Package 'boostmtree'

April 10, 2026

Version 2.0.0

Date 2026-04-08

Title Boosted Multivariate Trees for Longitudinal Data

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

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

Depends R (>= 4.3.0)

Imports randomForestSRC (>= 3.5.0), parallel, splines, nlme

Description Implements Friedman's gradient descent boosting algorithm for modeling longitudinal response using multivariate tree base learners. Longitudinal response could be continuous, binary, nominal or ordinal. A time-covariate interaction effect is modeled using penalized B-splines (P-splines) with estimated adaptive smoothing parameter. Although the package is design for longitudinal data, it can handle cross-sectional data as well. Implementation details are provided in Pande et al. (2017), Mach Learn <DOI:10.1007/s10994-016-5597-1>.

License GPL (>= 3)

URL https://ishwaran.org/

NeedsCompilation no

Repository CRAN

Date/Publication 2026-04-10 08:00:27 UTC

Contents

`boostmtree-package` *Boosted multivariate trees for longitudinal data.*

Description

Multivariate boosted tree models for repeated-measures outcomes. The method combines gradient boosting, subject-level tree partitioning, and smooth time effects to estimate subject-specific mean trajectories or class probabilities over time. Supports continuous, binary, nominal, and ordinal longitudinal data, as well as univariate responses.

Package Overview

This package contains many useful functions and users should read the help file in its entirety for details. However, we briefly mention several key functions that may make it easier to navigate and understand the layout of the package.

1. `boostmtree`

   This is the main entry point to the package. It grows a multivariate tree using user supplied training data. Trees are grown using the **randomForestSRC** R-package.

2. `predict.boostmtree` (`predict`)

   Used for prediction. Predicted values are obtained by dropping the user supplied test data down the grow forest. The resulting object has class (`rfsrc`, `predict`).

Author(s)

Amol Pande, Udaya B. Kogalur and Hemant Ishwaran

References

Friedman J.H. (2001). Greedy function approximation: a gradient boosting machine. *The Annals of Statistics*, 29(5):1189–1232.

Friedman J.H. (2002). Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4):367–378.

Pande A., Li L., Rajeswaran J., Ehrlinger J., Kogalur U.B., Blackstone E.H., Ishwaran H. (2017). Boosted multivariate trees for longitudinal data. *Machine Learning*, 106(2):277–305.

Pande A., Ishwaran H., Blackstone E.H., Rajeswaran J., and Gillanov M. (2022). Application of gradient boosting in evaluating surgical ablation for atrial fibrillation. *SN Computer Science*, 3:466.

Pande A., Ishwaran H., and Blackstone E.H. (2022). Boosting for multivariate longitudinal responses. *SN Computer Science*, 3:186.

Pande A. (2017). *Boosting for longitudinal data*. Ph.D. dissertation, Miller School of Medicine, University of Miami.

See Also

boostmtree, boostmtree.control, plot.boostmtree, predict.boostmtree, print.boostmtree, simLong, vimp.boostmtree,

---

AF *Atrial Fibrillation Data*

Description

Atrial Fibrillation (AF) data from a randomized clinical trial examining the effect of surgical ablation as a treatment for AF in patients with persistent or long-standing persistent AF who required mitral valve surgery. Patients were randomized into two groups: mitral valve surgery with ablation and mitral valve surgery without ablation. Patients assigned to the ablation group were further randomized to one of two procedures: pulmonary vein isolation (PVI) or biatrial maze procedure. All patients were followed weekly over a 12-month period. The primary outcome is the presence or absence of AF, measured as a binary longitudinal response. The dataset includes 228 patients, with a total of 7,949 AF measurements and an average of 35 observations per patient.

Format

A list containing four elements:

1. The 84 patient variables (features).
2. Time points (time).
3. Unique patient identifier (id).
4. Presence or absence of AF (y).

References

Gillinov A. M., Gelijns A.C., Parides M.K., DeRose J.J.Jr., Moskowitz~A.J. et al. Surgical ablation of atrial fibrillation during mitral valve surgery.

The New England Journal of Medicine 372(15):1399–1408, 2015.

Examples

```
data(AF, package = "boostmtree")
```

`boostmtree` *Boosted multivariate trees for longitudinal data*

Description

Fit multivariate boosted tree models for repeated-measures outcomes. The method combines gradient boosting, subject-level tree partitioning, and smooth time effects to estimate subject-specific mean trajectories or class probabilities over time. Supports continuous, binary, nominal, and ordinal longitudinal data, as well as univariate responses.

Usage

```
boostmtree(
  x,
  tm = NULL,
  id = NULL,
  y,
  family = c("continuous", "binary", "nominal", "ordinal"),
  y.reference = NULL,
  M = 200,
  nu = 0.05,
  na.action = c("na.omit", "na.impute")[2],
  k = 5,
  mtry = NULL,
  n.knots = 10,
  d = 3,
  pen.ord = 3,
  lambda = NULL,
  rho = NULL,
  lambda.max = 1e6,
  lambda.iter = 2,
  svd.tol = 1e-6,
  verbose = TRUE,
  cv.flag = FALSE,
  eps = 1e-5,
  mod.grad = TRUE,
  nr.iter = 3,
  control = boostmtree.control()
)
```

Arguments

x	A data frame or matrix of covariates. For longitudinal fitting, rows must be repeated so that the covariate row for subject i appears once for each observation time recorded for that subject.

`tm`	A vector of observation times with one entry per row of `x`. If `tm` and `id` are both omitted, the function switches to univariate mode and treats each row of `x` as a single observation.
`id`	A subject identifier with one entry per row of `x`. Must be supplied whenever `tm` is supplied.
`y`	Observed responses with one entry per row of `x`.
`family`	Response type. Choose one of `"continuous"`, `"binary"`, `"nominal"`, or `"ordinal"`.
`y.reference`	Reference level used when `family = "nominal"`. If `NULL`, the first observed response level is used. Ignored for the other families.
`M`	Number of boosting iterations. Larger values allow a more flexible fit, while `cv.flag = TRUE` can be used to select a stopping iteration automatically.
`nu`	Boosting step size. Supply either a scalar or a length-two vector. The first element is used for the intercept part of the time basis and the second for the remaining time-basis coefficients. All values must lie in $(0, 1]$. Smaller values generally require larger `M`.
`na.action`	How to handle missing values in the covariates `x`. Use `"na.omit"` to remove subjects with incomplete covariates, or `"na.impute"` to keep subjects unless all covariates are missing. Missing values in `tm`, `id`, or `y` are not allowed.
`k`	Requested number of terminal nodes in the tree base learner.
`mtry`	Number of candidate covariates sampled at each split. If `NULL`, all covariates are considered.
`n.knots`	Number of internal knots used for the B-spline time basis.
`d`	Degree of the B-spline basis. Values smaller than 1 remove the smooth time component and leave an intercept-only time design.
`pen.ord`	Differencing order used in the P-spline penalty. Larger values encourage smoother time profiles.
`lambda`	Smoothing parameter for the time-basis penalty. If `NULL`, the function estimates `lambda` adaptively when the time basis has sufficient degrees of freedom. A scalar is recycled across subproblems; otherwise the supplied vector must have length `n.q`.
`rho`	Working within-subject correlation parameter. If `NULL`, it is updated during fitting. A scalar is recycled across subproblems; otherwise the supplied vector must have length `n.q`.
`lambda.max`	Upper bound used during adaptive estimation of `lambda`.
`lambda.iter`	Number of updates used when `lambda` is estimated adaptively.
`svd.tol`	Tolerance used in the singular-value decomposition of the penalty matrix.
`verbose`	Logical; should progress output be printed?
`cv.flag`	Logical; should out-of-bag cross-validation be tracked to select an optimal stopping iteration? When `cv.flag = TRUE`, the fit must have out-of-bag subjects at each boosting iteration. The default control setting `control = boostmtree.control()` already provides this through `bootstrap = "by.root"`. If a no-OOB scheme such as `bootstrap = "none"` is requested, `boostmtree()` switches back to `"by.root"` with a warning.

