

Package ‘ggRandomForests’

May 8, 2026

Type Package

Title Visually Exploring Random Forests

Version 2.7.2

Date 2026-04-27

License MIT + file LICENSE

Encoding UTF-8

Language en-US

URL <https://github.com/ehrlinger/ggRandomForests>,
<https://ehrlinger.github.io/ggRandomForests/>

BugReports <https://github.com/ehrlinger/ggRandomForests/issues>

Description Graphic elements for exploring Random Forests using the 'randomForest' or 'randomForestSRC' package for survival, regression and classification forests and 'ggplot2' package plotting. Implements visualisations of the methods described in Breiman (2001) <[doi:10.1023/A:1010933404324](https://doi.org/10.1023/A:1010933404324)> and Ishwaran, Kogalur, Blackstone, and Lauer (2008) <[doi:10.1214/08-AOAS169](https://doi.org/10.1214/08-AOAS169)>.

Depends R (>= 4.4.0), randomForestSRC (>= 3.4.0), randomForest

Imports survival, parallel, tidyr, dplyr, ggplot2, stringr

Suggests testthat, bookdown, RColorBrewer, MASS, lintr, covr, vdiff, datasets, rmarkdown, quarto, pkgdown, pkgload, knitr, plotly

VignetteBuilder quarto

RoxygenNote 7.3.3

NeedsCompilation no

Author John Ehrlinger [aut, cre]

Maintainer John Ehrlinger <john.ehrlinger@gmail.com>

Repository CRAN

Date/Publication 2026-05-02 09:30:02 UTC

Contents

ggRandomForests-package	2
calc_auc	4
calc_roc.rfsrc	5
gg_error	6
gg_partial	10
gg_partialpro	11
gg_partial_rfsrc	12
gg_rfsrc.rfsrc	14
gg_roc.rfsrc	17
gg_survival	18
gg_variable	20
gg_vimp	23
kaplan	26
nelson	27
plot.gg_error	28
plot.gg_partial	32
plot.gg_partialpro	33
plot.gg_partial_rfsrc	34
plot.gg_rfsrc	36
plot.gg_roc	39
plot.gg_survival	41
plot.gg_variable	42
plot.gg_vimp	45
quantile_pts	46
surv_partial.rfsrc	47
varpro_feature_names	49

Index	52
--------------	-----------

ggRandomForests-package

ggRandomForests: Visually Exploring Random Forests

Description

ggRandomForests is a utility package for randomForestSRC (Ishwaran and Kogalur) for survival, regression and classification forests and uses the ggplot2 (Wickham 2009) package for plotting results. ggRandomForests is structured to extract data objects from the random forest and provides S3 functions for printing and plotting these objects. Requires randomForestSRC >= 3.4.0.

The randomForestSRC package provides a unified treatment of Breiman's (2001) random forests for a variety of data settings. Regression and classification forests are grown when the response is numeric or categorical (factor) while survival and competing risk forests (Ishwaran et al. 2008, 2012) are grown for right-censored survival data.

Many of the figures created by the ggRandomForests package are also available directly from within the randomForestSRC package. However, ggRandomForests offers the following advantages:

- Separation of data and figures: `ggRandomForest` contains functions that operate on either the `rfsrc` forest object directly, or on the output from `randomForestSRC` post processing functions (i.e. `plot.variable`) to generate intermediate `ggRandomForests` data objects. S3 functions are provide to further process these objects and plot results using the `ggplot2` graphics package. Alternatively, users can use these data objects for additional custom plotting or analysis operations.
- Each data object/figure is a single, self contained object. This allows simple modification and manipulation of the data or `ggplot2` objects to meet users specific needs and requirements.
- The use of `ggplot2` for plotting. We chose to use the `ggplot2` package for our figures to allow users flexibility in modifying the figures to their liking. Each S3 plot function returns either a single `ggplot2` object, or a list of `ggplot2` objects, allowing users to use additional `ggplot2` functions or themes to modify and customize the figures to their liking.

The `ggRandomForests` package contains the following data functions:

- `gg_rfsrc`: `randomForest[SRC]` predictions.
- `gg_error`: `randomForest[SRC]` convergence rate based on the OOB error rate.
- `gg_roc`: ROC curves for `randomForest` classification models.
- `gg_vimp`: Variable Importance ranking for variable selection. (Ishwaran et.al. 2010).
- `gg_variable`: Marginal variable dependence.
- `gg_survival`: Kaplan-Meier/Nelson-Aalen hazard analysis.

Each of these data functions has an associated S3 plot function that returns `ggplot2` objects, either individually or as a list, which can be further customized using standard `ggplot2` commands.

References

- Breiman, L. (2001). Random forests, *Machine Learning*, 45:5-32.
- Ishwaran H. and Kogalur U.B. `randomForestSRC`: Random Forests for Survival, Regression and Classification. R package version $\geq 3.4.0$. <https://cran.r-project.org/package=randomForestSRC>
- 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. *Ann. Appl. Statist.* 2(3), 841–860.
- Ishwaran, H., U. B. Kogalur, E. Z. Gorodeski, A. J. Minn, and M. S. Lauer (2010). High-dimensional variable selection for survival data. *J. Amer. Statist. Assoc.* 105, 205-217.
- Ishwaran, H. (2007). Variable importance in binary regression trees and forests. *Electronic J. Statist.*, 1, 519-537.
- Wickham, H. `ggplot2`: elegant graphics for data analysis. Springer New York, 2009.

calc_auc	<i>Area Under the ROC Curve calculator</i>
----------	--

Description

Area Under the ROC Curve calculator

Usage

```
calc_auc(x)
```

Arguments

x [gg_roc](#) object

Details

calc_auc uses the trapezoidal rule to calculate the area under the ROC curve. This is a helper function for the [gg_roc](#) functions.

Value

AUC. 50% is random guessing, higher is better.

See Also

[calc_roc](#) [gg_roc](#)

[plot.gg_roc](#)

Examples

```
##  
## Taken from the gg_roc example  
rfsrc_iris <- rfsrc(Species ~ ., data = iris)  
  
gg_dta <- gg_roc(rfsrc_iris, which_outcome = 1)  
  
calc_auc(gg_dta)  
  
gg_dta <- gg_roc(rfsrc_iris, which_outcome = 2)  
  
calc_auc(gg_dta)  
  
## randomForest tests  
rf_iris <- randomForest::randomForest(Species ~ ., data = iris)  
gg_dta <- gg_roc(rfsrc_iris, which_outcome = 2)  
  
calc_auc(gg_dta)
```

calc_roc.rfsrc	<i>Receiver Operator Characteristic calculator</i>
----------------	--

Description

Receiver Operator Characteristic calculator

Usage

```
## S3 method for class 'rfsrc'
calc_roc(object, dta, which_outcome = "all", oob = TRUE, ...)
```

Arguments

object	A fitted rfsrc , predict.rfsrc , or randomForest classification object containing predicted class probabilities.
dta	A factor (or coercible to factor) of the true observed class labels, one per observation. Typically <code>object\$yvar</code> for <code>rfsrc</code> or <code>object\$y</code> for <code>randomForest</code> .
which_outcome	Integer index of the class for which the ROC curve is computed (e.g. 1 for the first class, 2 for the second). Use "all" to request all classes (currently falls back to class 1 with a warning).
oob	Logical; if TRUE (default for <code>rfsrc</code>) use OOB predicted probabilities. Forced to FALSE for <code>randomForest</code> objects.
...	Extra arguments passed to helper functions (currently unused).

Details

For a `randomForestSRC` prediction and the actual response value, calculate the specificity (1-False Positive Rate) and sensitivity (True Positive Rate) of a predictor.

This is a helper function for the [gg_roc](#) functions, and not intended for use by the end user.

Value

A `gg_roc` `data.frame` with columns `sens` (sensitivity), `spec` (specificity), and `pct` (the probability threshold), with one row per unique prediction value. Suitable for passing to [calc_auc](#) or [plot.gg_roc](#).

See Also

[calc_auc](#) [gg_roc](#)
[plot.gg_roc](#)

Examples

```
## Taken from the gg_roc example
rfsrc_iris <- rfsrc(Species ~ ., data = iris)

gg_dta <- calc_roc(rfsrc_iris, rfsrc_iris$yvar,
  which_outcome = 1, oob = TRUE
)
gg_dta <- calc_roc(rfsrc_iris, rfsrc_iris$yvar,
  which_outcome = 1, oob = FALSE
)

rf_iris <- randomForest(Species ~ ., data = iris)
gg_dta <- calc_roc(rf_iris, rf_iris$yvar,
  which_outcome = 1
)
gg_dta <- calc_roc(rf_iris, rf_iris$yvar,
  which_outcome = 2
)
```

gg_error

Random forest error trajectory data object

Description

Extract the cumulative out-of-bag (OOB) or in-bag training error rate from `randomForestSRC` and `randomForest` fits as a function of the number of grown trees.

Usage

```
gg_error(object, ...)
```

Arguments

<code>object</code>	A fitted <code>rfsrc</code> or <code>randomForest</code> object.
<code>...</code>	Optional arguments passed to the methods. Set <code>training = TRUE</code> to append the in-bag error trajectory when supported.

Details

For `randomForestSRC` objects the function reshapes the `rfsrc$err.rate` matrix and annotates it with the tree index required by `plot.gg_error`. When supplied a `randomForest` object, the method inspects either the `$mse` or `$err.rate` component and, when `training = TRUE` is requested, reconstructs the original training set via the model call to compute an in-bag error curve using per-tree predictions. Training curves are only available when the forest was stored (`keep.forest = TRUE`) and the original data can be recovered.

Value

A `gg_error` data.frame containing at least the cumulative OOB error columns and an `n tree` counter. When `training = TRUE` is honored an additional `train` column is included.

