Package ‘boostmtree’

November 21, 2019

Version 1.4.1

Date 2019-11-21

Title Boosted Multivariate Trees for Longitudinal Data

Author Hemant Ishwaran <hemant.ishwaran@gmail.com>, Amol Pande <amoljpande@gmail.com>

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

Depends R (>= 3.5.0)

Imports randomForestSRC (>= 2.9.0), parallel, splines, nlme

Description Implements Friedman's gradient descent boosting algorithm for modeling of continuous or binary longitudinal response using multivariate tree base learners. A time-covariate interaction effect is modeled using penalized B-splines (P-splines) with estimated adaptive smoothing parameter.

License GPL (>= 3)

URL http://web.ccs.miami.edu/~hishwaran

NeedsCompilation no

Repository CRAN

Date/Publication 2019-11-21 22:40:16 UTC

R topics documented:

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

Description

Multivariate extension of Friedman's (2001) gradient descent boosting method for modeling continuous or binary longitudinal data using multivariate tree base learners. Covariate-time interactions are modeled using penalized B-splines (P-splines) with estimated adaptive smoothing parameter.

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)

Hemant Ishwaran, Amol Pande and Udaya B. Kogalur

References

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

Friedman J.H. (2002). Stochastic gradient boosting. *Comp. Statist. Data Anal.*, 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.

See Also

`partialPlot`, `plot.boostmtree`, `predict.boostmtree`, `print.boostmtree`, `simLong`

`AF` *Atrial Fibrillation Data*

Description

Atrial Fibrillation (AF) data is obtained from a randomized trial to study the effect of surgical ablation as a treatment option for AF among patients with persistent and long-standing persistent AF who requires mitral valve surgery. Patients were randomized into two groups: mitral valve surgery with ablation and mitral valve surgery without ablation. Patients in the ablation group were further randomized into two types of procedure: pulmonary vain isolation (PVI) and biatrial maze procedure. These patients were followed weekly for a period of 12 months. The primary outcome of the study is the presence/absence of AF (binary longitudinal response). Data includes 228 patients. From 228 patients, 7949 AF measurements are available with average of 35 measurements per patient.

Format

A list containing four elements:

1. The 34 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

Multivariate extension of Friedman's gradient descent boosting method for modeling continuous or binary longitudinal response using multivariate tree base learners (Pande et al., 2017). Covariate-time interactions are modeled using penalized B-splines (P-splines) with estimated adaptive smoothing parameter.

Usage

```
boostmtree(x,
           tm,
           id,
           y,
           family = c("Continuous","Binary"),
           M = 200,
           nu = 0.05,
           K = 5,
           nknots = 10,
           d = 3,
           pen.ord = 3,
           lambda,
           lambda.max = 1e6,
           lambda.iter = 2,
           svd.tol = 1e-6,
           forest.tol = 1e-3,
           verbose = TRUE,
           cv.flag = FALSE,
           eps = 1e-5,
           mod.grad = TRUE,
           NR.iter = 3,
           ...)
```

Arguments

`x`	Data frame (or matrix) containing the x-values. Rows must be duplicated to match the number of time points for an individual. That is, if individual *i* has *n[i]* outcome y-values, then there must be *n[i]* duplicate rows of *i*'s x-value.
`tm`	Vector of time values, one entry for each row in `x`.
`id`	Unique subject identifier, one entry for each row in `x`.
`y`	Observed y-value, one entry for each row in `x`.
`family`	Family of the response variable `y`. Use any one from {"Continuous", "Binary"} based on the scale of `y`.
`M`	Number of boosting iterations
`nu`	Boosting regularization parameter. A value in (0,1].
`K`	Number of terminal nodes used for the multivariate tree learner.
`nknots`	Number of knots used for the B-spline for modeling the time interaction effect.
`d`	Degree of the piecewise B-spline polynomial (no time effect is fit when `d` < 1).
`pen.ord`	Differencing order used to define the penalty with increasing values implying greater smoothness.
`lambda`	Smoothing (penalty) parameter used for B-splines with increasing values associated with increasing smoothness/penalization. If missing, or non-positive, the value is estimated adaptively using a mixed models approach.
`lambda.max`	Tolerance used for adaptively estimated lambda (caps it). For experts only.

`lambda.iter`	Number of iterations used to estimate lambda (only applies when lambda is not supplied and adaptive smoothing is employed).
`svd.tol`	Tolerance value used in the SVD calculation of the penalty matrix. For experts only.
`forest.tol`	Tolerance used for forest weighted least squares solution. Experimental and for experts only.
`verbose`	Should verbose output be printed?
`cv.flag`	Should in-sample cross-validation (CV) be used to determine optimal stopping using out of bag data?
`eps`	Tolerance value used for determining the optimal `M`. Applies only if `cv.flag` = TRUE. For experts only.
`mod.grad`	Use a modified gradient? See details below.
`NR.iter`	Number of Newton-Raphson iteration. Applied for `family` = "Binary".
`...`	Further arguments passed to or from other methods.

