

Package ‘randomSurvivalForest’

April 19, 2009

Version 3.5.1

Date 2008-08-28

Title Ishwaran and Kogalur’s Random Survival Forest

Author Hemant Ishwaran <hemant.ishwaran@gmail.com>, Udaya B. Kogalur
<ubk2101@columbia.edu>

Maintainer Udaya B. Kogalur <ubk2101@columbia.edu>

Depends R (>= 2.6.0)

Suggests XML

Description Ensemble survival analysis based on a random forest of trees using random inputs.

License GPL (>= 2)

URL <http://www.bio.ri.ccf.org/Resume/Pages/Ishwaran/ishwaran.html>, <http://www.kogalur-shear.com>

Repository CRAN

Date/Publication 2008-09-05 17:19:57

R topics documented:

burn	2
find.interaction	3
interaction.rsfc	5
pbcc	7
plot.ensemble	8
plot.error	10
plot.proximity	11
plot.variable	12
pmml2rsfc	14
predict.rsfc	16
print.rsfc	19
recid	20

rsf.default	20
rsf.news	29
rsf2pmml	29
Survrsf	31
veteran	32

Index	34
--------------	-----------

burn	<i>Burn Data</i>
------	------------------

Description

Data on 154 burn patients.

Format

A data frame containing:

Z1	treatment: 0=routine bathing, 1=Body cleansing
Z2	gender: 0=male, 1=female
Z3	race: 0=nonwhite 1=white
Z4	percentage of total surface area burned
Z5	burn site indicator head: 1=yes, 0=no
Z6	burn site indicator buttock: 1=yes, 0=no
Z7	burn site indicator trunk: 1=yes, 0=no
Z8	burn site indicator upper leg: 1=yes, 0=no
Z9	burn site indicator lower leg: 1=yes, 0=no
Z10	respiratory tract: 1=yes, 0=no
Z11	type of burn: 1=chemical, 2=scald, 3=electric, 4=flame
T1	time to excision or on study time
D1	excision indicator: 1=yes, 0=no
T2	time to prophylactic antibiotic treatment or on study time
D2	prophylactic antibiotic treatment: 1=yes 0=no
T3	time to straphylocous aureaus infection or on study time
D3	straphylocous aureaus infection: 1=yes 0=no

Source

Klein and Moeschberger (1997). *Survival Analysis: Techniques for Censored and Truncated Data*, Springer: New York.

References

Ichida et al. (1993) *Stat. Med.*, 12:301-310.

Examples

```
data(burn, package = "randomSurvivalForest")
```

find.interaction *Find Interactions Between Pairs of Variables*

Description

Test for pairwise interactions between variables by comparing pairwise importance values to additive individual importance values.

Usage

```
find.interaction(object,
                 predictorNames = NULL,
                 sorted = TRUE,
                 npred = NULL,
                 subset = NULL,
                 nrep = 1,
                 rough = FALSE,
                 importance = c("randomsplit", "permute")[1],
                 seed = NULL,
                 do.trace = FALSE,
                 ...)
```

Arguments

object	An object of class (rsf, grow) or (rsf, forest). Note: forest=TRUE must be used in the original rsf call.
predictorNames	Character vector of variable names to be considered. Default is to use all variables.
sorted	Should variables be sorted by importance values? Only applies when predictorNames=NULL.
npred	Use the first npred variables as ordered by VIMP (only applies when predictorNames=NULL). Default uses all variables.
subset	Indices indicating which rows of the predictor matrix to be used (note: this applies to the <i>object</i> predictor matrix, <i>predictors</i>). Default is to use all rows.
nrep	Number of Monte Carlo replicates.
rough	Logical value indicating whether fast approximation should be used. Default is FALSE.
importance	Method used to compute variable importance (VIMP).
seed	Seed for random number generator. Must be a negative integer (the R wrapper handles incorrectly set seed values).
do.trace	Logical. Should trace output be enabled? Default is FALSE. Integer values can also be passed. A positive value causes output to be printed each do.trace iteration.
...	Further arguments passed to or from other methods.

Details

Using a previously grown forest, identify pairwise interactions for all pairs of variables from a specified list. Two variables are paired and their paired VIMP calculated (referred to as 'Paired' importance). The VIMP for each separate variable is also calculated. The sum of these two values is referred to as 'Additive' importance. A large positive or negative difference between 'Paired' and 'Additive' indicates an association worth pursuing if the VIMP's for each variable are reasonably large (Ishwaran, 2007).

Depending on the size of the data, computations might be slow. In such cases, users should consider setting `npred` to a smaller number, or restricting the analysis to a subset of the data.

If `nrep` is greater than 1, the analysis is repeated `nrep` times and results averaged over the replications.

Note that `find.interaction` calls the lower level function `interaction.rsfc`. For programming purposes, users may consider doing likewise.

Value

Invisibly, the interaction table.

Author(s)

Hemant Ishwaran (hemant.ishwaran@gmail.com) and Udaya B. Kogalur (ubk2101@columbia.edu)

References

H. Ishwaran (2007). Variable importance in binary regression trees and forests, *Electronic J. Statist.*, 1:519-537.

See Also

`interaction.rsfc`.

Examples

```
#-----
# Explore relationship between the top two predictors from veteran data.

data(veteran, package = "randomSurvivalForest")
v.out <- rsf(Survrsf(time,status)~., veteran, ntree = 1000, forest = TRUE)
find.interaction(v.out, npred = 2, nrep=1)

## Not run:
#-----
# All pairwise interactions: PBC data.
# Use fast approximation to speed up computations.

data(pbc, package = "randomSurvivalForest")
rsf.out <- rsf(Survrsf(days,status)~., pbc, ntree = 1000, forest = TRUE)
find.interaction(rsf.out, nrep=3, rough=T)
## End(Not run)
```

interaction.rsf *VIMP for Single or Grouped Variables*

Description

Calculate variable importance (VIMP) for a single variable or group of variables.

Usage

```
interaction.rsf(object,
               predictorNames = NULL,
               subset = NULL,
               joint = TRUE,
               rough = FALSE,
               importance = c("randomsplit", "permute", "none")[1],
               seed = NULL,
               do.trace = FALSE,
               ...)
```

Arguments

object	An object of class (rsf, grow) or (rsf, forest). Note: forest=TRUE must be used in the original rsf call.
predictorNames	Character vector of x-variable names to be considered. If NULL (the default) all variables are used. Only x-variables listed in the object predictor matrix will be used.
subset	Indices indicating which rows of the predictor matrix to be used (note: this applies to the <i>object</i> predictor matrix, <code>predictors</code>). Default is to use all rows.
joint	Should joint-VIMP or individual VIMP be calculated? See details below.
rough	Logical value indicating whether fast approximation should be used. Default is FALSE.
importance	Method used to compute variable importance (VIMP).
seed	Seed for random number generator. Must be a negative integer (the R wrapper handles incorrectly set seed values).
do.trace	Logical. Should trace output be enabled? Default is FALSE. Integer values can also be passed. A positive value causes output to be printed each <code>do.trace</code> iteration.
...	Further arguments passed to or from other methods.

Details

Using a previously grown forest, and restricting the data to that indicated by `subset`, calculate the VIMP for variables listed in `predictorNames`. If `joint=TRUE`, a joint-VIMP is calculated. The joint-VIMP is the importance for the group of variables, when the group is perturbed simultaneously. If `joint=FALSE`, the VIMP for each variable considered separately is calculated.

Depending upon the option `importance`, VIMP is calculated either by random daughter assignment, by random permutation of the variable(s), or none (no perturbing).

