

Package: ggBoostedTrees (via r-universe)

October 22, 2026

Type Package

Title Visually Exploring Boosted Tree Models

Version 0.10.0

Date 2026-10-21

License GPL (>= 3)

Encoding UTF-8

Language en-US

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

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

Description Graphic elements for exploring boosted tree models fit with 'boostmtree' and 'BoostMLR', which implement Friedman's gradient descent boosting with multivariate tree base learners for longitudinal responses. Separates data extraction from rendering so that additional boosting backends can be added without changes to the plotting layer.

Depends R (>= 4.4.0)

Imports boostmtree (>= 2.0.1), ggplot2, rlang, stats

Suggests BoostMLR, covr, knitr, lintr, quarto, testthat, vdiff

Remotes boostmtree=ehrlinger/boostmtree_src/boostmtree@v2.0.8-ccf

VignetteBuilder quarto

SystemRequirements Quarto command line tool
(<<https://github.com/quarto-dev/quarto-cli>>)

Roxygen list(markdown = TRUE)

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

Config/pak/sysreqs cmake libglpk-dev make libicu-dev libuv1-dev libxml2-dev libx11-dev

Repository <https://ehrlinger.r-universe.dev>

Date/Publication 2026-10-21 21:42:25 UTC

RemoteUrl <https://github.com/ehrlinger/ggBoostedTrees>
RemoteRef HEAD
RemoteSha 0f06d194547428007874a5fabacad73cb9ec66b7

Contents

autoplot.boostmtree	2
autoplot.gg_boost_calibration	3
autoplot.gg_boost_effect	4
autoplot.gg_boost_error	5
autoplot.gg_boost_path	6
autoplot.gg_boost_trajectory	7
autoplot.gg_boost_vimp	9
boostmlr_partial	10
gg_boost_calibration	11
gg_boost_effect	13
gg_boost_error	14
gg_boost_path	16
gg_boost_trajectory	17
gg_boost_vimp	18

Index	20
--------------	-----------

autoplot.boostmtree	<i>Plot a boostmtree fit</i>
---------------------	------------------------------

Description

A shortcut to the error trajectory, which is the first thing worth looking at after a fit. Every other figure is reached through its own gg_boost_*() extractor.

Usage

```
## S3 method for class 'boostmtree'  
autoplot(object, ...)
```

Arguments

- object A fitted [boostmtree](#) object grown with cv.flag = TRUE.
- ... Passed to [autoplot.gg_boost_error](#).

Value

A ggplot object, as produced by [autoplot.gg_boost_error](#).

See Also

[gg_boost_error](#), [gg_boost_path](#), [gg_boost_trajectory](#)

Examples

```
sim <- boostmtree::simLong(n = 25, n.time = 4, model = 1)$data.list
fit <- boostmtree::boostmtree(
  x = sim$features, tm = sim$time, id = sim$id, y = sim$y,
  M = 50, cv.flag = TRUE, verbose = FALSE
)
ggplot2::autoplot(fit)
```

autoplot.gg_boost_calibration

Plot a [gg_boost_calibration](#) object

Description

Observed against fitted over follow-up: bin means of the observed values with their intervals, the fitted bin means beside them, and the cohort mean curve when the object carries one.

Usage

```
## S3 method for class 'gg_boost_calibration'
autoplot(object, ci = TRUE, ...)
```

```
## S3 method for class 'gg_boost_calibration'
plot(x, ci = TRUE, ...)
```

Arguments

object	A gg_boost_calibration object.
ci	Logical. Draw 95% to TRUE; bins holding one observation have none.
...	Not used; present for S3 consistency.
x	A gg_boost_calibration object.

Details

Each bin is drawn as a thin horizontal segment spanning its time range at the observed mean. With equal-count bins that segment is the tail-support signal: where observations are sparse, a bin has to stretch further to collect its share, so a long segment late in follow-up means a thin tail.