References

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

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

Ishwaran H. and Kogalur U.B. `randomForestSRC`: Random Forests for Survival, Regression and Classification. R package version $\geq 3.4.0$. <https://cran.r-project.org/package=randomForestSRC>

See Also

[plot.gg_error](#), [gg_vimp](#), [gg_variable](#), [rfsrc](#), [randomForest](#), [plot.rfsrc](#)

Examples

```
## Examples from RFSRC package...
## -----
## classification example
## -----
## ----- iris data
## You can build a randomForest
rfsrc_iris <- rfsrc(Species ~ ., data = iris, tree.err = TRUE)

# Get a data.frame containing error rates
gg_dta <- gg_error(rfsrc_iris)

# Plot the gg_error object
plot(gg_dta)

## RandomForest example
rf_iris <- randomForest::randomForest(Species ~ .,
  data = iris,
  tree.err = TRUE,
)
gg_dta <- gg_error(rf_iris)
plot(gg_dta)

gg_dta <- gg_error(rf_iris, training = TRUE)
plot(gg_dta)
## -----
## Regression example
## -----

## ----- airq data
rfsrc_airq <- rfsrc(Ozone ~ .,
  data = airquality,
  na.action = "na.impute", tree.err = TRUE,
)
```

```

# Get a data.frame containing error rates
gg_dta <- gg_error(rfsrc_airq)

# Plot the gg_error object
plot(gg_dta)

## ----- Boston data
data(Boston, package = "MASS")
Boston$chas <- as.logical(Boston$chas)
rfsrc_boston <- rfsrc(medv ~ .,
  data = Boston,
  forest = TRUE,
  importance = TRUE,
  tree.err = TRUE,
  save.memory = TRUE
)

# Get a data.frame containing error rates
gg_dta <- gg_error(rfsrc_boston)

# Plot the gg_error object
plot(gg_dta)

## ----- mtcars data
rfsrc_mtcars <- rfsrc(mpg ~ ., data = mtcars, tree.err = TRUE)

# Get a data.frame containing error rates
gg_dta <- gg_error(rfsrc_mtcars)

# Plot the gg_error object
plot(gg_dta)

## -----
## Survival example
## -----
## ----- veteran data
## randomized trial of two treatment regimens for lung cancer
data(veteran, package = "randomForestSRC")
rfsrc_veteran <- rfsrc(Surv(time, status) ~ ., data = veteran,
  tree.err = TRUE)

gg_dta <- gg_error(rfsrc_veteran)
plot(gg_dta)

## ----- pbc data
# Load a cached randomForestSRC object
# We need to create this dataset
data(pbc, package = "randomForestSRC",)
# For whatever reason, the age variable is in days... makes no sense to me
for (ind in seq_len(dim(pbc)[2])) {

```

```

if (!is.factor(pbc[, ind])) {
  if (length(unique(pbc[which(!is.na(pbc[, ind])), ind])) <= 2) {
    if (sum(range(pbc[, ind], na.rm = TRUE) == c(0, 1)) == 2) {
      pbc[, ind] <- as.logical(pbc[, ind])
    }
  }
} else {
  if (length(unique(pbc[which(!is.na(pbc[, ind])), ind])) <= 2) {
    if (sum(sort(unique(pbc[, ind])) == c(0, 1)) == 2) {
      pbc[, ind] <- as.logical(pbc[, ind])
    }
    if (sum(sort(unique(pbc[, ind])) == c(FALSE, TRUE)) == 2) {
      pbc[, ind] <- as.logical(pbc[, ind])
    }
  }
}
if (!is.logical(pbc[, ind]) &
    length(unique(pbc[which(!is.na(pbc[, ind])), ind])) <= 5) {
  pbc[, ind] <- factor(pbc[, ind])
}
}
#Convert age to years
pbc$age <- pbc$age / 364.24

pbc$years <- pbc$days / 364.24
pbc <- pbc[, -which(colnames(pbc) == "days")]
pbc$treatment <- as.numeric(pbc$treatment)
pbc$treatment[which(pbc$treatment == 1)] <- "DPCA"
pbc$treatment[which(pbc$treatment == 2)] <- "placebo"
pbc$treatment <- factor(pbc$treatment)
dta_train <- pbc[-which(is.na(pbc$treatment)), ]
# Create a test set from the remaining patients
pbc_test <- pbc[which(is.na(pbc$treatment)), ]

#####
# build the forest:
rfsrc_pbc <- randomForestSRC::rfsrc(
  Surv(years, status) ~ .,
  dta_train,
  nsplit = 10,
  na.action = "na.impute",
  tree.err = TRUE,
  forest = TRUE,
  importance = TRUE,
  save.memory = TRUE
)

gg_dta <- gg_error(rfsrc_pbc)
plot(gg_dta)

```

<code>gg_partial</code>	<i>Split partial dependence data into continuous or categorical datasets</i>
-------------------------	--

Description

Takes the list returned by `rfsrc::plot.variable(partial = TRUE)` and separates the variables into two data frames: one for continuous predictors and one for categorical (factor-like) predictors. The split is controlled by `cat_limit`: variables with more unique x-values than this threshold are treated as continuous; all others are categorical.

Usage

```
gg_partial(part_dta, nvars = NULL, cat_limit = 10, model = NULL)
```

Arguments

<code>part_dta</code>	partial plot data from <code>rfsrc::plot.variable</code>
<code>nvars</code>	how many of the partial plot variables to calculate
<code>cat_limit</code>	Categorical features are built when there are fewer than <code>cat_limit</code> unique feature values.
<code>model</code>	a label name applied to all features. Useful when combining multiple partial plot objects in figures.

Value

A named list with two elements:

continuous data.frame with columns `x`, `yhat`, `name` (and optionally `model`) for continuous variables

categorical data.frame with the same columns but with `x` as a factor, for low-cardinality / categorical variables

See Also

[gg_partial_rfsrc](#) [gg_partialpro](#)

Examples

```
## Build a small regression forest on the airquality dataset
set.seed(42)
airq <- na.omit(airquality)
rf <- rfsrc(Ozone ~ ., data = airq, ntree = 50)

## Compute partial dependence via plot.variable (show.plots = FALSE to
## suppress the base-graphics output - we only want the data)
pv <- randomForestSRC::plot.variable(rf, partial = TRUE,
                                     show.plots = FALSE)

## Split into continuous and categorical data frames
```

```

result <- gg_partial(pv)
head(result$continuous)

## Label this model for later comparison with a second forest
result_labelled <- gg_partial(pv, model = "airq_model")
unique(result_labelled$continuous$model)

```

gg_partialpro	<i>Split varpro partial dependence data into continuous or categorical datasets</i>
---------------	---

Description

Takes the list returned by `varpro::partialpro` and separates variables into two data frames: one for continuous predictors (parametric, non- parametric, and causal effect curves) and one for categorical predictors (one row per observation per category level).

Usage

```
gg_partialpro(part_dta, nvars = NULL, cat_limit = 10, model = NULL)
```

Arguments

<code>part_dta</code>	partial plot data from <code>varpro::partialpro</code> . Each element of the list must contain fields <code>xvirtual</code> , <code>xorg</code> , <code>yhat.par</code> , <code>yhat.nonpar</code> , and <code>yhat.causal</code> .
<code>nvars</code>	how many variables (list elements) to process. Defaults to all variables in <code>part_dta</code> .
<code>cat_limit</code>	Variables with <code>length(xvirtual) ≤ cat_limit</code> are treated as categorical. Default 10.
<code>model</code>	a label applied to all rows. Useful when combining results from multiple models in a single figure.

Details

The split is governed by `cat_limit`: a variable is treated as continuous when `length(xvirtual) > cat_limit`; otherwise it is treated as categorical and the per-category rows are stacked.

Value

A named list with two elements:

continuous data.frame with columns `variable`, `parametric`, `nonparametric`, `causal`, `name` (and optionally `model`)

categorical data.frame with the same columns but one row per observation per category level

See Also

[gg_partial varpro_feature_names](#)

Examples

```
## Construct mock varpro partialpro output:
## - "age": a continuous predictor (xvirtual has > 10 points)
## - "sex": a categorical predictor (xvirtual has 2 points)
set.seed(42)
n_obs <- 30 # number of observations (rows in yhat matrices)
n_pts <- 15 # number of evaluation points for continuous variables

mock_data <- list(
  age = list(
    # xvirtual: evaluation grid for the marginal effect curve
    xvirtual = seq(30, 80, length.out = n_pts),
    # xorg: original observed values (used only for categorical detection)
    xorg = sample(seq(30, 80, by = 5), n_obs, replace = TRUE),
    # yhat matrices: n_obs rows x n_pts columns (predictions at each grid pt)
    yhat.par = matrix(rnorm(n_obs * n_pts), nrow = n_obs),
    yhat.nonpar = matrix(rnorm(n_obs * n_pts), nrow = n_obs),
    yhat.causal = matrix(rnorm(n_obs * n_pts), nrow = n_obs)
  ),
  sex = list(
    # Two categories: the xvirtual grid has only 2 points
    xvirtual = c(0, 1),
    xorg = sample(c(0, 1), n_obs, replace = TRUE),
    # Two-column yhat matrices (one column per category)
    yhat.par = matrix(rnorm(n_obs * 2), nrow = n_obs),
    yhat.nonpar = matrix(rnorm(n_obs * 2), nrow = n_obs),
    yhat.causal = matrix(rnorm(n_obs * 2), nrow = n_obs)
  )
)

result <- gg_partialpro(mock_data)

## Continuous result: one row per evaluation grid point
head(result$continuous)

## Categorical result: n_obs rows per category level
head(result$categorical)
```

Description

Computes partial dependence for one or more predictors by calling `partial.rfsrc` internally, then splits the results into separate data frames for continuous and categorical variables. Unlike `gg_partial`, no separate `plot.variable` call is required — supply the fitted `rfsrc` object directly.

Usage

```
gg_partial_rfsrc(
  rf_model,
  xvar.names = NULL,
  xvar2.name = NULL,
  newx = NULL,
  partial.time = NULL,
  partial.type = c("surv", "chf", "mort"),
  cat_limit = 10,
  n_eval = 25
)
```

Arguments

<code>rf_model</code>	A fitted <code>rfsrc</code> object.
<code>xvar.names</code>	Character vector of predictor names for which partial dependence should be computed. Must be a subset of <code>rf_model\$xvar.names</code> .
<code>xvar2.name</code>	Optional single character name of a grouping variable in <code>newx</code> . When supplied, partial dependence is computed separately for each unique level of this variable and a <code>grp</code> column is appended.
<code>newx</code>	Optional data.frame of predictor values to evaluate partial effects at. Defaults to the training data stored in <code>rf_model\$xvar</code> . All column names must match <code>rf_model\$xvar.names</code> .
<code>partial.time</code>	Numeric vector of desired time points for survival forests (ignored for regression/classification). Values are automatically snapped to the nearest entry in <code>rf_model\$time.interest</code> — see the Survival forests section below. When <code>NULL</code> (default), three quartile points of <code>time.interest</code> are used.
<code>partial.type</code>	Character; type of predicted value for survival forests, passed through to <code>partial.rfsrc</code> . One of "surv" (default), "chf", or "mort". Ignored for non-survival forests. <code>partial.rfsrc()</code> requires a non-NULL value for survival families; supplying it here avoids a cryptic "argument is of length zero" error from the underlying C code.
<code>cat_limit</code>	Variables with fewer than <code>cat_limit</code> unique values in <code>newx</code> are treated as categorical; all others are continuous. Defaults to 10.
<code>n_eval</code>	Number of evaluation points for continuous variables. Instead of passing all observed values (which can be slow, especially for survival forests), continuous predictors are evaluated on a quantile grid of this many points. Categorical variables always use all unique levels. Defaults to 25.

