

# Package ‘randomSurvivalForest’

January 2, 2012

**Version** 3.6.3

**Date** 2010-05-21

**Title** Random Survival Forests

**Author** Hemant Ishwaran <hemant.ishwaran@gmail.com> and Udaya B. Kogalur <kogalurshear@gmail.com>

**Maintainer** Udaya B. Kogalur <kogalurshear@gmail.com>

**Depends** R (>= 2.8.0)

**Suggests** XML

**Description** Random survival forests for right-censored and competing risks survival data.

**License** GPL (>= 2)

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

**Repository** CRAN

**Date/Publication** 2010-05-26 14:15:14

## R topics documented:

breast . . . . .	2
competing.risk . . . . .	2
find.interaction . . . . .	3
follic . . . . .	6
hd . . . . .	6
impute.rsfc . . . . .	6
max.subtree . . . . .	8
pbc . . . . .	10
plot.ensemble . . . . .	10
plot.error . . . . .	11
plot.proximity . . . . .	13

plot.variable . . . . .	14
pmml2rsf . . . . .	16
predict.rsfc . . . . .	18
print.rsfc . . . . .	22
rsfc . . . . .	23
rsfc.news . . . . .	32
rsfc2pmml . . . . .	33
rsfc2rfz . . . . .	35
varSel . . . . .	36
vdv . . . . .	40
veteran . . . . .	40
vimp . . . . .	41
wihs . . . . .	43

<b>Index</b>	<b>45</b>
--------------	-----------

---

breast	<i>German Breast Cancer Data</i>
--------	----------------------------------

---

### Description

Breast cancer survival data.

### References

M. Schumacher, et al. (1994). Randomized  $2 \times 2$  trial evaluating hormonal treatment and the duration of chemotherapy in node-positive breast cancer patients. The German Breast Cancer Study Group, *J. Clinical Oncology*, 12:2086-2093.

---

competing.risk	<i>Summary Plots for Competing Risks</i>
----------------	--

---

### Description

Plot the ensemble cumulative incidence function (CIF), ensemble subsurvival function, ensemble conditional survival function, and ensemble unconditional survival function from a random survival forests competing risk analysis (Ishwaran et al., 2010).

### Usage

```
competing.risk(x, plot = TRUE, ...)
```

### Arguments

x	An object of class (rsfc, grow) or (rsfc, predict).
plot	Should curves be plotted?
...	Further arguments passed to or from other methods.

**Details**

From top to bottom, left to right are plots of the: (a) ensemble CIF; (b) ensemble subsurvival function; (c) ensemble conditional survival functions; and (d) ensemble (unconditional) survival function. See Gray (1988) for motivation regarding the CIF and subsurvival functions.

For right-censored data, only the survival function is plotted.

Whenever possible, out-of-bag (OOB) values are plotted.

**Value**

Invisibly, the ensemble CIF, ensemble subsurvival function and conditional mortality for each event type. Whenever possible, OOB values are returned.

**Author(s)**

Hemant Ishwaran <hemant.ishwaran@gmail.com>

Udaya B. Kogalur <kogalurshear@gmail.com>

**References**

Gray R.J. (1988). A class of k-sample tests for comparing the cumulative incidence of a competing risk, *Ann. Statist.*, 16:1141-1154.

Ishwaran H., Kogalur U.B., Moore R.D., Gange S.J. and Lau B.M. (2010). Random survival forests for competing risks.

**See Also**

rsf, wihs.

**Examples**

```
## Not run:
data(follic, package = "randomSurvivalForest")
follic.out <- rsf(Surv(time, status) ~ ., follic, nsplit = 3, ntree = 100)
competing.risk(follic.out)

## End(Not run)
```

---

find.interaction

*Find Interactions Between Pairs of Variables*

---

**Description**

Find pairwise interactions between variables.

**Usage**

```
find.interaction(object, predictorNames = NULL,
  method = c("maxsubtree", "vimp")[1], 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).
predictorNames	Character vector of names of target x-variables. Default is to use all variables.
method	Method of analysis: maximal subtree or VIMP. See details below.
sorted	Should variables be sorted?
npred	Use the first npred ordered variables. Default is to use all variables.
subset	Indices indicating which rows of the predictor matrix to be used (note: this applies to the <i>object</i> predictor matrix, predictors). Default is to use all rows.
nrep	Number of Monte Carlo replicates. Applies only when method="vimp".
rough	Should fast approximation be used? Applies only when method="vimp".
importance	Type of variable importance (VIMP). Applies only when method="vimp".
seed	Seed (negative integer) for random number generator.
do.trace	Logical. Should trace output be enabled? Integer values can also be passed. A positive value causes output to be printed each do.trace iteration. Applies only when method="vimp".
...	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. There are two distinct approaches specified by the `method` option.

If `method="maxsubtree"`, then a maximal subtree analysis is used. In this case, a matrix is returned where entries  $[i][i]$  are the normalized minimal depth of variable  $[i]$  relative to the root node (normalized w.r.t. the size of the tree) and entries  $[i][j]$  indicate the normalized minimal depth of a variable  $[j]$  w.r.t. the maximal subtree for variable  $[i]$  (normalized w.r.t. the size of  $[i]$ 's maximal subtree). Smaller  $[i][i]$  entries indicate predictive variables. Small  $[i][j]$  entries having small  $[i][i]$  entries are a sign of an interaction between variable  $i$  and  $j$  (note: the user should scan rows, not columns, for small entries). See Ishwaran et al. (2010) for more details.

If `method="vimp"`, then a joint-VIMP approach is used. 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. See Ishwaran (2007) for more details.

Computations might be slow depending upon the size of the data and the forest. In such cases, consider setting `npred` to a smaller number, or using `rough=TRUE` if `method="vimp"`. If `method="maxsubtree"`, consider using a smaller number of trees in the original `grow` call.

If `nrep` is greater than 1, the analysis is repeated `nrep` times and results averaged over the replications (applies only when `method="vimp"`).

For competing risk data, maximal subtree analyses correspond to unconditional values (i.e., they are non-event specific). Setting `method="vimp"`, however, yields pairwise interactions for both event and non-event specific settings.

## Value

Invisibly, the interaction table (a list for competing risk data) or the maximal subtree matrix.

## Author(s)

Hemant Ishwaran <hemant.ishwaran@gmail.com>

Udaya B. Kogalur <kogalurshear@gmail.com>

## References

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

Ishwaran H., Kogalur U.B., Gorodeski E.Z, Minn A.J. and Lauer M.S. (2010). High-dimensional variable selection for survival data. *J. Amer. Statist. Assoc.*, 105:205-217.

## See Also

`max.subtree`, `vimp`.

## Examples

```
## Not run:
#-----
# Maximal subtree approach, top 8 predictors (PBC data).

data(pbc, package = "randomSurvivalForest")
pbc.out <- rsf(Surv(days,status) ~ ., pbc, nsplit = 10)
find.interaction(pbc.out, npred = 8)

#-----
# VIMP approach (PBC data).
# Use fast approximation to speed up computations.

data(pbc, package = "randomSurvivalForest")
pbc.out <- rsf(Surv(days,status) ~ ., pbc, nsplit = 10)
find.interaction(pbc.out, method = "vimp", nrep=3, rough=T)

#-----
# Competing risks (WIHS data).

data(wihs, package = "randomSurvivalForest")
wihs.out <- rsf(Surv(time, status) ~ ., wihs, nsplit = 3, ntree = 200)
find.interaction(wihs.out, method = "vimp")
```