Observed bin means are points with vertical 95% fitted bin means are a second point series in another shape, joined by a dashed line. Where the two agree bin by bin, the model is calibrated over that stretch of follow-up. The cohort curve, drawn as a heavier line, is the mean predicted curve across

all subjects; it can depart from the bin means late in follow-up when the subjects still being measured differ from the cohort, which is a property of follow-up rather than of the model.

The returned plot carries no theme.

Value

A ggplot object.

See Also

[gg_boost_calibration](#)

Examples

```
sim <- boostmtree::simLong(n = 25, n.time = 4, model = 1)$data.list
fit <- boostmtree::boostmtree(
  x = sim$features, tm = sim$time, id = sim$id, y = sim$y,
  M = 50, verbose = FALSE
)
plot(gg_boost_calibration(fit, pred = predict(fit, x = fit$x)))
```

autoplot.gg_boost_effect

Plot a [gg_boost_effect](#) object

Description

Covariate effect curves, one line per time point, faceted by variable and, for a multi-response fit, by response.

Usage

```
## S3 method for class 'gg_boost_effect'
autoplot(object, ...)

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

Arguments

object	A gg_boost_effect object.
...	Passed to geom_line (continuous covariates) or geom_point (discrete covariates).
x	A gg_boost_effect object.

Details

Time is mapped to colour rather than to a facet because the question this figure answers is how a covariate's effect *changes* across follow-up, and that change is legible only when the curves share one panel. Variables get the facets instead, since their x scales are unrelated.

A continuous covariate is drawn as a line per time point across the covariate grid. A discrete (factor) covariate has one row per level, so a "curve" would connect points along an axis with no ordering; it is drawn instead as a point per level per time point, on an axis labelled with the level names. Because ggplot2 permits only one scale type per aesthetic across all facets, an object mixing continuous and discrete covariates cannot be drawn as a single figure and is refused with an error asking the caller to plot them one at a time.

The returned plot carries no theme.

Value

A ggplot object.

See Also

[gg_boost_effect](#)

Examples

```
sim <- boostmtree::simLong(n = 25, n.time = 4, model = 1)$data.list
fit <- boostmtree::boostmtree(
  x = sim$features, tm = sim$time, id = sim$id, y = sim$y,
  M = 50, verbose = FALSE
)
pp <- boostmtree::partial.plot(
  fit, x.var.names = "x1", output = "data", verbose = FALSE
)
plot(gg_boost_effect(pp))
```

autoplot.gg_boost_error

Plot a [gg_boost_error](#) object

Description

The error path of a boosted multivariate tree fit against the boosting iteration, with the cross-validated optimal iteration marked.

Usage

```
## S3 method for class 'gg_boost_error'
autoplot(object, optimal = TRUE, ...)

## S3 method for class 'gg_boost_error'
plot(x, optimal = TRUE, ...)
```

Arguments

object	A gg_boost_error object.
optimal	Logical. Draw a vertical rule at the optimal iteration. Defaults to TRUE.
...	Passed to geom_line .
x	A gg_boost_error object.

Details

The returned plot carries no theme, so it composes with any ggplot2 theme or scale. Several responses are drawn as facets rather than as coloured series, because the responses do not share a y scale.

Value

A ggplot object.

See Also

[gg_boost_error](#)

Examples

```
sim <- boostmtree::simLong(n = 25, n.time = 4, model = 1)$data.list
fit <- boostmtree::boostmtree(
  x = sim$features, tm = sim$time, id = sim$id, y = sim$y,
  M = 50, cv.flag = TRUE, verbose = FALSE
)
plot(gg_boost_error(fit))
```

autoplot.gg_boost_path

Plot a [gg_boost_path](#) object

Description

The estimated correlation, variance, and smoothing parameter paths of a boosted multivariate tree fit against the boosting iteration.

Usage

```
## S3 method for class 'gg_boost_path'
autoplot(object, ...)

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

Arguments

object	A gg_boost_path object.
...	Passed to geom_line .
x	A gg_boost_path object.

Details

The three parameters live on entirely different scales, so they are always faceted with a free y axis rather than drawn together. With several responses the facet grid is parameter by response.

The returned plot carries no theme.

Value

A ggplot object.