Details

Each individual has observed y-values, over possibly different time points, with possibly differing number of time points. Given y, the time points, and x, the conditional mean time profile of y is estimated using gradient boosting in which the gradient is derived from a criterion function involving a working variance matrix for y specified as an equicorrelation matrix with parameter *rho* multiplied by a variance parameter *phi*. Multivariate trees are used for base learners and weighted least squares is used for solving the terminal node optimization problem. This provides solutions to the core parameters of the algorithm. For ancillary parameters, a mixed-model formulation is used to estimate the smoothing parameter associated with the B-splines used for the time-interaction effect, although the user can manually set the smoothing parameter as well. Ancillary parameters *rho* and *phi* are estimated using GLS (generalized least squares).

In the original boostmtree algorithm (Pande et al., 2017), the equicorrelation parameter *rho* is used in two places in the algorithm: (1) for growing trees using the gradient, which depends upon *rho*; and (2) for solving the terminal node optimization problem which also uses the gradient. However, Pande (2017) observed that setting *rho* to zero in the gradient used for growing trees improved performance of the algorithm, especially in high dimensions. For this reason the default setting used in this algorithm is to set *rho* to zero in the gradient for (1). The `rho` in the gradient for (2) is not touched. The option `mod.grad` specifies whether a modified gradient is used in the tree growing process and is TRUE by default.

By default, trees are grown from a bootstrap sample of the data – thus the boosting method employed here is a modified example of stochastic gradient descent boosting (Friedman, 2002). Stochastic descent often improves performance and has the added advantage that out-of-sample data (out-of-bag, OOB) can be used to calculate variable importance (VIMP).

The package implements R-side parallel processing by replacing the R function `lapply` with `mclapply` found in the **parallel** package. You can set the number of cores accessed by `mclapply` by issuing the command `options(mc.cores = x)`, where `x` is the number of cores. The options command can also be placed in the users .Rprofile file for convenience. You can, alternatively, initialize the environment variable `MC_CORES` in your shell environment.

As an example, issuing the following options command uses all available cores for R-side parallel processing:

`options(mc.cores=detectCores())`

However, be cautious when setting `mc.cores`. This can create not only high CPU usage but also high RAM usage, especially when using functions `partialPlot` and `predict`.

The method can impute the missing observations in x (covariates) using on the fly imputation. Details regarding can be found in the **randomForestSRC** package. If missing values are present in the `tm`, `id` or `y`, the user should either impute or delete these values before executing the function.

Finally note `cv.flag` can be used for an in-sample cross-validated estimate of prediction error. This is used to determine the optimized number of boosting iterations *Mopt*. The final mu predictor is evaluated at this value and is cross-validated. The prediction error returned via `err.rate` is standardized by the overall standard deviation of y.

Value

An object of class `(boostmtree,grow)` with the following components:

`x`	The x-values, but with only one row per individual (i.e. duplicated rows are removed). Values sorted on `id`.
`xvar.names`	X-variable names.
`time`	List with each component containing the time points for a given individual. Values sorted on `id`.
`id`	Sorted subject identifier.
`y`	List with each component containing the observed `y`-values for a given individual. Values sorted on `id`.
`family`	Family of `y`.
`ymean`	Overall mean of `y`-values for all individuals. If `family` = "Binary", `ymean` = 0.
`ysd`	Overall standard deviation of `y`-values for all individuals. If `family` = "Binary", `ysd` = 1.
`n`	Total number of subjects.
`ni`	Number of repeated measures for each subject.
`tm.unq`	Unique time points.
`gamma`	List of length *M*, with each component containing the boosted tree fitted values.
`mu`	List with each component containing the estimated mean values for an individual. That is, each component contains the estimated time-profile for an individual. When in-sample cross-validation is requested using `cv.flag`=TRUE, the estimated mean is cross-validated and evaluated at the optimal number of iterations `Mopt`.
`lambda`	Smoothing parameter.
`phi`	Variance parameter.
`rho`	Correlation parameter.
`baselearner`	List of length *M* containing the base learners.
`membership`	List of length *M*, with each component containing the terminal node membership for a given boosting iteration.

`X.tm`	Design matrix for all the unique time points.
`D`	Design matrix for each subject.
`d`	Degree of the piecewise B-spline polynomial.
`pen.ord`	Penalization difference order.
`K`	Number of terminal nodes.
`M`	Number of boosting iterations.
`nu`	Boosting regularization parameter.
`ntree`	Number of trees.
`cv.flag`	Whether in-sample CV is used or not?
`err.rate`	In-sample standardized estimate of l1-error and RMSE.
`rmse`	In-sample standardized RMSE at optimized `M`.
`Mopt`	The optimized `M`.
`gamma.i.list`	Estimate of gamma obtained from in-sample CV if `cv.flag` = TRUE, else NULL
`forest.tol`	Forest tolerance value (needed for prediction).

Author(s)

Hemant Ishwaran, Amol Pande and Udaya B. Kogalur

References

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

Friedman J.H. (2002). Stochastic gradient boosting. *Comp. Statist. Data Anal.*, 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. (2017). *Boosting for longitudinal data*. Ph.D. Dissertation, Miller School of Medicine, University of Miami.

