Title       Efficiency Analysis Trees
Version     0.1.4
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
License     GPL-3
Encoding    UTF-8
LazyData    true
RoxygenNote 7.2.1
Imports     dplyr, conflicted, stats, ggplot2, ggparty, partykit, ggrepel,
            Rdpack, lpSolveAPI, utils, reshape2
Suggests    rmarkdown, testthat, knitr, kableExtra, 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>),
             Víctor España [aut] (<https://orcid.org/0000-0002-1807-6180>),
             Juan Aparicio [aut] (<https://orcid.org/0000-0002-0867-0004>),
             Xavier Barber [aut] (<https://orcid.org/0000-0003-3079-5855>)
Maintainer   Miriam Esteve <mestevecampello@gmail.com>
Repository   CRAN
Date/Publication 2023-01-10 09:20:16 UTC
**R topics documented:**

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>alpha</td>
<td>3</td>
</tr>
<tr>
<td>bagging</td>
<td>4</td>
</tr>
<tr>
<td>barplot_importance</td>
<td>4</td>
</tr>
<tr>
<td>bestEAT</td>
<td>5</td>
</tr>
<tr>
<td>bestRFEAT</td>
<td>6</td>
</tr>
<tr>
<td>CEAT_BCC_in</td>
<td>7</td>
</tr>
<tr>
<td>CEAT_BCC_out</td>
<td>8</td>
</tr>
<tr>
<td>CEAT_DDF</td>
<td>9</td>
</tr>
<tr>
<td>CEAT_RSL_in</td>
<td>9</td>
</tr>
<tr>
<td>CEAT_RSL_out</td>
<td>10</td>
</tr>
<tr>
<td>CEAT_WAM</td>
<td>11</td>
</tr>
<tr>
<td>checkEAT</td>
<td>11</td>
</tr>
<tr>
<td>comparePareto</td>
<td>12</td>
</tr>
<tr>
<td>deepEAT</td>
<td>12</td>
</tr>
<tr>
<td>EAT</td>
<td>13</td>
</tr>
<tr>
<td>EAT_BCC_in</td>
<td>15</td>
</tr>
<tr>
<td>EAT_BCC_out</td>
<td>16</td>
</tr>
<tr>
<td>EAT_DDF</td>
<td>16</td>
</tr>
<tr>
<td>EAT_frontier_levels</td>
<td>17</td>
</tr>
<tr>
<td>EAT_leaf_stats</td>
<td>18</td>
</tr>
<tr>
<td>EAT_object</td>
<td>19</td>
</tr>
<tr>
<td>EAT_RSL_in</td>
<td>20</td>
</tr>
<tr>
<td>EAT_RSL_out</td>
<td>20</td>
</tr>
<tr>
<td>EAT_size</td>
<td>21</td>
</tr>
<tr>
<td>EAT_WAM</td>
<td>22</td>
</tr>
<tr>
<td>efficiencyCEAT</td>
<td>22</td>
</tr>
<tr>
<td>efficiencyDensity</td>
<td>24</td>
</tr>
<tr>
<td>efficiencyEAT</td>
<td>25</td>
</tr>
<tr>
<td>efficiencyJitter</td>
<td>26</td>
</tr>
<tr>
<td>efficiencyRFEAT</td>
<td>27</td>
</tr>
<tr>
<td>estimEAT</td>
<td>28</td>
</tr>
<tr>
<td>frontier</td>
<td>29</td>
</tr>
<tr>
<td>generateLv</td>
<td>30</td>
</tr>
<tr>
<td>imp_var_EAT</td>
<td>30</td>
</tr>
<tr>
<td>imp_var_RFEAT</td>
<td>31</td>
</tr>
<tr>
<td>isFinalNode</td>
<td>31</td>
</tr>
<tr>
<td>layout</td>
<td>32</td>
</tr>
<tr>
<td>mse</td>
<td>32</td>
</tr>
<tr>
<td>mtry_inputSelection</td>
<td>33</td>
</tr>
<tr>
<td>M_Breiman</td>
<td>33</td>
</tr>
<tr>
<td>PIS Aindex</td>
<td>34</td>
</tr>
<tr>
<td>plotEAT</td>
<td>35</td>
</tr>
<tr>
<td>plotRFEAT</td>
<td>35</td>
</tr>
<tr>
<td>posIdNode</td>
<td>36</td>
</tr>
<tr>
<td>predict.EAT</td>
<td>37</td>
</tr>
<tr>
<td>predict.RFEAT</td>
<td>37</td>
</tr>
</tbody>
</table>
**Description**

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

**Usage**

```r
alpha(tree)
```

**Arguments**

- `tree`: A list containing the EAT nodes.

