Package 'mobForest'

January 3, 2018

Type Package

Title Model Based Random Forest Analysis

Version 1.3.0

Date 2018-01-03

Author Nikhil Garge [aut], Barry Eggleston [aut], Georgiy Bobashev [aut], Benjamin Carper [cre], Kasey Jones [ctb, cre], Torsten Hothorn [ctb], Kurt Hornik [ctb], Carolin Strobl [ctb], Achim Zeileis [ctb]

Maintainer Kasey Jones <krjones@rti.org>

Description Functions to implements random forest method for model based recursive partitioning. The mob() function, developed by Zeileis et al. (2008), within 'party' package, is modified to construct model-based decision trees based on random forests methodology. The main input function mobforest.analysis() takes all input parameters to construct trees, compute out-of-bag errors, predictions, and overall accuracy of forest. The algorithm performs parallel computation using cluster functions within 'parallel' package.

License GPL (>= 2)

Depends parallel (>= 3.4.1), party (>= 1.2-4), sandwich (>= 2.4.0), strucchange (>= 1.5-1), zoo (>= 1.8-0)

Imports methods, modeltools, stats, graphics

Suggests testthat (>= 1.0.2), mlbench (>= 2.1), lattice

RoxygenNote 6.0.1

NeedsCompilation no

Repository CRAN

Date/Publication 2018-01-03 18:45:25

R topics documented:

`bootstrap` *This method computes predicted outcome for each observation in the data frame using the tree model supplied as an input argument.*

Description

This method computes predicted outcome for each observation in the data frame using the tree model supplied as an input argument.

Usage

```
bootstrap(i, data, main_model, partition_vars, mtry, new_test_data,
  mobforest_controls, fraction, replace, model, family, prob_cutoff = 0.5)
```

Arguments

i	the tree
data	A data frame containing the variables in the model.
main_model	A model in character format
partition_vars	A vector of partition variables
mtry	A Random subset of partition variables to be considered at each
new_test_data	A data frame representing test data for validating random forest model. This data is not used in in tree building process.
mobforest_controls	An object of class "mobforest.control" returned by mobforest.control(), that contains parameters controlling the construction of random forest.
fraction	number of observations to draw without replacement (only relevant if replace = FALSE)
replace	TRUE or FALSE. replace = TRUE (default) performs bootstrapping. replace = FALSE performs sampling without replacement.
model	A model of class "StatModel" used for fitting observations in current node. This parameter allows fitting a linear model or generalized linear model with formula y ~ x_1 + ... + x_k. The Parameter "linearModel" fits linear model. The parameter "glinearModel" fits Poisson or logistic regression model depending upon the specification of parameter "family" (explained next). If "family" is specified as binomial() then logistic regression is performed. If the "family" is specified as poisson() then Poisson regression is performed.
family	A description of error distribution and link function to be used in the model. This parameter needs to be specified if generalized linear model is considered. The parameter "binomial()" is to be specified when logistic regression is considered and "poisson()" when Poisson regression is considered as the node model. The values allowed for this parameter are binomial() and poisson().
prob_cutoff	In case of logistic regression as a node model, the predicted probabilities for OOB cases are converted into classes (yes/no, high/low, etc as specified) based on this probability cutoff. If logistic regression is not considered as node model, the prob_cutoff = NULL. By default it is 0.5 when parameter not specified (and logistic regression considered).

Value

A list model performance metrics including R2/accuracy, predictions, MSE, and variable importance

Examples

```
## Not run:
formula <- as.formula(medv ~ lstat)
# load data
data("BostonHousing", package = "mlbench")
mobforest_controls <-
  mobforest.control(ntree = 1, mtry = 2, replace = TRUE,
```

```
                    alpha = 0.05, bonferroni = TRUE, minsplit = 25)

out <- bootstrap(i, data = BostonHousing, main_model = string.formula(formula),
                 partition_vars = partition_vars <- c("rad", "crim", "tax"),
                 mtry = 2, new_test_data = as.data.frame(matrix(0,0,0)),
                 mobforest_controls = mobforest_controls@mob_control, fraction = 1,
                 replace = TRUE, model = linearModel, family = "", prob_cutoff = .5)
out

## End(Not run)
```

`compute.acc` *Predictive accuracy estimates across trees for logistic regression model*

Description

Compute predictive accuracy of response variable with binary outcome. The function takes observed and predicted binary responses as input and computes proportion of observations classified in same group.