See Also

[gg_boost_path](#)

Examples

```
sim <- boostmtree::simLong(n = 25, n.time = 4, model = 1)$data.list
fit <- boostmtree::boostmtree(
  x = sim$features, tm = sim$time, id = sim$id, y = sim$y,
  M = 50, cv.flag = TRUE, verbose = FALSE
)
plot(gg_boost_path(fit))
```

autoplot.gg_boost_trajectory

Plot a [gg_boost_trajectory](#) object

Description

Observed and fitted subject trajectories over time: one line per subject through the fitted values, with the observed values as points.

Usage

```
## S3 method for class 'gg_boost_trajectory'
autoplot(
  object,
  subset = NULL,
  n_max = 100,
  observed = TRUE,
  alpha = NULL,
  ...
)

## S3 method for class 'gg_boost_trajectory'
plot(x, subset = NULL, n_max = 100, observed = TRUE, alpha = NULL, ...)
```

Arguments

object	A gg_boost_trajectory object.
subset	Character or numeric vector of subject identifiers to keep. NULL (default) keeps all of them.
n_max	Maximum number of subjects to draw. Defaults to 100; Inf draws every subject.
observed	Logical. Draw the observed values as points. Defaults to TRUE, and is ignored when the fit records no observed values.
alpha	Numeric transparency for both layers. NULL (default) computes one from the number of subjects drawn.
...	Passed to geom_line .
x	A gg_boost_trajectory object.

Details

The extractor returns every subject, so thinning happens here and the tidy data frame always represents the whole fit. `subset` names the subjects to keep; `n_max` caps how many are drawn, sampling at random and saying so. Set `n_max = Inf` to draw all of them. Random sampling is not seeded – call `set.seed()` first for a reproducible figure.

Transparency is doing real work in this figure rather than decorating it. A cohort of any size overplots, and partial transparency turns the tangle into something you can read densities off: where many subjects follow the same path the ink accumulates. The default `alpha` therefore falls as the number of subjects drawn rises. Pass `alpha` explicitly to override.

The returned plot carries no theme.

Value

A ggplot object.

See Also

[gg_boost_trajectory](#)

Examples

```
sim <- boostmtree::simLong(n = 25, n.time = 4, model = 1)$data.list
fit <- boostmtree::boostmtree(
  x = sim$features, tm = sim$time, id = sim$id, y = sim$y,
  M = 50, verbose = FALSE
)
plot(gg_boost_trajectory(fit))
```

```
autoplot.gg_boost_vimp
```

Plot a [gg_boost_vimp](#) object

Description

Variable importance as a horizontal bar chart, ordered by importance and faceted by component.

Usage

```
## S3 method for class 'gg_boost_vimp'
autoplot(object, ...)
```

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

Arguments

object	A gg_boost_vimp object.
...	Passed to geom_col .
x	A gg_boost_vimp object.

Details

Bars run horizontally because variable names are text and read better along the y axis than rotated beneath it, and variables are ordered by importance because an importance chart in arbitrary order is close to unreadable.

The x axis is labelled with the `metric` attribute carried by the extracted object. That metric is a property of how importance was computed rather than a fixed quantity, so a hard-coded label would misdescribe some objects.

A negative importance is not an error: with a permutation-style metric a variable that contributes nothing can score slightly below zero by chance.

The returned plot carries no theme.

Value

A ggplot object.

See Also[gg_boost_vimp](#)**Examples**

```

sim <- boostmtree::simLong(n = 25, n.time = 4, model = 1)$data.list
fit <- boostmtree::boostmtree(
  x = sim$features, tm = sim$time, id = sim$id, y = sim$y,
  M = 50, cv.flag = TRUE, verbose = FALSE
)
plot(gg_boost_vimp(boostmtree::vimp.boostmtree(fit)))

```

boostmlr_partial

*Partial effects from a BoostMLR fit***Description**

Compute partial effect surfaces for one or more covariates of a BoostMLR fit, in a form [gg_boost_effect](#) accepts.