```
## End(Not run)
```

---

follic	<i>Follicular Cell Lymphoma Data</i>
--------	--------------------------------------

---

### Description

Competing risk data set involving follicular cell lymphoma (Table 1.4b, *Competing Risks: A Practical Perspective*).

### References

Pintilie M., (2006) *Competing Risks: A Practical Perspective*. West Sussex: John Wiley and Sons.

---

hd	<i>Hodgkin's Disease Data</i>
----	-------------------------------

---

### Description

Competing risk data set involving Hodgkin's disease (Table 1.6b, *Competing Risks: A Practical Perspective*).

### References

Pintilie M., (2006) *Competing Risks: A Practical Perspective*. West Sussex: John Wiley and Sons.

---

impute.rsf	<i>Impute Only Mode</i>
------------	-------------------------

---

### Description

Imputation for right censored survival and competing risk data. A random survival forest is grown and used to impute missing data. No ensemble estimates or error rates are calculated. This is a fast way to impute data.

### Usage

```
impute.rsf(formula, data = NULL, ntree = 1000, mtry = NULL,
  nodesize = NULL, splitrule = NULL, nsplit = 0, big.data = FALSE,
  nimpute = 1, predictorWt = NULL, seed = NULL, do.trace = FALSE,
  ...)
```

**Arguments**

formula	A symbolic description of the model to be fit.
data	Data frame containing the data to be imputed.
ntree	Number of trees to grow.
mtry	Number of variables randomly sampled at each split.
nodesize	Minimum terminal node size.
splitrule	Splitting rule used to grow trees.
nsplit	Non-negative integer value used to specify random splitting.
big.data	Set this value to TRUE for large data.
nimpute	Number of iterations of missing data algorithm.
predictorWt	Weights for selecting variables for splitting on.
seed	Seed for random number generator.
do.trace	Should trace output be enabled?
...	Further arguments passed to or from other methods.

**Details**

Grows a RSF and uses this to impute missing data. All external calculations such as ensemble calculations, error rates, etc. are turned off. Use this function if your only interest is imputing the data.

All options are the same as for `rsf`.

**Value**

Invisibly, the data frame containing the original data with imputed data overlaid.

**Author(s)**

Hemant Ishwaran <hemant.ishwaran@gmail.com>

Udaya B. Kogalur <kogalurshear@gmail.com>

**References**

Ishwaran H., Kogalur U.B., Blackstone E.H. and Lauer M.S. (2008). Random survival forests, *Ann. App. Statist.*, 2:841-860.

**See Also**

`rsf`.

**Examples**

```
## Not run:
data(pbc, package = "randomSurvivalForest")
imputed.data <- impute.rsf(Surv(days, status) ~ ., data = pbc, nsplit = 3)

## End(Not run)
```

---

max.subtree

*Extract Maximal Subtree Information*


---

### Description

Extract maximal subtree information from a forest. Used for variable selection and identifying interactions between variables.

### Usage

```
max.subtree(object, max.order = 2, sub.order = FALSE, ...)
```

### Arguments

object	An object of class (rsf, grow) or (rsf, forest).
max.order	Non-negative integer specifying the target number of order depths. Default is to return the first and second order depths. Used to identify predictive variables. See details below.
sub.order	Set this value to TRUE to return the minimal depth of each variable relative to another variable. Used to identify interrelationship between variables. See details below.
...	Further arguments passed to or from other methods.

### Details

The maximal subtree for a variable  $x$  is the largest subtree whose root node splits on  $x$ . Thus, all parent nodes of  $x$ 's maximal subtree have nodes that split on variables other than  $x$ . The largest maximal subtree possible is the root node. In general, however, there can be more than one maximal subtree for a variable. A maximal subtree may also not exist if there are no splits on the variable. For details see Ishwaran et al. (2010).

The minimal depth of a maximal subtree measures predictiveness of a variable  $x$ . It equals the shortest distance (the depth) from the root node to the parent node of the maximal subtree (zero is the smallest value possible). The smaller the minimal depth, the more impact  $x$  has on prediction. The second order depth is the shortest distance from the root node to the second node split using  $x$ . To specify the target order depth, use the `max.order` option (e.g., setting `max.order=2` returns the first and second order depths).

Set `sub.order=TRUE` to obtain the minimal depth of a variable relative to another variable. This returns a  $p \times p$  matrix, where  $p$  is the number of variables, and entries  $[i][j]$  are the normalized relative minimal depth of a variable  $[j]$  within the maximal subtree for variable  $[i]$ , where normalization adjusts for the size of  $[i]$ 's maximal subtree. Entry  $[i][i]$  is the normalized minimal depth of  $i$  relative to the root node. The matrix should be read by looking across rows (not down columns) and identifies interrelationship between variables. Small  $[i][j]$  entries indicate interactions. See `find.interaction` for further details.

Applies to competing risk data, but the analysis is non-event specific.

**Value**

A list with the following components:

mean	Minimal depth averaged over a tree and forest for each variable.
order	Order depths for a given variable up to max.order averaged over a tree and the forest. Matrix of dimension p x max.order. If max.order=0, a matrix of p x ntree is returned containing the minimum maximal subtree distance for each variable by tree.
count	Averaged number of maximal subtrees, normalized by the size of a tree, for each variable.
terminal	Average terminal depth of each tree.
nodesAtDepth	Number of nodes per depth per tree. Matrix of dimension maxDepth x ntree.
subOrder	Average minimal depth of a variable relative to another variable. Matrix of dimension p x p. Can be NULL.
threshold	Threshold used to select variables. Variables whose minimal depth exceeds this value are considered to be noise.

**Author(s)**

Hemant Ishwaran <hemant.ishwaran@gmail.com>

Udaya B. Kogalur <kogalurshear@gmail.com>

**References**

Ishwaran H., Kogalur U.B., Gorodeski E.Z, Minn A.J. and Lauer M.S. (2010). High-dimensional variable selection for survival data. *J. Amer. Statist. Assoc.*, 105:205-217.

**See Also**

find.interaction, varSel.

**Examples**

```
## Not run:
# First and second order depths for all variables
data(veteran, package = "randomSurvivalForest")
veteran.out <- rsf(Surv(time, status) ~ . , data = veteran)
v <- max.subtree(veteran.out)

# first and second order depths
print(round(v$order, 3))

# weak variables have minimal depth greater than the following threshold
print(v$threshold)

## End(Not run)
```



Applies to competing risk analyses but plots are non-event specific. Use `competing.risk` for event-specific curves and for a more comprehensive analysis in such cases.

Whenever possible, out-of-bag (OOB) values are used.

### Author(s)

Hemant Ishwaran <hemant.ishwaran@gmail.com>

Udaya B. Kogalur <kogalurshear@gmail.com>

### References

Gerds T.A., Cai T. and Schumacher M. (2008). The performance of risk prediction models, *Biometrical J.*, 4:457-479.

Graf E., Schmoor C., Sauerbrei W. and Schumacher M. (1999). Assessment and comparison of prognostic classification schemes for survival data, *Statist. in Med.*, 18:2529-2545.

### See Also

`rsf`, `predict.rsfc`.

### Examples

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

# plot of ensemble survival for a single individual
surv.ensb <- t(exp(-v.out$oob.ensemble))
plot(v.out$timeInterest, surv.ensb[, 1])
```

---

plot.error

*Plot Error Rate and Variable Importance*

---

### Description

Plot the out-of-bag (OOB) error rates for the ensemble CHF. Also plot variable importance (VIMP) for predictors. This is the default plot method for the package.

### Usage

```
plot.error(x, sorted = TRUE, ...)
plot.rsfc(x, sorted = TRUE, ...)
```

### Arguments

<code>x</code>	An object of class ( <code>rsfc</code> , <code>grow</code> ) or ( <code>rsfc</code> , <code>predict</code> ).
<code>sorted</code>	Should variables be sorted by importance values?
<code>...</code>	Further arguments passed to or from other methods.

## Details

Plot of OOB error rate, with the `bth` 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.

Plots VIMP for the `x`-predictors and prints these values using a matrix with up to 3 columns. The first column is VIMP, the second column is standardized VIMP (divided by the maximum importance value), the third column is `predictorWt` (this is only printed if its values are distinct).