See Also

`marginalPlot` `partialPlot`, `plot.boostmtree`, `predict.boostmtree`, `print.boostmtree`, `simLong`, `vimpPlot`

Examples

```
##------------------------------------------------------------
## synthetic example (Response y is continuous)
## 0.8 correlation, quadratic time with quadratic interaction
##-------------------------------------------------------------
#simulate the data (use a small sample size for illustration)
dta <- simLong(n = 50, N = 5, rho =.80, model = 2,family = "Continuous")$dtaL

#basic boosting call (M set to a small value for illustration)
```

```
boost.grow <- boostmtree(dta$features, dta$time, dta$id, dta$y,family = "Continuous",M = 20)

#print results
print(boost.grow)

#plot.results
plot(boost.grow)

##------------------------------------------------------------
## synthetic example (Response y is binary)
## 0.8 correlation, quadratic time with quadratic interaction
##-------------------------------------------------------------
#simulate the data (use a small sample size for illustration)
dta <- simLong(n = 50, N = 5, rho =.80, model = 2, family = "Binary")$dtaL

#basic boosting call (M set to a small value for illustration)
boost.grow <- boostmtree(dta$features, dta$time, dta$id, dta$y,family = "Binary", M = 20)

#print results
print(boost.grow)

#plot.results
plot(boost.grow)

## Not run:
##------------------------------------------------------------
## Same synthetic example as above with continuous response
## but with in-sample cross-validation estimate for RMSE
##-------------------------------------------------------------
dta <- simLong(n = 50, N = 5, rho =.80, model = 2,family = "Continuous")$dtaL
boost.cv.grow <- boostmtree(dta$features, dta$time, dta$id, dta$y,
                 family = "Continuous", M = 300, cv.flag = TRUE)
plot(boost.cv.grow)
print(boost.cv.grow)

##----------------------------------------------------------------------------
## spirometry data (Response is continuous)
##----------------------------------------------------------------------------
data(spirometry, package = "boostmtree")

#boosting call: cubic B-splines with 15 knots
spr.obj <- boostmtree(spirometry$features, spirometry$time, spirometry$id, spirometry$y,
                        family = "Continuous",M = 100, nu = .025, nknots = 15)
plot(spr.obj)


##----------------------------------------------------------------------------
## Atrial Fibrillation data (Response is binary)
##----------------------------------------------------------------------------
data(AF, package = "boostmtree")

#boosting call: cubic B-splines with 15 knots
AF.obj <- boostmtree(AF$feature, AF$time, AF$id, AF$y,
                        family = "Binary",M = 100, nu = .025, nknots = 15)
plot(AF.obj)


##----------------------------------------------------------------------------
## sneaky way to use boostmtree for (univariate) regression: boston housing
##----------------------------------------------------------------------------

if (library("mlbench", logical.return = TRUE)) {

  ## assemble the data
  data(BostonHousing)
  x <- BostonHousing; x$medv <- NULL
  y <- BostonHousing$medv
  trn <- sample(1:nrow(x), size = nrow(x) * (2 / 3), replace = FALSE)

  ## run boosting in univariate mode
  o <- boostmtree(x = x[trn,], y = y[trn],family = "Continuous")
  o.p <- predict(o, x = x[-trn, ], y = y[-trn])
  print(o)
  plot(o.p)

  ## run boosting in univariate mode to obtain RMSE and vimp
  o.cv <- boostmtree(x = x, y = y, M = 100,family = "Continuous",cv.flag = TRUE)
  print(o.cv)
  plot(o.cv)
}


## End(Not run)
```

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)

Hemant Ishwaran, Amol Pande and Udaya B. Kogalur

`marginalPlot` *Marginal plot analysis*

Description

Marginal plot of x against the unadjusted predicted y. This is mainly used to obtain marginal relationships between x and the unadjusted predicted y. Marginal plots have a faster execution compared to partial plots (Friedman, 2001).

Usage

```
marginalPlot(object,
             xvar.names,
             tm.unq,
             subset,
             plot.it = FALSE,
             ...)
```

Arguments

`object`	A boosting object of class `(boostmtree,grow)`.
`xvar.names`	Names of the x-variables to be used. By default, all variables are plotted.
`tm.unq`	Unique time points used for the plots of x against y. By default, the deciles of the observed time values are used.
`subset`	Vector indicating which rows of the x-data to be used for the analysis. The default is to use the entire data.
`plot.it`	Should plots be displayed? If `xvar.names` is a vector with more than one variable name, then instead of displaying, plot is stored as "MarginalPlot.pdf" in the current working directory.
`...`	Further arguments passed to or from other methods.

Details

Marginal plot of x values specified by `xvar.names` against the unadjusted predicted y-values over a set of time points specified by `tm.unq`. Analysis can be restricted to a subset of the data using `subset`.