Value

A named list with two elements:

continuous A data.frame with columns `x` (numeric), `yhat`, `name` (variable name), and optionally `grp` (the level of `xvar2.name`) and `time` (survival forests only) for all continuous predictors.

categorical A data.frame with the same columns but `x` kept as character, for low-cardinality predictors.

Survival forests and `partial.time`

`partial.rfsrc` requires that every value in `partial.time` be an exact member of the model's `time.interest` vector (the unique observed event times stored in the fitted object). Passing arbitrary time values — even plausible ones such as `c(1, 3)` for a study measured in years — causes a C-level prediction error inside `partial.rfsrc`.

`gg_partial_rfsrc` handles this automatically: every element of `partial.time` is silently snapped to its nearest `time.interest` value before the call is made. To target a specific follow-up horizon, find the closest grid point yourself and pass it explicitly:

```
ti <- rf_model$time.interest
t1 <- ti[which.min(abs(ti - 1))] # nearest to 1 year
pd <- gg_partial_rfsrc(rf_model, xvar.names = "x", partial.time = t1)
```

Logical predictor columns

`partial.rfsrc` does not handle logical predictor columns correctly in survival forests (randomForestSRC <= 3.5.1). If your training data contains binary 0/1 columns, convert them to `factor` rather than `logical` before fitting the model.

See Also

[gg_partial](#), [partial.rfsrc](#), [get.partial.plot.data](#)

Examples

```
## -----
##
## regression
##
## -----

airq.obj <- rfsrc(Ozone ~ ., data = airquality)

## partial effect for wind
prt_dta <- gg_partial_rfsrc(airq.obj,
                           xvar.names = c("Wind"))
```

`gg_rfsrc.rfsrc`

Predicted response data object

Description

Extracts the predicted response values from the `rfsrc` object, and formats data for plotting the response using `plot.gg_rfsrc`.

Usage

```
## S3 method for class 'rfsrc'
gg_rfsrc(object, oob = TRUE, by, ...)
```

Arguments

<code>object</code>	A fitted <code>rfsrc</code> or <code>randomForest</code> object.
<code>oob</code>	Logical; if TRUE (default) return out-of-bag predictions. Set to FALSE to use full in-bag (training) predictions. Forced to FALSE automatically for <code>predict.rfsrc</code> objects, which carry no OOB estimates.
<code>by</code>	Optional stratifying variable. Either a character column name present in the training data, or a vector/factor of the same length as the training set. When supplied, a group column is added to the returned data and bootstrap CI bands (survival) are computed per group. Omit or leave missing to return an unstratified result.
<code>...</code>	Additional arguments controlling output for specific forest families: <ul style="list-style-type: none"> surv_type Character; one of "surv" (default), "chf", or "mortality" for survival forests. conf.int Numeric coverage probability (e.g. 0.95) to request bootstrap pointwise confidence bands for survival forests. Triggers wide-format output with lower, upper, median, and mean columns. bs.sample Integer; number of bootstrap resamples when <code>conf.int</code> is set. Defaults to the number of observations.

Details

For survival forests, use the `surv_type` argument ("surv", "chf", or "mortality") to select the predicted quantity. Bootstrap confidence bands are requested by passing `conf.int` (e.g. `conf.int = 0.95`); the number of resamples is controlled by `bs.sample`.

Value

A `gg_rfsrc` object (a classed `data.frame`) whose structure depends on the forest family:

regression Columns `yhat` and the response name; optionally a group column when `by` is supplied.

classification One column per class with predicted probabilities; a `y` column with observed class labels; optionally `group`.

survival (no CI/ grouping) Long-format with columns `variable` (event time), `value` (survival probability), `obs_id`, and `event`.

survival (with `conf.int` or `by`) Wide-format with pointwise bootstrap CI columns (`lower`, `upper`, `median`, `mean`) per time point; a group column when `by` is supplied.

The object carries class attributes for the forest family so that `plot.gg_rfsrc` dispatches correctly.

See Also

[plot.gg_rfsrc](#), [rfsrc](#), [gg_survival](#)

Examples

```

## -----
## classification example (small, runs on CRAN)
## -----
## ----- iris data
set.seed(42)
rfsrc_iris <- rfsrc(Species ~ ., data = iris, ntree = 50)
gg_dta <- gg_rfsrc(rfsrc_iris)
plot(gg_dta)

## -----
## Additional regression / survival examples are guarded with
## \donttest because the cumulative example time exceeds the
## 10-second CRAN budget. Run locally with `R CMD check --run-donttest`
## (or `devtools::check(run_dont_test = TRUE)` ) to exercise them.
## -----

## ----- air quality data (regression)
rfsrc_airq <- rfsrc(Ozone ~ ., data = airquality,
                  na.action = "na.impute", ntree = 50)
plot(gg_rfsrc(rfsrc_airq))

## ----- Boston data (rfsrc + randomForest)
if (requireNamespace("MASS", quietly = TRUE)) {
  data(Boston, package = "MASS")
  Boston$chas <- as.logical(Boston$chas)
  rfsrc_boston <- rfsrc(medv ~ ., data = Boston, ntree = 50,
                      forest = TRUE, importance = TRUE,
                      tree.err = TRUE, save.memory = TRUE)
  plot(gg_rfsrc(rfsrc_boston))

  rf_boston <- randomForest::randomForest(medv ~ ., data = Boston,
                                         ntree = 50)

  plot(gg_rfsrc(rf_boston))
}

## ----- mtcars data
rfsrc_mtcars <- rfsrc(mpg ~ ., data = mtcars, ntree = 50)
plot(gg_rfsrc(rfsrc_mtcars))

## ----- veteran data (survival; with CI and group-by)
data(veteran, package = "randomForestSRC")
rfsrc_veteran <- rfsrc(Surv(time, status) ~ ., data = veteran,
                     ntree = 50)
plot(gg_rfsrc(rfsrc_veteran))
plot(gg_rfsrc(rfsrc_veteran, conf.int = .95))
plot(gg_rfsrc(rfsrc_veteran, by = "trt"))

```

gg_roc.rfsrc	<i>ROC (Receiver Operating Characteristic) curve data from a classification forest.</i>
--------------	---

Description

Computes sensitivity (true positive rate) and specificity (1 - false positive rate) across all prediction thresholds for one class of a classification `rfsrc` or `randomForest` object.

Usage

```
## S3 method for class 'rfsrc'
gg_roc(object, which_outcome, oob = TRUE, ...)
```

Arguments

<code>object</code>	A classification <code>rfsrc</code> or <code>randomForest</code> object. Only forests with <code>family == "class"</code> (<code>rfsrc</code>) or <code>type == "classification"</code> (<code>randomForest</code>) are supported.
<code>which_outcome</code>	Integer index or character name of the class for which the ROC curve is computed. For binary forests this is typically 1 or 2; for multi-class forests any valid class index. Use <code>which_outcome = 0</code> to obtain the overall (averaged) ROC.
<code>oob</code>	Logical; if TRUE (default) use out-of-bag predicted probabilities for the curve. Set to FALSE to use full in-bag predictions.
<code>...</code>	Extra arguments (currently unused).

Value

A `gg_roc` data.frame with one row per unique prediction threshold and columns:

sens Sensitivity (true positive rate) at each threshold.
spec Specificity (true negative rate) at each threshold.
yvar The observed class label for each observation.

Pass to `calc_auc` for the area under the curve.

See Also

[plot.gg_roc](#), [calc_roc](#), [calc_auc](#), [rfsrc](#), [randomForest](#)

Examples

```
## -----
## classification example
## -----
## ----- iris data
rfsrc_iris <- rfsrc(Species ~ ., data = iris)
```

```
# ROC for setosa
gg_dta <- gg_roc(rfsrc_iris, which_outcome = 1)
plot(gg_dta)

# ROC for versicolor
gg_dta <- gg_roc(rfsrc_iris, which_outcome = 2)
plot(gg_dta)

# ROC for virginica
gg_dta <- gg_roc(rfsrc_iris, which_outcome = 3)
plot(gg_dta)

## ----- iris data
rf_iris <- randomForest::randomForest(Species ~ ., data = iris)

# ROC for setosa
gg_dta <- gg_roc(rf_iris, which_outcome = 1)
plot(gg_dta)

# ROC for versicolor
gg_dta <- gg_roc(rf_iris, which_outcome = 2)
plot(gg_dta)

# ROC for virginica
gg_dta <- gg_roc(rf_iris, which_outcome = 3)
plot(gg_dta)
```

gg_survival

Nonparametric survival estimates.

Description

Nonparametric survival estimates.

Usage

```
gg_survival(
  object = NULL,
  interval = NULL,
  censor = NULL,
  by = NULL,
  data = NULL,
  type = c("kaplan", "nelson"),
  ...
)

## S3 method for class 'rfsrc'
gg_survival(
```

```

    object,
    interval = NULL,
    censor = NULL,
    by = NULL,
    data = NULL,
    type = c("kaplan", "nelson"),
    ...
)

## Default S3 method:
gg_survival(
  object = NULL,
  interval = NULL,
  censor = NULL,
  by = NULL,
  data = NULL,
  type = c("kaplan", "nelson"),
  ...
)

```

Arguments

object	For the <code>rfsrc</code> method: a fitted rfsrc survival forest. For the default method: pass <code>NULL</code> (or omit) and supply <code>interval</code> , <code>censor</code> , and <code>data</code> instead.
interval	Character; name of the time-to-event column in <code>data</code> (default method only).
censor	Character; name of the event-indicator column in <code>data</code> (1 = event, 0 = censored; default method only).
by	Optional character; name of a grouping column for stratified estimates. For the <code>rfsrc</code> method, <code>by</code> must be a column in <code>object\$xvar</code> .
data	A <code>data.frame</code> containing survival data (default method only).
type	One of "kaplan" (Kaplan-Meier, default) or "nelson" (Nelson-Aalen cumulative hazard). Default method only.
...	Additional arguments passed to kaplan or nelson .