Value

A list with the following components:

<code>err.rate</code>	OOB error rate for the (unperturbed) ensemble restricted to the subsetted data.
<code>err.perturb.rate</code>	OOB error rate for the perturbed ensemble restricted to the subsetted data. Either a vector or a single number depending upon the option <code>joint</code> .
<code>importance</code>	Variable importance (VIMP). Either a vector or a single number depending upon the option <code>joint</code> .

Author(s)

Hemant Ishwaran (hemant.ishwaran@gmail.com) and Udaya B. Kogalur (ubk2101@columbia.edu)

References

H. Ishwaran (2007). Variable importance in binary regression trees and forests, *Electronic J. Statist.*, 1:519-537.

See Also

`find.interaction`.

Examples

```
#-----
# Example of paired-VIMP.
# Veteran data.

data(veteran, package = "randomSurvivalForest")
v.out <- rsf(Survrsf(time,status)~., veteran, ntree = 1000, forest = TRUE)
interaction.rsf(v.out, c("karno", "celltype"))$importance

## Not run:
#-----
# Individual VIMP for data restricted to events only.
# PBC data.

data(pbc, package = "randomSurvivalForest")
rsf.out <- rsf(Survrsf(days,status)~., pbc, ntree = 1000, forest = TRUE)
o.r <- rev(order(rsf.out$importance))
```

```

VIMP <- rsf.out$importance[o.r]
VIMP.events <- rep(0, length(VIMP))
names(VIMP.events) <- names(VIMP)
events <- which(rsf.out$cens == 1)
VIMP.events <-
  interaction.rsfc(rsf.out, names(VIMP), events, joint = FALSE)$importance
VIMP.all <- as.data.frame(cbind(VIMP.events = VIMP.events, VIMP = VIMP))
print(round(VIMP.all, 3))

#-----
# Estimate variability of VIMP in two ways (PBC data):
# (i) Monte Carlo: Estimates variability of the procedure
# (ii) Bootstrap: Estimates statistical variability

data(pbc, package = "randomSurvivalForest")
rsf.out <- rsfc(Survrsfc(days,status)~., pbc, ntree = 1000, forest = TRUE)
o.r <- rev(order(rsf.out$importance))
VIMP <- rsf.out$importance[o.r]
subset.index <- 1:nrow(rsf.out$predictors)
VIMP.mc <- VIMP.boot <- NULL
for (k in 1:100) {
  cat("iteration:", k , "\n")
  VIMP.mc <-
    cbind(VIMP.mc, interaction.rsfc(rsf.out, names(VIMP), joint = FALSE)$importance)
  VIMP.boot <-
    cbind(VIMP.boot, interaction.rsfc(rsf.out, names(VIMP),
      subset = sample(subset.index, replace = TRUE), joint = FALSE)$importance)
}
VIMP.mc <- as.data.frame(cbind(VIMP.mean = apply(VIMP.mc, 1, mean),
  VIMP.sd = apply(VIMP.mc, 1, sd)))
VIMP.boot <- as.data.frame(cbind(VIMP.mean = apply(VIMP.boot, 1, mean),
  VIMP.sd = apply(VIMP.boot, 1, sd)))
rownames(VIMP.mc) <- rownames(VIMP.boot) <- names(VIMP)
print(round(VIMP.mc, 3))
print(round(VIMP.boot, 3))
## End(Not run)

```

pbc

*Primary Biliary Cirrhosis (PBC) Data***Description**

Data from the Mayo Clinic trial in primary biliary cirrhosis (PBC) of the liver conducted between 1974 and 1984. A total of 424 PBC patients, referred to Mayo Clinic during that ten-year interval, met eligibility criteria for the randomized placebo controlled trial of the drug D-penicillamine. The first 312 cases in the data set participated in the randomized trial and contain largely complete data. The additional 112 cases did not participate in the clinical trial, but consented to have basic measurements recorded and to be followed for survival. Six of those cases were lost to follow-up shortly after diagnosis, so the data here are on an additional 106 cases as well as the 312 randomized participants. Missing data items are denoted by NA.

Format

A data frame containing:

days	survival time in days
status	censoring indicator
drugs	1=D-penicillamine, 2=placebo
age	age in days
sex	0=female, 1=male
asictes	presence of asictes, 0=no 1=yes
hepatom	presence of hepatomegaly, 0=no 1=yes
spiders	presence of spiders, 0=no 1=yes
edema	presence of edema (0, 0.5, 1)
bili	serum bilirubin in mg/dl
chol	serum cholesterol in mg/dl
albumin	albumin in gm/dl
copper	urine copper in ug/day
alk	alkaline phosphatase in U/liter
sgot	SGOT in U/ml
trig	triglycerides in mg/dl
platelet	platelets per cubic ml/1000
prothrombin	prothrombin time in seconds
stage	histologic stage of disease

Source

Flemming and Harrington, 1991, Appendix D.1.

References

Flemming T.R and Harrington D.P., (1991) *Counting Processes and Survival Analysis*. New York: Wiley.

Examples

```
data(pbc, package = "randomSurvivalForest")
```

plot.ensemble *Plot of Ensemble Estimates*

Description

Plot ensemble survival curves and ensemble estimates of mortality.

Usage

```
plot.ensemble(x, plots.one.page = TRUE, ...)
```

Arguments

`x` An object of class `(rsf, grow)` or `(rsf, predict)`.
`plots.one.page` Logical. Should plots be placed on one page? Default is TRUE.
`...` Further arguments passed to or from other methods.

Details

Four plots are produced. Going from top to bottom, left to right: (1) This shows the ensemble survival function for each individual in the data. Thick red line is overall ensemble survival, thick green line is Nelson-Aalen estimator. (2) This is a comparison of the population ensemble survival function to the Nelson-Aalen estimator. (3) The Brier score, a value between 0 and 1 (where 0=perfect, 1=poor, and 0.25=guessing) is plotted at each of the unique event times with plot stratified by ensemble mortality value (see Graf et al. for more background on Brier scores). Stratification is into 4 groups corresponding to the 0-25, 25-50, 50-75 and 75-100 percentile values of mortality. Red line is non-stratified score. (4) Plot of estimated mortality versus observed time. Points in blue correspond to events, black points are censored observations.

Note that when `x` is of class `(rsf, predict)` not all plots will be produced.

Author(s)

Hemant Ishwaran (hemant.ishwaran@gmail.com) and Udaya B. Kogalur (ubk2101@columbia.edu)

References

E. Graf, C. Schmoor, W. Sauerbrei and M. Schumacher M (1999). Assessment and comparison of prognostic classification schemes for survival data, *Statistics in Medicine*, 18:2529-2545.
H. Ishwaran, U.B. Kogalur (2007). Random survival forests for R, *Rnews*, 7/2:25-31.

See Also

`rsf`, `predict.rsfc`.

Examples

```
data(veteran, package = "randomSurvivalForest")
v.out <- rsf(Survrsf(time, status)~., veteran, ntree = 1000)
plot.ensemble(v.out)
```

plot.error

Plot of Error Rate and Variable Importance

Description

Plot out-of-bag (OOB) error rate for the ensemble as a function of number of trees in the forest. Also plots importance values for predictors. Note this is the default plot method for the package.

Usage

```
plot.error(x, ...)
plot.rsf(x, ...)
```

Arguments

`x` An object of class `(rsf, grow)` or `(rsf, predict)`.
`...` Further arguments passed to or from other methods.

Details

Plot of OOB error rate, with the `b`-th value being the error rate for the ensemble computed using the first `b` trees. Error rate is $1-C$, where C is Harrell's concordance index. Rates given are between 0 and 1, with 0.5 representing the benchmark value of a procedure based on random guessing. A value of 0 is perfect.