For competing risks, error rates and VIMPs are given for the ensemble CHF and for the ensemble conditional CHF for each event type. For more details see Ishwaran et al. (2010).

## Author(s)

Hemant Ishwaran <hemant.ishwaran@gmail.com>

Udaya B. Kogalur <kogalurshear@gmail.com>

## References

Ishwaran H., Kogalur U.B., Blackstone E.H. and Lauer M.S. (2008). Random survival forests, *Ann. App. Statist.*, 2:841-860.

Ishwaran H., Kogalur U.B., Moore R.D., Gange S.J. and Lau B.M. (2010). Random survival forests for competing risks.

Breiman L. (2001). Random forests, *Machine Learning*, 45:5-32.

Harrell F.E. et al. (1982). Evaluating the yield of medical tests, *J. Amer. Med. Assoc.*, 247, 2543-2546.

## See Also

`rsf`, `predict.rsfc`.

## Examples

```
## Not run:
data(veteran, package = "randomSurvivalForest")
v.out <- rsf(Surv(time, status) ~ ., veteran, ntree = 1000, nsplit = 3)
plot.error(v.out)

## End(Not run)
```

---

plot.proximity      *Extract and Plot the Proximity Matrix*

---

**Description**

Multidimensional scaling plot of the proximity matrix.

**Usage**

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

**Arguments**

x	An object of class (rsf, grow) or (rsf, predict).
plot	Should proximity be plotted.
...	Further arguments passed to or from other methods.

**Details**

Extracts the proximity information from the object x and transforms this to a symmetric proximity matrix. Dissimilarities between points are converted into distances using the 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>

Udaya B. Kogalur <kogalurshear@gmail.com>

**See Also**

rsf, predict.rsfc.

**Examples**

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

---

plot.variable *Plot Survival Effect of Variables*

---

### Description

Plot of ensemble mortality, predicted survival, or predicted survival time against a given x-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, percentile = 50, ...)
```

### 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)?
type	Select type of value to be plotted on the vertical axis. See details.
partial	Should partial plots be created?
predictorNames	Character vector of x-variables to be plotted. Default is all.
npred	Number of variables to be plotted. 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, predictors of the object). Default is to use all rows.
percentile	Percentile of follow up time used for plotting predicted survival. See details below.
...	Further arguments passed to or from other methods.

### Details

Either mortality, relative frequency of mortality, predicted 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 can be specified using predictorNames. The choice of y-value is controlled by type. There are 4 different choices: (1) "mort" is ensemble mortality; (2) "rel.freq" is standardized mortality; (3) "surv" is predicted survival at a given time point (the default is the median follow up time, but this can be set using the option percentile); (4) "time" is the predicted survival time

(this last option only applies to partial plots). For continuous variables, points are colored with blue corresponding to events, and black to censored observations.

Ensemble mortality should be interpreted in terms of total number of deaths. For example, if `i` has a mortality value of 100, then if all individuals were the same as `i`, the expected number of deaths would be 100. If `type="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. Standardized mortality values do not indicate total deaths, but rather relative mortality.

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

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

where  $\hat{f}$  is the predicted value and where for each individual  $i$ ,  $x_{i,o}$  represents the value for all other variables other than  $X$ . For continuous variables, red points are used to indicate partial values and dashed red lines represent an error bar of  $\pm$  two standard errors. A black dashed line indicates 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 provided only as a guide and should be interpreted with caution.

Partial plots can be slow. Setting `type="time"` can improve matters. Setting `npts` to a smaller number should also be tried.

For competing risk analyses, plots correspond to unconditional values (i.e., they are non-event specific). Use `competing.risk` for event-specific curves and for a more comprehensive analysis in such cases.

### Author(s)

Hemant Ishwaran <hemant.ishwaran@gmail.com>

Udaya B. Kogalur <kogalurshear@gmail.com>

### References

Ishwaran H., Kogalur U.B. (2007). Random survival forests for R, *Rnews*, 7(2):25-31.

Ishwaran H., Kogalur U.B., Blackstone E.H. and Lauer M.S. (2008). Random survival forests, *Ann. App. Statist.*, 2:841-860.

Friedman J.H. (2001). Greedy function approximation: a gradient boosting machine, *Ann. of Statist.*, 5:1189-1232.

Liaw A. and Wiener M. (2002). Classification and regression by randomForest, *R News*, 2:18-22.

### See Also

`rsf`, `predict.rsfc`.

## Examples

```
# Some examples applied to veteran data.
data(veteran, package = "randomSurvivalForest")
v.out <- rsf(Surv(time,status) ~ ., veteran, nsplit = 10, 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 = "surv", npred = 1, percentile = 50)
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(Surv(days,status) ~ ., pbc, ntree = 1000, nsplit = 3)
plot.variable(pbc.out, type = "time", partial = TRUE, npred=8)

## End(Not run)
```

---

pmm12rsf

*Restore Random Survival Forest from PMML*

---

## Description

pmm12rsf 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

```
pmm12rsf(pmm1Root, ...)
```

## Arguments

pmm1Root	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 pmm12rsf 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 pmm12rsf 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 rsf2pmm1.

**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>

Udaya B. Kogalur <kogalurshear@gmail.com>

**References**

<http://www.dmg.org>

**See Also**

xmlTreeParse, xmlRoot, saveXML, rsf2pmm1.

**Examples**

```
## Not run:
# 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.

library("XML")

data(veteran, package = "randomSurvivalForest")
veteran.out <- rsf(Surv(time, status)~., data = veteran, ntree = 5)
veteran.forest <- veteran.out$forest
veteran.pmm1 <- rsf2pmm1(veteran.forest)

