

# Package ‘eat’

April 9, 2021

**Title** Efficiency Analysis Trees

**Version** 0.1.0

**Description** Functions are provided to determine production frontiers and technical efficiency measures through non-parametric techniques based upon regression trees. The package includes code for estimating radial input, output, directional and additive measures, plotting graphical representations of the scores and the production frontiers by means of trees, and determining rankings of importance of input variables in the analysis. Additionally, an adaptation of Random Forest by a set of individual Efficiency Analysis Trees for estimating technical efficiency is also included. More details in: <doi:10.1016/j.eswa.2020.113783>.

**License** GPL-3

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.1.1

**Imports** dplyr, conflicted, stats, knitr, ggplot2, ggparty, partykit, ggrepel, Rdpack, lpSolveAPI, utils, reshape2

**Suggests** rmarkdown, testthat, kableExtra, usethis, devtools

**RdMacros** Rdpack

**VignetteBuilder** knitr

**Depends** R (>= 2.10)

**URL** <https://efficiencytools.wordpress.com/>

**BugReports** <https://github.com/MiriamEsteve/EAT/issues>

**NeedsCompilation** no

**Author** Miriam Esteve [cre, aut] (<<https://orcid.org/0000-0002-5908-0581>>),  
V́ctor Espa~na [aut],  
Juan Aparicio [aut] (<<https://orcid.org/0000-0002-0867-0004>>),  
Xavier Barber [aut] (<<https://orcid.org/0000-0003-3079-5855>>)

**Maintainer** Miriam Esteve <[miriam.estevec@umh.es](mailto:miriam.estevec@umh.es)>

**Repository** CRAN

**Date/Publication** 2021-04-09 08:50:02 UTC

**R topics documented:**

alpha . . . . .	3
bagging . . . . .	4
barplot_importance . . . . .	4
bestEAT . . . . .	5
bestRFEAT . . . . .	6
CEAT_BCC_in . . . . .	7
CEAT_BCC_out . . . . .	8
CEAT_DDF . . . . .	9
CEAT_RSL_in . . . . .	9
CEAT_RSL_out . . . . .	10
CEAT_WAM . . . . .	11
checkEAT . . . . .	11
comparePareto . . . . .	12
deepEAT . . . . .	12
descrEAT . . . . .	13
EAT . . . . .	13
EAT_BCC_in . . . . .	16
EAT_BCC_out . . . . .	16
EAT_DDF . . . . .	17
EAT_object . . . . .	18
EAT_RSL_in . . . . .	19
EAT_RSL_out . . . . .	19
EAT_WAM . . . . .	20
efficiencyCEAT . . . . .	21
efficiencyDensity . . . . .	22
efficiencyEAT . . . . .	23
efficiencyJitter . . . . .	24
efficiencyRFEAT . . . . .	25
estimEAT . . . . .	26
frontier . . . . .	27
frontier.levels . . . . .	28
generateLv . . . . .	28
imp_var_EAT . . . . .	29
imp_var_RFEAT . . . . .	29
isFinalNode . . . . .	30
layout . . . . .	30
mse . . . . .	31
mtry_inputSelection . . . . .	31
M_Breiman . . . . .	32
PISAindex . . . . .	32
plotEAT . . . . .	33
plotRFEAT . . . . .	34
posIdNode . . . . .	35
predictEAT . . . . .	35
predictFDH . . . . .	36
predictor . . . . .	36

predictRFEAT . . . . .	37
preProcess . . . . .	38
RandomEAT . . . . .	39
rankingEAT . . . . .	39
rankingRFEAT . . . . .	40
RBranch . . . . .	41
RCV . . . . .	41
RFEAT . . . . .	42
RFEAT_object . . . . .	43
RF_predictor . . . . .	44
scores . . . . .	45
selectTk . . . . .	45
select_mtry . . . . .	46
SERules . . . . .	47
size . . . . .	47
split . . . . .	48
split_forest . . . . .	49
treesForRCV . . . . .	49
X2Y2.sim . . . . .	50
Y1.sim . . . . .	50

<b>Index</b>	<b>51</b>
--------------	-----------

---

alpha

*Alpha Calculation for Pruning Procedure*

---

### Description

This function gets the minimum alpha for each subtree evaluated during the pruning procedure of Efficiency Analysis Trees.

### Usage

```
alpha(tree)
```

### Arguments

tree            A list containing EAT nodes.

### Value

Numeric value corresponding to the minimum alpha associated with a suitable node to be pruned.

---

bagging

*Bagging data*

---

### Description

Bootstrap aggregating for data.

### Usage

```
bagging(data, x, y)
```

### Arguments

data	Dataframe containing the variables in the model.
x	Column input indexes in data.
y	Column output indexes in data.

### Value

List containing training dataframe and list with binary response as 0 if the observations have been selected for training and 0 in any other case.

---

barplot\_importance

*Barplot Variable Importance*

---

### Description

This function generates a barplot with the importance of each predictor.

### Usage

```
barplot_importance(m, threshold)
```

### Arguments

m	Dataframe with the importance of each predictor.
threshold	Importance score value in which a line should be graphed.

### Value

Barplot representing each variable on the x-axis and its importance on the y-axis.

**Description**

This function calculates the root mean square error (RMSE) for an Efficiency Analysis Trees model built with a set of given hyperparameters.

**Usage**

```
bestEAT(  
  training,  
  test,  
  x,  
  y,  
  numStop = 5,  
  fold = 5,  
  max.depth = NULL,  
  max.leaves = NULL,  
  na.rm = TRUE  
)
```

**Arguments**

<code>training</code>	Training dataframe or matrix containing the variables for model construction.
<code>test</code>	Test dataframe or matrix containing the variables for model assessment.
<code>x</code>	Vector. Column input indexes in data.
<code>y</code>	Vector. Column output indexes in data.
<code>numStop</code>	Vector. Set of minimum number of observations in a node for a split to be attempted.
<code>fold</code>	Vector. Set of number of folds in which the dataset to apply cross-validation during the pruning is divided.
<code>max.depth</code>	Integer. Depth of the tree.
<code>max.leaves</code>	Integer. Maximum number of leaf nodes.
<code>na.rm</code>	Logical. If TRUE, NA rows are omitted.

**Value**

Dataframe in which each row corresponds to a given set of hyperparameters and the root mean square error (RMSE).

## Examples

```

data("PISAindex")

n <- nrow(PISAindex) # Observations in the dataset
selected <- sample(1:n, n * 0.7) # Training indexes
training <- PISAindex[selected, ] # Training set
test <- PISAindex[-selected, ] # Test set

bestEAT(training = training,
        test = test,
        x = 6:9,
        y = 3,
        numStop = c(3, 5, 7),
        fold = c(5, 7, 10))

```

---

bestRFEAT

*Tuning a Random Forest + Efficiency Analysis Trees model*

---

## Description

This function calculates the root mean square error (RMSE) for a Random Forest + Efficiency Analysis Trees model built with a set of given hyperparameters.

## Usage

```

bestRFEAT(
  training,
  test,
  x,
  y,
  numStop = 5,
  m = 50,
  s_mtry = c("5", "BRM"),
  na.rm = TRUE
)

```

## Arguments

training	Training dataframe or matrix containing the variables for model construction.
test	Test dataframe or matrix containing the variables for model assessment.
x	Vector. Column input indexes in data.
y	Vector. Column output indexes in data.
numStop	Vector. Set of minimum number of observations in a node for a split to be attempted.

m	Vector. Set of number of trees to be built.
s_mtry	Character vector. Set of options for selecting the number of inputs to be selected in each split.
na.rm	Logical. If TRUE, NA rows are omitted.