In the original call, if `importance=TRUE` (the default setting), then variable importance (VIMP) values for predictors are plotted. A matrix with up to 3 columns is also printed. First column are VIMP values, second column are standardized VIMP values (divided by the maximum importance value), third column is the vector `predictorWt` (only printed if values are distinct).

Author(s)

Hemant Ishwaran <hemant.ishwaran@gmail.com> and Udaya B. Kogalur <ubk2101@columbia.edu>

References

H. Ishwaran, Udaya B. Kogalur, Eugene H. Blackstone and Michael S. Lauer (2007). Random Survival Forests. *Cleveland Clinic Technical Report*.
 L. Breiman (2001). Random forests, *Machine Learning*, 45:5-32.
 F.E. Harrell et al. (1982). Evaluating the yield of medical tests, *J. Amer. Med. Assoc.*, 247, 2543-2546.

See Also

`rsf`, `predict.rsf`.

Examples

```
data(veteran, package = "randomSurvivalForest")
v.out <- rsf(Survrsf(time, status)~., veteran, ntree = 1000)
plot.error(v.out, veteran)
```

plot.proximity *Plot of Proximity*

Description

Multidimensional scaling plot of proximity matrix.

Usage

```
plot.proximity(x, plot = TRUE, ...)
```

Arguments

x	An object of class (rsf, grow) or (rsf, predict). Note that proximity=TRUE must be used in the original rsf call.
plot	Logical. If TRUE, proximity is plotted.
...	Further arguments passed to or from other methods.

Details

Extracts proximity information from x and transforms this to a symmetric proximity matrix. Dissimilarities between points are then converted into distances using the R multidimensional scaling function cmdscale and then plotted. Overlaid on the plot are mortality values, rescaled from 1-100, with 1 indicating low mortality, and 100 indicating high mortality. Mortality values will be well separated in successful analyses. Note that points in blue correspond to events, whereas black points are censored observations.

Value

Invisibly, the proximity matrix with entries transformed to relative frequencies.

Author(s)

Hemant Ishwaran <hemant.ishwaran@gmail.com> and Udaya B. Kogalur <ubk2101@columbia.edu>

References

H. Ishwaran, Udaya B. Kogalur, Eugene H. Blackstone and Michael S. Lauer (2007). Random Survival Forests. *Cleveland Clinic Technical Report*.

See Also

rsf, predict.rsfc

Examples

```
data(pbc, package = "randomSurvivalForest")
pbc.prox.out <- rsf(Survrsf(days,status)~., pbc, ntree = 100, proximity = TRUE)
plot.proximity(pbc.prox.out)
```

plot.variable *Plot of Ensemble Survival Effect of Variables*

Description

Plot of ensemble mortality, predicted median survival, or predicted survival time for each variable. Users can select between marginal and partial plots.

Usage

```
plot.variable(x,
             plots.per.page = 4,
             granule = 5,
             sorted = TRUE,
             type = c("mort", "rel.freq", "surv", "time")[1],
             partial = FALSE,
             predictorNames = NULL,
             npred = NULL,
             npts = 25,
             subset = NULL,
             ...)
```

Arguments

x	An object of class (rsf, grow) or (rsf, predict).
plots.per.page	Integer value controlling page layout.
granule	Integer value controlling whether a plot for a specific variable should be given as a boxplot or scatter plot. Larger values coerce boxplots.
sorted	Should variables be sorted by importance values (only applies if importance values are available)? Default is TRUE.
type	Select type of value to be plotted on the vertical axis. See details.
partial	Logical. Should partial plots be created? Default is FALSE.
predictorNames	Character vector of variable names. Only these variables will be plotted. Default is all.
npred	Number of variables to be plotted (only applies when predictorNames=NULL). Default is all.
npts	Maximum number of points used when generating partial plots for continuous variables.

subset	Indices indicating which rows of the predictor matrix to be used (note: this applies to the <i>processed</i> predictor matrix, <code>predictors</code> of the object). Default is to use all rows.
...	Further arguments passed to or from other methods.

Details

Either mortality, relative frequency of mortality, predicted median survival, or predicted survival times are plotted on the vertical axis (y-value) against x-variables on the horizontal axis. The choice of x-variables are specified by `predictorNames` or using `npred`. The choice of y-value is controlled by `type`. There are 4 different choices: `mort` is ensemble mortality, `rel.freq` is standardized mortality, `surv` is predicted median survival (predicted survival at the median survival time), `time` is the predicted survival time (this last option only applies to partial plots, however). For continuous variables, points are colored so that blue corresponds to events, whereas black points represent censored observations.

Ensemble mortality values should be interpreted in terms of total number of deaths. For example, if individual `i` has a mortality value of 100, then if all individuals were the same as `i`, we would expect to find 100 deaths on average in the data. If `type` is set to `rel.freq`, then mortality values are divided by an adjusted sample size, defined as the maximum of the sample size and the maximum mortality value. The standardized mortality values no longer indicate total deaths, but instead reflect relative mortality.

Partial plots are created when `partial=TRUE`. Interpretation for these are different than marginal plots. The y-value for a variable X , evaluated at $X = x$, is

$$\tilde{f}(x) = \frac{1}{n} \sum_{i=1}^n \hat{f}(x, x_{i,O}),$$

where $x_{i,O}$ represents the value for all other variables other than X for individual i and \hat{f} is the predicted value. Generating partial plots can be very slow. Choosing a small value for `npts` can speed up computational times as this restricts the number of distinct x values used in computing \tilde{f} .

For continuous variables, red points are used to indicate partial values and dashed red lines represent a lowess smoothed error bar of +/- two standard errors. Black dashed line is the lowess estimate of the partial values. For discrete variables, partial values are indicated using boxplots with whiskers extending out approximately two standard errors from the mean. Standard errors are meant only to be a guide and should be interpreted with caution.

Partial plots can be slow. Setting `type` to `time` can greatly speed things up. Setting `npts` to a smaller number should also be tried.

Author(s)

Hemant Ishwaran (hemant.ishwaran@gmail.com) and Udaya B. Kogalur (ubk2101@columbia.edu)

References

- H. Ishwaran, U.B. Kogalur (2007). Random survival forests for R, *Rnews*, 7/2:25-31.
- J.H. Friedman (2001). Greedy function approximation: a gradient boosting machine, *Ann. of Stat.*, 5:1189-1232.
- A. Liaw and M. Wiener (2002). Classification and regression by randomForest, *R News*, 2:18-22.

See Also

rsf, predict.rsfc.

Examples

```
#-----
# Some examples: veteran data.

data(veteran, package = "randomSurvivalForest")
v.out <- rsf(Survrsf(time,status)~., veteran, forest = TRUE, ntree = 1000)
plot.variable(v.out, plots.per.page = 3)
plot.variable(v.out, plots.per.page = 2, predictorNames = c("trt", "karno", "age"))
plot.variable(v.out, type = "rel.freq", partial = TRUE, plots.per.page = 2, npred=3)

## Not run:
#-----
# Fast partial plots using 'time' type.
# Top 8 predictors from PBC data.

data(pbc, package = "randomSurvivalForest")
pbc.out <- rsf(Survrsf(days,status)~., pbc, ntree = 1000, forest = TRUE)
plot.variable(pbc.out, type = "time", partial = TRUE, npred=8)
## End(Not run)
```

pmml2rsf

Restore Random Survival Forest From PMML

Description

pmml2rsf implements the Predictive Model Markup Language specification for a **randomSurvivalForest** forest object. In particular, this function gives the user the ability to restore the geometry of a forest from a PMML XML document.

