncated C-index. The function requires as arguments: (i) a list of models for which evaluating the performance index; (ii) the formula used for growing the tree; (iii) the dataset on which evaluating the performance and (iv) a vector of times on which evaluating the index. Different methods for estimating Inverse Probability Censoring Weights (IPCWs) are available; among these there are the Cox regression PH model (*"cox"*) and the KM estimator (*"marginal"*), that is the default method. Below an example of code for evaluating performance of both RRTs and CITs at different time points. The procedure used by *cindex* in *pec* is similar to that shown above with the *rcorr.cens* and *Cindex* functions.

```
library(pec)
TruncC_RRT_CIT <- cindex(object=list("RRT"=RRT_pec, "CIT"=CIT_pec),
                         formula=Surv(eventtime, status)~., data=data_test,
                         eval.times=tvec)
```

```
##
## The c-index for right censored event times
##
## Prediction models:
##
```

```
## RRT CIT
## RRT CIT
##
## Right-censored response of a survival model
##
## No.Observations: 500
##
## Pattern:
##                 Freq
##  event          360
##  right.censored 140
##
## Censoring model for IPCW: marginal model (Kaplan-Meier for censoring distribution)
##
## No data splitting: either apparent or independent test sample performance
##
## Estimated C-index in %
##
## $AppCindex
##     time=1 time=1.1 time=1.2 time=1.3 time=1.4 time=1.5 time=1.6 time=1.7
## RRT   75.9     76.2     76.4     76.3     76.4     76.5     76.5     76.4
## CIT   75.9     76.2     76.4     76.3     76.4     76.5     76.5     76.4
##     time=1.8 time=1.9 time=2 time=2.1 time=2.2 time=2.3 time=2.4 time=2.5
## RRT     76.4     76.3   31.8     32.0     32.2     32.6     32.8     32.9
## CIT     76.4     76.3   76.4     76.4     76.3     76.2     76.2     76.2
##     time=2.6 time=2.7 time=2.8 time=2.9 time=3
## RRT     33.2     33.4     33.5     33.7   33.8
## CIT     76.2     76.2     76.2     76.2   76.2
```

We can see that this function allows to evaluate simultaneously the index at different timepoints. Due to the presence of NA in RRT predictions (as seen above) the evaluation of the index for RRT starting from time 2 is biased.

Time-dependent AUC

Time-dependent AUCs can be evaluated through *Score* and *AUC.hc* functions in *riskRegression* and *survAUC* packages respectively. The first function requires (i) a list of models to evaluate (**object**); (ii) the metrics; (iii) the used formula; (iv) the data; (v) the censoring model to use; (vi) a vector of times on which evaluating the measure of interest. Unfortunately, *Score* provides an error related to the internal function *predictRisk* when evaluating a *ctree* object. The first reason is that *Score* requires a *party* object and not a *partykit* one:

```
library(riskRegression)
AUC_Score_CIT_partykit <- Score(object=list("CIT_partykit"=CIT), metrics="AUC",
                                formula=Surv(eventtime, status)~X1+X2+X3+X4+X5+
                                X6+X7+X8+X9+X10, data=data_test, cens.model="km",
                                times=tvec)
```

```
## Error: Cannot find function (S3-method) called predictRisk.constpartyCannot find
## function (S3-method) called predictRisk.party
```

```
# Fit CIT with the old package party
CIT_party <- party::ctree(Surv(eventtime, status)~., data=data_train)
```

```
AUC_Score_CIT <- Score(object=list("CIT_party"=CIT_party), metrics="AUC",
                       formula=Surv(eventtime, status)~X1+X2+X3+X4+X5+X6+X7+X8+
                       X9+X10, data=data_test, cens.model="km",
```

```
                        times=tvec)
```

The second reason is related to the measure that the function extracts from the tree.

```
## Error: Column 'risk' is type 'list' which is not supported for ordering currently.
```