Details

`gg_survival` is an S3 generic for generating nonparametric survival estimates. It dispatches on the class of its first argument:

rfsrc Extracts the response data from the fitted forest and delegates to [kaplan](#). Use the `by` argument to stratify on a predictor stored in the model's `xvar` slot.

default Accepts raw survival data columns via the `interval`, `censor`, `by`, and `data` arguments, delegating to either [kaplan](#) (default) or [nelson](#).

Value

A `gg_survival` `data.frame` with columns `time`, `surv`, `cum_haz`, `lower`, `upper`, `n.risk`, and optionally `groups` when `by` is supplied.

See Also[kaplan nelson](#)[plot.gg_survival](#)**Examples**

```
## ----- pbc data (default method - raw data columns)
data(pbc, package = "randomForestSRC")
pbc$time <- pbc$days / 364.25

gg_dta <- gg_survival(interval = "time", censor = "status", data = pbc)
plot(gg_dta, error = "none")

# Stratified
gg_dta <- gg_survival(
  interval = "time", censor = "status",
  data = pbc, by = "treatment"
)
plot(gg_dta)
```

`gg_variable`*Marginal variable dependence data object.*

Description

`plot.variable` generates a data.frame containing the marginal variable dependence or the partial variable dependence. The `gg_variable` function creates a data.frame of containing the full set of covariate data (predictor variables) and the predicted response for each observation. Marginal dependence figures are created using the `plot.gg_variable` function. For randomForest fits the original model frame is rebuilt from the stored call so that the same predictors can be paired with the in-sample predictions.

Optional arguments include `time` (scalar or vector of survival times of interest), `time_labels` (labels for multiple survival horizons) and `oob` which toggles between out-of-bag and in-bag predictions when the forest stores both.

Usage

```
gg_variable(object, ...)
```

Arguments

<code>object</code>	A <code>rfsrc</code> or <code>randomForest</code> object, or a <code>plot.variable</code> result.
<code>...</code>	Optional arguments such as <code>time</code> , <code>time_labels</code> , and <code>oob</code> that tailor the marginal dependence extraction.

Details

The marginal variable dependence is determined by comparing relation between the predicted response from the randomForest and a covariate of interest.

The gg_variable function operates on a rfsrc object, the output from the plot.variable function, or on a fitted randomForest object via the formula interface.

Value

A gg_variable object: a data.frame of all predictor columns from the training data paired with the OOB (or in-bag) predicted response. For survival forests each requested time horizon produces an additional column named by time_labels. The object carries a "family" class attribute ("regr", "class", or "surv") used by plot.gg_variable for dispatch.

See Also

[plot.gg_variable](#), [plot.variable](#)

Examples

```
## -----
## classification (small, runs on CRAN)
## -----
## ----- iris data
set.seed(42)
rfsrc_iris <- rfsrc(Species ~ ., data = iris, ntree = 50)

gg_dta <- gg_variable(rfsrc_iris)
plot(gg_dta, xvar = "Sepal.Width")

## -----
## Additional classification / regression / survival examples are
## guarded with \donttest because the cumulative example time exceeds
## the 10-second CRAN budget. Run locally with `R CMD check
## --run-donttest` (or `devtools::check(run_dont_test = TRUE)`) to
## exercise them.
## -----
plot(gg_dta, xvar = "Sepal.Length")
plot(gg_dta, xvar = rfsrc_iris$xvar.names, panel = TRUE)

## -----
## regression
## -----

## ----- air quality data
rfsrc_airq <- rfsrc(Ozone ~ ., data = airquality, ntree = 50)
gg_dta <- gg_variable(rfsrc_airq)

# an ordinal variable
gg_dta[, "Month"] <- factor(gg_dta[, "Month"])
```

```

plot(gg_dta, xvar = "Wind")
plot(gg_dta, xvar = "Temp")
plot(gg_dta, xvar = "Solar.R")
plot(gg_dta, xvar = c("Solar.R", "Wind", "Temp", "Day"), panel = TRUE)
plot(gg_dta, xvar = "Month", notch = TRUE)

## ----- motor trend cars data
rfsrc_mtcars <- rfsrc(mpg ~ ., data = mtcars, ntree = 50)

gg_dta <- gg_variable(rfsrc_mtcars)

# mtcars$cyl is an ordinal variable
gg_dta$cyl <- factor(gg_dta$cyl)
gg_dta$am <- factor(gg_dta$am)
gg_dta$vs <- factor(gg_dta$vs)
gg_dta$gear <- factor(gg_dta$gear)
gg_dta$carb <- factor(gg_dta$carb)

plot(gg_dta, xvar = "cyl")
plot(gg_dta, xvar = "disp")
plot(gg_dta, xvar = "hp")
plot(gg_dta, xvar = "wt")
plot(gg_dta, xvar = c("disp", "hp", "drat", "wt", "qsec"), panel = TRUE)
plot(gg_dta,
     xvar = c("cyl", "vs", "am", "gear", "carb"), panel = TRUE,
     notch = TRUE
)

## ----- Boston data
if (requireNamespace("MASS", quietly = TRUE)) {
  data(Boston, package = "MASS")
  rf_boston <- randomForest::randomForest(medv ~ ., data = Boston)
  gg_dta <- gg_variable(rf_boston)
  plot(gg_dta)
  plot(gg_dta, panel = TRUE)
}

## -----
## survival examples
## -----

## ----- veteran data
data(veteran, package = "randomForestSRC")
rfsrc_veteran <- rfsrc(Surv(time, status) ~ ., veteran,
  nsplit = 10,
  ntree = 50
)

# get the 90-day survival time.
gg_dta <- gg_variable(rfsrc_veteran, time = 90)

# Generate variable dependence plots for age and diagtime
plot(gg_dta, xvar = "age")

```

```
plot(gg_dta, xvar = "diagtime")

# Generate coplots
plot(gg_dta, xvar = c("age", "diagtime"), panel = TRUE, se = FALSE)

# Compare survival at 30, 90, and 365 days simultaneously
gg_dta <- gg_variable(rfsrc_veteran, time = c(30, 90, 365))
plot(gg_dta, xvar = "age")
```

gg_vimp

Variable Importance (VIMP) data object

Description

`gg_vimp` Extracts the variable importance (VIMP) information from a `rfsrc` or `randomForest` object and reshapes it into a tidy data set.

Usage

```
gg_vimp(object, nvar, ...)
```

Arguments

<code>object</code>	A <code>rfsrc</code> object, the output from <code>vimp</code> , or a fitted <code>randomForest</code> .
<code>nvar</code>	argument to control the number of variables included in the output.
<code>...</code>	arguments passed to the <code>vimp.rfsrc</code> function if the <code>rfsrc</code> object does not contain importance information.

Value

`gg_vimp` object. A `data.frame` of VIMP measures, in rank order, optionally containing class-specific scores and a relative importance column. When `randomForest` objects lack stored importance values a warning is issued and NA placeholders are returned so plots remain reproducible.

References

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

See Also

[plot.gg_vimp rfsrc](#)
[vimp](#)

Examples

```

## -----
## classification example
## -----
## ----- iris data
rfsrc_iris <- rfsrc(Species ~ .,
  data = iris,
  importance = TRUE
)
gg_dta <- gg_vimp(rfsrc_iris)
plot(gg_dta)

## -----
## regression example
## -----

## ----- air quality data
rfsrc_airq <- rfsrc(Ozone ~ ., airquality,
  importance = TRUE
)
gg_dta <- gg_vimp(rfsrc_airq)
plot(gg_dta)

## ----- Boston data
data(Boston, package = "MASS")
rfsrc_boston <- randomForestSRC::rfsrc(medv ~ ., Boston,
  importance = TRUE
)
gg_dta <- gg_vimp(rfsrc_boston)
plot(gg_dta)

## ----- Boston data
rf_boston <- randomForest::randomForest(medv ~ ., Boston)
gg_dta <- gg_vimp(rf_boston)
plot(gg_dta)

## ----- mtcars data
rfsrc_mtcars <- rfsrc(mpg ~ .,
  data = mtcars,
  importance = TRUE
)
gg_dta <- gg_vimp(rfsrc_mtcars)
plot(gg_dta)

## -----
## survival example
## -----

## ----- veteran data
data(veteran, package = "randomForestSRC")

```

```

rfsrc_veteran <- rfsrc(Surv(time, status) ~ .,
  data = veteran,
  ntree = 100,
  importance = TRUE
)

gg_dta <- gg_vimp(rfsrc_veteran)
plot(gg_dta)

## ----- pbc data
# We need to create this dataset
data(pbc, package = "randomForestSRC", )
# For whatever reason, the age variable is in days...
# makes no sense to me
for (ind in seq_len(dim(pbc)[2])) {
  if (!is.factor(pbc[, ind])) {
    if (length(unique(pbc[which(!is.na(pbc[, ind])), ind])) <= 2) {
      if (sum(range(pbc[, ind], na.rm = TRUE) == c(0, 1)) == 2) {
        pbc[, ind] <- as.logical(pbc[, ind])
      }
    }
  } else {
    if (length(unique(pbc[which(!is.na(pbc[, ind])), ind])) <= 2) {
      if (sum(sort(unique(pbc[, ind])) == c(0, 1)) == 2) {
        pbc[, ind] <- as.logical(pbc[, ind])
      }
      if (sum(sort(unique(pbc[, ind])) == c(FALSE, TRUE)) == 2) {
        pbc[, ind] <- as.logical(pbc[, ind])
      }
    }
  }
  if (!is.logical(pbc[, ind]) &
    length(unique(pbc[which(!is.na(pbc[, ind])), ind])) <= 5) {
    pbc[, ind] <- factor(pbc[, ind])
  }
}
# Convert age to years
pbc$age <- pbc$age / 364.24

pbc$years <- pbc$days / 364.24
pbc <- pbc[, -which(colnames(pbc) == "days")]
pbc$treatment <- as.numeric(pbc$treatment)
pbc$treatment[which(pbc$treatment == 1)] <- "DPCA"
pbc$treatment[which(pbc$treatment == 2)] <- "placebo"
pbc$treatment <- factor(pbc$treatment)
dta_train <- pbc[-which(is.na(pbc$treatment)), ]
# Create a test set from the remaining patients
pbc_test <- pbc[which(is.na(pbc$treatment)), ]

# =====
# build the forest:
rfsrc_pbc <- randomForestSRC::rfsrc(
  Surv(years, status) ~ .,

