Package: randomForestRHF (via r-universe)

October 17, 2026

Version 2.1.0

Date 2026-09-17

Title Random Hazard Forests

Author Hemant Ishwaran [aut], Udaya B. Kogalur [aut, cre]

Maintainer Udaya B. Kogalur <ubk@kogalur.com>

Depends R (>= 4.3.0),

Imports randomForestSRC (>= 3.3.1), varPro (>= 3.0.0), survival

Suggests mlbench, interp, glmnet

SystemRequirements OpenMP

Description Random Hazard Forests (RHF) extend Random Survival Forests (RSF) by directly estimating the hazard function and by accommodating time-dependent covariates through counting-process style inputs. The package fits tree ensembles for dynamic survival prediction, returning hazard, cumulative hazard, integrated hazard, and related performance summaries for training and test data. The methods build on Random Survival Forests described by Ishwaran et al. (2008) <doi:10.1214/08-AOAS169> and on nonparametric hazard modeling with time-dependent covariates described by Lee et al. (2021) <doi:10.1214/20-AOS2028>.

License GPL (>= 3)

URL https://www.randomforestsrc.org/ https://ishwaran.org/

NeedsCompilation yes

Repository https://cran.r-universe.dev

Date/Publication 2026-09-17 13:20:02 UTC

RemoteUrl https://github.com/cran/randomForestRHF

RemoteRef HEAD

RemoteSha acc7619f22987ca4a40f6445ea5c799396215cbc

Contents

auct.rhf 2
importance.rhf 7
plot.rhf 15
predict.rhf 18
print.rhf 21
rhf 22
rhf.news 30
smoothed.hazard.rhf 30
tune.treesize.rhf 33

Index **38**

`auct.rhf` *Time-varying AUC(t) and iAUC for Random Hazard Forests (TDC)*

Description

Compute the time-varying AUC(t) and its time-aggregated summaries for Random Hazard Forests (RHF) with time-dependent covariates (TDC).

Usage

```
auct.rhf(
  object,
  marker = c("cumhaz", "hazard", "chf", "haz"),
  method = c("cumulative", "incident"),
  tau = NULL,
  riskset = c("subject", "record"),
  min.controls = 25,
  nfrac.controls = 0.10,
  min.cases = 1,
  g.floor = 0.10,
  g.floor.q = NULL,
  power = 2,
  ydata = NULL,
  winsor.q = NULL,
  eps = 1e-12,
  bootstrap.rep = 0L,
  bootstrap.refit = FALSE,
  bootstrap.conf = 0.95,
  bootstrap.seed = NULL,
  verbose = TRUE
)

## S3 method for class 'auct.rhf'
print(x, digits = 4, max.rows = 8, ...)
```

```
## S3 method for class 'auct.rhf'
plot(x, bass = 10, xlab = "Time", ylab = NULL,
         main = NULL, ylim = NULL, pch = 16, alpha = .05, ...)
```

Arguments

object	An RHF object. For grow/restore objects (class includes "rhf" and "grow"), OOB prediction matrices are used. For predict objects (class includes "rhf" and "predict"), test data is used (supply ydata= if the object does not include id/yvar).
marker	Which time-varying score to evaluate: "cumhaz" (alias "chf") or "hazard" (alias "haz").
method	Target estimand. "incident" (incident/dynamic) uses cases who fail at time t and controls who are at risk at t; this targets instantaneous hazard discrimination. "cumulative" (cumulative/dynamic) uses cases who have failed by t and controls who are event-free at t; this matches the more commonly used time-dependent AUC (Heagerty and Zheng, 2005).
tau	Optional time horizon. Evaluation is restricted to times t <= tau.
riskset	Risk-set definition for method = "incident": "subject" treats a subject as at risk when entry < t <= stop; "record" uses counting-process rows. Ignored when method = "cumulative".
min.controls	Minimum number of controls required at each evaluated time.
nfrac.controls	Minimum fraction of the total sample to be available as controls at each evaluated time.
min.cases	Minimum number of cases required at each evaluated time.
g.floor	Lower bound used to stabilize inverse-probability-of-censoring weights G(t) (global KM).
g.floor.q	Optional quantile-based floor (computed over the evaluation grid) to further protect tails.
power	Exponent in the Uno-style time weight, effectively using weights proportional to 1 / G(t)^{power}. The classical choice is power = 2.
ydata	Counting-process data frame (id, start, stop, event, ...). Required when object is a predict object that does not carry id/yvar.
winsor.q	Optional winsorization quantile (e.g., 0.99) applied to the time weights to minimize extreme influence.
eps	Small positive constant for numerical stability in weight calculations.
bootstrap.rep	Number of bootstrap replicates for standard errors. Default is 0 (no bootstrap).
bootstrap.refit	Logical. If FALSE (default), uses a subject-level pairs bootstrap with the fitted marker held fixed ("plug-in"). If TRUE, performs a *full refit* at each replicate. Complete counting-process histories are resampled by subject with multiplicity, and each sampled copy receives a new bootstrap ID. Refit models use the original RHF tuning parameters and inherit the saved hazard/CHF settings from object$forest$parms$hazard.config.

`bootstrap.conf` Confidence level for pointwise normal bands on AUC(t) when `bootstrap.rep > 0` (e.g., `0.95`). Set to `NA` to suppress bands.

`verbose` Logical; if `TRUE`, enable bootstrap progress messages. Messages can be silenced with `suppressMessages()`.

`bootstrap.seed` Optional integer seed used before bootstrap resampling for reproducibility.

`x`, `digits`, `max.rows`, `bass`, `xlab`, `ylab`, `main`, `ylim`, `pch`, `alpha`, `...`
Standard arguments for the `print` and `plot` methods.

Details

What is estimated. At each evaluation time t, the function computes an AUC for the chosen marker (either cumulative hazard or hazard), using the specified case/control definition:

- *Incident/dynamic* (`method = "incident"`): cases fail at t; controls are at risk at t.
- *Cumulative/dynamic* (`method = "cumulative"`): cases have failed by t; controls are event-free at t.

The per-time AUC(t) values are combined in two ways:

- `iAUC.uno`: an inverse-probability-of-censoring weighted (IPCW) average of AUC(t) over time, using a global KM estimate `G(t)` and a weight proportional to `1 / G(t)^{power}` (Uno-style time weighting). Stabilization parameters include `g.floor`, `g.floor.q`, and `winsor.q`.
- `iAUC.std`: a simple time-standardized mean of AUC(t) over the evaluation grid (trapezoidal rule divided by span). This quantity is sensitive to the time window and the shape of AUC(t) across time.

Markers. For `method = "cumulative"`, using `marker = "cumhaz"` matches standard time-dependent AUCs. Using `marker = "hazard"` with `"cumulative"` often yields smaller AUC(t), as it targets instantaneous risk rather than cumulative risk.

Risk-set choice. When `method = "incident"` there are two ways to define controls at time `t`:

- `riskset = "subject"`: at risk when `entry < t <= Tstop` (one interval per subject). Fast and appropriate when subject rows tile continuous follow-up (no gaps).
- `riskset = "record"`: at risk when a counting-process row satisfies `start < t <= stop`. Exact under gaps but may yield fewer controls at sparse times.

If rows tile the entire follow-up, the two definitions coincide (up to ties). With gaps, `"subject"` can over-include controls (lower variance but a conceptual mismatch), while `"record"` is exact but can be sparser. For `method = "cumulative"` the risk-set choice is ignored.

Bootstrap. When `bootstrap.rep > 0`, two modes are available:

- *Plug-in* (`bootstrap.refit = FALSE`): subject-level pairs bootstrap holding the fitted marker matrix fixed. Weights the within-time AUC(t) using resampled multiplicities and recomputes Uno's time weights via a weighted KM.
- *Refit* (`bootstrap.refit = TRUE`): for each replicate, resample complete subject histories with replacement, retaining repeated draws as separate bootstrap subjects, *refit* the RHF using the original tuning parameters and saved hazard/CHF configuration, and re-evaluate AUC(t). The returned per-time SEs are matched to the original evaluation grid. For predict objects, the refit

is performed using the data carried in the object; predictions are then recomputed for those data. This is computationally heavier but captures variability from model fitting. If every refit fails, the function reports the first captured failure rather than silently returning an all-NA bootstrap result.

The plot method draws a base-R shaded band for AUC(t) when bootstrap SEs are present.

Value

An object of class "auct.rhf" with elements:

- AUC.by.time: data frame with columns time, AUC, n.cases, n.ctrl, G, W.
- iAUC.uno: IPCW (Uno-style) time-averaged AUC.
- iAUC.std: time-standardized mean of AUC(t).
- marker, method, riskset, power, g.floor, g.floor.q, winsor.q, times: metadata.
- diag.riskset: Quick diagnostic comparing control counts under riskset = "subject" vs "record" on a small subset of times (list with times, n.ctrl.subject, n.ctrl.record, n.diff, prop.times.different).
- boot: present when bootstrap.rep > 0 with AUC.se (per-time SE matched to the reporting grid), iAUC.uno.se, iAUC.std.se, conf.level, AUC.lower, AUC.upper, rep, and mode ("plug-in" or "refit").