Indeed, the internal function *riskRegression:::predictRisk.BinaryTree* does not extract the right quantity. Let's see the structure of the function:

```
riskRegression:::predictRisk.BinaryTree
```

```
## function (object, newdata, ...)
## {
##     requireNamespace("party")
##     treeresponse <- party::treeresponse
##     sapply(treeresponse(object, newdata = newdata), function(x) x[1])
## }
## <bytecode: 0x000000002edfac90>
## <environment: namespace:riskRegression>
```

It uses the *treeresponse* function. Let's try to use it:

```
predrisk_CIT <- sapply(party:::treeresponse(CIT_party, newdata=data_test),function(x) x[1])
predrisk_CIT[,1:3]
```

```
##           [,1]       [,2]        [,3]
## n         183        160         183
## time      Numeric,47 Numeric,160 Numeric,47
## n.risk    Numeric,47 Numeric,160 Numeric,47
## n.event   Numeric,47 Numeric,160 Numeric,47
## n.censor  Numeric,47 Numeric,160 Numeric,47
## surv      Numeric,47 Numeric,160 Numeric,47
## std.err   Numeric,47 Numeric,160 Numeric,47
## cumhaz    Numeric,47 Numeric,160 Numeric,47
## std.chaz  Numeric,47 Numeric,160 Numeric,47
## type      "right"    "right"     "right"
## logse     TRUE       TRUE        TRUE
## conf.int  0.95       0.95        0.95
## conf.type "log"      "log"       "log"
## lower     Numeric,47 Numeric,160 Numeric,47
## upper     Numeric,47 Numeric,160 Numeric,47
## call      Expression Expression  Expression
```

It can be seen that instead of extracting a measure of risk, *treeresponse* extracts for each observation a list of elements including the timepoints, the number of subjects, the number of events and of censored observations and the estimated survival probability.

The issue can be solved modifying this function, extracting a measure of risk from the tree (e.g. KM hazard estimate):

```
predictRisk.BinaryTree <- function (object, newdata, ...)
{
  requireNamespace("party")
  treeresponse <- party::treeresponse
  sapply(treeresponse(object, newdata = newdata),
         function(x) sum(x$n.event)/x$n)
}
assignInNamespace("predictRisk.BinaryTree", predictRisk.BinaryTree,
```

```
                  pos="package:riskRegression")

predrisk_CIT <- riskRegression:::predictRisk.BinaryTree(CIT_party, data_test)
head(predrisk_CIT)
```

```
## [1] 0.2513661 1.0000000 0.2513661 0.2513661 1.0000000 0.9617834
```

Now, the function seems to work correctly:

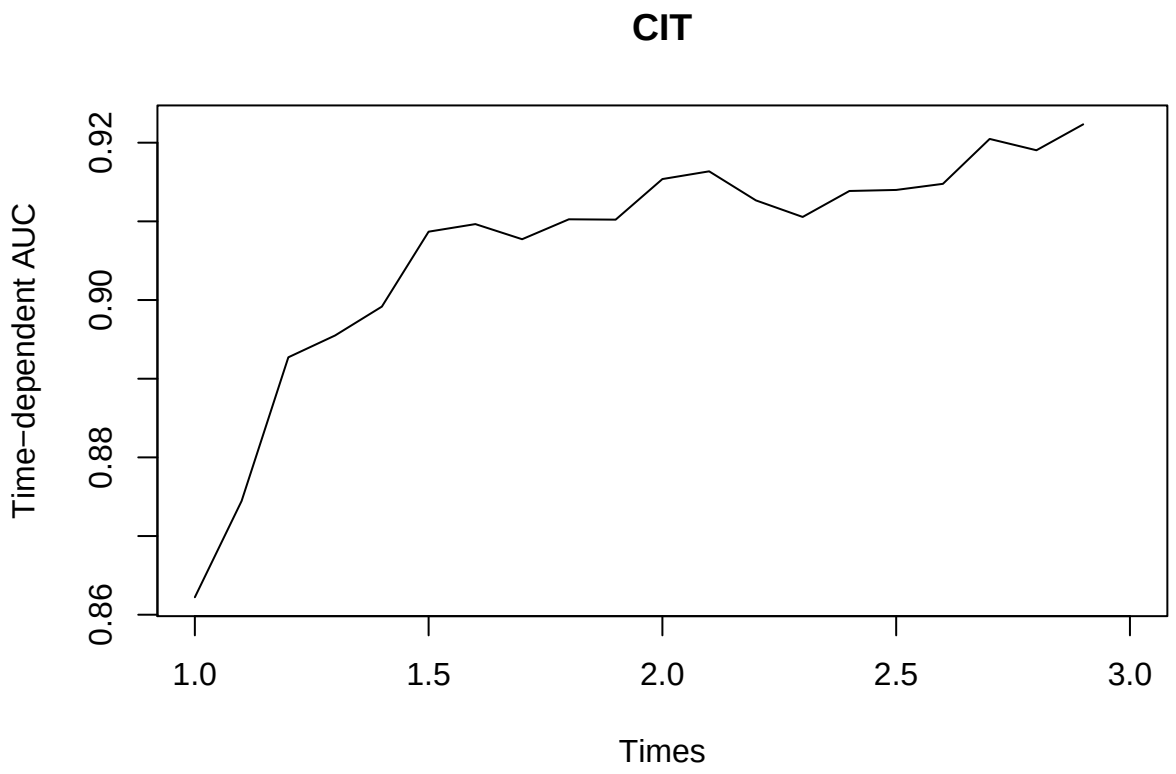
```
AUC_Score_CIT <- Score(object=list("CIT_party"=CIT_party), metrics="AUC",
                       formula=Surv(eventtime, status)~X1+X2+X3+X4+X5+X6+X7+X8+
                       X9+X10, data=data_test, cens.model="km",
                       times=tvec)
```

```
##
## Metric AUC:
##
## Results by model:
##
##         model times  AUC lower upper
##  1: CIT_party     1 86.2  83.2  89.3
##  2: CIT_party     1 87.4  84.5  90.4
##  3: CIT_party     1 89.3  86.6  91.9
##  4: CIT_party     1 89.5  87.0  92.1
##  5: CIT_party     1 89.9  87.4  92.5
##  6: CIT_party     2 90.9  88.5  93.2
##  7: CIT_party     2 91.0  88.6  93.3
##  8: CIT_party     2 90.8  88.4  93.2
##  9: CIT_party     2 91.0  88.7  93.4
## 10: CIT_party     2 91.0  88.7  93.4
## 11: CIT_party     2 91.5  89.2  93.8
## 12: CIT_party     2 91.6  89.3  93.9
## 13: CIT_party     2 91.3  88.9  93.6
## 14: CIT_party     2 91.1  88.7  93.4
## 15: CIT_party     2 91.4  89.0  93.7
## 16: CIT_party     2 91.4  89.0  93.7
## 17: CIT_party     3 91.5  89.1  93.8
## 18: CIT_party     3 92.0  89.8  94.3
## 19: CIT_party     3 91.9  89.6  94.2
## 20: CIT_party     3 92.2  90.0  94.4
## 21: CIT_party     3  NaN   NaN   NaN
##         model times  AUC lower upper
```

```
##
## NOTE: Values are multiplied by 100 and given in %.
```

```
## NOTE: The higher AUC the better.
```

```
plot(x=tvec, y=AUC_Score_CIT$AUC$score$AUC, type="l", xlab="Times",
     ylab="Time-dependent AUC", main="CIT")
```