### Value

Dataframe in which each row corresponds to a given set of hyperparameters and the root mean square error (RMSE).

### Examples

```
data("PISAindex")

n <- nrow(PISAindex) # Observations in the dataset
selected <- sample(1:n, n * 0.7) # Training indexes
training <- PISAindex[selected, ] # Training set
test <- PISAindex[- selected, ] # Test set

bestRFEAT(training = training,
           test = test,
           x = 6:9,
           y = 3,
           numStop = c(3, 5),
           m = c(20, 30),
           s_mtry = c("1", "BRM"))
```

---

CEAT_BCC_in	<i>Banker, Charnes and Cooper programming model with input orientation for a convex Efficiency Analysis Tree model</i>
-------------	--

---

### Description

Banker, Charnes and Cooper programming model with input orientation for a convex Efficiency Analysis Tree model.

### Usage

```
CEAT_BCC_in(j, scores, x_k, y_k, atreeTk, ytreeTk, nX, nY, N_leaves)
```

### Arguments

j	Integer. Number of DMUs.
scores	Matrix. Empty matrix for scores.
x_k	Dataframe. Set of input variables.

y_k	Dataframe. Set of output variables.
atreeTk	Matrix. Set of "a" Pareto-coordinates.
ytreeTk	Matrix. Set of predictions.
nX	Integer. Number of inputs.
nY	Integer. Number of outputs.
N_leaves	Integer. Number of leaf nodes.

**Value**

A numerical vector with scores.

---

CEAT_BCC_out	<i>Banker, Charnes and Cooper programming model with output orientation for a convex Efficiency Analysis Tree model</i>
--------------	---

---

**Description**

Banker, Charnes and Cooper programming model with output orientation for a convex Efficiency Analysis Tree model.

**Usage**

CEAT\_BCC\_out(j, scores, x\_k, y\_k, atreeTk, ytreeTk, nX, nY, N\_leaves)

**Arguments**

j	Integer. Number of DMUs.
scores	Matrix. Empty matrix for scores.
x_k	Dataframe. Set of input variables.
y_k	Dataframe. Set of output variables.
atreeTk	Matrix. Set of "a" Pareto-coordinates.
ytreeTk	Matrix. Set of predictions.
nX	Integer. Number of inputs.
nY	Integer. Number of outputs.
N_leaves	Integer. Number of leaf nodes.

**Value**

A numerical vector with efficiency scores.



---

CEAT_DDF	<i>Directional Distance Function mathematical programming model for a convex Efficiency Analysis Tree model</i>
----------	---

---

**Description**

Directional Distance Function for a convex Efficiency Analysis Tree model.

**Usage**

CEAT\_DDF(j, scores, x\_k, y\_k, atreeTk, ytreeTk, nX, nY, N\_leaves)

**Arguments**

j	Integer. Number of DMUs.
scores	Matrix. Empty matrix for scores.
x_k	Dataframe. Set of input variables.
y_k	Dataframe. Set of output variables.
atreeTk	Matrix. Set of "a" Pareto-coordinates.
ytreeTk	Matrix. Set of predictions.
nX	Integer. Number of inputs.
nY	Integer. Number of outputs.
N_leaves	Integer. Number of leaf nodes.

**Value**

A numerical vector with scores.

---

CEAT_RSL_in	<i>Russell Model with input orientation for a convex Efficiency Analysis Tree model</i>
-------------	---

---

**Description**

Russell Model with input orientation for a convex Efficiency Analysis Tree model.

**Usage**

CEAT\_RSL\_in(j, scores, x\_k, y\_k, atreeTk, ytreeTk, nX, nY, N\_leaves)

**Arguments**

j	Integer. Number of DMUs.
scores	Matrix. Empty matrix for scores.
x_k	Dataframe. Set of input variables.
y_k	Dataframe. Set of output variables.
atreeTk	Matrix. Set of "a" Pareto-coordinates.
ytreeTk	Matrix. Set of predictions.
nX	Integer. Number of inputs.
nY	Integer. Number of outputs.
N_leaves	Integer. Number of leaf nodes.

**Value**

A numerical vector with scores.

---

CEAT_RSL_out	<i>Russell Model with output orientation for a convex Efficiency Analysis Tree model</i>
--------------	--

---

**Description**

Russell Model with output orientation for a convex Efficiency Analysis Tree model.

**Usage**

```
CEAT_RSL_out(j, scores, x_k, y_k, atreeTk, ytreeTk, nX, nY, N_leaves)
```

**Arguments**

j	Integer. Number of DMUs.
scores	Matrix. Empty matrix for scores.
x_k	Dataframe. Set of input variables.
y_k	Dataframe. Set of output variables.
atreeTk	Matrix. Set of "a" Pareto-coordinates.
ytreeTk	Matrix. Set of predictions.
nX	Integer. Number of inputs.
nY	Integer. Number of outputs.
N_leaves	Integer. Number of leaf nodes.

**Value**

A numerical vector with scores.

---

CEAT_WAM	<i>Weighted Additive Model for a convex Efficiency Analysis Tree model</i>
----------	--

---

**Description**

Weighted Additive Model for a convex Efficiency Analysis Tree model.

**Usage**

CEAT\_WAM(j, scores, x\_k, y\_k, atreeTk, ytreeTk, nX, nY, N\_leaves, weights)

**Arguments**

j	Integer. Number of DMUs.
scores	Matrix. Empty matrix for scores.
x_k	Dataframe. Set of input variables.
y_k	Dataframe. Set of output variables.
atreeTk	Matrix. Set of "a" Pareto-coordinates.
ytreeTk	Matrix. Set of predictions.
nX	Integer. Number of inputs.
nY	Integer. Number of outputs.
N_leaves	Integer. Number of leaf nodes.
weights	Character. "MIP" for Measure of Inefficiency Proportion or "RAM" for Range Adjusted Measure of Inefficiency.

**Value**

A numerical vector with scores.

---

checkEAT	<i>Check Efficiency Analysis Trees.</i>
----------	---

---

**Description**

This function verifies if a specific tree keeps to Pareto-dominance properties.

**Usage**

checkEAT(tree)

**Arguments**

tree	A list containing EAT nodes.
------	------------------------------

**Value**

Message indicating if the tree is acceptable or warning in case of breaking any Pareto-dominance relationship.

---

comparePareto	<i>Pareto-dominance relationships</i>
---------------	---------------------------------------

---

**Description**

This function denotes if a node dominates another one or if there is no Pareto-dominance relationship.

**Usage**

```
comparePareto(t1, t2)
```

**Arguments**

t1	A first node.
t2	A second node.

**Value**

-1 if t1 dominates t2, 1 if t2 dominates t1 and 0 if there are no Pareto-dominance relationships.

---

deepEAT	<i>Deep Efficiency Analysis Trees</i>
---------	---------------------------------------

---

**Description**

This function creates a deep tree and a set of possible prunings by pruning through the weakest-link pruning procedure.

**Usage**

```
deepEAT(data, x, y, numStop = 5, max.depth = NULL, max.leaves = NULL)
```

**Arguments**

data	Dataframe or matrix containing the variables in the model.
x	Vector. Column input indexes in data.
y	Vector. Column output indexes in data.
numStop	Integer. Minimum number of observations in a node for a split to be attempted.
max.depth	Integer. Depth of the tree.
max.leaves	Integer. Maximum number of leaf nodes.