Author(s)

Hemant Ishwaran, Amol Pande and Udaya B. Kogalur

References

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

Examples

```
## Not run:
##------------------------------------------------------------
## Synthetic example (Response is continuous)
## High correlation, quadratic time with quadratic interaction
##-------------------------------------------------------------
#simulate the data
dta <- simLong(n = 50, N = 5, rho =.80, model = 2,family = "Continuous")$dtaL

#basic boosting call
boost.grow <- boostmtree(dta$features, dta$time, dta$id, dta$y, family = "Continuous", M = 300)

#plot results
#x1 has a linear main effect
#x2 is quadratic with quadratic time trend
marginalPlot(boost.grow, "x1",plot.it = TRUE)
marginalPlot(boost.grow, "x2",plot.it = TRUE)

#Plot of all covariates. The plot will be stored as the "MarginalPlot.pdf"
# in the current working directory.
marginalPlot(boost.grow,plot.it = TRUE)


##------------------------------------------------------------
## Synthetic example (Response is binary)
## High correlation, quadratic time with quadratic interaction
##-------------------------------------------------------------
#simulate the data
dta <- simLong(n = 50, N = 5, rho =.80, model = 2,family = "Binary")$dtaL

#basic boosting call
boost.grow <- boostmtree(dta$features, dta$time, dta$id, dta$y, family = "Binary", M = 300)

#plot results
#x1 has a linear main effect
#x2 is quadratic with quadratic time trend
marginalPlot(boost.grow, "x1",plot.it = TRUE)
marginalPlot(boost.grow, "x2",plot.it = TRUE)

#Plot of all covariates. The plot will be stored as the "MarginalPlot.pdf"
# in the current working directory.
marginalPlot(boost.grow,plot.it = TRUE)

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

```
#boosting call: cubic B-splines with 15 knots
spr.obj <- boostmtree(spirometry$features, spirometry$time, spirometry$id, spirometry$y,
            family = "Continuous",M = 300, nu = .025, nknots = 15)

#marginal plot of double-lung group at 5 years
dltx <- marginalPlot(spr.obj, "AGE", tm.unq = 5, subset = spr.obj$x$DOUBLE==1,plot.it = TRUE)

#marginal plot of single-lung group at 5 years
sltx <- marginalPlot(spr.obj, "AGE", tm.unq = 5, subset = spr.obj$x$DOUBLE==0,plot.it = TRUE)

#combine the two plots
dltx <- dltx[[2]][[1]]
sltx <- sltx[[2]][[1]]
plot(range(c(dltx[[1]][, 1], sltx[[1]][, 1])), range(c(dltx[[1]][, -1], sltx[[1]][, -1])),
     xlab = "age", ylab = "predicted y", type = "n")
lines(dltx[[1]][, 1][order(dltx[[1]][, 1]) ], dltx[[1]][, -1][order(dltx[[1]][, 1]) ],
      lty = 1, lwd = 2, col = "red")
lines(sltx[[1]][, 1][order(sltx[[1]][, 1]) ], sltx[[1]][, -1][order(sltx[[1]][, 1]) ],
      lty = 1, lwd = 2, col = "blue")
legend("topright", legend = c("DLTx", "SLTx"), lty = 1, fill = c(2,4))

## End(Not run)
```

---

partialPlot *Partial plot analysis*

Description

Partial dependence plot of x against adjusted predicted y.

Usage

```
partialPlot(object,
            xvar.names,
            tm.unq,
            xvar.unq = NULL,
            npts = 25,
            subset,
            conditional.xvars = NULL,
            conditional.values = NULL,
            plot.it = FALSE,
            Variable_Factor = FALSE,
            ...)
```

Arguments

`object`	A boosting object of class (`boostmtree,grow`).
`xvar.names`	Names of the x-variables to be used. By default, all variables are plotted.

`tm.unq`	Unique time points used for the plots of x against y. By default, the deciles of the observed time values are used.
`xvar.unq`	Unique values used for the partial plot. Default is NULL in which case unique values are obtained uniformaly based on the range of variable. Values must be provided using list with same length as lenght of `xvar.names`.
`npts`	Maximum number of points used for x. Reduce this value if plots are slow.
`subset`	Vector indicating which rows of the x-data to be used for the analysis. The default is to use the entire data.
`conditional.xvars`	Vector of character values indicating names of the x-variables to be used for further conditioning (adjusting) the predicted y values. Variable names should be different from `xvar.names`.
`conditional.values`	Vector of values taken by the variables from `conditional.xvars`. The length of the vector should be same as the length of the vector for `conditional.xvars`, which means only one value per conditional variable.
`plot.it`	Should plots be displayed?
`Variable_Factor`	Default is FALSE. Use TRUE if the variable specified in `xvar.names` is a factor.
`...`	Further arguments passed to or from other methods.

Details

Partial dependence plot (Friedman, 2001) of x values specified by `xvar.names` against the adjusted predicted y-values over a set of time points specified by `tm.unq`. Analysis can be restricted to a subset of the data using `subset`. Further conditioning can be imposed using `conditional.xvars`.

Author(s)

Hemant Ishwaran, Amol Pande and Udaya B. Kogalur

References

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

Examples

```
## Not run:
##------------------------------------------------------------
## Synthetic example (Response is continuous)
## high correlation, quadratic time with quadratic interaction
##-------------------------------------------------------------
#simulate the data
dta <- simLong(n = 50, N = 5, rho =.80, model = 2,family = "Continuous")$dtaL

#basic boosting call
boost.grow <- boostmtree(dta$features, dta$time, dta$id, dta$y,family = "Continuous",M = 300)
```