Usage

```
compute.acc(response, predictions, prob_cutoff = 0.5)
```

Arguments

`response`	A vector of binary outcome.
`predictions`	A matrix of predicted probabilities (logit model) for out-of-bag observations for each tree.
`prob_cutoff`	The threshold for predicting 1's & 0's.

Value

Predictive accuracy estimate ranging between 0 and 1.

Examples

```
response <- as.data.frame( c(rep(0, 10000), rep(1, 10000)))
predictions <-
  matrix(nrow = 20000, ncol = 3,
         data = c(rep(.1, 15000), rep(.8, 5000), rep(.1, 15000),
                  rep(.8, 5000), rep(.1, 15000), rep(.8, 5000)))
compute.acc(response, predictions, prob_cutoff = .5)
```

`compute.mse` *Predictive accuracy estimates (MSE) across trees for linear or poisson regression model.*

Description

Predictive accuracy estimates (MSE) across trees for linear or poisson regression model.

Usage

```
compute.mse(response, predictions)
```

Arguments

`response`	A vector of actual response of outcome variable.
`predictions`	A vector of predicted response for the same outcome variable.

Value

MSE estimates

Examples

```
# The MSE should be 2.5. Off by 2 half the time, off by 1 the other half
response <- matrix(c(rep(0,100), rep(10,100)))
predictions <-
    matrix(nrow=20, ncol = 3,
           data = c(rep(1,100), rep(8,100), rep(1,100), rep(8,100),
                    rep(1,100), rep(8,100)))
compute.mse(response, predictions)
```

`compute.r2` *Predictive accuracy estimates across trees for linear or poisson regression*

Description

pseudo R-square (R2) computation - proportion of total variance in response variable explained by the tree model. The function takes observed and predicted responses as input arguments and computes pseudo-R2 to determine how well the tree model fits the given data.

Usage

```
compute.r2(response, predictions)
```

Arguments

response	A vector of actual response of outcome variable.
predictions	A vector of predictions for the same outcome variable

Value

Predictive accuracy estimates ranging between 0 and 1.

Examples

```
# This example explains 90% of the variance
response <- matrix(c(rep(0, 100), rep(10, 100)))
predictions <-
  matrix(nrow = 200, ncol = 3,
         data = c(rep(1, 100), rep(8, 100), rep(1, 100), rep(8, 100),
                  rep(1, 100), rep(8, 100)))
compute.r2(response, predictions)
```

---

get.mf.object.glm *Fit a general linear model to a mobForest model*

Description

This method computes predicted outcome for each observation in the data frame using the tree model supplied as an input argument.

Usage

```
get.mf.object.glm(object, main_model, partition_vars, data, new_test_data,
  ntree, fam, prob_cutoff = 0.5)
```

Arguments

object	A bootstrap model object created by bootstrap()
main_model	A model in character format.
partition_vars	A vector of partition variables.
data	A data frame containing the variables in the model.
new_test_data	A data frame representing test data for validating random forest model. This data is not used in the tree building process
ntree	Number of trees to be constructed in forest (default = 300).
fam	A description of error distribution and link function to be used in the model. This parameter needs to be specified if generalized linear model is considered. The parameter "binomial()" is to be specified when logistic regression is considered and "poisson()" when Poisson regression is considered as the node model. The values allowed for this parameter are binomial() and poisson().

prob_cutoff	In case of logistic regression as a node model, the predicted probabilities for OOB cases are converted into classes (yes/no, high/low, etc as specified) based on this probability cutoff. If logistic regression is not considered as node model, the prob_cutoff = NULL. By default it is 0.5 when parameter not specified (and logistic regression considered).

Value

An object of class mobforest.output.

See Also

mobforest.control(), mobforest.output-class

---

get.mf.object.lm *Fit a linear model to a mobForest model*

Description

This method computes predicted outcome for each observation in the data frame using the tree model supplied as an input argument.