Generic methods print and plot are provided for compact summaries and visualization.

Author(s)

Hemant Ishwaran and Udaya B. Kogalur

References

Heagerty, P. J., Lumley, T., and Pepe, M. S. (2000). Time-dependent ROC curves for censored survival data and a diagnostic marker. *Biometrics*, 56(2), 337–344.

Heagerty, P. J., and Zheng, Y. (2005). Survival model predictive accuracy and ROC curves. *Biometrics*, 61(1), 92–105.

Uno, H., Tian, L., Cai, T., Kohane, I. S., Wei, L.-J. (2013). A unified inference procedure for a class of measures to assess improvement in risk prediction systems with survival data. *Statistics in Medicine*, 32(14), 2430–2442.

See Also

rhf, predict.rhf

Examples

```
## ------------------------------------------------------------
## Peak VO2 example
## ------------------------------------------------------------
```

```
data(peakVO2, package = "randomForestSRC")
d <- convert.counting(Surv(ttodead, died) ~ ., peakVO2)
f <- "Surv(id, start, stop, event) ~ ."

o <- rhf(f, d, ntree = 25, nodesize = 5)

## AUC(t) with cumulative/dynamic definition and cumhaz marker
a.chf <- auct.rhf(o, marker = "cumhaz", method = "cumulative")

## AUC(t) with incident/dynamic definition and hazard marker
a.haz <- auct.rhf(o, marker = "hazard", method = "incident")

print(a.chf)
print(a.haz)

oldpar <- par(mfrow = c(1, 2))
plot(a.chf, main = "AUC(t): cumulative + cumhaz")
plot(a.haz, main = "AUC(t): incident + hazard")
par(oldpar)

## ------------------------------------------------------------
##  TDC illustration with training/testing
## ------------------------------------------------------------

trn <- hazard.simulation(1)$dta
tst <- hazard.simulation(1)$dta
f <- "Surv(id, start, stop, event) ~ ."
o <- rhf(f, trn, ntree = 25)
p <- predict(o, tst)
a.trn <- auct.rhf(o)
a.tst <- auct.rhf(p)

oldpar <- par(mfrow = c(1, 2))
plot(a.trn, main = "AUC(t): chf marker, train", ylim = c(0.5,1))
plot(a.tst, main = "AUC(t): chf marker, test",  ylim = c(0.5,1))
par(oldpar)

## ------------------------------------------------------------
## Bootstrap SEs and shaded band
## ------------------------------------------------------------
d <- hazard.simulation(1)$dta
f <- "Surv(id, start, stop, event) ~ ."
o <- rhf(f, d)


oldpar <- par(mfrow = c(1, 2))


## plug-in bootstrap
a.bs1 <- auct.rhf(o, marker = "cumhaz", method = "cumulative",
                  bootstrap.rep = 20, bootstrap.seed = 123)
plot(a.bs1, main = "AUC(t) with bootstrap band (plug-in)")
```

```
## refit bootstrap (can be slow)
a.bs2 <- auct.rhf(o, marker = "cumhaz", method = "cumulative",
                  bootstrap.rep = 10, bootstrap.refit = TRUE, bootstrap.seed = 7)
plot(a.bs2, main = "AUC(t) with bootstrap band (refit)")


par(oldpar)
```

`importance.rhf` *Time-localized VarPro importance for random hazard forests*

Description

Computes time-localized variable importance for a random hazard forest (RHF) by using VarPro (variable priority) importance and restricting rule and near-miss memberships to pseudo-individuals whose start-stop intervals overlap selected windows from the master time grid `time.interest`. The working response used for VarPro importance is now taken directly from the RHF object as the logarithm of the upstream integrated hazard exposure, giving a fast localized view of how variable importance evolves over time.

Usage

```
importance.rhf(o,
 cache = NULL,
 time.index = NULL,
 trim = 0.1,
 sort = TRUE,
 max.rules.tree,
 max.tree,
 eps = 1e-6,
 y.external = NULL,
 verbose = FALSE,
 ...)

varpro.cache.rhf(o,
 max.rules.tree = 150L,
 max.tree = 150L,
 y.external = NULL,
 eps = 1e-6,
 verbose = FALSE)

## S3 method for class 'importance.rhf'
print(x,
 top = 10L,
 rank.by = c("q90", "median", "mean", "max"),
 digits = 4L,
 scientific.threshold = 1e4,
```

```
 ...)

## S3 method for class 'importance.rhf'
as.data.frame(x,
  row.names = NULL,
  optional = FALSE,
  format = c("long", "variable_by_time", "time_by_variable"),
  ...)

dotmatrix.importance.rhf(x,
 vars = NULL,
 top.n.union = 15L,
 variable.labels = NULL,
 time.labels = NULL,
 sort.by = c("q90", "sum", "max", "mean", "median", "alphabetical", "cluster", "none"),
 sort.abs = TRUE,
 transform = c("none", "log10"),
 color.by = c("value", "sign", "single", "none"),
 point.color = "steelblue4",
 value.colors = c("grey85", "steelblue4"),
 sign.colors = c("firebrick3", "grey90", "steelblue4"),
 pch = 16,
 cex.range = c(0.6, 3.2),
 size.cap = 0.99,
 color.cap = 0.99,
 alpha = 0.9,
 show.grid = FALSE,
 grid.col = "grey92",
 legend = TRUE,
 legend.args = list(),
 display.note = FALSE,
 xlab = "",
 ylab = "",
 main = "RHF time-localized VarPro importance",
 axis.cex = 0.7,
 var.cex = 0.7,
 time.label.srt = 45,
 save.plot = FALSE,
 out.file = "rhf_time_varpro_dotmatrix.pdf",
 width = 11,
 height = NULL,
 mar = NULL,
 legend.width = 0.7,
 point.args = list(),
 ...)

barplot.importance.rhf(x,
 vars = NULL,
```

```
 top.n.union = 15L,
 variable.labels = NULL,
 time.labels = NULL,
 sort.by = c("q90", "sum", "max", "mean", "median", "alphabetical", "cluster", "none"),
 sort.abs = TRUE,
 transform = c("none", "log10"),
 color.by = c("value", "sign", "single", "none"),
 bar.color = "steelblue4",
 value.colors = c("grey85", "steelblue4"),
 sign.colors = c("firebrick3", "grey90", "steelblue4"),
 bar.width = 0.65,
 bar.max.height = NULL,
 size.cap = 0.99,
 color.cap = 0.99,
 alpha = 0.9,
 show.grid = FALSE,
 grid.col = "grey92",
 zero.line = TRUE,
 zero.col = "grey65",
 legend = TRUE,
 legend.args = list(),
 display.note = FALSE,
 xlab = "",
 ylab = "",
 main = "RHF time-localized VarPro importance",
 axis.cex = 0.7,
 var.cex = 0.7,
 time.label.srt = 45,
 save.plot = FALSE,
 out.file = "rhf_time_varpro_barplot.pdf",
 width = 11,
 height = NULL,
 mar = NULL,
 legend.width = 0.7,
 border = NA,
 bar.args = list(),
 ...)

## S3 method for class 'importance.rhf'
plot(x,
 type = c("barplot", "dotmatrix", "lines"),
 vars = NULL,
 variable.labels = NULL,
 top = 10L,
 rank.by = c("q90", "median", "mean", "max"),
 curve = c("step", "line", "lowess"),
 smooth.f = 2/3,
 display.cap = 0.99,
```

```
display.note = FALSE,
xlab = NULL,
ylab = NULL,
lty = 1,
lwd = 2,
line.colors = NULL,
line.args = list(),
legend = TRUE,
legend.args = list(),
...)
```

Arguments

o	A RHF object with class "rhf".
cache	Optional cache object returned by varpro.cache(). If NULL, the cache is built internally. Supplying a cache is useful when repeated calls are made.
time.index	Optional vector identifying which windows of the time grid o$time.interest are to be analyzed. This may be an integer index vector or a logical vector of length length(o$time.interest). If omitted, all windows are used.
trim	Tuning parameter passed to the underlying VarPro importance workhorse. trim controls winsorized aggregation across trees.
sort	Logical. If TRUE, variables are ordered within each window in decreasing importance before the long-format output is assembled.
max.rules.tree, max.tree	Arguments controlling rule extraction when the cache is built.
y.external	Optional externally supplied working response. When NULL, the working response is built internally from the RHF object's integrated hazard exposure values.
eps	Nonnegative value added before taking the logarithm of the integrated hazard exposure when y.external is not supplied.
verbose	Logical. If TRUE, reports cache construction and per-window progress.
x	An object of class "importance.rhf".
top, rank.by	Arguments used by print() and by plot(type = "lines"). Printing now ranks variables robustly over time by default using rank.by = "q90". The line plot also uses rank.by when vars is omitted.
digits, scientific.threshold	Formatting controls for print(), used to keep very large importance values readable.
row.names, optional	Included for compatibility with as.data.frame().
format	Output format for as.data.frame(). "long" returns the long-format table, "variable_by_time" returns a data frame whose rows are variables and columns are times, and "time_by_variable" returns the transpose with window metadata.