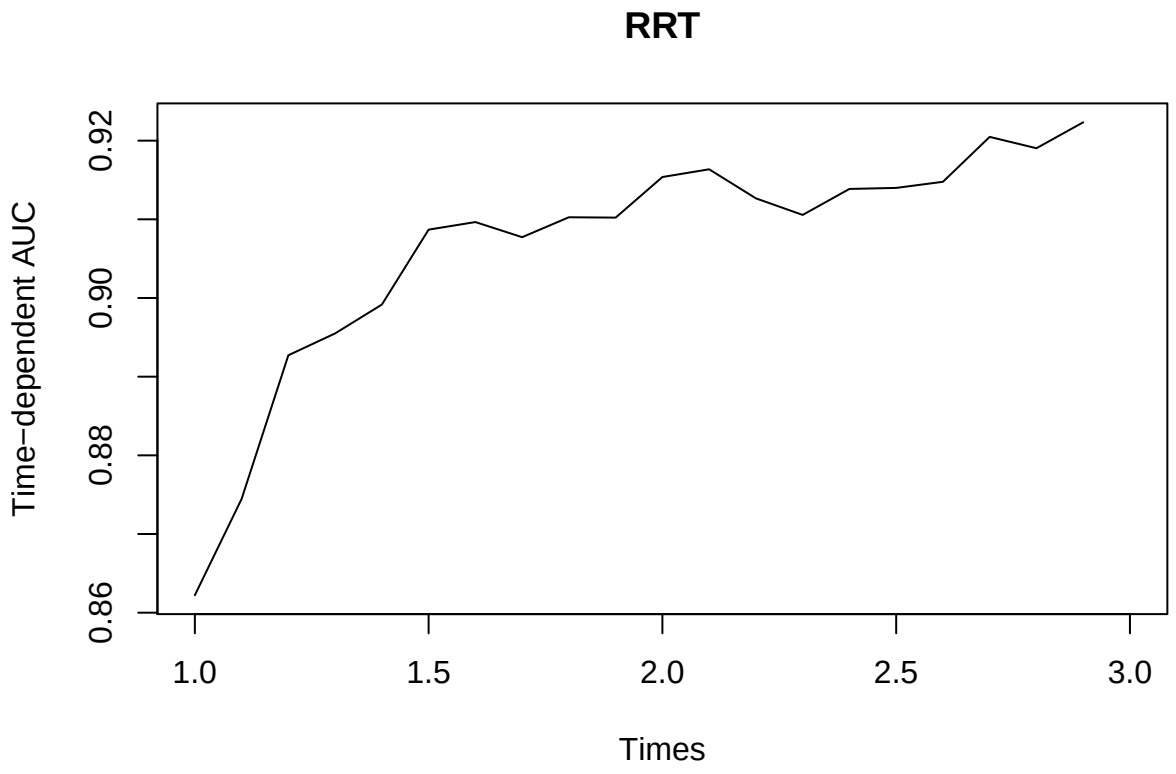
Evaluating time-dependent AUCs for an *rpart* object with *Score* is simpler. An example is provided below:

```
AUC_Score_RRT <- Score(object=list("RRT"=pruned_RRT), metrics="AUC",
                       formula=Surv(eventtime, status)~X1+X2+X3+X4+X5+X6+X7+X8+
                       X9+X10, data=data_test, cens.model="km",
                       times=tvec)
```

```
##
## Metric AUC:
##
## Results by model:
##
##     model times  AUC lower upper
##  1:   RRT     1 86.2  83.2  89.3
##  2:   RRT     1 87.4  84.5  90.4
##  3:   RRT     1 89.3  86.6  91.9
##  4:   RRT     1 89.5  87.0  92.1
##  5:   RRT     1 89.9  87.4  92.5
##  6:   RRT     2 90.9  88.5  93.2
##  7:   RRT     2 91.0  88.6  93.3
##  8:   RRT     2 90.8  88.4  93.2
##  9:   RRT     2 91.0  88.7  93.4
## 10:   RRT     2 91.0  88.7  93.4
## 11:   RRT     2 91.5  89.2  93.8
## 12:   RRT     2 91.6  89.3  93.9
## 13:   RRT     2 91.3  88.9  93.6
## 14:   RRT     2 91.1  88.7  93.4
```

```
## 15:   RRT     2 91.4  89.0  93.7
## 16:   RRT     2 91.4  89.0  93.7
## 17:   RRT     3 91.5  89.1  93.8
## 18:   RRT     3 92.0  89.8  94.3
## 19:   RRT     3 91.9  89.6  94.2
## 20:   RRT     3 92.2  90.0  94.4
## 21:   RRT     3  NaN   NaN   NaN
##     model times  AUC lower upper
##
## NOTE: Values are multiplied by 100 and given in %.
## NOTE: The higher AUC the better.
```

```
plot(x=tvec, y=AUC_Score_RRT$AUC$score$AUC, type="l", xlab="Times",
     ylab="Time-dependent AUC", main="RRT")
```



Differently, *survAUC::AUC.hc* requires a *Surv* object containing the outcome of the training data; a *Surv* object containing the outcome of the test set and a vector of risk predictions.