Usage

```
get.mf.object.lm(object, main_model, partition_vars, data, new_test_data, ntree,
  fam)
```

Arguments

object	A bootstrap model object created by bootstrap()
main_model	A model in character format.
partition_vars	A vector of partition variables.
data	A data frame containing the variables in the model.
new_test_data	A data frame representing test data for validating random forest model. This data is not used in in tree building process.
ntree	Number of trees to be constructed in forest (default = 300)
fam	A description of error distribution and link function to be used in the model. This parameter needs to be specified if generalized linear model is considered. The parameter "binomial()" is to be specified when logistic regression is considered and "poisson()" when Poisson regression is considered as the node model. The values allowed for this parameter are binomial() and poisson().

Value

An object of class mobforest.output.

See Also

mobforest.control(), mobforest.output-class

`get.pred.values` *Get predictions summarized across trees for out-of-bag cases or all cases for cases from new test data*

Description

Get predictions summarized across trees for out-of-bag cases or all cases for cases from new test data

Usage

```
get.pred.values(rf, OOB = T, newdata = F)
```

Arguments

`rf`	An object of class `mobforest.output`.
`OOB`	a logical determining whether to return predictions from the out-of-bag sample or the learning sample (not suggested).
`newdata`	a logical determining whether to return predictions from test data. If newdata = TRUE, then OOB argument is ignored.

Value

matrix with three columns: 1) Mean Predictions across trees, 2) Standard deviation of predictions across trees, and 3) Residual (mean predicted - observed). The third column is applicable only when linear regression is considered as the node model.

Examples

```
## Not run:
library(mlbench)
set.seed(1111)
# Random Forest analysis of model based recursive partitioning load data
data("BostonHousing", package = "mlbench")
BostonHousing <- BostonHousing[1:90, c("rad", "tax", "crim", "medv", "lstat")]

# Recursive partitioning based on linear regression model medv ~ lstat with 3
# trees.  1 core/processor used.
rfout <- mobforest.analysis(as.formula(medv ~ lstat), c("rad", "tax", "crim"),
    mobforest_controls = mobforest.control(ntree = 3, mtry = 2, replace = TRUE,
    alpha = 0.05, bonferroni = TRUE, minsplit = 25), data = BostonHousing,
    processors = 1, model = linearModel, seed = 1111)

# Obtain out-of-bag predicted values
OOB_pred_mat <- get.pred.values(rfout, OOB = TRUE)
```

```
OOB_pred = OOB_pred_mat[, 1]

## End(Not run)
```

`get.varimp` *Variable importance scores computed through random forest analysis*

Description

Variable importance scores computed through random forest analysis

Usage

```
get.varimp(rf)
```

Arguments

`rf`	An object of class `mobforest.output` returned by mobforest.analysis()

References

Leo Breiman (2001). Random Forests. *Machine Learning*, 45(1), 5-32.

Examples

```
## Not run:
library(mlbench)
set.seed(1111)
# Random Forest analysis of model based recursive partitioning load data
data("BostonHousing", package = "mlbench")
BostonHousing <- BostonHousing[1:90, c("rad", "tax", "crim", "medv", "lstat")]

# Recursive partitioning based on linear regression model medv ~ lstat with 3
# trees.  1 core/processor used.
rfout <- mobforest.analysis(as.formula(medv ~ lstat), c("rad", "tax", "crim"),
    mobforest_controls = mobforest.control(ntree = 3, mtry = 2, replace = TRUE,
        alpha = 0.05, bonferroni = TRUE, minsplit = 25), data = BostonHousing,
    processors = 1, model = linearModel, seed = 1111)
# Returns a vector of variable importance scores
get.varimp(rfout)

## End(Not run)
```

logistic.acc *Contingency table: Predicted vs. Observed Outcomes*

Description

This function takes predicted probabilities (for out of bag cases) obtained through logistic regression-based tree models and converts them into binary classes (based on specified probability threshold). The predicted classifications are then compared to actual binary response.

Usage

```
logistic.acc(response, predicted, prob_thresh = 0.5)
```

Arguments

response	A vector of binary classes of out-of-cases for a given tree.
predicted	A vector of predicted probabilities of out-of-cases using same tree.
prob_thresh	Probability threshold for classification (default = .5).

Examples

```
# We should get 15 predictions correct and miss 5
response <- matrix(c(rep(0,10), rep(1,10)))
predicted <- c(rep(.1,15), rep(.8,5))
logistic.acc(response, predicted, .5)
```

mob.rf.tree *Model based recursive partitioning - randomized subset of partition variables considered during each split.*

Description

The mob function in party package is modified so that a random subset of predictor variables are considered during each split. mtry represents the number of predictor variables to be considered during each split.