type, vars, top.n.union
: Arguments controlling which variables are displayed and which plot is produced. type = "barplot" (the default) gives the time-by-variable bar-matrix display, type = "dotmatrix" gives the companion dot-matrix display, and type = "lines" gives a line, step, or smoothed view for selected variables. When vars is omitted, the line plot chooses top variables using rank.by, while the dot-matrix and bar-matrix plots use the union of the top top.n.union variables across time.

curve, smooth.f, lty, lwd, line.colors, line.args, display.cap, display.note
: Arguments for type = "lines". curve chooses between step, ordinary line, and lowess-smoothed displays; smooth.f is passed to stats::lowess() when needed; lty, lwd, and line.colors control line appearance; and line.args is a named list of additional arguments for the graphics::lines() layer. display.cap applies display-only quantile capping to stabilize the vertical scale in the presence of extreme spikes. display.note toggles the on-plot capping note for every plot type.

variable.labels, time.labels, sort.by, sort.abs
: Arguments controlling labels and ordering. variable.labels is used by all plot types and may be a named vector or a two-column data frame. time.labels, sort.by, and sort.abs apply to the dot-matrix and bar-matrix displays. An unnamed time.labels vector must have one label per time window. Variables may be ordered by robust aggregate importance, alphabetically, hierarchical clustering, or left in their existing order.

transform, color.by, point.color, bar.color, value.colors, sign.colors, pch, cex.range, bar.width, bar.max.height, size.cap, color.cap, alpha
: Arguments controlling dot size or bar height, color encoding, display-only quantile capping, and transparency. For the bar-matrix plot, bar.width controls the width of each bar within a time cell and bar.max.height controls the largest bar height within a variable row. When color.by = "sign", magnitude controls size or height and signed importance controls color; signed bars extend upward or downward from the row center. Negative values are omitted, with a warning, for the other color modes. A one-color value.colors or sign.colors vector is allowed and produces constant coloring. To omit the continuous color-scale legend as well, use color.by = "single" together with point.color for a dot matrix or bar.color for a bar matrix.

legend
: Logical. If TRUE, draw the applicable matrix or line-plot legend.

legend.args
: Named list controlling the legend for every plot type. For type = "dotmatrix" and type = "barplot", supported entries are title, color.title, cex, and title.cex. Omitting title, or setting it to TRUE, retains the automatic magnitude title. Setting title = FALSE, NULL, or "" suppresses both matrix-legend headings; a character string supplies a custom magnitude title. color.title can independently retain, suppress, or replace the color-scale title. cex controls numeric legend labels and defaults to 0.85; title.cex controls legend headings and defaults to 0.9. A specifically supplied color.title takes precedence over the general title suppression. Increase legend.width when using a long custom title or unusually large legend text. For type = "lines", the list is passed to graphics::legend(), allowing options such as x, cex, bty, ncol, and title.

point.args, bar.args
: Named lists passed to the dot graphics::points() layer and bar graphics::rect() layer, respectively. Data-driven coordinates, sizes, heights, and colors are protected and cannot be replaced through these lists.

xlab, ylab, main, axis.cex, var.cex, time.label.srt, show.grid, grid.col, zero.line, zero.col, border, mar, legend.width, width, height, save.plot, out.file
: Display, layout, and export options for the plotting helpers. The matrix plots use blank axis labels and do not draw guide lines by default. When mar = NULL, the left margin is computed from the physical width of the resolved variable labels, retaining space for long informatics names without imposing a large fixed margin for short names. The bar-matrix magnitude key uses compact, non-overlapping reference-bar spacing; when the magnitude and color titles are identical, the color key moves upward to avoid unused space between the legend sections. Files can be written with save.plot = TRUE. zero.line and zero.col control row-center reference lines for signed bar plots; border controls rectangle borders.

...
: Additional named arguments for the initial graphics::plot() frame. Point, bar, and line layers use point.args, bar.args, and line.args, respectively. Legend formatting uses legend.args for both matrix and line plots. Structural matrix geometry such as xlim and ylim is controlled by the matrix helpers and cannot be replaced through ...; line plots accept user-supplied xlim and ylim through ....

Details

Implements time-localized RHF variable priority. The time grid is given by time.interest. For each selected time window, the method uses the counting process start–stop records whose intervals overlap that window. The forest rules and near-miss sets are taken from the fitted RHF object and are reused across time windows.

For RHF objects, variable priority (VarPro) is applied in a regression-style form. The working response is the logarithm of the integrated hazard exposure assigned by RHF over the relevant interval-window overlap. Local priority for a variable is computed by comparing this working response among records satisfying a forest rule with the corresponding response among records in the rule's near-miss set, obtained by releasing that variable from the rule. Temporal localization comes from restricting the calculation to records active in each time window, rather than rebuilding the rules or refitting the forest at each time point.

The helper varpro.cache() stores the minimum information needed for repeated localized importance calculations: a regression-style rule template, window metadata, the working response source, and precomputed window-local rule statistics. During cache construction, raw OOB and complementary memberships are converted into compact per-window rule summaries, so the later window sweep does not need to rescan membership vectors.

The returned importance matrix has variables in rows and selected time windows in columns. Column names correspond to the right endpoints of the selected windows. The long-format table contains the same values together with window metadata such as start, stop, midpoint, number at risk, and number of active rules.

Printing and plotting share a robust strategy. Summaries default to a robust over-time ranking based on the 90th percentile, and the plotting helpers apply optional quantile capping for display only.

This prevents rare extreme spikes from flattening curves while preserving the original importance matrix for downstream analyses.

For consistent base-graphics behavior, `...` customizes the initial plot frame for every plot type. Layer-specific options belong in `point.args`, `bar.args`, or `line.args`. The single `legend.args` list controls the custom matrix legend or is passed to `graphics::legend()` for a line plot. Data-driven coordinates, sizes, heights, and colors remain under control of the plotting helper. Line-plot limits may still be supplied as `xlim` or `ylim` through `...`.

Value

`varpro.cache()` returns an object containing cached rule memberships, the working response used for importance, start-stop information for pseudo-individuals, time-window metadata, and the rule extraction settings.

`importance.rhf()` returns a list including:

- `importance.matrix`: matrix of localized importance values with variables in rows and selected time windows in columns.
- `importance.long`: long-format data frame containing variable, time, window metadata, and localized importance.
- `window.info`: data frame describing the analyzed windows, including start, stop, midpoint, `n.risk`, and `n.rules`.
- `y.source`: source of the working response. This is `"int.haz.oob"`, `"int.haz.test"`, or `"y.external"`.
- `trim`: tuning value used in importance aggregation.

`print()` returns its input invisibly after displaying a short summary that includes robust over-time summaries for the leading variables.

`as.data.frame()` returns one of the supported data-frame views.

`dotmatrix.importance.rhf()` produces a base-R dot-matrix plot and returns plotting metadata invisibly, including the resolved `legend.args` settings.

`barplot.importance.rhf()` produces a base-R bar-matrix plot and returns plotting metadata invisibly, including the resolved `legend.args` settings.

`plot()` returns invisibly the result of the underlying plotting helper. For `type = "lines"`, the returned display-capped matrix has attributes containing the selected variable codes, resolved labels and time values.

References

Ishwaran H. (2025). Multivariate Statistics: Classical Foundations and Modern Machine Learning. Chapman and Hall.

Ishwaran H., Kogalur U.B., Hsich E.M. and Lee D.K. (2026). Random hazard forests.

Lu, M. and Ishwaran, H. (2024). Model-independent variable selection via the rule-based variable priority. arXiv e-prints, pp.arXiv-2409.

Lu, M. and Ishwaran, H. (2025). Individual variable priority: a model-independent local gradient method for variable importance. *Artificial Intelligence Review*, 58:407.

Zhou L., Lu M. and Ishwaran H. (2026). Variable priority for unsupervised variable selection. *Pattern Recognition*, 172:112727.