**Value**

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

---

**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.
**bestEAT**

*Tuning an Efficiency Analysis Trees model*

**Description**

This function computes the root mean squared error (RMSE) for a set of Efficiency Analysis Trees models built with a grid of given hyperparameters.

**Usage**

```r
bestEAT(
  training,  # Training data.frame or matrix containing the variables for model construction.
  test,      # Test data.frame or matrix containing the variables for model assessment.
  x,         # Column input indexes in training.
  y,         # Column output indexes in training.
  numStop = 5,
  fold = 5,
  max.depth = NULL,
  max.leaves = NULL,
  na.rm = TRUE
)
```

**Arguments**

- **training**: Training data.frame or matrix containing the variables for model construction.
- **test**: Test data.frame or matrix containing the variables for model assessment.
- **x**: Column input indexes in training.
- **y**: Column output indexes in training.
- **numStop**: Minimum number of observations in a node for a split to be attempted.
- **fold**: Folds in which the dataset to apply cross-validation during the pruning is divided.
- **max.depth**: Maximum depth of the tree.
- **max.leaves**: Maximum number of leaf nodes.
- **na.rm**: logical. If TRUE, NA rows are omitted.

**Value**

A data.frame with the sets of hyperparameters and the root mean squared error (RMSE) associated for each model.
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 computes the root mean squared error (RMSE) for a set of Random Forest + Efficiency Analysis Trees models built with a grid of given hyperparameters.

**Usage**

```
bestRFEAT(
  training,  # Training data.frame or matrix containing the variables for model construction.
  test,      # Test data.frame or matrix containing the variables for model assessment.
  x,         # Column input indexes in training.
  y,         # Column output indexes in training.
  numStop = 5, # Minimum number of observations in a node for a split to be attempted.
  m = 50,    # Number of trees.
  s_mtry = c("5", "BRM"), # Number of variables to consider at each split.
  na.rm = TRUE # Ignore missing values.
)
```

**Arguments**

- `training`: Training data.frame or matrix containing the variables for model construction.
- `test`: Test data.frame or matrix containing the variables for model assessment.
- `x`: Column input indexes in training.
- `y`: Column output indexes in training.
- `numStop`: Minimum number of observations in a node for a split to be attempted.
CEAT_BCC_in

- `m`: Number of trees to be built.
- `s_mtry`: Character. Number of inputs to be selected in each split. See "s".
- `na.rm`: Logical. If TRUE, NA rows are omitted.

**Value**

A data.frame with the sets of hyperparameters and the root mean squared error (RMSE) associated for each model.

**Examples**

```r
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"))
```

---

**Description**

Banker, Charnes and Cooper programming model with input orientation for a Convexified Efficiency Analysis Trees model.

**Usage**

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

**Arguments**

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

Value

A numerical vector with scores.

---

**CEAT_BCC_out**

Banker, Charnes and Cooper programming model with output orientation for a Convexified Efficiency Analysis Trees model

---

**Description**

Banker, Charnes and Cooper programming model with output orientation for a Convexified Efficiency Analysis Trees model.

**Usage**

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

**Arguments**

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

**Value**

A numerical vector with efficiency scores.
**CEAT_DDF**

*Directional Distance Function mathematical programming model for a Convexified Efficiency Analysis Trees model*

**Description**

Directional Distance Function for a Convexified Efficiency Analysis Trees model.

**Usage**

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

**Arguments**

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

**Value**

A numerical vector with scores.

---

**CEAT_RSL_in**

*Russell Model with input orientation for a Convexified Efficiency Analysis Trees model*

**Description**

Russell Model with input orientation for a Convexified Efficiency Analysis Trees model.

**Usage**

```r
CEAT_RSL_in(j, scores, x_k, y_k, atreeTk, ytreeTk, nX, nY, N_leaves)
```
Arguments

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

Value

A numerical vector with scores.

CEAT_RSL_out

Russell Model with output orientation for a Convexified Efficiency Analysis Trees model

Description

Russell Model with output orientation for a Convexified Efficiency Analysis Trees model.