**Value**

List containing each possible pruning for the deep tree and its associated alpha value.

---

descrEAT

*Leaf Nodes Descriptive*

---

**Description**

This function returns a set of common measures for the leaf nodes of an Efficiency Analysis Trees model.

**Usage**

```
descrEAT(object)
```

**Arguments**

object            An EAT object.

**Value**

List with centralization and dispersion measures and the root mean square error (RMSE) for each node. In case of a single output, the result of the function is a data frame.

**Examples**

```
simulated <- Y1.sim(N = 50, nX = 3)
model <- EAT(data = simulated, x = c(1, 2, 3), y = 4, numStop = 10, fold = 5)
descrEAT(model)
```

---

EAT

*Efficiency Analysis Trees*

---

**Description**

This function estimates a stepped production frontier through regression trees.

**Usage**

```
EAT(
  data,
  x,
  y,
  numStop = 5,
  fold = 5,
  max.depth = NULL,
  max.leaves = NULL,
  na.rm = TRUE
)
```

**Arguments**

<code>data</code>	Data frame or matrix containing the variables in the model.
<code>x</code>	Vector. Column input indexes in data.
<code>y</code>	Vector. Column output indexes in data.
<code>numStop</code>	Integer. Minimum number of observations in a node for a split to be attempted.
<code>fold</code>	Integer. Set of number of folds in which the dataset to apply cross-validation during the pruning is divided.
<code>max.depth</code>	Integer. Depth of the tree.
<code>max.leaves</code>	Integer. Maximum number of leaf nodes.
<code>na.rm</code>	Logical. If TRUE, NA rows are omitted.

**Details**

The EAT function generates a regression tree model based on CART (Breiman et al. 1984) under a new approach that guarantees obtaining a stepped production frontier that fulfills the property of free disposability. This frontier shares the aforementioned aspects with the FDH frontier (Deprins and Simar 1984) but enhances some of its disadvantages such as the overfitting problem or the underestimation of technical inefficiency. More details in Esteve et al. (2020).

**Value**

An EAT object containing:

- `data`
  - `df`: data frame containing the variables in the model.
  - `x`: input indexes in data.
  - `y`: output indexes in data.
  - `input_names`: input variable names.
  - `output_names`: output variable names.
  - `row_names`: rownames in data.
- `control`
  - `fold`: fold hyperparameter value.

- numStop: numStop hyperparameter value.
- max.leaves: max.leaves hyperparameter value.
- max.depth: max.depth hyperparameter value.
- na.rm: na.rm hyperparameter value.
- tree: list structure containing the EAT nodes.
- nodes\_df: data frame containing the following information for each node.
  - id: node index.
  - SL: left child node index.
  - N: number of observations at the node.
  - Proportion: proportion of observations at the node.
  - the output predictions.
  - R: the error at the node.
  - index: observation indexes at the node.
- model
  - nodes: total number of nodes at the tree.
  - leaf\_nodes: number of leaf nodes at the tree.
  - a: lower bound of the nodes.
  - y: output predictions.

## References

Breiman L, Friedman J, Stone CJ, Olshen RA (1984). *Classification and regression trees*. CRC press.

Deprins D, Simar L (1984). “Measuring labor efficiency in post offices, The Performance of Public Enterprises: Concepts and Measurements, M. Marchand, P. Pestieau and H. Tulkens.”

Esteve M, Aparicio J, Rabasa A, Rodriguez-Sala JJ (2020). “Efficiency analysis trees: A new methodology for estimating production frontiers through decision trees.” *Expert Systems with Applications*, **162**.

## Examples

```
# ===== #
# Single output scenario #
# ===== #

simulated <- Y1.sim(N = 50, nX = 3)
EAT(data = simulated, x = c(1, 2, 3), y = 4, numStop = 10, fold = 5, max.leaves = 6)

# ===== #
# Multi output scenario #
# ===== #

simulated <- X2Y2.sim(N = 50, border = 0.1)
EAT(data = simulated, x = c(1,2), y = c(3, 4), numStop = 10, fold = 7, max.depth = 7)
```

---

EAT_BCC_in	<i>Banker, Charnes and Cooper Programming Model with Input Orientation for an Efficiency Analysis Trees model</i>
------------	---

---

**Description**

Banker, Charnes and Cooper programming model with input orientation for an Efficiency Analysis Tree model.

**Usage**

EAT\_BCC\_in(j, scores, x\_k, y\_k, atreeTk, ytreeTk, nX, nY, N\_leaves)

**Arguments**

j	Integer. Number of DMUs.
scores	Matrix. Empty matrix for scores.
x_k	Dataframe. Set of input variables.
y_k	Dataframe. Set of output variables.
atreeTk	Matrix. Set of "a" Pareto-coordinates.
ytreeTk	Matrix. Set of predictions.
nX	Integer. Number of inputs.
nY	Integer. Number of outputs.
N_leaves	Integer. Number of leaf nodes.

**Value**

A numerical vector with efficiency scores.

---

EAT_BCC_out	<i>Banker, Charnes and Cooper Programming Model with Output Orientation for an Efficiency Analysis Trees model</i>
-------------	--

---

**Description**

Banker, Charnes and Cooper programming model with output orientation for an Efficiency Analysis Tree model.

**Usage**

EAT\_BCC\_out(j, scores, x\_k, y\_k, atreeTk, ytreeTk, nX, nY, N\_leaves)



**Arguments**

j	Integer. Number of DMUs.
scores	Matrix. Empty matrix for scores.
x_k	Dataframe. Set of input variables.
y_k	Dataframe. Set of output variables.
atreeTk	Matrix. Set of "a" Pareto-coordinates.
ytreeTk	Matrix. Set of predictions.
nX	Integer. Number of inputs.
nY	Integer. Number of outputs.
N_leaves	Integer. Number of leaf nodes.

**Value**

A numerical vector with efficiency scores.

---

EAT_DDF	<i>Directional Distance Function Programming Model for an Efficiency Analysis Trees model</i>
---------	---

---

**Description**

Directional Distance Function for an Efficiency Analysis Tree model.

**Usage**

EAT\_DDF(j, scores, x\_k, y\_k, atreeTk, ytreeTk, nX, nY, N\_leaves)

**Arguments**

j	Integer. Number of DMUs.
scores	Matrix. Empty matrix for scores.
x_k	Dataframe. Set of input variables.
y_k	Dataframe. Set of output variables.
atreeTk	Matrix. Set of "a" Pareto-coordinates.
ytreeTk	Matrix. Set of predictions.
nX	Integer. Number of inputs.
nY	Integer. Number of outputs.
N_leaves	Integer. Number of leaf nodes.

**Value**

A numerical vector with efficiency scores.

---

EAT\_object

*EAT object*


---

### Description

This function saves information about the EAT model

### Usage

```
EAT_object(
  data,
  x,
  y,
  register_names,
  numStop,
  fold,
  max.depth,
  max.leaves,
  na.rm,
  tree
)
```

### Arguments

<code>data</code>	Dataframe or matrix containing the variables in the model.
<code>x</code>	Vector. Column input indexes in data.
<code>y</code>	Vector. Column output indexes in data.
<code>register_names</code>	String vector. Data rownames.
<code>numStop</code>	Integer. Minimum number of observations in a node for a split to be attempted.
<code>fold</code>	Integer. Set of number of folds in which the dataset to apply cross-validation during the pruning is divided.
<code>max.depth</code>	Integer. Maximum number of leaf nodes.
<code>max.leaves</code>	Integer. Depth of the tree.
<code>na.rm</code>	Logical. If True, NA rows are omitted. If False, an error occurs in case of NA rows.
<code>tree</code>	A list containing the nodes of the EAT pruned tree.