Usage

```
mob.rf.tree(main_model, partition_vars, mtry, weights, data = list(),
  na.action = na.omit, model = glinearModel, control = mob_control(), ...)
```

Arguments

main_model	A model in character format
partition_vars	A vector of partition variables
mtry	A Random subset of partition variables to be considered at each node of decision tree
weights	An optional vector of weights, as described in mob
data	A data frame containing the variables in the model.
na.action	A function which indicates what should happen when the data contain NAs, as described in mob
model	A model of class StatModel
control	A list with control parameters as returned by mob_control
...	Additional arguments passed to the fit call for the model.

Value

An object of class mob inheriting from BinaryTree. Every node of the tree is additionally associated with a fitted model.

References

Achim Zeileis, Torsten Hothorn, and Kurt Hornik (2008). Model-Based Recursive Partitioning. *Journal of Computational and Graphical Statistics*, 17(2), 492-514.

---

mobforest.analysis *Model-based random forest analysis*

Description

Main function that takes all the necessary arguments to start model-based random forest analysis.

Usage

```
mobforest.analysis(formula, partition_vars, data,
  mobforest_controls = mobforest.control(),
  new_test_data = as.data.frame(matrix(0, 0, 0)), processors = 1,
  model = linearModel, family = NULL, prob_cutoff = NULL,
  seed = sample(1:1e+07, 1))
```

Arguments

formula	An object of class formula specifying the model. This should be of type y ~ x_1 + ... + x_k, where the variables x_1, x_2, ..., x_k are predictor variables and y represents an outcome variable. This model is referred to as the node model
partition_vars	A character vector specifying the partition variables
data	An input dataset that is used for constructing trees in random forest.
mobforest_controls	An object of class "mobforest.control" returned by mobforest.control(), that contains parameters controlling the construction of random forest.
new_test_data	A data frame representing test data for validating random forest model. This data is not used in in tree building process.
processors	A number of processors/cores on your computer that should be used for parallel computation.
model	A model of class "StatModel" used for fitting observations in current node. This parameter allows fitting a linear model or generalized linear model with formula y ~ x_1 + ... + x_k. The Parameter "linearModel" fits linear model. The parameter "glinearModel" fits Poisson or logistic regression model depending upon the specification of parameter "family" (explained next). If "family" is specified as binomial() then logistic regression is performed. If the "family" is specified as poisson() then Poisson regression is performed.
family	A description of error distribution and link function to be used in the model. This parameter needs to be specified if generalized linear model is considered. The parameter "binomial()" is to be specified when logistic regression is considered and "poisson()" when Poisson regression is considered as the node model. The values allowed for this parameter are binomial() and poisson().
prob_cutoff	In case of logistic regression as a node model, the predicted probabilities for OOB cases are converted into classes (yes/no, high/low, etc as specified) based on this probability cutoff. If logistic regression is not considered as node model, the prob_cutoff = NULL. By default it is 0.5 when parameter not specified (and logistic regression considered).
seed	Since this function uses parallel processes, to replicate results, set the cluster "clusterSetRNGStream()" seed.

Details

mobforest.analysis is the main function that takes all the input parameters - model, partition variables, and forest control parameters - and starts the model-based random forest analysis. mobforest.analysis calls bootstrap function which constructs decision trees, computes out-of-bag (OOB) predictions, OOB predictive accuracy and perturbation in OOB predictive accuracy through permutation. bootstrap constructs trees on multiple cores/processors simultaneously through parallel computation. Later, the get.mf.object function wraps the analysis output into mobforest.output object.

Predictive accuracy estimates are computed using pseudo-R2 metric, defined as the proportion of total variation in outcome variable explained by a tree model on out-of-bag cases. R2 ranges from 0 to 1. R2 of zero suggests worst tree model (in terms of predicting outcome) and R2 of 1 suggests

perfect tree model.

Value

An object of class mobforest.output.

References

Achim Zeileis, Torsten Hothorn, and Kurt Hornik (2008). Model-Based Recursive Partitioning. *Journal of Computational and Graphical Statistics*, 17(2), 492-514.