Usage

```
boostmlr_partial(object, xvar.names = NULL, ...)
```

Arguments

<code>object</code>	A BoostMLR grow object, as returned by <code>BoostMLR::BoostMLR()</code> .
<code>xvar.names</code>	Character vector of covariate names. Defaults to every covariate in the fit.
<code>...</code>	Passed to <code>BoostMLR::partial.BoostMLR()</code> , for example <code>n.x</code> , <code>n.tm</code> or <code>Mopt</code> .

Details

`BoostMLR::partial.BoostMLR()` takes one covariate per call and returns a bare list with neither the covariate's name nor the responses' names, so it cannot be told apart from any other list. This wrapper calls it once per covariate with plotting switched off, and records the covariate names and the fit's response names (`y_Names`) alongside the results. It is the BoostMLR counterpart of `boostmtree::partial.plot(output = "data")`.

Each result holds the raw partial effect (`pList`) and a lowess smooth of it over time (`sList`), as covariate-by-time matrices, one per response. [gg_boost_effect](#) extracts the smooth.

Value

A `boostmlr_partial` object: a list with elements `curves` (one `partial.BoostMLR()` result per covariate, named by covariate), `x.var.names` and `response.labels`.

See Also[gg_boost_effect](#)**Examples**

```

if (requireNamespace("BoostMLR", quietly = TRUE)) {
  sim <- BoostMLR::simLong(
    n = 20, N = 3, rho = 0.8, model = 1, q_x = 2, q_y = 0
  )$dtaL
  fit <- BoostMLR::BoostMLR(
    x = sim$features, tm = sim$time, id = sim$id, y = sim$y,
    M = 50, Verbose = FALSE
  )
  plot(gg_boost_effect(boostmlr_partial(fit, "x1")))
}

```

gg_boost_calibration *Calibration over follow-up data object*

Description

Compare observed and fitted values over follow-up time, binned by time, with an optional cohort mean curve.

Usage

```
gg_boost_calibration(object, pred = NULL, n_bins = 10, breaks = NULL, ...)
```

Arguments

object	A fitted boostmtree object, or a fitted BoostMLR object. Not a predict object: pass that as pred.
pred	Optional boostmtree predict object from <code>predict(fit, x = ...)</code> , used for the cohort curve.
n_bins	Number of equal-count time bins. Defaults to 10.
breaks	Optional numeric vector of bin edges in the fit's time units. Overrides n_bins.
...	Not used; present for S3 consistency.

Details

A cohort's predicted curves say what the model expects; they cannot say whether it matches the data. This figure answers that. Observations are binned by time, and each bin reports the mean observed value, the mean fitted value at the same observations, and how many observations and distinct subjects stand behind it.

Bins hold roughly equal numbers of observations by default: the edges are quantiles of observed time, computed per response. Tied times, common in clinical data where many measurements share a visit time, merge quantile edges; duplicates are removed and a message says how many bins survived. Supply breaks, in the fit's own time units, to choose the edges yourself. Observations outside the range of breaks are dropped, with a message giving the count.

With equal-count bins, `n_obs` is nearly constant by construction, so it cannot show where data thins out. Two columns can: bin width (`time_hi - time_lo`), which grows where observations are sparse, and `n_subject`, which falls when a late bin rests on repeat measurements of a few subjects.

The interval on observed is a normal approximation that treats observations as independent. Repeated measurements within a subject make it too narrow. Read it as a visual guide, not an inferential claim.

Supplying `pred` adds the cohort curve: the mean over subjects of the predicted curves on a common time grid, attached as the cohort attribute. Make `pred` with `predict(fit, x = ...)` and no `tm` or `id`; with them, each subject is predicted at its own times and a mean across subjects changes composition from one time to the next. The cohort curve is available for `boostmtree` fits only.

This figure accepts either a `boostmtree` or a `BoostMLR` grow fit. It reads its rows from [gg_boost_trajectory](#), so both backends' storage layouts are handled there.