### Value

An EAT object

---

EAT_RSL_in	<i>Russell Model with Input Orientation for an Efficiency Analysis Trees model</i>
------------	--

---

**Description**

Russell Model with input orientation for an Efficiency Analysis Trees model.

**Usage**

EAT\_RSL\_in(j, scores, x\_k, y\_k, atreeTk, ytreeTk, nX, nY, N\_leaves)

**Arguments**

j	Integer. Number of DMUs.
scores	Matrix. Empty matrix for scores.
x_k	Dataframe. Set of input variables.
y_k	Dataframe. Set of output variables.
atreeTk	Matrix. Set of "a" Pareto-coordinates.
ytreeTk	Matrix. Set of predictions.
nX	Integer. Number of inputs.
nY	Integer. Number of outputs.
N_leaves	Integer. Number of leaf nodes.

**Value**

A numerical vector with efficiency scores.

---

EAT_RSL_out	<i>Russell Model with Output Orientation for an Efficiency Analysis Trees model</i>
-------------	---

---

**Description**

Russell Model with output orientation for an Efficiency Analysis Tree model.

**Usage**

EAT\_RSL\_out(j, scores, x\_k, y\_k, atreeTk, ytreeTk, nX, nY, N\_leaves)

**Arguments**

j	Integer. Number of DMUs.
scores	Matrix. Empty matrix for scores.
x_k	Dataframe. Set of input variables.
y_k	Dataframe. Set of output variables.
atreeTk	Matrix. Set of "a" Pareto-coordinates.
ytreeTk	Matrix. Set of predictions.
nX	Integer. Number of inputs.
nY	Integer. Number of outputs.
N_leaves	Integer. Number of leaf nodes.

**Value**

A numerical vector with efficiency scores.

---

EAT\_WAM

*Weighted Additive Model for an Efficiency Analysis Trees model*

---

**Description**

Weighted Additive Model for an Efficiency Analysis Trees model.

**Usage**

EAT\_WAM(j, scores, x\_k, y\_k, atreeTk, ytreeTk, nX, nY, N\_leaves, weights)

**Arguments**

j	Integer. Number of DMUs.
scores	Matrix. Empty matrix for scores.
x_k	Dataframe. Set of input variables.
y_k	Dataframe. Set of output variables.
atreeTk	Matrix. Set of "a" Pareto-coordinates.
ytreeTk	Matrix. Set of predictions.
nX	Integer. Number of inputs.
nY	Integer. Number of outputs.
N_leaves	Integer. Number of leaf nodes.
weights	Character. "MIP" for Measure of Inefficiency Proportion or "RAM" for Range Adjusted Measure of Inefficiency.

**Value**

A numerical vector with efficiency scores.

---

efficiencyCEAT      *Convex Efficiency Analysis Tree Efficiency Scores*

---

### Description

This function calculates the efficiency scores for each DMU through a convex Efficiency Analysis Trees model.

### Usage

```
efficiencyCEAT(
  data,
  x,
  y,
  object,
  scores_model,
  digits = 3,
  DEA = TRUE,
  na.rm = TRUE
)
```

### Arguments

<code>data</code>	Dataframe containing the DMU for which the efficiency score is calculated.
<code>x</code>	Vector. Column input indexes in data.
<code>y</code>	Vector. Column output indexes in data.
<code>object</code>	An EAT object.
<code>scores_model</code>	Mathematical programming model to calculate scores. <ul style="list-style-type: none"> <li>• <code>BCC.OUT</code> BCC model. Output-oriented.</li> <li>• <code>BCC.INP</code> BCC model. Input-oriented.</li> <li>• <code>DDF</code> Directional Distance Function.</li> <li>• <code>RSL.OUT</code> Russell model. Output-oriented.</li> <li>• <code>RSL.INP</code> Russell model. Input-oriented.</li> <li>• <code>WAM.MIP</code> Weighted Additive Model. Measure of Inefficiency Proportions.</li> <li>• <code>WAM.RAM</code> Weighted Additive Model. Range Adjusted Measure of Inefficiency.</li> </ul>
<code>digits</code>	Integer. Decimal units for scores.
<code>DEA</code>	Logical. If TRUE, DEA scores are calculated with the programming model selected in <code>scores_model</code>
<code>na.rm</code>	Logical. If TRUE, NA rows are omitted.

### Value

Dataframe with input variables and efficiency scores through a convex EAT model.

**Examples**

```

simulated <- X2Y2.sim(N = 50, border = 0.2)
EAT_model <- EAT(data = simulated, x = c(1,2), y = c(3, 4))

efficiencyCEAT(data = simulated, x = c(1, 2), y = c(3, 4), object = EAT_model,
               scores_model = "BCC.OUT", digits = 2, DEA = TRUE, na.rm = TRUE)

```

---

efficiencyDensity      *Efficiency Scores Density Plot*

---

**Description**

Density plot for efficiency scores.

**Usage**

```
efficiencyDensity(df_scores, model = c("EAT", "FDH"))
```

**Arguments**

`df_scores`      Dataframe with efficiency scores.

`model`            String vector. Scoring models in the order shown in `df_scores` by columns. The available models are: "EAT", "FDH", "CEAT", "DEA" and "RFEAT".

**Value**

Density plot for efficiency scores.

**Examples**

```

simulated <- X2Y2.sim(N = 50, border = 0.2)
EAT_model <- EAT(data = simulated, x = c(1,2), y = c(3, 4))

scores <- efficiencyEAT(data = simulated, x = c(1, 2), y = c(3, 4), object = EAT_model,
                       scores_model = "BCC.OUT", digits = 2, FDH = TRUE, na.rm = TRUE)

efficiencyDensity(df_scores = scores[, 5:6],
                  model = c("EAT", "FDH"))

```

---

efficiencyEAT                      *Efficiency Analysis Trees Efficiency Scores*

---

### Description

This function calculates the efficiency scores for each DMU by an Efficiency Analysis Trees model.

### Usage

```
efficiencyEAT(
  data,
  x,
  y,
  object,
  scores_model,
  digits = 3,
  FDH = TRUE,
  na.rm = TRUE
)
```

### Arguments

data	Dataframe containing the DMU for which the efficiency score is calculated.
x	Vector. Column input indexes in data.
y	Vector. Column output indexes in data.
object	An EAT object.
scores_model	Mathematical programming model to calculate scores. <ul style="list-style-type: none"> <li>• BCC.OUT BCC model. Output-oriented.</li> <li>• BCC.INP BCC model. Input-oriented.</li> <li>• DDF Directional Distance Function.</li> <li>• RSL.OUT Russell model. Output-oriented.</li> <li>• RSL.INP Russell model. Input-oriented.</li> <li>• WAM.MIP Weighted Additive Model. Measure of Inefficiency Proportions.</li> <li>• WAM.RAM Weighted Additive Model. Range Adjusted Measure of Inefficiency.</li> </ul>
digits	Integer. Decimal units for scores.
FDH	Logical. If TRUE, FDH scores are also calculated with the programming model selected in scores_model.
na.rm	Logical. If TRUE, NA rows are omitted.

### Value

Dataframe introduced as argument with efficiency scores calculated through an Efficiency Analysis Trees model.