See Also

rhf

Examples

```
####################################################################
##
## simulation model
##
####################################################################

## draw simulation (can be modified)
n <- 400
p <- 10
simid <- 2
d <- hazard.simulation(type = simid, n = n, p = p, nrecords = 4)$dta

## fit a RHF model with weighted mtry (use for high-dimension)
f <- "Surv(id, start, stop, event) ~ ."
o <- rhf(f, d, ntree = 50, nsplit = 5, xvar.wt = xvar.wt.rhf(f, d))
print(o)

## time-localized RHF importance across the full time grid
imp.t <- importance.rhf(o)
print(imp.t)

## extract the variable-by-time matrix
print(head(imp.t$importance.matrix))

oldpar <- par(mfrow=c(1,1))

## bar-matrix importance plot (default)
plot(imp.t)

## companion dot-matrix importance plot
plot(imp.t, type = "dotmatrix")

## suppress matrix-legend titles and enlarge its numeric labels
plot(imp.t, legend.args = list(title = FALSE, cex = 1.15))

## retain titles but control their size separately
plot(imp.t, legend.args = list(cex = 1.05, title.cex = 0.9))

## step-style importance line plot for the top variables
## (ranked by the 90th percentile over time and display-capped at q99)
plot(imp.t, type = "lines", top = 10)

## smoothed importance plot for all variables with display capping
plot(imp.t, type = "lines", curve = "lowess", smooth.f = 0.5, display.cap = 0.95)

## robust ordering and display capping
plot(imp.t, sort.by = "q90", size.cap = 0.99, color.cap = 0.99)
```

```
## bar-matrix plot using signed colors, if signed importance values are present
plot(imp.t, color.by = "sign")

par(oldpar)


## reuse a cache for repeated calls on subsets of the time grid
cache <- varpro.cache(o)
imp.t.sub <- importance.rhf(
  o,
  cache = cache,
  time.index = seq(1, length(o$time.interest), by = 5),
  verbose = TRUE
)

## long-format export
print(head(as.data.frame(imp.t.sub)))
```

`plot.rhf` *Plot smoothed hazard and cumulative hazard plots from RHF analysis*

Description

Plot case specific hazard and cumulative hazard (CHF) from a fitted random hazard forest (RHF) object. Hazards are smoothed with `stats::supsmu`. Optional scaling places hazards on the same scale as the CHF for easier comparison.

Usage

```
## S3 method for class 'rhf'
plot(x, idx = NULL, scale.hazard = FALSE,
         ngrid = 30, bass = 0, q = 0.99, grid = FALSE,
         col = NULL, lty = NULL, legend.loc = "topright",
         jitter.factor = 1, lwd = 4,
         hazard.only = TRUE, legend.show = TRUE, ...)
```

Arguments

`x`	An `rhf` fit object returned by `rhf`.
`idx`	Subject identifiers selecting which cases to plot. Values must match those in `unique(x$id)`; by default the first value is used.
`scale.hazard`	Logical or numeric. If `TRUE`, hazards are multiplied by the local time step so they approximate CHF increments. If numeric, hazards are multiplied by the given constant.
`ngrid`	Number of equally spaced points used when `grid = TRUE`.
`bass`	Smoothing parameter passed to `stats::supsmu`.

q	Quantile used to trim extreme hazard values before smoothing.
grid	If TRUE, overlays smoothed CHF and its derivative.
col	Colors for selected cases. Recycled as needed.
lty	Line types for hazard and CHF curves.
legend.loc	Legend location. Passed to legend.
jitter.factor	Amount of horizontal jitter applied to hazard spikes.
lwd	Line width for hazard spikes.
hazard.only	If TRUE, only the hazard is drawn.
legend.show	If TRUE, show a legend when plotting multiple cases.
...	Additional graphics parameters passed to plot, including optional axis labels such as xlab and ylab.

Details

The function displays out of bag hazard and CHF estimates on the grid x$time.interest. Hazards are smoothed using supsmu after trimming large values for stability. Scaling the hazard allows direct visual comparison with CHF. Plots are drawn to the current device; no model re-estimation is performed.

Value

Invisibly returns NULL. The function is used for its plotting side effects.

Author(s)

Hemant Ishwaran and Udaya B. Kogalur

References

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

Ishwaran H., Kogalur U.B., Blackstone E.H. and Lauer M.S. (2008). Random survival forests. *Annals of Applied Statistics*, 2: 841–860.

Lee D.K., Chen N., and Ishwaran H. (2021). Boosted nonparametric hazards with time dependent covariates. *Annals of Statistics*, 49: 2101–2128.

Ishwaran H., Lee D.K. and Hsich E.M. (2025). Random hazard forests.

See Also

hazard.simulation, predict.rhf, rhf, smoothed.hazard.rhf

Examples

```
## ------------------------------------------------------------
##  time static peakVO2
## ------------------------------------------------------------
data(peakVO2, package = "randomForestSRC")
d <- convert.counting(Surv(ttodead, died)~., peakVO2)
f <- "Surv(id, start, stop, event) ~ ."

set.seed(1)
o <- rhf(f, d)

oldpar <- par(mfrow=c(1,1))

## plot selected cases
ids <- o$ensemble.id[1:3]
plot(o, idx = ids)

## custom axis label
plot(o, idx = ids, ylab = "my hazard")

## hazard only
plot(o, idx = ids)

## scaled hazard with CHF
plot(o, idx = ids, scale.hazard = TRUE, hazard.only = FALSE)

## auxiliary grid with smoother control
plot(o, idx = ids, grid = TRUE, ngrid = 60, bass = 2, hazard.only = FALSE)

## multiple cases, no legend
plot(o, idx = o$ensemble.id[1:10], lwd = 0, legend.show = FALSE)

## lowess median smoothed hazard
s <- smoothed.hazard(o)  ## default method="median.loess"
plot(s, idx = o$ensemble.id[1:10])
plot(s, idx = o$ensemble.id[1:10], lwd = 0)


par(oldpar)

## ------------------------------------------------------------
##  complex simulated time dependent covariate hazards
## ------------------------------------------------------------
f <- "Surv(id, start, stop, event) ~ ."
sim2 <- hazard.simulation(2)$dta
o2 <- rhf(f, sim2)

oldpar <- par(mfrow=c(1,1))
plot(o2, 1)
par(oldpar)
```

```
## ------------------------------------------------------------
##  same complex simulation, but using smoothed hazard
## ------------------------------------------------------------
f <- "Surv(id, start, stop, event) ~ ."
sim2 <- hazard.simulation(2)$dta
o2 <- rhf(f, sim2)

s2.loess <- smoothed.hazard(o2, method="loess")
s2.med.loess <- smoothed.hazard(o2, method="median.loess")

oldpar <- par(mfrow=c(1,1))
plot(o2, 1)
plot(s2.loess, 1)
plot(s2.med.loess, 1)
plot(s2.med.loess, 1:10, lwd = 0)
par(oldpar)
```

`predict.rhf` *Prediction on Test Data for Random Hazard Forests*

Description

Obtain predicted values on test data using a trained random hazard forest, or restore predictions for its original training data.

Usage

```
## S3 method for class 'rhf'
predict(object, newdata,  get.tree = NULL,
     block.size = 10, membership = TRUE, adaptive = TRUE,
     seed = NULL, do.trace = FALSE,...)
```

Arguments

`object`	An `rhf` object returned from a previous training call to `rhf`.
`newdata`	Test data frame. For a forest grown from ordinary `Surv(time, event)` input, `newdata` may be supplied in the same ordinary survival format using the original time and event column names. For a forest grown from counting-process input, `newdata` uses `Surv(id, start, stop, event)` columns as in the training data. If omitted, the original training data are used and the full training forest is restored.
`get.tree`	Optional vector of integer indices specifying which trees to use for ensemble predictions. Defaults to using all trees in the forest. Currently ignored, future functionality.

block.size	Controls how cumulative error rate is reported. To obtain cumulative error every n trees, set this to an integer between 1 and ntree. Currently ignored.
membership	Logical flag indicating whether terminal node membership and inbag information should be returned.
adaptive	Logical flag controlling the default trimmed-mean protocol used by prediction and restore mode. The default TRUE inherits the adaptive trim configuration saved with the trained forest. Setting adaptive = FALSE replaces the default or inherited trim grid by the fixed value 0.05, unless an expert-level coe.trim value is supplied through ....
seed	Negative integer specifying the random seed for reproducibility.
do.trace	Number of seconds between progress updates printed to the console.
...	Additional optional arguments passed to internal methods.

Details

For a forest grown directly from Surv(time, event) data, predict.rhf() retains the original response names from the training call. When both response columns are present in newdata, each test row is converted internally to one interval with start = 0, stop = time, and the supplied event indicator. This gives the same test-data representation that would have been obtained by calling convert.counting() explicitly. The time and event columns must be either both present or both absent. When both are absent, prediction can still return terminal membership and training-derived node summaries, but case-specific test hazard and cumulative-hazard trajectories are not assembled because no test follow-up interval has been supplied.