# Save the document to disk.
userFile = file("veteran.forest.xml")
saveXML(veteran.pmm1, userFile)
close(userFile)

# Read the just written document.
veteran.pmm1 <- 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],
  outcome = c("train", "test")[1],
  proximity = FALSE, split.depth = FALSE, seed = NULL,
  do.trace = FALSE, ...)

```

## Arguments

object	An object of class (rsf, grow) or (rsf, forest).
test	Data frame containing test data. Missing values allowed.

importance	Method used to compute variable importance (VIMP). Only applies when test data contains survival outcomes.
na.action	Action to be taken if the data contains NA's. Possible values are "na.omit", which removes the entire record if even one of its entries is NA, and "na.impute", which imputes the test data. See details below.
outcome	Data frame used in calculating the ensemble. By default this is always the training data, but see details below.
proximity	Should proximity measure between test observations be calculated? Can be large.
split.depth	Return minimal depth for each variable for each test set individual?
seed	Seed (negative integer) for random number generator.
do.trace	Logical. Should trace output be enabled? 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

The test data is dropped down the grow-forest (i.e., the forest grown from the training data) yielding the ensemble cumulative hazard function (CHF) for each individual in the test data evaluated at each unique death time point from the grow data. If survival outcome information is present in the test data, the overall error rate and VIMP for each variable is also returned. Setting `na.action="na.impute"` imputes missing test data (x-variables or outcomes). Imputation uses the grow-forest such that only training data is used when imputing test data to avoid biasing error rates and VIMP (Ishwaran et al. 2008).

For competing risks, the ensemble conditional CHF (CCHF) is computed for each event type in addition to the ensemble CHF.

If `outcome="test"`, the ensemble is calculated by specifically using survival information from the test data (survival information must be present). In this case, the terminal nodes from the grow-forest are recalculated using survival data from the test set. This yields a modified predictor in which the topology of the forest is based solely on the training data, but where the predicted value is based on the test data. Error rates and VIMP are calculated by bootstrapping the test data and using out-of-bagging to ensure unbiased estimates. See Examples 2 and 3 below for illustration.

### Value

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

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

ndead	Number of deaths in test data (can be NULL).
time	Vector of survival times from test data (can be NULL).
cens	Vector of censoring indicators from test data (can be NULL).
predictorNames	Character vector of variable names.
predictors	Data frame comprising x-variables used for prediction.
ensemble	Matrix containing the ensemble CHF for the test data. Each row corresponds to the CHF for an individual in the test set evaluated at each of the time points in timeInterest. For competing risks, a 3-D array where the 3rd dimension is for the ensemble CHF and each of the CCHFs, respectively.
poe	Matrix containing the ensemble probability of an event (POE) for each test set individual: used to estimate the CIF. Rows correspond to each of the event types. Applies only to competing risk data. NULL otherwise.
mortality	Vector containing ensemble mortality for each individual in the test data. Ensemble mortality should be interpreted in terms of total number of training deaths if outcome="train".
err.rate	Vector of length ntree of the test-set error rate. For competing risks, a matrix of test-set errors with rows corresponding to the ensemble CHF and each of the CCHFs, respectively. Can be NULL. If outcome="test" the test-set error is non-cumulative (i.e., it is for the full forest).
importance	VIMP of each variable in the test data. For competing risks, a matrix with rows corresponding to the ensemble CHF and each of the CCHFs, respectively. Can be NULL.
proximity	If proximity=TRUE, 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 plot.proximity to extract this information.
imputedIndv	Vector of indices of records in test data with missing values. Can be NULL.
imputedData	Data frame containing the imputed test data. First two columns are censoring and survival time, respectively. The remaining columns are the x-variables. Row i contains imputed outcomes and x-variables for row imputedIndv[i] of predictors. Can be NULL.
splitDepth	Matrix where entry [i][j] is the mean minimal depth for variable [j] for case [i] in the test data. Used for variable selection (see max.subtree). Can be NULL.

### Author(s)

Hemant Ishwaran <hemant.ishwaran@gmail.com>

Udaya B. Kogalur <kogalurshear@gmail.com>

### References

- Breiman L. (2001). Random forests, *Machine Learning*, 45:5-32.
- Ishwaran H., Kogalur U.B. (2007). Random survival forests for R, *Rnews*, 7(2):25-31.
- Ishwaran H., Kogalur U.B., Blackstone E.H. and Lauer M.S. (2008). Random survival forests, *Ann. App. Statist.*, 2:841-860.
- Ishwaran H., Kogalur U.B., Moore R.D., Gange S.J. and Lau B.M. (2010). Random survival forests for competing risks.

**See Also**

rsf.

**Examples**

```

#-----
# Example 1: Typical call (veteran data)

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

## Not run:
#-----
# Example 2: Get out-of-bag error rate using the training
# data as test data (pbc example)

data(pbc, package = "randomSurvivalForest")
pbc.grow <- rsf(Surv(days, status) ~ ., pbc, nsplit = 3)
pbc.pred <- predict(pbc.grow, pbc, outcome = "test")
cat("GROW error rate :", round(pbc.grow$err.rate[1000], 3))
cat("PRED error rate :", round(pbc.pred$err.rate, 3))

#-----
# Example 3: Verify reproducibility of forest (pbc data)

#primary call
data(pbc, package = "randomSurvivalForest")
pt.train <- sample(1:nrow(pbc), round(nrow(pbc)*0.50))
pbc.out <- rsf(Surv(days, status) ~ ., nsplit = 3,
              data = pbc[pt.train, ])

#make separate predict calls using the outcome option
pbc.train <- predict(pbc.out, pbc[-pt.train, ], outcome = "train")
pbc.test <- predict(pbc.out, pbc[-pt.train, ], outcome = "test")

#check forest reproducibility by comparing predicted survival curves
timeInterest <- pbc.out$timeInterest
surv.train <- exp(-pbc.train$ensemble)
surv.test <- exp(-pbc.test$ensemble)
matplot(timeInterest, t(surv.train - surv.test), type = "l")

#test reproducibility by repeating B times
#compute l1-difference in predicted survival
B <- 25
l1.valid <- rep(NA, B)
for (b in 1:B) {
  cat("Replication:", b, "\n")
  pt.train <- sample(1:nrow(pbc), round(nrow(pbc)*0.50))
  pbc.out <- rsf(Surv(days, status) ~ ., nsplit = 3,
                data = pbc[pt.train, ])

```

```

surv.train <- exp(-predict(pbc.out, pbc[-pt.train, ],
  outcome = "train")$ensemble)
surv.test <- exp(-predict(pbc.out, pbc[-pt.train, ],
  outcome = "test")$ensemble)
l1.valid <-
  mean(apply(abs(surv.train - surv.test), 1, mean, na.rm = TRUE), na.rm = TRUE)
}
cat("l1-reproducibility:", round(mean(l1.valid, na.rm = TRUE), 3), "\n")

## End(Not run)

```

---

print.rsf

---

*Print Summary Output of Analysis*


---

### Description

Print summary output from a 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>  
 Udaya B. Kogalur <kogalurshear@gmail.com>

### See Also

rsf, predict.rsf.

### Examples

```

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

```

## Description

Prediction and variable selection for right censored survival and competing risk data using Random Survival Forests (RSF) (Ishwaran, Kogalur, Blackstone and Lauer, 2008). A random forest (Breiman, 2001) of survival trees is used for ensemble estimation of the cumulative hazard function (CHF) in right-censored settings and the conditional cumulative hazard function (CCHF) in the case of competing risks. Different survival tree splitting rules can be used to grow trees. An “out-of-bag” estimate of Harrell’s concordance index (Harrell, 1982) is provided for assessing prediction accuracy. Variable importance (VIMP) for single, as well as grouped variables, can be used to filter variables and to assess variable predictiveness. Minimal depth variable selection is also available. Missing data (x-variables, survival times, censoring indicators) can be imputed on both training and test data.

## Usage