Usage

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

Arguments

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

Value

A numerical vector with scores.
**CEAT_WAM**

*Weighted Additive Model for a Convexified Efficiency Analysis Trees model*

---

**Description**

Weighted Additive Model for a Convexified Efficiency Analysis Trees model.

**Usage**

```r
CEAT_WAM(j, scores, x_k, y_k, atreeTk, ytreeTk, nX, nY, N_leaves, weights)
```

**Arguments**

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

**Value**

A numerical vector with scores.

---

**checkEAT**

*Check Efficiency Analysis Trees.*

---

**Description**

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

**Usage**

```r
checkEAT(tree)
```

**Arguments**

- `tree`: A list containing the EAT nodes.
Value
Message indicating if the tree is acceptable or warning in case of breaking any Pareto-dominance relationship.

| comparePareto | Pareto-dominance relationships |

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 | Deep Efficiency Analysis Trees |

Description
This function creates a deep Efficiency Analysis Tree and a set of possible prunings by the weakest-link pruning procedure.

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

Arguments
data data.frame or matrix containing the variables in the model.
x Column input indexes in data.
y Column output indexes in data.
umStop Minimum number of observations in a node for a split to be attempted.
max.depth Maximum depth of the tree.
max.leaves Maximum number of leaf nodes.
**Value**

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

---

**EAT**

Efficiency Analysis Trees

**Description**

This function estimates a stepped production frontier through regression trees.

**Usage**

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

**Arguments**

- `data` data.frame or matrix containing the variables in the model.
- `x` Column input indexes in data.
- `y` Column output indexes in data.
- `numStop` Minimum number of observations in a node for a split to be attempted.
- `fold` Set of number of folds in which the dataset to apply cross-validation during the pruning is divided.
- `max.depth` Depth of the tree.
- `max.leaves` Maximum number of leaf nodes.
- `na.rm` 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


EAT_BCC_in

Examples

```r
# ==============================================================
# 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)
```

---

### Description

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

### Usage

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

### Arguments

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

### Value

A numerical vector with efficiency scores.
EAT_BCC_out  Banker, Charnes and Cooper Programming Model with Output Orientation for an Efficiency Analysis Trees model

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

Usage
EAT_BCC_out(j, scores, x_k, y_k, atreeTk, ytreeTk, nX, nY, N_leaves)

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

Value
A numerical vector with efficiency scores.

EAT_DDF  Directional Distance Function Programming Model for an Efficiency Analysis Trees model

Description
Directional Distance Function for an Efficiency Analysis Trees model.

Usage
EAT_DDF(j, scores, x_k, y_k, atreeTk, ytreeTk, nX, nY, N_leaves)
Arguments

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

Value

A numerical vector with efficiency scores.

---

**EAT_frontier_levels**

*Output Levels in an Efficiency Analysis Trees model*

Description

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

Usage

```
EAT_frontier_levels(object)
```

Arguments

- object: An EAT object.

Value

A data.frame with the frontier output levels at the leaf nodes of the Efficiency Analysis Trees model introduced.

Examples

```
simulated <- Y1.sim(N = 50, nX = 3)
EAT_model <- EAT(data = simulated, x = c(1, 2, 3), y = 4, numStop = 10, fold = 5)
EAT_frontier_levels(EAT_model)
```
Descriptive Summary Statistics Table for the Leaf Nodes of an Efficiency Analysis Trees model

Description

This function returns a descriptive summary statistics table for each output variable calculated from the leaf nodes observations of an Efficiency Analysis Trees model. Specifically, it computes the number of observations, the proportion of observations, the mean, the variance, the standard deviation, the minimum, the first quartile, the median, the third quartile, the maximum and the root mean squared error.

Usage

EAT_leaf_stats(object)

Arguments

object An EAT object.

Value

A list or a data.frame (for 1 output scenario) with the following summary statistics:

- N: number of observations.
- Proportion: proportion of observations.
- mean: mean.
- var: variance.
- sd: standard deviation.
- min: minimum.
- Q1: first quartile.
- median: median.
- Q3: third quartile.
- max: maximum.
- RMSE: root mean squared error.

Examples

simulated <- Y1.sim(N = 50, nX = 3)
EAT_model <- EAT(data = simulated, x = c(1, 2, 3), y = 4, numStop = 10, fold = 5)
EAT_leaf_stats(EAT_model)
**EAT_object**

Create a EAT object

---

**Description**

This function saves information about the Efficiency Analysis Trees model.