Forests grown from Surv(id, start, stop, event) data retain the existing counting-process prediction interface. In either format, predictor names must match those used to grow the forest.

Prediction inherits the hazard calculation configuration saved with the trained forest. In ordinary test-data prediction there is no OOB objective for reselecting the trim value, so the grow-time selected index is used. In restore mode, the stored forest can reconstruct OOB quantities and may reselect the trim value over the active candidate grid. Setting adaptive = FALSE changes that grid to the single value 0.05, unless coe.trim is supplied through ....

Value

An object of class c("rhf", "predict", family). The returned list contains the fitted forest together with prediction summaries on the evaluation grid time.interest. Important components include:

- hazard.test, chf.test, risk.test, and int.haz.test: test-set hazard, cumulative hazard, risk, and integrated-hazard summaries when newdata is supplied.
- hazard.oob, chf.oob, risk.oob, and int.haz.oob: out-of-bag summaries for the training data.
- hazard.inbag, chf.inbag, risk.inbag, and int.haz.inbag: in-bag summaries when available.
- id, yvar, and xvar: identifiers and processed outcome/predictor data used by the returned prediction object.

- `event.process` and `input.info`: the trained event-process classification and the original survival-input format and response map.
- `pseudo.membership` and `inbag`: terminal-node membership and inbag information when `membership = TRUE`.
- `forest`: the fitted forest object used to generate the predictions.

If `newdata` is omitted, the function restores predictions for the original training data using the stored forest and returns the same class of object.

Author(s)

Hemant Ishwaran and Udaya B. Kogalur

References

Ishwaran H. and 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.

Lee, D.K. and Chen N. and Ishwaran H (2021). Boosted nonparametric hazards with time-dependent covariates. *Annals of Statistics*, 49: 2101-2128.

Ishwaran H. (2025). Multivariate Statistics: Classical Foundations and Modern Machine Learning. Chapman and Hall.

Ishwaran H., Kogalur U.B., Hsich E.M. and Lee D.K. (2026). Random hazard forests.

See Also

`rhf`

Examples

```
## ------------------------------------------------------------
## canonical train/test example (synthetic data)
## ------------------------------------------------------------

simID <- 1
trn <- hazard.simulation(simID)$dta
tst <- hazard.simulation(simID)$dta
f <- "Surv(id, start, stop, event) ~ ."

## training
o <- rhf(f, trn, ntree = 3)
print(o)

## testing
p <- predict(o, tst)
print(p)

## fixed trim protocol
p.fixed <- predict(o, tst, adaptive = FALSE)
```

```
## ------------------------------------------------------------
##  pbc: train/test example
## ------------------------------------------------------------

library("randomForestSRC")
data(pbc, package = "randomForestSRC")
pbc.raw <- na.omit(pbc)
set.seed(7)
trn <- sample(
  seq_len(nrow(pbc.raw)),
  size = floor(nrow(pbc.raw) * .75),
  replace = FALSE
)

pbc.trn <- pbc.raw[trn, , drop = FALSE]
pbc.tst <- pbc.raw[-trn, , drop = FALSE]
f <- Surv(days, status) ~ .

## train and predict directly with ordinary right-censored data
o <- rhf(f, pbc.trn)
print(predict(o, pbc.tst))

## restore the training forest
print(predict(o))
```

`print.rhf` *Print Output from a Random Hazard Forest Analysis*

Description

Print a compact summary of a Random Hazard Forest analysis. This is the default print method for objects produced by `grow` or `predict`.

Usage

```
## S3 method for class 'rhf'
print(x, digits = 3, label.width = 34, ...)
```

Arguments

`x`	An object of class `rhf` created by `grow` or `predict`. When `predict` is called without new data ("restore mode"), the original forest and its OOB risk are returned.
`digits`	Number of digits used for printed numeric summaries.

label.width	Width used to right-justify labels in the printed output.
...	Additional arguments (currently unused).

Details

The printed summary includes basic sample information, forest size characteristics, event counts when available, and risk metrics (in-bag, OOB, or test risk depending on how the object was created).

Value

Returns the object x invisibly.

Author(s)

Hemant Ishwaran and Udaya B. Kogalur

---

rhf *Random Hazard Forests*

Description

Random Hazard Forest (RHF) is a tree-ensemble survival method that estimates case-specific hazards and cumulative hazard functions on a working time grid. It accepts ordinary right-censored data through `Surv(time, event)` and time-dependent covariates through counting-process `Surv(id, start, stop, event)` input.

Usage

```
rhf(formula,
      data,
      ntree = 500,
      nsplit = 10,
      treesize = NULL,
      nodesize = NULL,
      block.size = 10,
      bootstrap = c("by.root", "none", "by.user"),
      samptype = c("swor", "swr"),
      samp = NULL,
      case.wt = NULL,
      membership = TRUE,
     sampsize = if (samptype == "swor") function(x){x * .632} else function(x){x},
      xvar.wt = NULL,
      ntime = 50,
      min.events.per.gap = 10,
      adaptive = TRUE,
      seed = NULL,
      do.trace = FALSE,
      ...)
```

Arguments

formula	Model formula specifying the response and predictors. The response may be ordinary right-censored survival data, `Surv(time, event)`, or RHF counting-process data, `Surv(id, start, stop, event)`.
data	Data frame containing the variables referenced in `formula`. Ordinary right-censored input has one row per subject; counting-process input may have multiple rows per subject.
ntree	Number of trees to grow in the ensemble.
nsplit	Non-negative integer specifying the number of random split points to consider for each variable. When set to 0, all possible split points are evaluated (deterministic splitting), which may be slower. The default is 10.
treesize	Specifies the desired tree size, defined as the number of terminal nodes. If unspecified, an internal default is used.
nodesize	Minimum number of cases required in a terminal node. If not specified, an internal default is used.
block.size	Controls the granularity of cumulative error rate reporting. Setting this to an integer between 1 and `ntree` yields the cumulative error every `block.size` trees. Currently ignored.
bootstrap	Bootstrap strategy for generating inbag samples. The default is `"by.root"`, which bootstraps by sampling with or without replacement (default is without; see `samptype`). If set to `"none"`, no bootstrapping is performed (OOB predictions and errors are then unavailable). The option `"by.user"` uses a user-defined bootstrap specified by `samp`.
samptype	Sampling type used when `bootstrap = "by.root"`. Options are `"swor"` (sampling without replacement, default) and `"swr"` (sampling with replacement).
samp	User-specified bootstrap when `bootstrap = "by.user"`. Should be an `n` by `ntree` array, where each entry gives the number of times a case appears in the inbag sample for a given tree.
case.wt	Non-negative vector of case weights (not required to sum to 1). Higher weights increase the probability of an observation being selected during bootstrapping or subsampling. Using real-valued weights is preferred over integer weights.
membership	Logical flag indicating whether to return terminal node membership and inbag information.
sampsize	Specifies the size of the bootstrap sample when `bootstrap = "by.root"`. For sampling without replacement, it represents the requested subsample size (default is 0.632 times the sample size). For sampling with replacement, it equals the sample size. Can be supplied as a numeric value or a function.
xvar.wt	Non-negative vector of variable selection probabilities for splitting. Values do not need to sum to 1. Defaults to uniform selection probability.
ntime	Controls the working time grid used for all ensemble calculations. Can be: `ntime >= 1` **(integer):** Request approximately `ntime` grid points chosen from the observed event times. When used with `min.events.per.gap`, grid points are selected adaptively so that each interval contains at least `min.events.per.gap` events (when possible); therefore the resulting grid may contain fewer than `ntime` points when events are sparse (especially in the tail).

numeric vector: A user-supplied set of time points. Each value is aligned (snapped) to the nearest observed event time and duplicates are removed.

`0` **or** `NULL`**:** Use all observed (unique) event times.

`min.events.per.gap`
: Minimum number of observed events required in each time interval when `ntime` is specified as an integer. Together with ntime, this provides an automatic event-balanced grid that allocates more resolution where events are dense and avoids sparse tail intervals.

`adaptive`
: Logical value controlling the default trimmed-mean aggregation protocol. When `TRUE`, the default, RHF evaluates a candidate vector of trimming fractions and retains the value selected by the OOB-risk criterion. When `FALSE`, the default candidate vector is replaced by the single value `coe.trim = 0.05`. This can be useful when tuning other parameters, such as `treesize`, without simultaneously optimizing the trimming fraction, otherwise leave at the default.

`seed`
: Negative integer setting the random seed for reproducibility.

`do.trace`
: Time in seconds between progress updates printed to the console.

`...`
: Additional expert-level controls passed to internal RHF methods.