## Examples

```
simulated <- X2Y2.sim(N = 50, border = 0.2)
EAT_model <- EAT(data = simulated, x = c(1,2), y = c(3, 4))

efficiencyEAT(data = simulated, x = c(1, 2), y = c(3, 4), object = EAT_model,
              scores_model = "BCC.OUT", digits = 2, FDH = TRUE, na.rm = TRUE)
```

---

efficiencyJitter      *Efficiency Scores Jitter Plot*

---

## Description

This function returns a jitter plot from ggplot2. This graphic shows how DMUs are grouped into leaf nodes in a model built using the EAT function. Each leaf node groups DMUs with the same level of resources. The dot and the black line represent, respectively, the mean value and the standard deviation of the scores of its node. Additionally, efficient DMU labels always are displayed based on the model entered in the scores\_model argument. Finally, the user can specify an upper bound upn and a lower bound lwb in order to show, in addition, the labels whose efficiency score lies between them.

## Usage

```
efficiencyJitter(object, df_scores, scores_model, upb = NULL, lwb = NULL)
```

## Arguments

object	An EAT object.
df_scores	Dataframe with efficiency scores (from efficiencyEAT or efficiencyCEAT).
scores_model	Mathematical programming model to calculate scores. <ul style="list-style-type: none"> <li>• BCC.OUT BCC model. Output-oriented.</li> <li>• BCC.INP BCC model. Input-oriented.</li> <li>• DDF Directional Distance Function.</li> <li>• RSL.OUT Russell model. Output-oriented.</li> <li>• RSL.INP Russell model. Input-oriented.</li> <li>• WAM.MIP Weighted Additive Model. Measure of Inefficiency Proportions.</li> <li>• WAM.RAM Weighted Additive Model. Range Adjusted Measure of Inefficiency.</li> </ul>
upb	Numeric. Upper bound for labeling.
lwb	Numeric. Lower bound for labeling.



**Value**

Jitter plot with DMUs and scores.

**Examples**

```
simulated <- X2Y2.sim(N = 50, border = 0.2)
EAT_model <- EAT(data = simulated, x = c(1,2), y = c(3, 4))

EAT_scores <- efficiencyEAT(data = simulated, x = c(1, 2), y = c(3, 4), object = EAT_model,
                           scores_model = "BCC.OUT", digits = 2, na.rm = TRUE)

efficiencyJitter(object = EAT_model, df_scores = EAT_scores, scores_model = "BCC.OUT")
```

---

efficiencyRFEAT	<i>Random Forest + Efficiency Analysis Trees Efficiency Scores</i>
-----------------	--

---

**Description**

This function calculates the efficiency scores for each DMU through a Random Forest + Efficiency Analysis Trees model and the Banker Charnes and Cooper mathematical programming model with output orientation.

**Usage**

```
efficiencyRFEAT(data, x, y, object, digits = 3, FDH = TRUE, na.rm = TRUE)
```

**Arguments**

data	Dataframe for which the efficiency score is calculated.
x	Vector. Column input indexes in data.
y	Vector. Column output indexes in data.
object	A RFEAT object.
digits	Integer. Decimal units for scores.
FDH	Logical. If TRUE, FDH scores are computed.
na.rm	Logical. If TRUE, NA rows are omitted.

**Value**

Dataframe with input variables and efficiency scores through a Random Forest + Efficiency Analysis Trees model.

## Examples

```
simulated <- X2Y2.sim(N = 50, border = 0.2)
RFEAT_model <- RFEAT(data = simulated, x = c(1,2), y = c(3, 4))

efficiencyRFEAT(data = simulated, x = c(1, 2), y = c(3, 4), object = RFEAT_model,
                digits = 2, FDH = TRUE, na.rm = TRUE)
```

---

estimEAT

*Estimation of child nodes*

---

## Description

This function gets the estimation of the response variable and updates Pareto-coordinates and the observation index for both new nodes.

## Usage

```
estimEAT(data, leaves, t, xi, s, y)
```

## Arguments

data	Data to be used.
leaves	List structure with leaf nodes or pending expansion nodes.
t	Node which is being split.
xi	Variable index that produces the split.
s	Value of xi variable that produces the split.
y	Column output indexes in data.

## Value

Left and right children nodes.

---

`frontier`*Efficiency Analysis Trees Frontier Graph*

---

**Description**

This function displays a plot with the frontier estimated by Efficiency Analysis Trees in a scenario of one input and one output.

**Usage**

```
frontier(  
  object,  
  FDH = FALSE,  
  observed.data = FALSE,  
  observed.color = "black",  
  pch = 19,  
  size = 1,  
  rwn = FALSE,  
  max.overlaps = 10  
)
```

**Arguments**

<code>object</code>	An EAT object.
<code>FDH</code>	Logical. If TRUE, FDH frontier is displayed.
<code>observed.data</code>	Logical. If TRUE, observed DMUs are displayed.
<code>observed.color</code>	String. Color for observed DMUs.
<code>pch</code>	Integer. Point shape.
<code>size</code>	Integer. Point size.
<code>rwn</code>	Logical. If TRUE, rownames are displayed.
<code>max.overlaps</code>	Exclude text labels that overlap too many things.

**Value**

Plot with estimated production frontier

**Examples**

```
simulated <- Y1.sim(N = 50, nX = 1)  
  
model <- EAT(data = simulated,  
             x = 1,  
             y = 2)  
  
frontier <- frontier(object = model,
```

```

      FDH = TRUE,
      observed.data = TRUE,
      rwn = TRUE)
plot(frontier)

```

---

frontier.levels	<i>Efficiency Analysis Trees Frontier Output Levels</i>
-----------------	---

---

### Description

This function returns the frontier output levels of an Efficiency Analysis Trees model.

### Usage

```
frontier.levels(object)
```

### Arguments

object            An EAT object.

### Value

Data frame with the frontier output levels at the leaf nodes.

### Examples

```

simulated <- Y1.sim(N = 50, nX = 3)
model <- EAT(data = simulated, x = c(1, 2, 3), y = 4, numStop = 10, fold = 5)
frontier.levels(model)

```

---

generateLv	<i>Train and Test Sets Generation</i>
------------	---------------------------------------

---

### Description

This function splits the original data in two new data sets: a train set and a test set.

### Usage

```
generateLv(data, fold)
```

### Arguments

data            Data to be split into train and test subsets.  
 fold            Parts in which the original set is divided, to perform Cross-Validation.

**Value**

List structure with the train and the test set.

---

imp_var_EAT	<i>Breiman's Variable Importance</i>
-------------	--------------------------------------

---

**Description**

This function recalculates all the possible splits, with the exception of the one being used, and for each node and variable gets the best split based on their degree of importance.

**Usage**

```
imp_var_EAT(data, tree, x, y, digits)
```

**Arguments**

data	Data from EAT object.
tree	Tree from EAT object.
x	Column input indexes in data.
y	Column output indexes in data.
digits	Decimal units.

**Value**

A dataframe with the best split for each node and its variable importance.

---

imp_var_RFEAT	<i>Variable Importance through Random Forest + Efficiency Analysis Trees</i>
---------------	--

---

**Description**

Variable Importance through Random Forest + Efficiency Analysis Trees.

**Usage**

```
imp_var_RFEAT(object, digits = 2)
```

**Arguments**

object	A RFEAT object
digits	Integer. Decimal units.

**Value**

Vector of input importance scores

---

isFinalNode	<i>Is Final Node</i>
-------------	----------------------

---

**Description**

This function evaluates a node and checks if it fulfills the conditions to be a final node.