**Usage**

\[
\text{EAT\_object}(\text{data}, \text{x}, \text{y}, \text{rownames}, \text{numStop}, \text{fold}, \text{max\_depth}, \text{max\_leaves}, \text{na\_rm}, \text{tree})
\]

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>data.frame or matrix containing the variables in the model.</td>
</tr>
<tr>
<td>x</td>
<td>Column input indexes in data.</td>
</tr>
<tr>
<td>y</td>
<td>Column output indexes in data.</td>
</tr>
<tr>
<td>rownames</td>
<td>string. Data rownames.</td>
</tr>
<tr>
<td>numStop</td>
<td>Minimum number of observations in a node for a split to be attempted.</td>
</tr>
<tr>
<td>fold</td>
<td>Set of number of folds in which the dataset to apply cross-validation during the pruning is divided.</td>
</tr>
<tr>
<td>max.depth</td>
<td>Maximum number of leaf nodes.</td>
</tr>
<tr>
<td>max.leaves</td>
<td>Depth of the tree.</td>
</tr>
<tr>
<td>na.rm</td>
<td>logical. If TRUE, NA rows are omitted. If FALSE, an error occurs in case of NA rows.</td>
</tr>
<tr>
<td>tree</td>
<td>list containing the nodes of the Efficiency Analysis Trees pruned model.</td>
</tr>
</tbody>
</table>

**Value**

An EAT object.
**EAT_RSL_in**  
*Russell Model with Input Orientation for an Efficiency Analysis Trees model*

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

**Usage**

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

**Arguments**

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

**Value**
A numerical vector with efficiency scores.

---

**EAT_RSL_out**  
*Russell Model with Output Orientation for an Efficiency Analysis Trees model*

**Description**
Russell Model with output orientation for an Efficiency Analysis Trees model.

**Usage**

```r
EAT_RSL_out(j, scores, x_k, y_k, atreeTk, ytreeTk, nX, nY, N_leaves)
```
**Arguments**

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

**Value**

A numerical vector with efficiency scores.

**Description**

This function returns the number of leaf nodes for an Efficiency Analysis Trees model.

**Usage**

```r
EAT_size(object)
```

**Arguments**

- **object** An EAT object.

**Value**

Number of leaf nodes of the Efficiency Analysis Trees model introduced.

**Examples**

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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>j</td>
<td>Number of DMUs.</td>
</tr>
<tr>
<td>scores</td>
<td>matrix. Empty matrix for scores.</td>
</tr>
<tr>
<td>x_k</td>
<td>data.frame. Set of input variables.</td>
</tr>
<tr>
<td>y_k</td>
<td>data.frame Set of output variables.</td>
</tr>
<tr>
<td>atreeTk</td>
<td>matrix Set of &quot;a&quot; Pareto-coordinates.</td>
</tr>
<tr>
<td>ytreeTk</td>
<td>matrix Set of predictions.</td>
</tr>
<tr>
<td>nX</td>
<td>Number of inputs.</td>
</tr>
<tr>
<td>nY</td>
<td>Number of outputs.</td>
</tr>
<tr>
<td>N_leaves</td>
<td>Number of leaf nodes.</td>
</tr>
<tr>
<td>weights</td>
<td>Character. &quot;MIP&quot; for Measure of Inefficiency Proportion or &quot;RAM&quot; for Range</td>
</tr>
<tr>
<td></td>
<td>Adjusted Measure of Inefficiency.</td>
</tr>
</tbody>
</table>

Value

A numerical vector with efficiency scores.

efficiencyCEAT

Efficiency Scores computed through a Convexified Efficiency Analysis Trees model.

Description

This function computes the efficiency scores for each DMU through a Convexified Efficiency Analysis Trees model.
efficiencyCEAT

Usage

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

Arguments

- `data` data.frame or matrix containing the variables in the model.
- `x` Column input indexes in data.
- `y` Column output indexes in data.
- `object` An EAT object.
- `scores_model` Mathematical programming model to calculate scores.
  - `DDF` Directional Distance Function. Efficiency level at 0.
  - `WAM.MIP` Weighted Additive Model. Measure of Inefficiency Proportions. Efficiency level at 0.
  - `WAM.RAM` Weighted Additive Model. Range Adjusted Measure of Inefficiency. Efficiency level at 0.
- `digits` Decimal units for scores.
- `DEA` logical. If TRUE, the DEA scores are also calculated with the programming model selected in `scores_model`.
- `print.table` logical. If TRUE, a summary descriptive table of the efficiency scores is displayed.
- `na.rm` logical. If TRUE, NA rows are omitted.