Details

Overview

`rhf()` is the main fitting function for Random Hazard Forests (RHF). Ordinary right-censored data can be supplied directly as `Surv(time, event)` with one row per subject. RHF converts this internally to a one-interval predictable process with `start = 0`, `stop = time`, and time-static predictors. Time-dependent data are supplied explicitly in counting-process form as `Surv(id, start, stop, event)`. Multiple rows for the same `id` then describe an evolving covariate path. Predictors may include time-static variables, repeated across rows, and time-dependent covariates that change from row to row.

RHF estimates hazards pathwise on the time grid `time.interest`. For a supplied predictable covariate path, each tree routes the current covariate state through its partition over time and returns the corresponding terminal-node hazard. Averaging across trees yields a case-specific hazard trajectory. A fitted RHF object stores in-sample ensemble estimates of the hazard and cumulative hazard on `time.interest`. Predictions for new data, or from restored fitted objects, are obtained with `predict.rhf()`.

The default hazard aggregation uses an adaptive trimming protocol: RHF evaluates a grid of candidate trimming fractions and retains the candidate selected by OOB risk. Setting `adaptive = FALSE` uses the fixed trimming fraction `coe.trim = 0.05` instead. This can be useful when another tuning procedure should hold the hazard aggregation protocol fixed; however the value 0.05 is arbitrary and is not guaranteed to give good peformance in all settings.

Predictable covariate paths

For counting-process input, the start–stop rows must represent a predictable covariate process. That is, the covariate values used to evaluate the hazard at a given time must be determined by information available before that time. Measurements or treatments recorded later must not be used to construct covariates for an earlier prediction time. `rhf()` assumes that this no-lookahead convention has already been encoded in the input data. For ordinary `Surv(time, event)` input, each subject's supplied predictors are treated as time-static over the interval from zero to the observed time.

Supported response formats

RHF accepts either of the following response forms.

`Surv(time, event)` Ordinary terminal right-censored data with one row per subject. The `time` and `event` arguments must each be column names in `data`. Times must be non-negative, and `event` must be coded `0` for censoring and `1` for an event. RHF creates a private subject identifier and internally represents each row as one counting-process interval `(0, time]`.

`Surv(id, start, stop, event)` Counting-process data for time-dependent covariates. The subject identifier `id` is repeated across all rows for that subject. Each row must satisfy `stop > start`. Intervals may be contiguous or separated by gaps, but they may not overlap within a subject. The row-level `event` increment must be coded `0/1`; terminal-event data may contain at most one event per subject, on that subject's final row.

Times are supplied in their original non-negative units. RHF maps them to its internal working scale during data processing and returns `time.interest` and time-based predictions in the original units.

Tree construction

Each tree is grown by a best-first search guided by reduction in empirical hazard risk. Candidate splits are evaluated using a nonparametric hazard likelihood for predictable covariate processes. For computational efficiency, tree induction uses a working model that assigns a time-constant hazard to each current node. Under this restriction, candidate splits can be scored using node-level at-risk exposure and event totals.

The node-constant model is used only to choose the partition. Once the partition is fixed, RHF removes this restriction and estimates a time-varying step-function hazard in each terminal node on `time.interest`. Both static and time-dependent covariates are eligible for splitting. The option `xvar.wt` controls how often individual variables are considered, and can be used, for example, to give additional weight to time-dependent covariates.

In-sample fitted trajectories

For the training data, RHF assembles case-specific fitted curves on `time.interest` using an active-record rule. Each start–stop row supplies the routing covariate state for the portion of the grid on which that row is active. Rows belonging to the same individual are then stitched together to form the fitted pathwise hazard trajectory. At a grid point t, the returned case-specific hazard is defined only when t lies in at least one of the individual's supplied $(start, stop]$ intervals. It is returned as `NA` in gaps and after the final supplied stop time; covariate values are not carried forward outside the observed path.

The corresponding cumulative hazard is calculated using the exact overlap between the working time-grid intervals and the supplied start–stop path. It therefore remains flat through gaps and after follow-up. An observed portion of a final grid interval still contributes to cumulative hazard even when the pointwise hazard at that grid interval's right endpoint lies outside the supplied path. These output conventions do not alter the likelihood contribution of the training data.

Data conversion

Manual conversion is not required for ordinary right-censored data: `rhf(Surv(time, event) ~ ., data)` performs the one-interval conversion internally. The helper `convert.counting()` remains available when an explicit counting-process data frame is desired.

Value

An object of class "rhf" containing the fitted forest and related results. Components include (among others) the working time grid time.interest and case-specific ensemble estimates of the hazard and cumulative hazard. When bootstrapping is used, these are typically returned as hazard.oob/chf.oob (out-of-bag) and hazard.inbag/chf.inbag (in-bag). The component input.info records whether the forest was grown from ordinary right-censored or counting-process input and retains the response names needed for prediction. Use predict.rhf to obtain estimates for new data.

Author(s)

Hemant Ishwaran and Udaya B. Kogalur

References

Ishwaran H. and 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.

Lee, D.K. and Chen N. and Ishwaran H (2021). Boosted nonparametric hazards with time-dependent covariates. *Annals of Statistics*, 49: 2101-2128.

Ishwaran H. (2025). Multivariate Statistics: Classical Foundations and Modern Machine Learning. Chapman and Hall.

Ishwaran H., Kogalur U.B., Hsich E.M. and Lee D.K. (2026). Random hazard forests.

See Also

auct.rhf, importance.rhf, plot.rhf, predict.rhf, smoothed.hazard.rhf, tune.treesize.rhf

Examples

```
## ------------------------------------------------------------
##  time-static pbc: parameters set for fast CRAN run
## ------------------------------------------------------------

## load the data
data(pbc, package = "randomForestSRC")
pbc.raw <- na.omit(pbc)

## ordinary right-censored data can be supplied directly
f <- Surv(days, status) ~ .
print((o <- rhf(f, pbc.raw, ntree = 3)))

## smooth hazard estimator for specific cases
plot(o, idx=c(1,5,10))
plot(o, idx=c(1,5,10), hazard.only=TRUE)
plot(o, idx=c(1,5,10), hazard.only=TRUE, lwd=0)
plot(o, idx=1:10, lwd=0, hazard.only=TRUE, legend.show=FALSE)
```

```
## ------------------------------------------------------------
##  time-static pbc: compare RHF to RSF
## ------------------------------------------------------------

data(pbc, package = "randomForestSRC")
pbc.raw <- na.omit(pbc)

## first we run rhf directly on the ordinary survival data
o.rhf <- rhf(Surv(days, status) ~ ., pbc.raw)
h.rhf <- o.rhf$hazard.oob
time <- o.rhf$time.interest
H.rhf <- o.rhf$chf.oob
S.rhf <- exp(-H.rhf)

## next we run rsf, using the same cases and time.interest grid
o.rsf <- randomForestSRC::rfsrc(
  Surv(days, status) ~ ., pbc.raw, ntime = time, use.uno = FALSE
)
H.rsf <- o.rsf$chf.oob
S.rsf <- o.rsf$survival.oob

## compare C-stat of RHF to RSF
cat("RHF C-index:",
 randomForestSRC::get.cindex(o.rhf$yvar[,2],o.rhf$yvar[,3],rowSums(H.rhf, na.rm = TRUE)),"\n")
cat("RSF C-index:",
 randomForestSRC::get.cindex(o.rsf$yvar[,1],o.rsf$yvar[,2],rowSums(H.rsf, na.rm = TRUE)),"\n")
cat("RSF C-index (from rsf object):",tail(o.rsf$err.rate, 1),"\n")


## graphical parameters
oldpar <- par(mfrow=c(2,2))

## plot survival results
matplot(time,t(S.rhf),type="l",col=gray(.6),ylab="RHF survival")
matplot(time,t(S.rsf),type="l",col=gray(.6),ylab="RSF survival")
boxplot(t(S.rhf-S.rsf),ylab="survival diff.",xlab="cases",xaxt="n",outline=FALSE,range=1e-10)
abline(h=0,lwd=3,col=2)

## plot hazard direct approach - see plot.rhf() for user friendly function
finite <- is.finite(h.rhf)
matplot(time,t(h.rhf),ylab="RHF hazard",
        pch=16,type="n",
        ylim=c(0,quantile(h.rhf[finite],.99)))
nO <- lapply(which(rowSums(finite) >= 2L), function(i) {
  lines(supsmu(time[finite[i,]],h.rhf[i,finite[i,]],bass=1),
        type="b",pch=16,col=gray(.6),cex=.5)
})



par(oldpar)
```

```
## ------------------------------------------------------------
##  TDC illustration (using built in hazard simulation function)
## ------------------------------------------------------------

d1 <- hazard.simulation(1)$dta
d2 <- hazard.simulation(2)$dta
d3 <- hazard.simulation(3)$dta
f <- "Surv(id, start, stop, event) ~ ."

o1 <- rhf(f, d1)
o2 <- rhf(f, d2)
o3 <- rhf(f, d3)

plot(o1,idx=4)
plot(o2,idx=1:3)
plot(o3,idx=2)


## ------------------------------------------------------------
##  peakVO2: demonstrates time-varying auc
## ------------------------------------------------------------

data(peakVO2, package = "randomForestSRC")
f <- Surv(ttodead, died) ~ .

## build the forest directly from ordinary right-censored data
o1 <- rhf(f, peakVO2, treesize=10)
o2 <- rhf(f, peakVO2, treesize=30)

## old-school C-stat
cstat.f <- function(o) {randomForestSRC::get.cindex(o$yvar[, 2],
  o$yvar[, 3],rowSums(o$chf.oob, na.rm =TRUE))}
cat("C-stat treesize 10:", cstat.f(o1), "\n")
cat("C-stat treesize 30:", cstat.f(o2), "\n")

## acquire auc-t
a1.chf <- auct.rhf(o1) ## same as auct.rhf(o1, marker = "chf")
a1.haz <- auct.rhf(o1, marker = "haz")
a2.chf <- auct.rhf(o2) ## same as auct.rhf(o2, marker = "chf")
a2.haz <- auct.rhf(o2, marker = "haz")

## print auc-t
print(a1.chf)
print(a2.chf)
print(a1.haz)
print(a2.haz)

## plot auc-t
oldpar <- par(mfrow=c(2,2))
plot(a1.chf, main="treesize 1")
plot(a1.haz, main="treesize 1")
```

```
plot(a2.chf, main="treesize 15")
plot(a2.haz, main="treesize 15")
par(oldpar)

## ------------------------------------------------------------
##  more detailed example of time-varying performance
## ------------------------------------------------------------

d <- hazard.simulation(1)$dta
f <- "Surv(id, start, stop, event) ~ ."
treesize <- c(10, 30, 100)

oldpar <- par(mfrow=c(3,2))
perf <- lapply(treesize, function(tz) {
  o <- rhf(f, d, treesize=tz)
  print(o)
  rO <- list(chf = auct.rhf(o, marker="chf"),
             haz = auct.rhf(o, marker="haz"))
  plot(rO$chf, main=paste("chf marker, treesize=", tz))
  plot(rO$haz, main=paste("hazard marker, treesize=", tz))
  rO
})
par(oldpar)

## ------------------------------------------------------------
##  example illustrating tuning the tree size
##  using function 'rhf.tune.treesize'
## ------------------------------------------------------------

d <- hazard.simulation(1)$dta
f <- "Surv(id, start, stop, event) ~ ."

tune <- tune.rhf(f, d)  ## same as rhf.tune.treesize(f,d)
oldpar <- par(mfrow=c(1,1))
plot(tune)
par(oldpar)

## ------------------------------------------------------------
##  example illustrating guided feature selection
##  using built in 'xvar.wt.rhf' helper
## ------------------------------------------------------------

d <- hazard.simulation(1)$dta
f <- "Surv(id, start, stop, event) ~ ."

o <- rhf(f,d)
o.gfs <- rhf(f,d, xvar.wt = xvar.wt.rhf(f, d))

print(o)
print(o.gfs)
```