Hothorn, T., Hornik, K. and Zeileis, A. (2006) Unbiased recursive partitioning: A conditional inference framework, *J Compute Graph Stat*, 15, 651-674.

Strobl, C., Malley, J. and Tutz, G. (2009) An introduction to recursive partitioning: rationale, application, and characteristics of classification and regression trees, bagging, and random forests, *Psychol Methods*, 14, 323-348.

See Also

mobforest.control(), mobforest.output-class

Examples

```
library(mlbench)
set.seed(1111)
# Random Forest analysis of model based recursive partitioning load data
data("BostonHousing", package = "mlbench")
BostonHousing <- BostonHousing[1:90, c("rad", "tax", "crim", "medv", "lstat")]

# Recursive partitioning based on linear regression model medv ~ lstat with 3
# trees.  1 core/processor used.
rfout <- mobforest.analysis(as.formula(medv ~ lstat), c("rad", "tax", "crim"),
    mobforest_controls = mobforest.control(ntree = 3, mtry = 2, replace = TRUE,
        alpha = 0.05, bonferroni = TRUE, minsplit = 25), data = BostonHousing,
    processors = 1, model = linearModel, seed = 1111)
## Not run:
rfout

## End(Not run)
```

`mobforest.control` *Control parameters for random forest*

Description

Various parameters that control the forest growing.

Usage

```
mobforest.control(ntree = 300, mtry = 0, replace = FALSE,
  fraction = 0.632, alpha = 1, bonferroni = FALSE, minsplit = 20,
  trim = 0.1, objfun = deviance, breakties = FALSE, parm = NULL,
  verbose = FALSE)
```

Arguments

`ntree`	Number of trees to be constructed in forest (default = 300).
`mtry`	Number of input variables randomly sampled as candidates at each node.
`replace`	logical. replace = TRUE (default) performs bootstrapping. replace = FALSE performs sampling without replacement.
`fraction`	Number of observations to draw without replacement (only relevant if replace = FALSE).
`alpha`	A node is considered for splitting if the p value for any partitioning variable in that node falls below alpha (default 0.05). Please see mob_control().
`bonferroni`	logical. Should p values be Bonferroni corrected? (default TRUE). Please see mob_control().
`minsplit`	An integer. The minimum number of observations in a node (default 20). Please see mob_control().
`trim`	A numeric, as defined in mob_control().
`objfun`	A function, as defined in mob_control().
`breakties`	A logical, as defined in mob_control().
`parm`	A numeric or vector, as defined in mob_control().
`verbose`	A logical, as defined in mob_control().

Details

This function is used to set up forest controls. The mob_control (from party 'package') object is used to set up control parameters for single tree model. For most parameters, please see: mob_control()

Value

An object of class `mobforest.control`.

References

Achim Zeileis, Torsten Hothorn, and Kurt Hornik (2008). Model-Based Recursive Partitioning. *Journal of Computational and Graphical Statistics*, 17(2), 492-514.

Examples

```
# create forest controls before starting random forest analysis
mobforest_control = mobforest.control(ntree = 400, mtry = 4, replace = TRUE,
    minsplit = 200)
```

---

mobforest.control-class

Class "mobforest.control" *of mobForest model*

Description

Control parameters for random forest

Objects from the Class

Objects can be created by mobforest.control.

Examples

```
# showClass("mobforest.control") The following code creates following forest
# controls: 400 trees to be constructed, sampling with replacement, a node
# contains at least 200 observations
mobforest_controls = mobforest.control(ntree = 400, mtry = 4,
    replace = TRUE, minsplit = 200)
```

---

mobforest.output *Model-based random forest object*

Description

Random Forest Output object that stores all the results including predictions, variable importance matrix, model, family of error distributions, and observed responses.

Usage

```
mobforest.output(oob_predictions, general_predictions, new_data_predictions,
  varimp_object, model_used, fam, train_response,
  new_response = data.frame(matrix(0, 0, 0)))
```

Arguments

oob_predictions	Predictions on out-of-bag data.
general_predictions	Predictions on learning data.
new_data_predictions	Predictions on new test data.
varimp_object	The variable importance object.
model_used	The model used.
fam	A description of the error distribution and link function to be used in the model.
train_response	Response outcome of training data.
new_response	Response outcome of test data.