Usage

```
pmml2rsf(pmmlRoot, ...)
```

Arguments

pmmlRoot	The top-level “XMLNode” object, or equivalently the root node, resulting from parsing an XML document. This node must be of type PMML.
...	Further arguments passed to or from other methods.

Details

The Predictive Model Markup Language is an XML based language which provides a way for applications to define statistical and data mining models and to share models between PMML compliant applications. More information about PMML and the Data Mining Group can be found at <http://www.dmg.org>.

Use of PMML and `pmml2rsf` requires the **XML** package. Be aware that XML is a very verbose data format. Reasonably sized trees and data sets can lead to extremely large text files. XML, while achieving interoperability, is not an efficient data storage mechanism in this case.

It is anticipated that `pmml2rsf` will be used to import the geometry of a forest from other PMML compliant applications. In addition, the user may wish to restore the geometry of a forest that was previously saved using `rsf2pmml`.

Value

A `randomSurvivalForest` forest object. See note below.

Note

One cautionary note is in order. The PMML representation of the forest object is incomplete, in that the object needs to be massaged in order for prediction to be possible. This will be clear in the examples. This deficiency will be addressed in future releases of this package. However, it was felt that the current functionality was important enough and mature enough to warrant release in this version of the product.

Author(s)

Hemant Ishwaran (hemant.ishwaran@gmail.com) and Udaya B. Kogalur (ubk2101@columbia.edu)

References

H. Ishwaran, Udaya B. Kogalur, Eugene H. Blackstone and Michael S. Lauer (2007). Random Survival Forests. *Cleveland Clinic Technical Report*.
<http://www.dmg.org>

See Also

`xmlTreeParse`, `xmlRoot`, `saveXML`, `rsf`, `predict.rsfc`, `rsf2pmml`.

Examples

```
# Example 1: Growing a forest, saving it as a PMML document,  
# restoring the forest from the PMML document, and using this forest to  
# perform prediction.  
  
## Not run:  
library("XML")  
  
data(veteran, package = "randomSurvivalForest")  
veteran.out <- rsf(Survrsf(time, status)~.,
```

```

        data = veteran,
        ntree = 5,
        forest = TRUE)
veteran.forest <- veteran.out$forest
veteran.pmml <- rsf2pmml(veteran.forest)

# Save the document to disk.
saveXML(veteran.pmml, "veteran.forest.xml")
veteran.pmml <- xmlRoot(xmlTreeParse("veteran.forest.xml"))

partial.forest <- pmml2rsf(veteran.pmml)

# The PMML forest object must be massaged before it can be used
# for prediction as follows:
veteran.restored.forest <- list(
  nativeArray=partial.forest$nativeArray,
  nativeFactorArray=partial.forest$nativeFactorArray,
  timeInterest=partial.forest$timeInterest,
  predictorNames=partial.forest$predictorNames,
  seed=partial.forest$seed
  formula=partialForest$formula,
  predictors=veteran.forest$predictors,
  time=veteran.forest$time,
  cens=veteran.forest$cens)

# The actual time, censoring and prediction values of the data set
# used to grow the forest are not contained in the PMML
# representation of the forest.  If the user has access to the original
# datafile that was used to grow the forest, this information can be
# easily recovered.  The names corresponding to the time, censoring and
# prediction data are all retained in the PMML representation of the forest.

class(veteran.restored.forest) <- c("rsf", "forest")
veteran.restored.out <- predict.rsf(veteran.restored.forest, test=veteran)

## End(Not run)

```

predict.rsf

Random Survival Forest Prediction

Description

Prediction on test data using Random Survival Forests.

Usage

```

predict.rsf(object = NULL,
            test = NULL,
            importance = c("randomsplit", "permute", "none")[1],
            na.action = c("na.omit", "na.impute")[1],

```

```

proximity = FALSE,
seed = NULL,
do.trace = FALSE,
...)
```

Arguments

<code>object</code>	An object of class <code>(rsf, grow)</code> or <code>(rsf, forest)</code> . Note that <code>forest=TRUE</code> must be used in the original <code>rsf</code> call for prediction to work.
<code>test</code>	Data frame containing test data. Missing values allowed.
<code>importance</code>	Method used to compute variable importance (VIMP). Only applies when test data contains outcomes.
<code>na.action</code>	Action to be taken if the data contain NA's. Possible values are <code>na.omit</code> , which removes the entire record if even one of its entries is NA, and <code>na.impute</code> , which imputes the test data. See details below.
<code>proximity</code>	Logical. Should proximity measure between test observations be calculated? Can be large. Default is FALSE.
<code>seed</code>	Seed for random number generator. Must be a negative integer (the R wrapper handles incorrectly set seed values).
<code>do.trace</code>	Logical. Should trace output be enabled? Default is FALSE. Integer values can also be passed. A positive value causes output to be printed each <code>do.trace</code> iteration.
<code>...</code>	Further arguments passed to or from other methods.

Details

`predict.rsfc` takes a test data set, drops it down the forest grown from the training data, and computes an ensemble cumulative hazard function (CHF). CHF's are calculated for each individual in the test data for all unique death time points from the original `grow` (training) data. Overall error rate and the VIMP for each variable are computed for the test data if outcome information is available. Setting `na.action=na.impute` imputes missing test data (x-variables or outcomes) using the forest grown from the training data (Ishwaran et al. 2008). Only training data is used in imputing test data to avoid biasing error rates.

Value

An object of class `(rsf, predict)`, which is a list with the following components:

<code>call</code>	The original <code>grow</code> call to <code>rsfc</code> .
<code>forest</code>	The <code>grow</code> forest.
<code>ntree</code>	Number of trees in the <code>grow</code> forest.
<code>leaf.count</code>	Number of terminal nodes for each tree in the <code>grow</code> forest. Vector of length <code>ntree</code> .
<code>timeInterest</code>	Sorted unique event times from <code>grow</code> (training) data. Ensemble values given for these time points only.
<code>n</code>	Sample size of test data (depends upon NA's, see <code>na.action</code>).

<code>ndead</code>	Number of deaths in test data (can be NULL).
<code>time</code>	Vector of survival times from test data (can be NULL).
<code>cens</code>	Vector of censoring indicators from test data (can be NULL).
<code>predictorNames</code>	Character vector of variable names.
<code>predictors</code>	Data frame comprising x-variables used for prediction.
<code>ensemble</code>	Matrix containing the ensemble CHF for the test data. Each row corresponds to a test data individual's CHF evaluated at each of the time points in <code>timeInterest</code> .
<code>mortality</code>	Vector containing ensemble mortality for each individual in the test data. Ensemble mortality values should be interpreted in terms of total number of training deaths.
<code>err.rate</code>	Vector of length <code>ntree</code> containing error rate of the test data. Can be NULL.
<code>importance</code>	VIMP of each variable in the test data. Can be NULL.
<code>proximity</code>	If <code>proximity=TRUE</code> , a matrix recording proximity of the inputs from test data is computed. Value returned is a vector of the lower diagonal of the matrix. Use <code>plot.proximity()</code> to extract this information.
<code>imputedIndv</code>	Vector of indices of records in test data with missing values. Can be NULL.
<code>imputedData</code>	Data frame comprising imputed test data. First two columns are censoring and survival time, respectively. The remaining columns are the x-variables. Row <code>i</code> contains imputed outcomes and x-variables for row <code>imputedIndv[i]</code> of <code>predictors</code> . Can be NULL.

Note

The key deliverable is the matrix `ensemble` which contains the ensemble CHF for each individual in the test data evaluated at a set of distinct time points.