**Usage**

```
isFinalNode(obs, data, numStop)
```

**Arguments**

obs	Observation in the evaluated node.
data	Data with predictive variable.
numStop	Minimum number of observations in a node to be split.

**Value**

True if the node is a final node and false in any other case.

---

layout	<i>Layout for nodes in plotEAT</i>
--------	------------------------------------

---

**Description**

This function modifies the coordinates of the nodes in the plotEAT function to overcome overlapping.

**Usage**

```
layout(py)
```

**Arguments**

py	a party object.
----	-----------------

**Value**

Dataframe with suitable modifications of the node layout.

---

mse	<i>Mean Square Error</i>
-----	--------------------------

---

**Description**

This function calculates the Mean Square Error between the predicted value and the observations in a given node.

**Usage**

```
mse(data, t, y)
```

**Arguments**

data	Data to be used.
t	A given node.
y	Column output indexes in data.

**Value**

Mean Square Error at a node.

---

mtry_inputSelection	<i>Random Selection of Variables</i>
---------------------	--------------------------------------

---

**Description**

This function randomly selects the variables that are evaluated to divide a node and removes those that do not present variability.

**Usage**

```
mtry_inputSelection(data, x, t, mtry)
```

**Arguments**

data	Dataframe containing the training set.
x	Vector. Column input indexes in data.
t	Node which is being split.
mtry	Integer. Number of inputs selected for a node to be split.

**Value**

Index of the variables by which the node is divided.

---

M_Breiman	<i>Breiman Importance</i>
-----------	---------------------------

---

**Description**

This function evaluates the importance of each predictor by the notion of surrogate splits.

**Usage**

```
M_Breiman(object, digits)
```

**Arguments**

object	An EAT object.
digits	Decimal units.

**Value**

Dataframe with one column and the importance of each variable in rows.

---

PISAindex	<i>PISA score and social index by country</i>
-----------	---

---

**Description**

A dataset containing the PISA score in mathematics, reading and science and 13 variables related to the social index by country for 2018.

**Usage**

```
PISAindex
```

**Format**

A data frame with 72 rows and 18 variables:

<b>Country</b>	Country name
<b>Continent</b>	Country continent
<b>S_PISA</b>	PISA score in Science
<b>R_PISA</b>	PISA score in Reading
<b>M_PISA</b>	PISA score in Mathematics
<b>NBMC</b>	Nutritional and Basic Medical Care
<b>WS</b>	Water and Sanitation
<b>S</b>	Shelter



**PS** Personal Safety  
**ABK** Access to Basic Knowledge  
**AIC** Access to Information and Communication  
**HW** Health and Wellness  
**EQ** Environmental Quality  
**PR** Personal Rights  
**PFC** Personal Freedom and Choice  
**I** Inclusiveness  
**AAE** Access to Advanced Education  
**GDP\_PPP** Gross Domestic Product per capita adjusted by purchasing power parity

**Source**

<https://www.socialprogress.org/>

[https://www.oecd.org/pisa/Combined\\_Executive\\_Summaries\\_PISA\\_2018.pdf](https://www.oecd.org/pisa/Combined_Executive_Summaries_PISA_2018.pdf)

---

plotEAT

*Efficiency Analysis Trees Plot*

---

**Description**

Plot a tree-structure for an Efficiency Analysis Trees model.

**Usage**

```
plotEAT(object)
```

**Arguments**

object            An EAT object.

**Value**

Plot object with the following elements for each node:

- id: node index.
- R: error at the node.
- n(t): number of observations at the node.
- an input name: splitting variable.
- y: output prediction.

## Examples

```
simulated <- X2Y2.sim(N = 50, border = 0.2)
EAT_model <- EAT(data = simulated, x = c(1,2), y = c(3, 4))

plotEAT(EAT_model)
```

---

plotRFEAT

*Random Forest + Efficiency Analysis Trees Plot*

---

## Description

Plot a graph with the Out-of-Bag error for the training dataset and a forest consisting of k trees.

## Usage

```
plotRFEAT(object)
```

## Arguments

object            A RFEAT object.

## Value

Line plot with the OOB error and the number of trees in the forest.

## Examples

```
simulated <- Y1.sim(N = 150, nX = 6)
RFmodel <- RFEAT(data = simulated, x = 1:6, y = 7, numStop = 10,
                 m = 50, s_mtry = "BRM", na.rm = TRUE)
plotRFEAT(RFmodel)
```

---

posIdNode	<i>Position of the node</i>
-----------	-----------------------------

---

**Description**

This function finds the node where a register is located.

**Usage**

```
posIdNode(tree, idNode)
```

**Arguments**

tree	A list containing EAT nodes.
idNode	Id of a specific node.

**Value**

Position of the node or -1 if it is not found.

---

predictEAT	<i>Model Prediction for Efficiency Analysis Trees.</i>
------------	--

---

**Description**

This function predicts the expected output by an EAT object.

**Usage**

```
predictEAT(object, newdata, x)
```

**Arguments**

object	An EAT object.
newdata	Dataframe. Set of input variables to predict on.
x	Inputs index.

**Value**

Data frame with the original data and the predicted values.

**Examples**

```

simulated <- X2Y2.sim(N = 50, border = 0.2)
EAT_model <- EAT(data = simulated, x = c(1,2), y = c(3, 4))

predictEAT(object = EAT_model, newdata = simulated, x = c(1,2))

```

---

<code>predictFDH</code>	<i>Model prediction for Free Disposal Hull</i>
-------------------------	--

---

**Description**

This function predicts the expected output by a Free Disposal Hull model.

**Usage**

```
predictFDH(data, x, y)
```

**Arguments**

<code>data</code>	Dataframe or matrix containing the variables in the model.
<code>x</code>	Vector. Column input indexes in data.
<code>y</code>	Vector. Column output indexes in data.

**Value**

Data frame with the original data and the predicted values through a Free Disposal Hull model.

---

<code>predictor</code>	<i>Efficiency Analysis Trees Predictor</i>
------------------------	--

---

**Description**

This function predicts the expected value based on a set of inputs.

**Usage**

```
predictor(tree, register)
```

**Arguments**

<code>tree</code>	List structure with the tree nodes.
<code>register</code>	Set of independent values.

**Value**

The expected value of the dependent variable based on the given register.

---

predictRFEAT	<i>Model prediction for Random Forest + Efficiency Analysis Trees model.</i>
--------------	--

---

**Description**

This function predicts the expected output by a RFEAT object.

**Usage**

```
predictRFEAT(object, newdata, x)
```

**Arguments**

object	A RFEAT object.
newdata	Dataframe. Set of input variables to predict on.
x	Inputs index

**Value**

Data frame with the original data and the predicted values.

**Examples**

```
simulated <- X2Y2.sim(N = 50, border = 0.2)
RFEAT_model <- RFEAT(data = simulated, x = c(1,2), y = c(3, 4))

predictRFEAT(object = RFEAT_model, newdata = simulated, x = c(1,2))
```

---

preProcess

*Data preprocessing*

---

### Description

This function displays error messages in relation with data such as presence of NA values, types of data not allowed or arguments entered incorrectly. Also, it prepares the data in the required format.