eps	Tolerance used when selecting the stopping iteration from the cross-validation error path.
mod.grad	Logical; controls the pseudo-response used to grow each tree base learner. During tree growing, the working gradient is first computed for each subject and then passed to rfsrc() as the multivariate pseudo-response used to determine tree splits. If mod.grad = FALSE, the inverse of the current working covariance matrix is used to compute the gradient, otherwise the inverse covariance factor is omitted. Operationally, mod.grad changes how candidate splits are evaluated, but it does *not* remove covariance weighting from the later terminal-node coefficient update. After the tree structure has been chosen, the node-specific coefficient estimates are still obtained from score and Hessian updates that use the inverse working covariance matrix.
nr.iter	Number of Newton–Raphson updates used inside the terminal node coefficient solver for non-continuous families.
control	A control object created by boostmtree.control().

Details

boostmtree() is designed for longitudinal data in which subject i is measured at times t_{ij} and is described by a baseline covariate vector x_i. The goal is to estimate how the conditional mean response changes over time and across subjects with different covariate profiles.

At a high level, the method combines three ideas.

First, it uses *gradient boosting*. Starting from an initial fit, the algorithm repeatedly adds small updates in the direction that improves the fit. If $\eta_i(t)$ denotes the current linear predictor, then boosting updates it according to

$$\eta_i^{(m)}(t) = \eta_i^{(m-1)}(t) + \nu b_m(x_i, t),$$

where b_m is the base learner fitted at iteration m and ν is the step size.

Second, the base learner is a *multivariate tree*. The tree partitions subjects into terminal nodes based on their covariates. Within each terminal node, the time profile is represented by coefficients of a smooth basis in time. In this way the model can capture nonlinear covariate effects, interactions, and different trajectory shapes for different groups of subjects.

Third, the time effect is modeled by a *penalized B-spline*. When d >= 1, the observed times are expanded into a spline basis and the terminal-node coefficients for that basis are penalized to avoid unstable or overly wiggly trajectories. The smoothing parameter lambda controls the strength of that penalty. If lambda is not supplied, the function can estimate it adaptively.

The fitted mean is linked to the response scale through

$$g\{\mu_i(t)\} = \eta_i(t),$$

where the link g depends on the family:

- For "continuous", the identity link is used and the method models the mean trajectory directly.
- For "binary", the logit link is used so that the fitted values represent estimated probabilities for the non-reference class.

- For "nominal", the response is decomposed into one-vs-reference logit submodels, one for each non-reference class. The final class probabilities are then reconstructed from those submodels.
- For "ordinal", the response is modeled through cumulative logit submodels. The final class probabilities are recovered from the fitted cumulative probabilities, with a monotonicity correction applied so that the fitted cumulative curves remain ordered.

For longitudinal data, boostmtree uses a working covariance model to describe the dependence among repeated measurements from the same subject after the mean trajectory has been fitted. For subject i with n_i observations, and for response component q, the working covariance has compound-symmetry form

$$V_{iq} = \phi_q\{(1-\rho_q)I_{n_i} + \rho_q J_{n_i}\},$$

where I_{n_i} is the $n_i \times n_i$ identity matrix and J_{n_i} is the $n_i \times n_i$ matrix of ones.

The roles of phi and rho are different:

- phi is a scale parameter. It controls the overall size of the residual variation. Larger values of phi mean that, after accounting for the fitted mean structure, the repeated measurements still show greater variability.
- rho is a within-subject correlation parameter. It controls how strongly repeated measurements from the same subject move together after accounting for the fitted mean structure. When rho = 0, the working residuals are uncorrelated within subject. Positive values indicate positive within-subject association.

Thus, phi answers the question "how much variation remains?", whereas rho answers the question "how is that remaining variation shared across measurements from the same subject?".

These parameters should be distinguished from lambda. The parameter lambda penalizes roughness in the time-basis coefficients, whereas phi and rho describe the residual covariance structure.

For continuous families there is a single pair (ϕ, ρ). For the binary, nominal, and ordinal families, the method is built from one or more working submodels, so the fitted object may contain a separate path (ϕ_q, ρ_q) for each response component q.

Out-of-bag tracking via cv.flag = TRUE records a standardized error path and selects an optimal stopping iteration m.opt. This package uses *out-of-bag* subjects in two places: for the cross-validation path stored when cv.flag = TRUE, and for grow-object variable importance returned by vimp.boostmtree(). The default control object uses bootstrap = "by.root", which naturally creates out-of-bag subjects. By contrast, bootstrap = "none" produces a deterministic in-bag fit with no out-of-bag data. That can still be useful for ordinary fitting and for prediction-object analyses based on a separate test set, but it is not suitable for out-of-bag CV or grow-object variable importance.

Note that when cv.flag = TRUE, boostmtree() checks the requested resampling rule before fitting. If the requested control object would yield no out-of-bag subjects, the function either adjusts the resampling rule to the default out-of-bag scheme or stops with a clear message when a user-supplied sample matrix is incompatible with OOB CV.

When tm and id are omitted, the function switches to univariate mode. In that case there is no longitudinal time basis, and the method acts as a boosted tree model for a single response per row.

Value

An object of class `c("boostmtree", "grow", ...)` with components:

base.learner Fitted tree base learners, indexed by boosted subproblem and boosting iteration.

control The normalized control object used for fitting.

cv.flag Logical; whether cross-validation tracking was requested.

d Degree of the time basis used in the fitted model.

err.rate Standardized cross-validation error summaries. For single-response fits this is typically a matrix with columns `"l1"` and `"l2"`; for multi-subproblem fits it is stored by subproblem. `NULL` when `cv.flag = FALSE`.

family The fitted response family.

gamma Terminal-node coefficient summaries by subproblem and boosting iteration.

gamma.i.list Cross-validated terminal-node coefficient summaries when `cv.flag = TRUE`; otherwise `NULL`.

id The sorted long-format subject identifier.

id.unique Unique subject identifiers in subject order.

k Requested number of terminal nodes.

lambda Path of smoothing-parameter values across boosting iterations and subproblems, or `NULL` in univariate mode.