rhf.news *Show the NEWS file*

Description

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

Usage

```
rhf.news(...)
```

Arguments

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

Value

None.

Author(s)

Hemant Ishwaran and Udaya B. Kogalur

smoothed.hazard.rhf *Smoothed hazard and cumulative hazard estimates from a random hazard forest*

Description

Compute smoothed case-specific hazard curves (and corresponding cumulative hazards) from a Random Hazard Forest (RHF) object. The returned object is a lightweight list designed to be directly usable by plot.rhf.

Usage

```
smoothed.hazard.rhf(
  o,
  method = c("median.loess", "loess"),
  oob = TRUE,
  span = 0.35,
  degree = 1,
  family = NULL,
  eps = 1e-12,
  suppress.warnings = TRUE,
  trace = FALSE
)
```

Arguments

o	A fitted RHF object or predict object.
method	Smoothing strategy. See **Details** for definitions of "loess" and "median.loess".
oob	By default out-of-bag (OOB) values are returned for RHF objects, but inbag values can be requested by setting this option to FALSE. Does not apply to RHF predict objects.
span	Loess span passed to loess.
degree	Loess degree passed to loess.
family	Loess family passed to loess. Must be "gaussian" or "symmetric". If NULL, defaults to "gaussian" for method="median.loess" and "symmetric" for method="loess".
eps	Small positive constant added before log-transforming hazards: log(h + eps).
suppress.warnings	If TRUE, warnings arising from loess() fitting/prediction are suppressed.
trace	If TRUE, print progress messages.

Details

Hazard curves are computed/smoothed on the grid o$time.interest. Smoothing is performed on the log-hazard scale: log(hazard + eps), and then back-transformed to the hazard scale.

method="loess" Applies loess smoothing to an already-constructed hazard matrix stored in o (e.g. o$hazard.oob and/or o$hazard.inbag), independently for each subject.

method="median.loess" For each subject and time bin, computes the median across trees of the terminal-node (leaf) log-hazard log(h + eps) (using a native C routine), then applies loess smoothing to the resulting median log-hazard curve and back-transforms. This method requires the fitted object to retain tree-specific hazard and membership structures (e.g., o$forest$t.hazard and related arrays).

Value

A list containing:

- id, ensemble.id, yvar, time.interest copied from o.
- Depending on target:
  - hazard.oob, chf.oob (out-of-bag), matrices of dimension nSubj x length(time.interest).
  - hazard.inbag, chf.inbag (in-bag), matrices of dimension nSubj x length(time.interest).
  - hazard.test, chf.test (test), matrices of dimension nSubjTest x length(time.interest).

See Also

plot.rhf, predict.rhf, rhf

Examples

```
## ------------------------------------------------------------
## canonical example using synthetic data
## includes both train/test scenarios
## ------------------------------------------------------------

simID <- 1
trn <- hazard.simulation(simID)$dta
tst <- hazard.simulation(simID)$dta
f <- "Surv(id, start, stop, event) ~ ."

## training/testing
o <- rhf(f, trn)
p <- predict(o, tst)

## default: median(log leaf hazard) + loess
so <- smoothed.hazard(o)
sp <- smoothed.hazard(p)

## the returned object can be passed directly to plot.rhf()
oldpar <- par(mfrow=c(1,1))
plot.rhf(so)
plot.rhf(sp)
par(oldpar)

## ------------------------------------------------------------
##  peak vo2
## ------------------------------------------------------------

data(peakVO2, package = "randomForestSRC")
d <- convert.counting(Surv(ttodead, died)~., peakVO2)
f <- "Surv(id, start, stop, event) ~ ."

## training
o <- rhf(f, d)

oldpar <- par(mfrow=c(1,1))


## median loess
s0 <- smoothed.hazard(o)
ids <- o$ensemble.id[1:3]
plot.rhf(s0, idx = ids)

## loess smoothing
s1 <- smoothed.hazard(o, method = "loess")
plot.rhf(s1, idx = ids)

par(oldpar)
```

`tune.treesize.rhf` *Tune tree size for Random Hazard Forests*

Description

Tune the `treesize` for a Random Hazard Forest using either out-of-bag (OOB) empirical risk or an OOB integrated AUC computed by `auct.rhf`.

Usage

```
tune.treesize.rhf(
  formula,
  data,
  ntree = 500,
  nsplit = 10,
  nodesize = NULL,
  ntime = 50,
  min.events.per.gap = 10,
  adaptive = TRUE,
  perf = c("risk", "iAUC"),
  auct.args = NULL,
  lower = 5L,
  upper = NULL,
  C = 3,
  method = c("golden", "bisect"),
  max.evals = 20L,
  bracket.tol = 2L,
  seed = NULL,
  verbose = TRUE,
  forest = TRUE,
  ...
)


tune.iAUC.rhf(
  formula,
  data,
  auct.args = NULL,
  adaptive = TRUE,
  ...
)

## S3 method for class 'tune.treesize.rhf'
plot(x, ylab = NULL, main = NULL,
    se.band = TRUE, se.mult = 1, ylim = NULL, ...)
```