### Usage

```
preProcess(  
  data,  
  x,  
  y,  
  numStop = 5,  
  fold = 5,  
  max.depth = NULL,  
  max.leaves = NULL,  
  na.rm = TRUE  
)
```

### Arguments

data	Dataframe or matrix containing the variables in the model.
x	Vector. Column input indexes in data.
y	Vector. Column output indexes in data.
numStop	Integer. Minimum number of observations in a node for a split to be attempted.
fold	Integer. Set of number of folds in which the dataset to apply cross-validation during the pruning is divided.
max.depth	Integer. Depth of the tree.
max.leaves	Integer. Maximum number of leaf nodes.
na.rm	Logical. If TRUE, NA rows are omitted.

### Value

List containing rownames and data processed in the [X, Y] format with only categories allowed.

---

RandomEAT	<i>Individual EAT for Random Forest</i>
-----------	---

---

**Description**

This function builds an individual tree for Random Forest

**Usage**

```
RandomEAT(data, x, y, numStop, s_mtry)
```

**Arguments**

data	Dataframe containing the training set.
x	Vector. Column input indexes in data.
y	Vector. Column output indexes in data.
numStop	Integer. Minimum number of observations in a node for a split to be attempted.
s_mtry	Number of variables randomly sampled as candidates at each split. The available options are: "BRM", "DEA1", "DEA2", "DEA3", "DEA4" or any integer.

**Value**

List of m trees in forest and the error that will be used in the ranking of the importance of the variables.

---

rankingEAT	<i>Ranking of Variables by Efficiency Analysis Trees</i>
------------	--

---

**Description**

This function calculates variable importance through an Efficiency Analysis Trees model.

**Usage**

```
rankingEAT(object, barplot = TRUE, threshold = 70, digits = 2)
```

**Arguments**

object	An EAT object.
barplot	Logical. If TRUE, a barplot with the importance scores is displayed.
threshold	Numeric. Importance score value in which a line is graphed.
digits	Integer. Decimal units.

**Value**

Dataframe with the importance scores. If `barplot = TRUE`, it is returned a list containing the scores and the barplot.

**Examples**

```
simulated <- X2Y2.sim(N = 50, border = 0.2)
EAT_model <- EAT(data = simulated, x = c(1,2), y = c(3, 4))

rankingEAT(object = EAT_model,
           barplot = TRUE,
           threshold = 70,
           digits = 2)
```

---

rankingRFEAT

*Ranking of variables by Random Forest + Efficiency Analysis Trees.*

---

**Description**

This function calculates variable importance through a Random Forest + Efficiency Analysis Trees model.

**Usage**

```
rankingRFEAT(object, barplot = TRUE, digits = 2)
```

**Arguments**

<code>object</code>	A RFEAT object.
<code>barplot</code>	Logical. If TRUE, a barplot with importance scores is displayed.
<code>digits</code>	Integer. Decimal units.

**Value**

Dataframe with the importance scores. If `barplot = TRUE`, it is returned a list containing the scores and the barplot.

**Examples**

```
simulated <- X2Y2.sim(N = 50, border = 0.2)
RFEAT_model <- RFEAT(data = simulated, x = c(1,2), y = c(3, 4))

rankingRFEAT(object = RFEAT_model,
             barplot = TRUE,
```



```
digits = 2)
```

---

RBranch

*Branch Pruning*


---

### Description

This function computes the error of a branch as the sum of the errors of its child nodes.

### Usage

```
RBranch(t, tree)
```

### Arguments

t	List. A given EAT node.
tree	A list containing EAT nodes.

### Value

List containing the sum of the errors of the child nodes of the pruned node and the total number of leaf nodes that come from it.

---

RCV

*RCV*


---

### Description

RCV

### Usage

```
RCV(N, Lv, y, alphaIprim, fold, TAiv)
```

### Arguments

N	Number of rows in data.
Lv	Test set.
y	Column output indexes in data.
alphaIprim	Alpha obtained as the square root of the product of two consecutive alpha values in tree_alpha list. It is used to find the best pruning tree.
fold	Parts in which the original data is divided into to perform Cross-Validation.
TAiv	List with each possible pruning for the deep tree generated with the train set and its associated alpha values.

**Value**

Set of best pruning and the associated error calculated with test sets.

---

RFEAT

*Random Forest EAT*


---

**Description**

This function builds  $m$  individual Efficiency Analysis Trees in a forest structure.

**Usage**

```
RFEAT(data, x, y, numStop = 5, m = 50, s_mtry = "BRM", na.rm = TRUE)
```

**Arguments**

<code>data</code>	Dataframe or matrix containing the variables in the model.
<code>x</code>	Vector. Column input indexes in data.
<code>y</code>	Vector. Column output indexes in data.
<code>numStop</code>	Integer. Minimum number of observations in a node for a split to be attempted.
<code>m</code>	Integer. Number of trees to be built.
<code>s_mtry</code>	Number of variables randomly sampled as candidates at each split. The available options are: <ul style="list-style-type: none"> <li>• "BRM": <math>in / 3</math></li> <li>• "DEA1": <math>(t.obs / 2) - out</math></li> <li>• "DEA2": <math>(t.obs / 3) - out</math></li> <li>• "DEA3": <math>t.obs - 2 * out</math></li> <li>• "DEA4": <math>\min(t.obs / out, (t.obs / 3) - out)</math></li> <li>• integer</li> </ul>
<code>na.rm</code>	Logical. If TRUE, NA rows are omitted.

**Value**

A RFEAT object containing:

- `data`
  - `df`: data frame containing the variables in the model.
  - `x`: input indexes in data.
  - `y`: output indexes in data.
  - `input_names`: input variable names.
  - `output_names`: output variable names.
  - `row_names`: rownames in data.
- `control`

- numStop: numStop hyperparameter value.
- m: m hyperparameter value.
- s\_mtry: s\_mtry hyperparameter value.
- na.rm: na.rm hyperparameter value.
- forest: list structure containing the individual EAT.
- error: Out-of-Bag error at the forest.
- OOB: list containing Out-of-Bag set for each tree.

## Examples

```
simulated <- X2Y2.sim(N = 50, border = 0.1)

RFmodel <- RFEAT(data = simulated, x = c(1,2), y = c(3, 4), numStop = 5,
                 m = 50, s_mtry = "BRM", na.rm = TRUE)
```

---

RFEAT\_object

*RFEAT object*


---

## Description

This function saves information about the RFEAT model

## Usage

```
RFEAT_object(
  data,
  x,
  y,
  register_names,
  numStop,
  m,
  s_mtry,
  na.rm,
  forest,
  error,
  OOB
)
```

## Arguments

data	Dataframe or matrix containing the variables in the model.
x	Vector. Column input indexes in data.
y	Vector. Column output indexes in data.

register_names	String vector. Data rownames.
numStop	Integer. Minimum number of observations in a node for a split to be attempted.
m	Integer. Number of trees to be built.
s_mtry	Select number of inputs in each split. <ul style="list-style-type: none"> <li>• "Breiman": in / 3</li> <li>• "DEA1": (t.obs / 2) -out</li> <li>• "DEA2": (t.obs / 3) -out</li> <li>• "DEA3": t.obs - 2 * out</li> <li>• "DEA4": min(t.obs / out, (t.obs / 3) -out)</li> </ul>
na.rm	Logical. If TRUE, NA rows are omitted.
forest	A list containing the individual EAT trees.
error	Error in forest.
OOB	List containing the observations with which each tree has been trained.

**Value**

A RFEAT object

---

RF_predictor	<i>Random Forest + Efficiency Analysis Trees Predictor</i>
--------------	--

---

**Description**

This function predicts the expected value based on a set of inputs.