M Number of boosting iterations.

m.opt Selected stopping iteration for each subproblem when `cv.flag = TRUE`; otherwise `NULL`.

membership Terminal-node memberships by subproblem and boosting iteration.

mu Fitted mean trajectories. For continuous and binary families this is a subject-level list of fitted trajectories. For nominal and ordinal families this is indexed first by boosted subproblem and then by subject.

n Number of subjects.

n.q Number of boosted subproblems.

na.action Missing-data rule applied to `x`.

ni Number of observations per subject.

ntree Number of trees requested through `control`. The current implementation requires `ntree = 1`.

nu Boosting step-size vector used internally.

oob.available Logical flag indicating whether the fitted resampling path produced out-of-bag subjects at each boosting iteration.

oob.subject.count Number of out-of-bag subjects at each boosting iteration. For single-response fits this is typically a vector of length `M`; for multi-subproblem fits it may be stored by subproblem.

pen.ord Differencing order used in the P-spline penalty.

phi Estimated path of the residual scale parameter. In the working covariance model, `phi` determines the marginal variance of repeated measurements after accounting for the fitted mean structure.

prob.class Class probabilities by subject for non-continuous families; NULL for the continuous family.

q.set Threshold levels (ordinal) or non-reference levels (binary or nominal) used to define the boosted subproblems.

q.total Total number of observed response levels for non-continuous families.

rho Estimated path of the within-subject correlation parameter. In the working covariance model, rho determines how strongly repeated measurements from the same subject are associated after accounting for the fitted mean structure. Under compound symmetry, the off-diagonal covariance is $\rho\phi$.

rmse Standardized root mean squared error evaluated at m.opt, or NULL when cv.flag = FALSE.

time A list of time vectors, one per subject.

time.design Subject-specific time-design matrices.

time.unique Sorted unique observation times used to build the time basis.

univariate Logical; whether the model was fit in univariate mode.

x Subject-level covariate data with one row per subject.

x.tm Time-basis design matrix defined on time.unique.

x.var.names Covariate names.

y Observed responses split by subject.

y.levels Observed response levels for non-continuous families. NA for the continuous family.

y.mean Overall response mean used for standardization.

y.org Observed responses represented at the boosted-subproblem level. For non-continuous families this is the encoded binary or cumulative response used by the corresponding submodel.

y.reference Reference response level for the nominal family. NULL otherwise.

y.sd Overall response standard deviation used for standardization.

Author(s)

Amol Pande, Udaya B. Kogalur and Hemant Ishwaran

References

Friedman J.H. (2001). Greedy function approximation: a gradient boosting machine. *The Annals of Statistics*, 29(5):1189–1232.

Friedman J.H. (2002). Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4):367–378.

Pande A., Li L., Rajeswaran J., Ehrlinger J., Kogalur U.B., Blackstone E.H., Ishwaran H. (2017). Boosted multivariate trees for longitudinal data. *Machine Learning*, 106(2):277–305.

Pande A., Ishwaran H., Blackstone E.H., Rajeswaran J., and Gillanov M. (2022). Application of gradient boosting in evaluating surgical ablation for atrial fibrillation. *SN Computer Science*, 3:466.

Pande A., Ishwaran H., and Blackstone E.H. (2022). Boosting for multivariate longitudinal responses. *SN Computer Science*, 3:186.

Pande A. (2017). *Boosting for longitudinal data*. Ph.D. dissertation, Miller School of Medicine, University of Miami.

See Also

AF, boostmtree.control, marginal.plot.boostmtree, partial.plot.boostmtree, plot.boostmtree, plot.vimp.boostmtree, predict.boostmtree, print.boostmtree, simLong, spirometry, vimp.boostmtree

Examples

```
## ------------------------------------------------------------
## Simulated continuous longitudinal response.
## Use fixed lambda and rho for a fast, example run.
## ------------------------------------------------------------

set.seed(7)
sim.obj <- simLong(n = 20, n.time = 4, rho = 0.5, model = 1,
                   family = "continuous")
dta <- sim.obj$data.list

fit <- boostmtree(
  x = dta$features,
  tm = dta$time,
  id = dta$id,
  y = dta$y,
  family = "continuous",
  M = 10,
  lambda = 0,
  rho = 0,
  verbose = FALSE,
  control = boostmtree.control(seed = 7)
)

print(fit)
plot.data <- plot(fit, output = "data")
str(plot.data[[1]], max.level = 1)

## ------------------------------------------------------------
## Univariate regression: omit tm and id.
## ------------------------------------------------------------

x.uni <- mtcars[, setdiff(names(mtcars), "mpg")]
y.uni <- mtcars$mpg

fit.uni <- boostmtree(
  x = x.uni,
  y = y.uni,
  family = "continuous",
  M = 10,
  lambda = 0,
  verbose = FALSE
)

print(fit.uni)
```

```
## ------------------------------------------------------------
## Simulated binary longitudinal response.
## This example uses a fast, deterministic settings.
## ------------------------------------------------------------

set.seed(17)
sim.bin <- simLong(n = 20, n.time = 4, rho = 0.5, model = 2,
                  family = "binary")
dta.bin <- sim.bin$data.list

fit.bin <- boostmtree(
  x = dta.bin$features,
  tm = dta.bin$time,
  id = dta.bin$id,
  y = dta.bin$y,
  family = "binary",
  M = 10,
  lambda = 0,
  rho = 0,
  verbose = FALSE
)

print(fit.bin)
plot(fit.bin)

## ------------------------------------------------------------
## AF data illustration.  Parameters selected for a fast run.
## ------------------------------------------------------------

data(AF, package = "boostmtree")

fit.af <- boostmtree(
  x = AF$features,
  tm = AF$time,
  id = AF$id,
  y = AF$y,
  family = "binary",
  M = 10,
  k = 15,
  nu = .025,
  n.knots = 15,
  verbose = TRUE
)


print(fit.af)


## ------------------------------------------------------------
## Ordinal response illustration.
## A latent continuous response is grouped into three ordered levels.
## ------------------------------------------------------------
```

```
set.seed(31)
sim.ord <- simLong(n = 25, n.time = 4, family = "continuous")
dta.ord <- sim.ord$data.list
rank.y <- rank(dta.ord$y, ties.method = "first")
brks <- c(0, floor(length(rank.y) / 3), floor(2 * length(rank.y) / 3), length(rank.y))
y.ord <- cut(
  rank.y,
  breaks = brks,
  labels = c(0, 1, 2),
  include.lowest = TRUE,
  ordered_result = TRUE
)

fit.ord <- boostmtree(
  x = dta.ord$features,
  tm = dta.ord$time,
  id = dta.ord$id,
  y = y.ord,
  family = "ordinal",
  M = 50,
  verbose = FALSE
)

print(fit.ord)
```

`boostmtree.control` *Create a control object for* `boostmtree`

Description

Construct a control object used by `boostmtree()` to manage resampling, split search, reproducibility, and a few advanced fitting options.