Author(s)

Hemant Ishwaran (hemant.ishwaran@gmail.com) and Udaya B. Kogalur (ubk2101@columbia.edu)

References

- L. Breiman (2001). Random forests, *Machine Learning*, 45:5-32.
- H. Ishwaran, U.B. Kogalur, E.H. Blackstone and M.S. Lauer (2008). Random survival forests, *To appear in Ann. App. Statist.*.
- H. Ishwaran, U.B. Kogalur (2007). Random survival forests for R, *Rnews*, 7/2:25-31.

See Also

`rsf`, `print.rsf`, `plot.ensemble`, `plot.variable`, `plot.error`, `plot.proximity`, `pmml2rsf`, `rsf2pmml`.

Examples

```
data(veteran, package = "randomSurvivalForest")
train.pt <- sample(1:nrow(veteran), round(nrow(veteran)*0.80))
veteran.out <- rsf(Survrsf(time, status) ~ ., forest = TRUE,
                  data = veteran[train.pt , ])
veteran.pred <- predict.rsf(veteran.out, veteran[-train.pt , ])
```

print.rsf

Print Summary Output of Analysis

Description

Print summary output from Random Survival Forests analysis. Note that this is the default print method for the package.

Usage

```
print.rsf(x, ...)
```

Arguments

`x` An object of class `(rsf, grow)` or `(rsf, predict)`.
`...` Further arguments passed to or from other methods.

Author(s)

Hemant Ishwaran (hemant.ishwaran@gmail.com) and Udaya B. Kogalur (ubk2101@columbia.edu)

References

H. Ishwaran, U.B. Kogalur, E.H. Blackstone and M.S. Lauer (2008). Random survival forests, *To appear in Ann. App. Statist.*

See Also

`rsf`, `predict.rsf`.

Examples

```
data(veteran, package = "randomSurvivalForest")
v.out=rsf(Survrsf(time, status)~.,veteran, ntree = 1000)
print.rsf(v.out)
```

 recid

Recidivism Data

Description

This data comprises 432 male inmates released from Maryland state prisons in the early 1970's. Dates of arrest, which constitutes an event, were recorded for each inmate up to 1 year following release. A major goal of the analysis was to determine if those who received financial aid had lower arrest rates than those who did not. Moreover, the effect of having financial aid is likely to be more important immediately following arrest and less important as time passes.

Format

A data frame containing:

week	survival time
arrest	censoring variable
fin	financial aid, 0=no 1=yes
age	age in years
race	race (1=black, 0=otherwise)
wexp	full time work experience (0=no, 1=yes)
mar	married (0=no, 1=yes)
paro	released on parole (0=no, 1=yes)
prio	number of prior convictions

Source

Rossi, Berk and Lenihan, 1980.

References

P.H. Rossi and R.A. Berk and K.J. Lenihan (1980). *Money, Work and Crime: Some Experimental Results*, Academic Press:New York.

Examples

```
data(recid, package = "randomSurvivalForest")
```

 rsf.default

Random Survival Forest Entry Point

Description

Random Survival Forests (RSF) (Ishwaran, Kogalur, Blackstone and Lauer, 2008) is an extension of Breiman's Random Forests (Breiman, 2001) to right-censored survival analysis settings. A forest of survival trees is grown and used to estimate an ensemble cumulative hazard function (CHF). Trees can be grown using different survival tree splitting rules. An "out-of-bag" estimate of Harrell's concordance index (Harrell, 1982) is provided for assessing prediction accuracy of the CHF. Variable importance (VIMP) can be computed for single, as well as grouped variables, as a means to filter variables and to assess variable predictiveness. RSF can be used to predict on test data. Missing data (x-variables, survival times, censoring indicators) can be imputed on both training and test data. Note this is the default generic method for the package.

Usage

```
## Default S3 method:
rsf(formula,
     data = NULL,
     ntree = 1000,
     mtry = NULL,
     nodesize = NULL,
     splitrule = c("logrank", "conserve", "logrankscore", "random")[1],
     nsplit = 0,
     importance = c("randomsplit", "permute", "none")[1],
     big.data = FALSE,
     na.action = c("na.omit", "na.impute")[1],
     nimpute = 1,
     predictorWt = NULL,
     forest = FALSE,
     proximity = FALSE,
     varUsed = NULL,
     seed = NULL,
     do.trace = FALSE,
     ...)
```

Arguments

<code>formula</code>	A symbolic description of the model to be fit. Details for model specification are given below.
<code>data</code>	Data frame containing the data used in the formula. Missing values allowed. See <code>na.action</code> for details.
<code>ntree</code>	Number of trees to grow. This should not be set to a number too small, in order to ensure that every input row gets predicted at least a few times.
<code>mtry</code>	Number of variables randomly sampled at each split. The default is \sqrt{p} , where p equals the number of variables.
<code>nodesize</code>	Minimum number of deaths with unique survival times required for a terminal node. Default is roughly $\min(3, \text{round}(0.632 * \text{ndead}))$. Larger values cause smaller trees to be grown.
<code>splitrule</code>	Splitting rule used to grow trees. See details below.

<code>nsplit</code>	Non-negative integer value. If non-zero, a random version of the specified tree splitting rule is implemented. This can significantly increase speed. See details below.
<code>importance</code>	Method used to compute variable importance. See details below.
<code>big.data</code>	Logical. Set this value to true when the number of variables <code>p</code> is <i>very</i> large, or the data is very large. See details below.
<code>na.action</code>	Action to be taken if the data contain NA's. Possible values are <code>na.omit</code> and <code>na.impute</code> . Default is <code>na.omit</code> , which removes the entire record if even one of its entries is NA (for x-variables this applies only to those specifically listed in 'formula'). The action <code>na.impute</code> implements a sophisticated tree imputation technique. See details below.
<code>nimpute</code>	Number of iterations of missing data algorithm.
<code>predictorWt</code>	Vector of non-negative weights where entry <code>k</code> , after normalizing, is the probability of selecting variable <code>k</code> as a candidate for splitting. Default is to use uniform weights. Vector must be of dimension <code>p</code> , where <code>p</code> equals the number of variables.
<code>forest</code>	Logical. Should the forest object be returned? Used for prediction on new data. Default is FALSE.
<code>proximity</code>	Logical. Should proximity measure between observations be calculated? Creates an <code>n</code> \times <code>n</code> matrix (which can be huge). Default is FALSE.
<code>varUsed</code>	Analyzes which variables are used (split upon) in the topology of the forest. Default is NULL. Possible values are <code>all.trees</code> , <code>by.tree</code> . See details below.
<code>seed</code>	Seed for random number generator. Must be a negative integer (the R wrapper handles incorrectly set seed values).
<code>do.trace</code>	Logical. Should trace output be enabled? Default is FALSE. Integer values can also be passed. A positive value causes output to be printed each <code>do.trace</code> iteration.
<code>...</code>	Further arguments passed to or from other methods.

Details

Four primary splitting rules are available for growing a survival forest. The default rule, `logrank`, splits tree nodes by maximization of the log-rank test statistic (Segal, 1988; Leblanc and Crowley, 1993). A second rule, `conserve`, splits nodes by finding daughters closest to the conservation of events principle (see Naftel, Blackstone and Turner, 1985). A third rule, `logrankscore`, uses a standardized log-rank statistic (Hothorn and Lausen, 2003). A fourth rule, `random`, implements pure random splitting. For each node, a variable is randomly selected from a random set of `mtry` variables and the node is split using a random split point (Lin and Jeon, 2006).