```

```
dta_train,
  nsplit = 10,
  na.action = "na.impute",
  forest = TRUE,
  importance = TRUE,
  save.memory = TRUE
)

gg_dta <- gg_vimp(rfsrc_pbc)
plot(gg_dta)

# Restrict to only the top 10.
gg_dta <- gg_vimp(rfsrc_pbc, nvar = 10)
plot(gg_dta)
```

kaplan

nonparametric Kaplan-Meier estimates

Description

nonparametric Kaplan-Meier estimates

Usage

```
kaplan(interval, censor, data, by = NULL, ...)
```

Arguments

interval	name of the interval variable in the training dataset.
censor	name of the censoring variable in the training dataset.
data	name of the training set data.frame
by	stratifying variable in the training dataset, defaults to NULL
...	arguments passed to the survfit function

Value

[gg_survival](#) object

See Also

[gg_survival](#) [nelson](#) [plot.gg_survival](#)

Examples

```

# These get run through the gg_survival examples.
data(pbc, package = "randomForestSRC")
pbc$time <- pbc$days / 364.25

# This is the same as gg_survival
gg_dta <- kaplan(
  interval = "time", censor = "status",
  data = pbc
)

plot(gg_dta, error = "none")
plot(gg_dta)

# Stratified on treatment variable.
gg_dta <- gg_survival(
  interval = "time", censor = "status",
  data = pbc, by = "treatment"
)

plot(gg_dta, error = "none")
plot(gg_dta)

```

nelson

nonparametric Nelson-Aalen estimates

Description

nonparametric Nelson-Aalen estimates

Usage

```
nelson(interval, censor, data, by = NULL, weight = NULL, ...)
```

Arguments

interval	name of the interval variable in the training dataset.
censor	name of the censoring variable in the training dataset.
data	name of the survival training data.frame
by	stratifying variable in the training dataset, defaults to NULL
weight	for each observation (default=NULL)
...	arguments passed to the survfit function

Value

`gg_survival` object

See Also

[gg_survival nelson plot.gg_survival](#)

Examples

```
# These get run through the gg_survival examples.
data(pbc, package = "randomForestSRC")
pbc$time <- pbc$days / 364.25

# This is the same as gg_survival
gg_dta <- nelson(
  interval = "time", censor = "status",
  data = pbc
)

plot(gg_dta, error = "none")
plot(gg_dta)

# Stratified on treatment variable.
gg_dta <- gg_survival(
  interval = "time", censor = "status",
  data = pbc, by = "treatment"
)

plot(gg_dta, error = "none")
plot(gg_dta, error = "lines")
plot(gg_dta)

gg_dta <- gg_survival(
  interval = "time", censor = "status",
  data = pbc, by = "treatment",
  type = "nelson"
)

plot(gg_dta, error = "bars")
plot(gg_dta)
```

plot.gg_error

Plot a [gg_error](#) object

Description

A plot of the cumulative OOB error rates of the random forest as a function of number of trees.

Usage

```
## S3 method for class 'gg_error'
plot(x, ...)
```

Arguments

- x A `gg_error` object created from either a `rfsrc` or a `randomForest` object. A raw forest object may also be supplied and will be passed through `gg_error` automatically before plotting.
- ... Extra arguments forwarded to the underlying `ggplot2` geometry calls (e.g. `size`, `linetype`).

Details

The `gg_error` plot is used to track the convergence of the `randomForest`. This figure is a reproduction of the error plot from the `plot.rfsrc` function.

Value

A `ggplot` object with `ntree` on the x-axis and OOB error rate on the y-axis. Single-outcome forests (regression, survival) produce a single line; multi-outcome forests (classification) produce one coloured line per class.

References

- Breiman L. (2001). Random forests, *Machine Learning*, 45:5-32.
- Ishwaran H. and Kogalur U.B. (2007). Random survival forests for R, *Rnews*, 7(2):25-31.
- Ishwaran H. and Kogalur U.B. `randomForestSRC`: Random Forests for Survival, Regression and Classification. R package version $\geq 3.4.0$. <https://cran.r-project.org/package=randomForestSRC>

See Also

[gg_error](#) [rfsrc](#) [plot.rfsrc](#)

Examples

```
## Examples from RFSRC package...
## -----
## classification example
## -----
## ----- iris data
## You can build a randomForest
rfsrc_iris <- rfsrc(Species ~ ., data = iris,
  forest = TRUE,
  importance = TRUE,
  tree.err = TRUE,
  save.memory = TRUE)

# Get a data.frame containing error rates
gg_dta <- gg_error(rfsrc_iris)

# Plot the gg_error object
plot(gg_dta)

## RandomForest example
```

```

rf_iris <- randomForest::randomForest(Species ~ .,
  data = iris,
  forest = TRUE,
  importance = TRUE,
  tree.err = TRUE,
  save.memory = TRUE
)
gg_dta <- gg_error(rf_iris)
plot(gg_dta)

gg_dta <- gg_error(rf_iris, training = TRUE)
plot(gg_dta)
## -----
## Regression example
## -----
## ----- airq data
rfsrc_airq <- rfsrc(Ozone ~ .,
  data = airquality,
  na.action = "na.impute",
  forest = TRUE,
  importance = TRUE,
  tree.err = TRUE,
  save.memory = TRUE
)

# Get a data.frame containing error rates
gg_dta <- gg_error(rfsrc_airq)

# Plot the gg_error object
plot(gg_dta)

## ----- Boston data
data(Boston, package = "MASS")
Boston$chas <- as.logical(Boston$chas)
rfsrc_boston <- rfsrc(medv ~ .,
  data = Boston,
  forest = TRUE,
  importance = TRUE,
  tree.err = TRUE,
  save.memory = TRUE
)

# Get a data.frame containing error rates
gg_dta <- gg_error(rfsrc_boston)

# Plot the gg_error object
plot(gg_dta)

## ----- mtcars data
rfsrc_mtcars <- rfsrc(mpg ~ ., data = mtcars,
  importance = TRUE,
  save.memory = TRUE,

```

```

    forest = TRUE,
    tree.err = TRUE)

# Get a data.frame containing error rates
gg_dta<- gg_error(rfsrc_mtcars)

# Plot the gg_error object
plot(gg_dta)

## -----
## Survival example
## -----
## ----- veteran data
## randomized trial of two treatment regimens for lung cancer
data(veteran, package = "randomForestSRC")
rfsrc_veteran <- rfsrc(Surv(time, status) ~ ., data = veteran,
                      tree.err = TRUE)

gg_dta <- gg_error(rfsrc_veteran)
plot(gg_dta)

## ----- pbc data
# Load a cached randomForestSRC object
# We need to create this dataset
data(pbc, package = "randomForestSRC",)
# For whatever reason, the age variable is in days... makes no sense to me
for (ind in seq_len(dim(pbc)[2])) {
  if (!is.factor(pbc[, ind])) {
    if (length(unique(pbc[which(!is.na(pbc[, ind])), ind])) <= 2) {
      if (sum(range(pbc[, ind], na.rm = TRUE) == c(0, 1)) == 2) {
        pbc[, ind] <- as.logical(pbc[, ind])
      }
    }
  } else {
    if (length(unique(pbc[which(!is.na(pbc[, ind])), ind])) <= 2) {
      if (sum(sort(unique(pbc[, ind])) == c(0, 1)) == 2) {
        pbc[, ind] <- as.logical(pbc[, ind])
      }
      if (sum(sort(unique(pbc[, ind])) == c(FALSE, TRUE)) == 2) {
        pbc[, ind] <- as.logical(pbc[, ind])
      }
    }
  }
  if (!is.logical(pbc[, ind]) &
      length(unique(pbc[which(!is.na(pbc[, ind])), ind])) <= 5) {
    pbc[, ind] <- factor(pbc[, ind])
  }
}
#Convert age to years
pbc$age <- pbc$age / 364.24

pbc$years <- pbc$days / 364.24

```

```

pbc <- pbc[, -which(colnames(pbc) == "days")]
pbc$treatment <- as.numeric(pbc$treatment)
pbc$treatment[which(pbc$treatment == 1)] <- "DPCA"
pbc$treatment[which(pbc$treatment == 2)] <- "placebo"
pbc$treatment <- factor(pbc$treatment)
dta_train <- pbc[-which(is.na(pbc$treatment)), ]
# Create a test set from the remaining patients
pbc_test <- pbc[which(is.na(pbc$treatment)), ]

#=====
# build the forest:
rfsrc_pbc <- randomForestSRC::rfsrc(
  Surv(years, status) ~ .,
  dta_train,
  nsplit = 10,
  na.action = "na.impute",
  tree.err = TRUE,
  forest = TRUE,
  importance = TRUE,
  save.memory = TRUE
)

gg_dta <- gg_error(rfsrc_pbc)
plot(gg_dta)

```

plot.gg_partial

Plot a [gg_partial](#) object

Description

Produces ggplot2 partial dependence curves from the named list returned by [gg_partial](#). Continuous predictors are shown as line plots; categorical predictors are shown as bar charts. Both panels are faceted by variable name so multiple predictors can be compared at a glance.

Usage

```

## S3 method for class 'gg_partial'
plot(x, ...)

```

Arguments

x	A gg_partial object (output of gg_partial).
...	Not currently used; reserved for future arguments.

Value

When only continuous or only categorical variables are present, a single ggplot object. When both are present, a named list with elements continuous and categorical, each a ggplot.

See Also

[gg_partial](#), [plot.gg_variable](#)

Examples

```
set.seed(42)
airq <- na.omit(airquality)
rf <- randomForestSRC::rfsrc(Ozone ~ ., data = airq, ntree = 50)
pv <- randomForestSRC::plot.variable(rf, partial = TRUE, show.plots = FALSE)
pd <- gg_partial(pv)
plot(pd)
```

plot.gg_partialpro *Plot a [gg_partialpro](#) object*

Description

Produces ggplot2 partial dependence curves from the named list returned by [gg_partialpro](#), which wraps varpro::partialpro output.