Usage

```
boostmtree.control(
  ntree = 1,
  bootstrap = "by.root",
  bootstrap.fraction = 0.632,
  sample.matrix = NULL,
  nsplit = NULL,
  samptype = "swor",
  xvar.wt = NULL,
  case.wt = NULL,
  seed = NULL,
  cv.lambda = FALSE,
  cv.rho = TRUE
)
```

Arguments

ntree	Number of trees requested. The current implementation supports ntree = 1 only.
bootstrap	Resampling rule passed to the underlying tree learner. Recognized values are "by.root", "by.user", and "none". The default "by.root" creates out-of-bag subjects and is the recommended choice whenever cv.flag = TRUE or grow-object variable importance will be used.
bootstrap.fraction	Sampling fraction used to build the in-bag sample when bootstrap = "by.user" and sample.matrix is not supplied.
sample.matrix	Optional user-supplied in-bag sampling matrix with dimensions n x M. Entry [i, m] gives the in-bag count for subject i at boosting iteration m. When this is used together with cv.flag = TRUE, every column must contain at least one zero so that out-of-bag subjects are available.
nsplit	Optional limit on the number of candidate split points examined at each node. Smaller values can reduce computation.
samptype	Sampling type passed to the underlying tree learner.
xvar.wt	Optional split-variable weights.
case.wt	Optional case weights.
seed	Optional integer seed used to reproduce the random parts of tree fitting. This affects reproducibility only; it does *not* turn out-of-bag sampling on or off.
cv.lambda	Logical; should the CV path use the cross-validated mean when updating lambda?
cv.rho	Logical; should the CV path use the cross-validated mean when updating rho and phi?

Details

The control object separates routine model arguments such as M, k, and nu from lower-level options that control how the tree base learner is grown.

The most important option is bootstrap. In this package, out-of-bag subjects are used for the cross-validation path stored when cv.flag = TRUE, and for grow-object variable importance returned by vimp.boostmtree(). The default bootstrap = "by.root" is therefore usually the right choice. Setting bootstrap = "none" disables OOB sampling. That can be useful for deterministic in-bag fits and for workflows that rely on a separate held-out test set, but it is not suitable for OOB CV.

When cv.flag = TRUE, boostmtree() checks the requested control object before fitting. If the requested resampling rule would yield no out-of-bag subjects, the fit is adjusted back to the default OOB rule or the function stops with a clear message when a user-supplied sampling matrix is not compatible with OOB CV.

Value

An object of class "boostmtree.control".

See Also

boostmtree

Examples

```
## Default OOB-producing control object.
boostmtree.control(seed = 7)

## Deterministic in-bag fitting with no OOB subjects.
boostmtree.control(bootstrap = "none", seed = 7)
```

`boostmtree.news` *Show the NEWS file*

Description

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

Usage

```
boostmtree.news(...)
```

Arguments

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

Value

None.

Author(s)

Amol Pande, Udaya B. Kogalur and Hemant Ishwaran

`marginal.plot` *Marginal fitted-response summaries for boostmtree models*

Description

Create marginal fitted-response summaries for one or more covariates from a fitted `boostmtree` object. The function can draw the resulting curves, write them to a PDF file, or return the raw and smoothed curve data.

Usage

```
marginal.plot(object, ...)

## S3 method for class 'boostmtree'
marginal.plot(
  object,
  M = NULL,
  x.var.names = NULL,
  time.points = NULL,
  subset = NULL,
  prob.class = FALSE,
  response.labels = NULL,
  output = c("plot", "data", "pdf"),
  file = NULL,
  verbose = TRUE,
  ...
)

## S3 method for class 'marginal.plot.boostmtree'
plot(
  x,
  output = c("plot", "pdf"),
  file = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

`object`	A fitted object of class (`boostmtree`, `grow`).
`M`	Number of boosting iterations to use. By default the function uses the fitted stopping value when available and otherwise uses the full fitted path.
`x.var.names`	Names of the covariates for which marginal summaries are requested. By default all fitted covariates are used.
`time.points`	Time points at which the fitted curves should be displayed. By default the deciles of the observed time values are used. Requested values are matched to the nearest observed training times.
`subset`	Optional vector selecting the fitted subjects to be used in the summary.
`prob.class`	Logical flag used only for ordinal fits. When `TRUE`, the function works with class probabilities rather than cumulative probabilities.
`response.labels`	Optional character vector selecting which response components to include in the summary. For continuous fits the only available label is `"response"`. For binary, nominal, and ordinal fits the available labels are taken from the fitted object.

output	Either "plot" to draw the plots on the active graphics device, "data" to return the computed summary object without drawing it, or "pdf" to write the plots to a PDF file.
file	Optional file name used when output = "pdf" or plot(..., output = "pdf") is requested.
verbose	Should the file location be reported when PDF output is requested?
x	An object returned by marginal.plot.
...	Further arguments passed to predict.boostmtree.

Details

Marginal plots display the relationship between a covariate and the fitted response without averaging over modified covariate values. The fitted mean response is first obtained on a common time grid from predict.boostmtree, and then the association between the chosen covariate and that fitted response is summarized separately at each requested time point.

Compared with partial dependence, this is a less adjusted summary because the other covariates are not perturbed. It is often quicker to compute and is useful for seeing the broad shape of the fitted longitudinal response surface. When multiple response components are available, response.labels can be used to restrict the display to a chosen subset.

Value

If output = "data", an object of class "marginal.plot.boostmtree" with components:

data Raw x-versus-fitted-response pairs for each requested time point.

smooth Smoothed marginal curves used for plotting.

time.points The observed time points used in the summary.

x.var.names Covariate names included in the summary.

response.labels Selected labels for the response component(s) included in the summary.

If output = "plot" or "pdf", the same object is returned invisibly after plotting.

References

Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, 29, 1189–1232.

Examples

```
## ------------------------------------------------------------
## Continuous longitudinal marginal summaries.
## ------------------------------------------------------------
set.seed(19)
sim.obj <- simLong(n = 40, n.time = 4, model = 2, family = "continuous")
dta <- sim.obj$data.list

fit <- boostmtree(
  x = dta$features,
```

```
  tm = dta$time,
  id = dta$id,
  y = dta$y,
  family = "continuous",
  M = 50,
  verbose = FALSE
)

## Draw directly on the active device.
marginal.plot(fit, x.var.names = c("x1", "x2"))

## Return the smoothed marginal curves.
mp <- marginal.plot(
  fit,
  x.var.names = "x2",
  output = "data"
)

str(mp$smooth)

## ------------------------------------------------------------
## Ordinal illustration using class probabilities.
## ------------------------------------------------------------

fit.ord <- boostmtree(
  x = dta$features,
  tm = dta$time,
  id = dta$id,
  y = cut(dta$y, breaks = 3, labels = c("low", "mid", "high")),
  family = "ordinal",
  M = 25,
  verbose = FALSE
)

marginal.plot(
  fit.ord,
  x.var.names = "x1",
  prob.class = TRUE,
  response.labels = c("low", "high")
)
```

`partial.plot` *Partial-dependence summaries for fitted boostmtree models*

Description

Create partial-dependence summaries for one or more covariates from a fitted `boostmtree` object. The function can either draw the plots immediately, write them to a PDF file, or return the computed curve data for further use.

Usage

```
partial.plot(object, ...)