```
#plot results
#x1 has a linear main effect
#x2 is quadratic with quadratic time trend
partialPlot(boost.grow, "x1",plot.it = TRUE)
partialPlot(boost.grow, "x2",plot.it = TRUE)

#partial plot using "x2" as the conditional variable
partialPlot(boost.grow, "x1", conditional.xvar = "x2", conditional.values = 1,plot.it = TRUE)
partialPlot(boost.grow, "x1", conditional.xvar = "x2", conditional.values = 2,plot.it = TRUE)

##-------------------------------------------------------------
## Synthetic example (Response is binary)
## high correlation, quadratic time with quadratic interaction
##-------------------------------------------------------------
#simulate the data
dta <- simLong(n = 50, N = 5, rho =.80, model = 2,family = "Binary")$dtaL

#basic boosting call
boost.grow <- boostmtree(dta$features, dta$time, dta$id, dta$y,family = "Binary",M = 300)

#plot results
#x1 has a linear main effect
#x2 is quadratic with quadratic time trend
partialPlot(boost.grow, "x1",plot.it = TRUE)
partialPlot(boost.grow, "x2",plot.it = TRUE)

##----------------------------------------------------------------------------
## spirometry data
##----------------------------------------------------------------------------
data(spirometry, package = "boostmtree")

#boosting call: cubic B-splines with 15 knots
spr.obj <- boostmtree(spirometry$features, spirometry$time, spirometry$id, spirometry$y,
            family = "Continuous",M = 300, nu = .025, nknots = 15)

#partial plot of double-lung group at 5 years
dltx <- partialPlot(spr.obj, "AGE", tm.unq = 5, subset=spr.obj$x$DOUBLE==1,plot.it = TRUE)

#partial plot of single-lung group at 5 years
sltx <- partialPlot(spr.obj, "AGE", tm.unq = 5, subset=spr.obj$x$DOUBLE==0,plot.it = TRUE)

#combine the two plots: we use lowess smoothed values
dltx <- dltx$l.obj[[1]]
sltx <- sltx$l.obj[[1]]
plot(range(c(dltx[, 1], sltx[, 1])), range(c(dltx[, -1], sltx[, -1])),
     xlab = "age", ylab = "predicted y (adjusted)", type = "n")
lines(dltx[, 1], dltx[, -1], lty = 1, lwd = 2, col = "red")
lines(sltx[, 1], sltx[, -1], lty = 1, lwd = 2, col = "blue")
legend("topright", legend = c("DLTx", "SLTx"), lty = 1, fill = c(2,4))

## End(Not run)
```

`plot.boostmtree` *Plot Summary Analysis*

Description

Plot summary analysis of the boosting analysis.

Usage

```
## S3 method for class 'boostmtree'
plot(x, use.rmse = TRUE, ...)
```

Arguments

`x`	An object of class `(boostmtree,grow)` or `(boostmtree,predict)`.
`use.rmse`	Report performance values in terms of standardized root-mean-squared-error (RMSE) or mean-squared-error (MSE)? Default is standardized RMSE.
`...`	Further arguments passed to or from other methods.

Details

Plot summary output, including predicted values and residuals. Also plots various parameters against the number of boosting iterations.

Author(s)

Hemant Ishwaran, Amol Pande and Udaya B. Kogalur

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.

`predict.boostmtree` *Prediction for Boosted multivariate trees for longitudinal data.*

Description

Obtain predicted values. Also returns test-set performance if the test data contains y-outcomes.

Usage

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

Arguments

`object`	A boosting object of class `(boostmtree,grow)`.
`x`	Data frame (or matrix) containing test set x-values. Rows must be duplicated to match the number of time points for an individual. If missing, the training `x` values are used and `tm`, `id` and `y` are not required and no performance values are returned.
`tm`	Time values for each test set individual with one entry for each row of `x`. Optional, but if missing, the set of unique time values from the training values are used for each individual and no test-set performance values are returned.
`id`	Unique subject identifier, one entry for each row in `x`. Optional, but if missing, each individual is assumed to have a full time-profile specified by the unique time values from the training data.
`y`	Test set y-values, with one entry for each row in `x`.
`M`	Fixed value for the boosting step number. Leave this empty to determine the optimized value obtained by minimizing test-set error.
`eps`	Tolerance value used for determining the optimal `M`. For experts only.
`...`	Further arguments passed to or from other methods.

Details

The predicted time profile and performance values are obtained for test data from the boosted object grown on the training data.

R-side parallel processing is implemented by replacing the R function `lapply` with `mclapply` found in the **parallel** package. You can set the number of cores accessed by `mclapply` by issuing the command `options(mc.cores = x)`, where `x` is the number of cores. As an example, issuing the following options command uses all available cores:

`options(mc.cores=detectCores())`

However, this can create high RAM usage, especially when using function `partialPlot` which calls the `predict` function.

Note that all performance values (for example prediction error) are standardized by the overall y-standard deviation. Thus, reported RMSE (root-mean-squared-error) is actually standardized RMSE. Values are reported at the optimal stopping time.

Value

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

`boost.obj`	The original boosting object.
`x`	The test x-values, but with only one row per individual (i.e. duplicated rows are removed).
`time`	List with each component containing the time points for a given test individual.
`id`	Sorted subject identifier.
`y`	List containing the test y-values.
`family`	Family of `y`.
`ymean`	Overall mean of y-values for all individuals. If `family` = "Binary", `ymean` = 0.
`ysd`	Overall standard deviation of y-values for all individuals. If `family` = "Binary", `ysd` = 1.
`xvar.names`	X-variable names.
`K`	Number of terminal nodes.
`n`	Total number of subjects.
`ni`	Number of repeated measures for each subject.
`nu`	Boosting regularization parameter.
`D`	Design matrix for each subject.
`df.D`	Number of columns of `D`.
`time.unq`	Vector of the unique time points.
`baselearner`	List of length *M* containing the base learners.
`gamma`	List of length *M*, with each component containing the boosted tree fitted values.
`membership`	List of length *M*, with each component containing the terminal node membership for a given boosting iteration.
`mu`	Estimated mean profile at the optimized `M`.
`muhat`	Extrapolated mean profile to all unique time points evaluated at the the optimized `M`.
`phi`	Variance parameter at the optimized `M`.
`rho`	Correlation parameter at the optimized `M`.
`err.rate`	Test set standardized l1-error and RMSE.
`rmse`	Test set standardized RMSE at the optimized `M`.
`Mopt`	The optimized `M`.

Author(s)

Hemant Ishwaran, Amol Pande and Udaya B. Kogalur

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.

See Also

plot.boostmtree, print.boostmtree

Examples