See Also

prediction.output, varimp.output

---

mobforest.output-class

Class "mobforest.output" *of mobforest model*

Description

Random Forest output for model based recursive partitioning

Usage

```
## S4 method for signature 'mobforest.output'
show(object)
```

Arguments

object	object of class mobforest.output

Objects from the Class

Objects can be created by mobforest.output.

See Also

prediction.output, varimp.output

Examples

```
## Not run:
library(mlbench)
set.seed(1111)
# Random Forest analysis of model based recursive partitioning load data
data("BostonHousing", package = "mlbench")
BostonHousing <- BostonHousing[1:90, c("rad", "tax", "crim", "medv", "lstat")]

# Recursive partitioning based on linear regression model medv ~ lstat with 3
# trees.  1 core/processor used.
rfout <- mobforest.analysis(as.formula(medv ~ lstat), c("rad", "tax", "crim"),
    mobforest_controls = mobforest.control(ntree = 3, mtry = 2, replace = TRUE,
        alpha = 0.05, bonferroni = TRUE, minsplit = 25), data = BostonHousing,
    processors = 1, model = linearModel, seed = 1111)

## End(Not run)
```

---

mob_fit_checksplit *Utility Function. Taken from party package to remove ":::" warning*

Description

Utility Function. Taken from party package to remove ":::" warning

Usage

```
mob_fit_checksplit(split, weights, minsplit)
```

Arguments

split	see party package
weights	see party package
minsplit	see party package

---

mob_fit_childweights *Utility Function. Taken from party package to remove ":::" warning*

Description

Utility Function. Taken from party package to remove ":::" warning

Usage

```
mob_fit_childweights(node, mf, weights)
```

Arguments

node	see party package
mf	see party package
weights	see party package

---

mob_fit_fluctests *Utility Function. Taken from party package to remove ":::" warning*

Description

Utility Function. Taken from party package to remove ":::" warning

Usage

```
mob_fit_fluctests(obj, mf, minsplit, trim, breakties, parm)
```

Arguments

obj	see party package
mf	see party package
minsplit	see party package
trim	see party package
breakties	see party package
parm	cheese?

---

mob_fit_getlevels *Utility Function. Taken from party package to remove ":::" warning*

Description

Utility Function. Taken from party package to remove ":::" warning

Usage

```
mob_fit_getlevels(x)
```

Arguments

x	see party package

mob_fit_getobjfun *Utility Function. Taken from party package to remove ":::" warning*

Description

Utility Function. Taken from party package to remove ":::" warning

Usage

```
mob_fit_getobjfun(obj, mf, weights, left, objfun = deviance)
```

Arguments

obj	see party package
mf	see party package
weights	see party package
left	see party package
objfun	see party package

mob_fit_setupnode *Utility Function. Taken from party package to remove ":::" warning*

Description

Utility Function. Taken from party package to remove ":::" warning

Usage

```
mob_fit_setupnode(obj, mf, weights, control)
```

Arguments

obj	see party package
mf	see party package
weights	see party package
control	see party package

mob_fit_splitnode *Utility Function. Taken from party package to remove ":::" warning*

Description

Utility Function. Taken from party package to remove ":::" warning

Usage

```
mob_fit_splitnode(x, obj, mf, weights, minsplit, objfun, verbose = TRUE)
```

Arguments

x	see party package
obj	see party package
mf	see party package
weights	see party package
minsplit	see party package
objfun	see party package
verbose	To print or not to print

prediction.output *Predictions and predictive accuracy estimates*

Description

This function takes predictions and predictive accuracy estimates as input arguments and creates objects of class prediction.output.

Usage

```
prediction.output(pred_mean = numeric(), pred_sd = numeric(),
  residual = numeric(), R2 = numeric(), mse = numeric(),
  overall_r2 = numeric(), pred_type = character())
```

Arguments

pred_mean	Mean predictions across trees.
pred_sd	Standard deviation predictions across trees.
residual	Residuals (predicted outcome - observed outcome).
R2	Predictive accuracy across trees
mse	MSE across trees
overall_r2	Overall R2
pred_type	Out-of-bag data or test data or learning data.

Value

An object of class "prediction.output()".

See Also

prediction.output, mobforest.analysis

---

prediction.output-class *Class* "prediction.output" *of mobForest model*

Description

The object of this class stores predictions and predictive accuracy estimates.

Objects from the Class

Objects can be created by calls of the form prediction.output.