## S3 method for class 'boostmtree'
partial.plot(
  object,
  M = NULL,
  x.var.names = NULL,
  time.points = NULL,
  x.var.values = NULL,
  n.points = 25,
  subset = NULL,
  prob.class = FALSE,
  response.labels = NULL,
  conditional.x.var.names = NULL,
  conditional.values = NULL,
  output = c("plot", "data", "pdf"),
  file = NULL,
  verbose = TRUE,
  use.cv.flag = NULL,
  ...
)

## S3 method for class 'partial.plot.boostmtree'
plot(
  x,
  output = c("plot", "pdf"),
  file = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

object	A fitted object of class (boostmtree, grow).
M	Number of boosting iterations to use. By default the function uses the fitted stopping value when available and otherwise uses the full fitted path.
x.var.names	Names of the covariates for which partial-dependence summaries are requested. By default all fitted covariates are used.
time.points	Time points at which the fitted curves should be displayed. By default the deciles of the observed time values are used. Requested values are matched to the nearest observed training times.
x.var.values	Optional named list giving the covariate values at which each partial-dependence curve should be evaluated. When omitted, the function uses evenly spaced observed values.
n.points	Maximum number of covariate values used for each partial-dependence curve when x.var.values is not supplied.

`subset` Optional vector selecting the fitted subjects to be used when averaging the partial-dependence curves.

`prob.class` Logical flag used only for ordinal fits. When `TRUE`, the function works with class probabilities rather than cumulative probabilities.

`response.labels` Optional character vector selecting which response components to include in the summary. For continuous fits the only available label is `"response"`. For binary, nominal, and ordinal fits the available labels are taken from the fitted object.

`conditional.x.var.names` Optional names of additional covariates whose values should be held fixed while the partial-dependence curves are computed.

`conditional.values` Values corresponding to `conditional.x.var.names`.

`output` Either `"plot"` to draw the plots on the active graphics device, `"data"` to return the computed curve object without drawing it, or `"pdf"` to write the plots to a PDF file.

`file` Optional file name used when `output = "pdf"` or `plot(..., output = "pdf")` is requested.

`verbose` Should the file location be reported when PDF output is requested?

`use.cv.flag` Should the partial-dependence curves be based on the OOB/CV coefficient path? The default is to use the fitted object setting. This option is only meaningful when the model was fitted with `cv.flag = TRUE`; for partial plots it is supported only when all fitted subjects are used in their original order.

`x` An object returned by `partial.plot`.

`...` Further arguments passed to `predict.boostmtree`.

Details

Partial-dependence plots show how the fitted mean response changes as one covariate is varied over a grid of values while the remaining covariates are held fixed at their observed values. In a longitudinal model this produces a family of curves, one for each requested time point.

For each chosen covariate value, the function constructs a modified subject-level covariate data set, obtains fitted values from `predict.boostmtree`, and then averages those fitted values over subjects at the selected times. This gives an adjusted view of the fitted effect of the chosen covariate, in the sense of Friedman (2001), while retaining the time-varying structure of the longitudinal fit.

The returned object stores the computed curve data and can be plotted later with `plot()`. For families with multiple boosted subproblems or multiple class probabilities, `response.labels` can be used to restrict attention to a subset of response components.

Value

If `output = "data"`, an object of class `"partial.plot.boostmtree"` with components:

`curves` Partial-dependence curves. For continuous and binary fits this is a named list indexed by covariate name. For nominal and ordinal fits it is a list indexed first by response component and then by covariate.

`time.points` The observed time points used in the summary.

`x.var.names` Covariate names included in the summary.

`response.labels` Selected labels for the response component(s) included in the summary.

If `output = "plot"` or `"pdf"`, the same object is returned invisibly after plotting.

References

Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, 29, 1189–1232.

Examples

```
## ------------------------------------------------------------
## Continuous longitudinal partial dependence.
## ------------------------------------------------------------
set.seed(19)
sim.obj <- simLong(n = 40, n.time = 4, model = 2, family = "continuous")
dta <- sim.obj$data.list

fit <- boostmtree(
  x = dta$features,
  tm = dta$time,
  id = dta$id,
  y = dta$y,
  family = "continuous",
  M = 50,
  verbose = FALSE
)

## Draw directly on the active device.
partial.plot(fit, x.var.names = c("x1", "x2"))

## Return the underlying curve data.
pp <- partial.plot(
  fit,
  x.var.names = "x2",
  output = "data"
)

str(pp$curves)

## ------------------------------------------------------------
## Ordinal illustration using class probabilities.
## ------------------------------------------------------------

fit.ord <- boostmtree(
  x = dta$features,
  tm = dta$time,
  id = dta$id,
  y = cut(dta$y, breaks = 3, labels = c("low", "mid", "high")),
  family = "ordinal",
```

```
  M = 25,
  verbose = FALSE
)

partial.plot(
  fit.ord,
  x.var.names = "x1",
  prob.class = TRUE,
  response.labels = c("low", "high")
)
```

`plot.boostmtree` *Plot fitted trajectories and diagnostic summaries for a boostmtree object*

Description

Display fitted trajectories, error paths, and related diagnostics for objects produced by `boostmtree()` or compatible prediction methods.

Usage

```
## S3 method for class 'boostmtree'
plot(
  x,
  use.rmse = TRUE,
  output = c("plot", "data", "pdf"),
  file = NULL,
  width = 10,
  height = 10,
  verbose = TRUE,
  ...
)
```

Arguments

`x`	An object of class `c("boostmtree", "grow", ...)` or `c("boostmtree", "predict", ...)`.
`use.rmse`	Logical; if `TRUE`, error paths are shown on the standardized RMSE scale. If `FALSE`, the returned or plotted error path is converted to mean squared error.
`output`	How the result should be returned. The default, `"plot"`, draws directly on the active graphics device. Use `"data"` to return the plot data without drawing, or `"pdf"` to write the plot to a PDF file.
`file`	File path used when `output = "pdf"`. Ignored otherwise.
`width`	Width of the PDF device in inches when `output = "pdf"`.
`height`	Height of the PDF device in inches when `output = "pdf"`.

verbose	Logical; when TRUE and output = "pdf", print the location of the written PDF file.
...	Currently ignored. Included for S3 compatibility.

Details

The plot method is intended to give a quick visual summary of a fitted boostmtree object.

For a grow object, the panels typically show some combination of:

- fitted trajectories over time,
- residual trajectories,
- observed-versus-fitted summaries,
- the cross-validation error path when cv.flag = TRUE, and
- the iteration paths for the nuisance parameters rho, phi, and lambda.

For a prediction object, the same plotting framework is used when stored test errors or variable-importance summaries are available.

Unlike the original implementation, the default behavior is now to draw on the active graphics device. A PDF file is created only when output = "pdf" is requested explicitly.

When output = "data", the method returns the numerical information used to make the plot. This is useful if you want to inspect the fitted curves or construct your own custom graphics.

Value

If output = "data", a list of plot-data objects is returned, one element per boosted subproblem. Each element can contain components such as time.by.subject, fitted.by.subject, observed.by.subject, error.path, rho.path, phi.path, lambda.path, and m.opt. Otherwise the same list is returned invisibly after plotting or writing the PDF file.

References

Pande A., Li L., Rajeswaran J., Ehrlinger J., Kogalur U.B., Blackstone E.H., Ishwaran H. (2017). Boosted multivariate trees for longitudinal data. *Machine Learning*, 106(2):277–305.

Pande A., Ishwaran H., Blackstone E.H., Rajeswaran J., and Gillanov M. (2022). Application of gradient boosting in evaluating surgical ablation for atrial fibrillation. *SN Computer Science*, 3:466.