```
## Not run:
##------------------------------------------------------------
## Synthetic example (Response is continuous)
##
##  High correlation, quadratic time with quadratic interaction
##  largish number of noisy variables
##
##  Illustrates how modified gradient improves performance
##  also compares performance to ideal and well specified linear models
##----------------------------------------------------------------------------

## simulate the data
## simulation 2: main effects (x1, x3, x4), quad-time-interaction (x2)
dtaO <- simLong(n = 100, ntest = 100, model = 2, family = "Continuous", q = 25)

## save the data as both a list and data frame
dtaL <- dtaO$dtaL
dta <- dtaO$dta

## get the training data
trn <- dtaO$trn

## save formulas for linear model comparisons
f.true <- dtaO$f.true
f.linr <- "y~g( x1+x2+x3+x4+x1*time+x2*time+x3*time+x4*time )"


## modified tree gradient (default)
o.1 <- boostmtree(dtaL$features[trn, ], dtaL$time[trn], dtaL$id[trn],dtaL$y[trn],
       family = "Continuous",M = 350)
p.1 <- predict(o.1, dtaL$features[-trn, ], dtaL$time[-trn], dtaL$id[-trn], dtaL$y[-trn])

## non-modified tree gradient (nmtg)
o.2 <- boostmtree(dtaL$features[trn, ], dtaL$time[trn], dtaL$id[trn], dtaL$y[trn],
       family = "Continuous",M = 350, mod.grad = FALSE)
p.2 <- predict(o.2, dtaL$features[-trn, ], dtaL$time[-trn], dtaL$id[-trn], dtaL$y[-trn])

## set rho = 0
o.3 <- boostmtree(dtaL$features[trn, ], dtaL$time[trn], dtaL$id[trn], dtaL$y[trn],
       family = "Continuous",M = 350, rho = 0)
p.3 <- predict(o.3, dtaL$features[-trn, ], dtaL$time[-trn], dtaL$id[-trn], dtaL$y[-trn])


##rmse values compared to generalized least squares (GLS)
##for true model and well specified linear models (LM)
cat("true LM           :", boostmtree:::gls.rmse(f.true,dta,trn),"\n")
cat("well specified LM :", boostmtree:::gls.rmse(f.linr,dta,trn),"\n")
cat("boostmtree        :", p.1$rmse,"\n")
cat("boostmtree  (nmtg):", p.2$rmse,"\n")
cat("boostmtree (rho=0):", p.3$rmse,"\n")

##predicted value plots
plot(p.1)
plot(p.2)
plot(p.3)



##--------------------------------------------------------------
## Synthetic example (Response is binary)
##
##  High correlation, quadratic time with quadratic interaction
##  largish number of noisy variables
##----------------------------------------------------------------------------

## simulate the data
## simulation 2: main effects (x1, x3, x4), quad-time-interaction (x2)
dtaO <- simLong(n = 100, ntest = 100, model = 2, family = "Binary", q = 25)

## save the data as both a list and data frame
dtaL <- dtaO$dtaL
dta <- dtaO$dta

## get the training data
trn <- dtaO$trn

## save formulas for linear model comparisons
f.true <- dtaO$f.true
f.linr <- "y~g( x1+x2+x3+x4+x1*time+x2*time+x3*time+x4*time )"


## modified tree gradient (default)
o.1 <- boostmtree(dtaL$features[trn, ], dtaL$time[trn], dtaL$id[trn],dtaL$y[trn],
       family = "Binary",M = 350)
p.1 <- predict(o.1, dtaL$features[-trn, ], dtaL$time[-trn], dtaL$id[-trn], dtaL$y[-trn])