A random version of the `logrank`, `conserve` and `logrankscore` splitting rules can be invoked using `nsplit`. If `nsplit` is set to a non-zero positive integer, then a maximum of `nsplit` split points are chosen randomly for each of the `mtry` variables within a node (this is in contrast to deterministic splitting where all possible split points for each of the `mtry` variables are considered). The splitting rule is applied to these random split points and the node is split on that variable and

random split point maximizing survival difference (as measured by the splitting rule). Note that `nsplit` has no effect if the splitting rule is `random`.

In terms of performance, a detailed study carried out by Ishwaran et al. (2008) found `logrank` and `logrankscore` to be the most accurate in terms of prediction error, followed by `conserve`. Setting `nsplit=1` and using `logrank` splitting gave performance close to `logrank`, but with significantly shorter computational times. Accuracy can be good further improved without overly compromising speed by using larger values of `nsplit`.

In addition to random splitting, computation times for *very* large data sets can be improved by discretizing continuous variables and/or the observed survival times. Discretization does not have to be overly granular for substantial gains to be seen. Users may also consider setting `big.data=TRUE` for data with a large number of variables. This bypasses the large overhead R needs to create design matrices and parse formula. Be aware, however, that variables are not processed and are interpreted *as is* when this option is turned on. Think of the data frame as containing time and censoring information and the rest of the data as the pre-processed design matrix when this option is on. In particular, transformations used in the formula (such as logs etc.) are ignored.

A typical RSF formula has the form `Survrsf(time, censoring) ~ terms`, where `time` is survival time and `censoring` is a binary censoring indicator. Note that censoring must be coded as 0=censored and 1=death (event) and `time` must be strictly positive.

Variables encoded as factors are treated as such. If the factor is ordered, then splits are similar to real valued variables. If the factor is unordered, a split will move a subset of the levels in the parent node to the left daughter, and the complementary subset to the right daughter. All possible complementary pairs are considered and apply to factors with an unlimited number of levels. However, there is an optimization check to ensure that the number of splits attempted is not greater than the number of cases in a node (this internal check will override the `nsplit` value in random splitting mode if `nsplit` is large enough). Note that when predicting on test data involving factors, the factor labels in the test data must be the same as in the `grow` (training) data. Consider setting labels that are unique in the test data to missing.

Other than factors, all other `x`-variables are coerced and treated as being real valued.

Variable importance (VIMP) is computed similar to Breiman (2001), although there are two ways to perturb a variable to determine its VIMP: `randomsplit`, `permute`. The default method is `randomsplit` which works as follows. To calculate VIMP for a variable `x`, out-of-bag (OOB) cases are dropped down the bootstrap (in-bag) survival tree. A case is assigned a daughter node randomly whenever an `x`-split is encountered. An OOB ensemble cumulative hazard function (CHF) is computed from the forest of such trees and its OOB error rate calculated. The VIMP for `x` is the difference between this and the OOB error rate for the original forest (without random node assignment using `x`). If `permute` is used, then `x` is randomly permuted in OOB data and dropped down the in-bag tree. See Ishwaran et al. (2008) for further details.

Prediction error is measured by $1-C$, where C is Harrell's concordance index. Prediction error is between 0 and 1, and measures how well the ensemble correctly ranks (classifies) any two individuals in terms of survival. A value of 0.5 is no better than random guessing. A value of 0 is perfect. Because VIMP is based on the concordance index, VIMP indicates how much misclassification increases, or decreases, for a new test case if a given variable were not available for that case (given that the forest was grown using that variable).

Setting `na.action` to `na.impute` implements a tree imputation method whereby missing data (`x`-variables or outcomes) are imputed dynamically as a tree is grown by randomly sampling from the distribution within the current node (Ishwaran et al. 2008). OOB data is not used in imputation

to avoid biasing prediction error and VIMP estimates. Final imputation for integer valued variables and censoring indicators use a maximal class rule, whereas continuous variables and survival time use a mean rule. Records in which all outcome and x-variable information are missing are removed. Variables having all missing values are removed. The algorithm can be iterated by setting `nimpute` to a positive integer greater than 1. A few iterations should be used in heavy missing data settings to improve accuracy of imputed values (see Ishwaran et al., 2008). Note if the algorithm is iterated, a side effect is that missing values in returned objects `predictors`, `time` and `cens` are replaced by imputed values. Further, imputed objects such as `imputedData` are set to `NULL`.

If `varUsed=all.trees`, a vector of size `p` is returned. Each element contains a count of the number of times a split has occurred on this variable. If `varUsed=by.tree`, a matrix of size `ntreexp` is returned. Each element `[i][j]` contains a count of the number of times a split has occurred on variable `[j]` in tree `[i]`.

Value

An object of class `(rsf, grow)`, which is a list with the following components:

<code>call</code>	The original call to <code>rsf</code> .
<code>formula</code>	The formula used in the call.
<code>n</code>	Sample size of the data (depends upon NA's, see <code>na.action</code>).
<code>ndead</code>	Number of deaths.
<code>ntree</code>	Number of trees grown.
<code>mtry</code>	Number of variables randomly selected for splitting at each node.
<code>nodesize</code>	Minimum size of terminal nodes.
<code>splitrule</code>	Splitting rule used.
<code>nsplit</code>	Number of randomly selected split points.
<code>time</code>	Vector of length <code>n</code> of survival times.
<code>cens</code>	Vector of length <code>n</code> of censoring information (0=censored, 1=death).
<code>timeInterest</code>	Sorted unique event times. Ensemble values are given for these time points only.
<code>predictorNames</code>	A character vector of the variable names used in growing the forest.
<code>predictorWt</code>	Vector of non-negative weights used for randomly sampling variables for splitting.
<code>predictors</code>	Data frame comprising x-variables used to grow the forest.
<code>ensemble</code>	A matrix of the bootstrap ensemble CHF with each row corresponding to an individual's CHF evaluated at each of the time points in <code>timeInterest</code> .
<code>oob.ensemble</code>	Same as <code>ensemble</code> , but based on the OOB CHF.
<code>mortality</code>	A vector of length <code>n</code> , with each value containing the bootstrap ensemble mortality for an individual in the data. Ensemble mortality values should be interpreted in terms of total number of deaths.
<code>oob.mortality</code>	Same as <code>mortality</code> , but based on <code>oob.ensemble</code> .

<code>err.rate</code>	Vector of length <code>ntree</code> containing OOB error rates for the ensemble, with the <code>b</code> -th element being the error rate for the ensemble formed using the first <code>b</code> trees. Error rates are measured using $1-C$, where C is Harrell's concordance index.
<code>leaf.count</code>	Number of terminal nodes for each tree in the forest. Vector of length <code>ntree</code> . A value of zero indicates a rejected tree (sometimes occurs when imputing missing data). Values of one indicate tree stumps.
<code>importance</code>	VIMP for each variable.
<code>forest</code>	If <code>forest=TRUE</code> , the forest object is returned. This object can then be used for prediction with new test data sets.
<code>proximity</code>	If <code>proximity=TRUE</code> , a matrix of dimension <code>n</code> by <code>n</code> recording the frequency pairs of data points occur within the same terminal node. Value returned is a vector of the lower diagonal of the matrix. Use <code>plot.proximity()</code> to extract this information.
<code>varUsed</code>	Count of the number of times a variable is used in growing the forest. Can be a vector, matrix, or NULL.
<code>imputedIndv</code>	Vector of indices for cases with missing values. Can be NULL.
<code>imputedData</code>	Data frame comprising imputed data. First two columns are censoring and survival time, respectively. Remaining columns are the <code>x</code> -variables. Row <code>i</code> contains imputed outcomes and <code>x</code> -variables for row <code>imputedIndv[i]</code> of <code>predictors</code> . Can be NULL.