Value

A `gg_boost_calibration` `data.frame` with one row per response and bin, ordered by response then time, with columns:

response Factor naming the response.

bin Factor bin label, ordered by time within each response.

time_lo, time_hi Numeric bin edges.

time Numeric median observed time within the bin.

n_obs Integer count of observations in the bin.

n_subject Integer count of distinct subjects in the bin.

observed Numeric mean observed value.

observed_lo, observed_hi Numeric 95\ NA when the bin holds one observation.

fitted Numeric mean fitted value at the same observations.

When `pred` is supplied, the attribute `cohort` holds a `data.frame` with columns `response` (factor), `time` and `fitted` (numeric).

See Also

[plot.gg_boost_calibration](#), [gg_boost_trajectory](#)

Examples

```
sim <- boostmtree::simLong(n = 25, n.time = 4, model = 1)$data.list
fit <- boostmtree::boostmtree(
  x = sim$features, tm = sim$time, id = sim$id, y = sim$y,
  M = 50, verbose = FALSE
```

```

)
pred <- predict(fit, x = fit$x)
gg <- gg_boost_calibration(fit, pred = pred)
plot(gg)

# The predicted curves themselves, for a look at who drives the tail:
plot(gg_boost_trajectory(pred), n_max = 10)

```

gg_boost_effect	<i>Partial and marginal effect data object</i>
-----------------	--

Description

Extract covariate effect curves over time from a [partial.plot](#) or [marginal.plot](#) object, or from the [boostmlr_partial](#) result for a BoostMLR fit.

Usage

```
gg_boost_effect(object, ...)
```

Arguments

object	A partial.plot.boostmtree or marginal.plot.boostmtree object, as returned by <code>boostmtree::partial.plot()</code> or <code>boostmtree::marginal.plot()</code> with <code>output = "data"</code> , <code>verbose = FALSE</code> , or a boostmlr_partial object.
...	Not used; present for S3 consistency.

Details

The two differ in what they hold constant. A partial effect varies one covariate while averaging over the others; a marginal effect reads the fitted surface as the data actually distribute it. Both are covariate-by-time surfaces, so both land in this one class, distinguished by kind.

`marginal.plot()` returns a raw scatter alongside its smoothed curve. That scatter is not the raw observations: it holds one unsmoothed fitted prediction per subject at the subject's own observed covariate value, not the observed response and not the stored fitted values. It is also not reconstructable from [gg_boost_trajectory](#), which carries no covariate column. `gg_boost_effect` extracts the smoothed curve instead, so that both levels of kind mean the same thing: the fitted effect.

For BoostMLR, [boostmlr_partial](#) returns both the raw partial effect and a lowess smooth of it over time; the smooth is extracted, matching the marginal choice above. The result has kind `partial` and one block per response, labelled from the fit's `y_Names`.

No source computes a confidence interval, so none is reported here.

`boostmtree` accepts factor covariates. For those, `partial.plot()` and `marginal.plot()` return a character (or, for `marginal.plot()` `$data, factor`) `x` column with one row per level rather than a numeric grid. `gg_boost_effect` detects this and maps each level to an integer position in `x`, carrying

the level itself in `x_label`; a continuous covariate keeps its numeric value in `x` and leaves `x_label` NA. The grid is resolved once per variable so every time point shares the same level ordering.

A multi-response fit (`family = "ordinal"` or `"nominal"`) nests `$curves / $smooth` as `[[response]][[variable]]`; a single-response fit flattens the outer level. `gg_boost_effect` accepts both and records the response in a response column, labelled from the object's `$response.labels` ("response" for a single-response fit). Each response of an ordinal or nominal fit is one model component, so its curves are on that component's scale.

Value

A `gg_boost_effect` data.frame with columns:

variable Factor covariate name.

x Numeric covariate value. For a continuous covariate this is the covariate itself; for a discrete (factor) covariate this is an integer position, one per level.

x_label Character level label for a discrete covariate, NA for a continuous one.

time Numeric time point.

estimate Numeric fitted effect.

kind Factor, partial or marginal.

response Factor naming the response.