```
rsf(formula, data = NULL, ntree = 1000, mtry = NULL,
    nodesize = NULL, splitrule = NULL, nsplit = 0,
    importance = c("randomsplit", "permute", "none")[1],
    big.data = FALSE, na.action = c("na.omit", "na.impute")[1],
    nimpute = 1, predictorWt = NULL, forest = TRUE,
    proximity = FALSE, varUsed = NULL, split.depth = FALSE,
    seed = NULL, do.trace = FALSE, ...)
```

## Arguments

formula	A symbolic description of the model to be fit.
data	Data frame containing the data used in the formula. Missing values allowed. See na.action for details.
ntree	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.
mtry	Number of variables randomly sampled at each split. The default is $\sqrt{p}$ , where p equals the number of variables.
nodesize	Minimum number of deaths with unique survival times required for a terminal node. Default is approximately 3 for right-censoring and 6 for competing risk data. Larger values create smaller trees.
splitrule	Splitting rule used to grow trees. See details below.
nsplit	Non-negative integer value. If non-zero, the specified tree splitting rule is randomized which can significantly increase speed. See details below.
importance	Method used to compute variable importance. See details below.
big.data	Set this value to TRUE when the number of variables p is very large, or the sample size is very large. See details below.

<code>na.action</code>	Action to be taken if the data contains NA's. Possible values are "na.omit" and "na.impute". Default is "na.omit", which removes the entire record if even one of its entries is NA (for x-variables this applies only to those specifically listed in <code>formula</code> ). The action "na.impute" implements a sophisticated tree imputation technique. See details below.
<code>nimpute</code>	Number of iterations of the 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>	Should the forest object be returned? Used for prediction on new data and required for many of the wrappers to work.
<code>proximity</code>	Should the proximity between observations be calculated? Creates an $n \times n$ matrix (which can be huge). Default is FALSE.
<code>varUsed</code>	Analyzes which variables are used (split upon) in the forest. Default is NULL. Possible values are "all.trees" and "by.tree". See details below.
<code>split.depth</code>	Return minimal depth for each variable for each case. Default is FALSE. Used for variable selection: 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>	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

—> Splitting Rules:

Four primary splitting rules are available for growing a survival forest for right-censored data: "logrank", "conserve", "logrankscore", and "random".

The default rule, "logrank", splits tree nodes by maximization of the log-rank test statistic (Segal, 1988; Leblanc and Crowley, 1993). The "conserve" rule splits nodes by finding daughters closest to the conservation of events principle (see Naftel, Blackstone and Turner, 1985). The "logrankscore" rule uses a standardized log-rank statistic (Hothorn and Lausen, 2003). The "random" rule 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 (Cutler and Zhao, 2001; Lin and Jeon, 2006). Note, however, that because random splitting promotes splits near the edges, node splitting can terminate early resulting in extremely unbalanced trees. To correct this, the definition of `nodesize` is taken in this setting (and this setting alone) to equal the minimum number of unique deaths within a node required to split the node.

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, i.e. `nsplit=0`, 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).

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 further improved without overly compromising speed by using larger values of `nsplit`.

Trees tend to favor splits on continuous variables (Loh and Shih, 1997), so it is good practice to use the `nsplit` option when the data contains a mix of continuous and discrete variables. Using a reasonably small value mitigates bias.

—> Large Data Sets:

Computation times for *very* large data sets can be improved by discretizing continuous variables and/or the observed survival times; in addition to using random splitting. 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* under this option. Think of the data frame as containing time and censoring information and the rest of the data as the pre-processed design matrix. In particular, transformations used in the formula (such as `logs` etc.) are ignored.

—> Formula:

A typical RSF formula has the form `Surv(time, censoring) ~ terms`, where "time" is survival time and "censoring" is a binary censoring indicator. Censoring must be coded as a non-negative integer with 0 reserved for censoring and (usually) 1=death (event). Also, "time" must be strictly positive.

—> Factors and Variable Types:

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 to avoid issues.

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

—> Variable Importance:

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. Out-of-bag (OOB) cases are dropped down the in-bag (bootstrap) 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:

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) two random 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).

—> Competing Risks:

The implementation is similar to right-censoring but with the following caveats:

- (1) Censoring must be coded as a non-negative integer where 0 indicates right-censoring and non-zero values indicate different event types. While  $0, 1, 2, \dots, J$  is standard, events can be coded non-sequentially, although 0 must always be used for censoring.
- (2) The default splitting rule is "logrankCR", a modified log-rank splitting rule tailored for competing risks. Over-riding this by manually selecting any split rule other than "logrankCR" or "random" will result in a right-censored analysis in which all (non-censored) events are treated as if they are one event type (indeed, they will be coerced as such). Note that `nsplit` works as in right-censoring.
- (3) The ensemble (see below) is a 3-D array in which the 3rd dimension is reserved for the ensemble CHF and each of the  $J$  ensemble conditional CHFs (CCHFs). The wrapper `competing.risk` can be used to process the ensemble and to generate event-specific cumulative incidence functions (CIF) and subsurvival functions (see Gray (1988) for background and definitions).
- (4) The cases within a terminal node are used to estimate both the unconditional survival function and the event-specific subsurvival functions and for this reason `nodesize` should generally be set larger than in right-censored data settings.

—> Missing Data and Imputation:

Setting `na.action="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 the returned objects `predictors`, `time` and `cens` are replaced by imputed values. Further, imputed objects such as `imputedData` are set to NULL. See the examples below. Also see the wrapper `impute.rsfc` for a fast impute interface.

—> Miscellanea:

Setting `varUsed="all.trees"` returns a vector where each element is a count of the number of times a split occurred on a variable. If `varUsed="by.tree"`, a matrix of size `n tree x p` is returned. Each element `[i][j]` is the count of the number of times a split occurred on variable `[j]` in tree `[i]`.

Setting `split.depth=TRUE` returns a matrix of size `n x p` where entry `[i][j]` is the mean minimal depth for variable `[j]` for case `[i]`. Used to select variables at the case-level. See `max.subtree` for more details regarding minimal depth.

## Value

An object of class `(rsf, grow)` 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).
<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>	Matrix for the in-bag ensemble CHF with each row corresponding to an individual's CHF evaluated at each of the time points in <code>timeInterest</code> . For competing risks, a 3-D array where the 3rd dimension is for the ensemble CHF and each of the CCHFs, respectively.
<code>oob.ensemble</code>	Same as <code>ensemble</code> , but based on OOB data.
<code>poe</code>	Matrix for the in-bag ensemble probability of an event ( <code>poe</code> ) for each individual: used to estimate the CIF. Rows correspond to each of the event types. Applies only to competing risk data. NULL otherwise.
<code>oob.poe</code>	Same as <code>poe</code> , but based on OOB data.
<code>mortality</code>	A vector of length <code>n</code> for the in-bag 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. For competing risks, a matrix with rows corresponding to the ensemble CHF and each of the CCHFs, respectively.
<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>	Vector recording VIMP for each variable. For competing risks, a matrix with rows corresponding to the ensemble CHF and each of the CCHFs, respectively.
<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 and is required for other R-wrappers.

proximity	If proximity=TRUE, a matrix of dimension $n \times n$ 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.
varUsed	Count of the number of times a variable is used in growing the forest. Can be a vector, matrix, or NULL.
imputedIndv	Vector of indices for cases with missing values. Can be NULL.
imputedData	Data frame comprising imputed data. First two columns are censoring and survival time, respectively. Remaining columns are the x-variables. Row $i$ contains imputed outcomes and x-variables for row $j$ of predictors, where $j = \text{imputedIndv}[i]$ . See the examples below. Can be NULL.
splitDepth	Matrix of size $n \times p$ where entry $[i][j]$ is the mean minimal depth for variable $[j]$ for case $[i]$ . Can be NULL.

### Note

The key deliverable is the matrix ensemble (and its OOB counterpart, `oob.ensemble`) containing the ensemble CHF function for each individual evaluated at a set of distinct time points. 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 to aid in parsing the ensemble.

### Author(s)

Hemant Ishwaran <hemant.ishwaran@gmail.com>

Udaya B. Kogalur <kogalurshear@gmail.com>

### References

- Breiman L. (2001). Random forests, *Machine Learning*, 45:5-32.
- Cutler A. and Zhao G. (2001). Pert-Perfect random tree ensembles. *Comp. Sci. Statist.*, 33: 490-497.
- Gray R.J. (1988). A class of k-sample tests for comparing the cumulative incidence of a competing risk, *Ann. Statist.*, 16:1141-1154.
- Harrell F.E. et al. (1982). Evaluating the yield of medical tests, *J. Amer. Med. Assoc.*, 247:2543-2546.
- Hothorn T. and Lausen B. (2003). On the exact distribution of maximally selected rank statistics, *Comp. Statist. Data Anal.*, 43:121-137.
- Ishwaran H. (2007). Variable importance in binary regression trees and forests, *Electronic J. Statist.*, 1:519-537.
- Ishwaran H., Kogalur U.B. (2007). Random survival forests for R, *Rnews*, 7(2):25-31.

- Ishwaran H., Kogalur U.B., Blackstone E.H. and Lauer M.S. (2008). Random survival forests, *Ann. App. Statist.*, 2:841-860.
- Ishwaran H., Kogalur U.B., Gorodeski E.Z, Minn A.J. and Lauer M.S. (2010). High-dimensional variable selection for survival data. *J. Amer. Statist. Assoc.*, 105:205-217.
- Ishwaran H., Kogalur U.B., Chen X. and Minn A.J. (2010). Random survival forests for high-dimensional data.
- Ishwaran H., Kogalur U.B., Moore R.D., Gange S.J. and Lau B.M. (2010). Random survival forests for competing risks.
- LeBlanc M. and Crowley J. (1993). Survival trees by goodness of split, *J. Amer. Statist. Assoc.*, 88:457-467.
- Liaw A. and Wiener M. (2002). Classification and regression by randomForest, *R News*, 2:18-22.
- Lin Y. and Jeon Y. (2006). Random forests and adaptive nearest neighbors, *J. Amer. Statist. Assoc.*, 101:578-590.
- Loh W.-Y and Shih Y.-S (1997). Split selection methods for classification trees, *Statist. Sinica*, 7:815-840, 1997.
- Naftel N.C., Blackstone E.H. and Turner M.E. (1985). Conservation of events, unpublished notes.
- Segal M.R. (1988). Regression trees for censored data, *Biometrics*, 44:35-47.

### See Also

competing.risk, find.interaction, impute.rsfc, max.subtree, plot.ensemble, plot.variable, plot.error, plot.proximity, pmml2rsf, predict.rsfc, print.rsfc, rsf2rfz, rsf2pmml, varSel, vimp.

### Examples