Pande A., Ishwaran H., and Blackstone E.H. (2022). Boosting for multivariate longitudinal responses. *SN Computer Science*, 3:186.

See Also

boostmtree, print.boostmtree, simLong

Examples

```
## Fit a small model and draw the default summary plot.
set.seed(23)
sim.obj <- simLong(n = 15, n.time = 4, family = "continuous")
dta <- sim.obj$data.list

fit <- boostmtree(
  x = dta$features,
  tm = dta$time,
  id = dta$id,
  y = dta$y,
  family = "continuous",
  M = 8,
  verbose = FALSE
)

plot(fit)

## Return the plot data instead of drawing.
plot.data <- plot(fit, output = "data")
names(plot.data[[1]])


## Write the plot to a PDF only when requested explicitly.
out.file <- tempfile(fileext = ".pdf")
plot(fit, output = "pdf", file = out.file, verbose = FALSE)
```

---

plot.vimp.boostmtree *Plot Method for Variable Importance Objects*

Description

Plot variable-importance objects returned by `vimp.boostmtree()`.

Usage

```
## S3 method for class 'vimp.boostmtree'
plot(
  x,
  show.interaction = TRUE,
  show.time.effect = TRUE,
  output = c("plot", "data", "pdf"),
  file = NULL,
  main = "Variable importance (%)",
  col = grey(0.80),
  cex.names = 0.8,
  eps = 0.1,
  ...
)
```

Arguments

x	An object of class vimp.boostmtree.
show.interaction	Logical value indicating whether time-interaction effects should be displayed when available.
show.time.effect	Logical value indicating whether the time-only effect should be reported in the plot margin when available.
output	One of "plot", "data", or "pdf". The default draws on the active graphics device.
file	File name used when output = "pdf".
main	Main title for the plot.
col	Bar color.
cex.names	Text magnification for variable names on the x-axis.
eps	Small amount added above the plotted range.
...	Additional arguments passed to barplot().

Details

The default behavior is to draw the variable-importance plot on the active graphics device.

For longitudinal fits, the plot shows main covariate effects above the x-axis and time-interaction effects below the x-axis. When available, the time-only effect is reported in the plot margin.

When output = "data", the function returns the plotting data instead of drawing anything. This can be useful if you want to customize the display.

Value

If output = "plot" or output = "pdf", the function returns the input object invisibly.

If output = "data", the function returns a list containing the plotting data for each response component.

Examples

```
set.seed(19)
sim.obj <- simLong(n = 15, n.time = 4, model = 1, family = "continuous")
dta <- sim.obj$data.list

fit <- boostmtree(
  x = dta$features,
  tm = dta$time,
  id = dta$id,
  y = dta$y,
  family = "continuous",
  M = 5,
  cv.flag = TRUE,
  verbose = FALSE,
```

```
  control = boostmtree.control(seed = 19)
)

vimp.obj <- vimp.boostmtree(fit, x.names = c("x1", "x2"))
plot(vimp.obj)

plot.data <- plot(vimp.obj, output = "data")
str(plot.data, max.level = 1)
```

`predict.boostmtree` *Predict longitudinal trajectories from a fitted boostmtree model*

Description

Generate fitted or predicted trajectories from a `boostmtree` fit. The method can return predictions for the original training subjects, for a new longitudinal test set, or for new subjects evaluated on a common time grid.

Usage

```
## S3 method for class 'boostmtree'
predict(
  object,
  x,
  tm,
  id,
  y,
  M = NULL,
  eps = 1e-5,
  use.cv.flag = FALSE,
  partial = FALSE,
  ...
)
```

Arguments

`object`	A fitted object returned by `boostmtree`.
`x`	New covariate values. Supply either one row per subject or a long-format data frame with one row per observation. If omitted, the training subjects stored in `object` are used.
`tm`	Observation times for the prediction data. When `x`, `tm`, and `id` are supplied together, predictions are made at the observed times in the new data. When `x` is supplied without `tm` and `id`, each row of `x` is treated as a new subject and predictions are returned on the training time grid.
`id`	Subject identifier for long-format prediction data.

y	Observed responses for the prediction data. When supplied with long-format prediction data, the method computes standardized test-set error summaries and, if M = NULL, chooses the stopping iteration by minimizing the test-set error path.
M	Optional fixed boosting iteration used for prediction. If NULL and y is supplied, the stopping iteration is selected by minimizing the test-set error path. If NULL and y is not supplied, the stored m.opt from the fitted object is used when available; otherwise the full fitted path is used.
eps	Tolerance used when selecting m.opt from the test-set error path.
use.cv.flag	Logical; should the prediction use the stored out-of-bag coefficient estimates from a model fit with cv.flag = TRUE? This option is intended for predictions on the original training subjects and is ignored for new prediction data.
partial	Logical; if TRUE, interpret x as one row per subject and tm as a common time grid at which each subject should be evaluated.
...	Currently ignored. Included for S3 compatibility.

Details

For longitudinal prediction on a new data set, supply x, tm, and id in long format, with the same subject-level covariates used in the training fit. The function collapses the covariates to one row per subject and then uses the stored terminal-node coefficient path to reconstruct the predicted trajectory.

There are three common use cases.

First, if x is omitted, the method returns fitted trajectories for the training subjects. This is useful for inspecting the stored fit or for producing training-set plots.

Second, if x is supplied without tm and id, each row of x is treated as a new subject and predictions are returned on the training time grid stored in object$time.unique. This is convenient when the user wants an entire fitted profile for new subjects.

Third, if partial = TRUE, x must contain one row per subject and tm supplies a common prediction grid. This returns fitted profiles on that user-specified grid. The method uses the fitted time basis from the training model, so predictions are meaningful when the supplied grid remains within or close to the observed training-time range.

For binary and nominal families, prediction proceeds through one-vs-reference submodels; for ordinal families, prediction proceeds through cumulative submodels followed by a monotonicity correction across thresholds. The returned mu component stores the predicted mean path for each boosted subproblem, and prob.class converts these to class probabilities for the original response scale.

When y is supplied for the prediction data, the method computes standardized RMSE along the prediction path. If M = NULL, the method selects m.opt by minimizing the test-set RMSE, using the same tolerance rule as in model fitting.