Value

A data.frame with the efficiency scores computed through a Convexified Efficiency Analysis Trees model. Optionally, a summary descriptive table of the efficiency scores can be displayed.
Examples

```r
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, DEA = TRUE, print.table = TRUE,
                        na.rm = TRUE)

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

Description
Density plot for efficiency scores.

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

Arguments
- `df_scores` | data.frame with efficiency scores.
- `model` | character vector. Scoring models in the order of `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, DEA = TRUE, print.table = TRUE,
                        na.rm = TRUE)

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

**efficiencyEAT**

*Efficiency Scores computed through an Efficiency Analysis Trees model.*

---

**Description**

This function computes the efficiency scores for each DMU through an Efficiency Analysis Trees model.

**Usage**

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

**Arguments**

- `data` : data.frame or matrix containing the variables in the model.
- `x` : Column input indexes in data.
- `y` : Column output indexes in data.
- `object` : An EAT object.
- `scores_model` : Mathematical programming model to calculate scores.
  - DDF Directional Distance Function. Efficiency level at 0.
  - WAM.MIP Weighted Additive Model. Measure of Inefficiency Proportions. Efficiency level at 0.
  - WAM.RAM Weighted Additive Model. Range Adjusted Measure of Inefficiency. Efficiency level at 0.
- `digits` : Decimal units for scores.
- `FDH` : logical. If TRUE, FDH scores are also computed with the programming model selected in `scores_model`.
- `print.table` : logical. If TRUE, a summary descriptive table of the efficiency scores is displayed.
- `na.rm` : logical. If TRUE, NA rows are omitted.
efficiencyJitter

Value
A data.frame with the efficiency scores computed through an Efficiency Analysis Trees model. Optionally, a summary descriptive table of the efficiency scores can be displayed.

Examples

```r
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, print.table = TRUE,
             na.rm = TRUE)
```

efficiencyJitter

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 upb and a lower bound lwb in order to show, in addition, the labels whose efficiency score lies between them.

Usage

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

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>An EAT object.</td>
</tr>
<tr>
<td>df_scores</td>
<td>data.frame with efficiency scores (from efficiencyEAT or efficiencyCEAT).</td>
</tr>
<tr>
<td>scores_model</td>
<td>Mathematical programming model to calculate scores.</td>
</tr>
</tbody>
</table>

- **BCC.OUT** BCC model. Output-oriented.
- **BCC.INP** BCC model. Input-oriented.
- **DDF** Directional Distance Function.
- **RSL.OUT** Russell model. Output-oriented.
- **RSL.INP** Russell model. Input-oriented.
- **WAM.MIP** Weighted Additive Model. Measure of Inefficiency Proportions.
**WAM.RAM** Weighted Additive Model. Range Adjusted Measure of Inefficiency.

- **upb**: Numeric. Upper bound for labeling.
- **lwb**: Numeric. Lower bound for labeling.

**Value**

Jitter plot with DMUs and scores.

**Examples**

```r
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")
```

**Description**

This function computes 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. Efficiency level at 1.

**Usage**

```r
efficiencyRFEAT(
  data,
  x,
  y,
  object,
  digits = 3,
  FDH = TRUE,
  print.table = FALSE,
  na.rm = TRUE
)
```
estimEAT

Arguments

- **data**: data.frame or matrix containing the variables in the model.
- **x**: Column input indexes in data.
- **y**: Column output indexes in data.
- **object**: A RFEAT object.
- **digits**: Decimal units for scores.
- **FDH**: logical. If TRUE, FDH scores are computed.
- **print.table**: logical. If TRUE, a summary descriptive table of the efficiency scores is displayed.
- **na.rm**: logical. If TRUE, NA rows are omitted.

Value

A data.frame with the efficiency scores computed through a Random Forest + Efficiency Analysis Trees model. Optionally, a summary descriptive table of the efficiency scores can be displayed.

Examples

```r
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

```r
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.

---

**Description**

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

**Usage**

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

**Arguments**

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

**Value**

Plot with estimated production frontier
Examples

```r
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)
```

---

**generateLv**  
*Train and Test Sets Generation*

**Description**

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

**Usage**

```r
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**

A list structure with the train and the test set.