Note

The key deliverable is the matrix `ensemble` containing the bootstrap ensemble CHF function for each individual evaluated at a set of distinct time points (an OOB ensemble, `oob.ensemble`, is also returned). The vector `mortality` (likewise `oob.mortality`) is a weighted sum over the columns of `ensemble`, weighted by the number of individuals at risk at the different time points. Entry `i` of the vector represents the estimated total mortality of individual `i` in terms of total number of deaths. In other words, if `i` has a mortality value of 100, then if all individuals had the same `x`-values as `i`, there would be on average 100 deaths in the dataset.

Different R wrappers are provided with the package to aid in interpreting the ensemble.

Author(s)

Hemant Ishwaran (hemant.ishwaran@gmail.com) and Udaya B. Kogalur (ubk2101@columbia.edu)

References

- L. Breiman (2001). Random forests, *Machine Learning*, 45:5-32.
- F.E. Harrell et al. (1982). Evaluating the yield of medical tests, *J. Amer. Med. Assoc.*, 247:2543-2546.
- T. Hothorn and B. Lausen (2003). On the exact distribution of maximally selected rank statistics, *Comp. Stat. Data Anal.*, 43:121-137.
- H. Ishwaran, U.B. Kogalur, E.H. Blackstone and M.S. Lauer (2008). Random survival forests, *To appear in Ann. App. Statist.*.

- H. Ishwaran, U.B. Kogalur (2007). Random survival forests for R, *Rnews*, 7/2:25-31.
- H. Ishwaran (2007). Variable importance in binary regression trees and forests, *Electronic J. Statist.*, 1:519-537.
- M. LeBlanc and J. Crowley (1993). Survival trees by goodness of split, *J. Amer. Stat. Assoc.*, 88:457-467.
- A. Liaw and M. Wiener (2002). Classification and regression by randomForest, *R News*, 2:18-22.
- Y. Lin and Y. Jeon (2006). Random forests and adaptive nearest neighbors, *J. Amer. Stat. Assoc.*, 101:578-590.
- D.C. Naftel, E.H. Blackstone and M.E. Turner (1985). Conservation of events, unpublished notes.
- M. R. Segal. (1988). Regression trees for censored data, *Biometrics*, 44:35-47.

See Also

plot.ensemble, plot.variable, plot.error, plot.proximity, predict.rsfc, print.rsfc, find.interaction, pmml2rsfc, rsfc2pmml, Survrsfc.

Examples

```
#-----
# Example 1: Veteran's Administration lung cancer trial from
# Kalbfleisch & Prentice. Randomized trial of two treatment
# regimens for lung cancer. Minimal argument call. Print
# results, then plot error rate and importance values.

data(veteran, package = "randomSurvivalForest")
veteran.out <- rsf(Survrsf(time, status)~., data = veteran)
print(veteran.out)
plot(veteran.out)

#-----
# Example 2: Richer argument call (veteran data).
# Forest is saved by setting 'forest' option to true
# (see 'rsfc.predict' for more details about prediction).
# Coerce variable 'celltype' as a factor, and karnofsky score
# as an ordered factor to illustrate factor useage in RSF.
# Use random splitting with 'nsplit'.
# Use 'varUsed' option.

data(veteran, package = "randomSurvivalForest")
veteran.f <- as.formula(Survrsf(time, status)~.)
veteran$celltype <- factor(veteran$celltype,
  labels=c("squamous", "smallcell", "adeno", "large"))
veteran$karno <- factor(veteran$karno, ordered = TRUE)
ntree <- 200
mtry <- 2
nodesize <- 3
splitrule <- "logrank"
nsplit <- 10
varUsed <- "by.tree"
forest <- TRUE
```

```

proximity <- TRUE
do.trace <- 1
veteran2.out <- rsf(veteran.f, veteran, ntree,
  mtry, nodesize, splitrule, nsplit,
  varUsed = varUsed, forest = forest,
  proximity = proximity, do.trace = do.trace)
print(veteran2.out)
plot.proximity(veteran2.out)

# Take a peek at the forest ...
head(veteran2.out$forest$nativeArray)

# Average number of times a variable was split on.
apply(veteran2.out$varUsed, 2, mean)

# Partial plot of top variable.
plot.variable(veteran2.out, partial = TRUE, npred=1)

## Not run:
#-----
# Example 3: Veteran data (again).
# Consider Karnofsky performance score. Compare to Kaplan-Meier.
# Assumes "survival" library is loaded.

if (library("survival", logical.return = TRUE))
{
  data(veteran, package = "randomSurvivalForest")
  veteran3.out <- rsf(Survrsf(time, status)~karno,
    veteran,
    ntree = 1000)
  plot.ensemble(veteran3.out)
  par(mfrow = c(1,1))
  plot(survfit(Surv(time, status)~karno, data = veteran))
}

#-----
# Example 4: Primary biliary cirrhosis (PBC) of the liver.
# Data found in Appendix D.1 of Fleming and Harrington, Counting
# Processes and Survival Analysis, Wiley, 1991 (modified so
# that age is in days and sex and stage variables are not
# missing for observations 313-418).

data(pbc, package = "randomSurvivalForest")
pbc.out <- rsf(Survrsf(days,status)~., pbc, ntree = 1000)
print(pbc.out)

#-----
# Example 5: Same as Example 4, but with imputation for
# missing values.

data(pbc, package = "randomSurvivalForest")
pbc2.out <- rsf(Survrsf(days,status)~., pbc, ntree = 1000,
  na.action="na.impute")

```

```

# summary of analysis
print(pbc2.out)
# Combine original data + imputed data.
pbc.imputed.data <- cbind(status=pbc2.out$cens, days=pbc2.out$time,
                          pbc2.out$predictors)
pbc.imputed.data[pbc2.out$imputedIndv,] <- pbc2.out$imputedData
tail(pbc)
tail(pbc.imputed.data)
# Iterate the missing data algorithm.
# Use logrank random splitting (with nsplit=5) to increase speed.
# Use trace to track algorithm in detail.
# Note that a side effect of iterating is that the original data
# are replaced by imputed values.
pbc3.out <- rsf(Survrsf(days,status)~., pbc, ntree = 1000, nsplit=5,
               na.action="na.impute", nimpute=3, do.trace = TRUE)
pbc.iterate.imputed.data <- cbind(status=pbc3.out$cens,
                                  days=pbc3.out$time, pbc3.out$predictors)

#-----
# Example 6: Compare Cox regression to RSF (PBC data).
# Compute OOB estimate of Harrell's concordance
# index for Cox regression using B = 100 bootstrap draws.
# Assumes "Hmisc" and "survival" libraries are loaded.

if (library("survival", logical.return = TRUE)
    & library("Hmisc", logical.return = TRUE))
{
  data(pbc, package = "randomSurvivalForest")
  pbc3.out <- rsf(Survrsf(days,status)~., pbc, mtry = 2, ntree = 1000)
  B <- 100
  cox.err <- rep(NA, B)
  cox.f <- as.formula(Surv(days,status)~.)
  pbc.data <- pbc[apply(is.na(pbc), 1, sum) == 0,] ##remove NA's
  cat("Out-of-bag Cox Analysis ...", "\n")
  for (b in 1:B) {
    cat("Cox bootstrap:", b, "\n")
    bag.sample <- sample(1:nrow(pbc.data),
                        nrow(pbc.data),
                        replace = TRUE)
    oob.sample <- setdiff(1:nrow(pbc.data), bag.sample)
    train <- pbc.data[bag.sample,]
    test <- pbc.data[oob.sample,]
    cox.out <- coxph(cox.f, train)
    cox.out <- tryCatch({coxph(cox.f, train)}, error=function(ex){NULL})
    if (is.list(cox.out)) {
      cox.predict <- predict(cox.out, test)
      cox.err[b] <- rcorr.cens(cox.predict,
                             Surv(pbc.data$days[oob.sample],
                                  pbc.data$status[oob.sample]))[1]
    }
  }
  cat("Error rates:", "\n")
  cat("Random Survival Forests:", pbc3.out$err.rate[pbc3.out$ntree], "\n")
}

```

```

    cat("          Cox Regression:", mean(cox.err, na.rm = TRUE), "\n")
  }
  ## End(Not run)

```

rsf.news

Show the NEWS file

Description

Show the NEWS file of the **randomSurvivalForest** package.