Value

An object of class c("boostmtree", "predict", ...) with components:

base.learner Stored tree learners from the fitted object.

boost.obj The fitted training object with large internal fitting components removed.

df.time.design Number of columns in the time-design matrices.

err.rate Standardized test-set error summaries. For single-response fits this is typically a matrix with columns `"l1"` and `"l2"`; for multi-subproblem fits it is stored by subproblem. `NULL` when prediction responses `y` are not supplied.

family The fitted response family.

gamma Stored terminal-node coefficient summaries used for prediction.

id Long-format subject identifier corresponding to the supplied prediction data.

id.unique Unique subject identifiers in subject order.

k Number of terminal nodes requested during fitting.

m.opt Selected stopping iteration for each boosted subproblem.

membership Predicted terminal-node memberships for each subproblem and boosting iteration.

mu Predicted mean trajectories at the time points requested. If `x`, `tm`, and `id` are supplied, then `mu` is evaluated at the supplied subject-specific times. If only `x` is supplied, then each new subject is predicted on the fitted training time grid, so `mu` is already a full profile on that grid. If `partial = TRUE`, then `mu` is evaluated on the user-supplied common grid `tm`. For continuous and binary families this is a subject-level list of predicted trajectories. For nominal and ordinal families this is indexed first by boosted subproblem and then by subject.

muhat Predicted full profiles reconstructed on `time.grid`, where `time.grid` is the fitted training time grid used by the model.

n Number of subjects in the prediction data.

n.q Number of boosted subproblems.

ni Number of observations or requested time points per subject.

nu Boosting step size used by the fitted model.

nu.vec Expanded step-size vector on the time-basis scale.

partial Logical; whether prediction was requested with a common user-supplied time grid.

prob.class Predicted class probabilities on the original response scale for non-continuous families; `NULL` for the continuous family.

prob.hat.class Class probabilities over time for non-continuous families; `NULL` for the continuous family or when `use.cv.flag = TRUE`.

q.set Threshold levels (ordinal) or non-reference levels (binary or nominal) defining the boosted subproblems.

q.total Total number of response levels for non-continuous families.

rmse Standardized test-set RMSE evaluated at `m.opt` when prediction responses `y` are supplied; otherwise `NULL`.

time A list of observed or requested times for each subject.

time.design Subject-specific time-design matrices used for prediction.

time.grid The common grid used for prediction; typically the training time grid.

time.unique Sorted unique times appearing in `time`.

use.cv.flag Logical; whether out-of-bag coefficient estimates were used.

x Prediction covariates with one row per subject.

x.var.names Covariate names expected by the fitted model.

y Observed prediction-set responses split by subject when supplied; otherwise NULL.

y.levels Observed response levels from the training fit for non-continuous families. NA for the continuous family.

y.mean Overall response mean used for standardization.

y.org Prediction-set responses encoded at the boosted-subproblem level when y is supplied; otherwise NULL. For continuous and binary families this is a subject-level list. For nominal and ordinal families it is indexed first by boosted subproblem and then by subject.

y.reference Reference response level used by the nominal family; NULL otherwise.

y.sd Overall response standard deviation used for standardization.

Author(s)

Amol Pande, Udaya B. Kogalur and Hemant Ishwaran

References

Friedman J.H. (2001). Greedy function approximation: a gradient boosting machine. *The Annals of Statistics*, 29(5):1189–1232.

Pande A., Li L., Rajeswaran J., Ehrlinger J., Kogalur U.B., Blackstone E.H., Ishwaran H. (2017). Boosted multivariate trees for longitudinal data. *Machine Learning*, 106(2):277–305.

Pande A., Ishwaran H., Blackstone E.H., Rajeswaran J., and Gillanov M. (2022). Application of gradient boosting in evaluating surgical ablation for atrial fibrillation. *SN Computer Science*, 3:466.

Pande A., Ishwaran H., and Blackstone E.H. (2022). Boosting for multivariate longitudinal responses. *SN Computer Science*, 3:186.

See Also

boostmtree, partial.plot.boostmtree, plot.boostmtree, print.boostmtree, simLong, spirometry, vimp.boostmtree

Examples

```
## ------------------------------------------------------------
## Continuous longitudinal prediction on a held-out test set.
## ------------------------------------------------------------
set.seed(31)
sim.obj <- simLong(n = 20, n.test = 10, n.time = 4, model = 1,
                   family = "continuous")
dta <- sim.obj$data.list
trn <- sim.obj$train.index

fit <- boostmtree(
  x = dta$features[trn, , drop = FALSE],
  tm = dta$time[trn],
  id = dta$id[trn],
  y = dta$y[trn],
  family = "continuous",
```

```
  M = 10,
  verbose = FALSE
)

pred.obj <- predict(
  fit,
  x = dta$features[-trn, , drop = FALSE],
  tm = dta$time[-trn],
  id = dta$id[-trn],
  y = dta$y[-trn]
)

print(pred.obj)

## ------------------------------------------------------------
## Predict full profiles for new subjects on the training time grid.
## ------------------------------------------------------------
new.subjects <- dta$features[trn, , drop = FALSE][1:3, ]
pred.obj <- predict(fit, x = new.subjects)
str(pred.obj$mu[[1]], max.level = 1)

## ------------------------------------------------------------
## Predict on a user-supplied common time grid.
## ------------------------------------------------------------
grid.time <- seq(min(dta$time[trn]), max(dta$time[trn]), length.out = 25)
pred.grid <- predict(
  fit,
  x = new.subjects,
  tm = grid.time,
  partial = TRUE
)

str(pred.grid$mu[[1]], max.level = 1)


## ------------------------------------------------------------
## Binary longitudinal prediction.
## ------------------------------------------------------------
set.seed(44)
sim.bin <- simLong(n = 25, n.test = 10, n.time = 4, model = 2,
                   family = "binary")
dta.bin <- sim.bin$data.list
trn.bin <- sim.bin$train.index

fit.bin <- boostmtree(
  x = dta.bin$features[trn.bin, , drop = FALSE],
  tm = dta.bin$time[trn.bin],
  id = dta.bin$id[trn.bin],
  y = dta.bin$y[trn.bin],
  family = "binary",
  M = 10,
  verbose = FALSE
)
```

```
pred.bin <- predict(
  fit.bin,
  x = dta.bin$features[-trn.bin, , drop = FALSE],
  tm = dta.bin$time[-trn.bin],
  id = dta.bin$id[-trn.bin],
  y = dta.bin$y[-trn.bin]
)

print(pred.bin)
```

`print.boostmtree` *Print a summary of a boostmtree fit or prediction object*

Description

Print a compact summary of a fitted `boostmtree` model or a compatible prediction object.

Usage

```
## S3 method for class 'boostmtree'
print(x, ...)
```

Arguments

`x`	An object of class `c("boostmtree", "grow", ...)` or `c("boostmtree", "predict", ...)`.
`...`	Currently ignored. Included for S3 compatibility.

Details

The printed summary is intended as a quick check that the model you fit is the model you meant to fit. It reports the response family, the requested number of terminal nodes, the sample size, and the structure of the repeated-measures design.

When cross-validation or test-set error information is available, the summary also reports the selected stopping iteration and the corresponding RMSE. For longitudinal fits it can additionally report the estimated working-correlation and working-variance parameters at the selected stopping iteration.

Value

The input object, returned invisibly.

References

Pande A., Li L., Rajeswaran J., Ehrlinger J., Kogalur U.B., Blackstone E.H., Ishwaran H. (2017). Boosted multivariate trees for longitudinal data. *Machine Learning*, 106(2):277–305.

Pande A., Ishwaran H., Blackstone E.H., Rajeswaran J., and Gillanov M. (2022). Application of gradient boosting in evaluating surgical ablation for atrial fibrillation. *SN Computer Science*, 3:466.

Pande A., Ishwaran H., and Blackstone E.H. (2022). Boosting for multivariate longitudinal responses. *SN Computer Science*, 3:186.