See Also

[plot.gg_boost_effect](#), [gg_boost_vimp](#), [boostmlr_partial](#)

Examples

```
sim <- boostmtree::simLong(n = 25, n.time = 4, model = 1)$data.list
fit <- boostmtree::boostmtree(
  x = sim$features, tm = sim$time, id = sim$id, y = sim$y,
  M = 50, verbose = FALSE
)
pp <- boostmtree::partial.plot(
  fit, x.var.names = "x1", output = "data", verbose = FALSE
)
plot(gg_boost_effect(pp))
```

gg_boost_error

Boosting error trajectory data object

Description

Extract the error path of a boosted multivariate tree fit as a function of the boosting iteration, together with the optimal iteration selected by cross-validation.

Usage

```
gg_boost_error(object, use.rmse = TRUE, ...)
```

Arguments

<code>object</code>	A fitted <code>boostmtree</code> object, or a fitted BoostMLR object.
<code>use.rmse</code>	Logical. When TRUE (default) return the standardized 12 error; when FALSE return the unstandardized squared error.
<code>...</code>	Not used; present for S3 consistency.

Details

The fit must have been grown with `cv.flag = TRUE`. `boostmtree` records `err.rate` and `m.opt` only when cross-validation ran; a default fit carries neither, and there is no error path to extract. If this function reports that the fit has no error path, refit with `cv.flag = TRUE`.

`boostmtree` stores the error on the standardized response scale in the 12 column. `use.rmse = FALSE` returns $(12 * y.sd)^2$, the squared error on the original scale.

This figure also accepts a BoostMLR `grow` fit (predict objects are not supported – see below). BoostMLR records `Error_Rate` as an M-by-response matrix already on a single scale, so there is no `use.rmse` argument for that backend; `use.rmse = FALSE` is refused, while TRUE, the default, names the scale it already returns. BoostMLR `grow` objects also select no optimal iteration anywhere in the object – `partial.BoostMLR()` takes `Mopt` as a user-supplied argument instead – so `optimal` is FALSE for every row; deriving an `argmin` here would report a choice the backend never made.

A BoostMLR **predict** object is refused outright, even though it shares the "BoostMLR" class and also carries an `Error_Rate`. Its `Error_Rate` is test error rather than the training path documented above, and it records a real `Mopt` that this extractor would otherwise silently discard and overwrite with `optimal = FALSE`.

Value

A `gg_boost_error` data.frame with columns:

iteration Integer boosting iteration, from 1.

value Numeric error at that iteration.

response Factor naming the response.

optimal Logical, TRUE at the cross-validated optimal iteration.

See Also

[plot.gg_boost_error](#), [gg_boost_path](#)

Examples

```
sim <- boostmtree::simLong(n = 25, n.time = 4, model = 1)$data.list
fit <- boostmtree::boostmtree(
  x = sim$features, tm = sim$time, id = sim$id, y = sim$y,
  M = 50, cv.flag = TRUE, verbose = FALSE
```

```
)
gg_dta <- gg_boost_error(fit)
plot(gg_dta)
```

gg_boost_path	<i>Boosting parameter path data object</i>
---------------	--

Description

Extract the estimated correlation, variance, and penalized-spline smoothing parameters of a boosted multivariate tree fit as functions of the boosting iteration.

Usage

```
gg_boost_path(object, parameters = c("rho", "phi", "lambda"), ...)
```

Arguments

<code>object</code>	A fitted <code>boostmtree</code> object, or a fitted BoostMLR object.
<code>parameters</code>	Character vector naming the paths to extract. For a <code>boostmtree</code> fit, any of "rho", "phi", and "lambda", defaulting to all three. For a BoostMLR fit, "rho" and/or "phi" only, defaulting to both.
<code>...</code>	Not used; present for S3 consistency.

Details

`boostmtree` re-estimates three quantities at every boosting iteration: `rho`, the within-subject correlation; `phi`, the variance component; and `lambda`, the P-spline smoothing parameter for the time-covariate interaction. Their paths diagnose whether the variance structure settled or is still drifting when boosting stops.

Unlike the error path, these are recorded on every fit and do not require `cv.flag = TRUE`. A parameter absent from the fit is dropped silently; it is an error only when none of the requested parameters is present.