## End(Not run)
```

print.boostmtree *Print Summary Output*

Description

Print summary output from the boosting analysis.

Usage

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

Arguments

`x`	An object of class `(boostmtree,grow)` or `(boostmtree,predict)`.
`...`	Further arguments passed to or from other methods.

Author(s)

Hemant Ishwaran, Amol Pande and Udaya B. Kogalur

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.

`simLong` *Simulate longitudinal data*

Description

Simulates longitudinal data with continuous or binary response from models with increasing complexity of covariate-time interactions.

Usage

```
simLong(n,
        ntest = 0,
        N = 5,
        rho = 0.8,
        type = c("corCompSym", "corAR1", "corSymm", "iid"),
        model = c(0, 1, 2, 3),
        family = c("Continuous","Binary"),
        phi = 1,
        q = 0,
        ...)
```

Arguments

`n`	Requested training sample size.
`ntest`	Requested test sample size.
`N`	Parameter controlling number of time points per subject.
`rho`	Correlation parameter.
`type`	Type of correlation matrix.

`model`	Requested simulation model.
`family`	Family of response `y`. Use any one from {"Continuous", "Binary"} based on the scale of `y`.
`phi`	Variance of measurement error.
`q`	Number of zero-signal variables (i.e., variables unrelated to `y`).
`...`	Further arguments passed to or from other methods.

Details

Simulates longitudinal data with 3 main effects and (possibly) a covariate-time interaction. Complexity of the model is specified using the option `model`:

1. `model=0`*:* Linear with no covariate-time interactions.
2. `model=1`*:* Linear covariate-time interaction.
3. `model=2`*:* Quadratic time-quadratic covariate interaction.
4. `model=3`*:* Quadratic time-quadratic two-way covariate interaction.

For details see Pande et al. (2017).

Value

An invisible list with the following components:

`dtaL`	List containing the simulated data in the following order: `features`, `time`, `id` and `y`.
`dta`	Simulated data given as a data frame.
`trn`	Index of `id` values identifying the training data.
`f.true`	Formula of the simulation model.

Author(s)

Hemant Ishwaran, Amol Pande and Udaya B. Kogalur

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.

Examples

```
## Not run:
##------------------------------------------------------------
##  Response is continuous
##----------------------------------------------------------------------------

## set the number of boosting iterations
M <- 500
```

```
## simulation 0: only main effects (x1, x3, x4)
dta <- simLong(n = 100, ntest = 100, model = 0, family = "Continuous", q = 5)
trn <- dta$trn
dtaL <- dta$dtaL
dta <- dta$dta
obj.0 <-  boostmtree(dtaL$features[trn, ], dtaL$time[trn], dtaL$id[trn], dtaL$y[trn],
          family = "Continuous", M = M)
pred.0 <- predict(obj.0, dtaL$features[-trn, ], dtaL$time[-trn], dtaL$id[-trn], dtaL$y[-trn])



##--------------------------------------------------------------
##  Response is binary
##----------------------------------------------------------------------------

## set the number of boosting iterations
M <- 500

## simulation 0: only main effects (x1, x3, x4)
dta <- simLong(n = 100, ntest = 100, model = 0, family = "Binary", q = 5)
trn <- dta$trn
dtaL <- dta$dtaL
dta <- dta$dta
obj.0 <-  boostmtree(dtaL$features[trn, ], dtaL$time[trn], dtaL$id[trn], dtaL$y[trn],
          family = "Binary", M = M)
pred.0 <- predict(obj.0, dtaL$features[-trn, ], dtaL$time[-trn], dtaL$id[-trn], dtaL$y[-trn])

## End(Not run)
```

`spirometry` *Spirometry Data*

Description

Data consists of 9471 longitudinal evaluations of forced 1-second expiratory volume (FEV1-percentage of predicted) after lung transplant from 509 patients who underwent lung transplant (LTx) at the Cleveland Clinic. Twenty three patient/procedure variables were collected at the time of the transplant. The major objectives are to evaluate the temporal trend of FEV1 after LTx, and to identify factors associated with post-LTx FEV1 and assessing the differences in the trends after Single LTx versus Double LTx.

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` *Variable Importance*

Description

Calculate VIMP score for each of the individual covariates or a joint VIMP of multiple covariates.

Usage

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

Arguments

`object`	A boosting object of class (`boostmtree,grow`) or class (`boostmtree,predict`).
`x.names`	Names of the x-variables for which VIMP is requested. If NULL, VIMP is calcuated for all the covariates
`joint`	Estimate individual VIMP for each covariate from `x.names` or a joint VIMP for all covariates combine.

Details

Variable Importance (VIMP) is calcuated for each of the covariates individually or a joint VIMP is calulated for all the covariates specfied in `x.names`.

Author(s)

Hemant Ishwaran, Amol Pande and Udaya B. Kogalur

References

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

Examples

```
## Not run:
##------------------------------------------------------------
## Synthetic example (Response is continuous)
## VIMP is based on in-sample CV using out of bag data
##-------------------------------------------------------------
#simulate the data
dta <- simLong(n = 50, N = 5, rho =.80, model = 2,family = "Continuous")$dtaL