See Also

boostmtree, plot.boostmtree, simLong

Examples

```
## Print a basic longitudinal fit.
set.seed(11)
sim.obj <- simLong(n = 15, n.time = 4, family = "continuous")
dta <- sim.obj$data.list

fit <- boostmtree(
  x = dta$features,
  tm = dta$time,
  id = dta$id,
  y = dta$y,
  family = "continuous",
  M = 8,
  verbose = FALSE
)

print(fit)

## Print a fit that includes out-of-bag stopping information.
set.seed(12)
sim.bin <- simLong(n = 15, n.time = 4, family = "binary")
dta.bin <- sim.bin$data.list

fit.bin <- boostmtree(
  x = dta.bin$features,
  tm = dta.bin$time,
  id = dta.bin$id,
  y = dta.bin$y,
  family = "binary",
  M = 10,
  cv.flag = TRUE,
  verbose = FALSE
)

print(fit.bin)
```

simLong *Simulate longitudinal data for boostmtree examples*

Description

Generate synthetic longitudinal data for continuous or binary responses.

Usage

```
simLong(
  n = 100,
  n.test = 0,
  n.time = 5,
  rho = 0.8,
  cor.type = c("cor.comp.sym", "cor.ar1", "cor.symm", "iid"),
  model = c(0, 1, 2, 3),
  family = c("continuous", "binary"),
  phi = 1,
  q = 0,
  ...
)
```

Arguments

n	Number of training subjects.
n.test	Number of additional test subjects to generate.
n.time	Controls the number and scale of repeated observation times.
rho	Working correlation parameter used by the data generator.
cor.type	Correlation structure used for the repeated errors.
model	Signal-generating model. Choose one of 0, 1, 2, or 3.
family	Response family. Choose "continuous" or "binary".
phi	Error-scale parameter.
q	Number of additional noise covariates to append to the baseline feature set.
...	Currently ignored.

Value

A list with components:

data.list A list containing features, time, id, and y.

data The full long-format simulated data frame.

train.index Indices identifying the training portion of data.

formula.true A character string describing the true signal model.

Author(s)

Hemant Ishwaran, Amol Pande, and Udaya B. Kogalur

See Also

boostmtree

Examples

```
set.seed(5)
sim.obj <- simLong(n = 10, n.time = 4, family = "continuous")
str(sim.obj$data.list, max.level = 1)
```

---

spirometry *Spirometry Data*

Description

Lung transplant data from 509 patients who underwent lung transplantation (LTx) at the Cleveland Clinic, comprising 9,471 longitudinal measurements of forced expiratory volume in one second (FEV1, expressed as a percentage of predicted). Twenty-three patient and procedure variables were recorded at the time of transplant. The primary objectives are to evaluate the temporal trend of FEV1 following LTx, to identify factors associated with post-LTx FEV1, and to assess differences in trajectories between single and double LTx procedures.

Format

A list containing four elements:

1. The 23 patient variables (features).
2. Time points (time).
3. Unique patient identifier (id).
4. FEV1-outcomes (y).

References

Mason D.P., Rajeswaran J., Li L., Murthy S.C., Su J.W., Pettersson G.B., Blackstone E.H. Effect of changes in postoperative spirometry on survival after lung transplantation. *J. Thorac. Cardiovasc. Surg.*, 144:197-203, 2012.

Examples

```
data(spirometry, package = "boostmtree")
```

`vimp.boostmtree` *Permutation Variable Importance for Boosted Tree Models*

Description

Compute permutation-based variable importance for fitted `boostmtree` objects and for prediction objects produced by `predict.boostmtree()`.

Usage

```
vimp.boostmtree(object, x.names = NULL, joint = FALSE)
```

Arguments

`object`	A fitted object of class `(boostmtree, grow)` or a prediction object of class `(boostmtree, predict)`.
`x.names`	Optional character vector naming the covariates to assess. If omitted, importance is computed for all available covariates.
`joint`	Logical value indicating whether the variables listed in `x.names` should be permuted jointly. When `FALSE`, each variable is permuted separately.

Details

Variable importance is computed by permuting one or more covariates and then measuring how much predictive accuracy deteriorates.

For grow objects, the procedure uses the out-of-bag prediction path stored in the fitted object. This requires two things: the model must have been fit with `cv.flag = TRUE`, and the resampling rule must have produced out-of-bag subjects at every boosting iteration used by the importance calculation. In ordinary use, this happens automatically because the default control object uses `bootstrap = "by.root"`. If a model was fit with `bootstrap = "none"`, then grow-object variable importance is not available because there are no OOB subjects to perturb.

For prediction objects, the procedure uses the supplied test responses and reports the relative increase in test-set RMSE after permutation. This route does *not* require OOB sampling because the comparison is made on the held-out prediction data.

In longitudinal settings, the returned object separates three effects:

`main` importance of the baseline covariate effect

`interaction` importance of the covariate-time interaction

`time.effect` importance of the time basis alone

The returned value has class `vimp.boostmtree`. It can be plotted directly with `plot()`.

Value

An object of class `vimp.boostmtree`. Its main components are:

`main` matrix of permutation importance values for the main covariate effects

`interaction` matrix of time-interaction importance values for longitudinal fits, or `NULL`

`time.effect` vector containing the importance of the time basis alone, or `NULL`

`x.var.names` names of the assessed covariates

`metric` description of the accuracy measure used in the comparison

References

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29, 1189–1232.

Pande, A., Ishwaran, H., Blackstone, E. H., Rajeswaran, J., and Gillinov, M. (2022). Application of gradient boosting in evaluating surgical ablation for atrial fibrillation. *SN Computer Science*, 3, 466.

Pande, A., Ishwaran, H., and Blackstone, E. H. (2022). Boosting for multivariate longitudinal responses. *SN Computer Science*, 3, 186.

Examples

```
## ------------------------------------------------------------
## Variable importance from a fitted continuous longitudinal model.
## For grow objects this requires cv.flag = TRUE. The default
## control object already provides OOB subjects.
## ------------------------------------------------------------
set.seed(19)
sim.obj <- simLong(n = 100, n.time = 4, model = 2, family = "continuous")
dta <- sim.obj$data.list

fit <- boostmtree(
  x = dta$features,
  tm = dta$time,
  id = dta$id,
  y = dta$y,
  family = "continuous",
  M = 50,
  cv.flag = TRUE,
  verbose = TRUE
)

vimp.obj <- vimp.boostmtree(fit, x.names = c("x1", "x2"))
plot(vimp.obj)


## ------------------------------------------------------------
## Variable importance from a held-out test set.
## This route does not rely on OOB sampling because the comparison
## is made on the prediction object.
```

```
## ------------------------------------------------------------
set.seed(23)
sim.obj <- simLong(n = 200, n.test = 100, n.time = 4, model = 2,
                   family = "continuous")
dta <- sim.obj$data.list
trn <- sim.obj$train.index

fit <- boostmtree(
  x = dta$features[trn, , drop = FALSE],
  tm = dta$time[trn],
  id = dta$id[trn],
  y = dta$y[trn],
  family = "continuous",
  M = 50,
  verbose = TRUE,
  control = boostmtree.control(bootstrap = "none", seed = 23)
)

pred.obj <- predict(
  fit,
  x = dta$features[-trn, , drop = FALSE],
  tm = dta$time[-trn],
  id = dta$id[-trn],
  y = dta$y[-trn]
)

vimp.test <- vimp.boostmtree(pred.obj, x.names = c("x1", "x2"))
plot(vimp.test)
```

Index