This figure also accepts a BoostMLR fit, which records only `Rho` and `Phi` as M-by-response matrices – the method's default parameters is `c("rho", "phi")` rather than all three. BoostMLR has no comparable `lambda`: its `Lambda_List` holds per-iteration basis coefficients rather than a scalar smoothing parameter per response, a different quantity, so requesting "lambda" from a BoostMLR fit is refused with that reason instead of silently dropped.

Value

A `gg_boost_path` data.frame with columns:

iteration Integer boosting iteration, from 1.
value Numeric parameter value at that iteration.
parameter Factor naming the parameter.
response Factor naming the response.

See Also

[plot.gg_boost_path](#), [gg_boost_error](#)

Examples

```
sim <- boostmtree::simLong(n = 25, n.time = 4, model = 1)$data.list
fit <- boostmtree::boostmtree(
  x = sim$features, tm = sim$time, id = sim$id, y = sim$y,
  M = 50, cv.flag = TRUE, verbose = FALSE
)
plot(gg_boost_path(fit))
```

gg_boost_trajectory	<i>Subject trajectory data object</i>
---------------------	---------------------------------------

Description

Extract observed and fitted longitudinal trajectories from a boosted multivariate tree fit, one row per observation.

Usage

```
gg_boost_trajectory(object, ...)
```

Arguments

object	A fitted boostmtree object, or a fitted BoostMLR object.
...	Not used; present for S3 consistency.

Details

This is the figure longitudinal boosting exists for: whether the model tracks individual subjects over time, rather than only fitting the population mean.

`boostmtree` stores `time`, `mu` and `y.org` as parallel lists of per-subject vectors, in the order observations were supplied rather than in time order. Rows here are sorted by time within each subject, and the assembled frame is response-major (all subjects for one response, then the next), so a line drawn through one subject's rows follows the trajectory instead of zigzagging. Subject identifiers come from the fit's own `id.unique`, not from a positional index.

`observed` is NA throughout when the fit carries no observed response, which happens for a prediction on new data.

This figure accepts either a `boostmtree` or a BoostMLR fit. BoostMLR stores `mu` and `y` as flat observation-by-response matrices, with `tm` and `id` as parallel flat vectors, rather than `boostmtree`'s nested per-subject lists – entirely different layout for the same information. Response labels come from `y_names`, since `mu` carries no column names. BoostMLR is natively multi-response, so a fit typically yields several response blocks even for a "single" model.

Value

A gg_boost_trajectory data.frame with columns:

id Factor subject identifier, taken from id.unique.

time Numeric observation time, ascending within subject.

fitted Numeric fitted value.

observed Numeric observed value, or NA.

response Factor naming the response.

See Also

[plot.gg_boost_trajectory](#), [gg_boost_error](#)

Examples

```
sim <- boostmtree::simLong(n = 25, n.time = 4, model = 1)$data.list
fit <- boostmtree::boostmtree(
  x = sim$features, tm = sim$time, id = sim$id, y = sim$y,
  M = 50, verbose = FALSE
)
plot(gg_boost_trajectory(fit))
```

gg_boost_vimp	<i>Variable importance data object</i>
---------------	--

Description

Extract variable importance from a [vimp.boostmtree](#) object, or from a BoostMLR predict object, for both the main effect of each covariate and its interaction with time.

Usage

```
gg_boost_vimp(object, components = c("main", "interaction"), ...)
```

Arguments

object	A vimp.boostmtree object, or a BoostMLR predict object made with importance = TRUE.
components	Character vector naming the components to extract, any of "main" and "interaction". Defaults to both.
...	Not used; present for S3 consistency.

Details

boostmtree reports importance in two parts. The main effect asks what is lost by perturbing a covariate; the interaction asks what is lost by perturbing its interaction with time. For a longitudinal model the second is often the interesting one, since it is where a covariate whose effect *changes* over follow-up shows up.

boostmtree labels the interaction rows `x1:time`, `x2:time` and so on. The suffix is stripped here, so a variable carries one label across both components and can be compared or dodged against itself.