```
#-----
# Example 1: Veteran's Administration lung cancer data
# Randomized trial of two treatment regimens for lung cancer
# See Kalbfleisch & Prentice

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

#-----
# Example 2: More detailed call (veteran data)

#read the data, set various options
data(veteran, package = "randomSurvivalForest")
veteran.f <- as.formula(Surv(time, status) ~ .)
ntree <- 200
nsplit <- 3
varUsed <- "by.tree"

# coerce 'celltype' as a factor and 'karnofsky score'
# as an ordered factor to illustrate factor usage
```

```

veteran$celltype <- factor(veteran$celltype,
  labels=c("squamous", "smallcell", "adeno", "large"))
veteran$karno <- factor(veteran$karno, ordered = TRUE)

# grow call
veteran2.out <- rsf(veteran.f, veteran,
  ntree = ntree, nsplit = nsplit, varUsed = varUsed)

# plot of ensemble survival for a single individual
surv.ensb <- t(exp(-veteran2.out$soob.ensemble))
plot(veteran2.out$timeInterest, surv.ensb[, 1])

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

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

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

## Not run:
#-----
# Example 3: Competing risks

# Follicular Cell Lymphoma
data(follic, package = "randomSurvivalForest")
follic.out <- rsf(Surv(time, status) ~ ., follic, nsplit = 3, ntree = 400)
print(follic.out)
plot(follic.out, sorted = FALSE)
competing.risk(follic.out)

# Hodgkin's disease
data(hd, package = "randomSurvivalForest")
hd.out <- rsf(Surv(time, status) ~ ., hd, nsplit = 3, ntree = 400)
print(hd.out)
plot(hd.out, sorted = FALSE)
competing.risk(hd.out)

#-----
# Example 4: Primary biliary cirrhosis (PBC) of the liver
# See Appendix D.1 of Fleming and Harrington

data(pbc, package = "randomSurvivalForest")
pbc.out <- rsf(Surv(days, status) ~ ., pbc, nsplit = 3)
print(pbc.out)

#-----
# Example 5: Same as Example 4, but with data imputation
# Also see the R-wrapper "impute.rsfc"

# rsf call with imputation

```

```

data(pbc, package = "randomSurvivalForest")
pbc2.out <- rsf(Surv(days, status)~., pbc,
               nsplit = 3, na.action="na.impute")
print(pbc2.out)

# here's a nice wrapper to combine original data + imputed data
combine.impute <- function(object) {
  imputed.data <- cbind(cens = object$cens,
                       time = object$time,
                       object$predictors)
  if (!is.null(object$imputedIndv)) {
    imputed.data[object$imputedIndv, ] <- object$imputedData
  }
  colnames(imputed.data)[c(2,1)] <- all.vars(object$formula)[1:2]
  imputed.data
}

# combine original data + imputed data
pbc.imputed.data <- combine.impute(pbc2.out)

# iterate the missing data algorithm
# compare to non-iterated algorithm
pbc3.out <- rsf(Surv(days, status)~., pbc, nsplit=5,
               na.action="na.impute", nimpute = 3)
pbc.iterate.imputed.data <- combine.impute(pbc3.out)
tail(pbc.imputed.data)
tail(pbc.iterate.imputed.data)

#-----
# Example 6: German breast cancer data
# Variable selection using minimal depth

data(breast, package = "randomSurvivalForest")
breast.out <- rsf(Surv(time, cens) ~ . , breast, nsplit = 3)

# use varSel to select variables
# see the help file of varSel for details/examples

breast.vs <- varSel(object=breast.out)

#-----
# Example 7: Compare Cox regression to RSF using PBC data
# OOB estimate of C-index for Cox based on 100 bootstraps
# Assumes "Hmisc" and "survival" libraries are loaded

if (library("survival", logical.return = TRUE)
    & library("Hmisc", logical.return = TRUE))
{
  data(pbc, package = "randomSurvivalForest")
  rsf.f <- as.formula(Surv(days, status) ~ .)
  pbc3.out <- rsf(rsf.f, pbc, nsplit = 10, mtry = 2)
  B <- 100
}

```

```

cox.err <- rep(NA, B)
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 <- tryCatch({coxph(rsf.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>

Udaya B. Kogalur <kogalurshear@gmail.com>

---

`rsf2pmm1`*Save Random Survival Forest as PMML*

---

### Description

`rsf2pmm1` 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

```
rsf2pmm1(object, ...)
```

### Arguments

<code>object</code>	An object of class ( <code>rsf</code> , <code>grow</code> ) or ( <code>rsf</code> , <code>forest</code> ).
<code>...</code>	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 `rsf2pmm1` 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 `rsf2pmm1` 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 `rsf2pmm1` together with `pmm12rsf`.

### 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>

Udaya B. Kogalur <kogalurshear@gmail.com>

**References**

<http://www.dmg.org>

**See Also**

xmlTreeParse, xmlRoot, saveXML, pmml2rsf.

**Examples**

```
## Not run:
# 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.

library("XML")

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

# Save the document to disk.
userFile = file("veteran.forest.xml")
saveXML(veteran.pmml, userFile)
close(userFile)

# Read the just written document.
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)

```

---

rsf2rfz

*Save Random Survival Forest as a .rfz Compressed File*


---

## Description

rsf2rfz saves a **randomSurvivalForest** forest object as an .rfz compressed file that is readable by the RSF Java plugin that is capable of visualizing the trees in the forest.

## Usage

```
rsf2rfz(object, forestName = NULL, ...)
```

## Arguments

object	An object of class (rsf, grow) or (rsf, forest).
forestName	The desired prefix name for forest as a string.
...	Further arguments passed to or from other methods.

## Details

An .rfz compressed file is actually a .zip file consisting of three files. The first is an ASCII file of type .txt containing the \$nativeArray component of the forest. The second is an ASCII file of type .factor.txt containing the \$nativefactorArray component of the forest. The third is an ASCII file of type .xml containing the PMML DataDictionary component.

PMML or 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>.

This function, rsf2rfz, is used to import the geometry of the forest to a Java plugin that is capable of visualizing the trees in the forest.

The geometry of the forest is saved as a file called forestName.rfz in the users working directory. This file can then be read by the **randomSurvivalForest** Java plugin.

Contact the authors on downloading the Java plugin.

## Value

None.

**Note**

Contact the authors on downloading the Java plugin.

**Author(s)**

Hemant Ishwaran <hemant.ishwaran@gmail.com>

Udaya B. Kogalur <kogalurshear@gmail.com>

**References**

<http://www.dmg.org>

**See Also**

rsf.

**Examples**

```
## Not run:
# Example 1: Growing a forest, saving it as a \emph{.rfz} file ready
# for import into the Java plugin.
```

```
library("XML")
```

```
data(veteran, package = "randomSurvivalForest")
veteran.out <- rsf(Surv(time, status)~., data = veteran, ntree = 5)
veteran.forest <- veteran.out$forest
rsf2rfz(veteran.forest, forestName="veteran")
```

```
## End(Not run)
```

---

varSel

*Variable Selection for Random Survival Forests*

---

**Description**

Variable selection for random survival forests using minimal depth (Ishwaran et al. 2010).