Usage

```
## S3 method for class 'gg_partialpro'
plot(x, type = c("parametric", "nonparametric", "causal"), ...)
```

Arguments

x	A gg_partialpro object.
type	Character vector; one or more of "parametric", "nonparametric", "causal". Defaults to all three.
...	Not currently used.

Details

Each variable produces up to three effect curves: parametric, nonparametric, and causal. The type argument controls which are shown.

Value

A single ggplot or a named list with continuous and categorical elements when both types of predictors are present.

See Also

[gg_partialpro](#)

Examples

```

## ggRandomForests does not depend on the varpro package; we construct a
## minimal mock of the partialpro() output so the example runs everywhere.
set.seed(42)
n_obs <- 30
n_pts <- 15
mock_data <- list(
  age = list(
    xvirtual   = seq(30, 80, length.out = n_pts),
    xorg       = sample(seq(30, 80, by = 5), n_obs, replace = TRUE),
    yhat.par   = matrix(rnorm(n_obs * n_pts), nrow = n_obs),
    yhat.nonpar = matrix(rnorm(n_obs * n_pts), nrow = n_obs),
    yhat.causal = matrix(rnorm(n_obs * n_pts), nrow = n_obs)
  ),
  sex = list(
    xvirtual   = c(0, 1),
    xorg       = sample(c(0, 1), n_obs, replace = TRUE),
    yhat.par   = matrix(rnorm(n_obs * 2), nrow = n_obs),
    yhat.nonpar = matrix(rnorm(n_obs * 2), nrow = n_obs),
    yhat.causal = matrix(rnorm(n_obs * 2), nrow = n_obs)
  )
)

pp <- gg_partialpro(mock_data)
result <- plot(pp)

# Continuous predictors get one panel per variable; categorical get
# box-plots over the parametric / nonparametric / causal effect types.
result$continuous
result$categorical

# Restrict to one or two effect types
plot(pp, type = "nonparametric")
plot(pp, type = c("parametric", "causal"))

```

plot.gg_partial_rfsrc *Plot a [gg_partial_rfsrc](#) object*

Description

Produces ggplot2 partial dependence curves from the named list returned by [gg_partial_rfsrc](#).

Usage

```

## S3 method for class 'gg_partial_rfsrc'
plot(x, ...)

```

Arguments

x A `gg_partial_rfsrc` object.
 ... Not currently used.

Details

For standard (non-survival) forests: continuous predictors are line plots, categorical predictors are bar charts, both faceted by variable name.

For survival forests (when a time column is present): each evaluation time point is a separate curve over the predictor's value, faceted by variable name. The y-axis label adapts to the `partial.type` stored on the object ("Predicted Survival", "Predicted CHF", or "Predicted Mortality").

For two-variable surface plots (when a `grp` column is present): each group level is a separate line, faceted by primary predictor name.

Value

A single `ggplot` object, or a named list with elements `continuous` and `categorical` when both types are present.

See Also

[gg_partial_rfsrc](#), [plot.gg_partial](#)

Examples

```
## -----
## Regression forest -- one continuous curve per variable
## -----
set.seed(42)
airq <- na.omit(airquality)
rfsrc_airq <- randomForestSRC::rfsrc(Ozone ~ ., data = airq, ntree = 50)

pd <- gg_partial_rfsrc(rfsrc_airq, xvar.names = c("Wind", "Temp"),
                      n_eval = 10)
plot(pd)

## -----
## Survival forest -- one curve per requested time horizon,
## faceted by variable. Y-axis label tracks `partial.type`.
## -----
# randomForestSRC's formula parser requires the unqualified Surv() symbol;
# it Depends on `survival`, so Surv is on the search path once
# randomForestSRC is loaded.
data(veteran, package = "randomForestSRC")
set.seed(42)
rfsrc_v <- randomForestSRC::rfsrc(Surv(time, status) ~ .,
                                  data = veteran, ntree = 50)

ti <- rfsrc_v$time.interest
t30 <- ti[which.min(abs(ti - 30))]
```

```

t90 <- ti[which.min(abs(ti - 90))]

# Default partial.type = "surv" -> y-axis "Predicted Survival"
pd_s <- gg_partial_rfsrc(rfsrc_v, xvar.names = "age",
                        partial.time = c(t30, t90), n_eval = 8)
plot(pd_s)

# partial.type = "chf" -> y-axis "Predicted CHF"
pd_c <- gg_partial_rfsrc(rfsrc_v, xvar.names = "age",
                        partial.time = c(t30, t90),
                        partial.type = "chf", n_eval = 8)
plot(pd_c)

```

plot.gg_rfsrc

Predicted response plot from a gg_rfsrc object.

Description

Plot the predicted response from a `gg_rfsrc` object, the `rfsrc` prediction, using the OOB prediction from the forest. The plot type adapts automatically to the forest family: jitter + boxplot for regression and classification, step curves for survival.

Usage

```

## S3 method for class 'gg_rfsrc'
plot(x, notch = TRUE, ...)

```

Arguments

<code>x</code>	A <code>gg_rfsrc</code> object, or a raw <code>rfsrc</code> object (which will be passed through <code>gg_rfsrc</code> automatically before plotting).
<code>notch</code>	Logical; whether to draw notched boxplots for regression and classification forests (default TRUE). Set <code>notch = FALSE</code> to suppress notches when sample sizes are too small for reliable confidence intervals on the median.
<code>...</code>	Additional arguments forwarded to the underlying <code>ggplot2</code> geometry calls. Commonly useful arguments include: <ul style="list-style-type: none"> <code>alpha</code> Numeric in $[0, 1]$; point/ribbon transparency. For survival plots with confidence bands the ribbon <code>alpha</code> is automatically halved relative to the value supplied here. <code>size</code> Point or line size passed to <code>geom_jitter</code>, <code>geom_step</code>, etc. Arguments that control <code>gg_rfsrc</code> (e.g. <code>conf.int</code> , <code>surv_type</code> , <code>by</code>) should be applied when constructing the <code>gg_rfsrc</code> object before calling <code>plot()</code> .

Value

A ggplot object. The plot appearance depends on the forest family stored in `x`:

Regression ("regr") Jitter + notched boxplot of OOB predicted values. If a group column is present the x-axis shows each group label; otherwise observations are collapsed to a single x-position.

Classification ("class") Binary: jitter + notched boxplot of the predicted class probability. Multi-class: jitter plot with one panel per class (class probabilities in long form).

Survival ("surv") Step curves of the ensemble survival function. When `gg_rfsrc` was called with `conf.int`, a shaded ribbon is added. When called with `by`, curves are coloured by group.

References

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

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

Ishwaran H. and Kogalur U.B. randomForestSRC: Random Forests for Survival, Regression and Classification. R package version >= 3.4.0. <https://cran.r-project.org/package=randomForestSRC>

See Also

[gg_rfsrc](#) [rfsrc](#) [randomForest](#)

Examples

```
## -----
## classification example
## -----
## ----- iris data
# Build a small classification forest (ntree=50 keeps example fast)
set.seed(42)
rfsrc_iris <- rfsrc(Species ~ ., data = iris, ntree = 50)
gg_dta <- gg_rfsrc(rfsrc_iris)

plot(gg_dta)

## -----
## Regression example
## -----
## ----- air quality data
# na.action = "na.impute" handles missing Ozone / Solar.R values
set.seed(42)
rfsrc_airq <- rfsrc(Ozone ~ ., data = airquality,
                  na.action = "na.impute", ntree = 50)
gg_dta <- gg_rfsrc(rfsrc_airq)

plot(gg_dta)

## ----- mtcars data
set.seed(42)
rfsrc_mtcars <- rfsrc(mpg ~ ., data = mtcars, ntree = 50)
```

```

gg_dta <- gg_rfsrc(rfsrc_mtcars)

plot(gg_dta)

## -----
## Survival example
## -----
## ----- veteran data
## randomized trial of two treatment regimens for lung cancer
data(veteran, package = "randomForestSRC")
set.seed(42)
rfsrc_veteran <- rfsrc(Surv(time, status) ~ ., data = veteran, ntree = 50)
gg_dta <- gg_rfsrc(rfsrc_veteran)
plot(gg_dta)

# With 95% pointwise bootstrap confidence bands
gg_dta <- gg_rfsrc(rfsrc_veteran, conf.int = .95)
plot(gg_dta)

# Stratified by treatment arm
gg_dta <- gg_rfsrc(rfsrc_veteran, by = "trt")
plot(gg_dta)

## ----- pbc data (larger dataset -- skipped on CRAN)

data(pbc, package = "randomForestSRC")
# For whatever reason, the age variable is in days; convert to years
for (ind in seq_len(dim(pbc)[2])) {
  if (!is.factor(pbc[, ind])) {
    if (length(unique(pbc[which(!is.na(pbc[, ind])), ind])) <= 2) {
      if (sum(range(pbc[, ind], na.rm = TRUE) == c(0, 1)) == 2) {
        pbc[, ind] <- as.logical(pbc[, ind])
      }
    }
  } else {
    if (length(unique(pbc[which(!is.na(pbc[, ind])), ind])) <= 2) {
      if (sum(sort(unique(pbc[, ind])) == c(0, 1)) == 2) {
        pbc[, ind] <- as.logical(pbc[, ind])
      }
      if (sum(sort(unique(pbc[, ind])) == c(FALSE, TRUE)) == 2) {
        pbc[, ind] <- as.logical(pbc[, ind])
      }
    }
  }
  if (!is.logical(pbc[, ind]) &
      length(unique(pbc[which(!is.na(pbc[, ind])), ind])) <= 5) {
    pbc[, ind] <- factor(pbc[, ind])
  }
}
# Convert age from days to years
pbc$age <- pbc$age / 364.24
pbc$years <- pbc$days / 364.24
pbc <- pbc[, -which(colnames(pbc) == "days")]

```

```

pbc$treatment <- as.numeric(pbc$treatment)
pbc$treatment[which(pbc$treatment == 1)] <- "DPCA"
pbc$treatment[which(pbc$treatment == 2)] <- "placebo"
pbc$treatment <- factor(pbc$treatment)
# Remove test-set patients (those with no assigned treatment)
dta_train <- pbc[-which(is.na(pbc$treatment)), ]

set.seed(42)
rfsrc_pbc <- randomForestSRC::rfsrc(
  Surv(years, status) ~ .,
  dta_train,
  nsplit = 10,
  na.action = "na.impute",
  forest = TRUE,
  importance = TRUE,
  save.memory = TRUE
)

gg_dta <- gg_rfsrc(rfsrc_pbc)
plot(gg_dta)

gg_dta <- gg_rfsrc(rfsrc_pbc, conf.int = .95)
plot(gg_dta)

gg_dta <- gg_rfsrc(rfsrc_pbc, by = "treatment")
plot(gg_dta)

```

plot.gg_roc

ROC plot generic function for a [gg_roc](#) object.