See Also

prediction.output, predictive.acc

---

predictive.acc *Predictive performance across all trees*

Description

Predictive performance across all trees

Usage

```
predictive.acc(object = "mfOutput", newdata = F, prob_cutoff = NULL,
  plot = T)
```

Arguments

object	An object of class mobforest.output
newdata	A logical value specifying if the performance needs to be summarized on test data supplied as new_test_data argument to mobforest.analysis function.
prob_cutoff	Predicted probabilities converted into classes (Yes/No, 1/0) based on this probability threshold. Only used for producing predicted Vs actual classes table.
plot	A logical value specifying if the use wishes to view performance plots

Value

A list with performance parameters

`oob_r2`	A vector of predictive accuracy estimates (ranging between 0 and 1) measured on Out-of-bag cases for each tree
`oob_mse`	A vector of MSE for Out-of-bag data for each tree. Valid only if the outcome is continuous.
`oob_overall_r2`	Overall predictive accuracy measured by combining Out-of-bag predictions across trees.
`oob_overall_mse`	Overall MSE measured by combining Out-of-bag predictions across trees.
`general_r2`	A vector of predictive accuracy (ranging between 0 and 1) measured on complete learning data for each tree
`general_mse`	A vector of MSE measured on complete learning data for each tree. Valid only if the outcome is continuous.
`general_overall_r2`	Overall predictive accuracy measured by combining predictions across trees.
`general_overall_mse`	Overall MSE measured by combining predictions across trees. Valid only if the outcome is continuous.
`model_used`	The node model and partition variables used for analysis
`family`	Error distribution assumptions of the model

Examples

```
## Not run:
library(mlbench)
set.seed(1111)
# Random Forest analysis of model based recursive partitioning load data
data("BostonHousing", package = "mlbench")
BostonHousing <- BostonHousing[1:90, c("rad", "tax", "crim", "medv", "lstat")]

# Recursive partitioning based on linear regression model medv ~ lstat with 3
# trees.  1 core/processor used.
rfout <- mobforest.analysis(as.formula(medv ~ lstat), c("rad", "tax", "crim"),
    mobforest_controls = mobforest.control(ntree = 3, mtry = 2, replace = T,
        alpha = 0.05, bonferroni = T, minsplit = 25), data = BostonHousing,
    processors = 1, model = linearModel, seed = 1111)

# get predictive performance estimates and produce a performance plot
pacc <- predictive.acc(rfout)

## End(Not run)
```

print.estimates *Predictive Accuracy Report*

Description

Predictive Accuracy Report

Usage

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

Arguments

x	An object of class 'predictive.acc' returned by "predictive.acc()" function
...	Additional arguments to print method

Examples

```
## Not run:
library(mlbench)
set.seed(1111)
# Random Forest analysis of model based recursive partitioning load data
data("BostonHousing", package = "mlbench")
BostonHousing <- BostonHousing[1:90, c("rad", "tax", "crim", "medv", "lstat")]

# Recursive partitioning based on linear regression model medv ~ lstat with 3
# trees.  1 core/processor used.
rfout <- mobforest.analysis(as.formula(medv ~ lstat), c("rad", "tax", "crim"),
    mobforest_controls = mobforest.control(ntree = 3, mtry = 2, replace = T,
        alpha = 0.05, bonferroni = T, minsplit = 25), data = BostonHousing,
    processors = 1, model = linearModel, seed = 1111)
# prints predictive accuracy output
pacc <- predictive.acc(rfout)

## End(Not run)
```

residual.plot *Produces two plots: a) histogram of residuals, b) predicted Vs residuals. This feature is applicable only when linear regression is considered as the node model.*

Description

Residuals are computed as difference between the predicted values of outcome (summarized across all trees) and observed values of outcome. The residual plots are typical when the fitted values are obtained through linear regression but not when logistic or Poisson regression is considered as a node model. Therefore, the residual plots are produced only when linear regression is considered. For logistic or Poisson models, a message is printed saying "Residual Plot not produced when logistic of Poisson regression is considered as the node model".