**Usage**

```
varSel(formula = NULL, data = NULL, object = NULL,
       method = c("vh", "vhVIMP", "md")[3],
       ntree=(if (method == "md") 1000 else 500),
       mvars = (if (!is.null(data) & method!="md")
                min(1000,round(ncol(data)/5)) else NULL),
       mtry = (if (!is.null(data) & method == "md")
                max(sqrt(ncol(data)),ncol(data)/3) else NULL),
```

```

nodesize = (if (method == "vh" | method == "vhVIMP")
              2 else NULL),
nsplit = 10, predictorWt = NULL, big.data = FALSE,
na.action = c("na.omit", "na.impute")[1],
do.trace = 0, always.use = NULL, nrep = 50, K = 5,
nstep = 1, verbose = TRUE, ... )

```

### Arguments

formula	A symbolic description of the model to be fit. Must be specified unless object is given.
data	Data frame containing the data used in the formula. Missing values allowed. Must be specified unless object is given.
object	An object of class (rsf, grow).
method	Variable selection method: "vh"=variable hunting; "vhVIMP" =variable hunting with VIMP; "md"=minimal depth. See details below.
nree	Number of trees to grow.
mvars	Number of randomly selected variables used in the variable hunting algorithm.
mtry	Number of variables randomly sampled at each split. Should be large when the goal is variable selection.
nodesize	Minimum number of deaths with unique survival times required for a terminal node. Should be small if number of variables is large.
nsplit	Non-negative integer value. If non-zero, the specified tree splitting rule is randomized which significantly increases speed.
predictorWt	Vector of non-negative weights specifying the probability of selecting a variable for splitting. Must be of dimension equal to the number of variables. Default (NULL) invokes a data-adaptive method.
big.data	Only set this value to TRUE when the sample size is very large.
na.action	Action to be taken if the data contains NA's.
do.trace	Should trace output be enabled? Default is FALSE. A positive integer value causes output to be printed each do.trace iteration.
always.use	Character vector of variable names to be always included in the model selection procedure and in the final selected model.
nrep	Number of Monte Carlo iterations of the variable hunting algorithm.
K	Integer value specifying the K-fold size used in the variable hunting algorithm.
nstep	Integer value controlling the step size used in the forward selection process of the variable hunting algorithm. Increasing this will encourage more variables to be selected.
verbose	Set to TRUE to get verbose output.
...	Further arguments passed to or from other methods.

## Details

Variable selection using minimal depth. The option method allows for two different approaches: (1) minimal depth: uses all data and all variables simultaneously; and (2) variable hunting: uses K-fold Monte Carlo validation, random selection of variables, and regularized forward selection.

The following is a brief description of the two methods. For complete details, one should see Ishwaran et al. (2010).

—> Minimal Depth variable selection (method="md")

The maximal subtree for a variable  $x$  is the largest subtree whose root node splits on  $x$  (all parent nodes of  $x$ 's maximal subtree have nodes that split on variables other than  $x$ ). The minimal depth of a maximal subtree equals the shortest distance (the depth) from the root node to the parent node of the maximal subtree (zero is the smallest value possible). The smaller the minimal depth, the more impact  $x$  has on prediction.

Variables are selected using an adaptive threshold based on minimal depth coupled with minor supervision using VIMP (variable importance).

Set `mtry` to larger values when the number of variables is high.

—> Variable Hunting (method="vh" or method="vhVIMP")

Variable hunting is used for problems where the number of variables is magnitudes larger than the sample size and the sample size is reasonably small. Microarray data is a good example.

Using training data from random K-fold subsampling, a forest is fit to a randomly selected set of variables of size `mvars` where variables are chosen with probability proportional to weights determined using an initial forest fit on the training data. The subset of variables are ordered by increasing minimal depth and added sequentially (starting from a minimal model) until joint VIMP no longer increases (signifying the final model). A forest is refit with these variables and applied to test data to estimate prediction error and VIMP. The process is repeated `nrep` times. Final selected variables are the top `P` ranked variables, where `P` is the average model size and variables are ranked by average minimal depth.

A rough rule for choosing `mvars` is to set it equal to some fraction of the number of variables.

The same algorithm is used when `method="vhVIMP"`, but variables are ordered using VIMP (including the final model). This is faster, but not as accurate.

If `method="vh"`, and the number of variables is large, set `nsplit` to a fairly large number, such as 10, to ensure that tree splitting is not overly influenced by noisy variables.

—> Miscellanea

If `big.data=TRUE`, and variable hunting is used, the training data is chosen to be of size `n/K`, where `n`=sample size (i.e., the size of the training data is swapped with the test data). This speeds up the algorithm. Increasing `K` also helps.

For efficiency, transformations used in the formula (such as logs etc.) are ignored. Variables are interpreted *as is*.

Can be used for competing risk data. Variable selection is based on the ensemble CHF.

## Value

A list with the following components:

`err.rate` Prediction error for the forest (a vector of length `nrep` if variable hunting used).

modelSize	Number of variables selected.
topvars	Character vector of names of the final selected variables.
varselect	Matrix of values used in determining the set of selected variables.
rsf.out	Refitted forest using the final set of selected variables. NULL if big.data=TRUE.

### Author(s)

Hemant Ishwaran <hemant.ishwaran@gmail.com>

Udaya B. Kogalur <kogalurshear@gmail.com>

### References

Ishwaran H., Kogalur U.B., Gorodeski E.Z, Minn A.J. and Lauer M.S. (2010). High-dimensional variable selection for survival data. *J. Amer. Statist. Assoc.*, 105:205-217.

Ishwaran H., Kogalur U.B., Chen X. and Minn A.J. (2010). Random survival forests for high-dimensional data.

### See Also

max.subtree, rsf.

### Examples

```
## Not run:
#-----
# Minimal depth variable selection: pbc data with noise

data(pbc, package = "randomSurvivalForest")
vs <- varSel(Surv(days, status) ~ ., pbc)

# As dimension increases, mtry should increase
pbc.noise <- cbind(pbc, noise = matrix(rnorm(nrow(pbc) * 1000), nrow(pbc)))
vs.bigp <- varSel(Surv(days, status) ~ ., pbc.noise, mtry = 100)

#-----
# Variable hunting: van de Vijver microarray breast cancer
# Note: nrep is small for illustration; typical values are nrep = 100

data(vdv, package = "randomSurvivalForest")
vh <- varSel(Surv(Time, Censoring) ~ ., vdv, method = "vh",
            nrep = 10, nstep = 5)

# Same analysis, but using predefined weights for selecting a gene
# for node splitting. We illustrate this using univariate cox p-values.

if (library("survival", logical.return = TRUE)
    & library("Hmisc", logical.return = TRUE))
{
  cox.weights <- function(rsf.f, rsf.data) {
```

```

event.names <- all.vars(rsf.f)[1:2]
p <- ncol(rsf.data) - 2
event.pt <- match(event.names, names(rsf.data))
predictor.pt <- setdiff(1:ncol(rsf.data), event.pt)
sapply(1:p, function(j) {
  cox.out <- coxph(rsf.f, rsf.data[, c(event.pt, predictor.pt[j])])
  pvalue <- summary(cox.out)$coef[5]
  if (is.na(pvalue)) 1.0 else 1/(pvalue + 1e-100)
})
}

data(vdv, package = "randomSurvivalForest")
rsf.f <- as.formula(Surv(Time, Censoring) ~ .)
cox.wts <- cox.weights(rsf.f, vdv)
vh.cox <- varSel(rsf.f, vdv, method = "vh", nstep = 5, predictorWt = cox.wts)

}

## End(Not run)

```

---

vdv

*van de Vijver Microarray Breast Cancer Data*


---

### Description

Gene expression profiling for predicting clinical outcome of breast cancer (van't Veer et al., 2002). Microarray breast cancer data set of 4707 expression values on 78 patients with survival information.