Description

ROC plot generic function for a [gg_roc](#) object.

Usage

```

## S3 method for class 'gg_roc'
plot(x, which_outcome = NULL, ...)

```

Arguments

x	A gg_roc object, or a raw rfsrc classification forest or randomForest classification object. When a forest is supplied, gg_roc is called automatically.
which_outcome	Integer; for multi-class problems, the index of the class whose ROC curve should be plotted. When NULL (default) and the forest has more than two classes, ROC curves for all classes are overlaid in a single plot. For binary forests NULL defaults to class index 2.

... Additional arguments forwarded to `gg_roc` when `x` is a raw forest object (e.g. `oob = FALSE`).

Value

A `ggplot` object. The x-axis shows 1 - Specificity (FPR) and the y-axis shows Sensitivity (TPR). A dashed red diagonal reference line marks the random-classifier baseline. The AUC value is annotated on the plot for single-class curves. Multi-class plots colour and style each class curve distinctly.

References

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

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

Ishwaran H. and Kogalur U.B. `randomForestSRC`: Random Forests for Survival, Regression and Classification. R package version $\geq 3.4.0$. <https://cran.r-project.org/package=randomForestSRC>

See Also

[gg_roc](#) [calc_roc](#) [calc_auc](#) [rfsrc](#) [randomForest](#)

Examples

```
## -----
## classification example
## -----
## ----- iris data
# Build a small classification forest (ntree=50 keeps example fast)
set.seed(42)
rfsrc_iris <- rfsrc(Species ~ ., data = iris, ntree = 50)

# ROC for setosa (outcome index 1)
gg_dta <- gg_roc(rfsrc_iris, which_outcome = 1)
plot(gg_dta)

# ROC for versicolor (outcome index 2)
gg_dta <- gg_roc(rfsrc_iris, which_outcome = 2)
plot(gg_dta)

# ROC for virginica (outcome index 3)
gg_dta <- gg_roc(rfsrc_iris, which_outcome = 3)
plot(gg_dta)

# Plot all three ROC curves in one call by iterating over outcome indices
n_cls <- ncol(rfsrc_iris$predicted)
for (i in seq_len(n_cls)) print(plot(gg_roc(rfsrc_iris, which_outcome = i)))
```

plot.gg_survival *Plot a gg_survival object.*

Description

Plot a `gg_survival` object.

Usage

```
## S3 method for class 'gg_survival'
plot(
  x,
  type = c("surv", "cum_haz", "hazard", "density", "mid_int", "life", "proplife"),
  error = c("shade", "bars", "lines", "none"),
  label = NULL,
  ...
)
```

Arguments

<code>x</code>	<code>gg_survival</code> or a survival <code>gg_rfsrc</code> object created from a <code>rfsrc</code> object
<code>type</code>	"surv", "cum_haz", "hazard", "density", "mid_int", "life", "proplife"
<code>error</code>	"shade", "bars", "lines" or "none"
<code>label</code>	Modify the legend label when <code>gg_survival</code> has stratified samples
<code>...</code>	not used

Value

A ggplot object. The y-axis shows the chosen type (e.g. survival probability for "surv") and the x-axis shows time. Confidence shading, bars, or lines are added when the input object carries confidence-interval columns.

See Also

[gg_survival](#), [kaplan](#), [nelson](#), [gg_rfsrc](#)

Examples

```
## ----- pbc data
data(pbc, package = "randomForestSRC")
pbc$time <- pbc$days / 364.25

# This is the same as kaplan
gg_dta <- gg_survival(
  interval = "time", censor = "status",
  data = pbc
)
```

```
plot(gg_dta, error = "none")
plot(gg_dta)

# Stratified on treatment variable.
gg_dta <- gg_survival(
  interval = "time", censor = "status",
  data = pbc, by = "treatment"
)

plot(gg_dta, error = "none")
plot(gg_dta)
plot(gg_dta, label = "treatment")

# ...with smaller confidence limits.
gg_dta <- gg_survival(
  interval = "time", censor = "status",
  data = pbc, by = "treatment", conf.int = .68
)

plot(gg_dta, error = "lines")
plot(gg_dta, label = "treatment", error = "lines")

# ...with smaller confidence limits.
gg_dta <- gg_survival(
  interval = "time", censor = "status",
  data = pbc, by = "sex", conf.int = .68
)

plot(gg_dta, error = "lines")
plot(gg_dta, label = "sex", error = "lines")
```

plot.gg_variable *Plot a [gg_variable](#) object,*

Description

Plot a [gg_variable](#) object,

Usage

```
## S3 method for class 'gg_variable'
plot(
  x,
  xvar,
  time,
  time_labels,
  panel = FALSE,
  oob = TRUE,
```

```

    points = TRUE,
    smooth = TRUE,
    ...
  )

```

Arguments

x	gg_variable object created from a rfsrc object
xvar	variable (or list of variables) of interest.
time	For survival, one or more times of interest
time_labels	string labels for times
panel	Should plots be faceted along multiple xvar?
oob	oob estimates (boolean)
points	plot the raw data points (boolean)
smooth	include a smooth curve (boolean)
...	arguments passed to the ggplot2 functions.

Value

A single ggplot object when `length(xvar) == 1` or `panel = TRUE`. Otherwise a named list of ggplot objects, one per variable in `xvar`.

References

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

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

Ishwaran H. and Kogalur U.B. randomForestSRC: Random Forests for Survival, Regression and Classification. R package version >= 3.4.0. <https://cran.r-project.org/package=randomForestSRC>

See Also

[gg_variable](#), [gg_partial](#), [plot.variable](#)

Examples

```

## -----
## classification
## -----
## ----- iris data
set.seed(42)
rfsrc_iris <- rfsrc(Species ~ ., data = iris, ntree = 50)

gg_dta <- gg_variable(rfsrc_iris)
plot(gg_dta, xvar = "Sepal.Width")
plot(gg_dta, xvar = "Sepal.Length")

## Panel plot across all predictors
plot(gg_dta,

```

```

    xvar = rfsrc_iris$xvar.names,
    panel = TRUE, se = FALSE
  )

  ## -----
  ## regression
  ## -----
  ## ----- air quality data
  # na.action = "na.impute" handles missing Ozone / Solar.R values
  set.seed(42)
  rfsrc_airq <- rfsrc(Ozone ~ ., data = airquality,
                    na.action = "na.impute", ntree = 50)
  gg_dta <- gg_variable(rfsrc_airq)

  # Treat Month as an ordinal factor for better visualisation
  gg_dta[, "Month"] <- factor(gg_dta[, "Month"])

  plot(gg_dta, xvar = "Wind")
  plot(gg_dta, xvar = "Temp")
  plot(gg_dta, xvar = "Solar.R")

  # Panel plot across continuous predictors
  plot(gg_dta, xvar = c("Solar.R", "Wind", "Temp", "Day"), panel = TRUE)

  # Factor variable uses notched boxplots
  plot(gg_dta, xvar = "Month", notch = TRUE)

  ## -----
  ## survival examples
  ## -----
  ## ----- veteran data
  data(veteran, package = "randomForestSRC")
  set.seed(42)
  rfsrc_veteran <- rfsrc(Surv(time, status) ~ ., veteran,
                       nsplit = 10,
                       ntree = 50
  )

  # Marginal survival at 90 days
  gg_dta <- gg_variable(rfsrc_veteran, time = 90)

  # Single-variable dependence plots
  plot(gg_dta, xvar = "age")
  plot(gg_dta, xvar = "diagtime")

  # Panel coplot for two predictors at a single time
  plot(gg_dta, xvar = c("age", "diagtime"), panel = TRUE)

  # Compare survival at 30, 90, and 365 days simultaneously
  gg_dta <- gg_variable(rfsrc_veteran, time = c(30, 90, 365))

  # Single-variable plot (one facet per time point)
  plot(gg_dta, xvar = "age")

```

```
# Panel coplot across two predictors and three time points
plot(gg_dta, xvar = c("age", "diagtime"), panel = TRUE)
```

plot.gg_vimp	<i>Plot a <code>gg_vimp</code> object, extracted variable importance of a <code>rfsrc</code> object</i>
--------------	---

Description

Plot a `gg_vimp` object, extracted variable importance of a `rfsrc` object

Usage

```
## S3 method for class 'gg_vimp'
plot(x, relative, lbls, ...)
```

Arguments

x	<code>gg_vimp</code> object created from a <code>rfsrc</code> object
relative	should we plot vimp or relative vimp. Defaults to vimp.
lbls	A vector of alternative variable labels. Item names should be the same as the variable names.
...	optional arguments passed to <code>gg_vimp</code> if necessary

Value

ggplot object

References

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

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

Ishwaran H. and Kogalur U.B. randomForestSRC: Random Forests for Survival, Regression and Classification. R package version >= 3.4.0. <https://cran.r-project.org/package=randomForestSRC>

See Also

`gg_vimp`

Examples

```
## -----
## classification example
## -----
## ----- iris data
rfsrc_iris <- rfsrc(Species ~ ., data = iris)
gg_dta <- gg_vimp(rfsrc_iris)
plot(gg_dta)

## -----
## regression example
## -----
## ----- air quality data
rfsrc_airq <- rfsrc(Ozone ~ ., airquality)
gg_dta <- gg_vimp(rfsrc_airq)
plot(gg_dta)
```

quantile_pts

Quantile-based cut points for coplots

Description

This helper wraps `quantile` to create well-spaced cut points for conditioning plots. When `intervals = TRUE` the lower boundary is nudged down so that `cut()` treats the minimum value as a valid observation.

The output can be passed directly into the `breaks` argument of the `cut` function for creating groups for coplots.