Attention has to be paid to this function. During some applications on real data it provided values greater than one!

```
library(survAUC)
AUC_hc_RRT <- AUC.hc(Surv.rsp=ySurv_train, Surv.rsp.new=ySurv_test,
                     lpnew=pred_risk_RRT, times=tvec)
```

```
## $auc
##  [1] 0.7715580 0.7883319 0.8098143 0.8136056 0.8202333 0.8387887 0.8435947
```

```
##  [8] 0.8499367 0.8528467 0.8587003 0.8715396 0.8771286 0.8634870 0.8439041
## [15] 0.8465800 0.8466676 0.8379205 0.8436683 0.8509202 0.8448936 0.0000000
##
## $times
##  [1] 1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.0 2.1 2.2 2.3 2.4 2.5 2.6 2.7 2.8
## [20] 2.9 3.0
##
## $iauc
## [1] 0.7868997
##
## attr(,"class")
## [1] "survAUC"
```

```
AUC_hc_CIT <- AUC.hc(Surv.rsp=ySurv_train, Surv.rsp.new=ySurv_test,
                     lpnew= -pred_surv_CIT, times=tvec)
```

```
## $auc
##  [1] 0.7715580 0.7883319 0.8098143 0.8136056 0.8202333 0.8387887 0.8435947
##  [8] 0.8499367 0.8528467 0.8587003 0.8715396 0.8771286 0.8634870 0.8439041
## [15] 0.8465800 0.8466676 0.8379205 0.8436683 0.8509202 0.8448936 0.0000000
##
## $times
##  [1] 1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.0 2.1 2.2 2.3 2.4 2.5 2.6 2.7 2.8
## [20] 2.9 3.0
##
## $iauc
## [1] 0.7868997
##
## attr(,"class")
## [1] "survAUC"
```

Brier Score and Integrated Brier Score

Two of the available packages for evaluating Brier Score (BS) and Integrated Brier Score (IBS) are *SurvMetrics* and *pec*. The two functions in *SurvMetrics* are *Brier* and *IBS*. They both require (i) a *Surv* object evaluated in the test set; (ii) a vector (for BS) or a matrix (for IBS) of survival probabilities of each observation in the time point(s) of interest and (iii) the time point (or vector of time points) at which evaluating the two indices. It uses KM IPCWs. An example is provided below.

```
library(SurvMetrics)
# Evaluation of BS at time 1 for RRT
brier_RRT_1 <- Brier(object=ySurv_test, pre_sp=pred_RRT_pec[,1], t_star=1)
```

```
## Brier Score
##   0.1347755
```

```
# Evaluation of BS at time 2 for RRT
brier_RRT_2 <- Brier(object=ySurv_test, pre_sp=pred_RRT_pec[,11], t_star=2)
```

```
## Error: The input probability vector cannot have NA
```

```
# Evaluation of BS at time 1 for CIT
brier_CIT_1 <- Brier(object=ySurv_test, pre_sp=pred_CIT_pec[,1], t_star=1)
```

```
## Brier Score
##   0.1347755
```

```
# Evaluation of BS at time 2 for CIT
brier_CIT_2 <- Brier(object=ySurv_test, pre_sp=pred_CIT_pec[,11], t_star=2)
```

```
## Brier Score
##   0.0921548
```