Usage

```
rsf.news(...)
```

Arguments

... Further arguments passed to or from other methods.

Value

None.

Author(s)

Hemant Ishwaran (hemant.ishwaran@gmail.com) and Udaya B. Kogalur (ubk2101@columbia.edu)

rsf2pmml

Save Random Survival Forest As PMML

Description

`rsf2pmml` implements the Predictive Model Markup Language specification for a **randomSurvivalForest** forest object. In particular, this function gives the user the ability to save the geometry of a forest as a PMML XML document.

Usage

```
rsf2pmml(rsfForest, ...)
```

Arguments

`rsfForest` The forest object contained in an object of class **randomSurvivalForest**, as that contained in the object returned by the function `rsf` with the parameter “forest=TRUE”.

... Further arguments passed to or from other methods.

Details

The Predictive Model Markup Language is an XML based language which provides a way for applications to define statistical and data mining models and to share models between PMML compliant applications. More information about PMML and the Data Mining Group can be found at <http://www.dmg.org>.

Use of PMML and `rsf2pmml` requires the **XML** package. Be aware that XML is a very verbose data format. Reasonably sized trees and data sets can lead to extremely large text files. XML, while achieving interoperability, is not an efficient data storage mechanism in this case.

It is anticipated that `rsf2pmml` will be used to export the geometry of the forest to other PMML compliant applications, including graphics packages that are capable of printing binary trees. In addition, the user may wish to save the geometry of the forest for later retrieval and prediction on new data sets using `rsf2pmml` together with `pmml2rsf`.

Value

An object of class `XMLNode` as that defined by the **XML** package. This represents the top level, or root node, of the XML document and is of type PMML.

Note

One cautionary note is in order. The PMML representation of the **randomSurvivalForest** forest object is incomplete, in that the object needs to be massaged in order for prediction to be possible. This will be clear in the examples. This deficiency will be addressed in future releases of this package. However, it was felt that the current functionality was important enough and mature enough to warrant release in this version of the product.

Author(s)

Hemant Ishwaran (hemant.ishwaran@gmail.com) and Udaya B. Kogalur (ubk2101@columbia.edu)

References

H. Ishwaran, Udaya B. Kogalur, Eugene H. Blackstone and Michael S. Lauer (2007). Random Survival Forests. *Cleveland Clinic Technical Report*.
<http://www.dmg.org>

See Also

`xmlTreeParse`, `xmlRoot`, `saveXML`, `rsf`, `predict.rsfc`, `pmml2rsfc`.

Examples

```
# Example 1: Growing a forest, saving it as a PMML document,  
# restoring the forest from the PMML document, and using this forest to  
# perform prediction.  
  
## Not run:  
library("XML")
```

```

data(veteran, package = "randomSurvivalForest")
veteran.out <- rsf(Survrsf(time, status)~.,
  data = veteran,
  ntree = 5,
  forest = TRUE)
veteran.forest <- veteran.out$forest
veteran.pmml <- rsf2pmml(veteran.forest)

# Save the document to disk.
saveXML(veteran.pmml, "veteran.forest.xml")
veteran.pmml <- xmlRoot(xmlTreeParse("veteran.forest.xml"))

partial.forest <- pmml2rsf(veteran.pmml)

# The PMML forest object must be massaged before it can be used
# for prediction as follows:
veteran.restored.forest <- list(
  nativeArray=partial.forest$nativeArray,
  nativeFactorArray=partial.forest$nativeFactorArray,
  timeInterest=partial.forest$timeInterest,
  predictorNames=partial.forest$predictorNames,
  seed=partial.forest$seed,
  formula=partialForest$formula,
  predictors=veteran.forest$predictors,
  time=veteran.forest$time,
  cens=veteran.forest$cens)

# The actual time, censoring and prediction values of the data set
# used to grow the forest are not contained in the PMML
# representation of the forest. If the user has access to the original
# datafile that was used to grow the forest, this information can be
# easily recovered. The names corresponding to the time, censoring and
# prediction data are all retained in the PMML representation of the forest.

class(veteran.restored.forest) <- c("rsf", "forest")
veteran.restored.out <- predict.rsf(veteran.restored.forest, test=veteran)

## End(Not run)

```

Survrsf

Definition of Time and Censoring Information

Description

Survrsf defines the time and censoring information that will be extracted from the data.

Usage

```
Survrsf(t, d)
```

Arguments

t survival time
 d censoring indicator (0=censored, 1=death)

Value

An object of class `rsf.formula`.

Author(s)

Hemant Ishwaran (hemant.ishwaran@gmail.com) and Udaya B. Kogalur (ubk2101@columbia.edu)

References

H. Ishwaran, Udaya B. Kogalur, Eugene H. Blackstone and Michael S. Lauer (2007). Random Survival Forests. *Cleveland Clinic Technical Report*.

See Also

`rsf`.

Examples

```
data(veteran, package = "randomSurvivalForest")
v.out <- rsf(Survrsf(time, status)~., veteran, ntree = 1000)
```

veteran

Veteran's Administration Lung Cancer Trial

Description

Randomized trial of two treatment regimens for lung cancer. This is a standard survival analysis data set.

Format

A data frame containing:

trt	treatment: 1=standard 2=test
celltype	cell-type: 1=squamous, 2=smallcell, 3=adeno, 4=large
time	survival time
status	censoring status
karno	Karnofsky performance score (100=good)
diagtime	months from diagnosis to randomisation
age	age in years
prior	prior therapy 0=no, 1=yes

Source

Kalbfleisch and Prentice, *The Statistical Analysis of Failure Time Data*.

References

Kalbfleisch J. and Prentice R, (1980) *The Statistical Analysis of Failure Time Data*. New York: Wiley.

Examples

```
data(veteran, package = "randomSurvivalForest")
```

Index

*Topic **datasets**

burn, 1
pbc, 7
recid, 19
veteran, 32

*Topic **documentation**

rsf.news, 28

*Topic **file**

find.interaction, 2
interaction.rsfc, 4
plot.ensemble, 8
plot.error, 9
plot.proximity, 10
plot.variable, 11
print.rsfc, 18
Survrsfc, 31

*Topic **survival**

pmml2rsfc, 14
predict.rsfc, 16
rsfc.default, 20
rsfc2pmml, 29

*Topic **tree**

pmml2rsfc, 14
predict.rsfc, 16
rsfc.default, 20
rsfc2pmml, 29

burn, 1

find.interaction, 2

interaction.rsfc, 4

pbc, 7

plot.ensemble, 8

plot.error, 9

plot.proximity, 10

plot.rsfc(plot.error), 9

plot.variable, 11

pmml2rsfc, 14

predict.rsfc, 16

print.rsfc, 18

randomSurvivalForest
(rsfc.default), 20

recid, 19

rsfc(rsfc.default), 20

rsfc.default, 20

rsfc.news, 28

rsfc2pmml, 29

Survrsfc, 31

veteran, 32