Usage

```
quantile_pts(object, groups, intervals = FALSE)
```

Arguments

<code>object</code>	Numeric vector of predictor values.
<code>groups</code>	Number of quantile points (or intervals) to compute.
<code>intervals</code>	Logical indicating whether to return interval boundaries suitable for <code>cut()</code> (length <code>groups + 1</code>) or the interior quantile points (length <code>groups</code>).

Value

Numeric vector of quantile points. When `intervals = TRUE` the result is strictly increasing and can be supplied to `cut()` to produce groups balanced strata.

See Also

cut

Examples

```

data(Boston, package = "MASS")
rfsrc_boston <- randomForestSRC::rfsrc(medv ~ ., Boston)

# To create 6 intervals, we want 7 points.
# quantile_pts will find balanced intervals
rm_pts <- quantile_pts(rfsrc_boston$xvar$rm, groups = 6, intervals = TRUE)

# Use cut to create the intervals
rm_grp <- cut(rfsrc_boston$xvar$rm, breaks = rm_pts)

summary(rm_grp)

```

surv_partial.rfsrc *Survival partial dependence data for one or more predictors*

Description

Deprecated. Use [gg_partial_rfsrc](#) instead, which returns a classed `gg_partial_rfsrc` object with a dedicated `plot()` method.

Usage

```
surv_partial.rfsrc(rforest, var_list, npts = 25, partial.type = "surv")
```

Arguments

<code>rforest</code>	A fitted rfsrc survival or competing-risk forest object.
<code>var_list</code>	Character vector of predictor names for which partial dependence should be computed. Each must appear in <code>rforest\$xvar.names</code> .
<code>npts</code>	Integer; the number of predictor grid points to evaluate (default 25). Evenly-spaced unique values are sampled from each predictor.
<code>partial.type</code>	The prediction type to return. For survival forests one of "surv" (default), "mort", or "chf". For competing risk forests one of "years.lost", "cif", or "chf". See partial.rfsrc for full details.

Details

Computes partial dependence curves for a survival or competing-risk [rfsrc](#) forest by calling [partial.rfsrc](#) at `npts` evenly-spaced unique values of each predictor across all stored event times.

Value

A named list with one element per variable in `var_list`. Each element is itself a list with:

name The predictor variable name (character).

dta The raw output of `get.partial.plot.data`, a list containing at minimum `x` (predictor values) and `yhat` (partial predictions), and for survival/competing risk, `partial.time`.

See Also

[gg_partial_rfsrc](#), [partial.rfsrc](#), [get.partial.plot.data](#)

Examples

```
## -----
## survival
## -----

data(veteran, package = "randomForestSRC")
v.obj <- randomForestSRC::rfsrc(Surv(time,status)~.,
  veteran, nsplit = 10, ntree = 100)

spart <- surv_partial.rfsrc(v.obj, var_list="age", partial.type = "mort")

## partial effect of age on mortality
partial.obj <- partial(v.obj,
  partial.type = "mort",
  partial.xvar = "age",
  partial.values = v.obj$xvar$age,
  partial.time = v.obj$time.interest)
pdta <- get.partial.plot.data(partial.obj)

plot(lowess(pdta$x, pdta$yhat, f = 1/3),
  type = "l", xlab = "age", ylab = "adjusted mortality")

## example where x is discrete - partial effect of age on mortality
## we use the granule=TRUE option
partial.obj <- partial(v.obj,
  partial.type = "mort",
  partial.xvar = "trt",
  partial.values = v.obj$xvar$trt,
  partial.time = v.obj$time.interest)
pdta <- get.partial.plot.data(partial.obj, granule = TRUE)
boxplot(pdta$yhat ~ pdta$x, xlab = "treatment", ylab = "partial effect")

## partial effects of karnofsky score on survival
karno <- quantile(v.obj$xvar$karno)
partial.obj <- partial(v.obj,
  partial.type = "surv",
  partial.xvar = "karno",
  partial.values = karno,
  partial.time = v.obj$time.interest)
```

```

pdta <- get.partial.plot.data(partial.obj)

matplot(pdta$partial.time, t(pdta$yhat), type = "l", lty = 1,
        xlab = "time", ylab = "karnofsky adjusted survival")
legend("topright", legend = paste0("karnofsky = ", karno), fill = 1:5)

## -----
## competing risk
## -----

data(follic, package = "randomForestSRC")
follic.obj <- rfsrc(Surv(time, status) ~ ., follic, nsplit = 3, ntree = 100)

## partial effect of age on years lost
partial.obj <- partial(follic.obj,
                      partial.type = "years.lost",
                      partial.xvar = "age",
                      partial.values = follic.obj$xvar$age,
                      partial.time = follic.obj$time.interest)
pdta1 <- get.partial.plot.data(partial.obj, target = 1)
pdta2 <- get.partial.plot.data(partial.obj, target = 2)

# Save and restore the user's graphical parameters per CRAN policy.
oldpar <- par(no.readonly = TRUE)
on.exit(par(oldpar))
par(mfrow = c(2, 2))
plot(lowess(pdta1$x, pdta1$yhat),
     type = "l", xlab = "age", ylab = "adjusted years lost relapse")
plot(lowess(pdta2$x, pdta2$yhat),
     type = "l", xlab = "age", ylab = "adjusted years lost death")

## partial effect of age on cif
partial.obj <- partial(follic.obj,
                      partial.type = "cif",
                      partial.xvar = "age",
                      partial.values = quantile(follic.obj$xvar$age),
                      partial.time = follic.obj$time.interest)
pdta1 <- get.partial.plot.data(partial.obj, target = 1)
pdta2 <- get.partial.plot.data(partial.obj, target = 2)

matplot(pdta1$partial.time, t(pdta1$yhat), type = "l", lty = 1,
        xlab = "time", ylab = "age adjusted cif for relapse")
matplot(pdta2$partial.time, t(pdta2$yhat), type = "l", lty = 1,
        xlab = "time", ylab = "age adjusted cif for death")

```

varpro_feature_names *Recover original variable names from varpro one-hot encoded feature names*

Description

varpro one-hot encodes factor variables, appending a numeric suffix for each level (e.g., sex becomes sex0 and sex1). This function strips those suffixes iteratively until every name in varpro_names can be matched back to a column in dataset.

Usage

```
varpro_feature_names(varpro_names, dataset)
```

Arguments

varpro_names character vector of names as output by varpro (may include one-hot encoded suffixed names such as "sex0", "sex1")

dataset the original data frame passed to varpro, used to look up valid column names

Value

character vector of unique original variable names (no suffixes)

See Also

[gg_partialpro](#)

Examples

```
## -----
## Simple case: one continuous variable + one binary factor
## -----
ds <- data.frame(age = c(25, 30, 45), sex = c("M", "F", "M"))

# varpro one-hot encodes 'sex' into 'sex0' and 'sex1'
varpro_names <- c("age", "sex0", "sex1")
varpro_feature_names(varpro_names, ds)
# Returns: c("age", "sex")

## -----
## Multi-level factor: three-level 'group' variable
## -----
ds2 <- data.frame(score = 1:6,
                  group = factor(rep(c("A", "B", "C"), 2)))

# varpro appends 0/1/2 for each level
vn2 <- c("score", "group0", "group1", "group2")
varpro_feature_names(vn2, ds2)
# Returns: c("score", "group")

## -----
## Already-clean names pass through unchanged
## -----
ds3 <- data.frame(x = 1:5, y = 1:5)
varpro_feature_names(c("x", "y"), ds3)
```

varpro_feature_names

51

Returns: c("x", "y")

Index

`calc_auc`, [4](#), [5](#), [17](#), [40](#)
`calc_roc`, [4](#), [17](#), [40](#)
`calc_roc(calc_roc.rfsrc)`, [5](#)
`calc_roc.rfsrc`, [5](#)

`factor`, [14](#)

`get.partial.plot.data`, [14](#), [48](#)
`gg_error`, [3](#), [6](#), [28](#), [29](#)
`gg_partial`, [10](#), [11](#), [12](#), [14](#), [32](#), [33](#), [43](#)
`gg_partial_rfsrc`, [10](#), [12](#), [34](#), [35](#), [47](#), [48](#)
`gg_partialpro`, [10](#), [11](#), [33](#), [50](#)
`gg_rfsrc`, [3](#), [36](#), [37](#), [41](#)
`gg_rfsrc(gg_rfsrc.rfsrc)`, [14](#)
`gg_rfsrc.rfsrc`, [14](#)
`gg_roc`, [3–5](#), [39](#), [40](#)
`gg_roc(gg_roc.rfsrc)`, [17](#)
`gg_roc.rfsrc`, [17](#)
`gg_survival`, [3](#), [15](#), [18](#), [26–28](#), [41](#)
`gg_variable`, [3](#), [7](#), [20](#), [42](#), [43](#)
`gg_vimp`, [3](#), [7](#), [23](#), [45](#)
`ggRandomForests-package`, [2](#)

`kaplan`, [19](#), [20](#), [26](#), [41](#)

`nelson`, [19](#), [20](#), [26](#), [27](#), [28](#), [41](#)

`partial.rfsrc`, [12–14](#), [47](#), [48](#)
`plot.gg_error`, [6](#), [7](#), [28](#)
`plot.gg_partial`, [32](#), [35](#)
`plot.gg_partial_rfsrc`, [34](#)
`plot.gg_partialpro`, [33](#)
`plot.gg_rfsrc`, [14](#), [15](#), [36](#)
`plot.gg_roc`, [4](#), [5](#), [17](#), [39](#)
`plot.gg_survival`, [20](#), [26](#), [28](#), [41](#)
`plot.gg_variable`, [20](#), [21](#), [33](#), [42](#)
`plot.gg_vimp`, [23](#), [45](#)
`plot.rfsrc`, [7](#), [29](#)
`plot.variable`, [20](#), [21](#), [43](#)
`predict.rfsrc`, [5](#)

`quantile`, [46](#)
`quantile_pts`, [46](#)

`randomForest`, [5–7](#), [15](#), [17](#), [20](#), [21](#), [23](#), [29](#), [37](#),
[39](#), [40](#)
`rfsrc`, [3](#), [5–7](#), [13–15](#), [17](#), [19–21](#), [23](#), [29](#), [36](#),
[37](#), [39–41](#), [43](#), [45](#), [47](#)

`surv_partial.rfsrc`, [47](#)

`varpro_feature_names`, [11](#), [49](#)
`vimp`, [23](#)
`vimp.rfsrc`, [23](#)