Arguments

formula	Survival formula for rhf. The response may be ordinary right-censored data, Surv(time, event), or counting-process data, Surv(id, start, stop, event).
data	Data frame containing the variables in formula. Ordinary right-censored input has one row per subject; counting-process input may have multiple rows per subject.
ntree	Number of trees grown at each candidate treesize.
nsplit	Number of random split points tried per variable.
nodesize	Minimum number of events in a terminal node; passed to rhf.
ntime	Controls the working time grid used by every candidate rhf fit and by the final refit. It has the same meaning and default as in rhf. When upper = NULL, it is also used when processing the event information for the automatic tuning range.
min.events.per.gap	Minimum number of observed events required in each working-time interval when ntime is specified as an integer. The value is passed to every candidate rhf fit and to the final refit. When upper = NULL, it also determines the package-default tree size and event-support limit used to construct the automatic upper bound.
adaptive	Logical flag passed to rhf. When TRUE, each candidate forest uses the default adaptive trimming protocol. When FALSE, each candidate forest uses the fixed trimming fraction 0.05, so the tuning criterion varies with treesize while holding this hazard aggregation setting fixed.
perf	Tuning criterion. "risk" minimizes mean OOB risk; "iAUC" maximizes integrated AUC obtained using auct.rhf.
auct.args	Optional list of arguments passed to auct.rhf when perf = "iAUC".
lower	Smallest treesize considered.
upper	Largest treesize considered. If NULL, the upper bound is computed from the event information used by rhf. Specifically, it is the smaller of C times the package-default tree size and the event-supported limit floor(ndead / min.events.per.gap), where ndead is the number of events remaining after RHF data processing.
C	Multiplier applied to the package-default tree size when computing the automatic upper bound. The resulting value is capped by the event-supported limit.
method	Discrete search strategy: "golden" (golden-section search) or "bisect" (bisection-style search).
max.evals	Maximum number of distinct treesize values evaluated by the search.
bracket.tol	Tolerance for the bracketing interval width used by the search algorithms.
seed	Optional random seed for reproducible comparisons across tree sizes.
verbose	Logical; if TRUE, print progress messages when evaluating new treesize values.
forest	Logical; if TRUE, return the RHF object at the best treesize.
...	Additional arguments passed to rhf.
x, ylab, main, se.band, se.mult, ylim	Arguments to customize plot.

Details

The alias `tune.rhf` is provided for convenience.

Repeatedly fits a hazard forest with different values of `treesize` and selects the value that optimizes the chosen criterion `perf`. Ordinary `Surv(time, event)` input is converted internally to the same one-interval counting-process representation used by `rhf`. The tuning preflight, automatic upper bound, all candidate forests, and the final refit therefore use the same effective subjects and event information. The returned forest retains the original right-censored response names and can be used directly with ordinary survival test data through `predict.rhf()`.

When `perf = "risk"`, the criterion is the mean OOB risk. When `perf = "iAUC"`, `auct.rhf` is applied to each fit and the criterion is `1 - iAUC.uno`. `tune.iAUC` is a wrapper around `tune.rhf` with `perf = "iAUC"`, using default `auct.rhf` settings when `auct.args = NULL`.

Because the default RHF fit adaptively selects the trimming fraction by OOB risk, tuning by OOB risk can otherwise combine the choice of `treesize` with adaptive trimming. Use `adaptive = FALSE` for a controlled tree-size search that fixes the trimming fraction at `coe.trim = 0.05`.

Value

An object of class `"tune.treesize.rhf"` with components

- `best.size`: optimal `treesize`;
- `best.err`: minimal criterion value (mean OOB risk if `perf = "risk"`, or `1 - iAUC.uno` if `perf = "iAUC"`);
- `path`: data frame with evaluated `treesize` values and corresponding criterion values (and an `iAUC` column when `perf = "iAUC"`);
- `n.subjects`: number of subjects remaining after RHF input processing;
- `adaptive`: logical value indicating whether adaptive trimming was used during tuning;
- `forest`: RHF fit at `best.size` when `forest = TRUE`.

See Also

`rhf`, `auct.rhf`

Examples

```
## ------------------------------------------------------------
## Example: tuning treesize by OOB empirical risk
## ------------------------------------------------------------

data(pbc, package = "randomForestSRC")
pbc.raw <- na.omit(pbc)
set.seed(7)
trn <- sample(
  seq_len(nrow(pbc.raw)),
  size = floor(0.75 * nrow(pbc.raw)),
  replace = FALSE
)

d <- pbc.raw[trn, , drop = FALSE]
```

```
tst <- pbc.raw[-trn, , drop = FALSE]
f <- Surv(days, status) ~ .

## tune.rhf is an alias for tune.treesize.rhf; no manual conversion is needed
tuneRisk <- tune.rhf(f, d)  ## same as tune.treesize.rhf(f, d)

## For a controlled tree-size search, keep the trimming fraction fixed.
tuneRisk.fixed <- tune.rhf(f, d, adaptive = FALSE)

oldpar <- par(mfrow = c(1, 1))
plot(tuneRisk)
par(oldpar)

tuneRisk$best.size    # treesize minimizing mean OOB risk
tuneRisk$best.err

## Access the tuned forest and refit AUC diagnostics if desired
best.forest <- tuneRisk$forest
a.risk <- auct.rhf(best.forest)
print(a.risk$iAUC.uno)

## the tuned forest accepts the ordinary PBC test data directly
print(predict(best.forest, tst))


## ------------------------------------------------------------
## Example: tuning treesize by iAUC
## ------------------------------------------------------------

set.seed(7)
f <- "Surv(id, start, stop, event) ~ ."
d <- hazard.simulation(1, n=100)$dta
tuneiAUC <- tune.iAUC(f, d)

oldpar <- par(mfrow = c(1, 1))
plot(tuneiAUC)
par(oldpar)

print(tuneiAUC$best.size)
print(1 - tuneiAUC$best.err)

best.forest.iauc <- tuneiAUC$forest
a.iauc <- auct.rhf(best.forest.iauc)
print(1 - a.iauc$iAUC.uno)


## ------------------------------------------------------------
## Example: tuning treesize by iAUC with bootstrap s.e.
## ------------------------------------------------------------

set.seed(7)
f <- "Surv(id, start, stop, event) ~ ."
d <- hazard.simulation(1, n=100)$dta
```

```
tuneiAUC.boot <- tune.iAUC(f, d, auct.args=list(bootstrap.rep=25))
oldpar <- par(mfrow = c(1, 1))
plot(tuneiAUC.boot)
par(oldpar)

## ------------------------------------------------------------
## Example: tuning by hazard-based incident AUC
## increase min.cases for stability
## ------------------------------------------------------------

set.seed(7)
f <- "Surv(id, start, stop, event) ~ ."
d <- hazard.simulation(1, n=100)$dta
tuneHazInc <- tune.iAUC(
  f, d,
  auct.args = list(
    min.cases    = 5,
    marker       = "hazard",
    method       = "incident"
  )
)

oldpar <- par(mfrow = c(1, 1))
plot(tuneHazInc)
par(oldpar)

print(tuneHazInc$best.size)
print(tuneHazInc$best.err)

## Confirm the tuned incident AUC on the selected forest
best.forest.haz <- tuneHazInc$forest
aHazInc <- auct.rhf(
  best.forest.haz,
  min.cases = 5,
  marker  = "hazard",
  method  = "incident"
)
print(aHazInc)
print(1 - aHazInc$iAUC.uno)
```

Index

∗ **AUC**
auct.rhf, 2
∗ **ROC**
auct.rhf, 2
∗ **documentation**
rhf.news, 30
∗ **models**
smoothed.hazard.rhf, 30
∗ **plot**
plot.rhf, 15
∗ **predict rhf**
predict.rhf, 18
∗ **print**
print.rhf, 21
∗ **rhf importance**
importance.rhf, 7
∗ **rhf**
plot.rhf, 15
rhf, 22
∗ **survival**
auct.rhf, 2
smoothed.hazard.rhf, 30
tune.treesize.rhf, 33
∗ **tuning**
tune.treesize.rhf, 33

as.data.frame.importance.rhf
*(*importance.rhf*)*, 7
auct *(*auct.rhf*)*, 2
auct.rhf, 2, *26*, *33–35*

barplot.importance *(*importance.rhf*)*, 7

dotmatrix.importance *(*importance.rhf*)*, 7

hazard.simulation, *16*

importance.rhf, 7, *26*

legend, *16*
loess, *31*

plot, *16*
plot.auct.rhf *(*auct.rhf*)*, 2
plot.importance.rhf *(*importance.rhf*)*, 7
plot.rhf, 15, *26*, *30, 31*
plot.tune.treesize.rhf
*(*tune.treesize.rhf*)*, 33
predict.rhf, *5*, *16*, 18, *26*, *31*
print.auct.rhf *(*auct.rhf*)*, 2
print.importance.rhf *(*importance.rhf*)*, 7
print.rhf, 21
print.vimp.rhf *(*print.rhf*)*, 21

rhf, *5*, *14–16*, *20*, 22, *31*, *34, 35*
rhf.news, 30

smoothed.hazard *(*smoothed.hazard.rhf*)*,
30
smoothed.hazard.rhf, *16*, *26*, 30
supsmu, *16*

tune.iAUC *(*tune.treesize.rhf*)*, 33
tune.rhf *(*tune.treesize.rhf*)*, 33
tune.treesize.rhf, *26*, 33

varpro.cache *(*importance.rhf*)*, 7