**Usage**

```
RF_predictor(forest, xn)
```

**Arguments**

forest	List containing individual Efficiency Analysis Trees.
xn	Row indexes in data.

**Value**

Vector of predictions.

---

scores	<i>Pruning Scores</i>
--------	-----------------------

---

**Description**

This function calculates the score for each pruning of `tree_alpha_list`.

**Usage**

```
scores(N, Lv_notLv, x, y, fold, numStop, Tk, tree_alpha_list)
```

**Arguments**

N	Number of rows in data.
Lv_notLv	List with train and test sets.
x	Column input indexes in data.
y	Column output indexes in data.
fold	Parts in which the original data set is divided to perform Cross-Validation.
numStop	Minimum number of observations on a node to be split.
Tk	Best pruned tree.
tree_alpha_list	List with all the possible pruning and its associated alpha.

**Value**

List with the best pruning for each fold, the pruning with a lower score and `tree_alpha_list` with scores updated.

---

selectTk	<i>Select Tk</i>
----------	------------------

---

**Description**

This function tries to find a new pruned tree with a shorter length and a score in the range generated for SE.

**Usage**

```
selectTk(Tk, tree_alpha_list, SE)
```

**Arguments**

Tk	Best pruned tree score.
tree_alpha_list	List with all the possible pruning and its associated alpha and scores.
SE	Value to get a range where new prunings is found.

**Value**

The same best tree or a new suitable one.

---

select_mtry	<i>Select Possible Inputs in Split.</i>
-------------	---

---

**Description**

This function selects the number of inputs for a split in Random Forest.

**Usage**

```
select_mtry(s_mtry, t, nX, nY)
```

**Arguments**

s_mtry	Select number of inputs. It could be: "BRM", "DEA1", "DEA2", "DEA3" or "DEA4" or any integer.
t	Node which is being split.
nX	Number of inputs in data.
nY	Number of outputs in data.

**Value**

Number of inputs selected according to the specified rule.

---

SERules

*SERules*


---

**Description**

Based on Validation tests over BestTivs, a new range of scores is obtained to find new pruned trees.

**Usage**

```
SERules(N, Lv, y, fold, Tk_score, BestTivs)
```

**Arguments**

N	Number of rows in data.
Lv	Test set.
y	Column output indexes in data.
fold	Parts in which the original data set is divided to perform Cross-Validation.
Tk_score	Best pruned tree score.
BestTivs	List of best pruned trees for each training set.

**Value**

Value to get a range where new pruning is found.

---

size

*Efficiency Analysis Trees Size*


---

**Description**

This function returns the number of leaf nodes at the tree.

**Usage**

```
size(object)
```

**Arguments**

object	An EAT object.
--------	----------------

**Value**

Print with the number of leaf nodes at the tree.

## Examples

```
simulated <- Y1.sim(N = 50, nX = 3)
model <- EAT(data = simulated, x = c(1, 2, 3), y = 4, numStop = 10, fold = 5)
size(model)
```

---

split	<i>Split node</i>
-------	-------------------

---

## Description

This function gets the variable and split value to be used in estimEAT, selects the best split and updates VarInfo, node indexes and leaves list.

## Usage

```
split(data, tree, leaves, t, x, y, numStop)
```

## Arguments

data	Data to be used.
tree	List structure with the tree nodes.
leaves	List with leaf nodes or pending expansion nodes.
t	Node which is being split.
x	Column input indexes in data.
y	Column output indexes in data.
numStop	Minimum number of observations in a node to be split.

## Value

Leaves and tree lists updated with the new child nodes.



---

split_forest	<i>Split Node in Random Forest EAT</i>
--------------	--

---

**Description**

This function gets the variable and split value to be used in estimEAT, selects the best split, node indexes and leaf list.

**Usage**

```
split_forest(data, tree, leaves, t, x, y, numStop, arrayK)
```

**Arguments**

data	Data to be used.
tree	List structure with the tree nodes.
leaves	List with leaf nodes or pending expansion nodes.
t	Node which is being split.
x	Column input indexes in data.
y	Column output indexes in data.
numStop	Minimum number of observations on a node to be split.
arrayK	Column input indexes in data selected by s_mtry.

**Value**

Leaves and tree lists updated with the new child nodes.

---

treesForRCV	<i>Trees for RCV</i>
-------------	----------------------

---

**Description**

This function generates a deep EAT and all pruning for each train set.

**Usage**

```
treesForRCV(notLv, x, y, fold, numStop)
```

**Arguments**

notLv	Train set.
x	Column input indexes in data.
y	Column output indexes in data.
fold	Parts in which the original set is divided to perform Cross-Validation.
numStop	Minimum number of observations in a node to be split.

**Value**

List with each possible pruning for the deep tree generated with train set and its associated alpha values.

---

X2Y2.sim

*2 Inputs & 2 Outputs Scenario*


---

**Description**

This function is used to simulate the data in a scenario with 2 inputs and 2 outputs.

**Usage**

```
X2Y2.sim(N, border, noise = NULL)
```

**Arguments**

N	Sample size.
border	Percentage of DMUs in the frontier.
noise	Random noise.

**Value**

Data frame with the values of a scenario with 2 inputs and 2 outputs.

---

Y1.sim

*Single Output Data Generation*


---

**Description**

This function is used to simulate the data in a single output scenario.

**Usage**

```
Y1.sim(N, nX)
```

**Arguments**

N	Integer. Sample size.
nX	Integer. Number of inputs. 1, 3, 6, 9, 12 and 15 are acceptable.

**Value**

Data frame with the values of a scenario with nX inputs and 1 output.

# Index

## \* datasets

- PISAindex, 32
- alpha, 3
- bagging, 4
- barplot\_importance, 4
- bestEAT, 5
- bestRFEAT, 6
- CEAT\_BCC\_in, 7
- CEAT\_BCC\_out, 8
- CEAT\_DDF, 9
- CEAT\_RSL\_in, 9
- CEAT\_RSL\_out, 10
- CEAT\_WAM, 11
- checkEAT, 11
- comparePareto, 12
- deepEAT, 12
- descrEAT, 13
- EAT, 13
- EAT\_BCC\_in, 16
- EAT\_BCC\_out, 16
- EAT\_DDF, 17
- EAT\_object, 18
- EAT\_RSL\_in, 19
- EAT\_RSL\_out, 19
- EAT\_WAM, 20
- efficiencyCEAT, 21
- efficiencyDensity, 22
- efficiencyEAT, 23
- efficiencyJitter, 24
- efficiencyRFEAT, 25
- estimEAT, 26
- frontier, 27
- frontier.levels, 28
- generateLv, 28
- imp\_var\_EAT, 29
- imp\_var\_RFEAT, 29
- isFinalNode, 30
- layout, 30
- M\_Breiman, 32
- mse, 31
- mtry\_inputSelection, 31
- PISAindex, 32
- plotEAT, 33
- plotRFEAT, 34
- posIdNode, 35
- predictEAT, 35
- predictFDH, 36
- predictor, 36
- predictRFEAT, 37
- preProcess, 38
- RandomEAT, 39
- rankingEAT, 39
- rankingRFEAT, 40
- RBranch, 41
- RCV, 41
- RF\_predictor, 44
- RFEAT, 42
- RFEAT\_object, 43
- scores, 45
- select\_mtry, 46
- selectTk, 45
- SERules, 47
- size, 47
- split, 48
- split\_forest, 49
- treesForRCV, 49
- X2Y2.sim, 50
- Y1.sim, 50