#basic boosting call
boost.grow <- boostmtree(dta$features, dta$time, dta$id, dta$y,
              family = "Continuous", M = 300,cv.flag = TRUE)
vimp.grow <- vimp.boostmtree(object = boost.grow,x.names=c("x1","x2"),joint = FALSE)
vimp.joint.grow <- vimp.boostmtree(object = boost.grow,x.names=c("x1","x2"),joint = TRUE)

##------------------------------------------------------------
## Synthetic example (Response is continuous)
## VIMP is based on test data
##-------------------------------------------------------------
#simulate the data
dtaO <- simLong(n = 100, ntest = 100, N = 5, rho =.80, model = 2, family = "Continuous")

## save the data as both a list and data frame
dtaL <- dtaO$dtaL
dta <- dtaO$dta

## get the training data
trn <- dtaO$trn

#basic boosting call
boost.grow <- boostmtree(dtaL$features[trn,], dtaL$time[trn], dtaL$id[trn], dtaL$y[trn],
              family = "Continuous", M = 300)
boost.pred <- predict(boost.grow,dtaL$features[-trn,], dtaL$time[-trn], dtaL$id[-trn],
              dtaL$y[-trn])
vimp.pred <- vimp.boostmtree(object = boost.pred,x.names=c("x1","x2"),joint = FALSE)
vimp.joint.pred <- vimp.boostmtree(object = boost.pred,x.names=c("x1","x2"),joint = TRUE)


## End(Not run)
```

vimpPlot *Variable Importance (VIMP) plot*

Description

Barplot displaying VIMP.

Usage

```
vimpPlot(vimp,
         Time_Interaction = TRUE,
         xvar.names = NULL,
         cex.xlab = NULL,
         ymaxlim = 0,
         ymaxtimelim = 0,
         subhead.cexval = 1,
         yaxishead = NULL,
         xaxishead = NULL,
         main = "Variable Importance (%)",
         col = grey(0.8),
         cex.lab = 1.5,
         subhead.labels = c("Time-Interactions Effects", "Main Effects"),
         ylbl = FALSE,
         seplim = NULL,
         eps = 0.1,
         Width_Bar = 1)
```

Arguments

`vimp`	VIMP values.
`Time_Interaction`	Whether VIMP is estimated from a longitudinal data, in which case VIMP is available for covariate and covariate-time interaction. Default is TRUE. If FALSE, VIMP is assumed to be estimated from a cross-sectional data.
`xvar.names`	Names of the covariates. If NULL, names are assigned as x1, x2,...,xp.
`cex.xlab`	Magnification of the names of the covariates above (and below) the barplot.
`ymaxlim`	By default, we use the range of the vimp values for the covariates for the ylim. If one wants to extend the ylim, add the amount with which the ylim will extend above.
`ymaxtimelim`	By default, we use the range of the vimp values for the covariates-time for the ylim. If one wants to extend the ylim, add the amount with which the ylim will extend below. Argument only works for the longitudinal setting.
`subhead.cexval`	Magnification of the `subhead.labels`. Argument only works for the longitudinal setting.
`yaxishead`	This represent a vector with two values which are points on the y-axis. Corresponding to the values, the lables for `subhead.labels` is shown. First argument corresponds to covariate-time interaction, whereas second argument is for the main effect. Argument only works for the longitudinal setting.
`xaxishead`	This represent a vector with two values which are points on the x-axis. Corresponding to the values, the lables for `subhead.labels` is shown. First argument corresponds to covariate-time interaction, whereas second argument is for the main effect. Argument only works for the longitudinal setting.
`main`	Main title for the plot.

col	Color of the plot.
cex.lab	Magnification of the x and y lables.
subhead.labels	Labels corresponding to the plot. Default is "Time-Interactions Effects" for the barplot below x-axis, and "Main Effects" for the barplot above x-axis.
ylbl	Should labels for the sub-headings be shown on left side of the y-axis.
seplim	if ylbl is TRUE, the distance between the lables of the sub-headings.
eps	Amount of gap between the top of the barplot and variable names.
Width_Bar	Width of the barplot.

Details

Barplot displaying VIMP. If the analysis is for the univariate case, VIMP is displayed above the x-axis. If the analysis is for the longitudinal case, VIMP for covariates (main effects) are shown above the x-axis while VIMP for covariate-time interactions (time interaction effects) are shown below the x-axis. In either case, negative vimp value is set to zero.

Author(s)

Hemant Ishwaran, Amol Pande and Udaya B. Kogalur

Examples

```
## Not run:
##------------------------------------------------------------
## Synthetic example
## high correlation, quadratic time with quadratic interaction
##-------------------------------------------------------------
#simulate the data
dta <- simLong(n = 50, N = 5, rho =.80, model = 2,family = "Continuous")$dtaL

#basic boosting call
boost.grow <- boostmtree(dta$features, dta$time, dta$id, dta$y,
              family = "Continuous",M = 300, cv.flag = TRUE)
vimp.grow <- vimp.boostmtree(object = boost.grow)

# VIMP plot
vimpPlot(vimp = vimp.grow, ymaxlim = 20, ymaxtimelim = 20,
         xaxishead = c(3,3), yaxishead = c(65,65),
         cex.xlab = 1, subhead.cexval = 1.2)

## End(Not run)
```

Index