The importance metric is a string on the source object rather than a fixed quantity, so it travels as the `metric` attribute and the renderer uses it to label the axis. The overall time effect has no variable to attach to and travels as the `time.effect` attribute.

A joint importance object (`vimp.boostmtree(fit, joint = TRUE)`) reports a single combined value, labelled `joint.vimp`. Note that CRAN `boostmtree` 2.0.0 cannot produce one at all; the patched build this package requires is needed to compute it.

BoostMLR computes importance inside prediction: pass the object returned by `BoostMLR::predictBoostMLR(fit, x, tm, id, y, importance = TRUE)`, whose `$vimp` holds one matrix per response. BoostMLR splits the covariate-time interaction across its overlapping time intervals (columns `Int_Eff.1`, `Int_Eff.2`, ...); these are summed here to one interaction value per covariate so the result has the same shape as for `boostmtree`. The per-interval matrices travel unsummed as the `interaction.by.interval` attribute, one per response. The metric is "Standardized VIMP", and no `time.effect` is recorded. A joint importance from `BoostMLR::vimp.BoostMLR(joint = TRUE)` is a bare list and is not accepted.

Value

A `gg_boost_vimp` data.frame with columns:

variable Factor covariate name.

importance Numeric importance.

component Factor, main or interaction.

response Factor naming the response.

with attributes `metric` (the importance metric) and `time.effect`, plus `interaction.by.interval` for BoostMLR.

See Also

[plot.gg_boost_vimp](#), [gg_boost_effect](#)

Examples

```
sim <- boostmtree::simLong(n = 25, n.time = 4, model = 1)$data.list
fit <- boostmtree::boostmtree(
  x = sim$features, tm = sim$time, id = sim$id, y = sim$y,
  M = 50, cv.flag = TRUE, verbose = FALSE
)
plot(gg_boost_vimp(boostmtree::vimp.boostmtree(fit)))
```

Index

autoplot.boostmtree, [2](#)
autoplot.gg_boost_calibration, [3](#)
autoplot.gg_boost_effect, [4](#)
autoplot.gg_boost_error, [2](#), [5](#)
autoplot.gg_boost_path, [6](#)
autoplot.gg_boost_trajectory, [7](#)
autoplot.gg_boost_vimp, [9](#)

boostmlr_partial, [10](#), [13](#), [14](#)
boostmtree, [2](#), [11](#), [15–17](#)

geom_col, [9](#)
geom_line, [4](#), [6–8](#)
geom_point, [4](#)
gg_boost_calibration, [3](#), [4](#), [11](#)
gg_boost_effect, [4](#), [5](#), [10](#), [11](#), [13](#), [19](#)
gg_boost_error, [3](#), [5](#), [6](#), [14](#), [17](#), [18](#)
gg_boost_path, [3](#), [6](#), [7](#), [15](#), [16](#)
gg_boost_trajectory, [3](#), [7](#), [8](#), [12](#), [13](#), [17](#)
gg_boost_vimp, [9](#), [10](#), [14](#), [18](#)

marginal.plot, [13](#)

partial.plot, [13](#)
plot.gg_boost_calibration, [12](#)
plot.gg_boost_calibration
 ([autoplot.gg_boost_calibration](#)),
 [3](#)
plot.gg_boost_effect, [14](#)
plot.gg_boost_effect
 ([autoplot.gg_boost_effect](#)), [4](#)
plot.gg_boost_error, [15](#)
plot.gg_boost_error
 ([autoplot.gg_boost_error](#)), [5](#)
plot.gg_boost_path, [17](#)
plot.gg_boost_path
 ([autoplot.gg_boost_path](#)), [6](#)
plot.gg_boost_trajectory, [18](#)
plot.gg_boost_trajectory
 ([autoplot.gg_boost_trajectory](#)),
 [7](#)

plot.gg_boost_vimp, [19](#)
plot.gg_boost_vimp
 ([autoplot.gg_boost_vimp](#)), [9](#)

vimp.boostmtree, [18](#)