### References

van't Veer L.J. et al. (2002). Gene expression profiling predicts clinical outcome of breast cancer. *Nature*, 12:530-536.

---

veteran

*Veteran's Administration Lung Cancer Trial Data*


---

### Description

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

### References

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

**Description**

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

**Usage**

```
vimp(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).
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, predictors). Default is to use all rows.
joint	Should joint-VIMP or individual VIMP be calculated? See details below.
rough	Should fast approximation be used?
importance	Type of VIMP.
seed	Seed (negative integer) for random number generator.
do.trace	Should trace output be enabled? 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, and restricting the data to that indicated by `subset`, calculate the VIMP for variables listed in `predictorNames`. If `joint=TRUE`, then joint-VIMP for the group of variables is calculated. This equals the amount that prediction error changes when the group of variables are simultaneously "noised-up" (perturbed). If `joint=FALSE`, the VIMP for each variable is calculated separately.

Depending upon the setting for `importance`, VIMP is determined by using either random daughter assignment ("randomsplit") or random permutation ("permute") noising-up of the variable(s). A NULL value is returned if `importance="none"`.

For competing risk data, VIMP and error rates are given for the ensemble CHF and the conditional CHF (CCHF) for each event type. For more details see Ishwaran et al. (2010).

**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. Dimension depends upon the option <code>joint</code> .
<code>importance</code>	Variable importance (VIMP). Dimension depends upon the option <code>joint</code> .

**Author(s)**

Hemant Ishwaran <hemant.ishwaran@gmail.com>

Udaya B. Kogalur <kogalurshear@gmail.com>

**References**

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

Ishwaran H., Kogalur U.B., Moore R.D., Gange S.J. and Lau B.M. (2010). Random survival forests for competing risks.

**See Also**

`find.interaction`.

**Examples**

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

data(veteran, package = "randomSurvivalForest")
v.out <- rsf(Survrsf(time,status) ~ ., veteran, ntree = 1000)
vimp(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(Surv(days,status) ~ ., pbc, ntree = 1000, nsplit = 3)
o.r <- rev(order(rsf.out$importance))
imp <- rsf.out$importance[o.r]
imp.events <- rep(0, length(imp))
events <- which(rsf.out$cens == 1)
imp.events <-
  vimp(rsf.out, names(imp), events, joint = FALSE)$importance
imp.all <- as.data.frame(cbind(imp.events = imp.events, imp = imp))
print(round(imp.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 <- rsf(Surv(days,status) ~ ., pbc, ntree = 1000, nsplit = 3)
o.r <- rev(order(rsf.out$importance))
imp.names <- names(rsf.out$importance[o.r])
subset.index <- 1:nrow(rsf.out$predictors)
imp.mc <- imp.boot <- NULL
for (k in 1:100) {
  cat("iteration:", k , "\n")
  imp.mc <-
    cbind(imp.mc, vimp(rsf.out, imp.names, joint = FALSE)$importance)
  imp.boot <-
    cbind(imp.boot, vimp(rsf.out, imp.names,
      subset = sample(subset.index, replace = TRUE), joint = FALSE)$importance)
}
imp.mc <- as.data.frame(cbind(imp.mean = apply(imp.mc, 1, mean),
  imp.sd = apply(imp.mc, 1, sd)))
imp.boot <- as.data.frame(cbind(imp.mean = apply(imp.boot, 1, mean),
  imp.sd = apply(imp.boot, 1, sd)))
print(round(imp.mc, 3))
print(round(imp.boot, 3))

## End(Not run)

```

---

wihs

*Women's Interagency HIV Study (WIHS)*


---

### Description

Competing risk data set involving AIDS in women.

### Format

A data frame containing:

time	time to event
status	censoring status: 0=censoring, 1=HAART initiation, 2=AIDS/Death before HAART
ageatfda	age in years at time of FDA approval of first protease inhibitor
idu	history of IDU: 0=no history, 1=history
black	race: 0=not African-American; 1=African-American
cd4nadir	CD4 count (per 100 cells/ul)

## Source

Study included 1164 women enrolled in WIHS, who were alive, infected with HIV, and free of clinical AIDS on December, 1995, when the first protease inhibitor (saquinavir mesylate) was approved by the Federal Drug Administration. Women were followed until the first of the following occurred: treatment initiation, AIDS diagnosis, death, or administrative censoring (September, 2006). Variables included history of injection drug use at WIHS enrollment, whether an individual was African American, age, and CD4 nadir prior to baseline.

## References

- Lau B.M, Cole S.R. and Gange S.J. (2009). Competing risk regression models for epidemiologic data, *Amer. J. Epidemiol.*, 170(2):244-56.
- Barkan S.E., Melnick S.L., Preston-Martin S., Weber K., Kalish L.A., Miotti P., Young M., Greenblatt R., Sacks H., Feldman J. (1998). The Women's Interagency HIV Study. WIHS Collaborative Study Group. *Epidemiology*, 9(2):117-25.

## Examples

```
## Not run:
# WIHS analysis
# CIF for HAART and AIDS stratified by IDU

data(wihs, package = "randomSurvivalForest")
wihs.out <- rsf(Surv(time, status) ~ ., wihs, nsplit = 3, ntree = 1000)
wihs.cr <- competing.risk(wihs.out)
cif <- wihs.cr$cif
Time <- wihs.out$timeInterest
idu <- wihs$idu
cif.haart <- cbind(apply(cif[,1][idu == 0,], 2, mean),
                  apply(cif[,1][idu == 1,], 2, mean))
cif.aids <- cbind(apply(cif[,2][idu == 0,], 2, mean),
                 apply(cif[,2][idu == 1,], 2, mean))
matplot(Time, cbind(cif.haart, cif.aids), type = "l",
         lty = c(1,2,1,2), col = c(4, 4, 2, 2), lwd = 3,
         ylab = "Cumulative Incidence")
legend("topleft",
       legend = c("HAART (Non-IDU)", "HAART (IDU)",
                 "AIDS (Non-IDU)", "AIDS (IDU)"),
       lty = c(1,2,1,2), col = c(4, 4, 2, 2), lwd = 3, cex = 1.5)

## End(Not run)
```

# Index

## \*Topic **datasets**

- breast, 2
- follic, 6
- hd, 6
- pbs, 10
- vdv, 40
- veteran, 40
- wihs, 43

## \*Topic **documentation**

- rsf.news, 32

## \*Topic **file**

- competing.risk, 2
- plot.ensemble, 10
- plot.error, 11
- plot.proximity, 13
- plot.variable, 14
- print.rsfc, 22
- vimp, 41

## \*Topic **survival**

- find.interaction, 3
- impute.rsfc, 6
- max.subtree, 8
- pmml2rsfc, 16
- predict.rsfc, 18
- rsfc, 23
- rsfc2pmml, 33
- rsfc2rfz, 35
- varSel, 36

## \*Topic **tree**

- find.interaction, 3
- impute.rsfc, 6
- max.subtree, 8
- pmml2rsfc, 16
- predict.rsfc, 18
- rsfc, 23
- rsfc2pmml, 33
- rsfc2rfz, 35
- varSel, 36

breast, 2

competing.risk, 2

find.interaction, 3

follic, 6

hd, 6

impute.rsfc, 6

max.subtree, 8

pbs, 10

plot.ensemble, 10

plot.error, 11

plot.proximity, 13

plot.rsfc(plot.error), 11

plot.variable, 14

pmml2rsfc, 16

predict.rsfc, 18

print.rsfc, 22

randomSurvivalForest(rsfc), 23

rsfc, 23

rsfc.news, 32

rsfc2pmml, 33

rsfc2rfz, 35

varSel, 36

vdv, 40

veteran, 40

vimp, 41

wihs, 43