Even in this case the two models have an equal performance at time 1 (remember that lower values of BS indicate better performance). It can be seen that, as hinted, from time 2 the BS cannot be evaluated for the RRT using the output of *predictSurvProb*, due to the presence of NA. The problem can be solved using the matrix in which we have replaed NAs with zeros. Using these estimates the function provide the same results of CIT, suggesting that the solution makes sense.

```
brier_RRT_2 <- Brier(object=ySurv_test, pre_sp=pred_surv_RRT_new[,11], t_star=2)
```

```
## Brier Score
##  0.09216824
```

Integrated Brier Score can then be evaluated as follows for RRT and CIT. However, the *IBS* function is not compatible with NA values, so it is not evaluable for RRT using the matrix obtaiend with *predictSurvProb*. The issue can be solved using the matrix with the zeros instead of NAs.

```
ibs_RRT <- IBS(object=ySurv_test, sp_matrix=pred_RRT_pec, IBSrange=tvec)
```

```
## Error: The input probability matrix cannot have NA
```

```
ibs_RRT <- IBS(object=ySurv_test, sp_matrix=pred_surv_RRT_new, IBSrange=tvec)
```

```
##        IBS
## 0.09799359
```

```
ibs_CIT <- IBS(object=ySurv_test, sp_matrix=pred_CIT_pec, IBSrange=tvec)
```

```
##        IBS
## 0.09798687
```

Both the two trees have a satisfactory performance.

The two indices can also be evaluated through the *pec* function in the homonymous package. This function allows to specify a different censoring model than KM (*"marginal"*). BS can be evaluated as follows:

```
library(pec)
bs_1 <- pec(object=list("RRT"=RRT_pec, "CIT"=CIT_pec), data=data_test,
            formula=Surv(eventtime, status)~., times=1, exact=F,
            cens.model="marginal")
```

```
##
## Prediction error curves
##
## Prediction models:
##
## Reference       RRT       CIT
## Reference       RRT       CIT
##
## Right-censored response of a survival model
##
## No.Observations: 500
##
## Pattern:
##                 Freq
```

```
##  event          360
##  right.censored 140
##
## IPCW: marginal model
##
## No data splitting: either apparent or independent test sample performance
##   time n.risk Reference   RRT   CIT
## 1    1    241      0.25 0.135 0.135
```

Finally, the IBS can be evaluated as below:

```
library(pec)
ibs_rrt <- pec(object=list("RRT"=RRT_pec), data=data_test,
               formula=Surv(eventtime,status)~., times=tvec,
               cens.model="marginal")
```

```
##
## Prediction error curves
##
## Prediction models:
##
## Reference       RRT
## Reference       RRT
##
## Right-censored response of a survival model
##
## No.Observations: 500
##
## Pattern:
##                 Freq
##  event          360
##  right.censored 140
##
## IPCW: marginal model
##
## No data splitting: either apparent or independent test sample performance
##
## Cumulative prediction error, aka Integrated Brier score  (IBS)
##  aka Cumulative rank probability score
##
## Range of integration: 0 and time=1.9 :
##
##
## Integrated Brier score (crps):
##
##           IBS[0;time=1.9)
## Reference           0.216
## RRT                 0.121
```

```
ibs_cit <- pec(object=list("CIT"=CIT_pec), data=data_test,
           formula=Surv(eventtime,status)~., times=tvec,
           cens.model="marginal")
```

```
##
## Prediction error curves
##
```

```
## Prediction models:
##
## Reference       CIT
## Reference       CIT
##
## Right-censored response of a survival model
##
## No.Observations: 500
##
## Pattern:
##                 Freq
##  event          360
##  right.censored 140
##
## IPCW: marginal model
##
## No data splitting: either apparent or independent test sample performance
##
## Cumulative prediction error, aka Integrated Brier score  (IBS)
##  aka Cumulative rank probability score
##
## Range of integration: 0 and time=3 :
##
##
## Integrated Brier score (crps):
##
##           IBS[0;time=3)
## Reference         0.216
## CIT               0.109
```

Also for this package the problem of NA values occurs. It can be noticed, indeed, that for RRT the IBS is evaluated until time=1.9 (just right before the first timepoint with NA values). However, for *pec* it is not easily possible to solve the issue because the function requires the tree object and not the predicted survival probabilities.