Usage

```
residual.plot(object, breaks = 50)
```

Arguments

object	An object of class 'mobforest.output'
breaks	Integer for number of breaks in histogram

Examples

```
## Not run:
library(mlbench)
set.seed(1111)
# Random Forest analysis of model based recursive partitioning load data
data("BostonHousing", package = "mlbench")
BostonHousing <- BostonHousing[1:90, c("rad", "tax", "crim", "medv", "lstat")]

# Recursive partitioning based on linear regression model medv ~ lstat with 3
# trees.  1 core/processor used.
rfout <- mobforest.analysis(as.formula(medv ~ lstat), c("rad", "tax", "crim"),
    mobforest_controls = mobforest.control(ntree = 3, mtry = 2, replace = T,
        alpha = 0.05, bonferroni = T, minsplit = 25), data = BostonHousing,
    processors = 1, model = linearModel, seed = 1111)
# get predictive performance estimates and produce a performance plot
residualPlot(rfout)

## End(Not run)
```

---

string.formula *Model in the formula object converted to a character*

Description

Model in the formula object converted to a character

Usage

```
string.formula(formula)
```

Arguments

formula	formula object

Value

character. model

Examples

```
aformula <- as.formula(medv ~ lstat)
astring <- string.formula(aformula)
print(astring)
```

---

tree.predictions *Predictions from tree model*

Description

This method computes predicted outcome for each observation in the data frame using the tree model supplied as an input argument.

Usage

```
tree.predictions(j, df, tree)
```

Arguments

j	the observation
df	A data frame containing the variables in the model.
tree	An object of class mob inheriting from BinaryTree

Value

A vector of predicted outcome

Examples

```
library(mlbench)
set.seed(1111)
# Random Forest analysis of model based recursive partitioning load data
data("BostonHousing", package = "mlbench")
data <- BostonHousing[1:90, c("rad", "tax", "crim", "medv", "lstat")]
fmBH <- mob.rf.tree(main_model = "medv ~ lstat",
                    partition_vars = c("rad", "tax", "crim"), mtry = 2,
                    control = mob_control(), data = data,
                    model = linearModel)
tree.predictions(j = 1, df = data, tree = fmBH@tree)
```

varimp.output — *Variable importance matrix containing the decrease in predictive accuracy after permuting the variables across all trees*

Description

Values of variable 'm' in the oob cases are randomly permuted and R2 obtained through variable-m-permuted oob data is subtracted from R2 obtained on untouched oob data. The average of this number over all the trees in the forest is the raw importance score for variable m.

Usage

```
varimp.output(varimp_matrix)
```

Arguments

varimp_matrix	a matrix containing decrease in predictive accuracy for all variables for each tree

Value

An object of class varimp.output.

References

Strobl, C., Malley, J. and Tutz, G. (2009) An introduction to recursive partitioning: rationale, application, and characteristics of classification and regression trees, bagging, and random forests, *Psychol Methods*, 14, 323-348.

varimp.output-class — *Class "varimp.output" of mobforest model*

Description

Variable importance

Objects from the Class

Objects can be created by calls of the form varimp.output.

References

Strobl, C., Malley, J. and Tutz, G. (2009) An introduction to recursive partitioning: rationale, application, and characteristics of classification and regression trees, bagging, and random forests, *Psychol Methods*, 14, 323-348.

See Also

varimp.output

---

varimplot *A plot with variable importance score on X-axis and variable name on Y-axis.*

Description

A plot with variable importance score on X-axis and variable name on Y-axis.

Usage

```
varimplot(object)
```

Arguments

object	An object of class mobforest.output returned by mobforest.analysis()

References

Leo Breiman (2001). Random Forests. *Machine Learning*, 45(1), 5-32.

See Also

get.varimp

Examples

```
## Not run:
library(mlbench)
set.seed(1111)
# Random Forest analysis of model based recursive partitioning load data
data("BostonHousing", package = "mlbench")
BostonHousing <- BostonHousing[1:90, c("rad", "tax", "crim", "medv", "lstat")]

# Recursive partitioning based on linear regression model medv ~ lstat with 3
# trees.  1 core/processor used.
rfout <- mobforest.analysis(as.formula(medv ~ lstat), c("rad", "tax", "crim"),
    mobforest_controls = mobforest.control(ntree = 3, mtry = 2, replace = T,
        alpha = 0.05, bonferroni = T, minsplit = 25), data = BostonHousing,
    processors = 1, model = linearModel, seed = 1111)
 varimplot(rfout)

## End(Not run)
```

Index