---

**imp_var_EAT**  
*Breiman’s Variable Importance*

**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**

```r
imp_var_EAT(data, tree, x, y, digits)
```
**imp_var_RFEAT**

**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.

---

**isFinalNode**

**Description**

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

**Usage**

```r
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**

*Layout for nodes in plotEAT*

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**

*Mean Squared Error*

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.
mtry_inputSelection

Value
Mean Square Error at a node.

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

Value
Index of the variables by which the node is divided.

M_Breiman

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.
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:

- **Country**: Country name
- **Continent**: Country continent
- **S_PISA**: PISA score in Science
- **R_PISA**: PISA score in Reading
- **M_PISA**: PISA score in Mathematics
- **NBMC**: Nutritional and Basic Medical Care
- **WS**: Water and Sanitation
- **S**: 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/
### plotEAT

**Efficiency Analysis Trees Plot**

**Description**

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

**Usage**

```r
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**

```r
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 a forest consisting of m trees.

**Usage**

```r
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)

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.
**predict.EAT**  
*Model Prediction for Efficiency Analysis Trees.*

**Description**

This function predicts the expected output by an EAT object.

**Usage**

```r
## S3 method for class 'EAT'
predict(object, newdata, x, ...)
```

**Arguments**

- `object` An EAT object.
- `newdata` `data.frame`. Set of input variables to predict on.
- `x` Inputs index.
- `...` further arguments passed to or from other methods.

**Value**

`data.frame` with the original data and the predicted values.

**Examples**

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

---

**predict.RFEAT**  
*Model prediction for Random Forest + Efficiency Analysis Trees model.*

**Description**

This function predicts the expected output by a RFEAT object.

**Usage**

```r
## S3 method for class 'RFEAT'
predict(object, newdata, x, ...)
```

**Examples**

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

Description

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

Usage

predictFDH(data, x, y)

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.

Value

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

*Efficiency Analysis Trees Predictor*

**Description**

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

**Usage**

predictor(tree, register)

**Arguments**

- **tree** list with the tree nodes.
- **register** Set of independent values.

**Value**

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

---

**preProcess**

*Data Preprocessing for Efficiency Analysis Trees*

**Description**

This function arranges the data in the required format and displays error messages.

**Usage**

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

Arguments

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

Value

It returns a data.frame in the required format.

Description

This function builds an individual tree for Random Forest

Usage

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

Arguments

data  data.frame containing the training set.
x    Vector. Column input indexes in data.
y    Vector. Column output indexes in data.
numStop    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

A list of m trees in forest and the error that will be used in the ranking of the importance of the variables.
**rankingEAT**

*Ranking of Variables by Efficiency Analysis Trees model.*

**Description**

This function computes the variable importance through an Efficiency Analysis Trees model.

**Usage**

```r
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` Importance score value in which a line is graphed.
- `digits` Decimal units.

**Value**

data.frame with the importance scores and a barplot representing the the variable importance if `barplot = TRUE`.

**Examples**

```r
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 model.*

**Description**

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

\texttt{rankingRFEAT(object, barplot = TRUE, digits = 2)}

Arguments

- \texttt{object}  
  A \texttt{RFEAT} object.
- \texttt{barplot}  
  logical. If \texttt{TRUE}, a barplot with importance scores is displayed.
- \texttt{digits}  
  Decimal units.

Value

\texttt{data.frame} with the importance scores and a barplot representing the variable importance if \texttt{barplot = TRUE}.

Examples

\begin{verbatim}
  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)
\end{verbatim}

\section*{RBranch

\textit{Branch Pruning}}

Description

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

Usage

\texttt{RBranch(t, tree)}

Arguments

- \texttt{t}  
  list. A given EAT node.
- \texttt{tree}  
  A list containing the EAT nodes.

Value

A list containing (1) the sum of the errors of the child nodes of the pruned node and (2) the total number of leaf nodes that come from it.
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 + Efficiency Analysis Trees

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

- **data**: data.frame or matrix containing the variables in the model.
- **x**: Column input indexes in data.
- **y**: Column output indexes in data.
- **numStop**: Minimum number of observations in a node for a split to be attempted.
- **m**: Number of trees to be built.
- **s_mtry**: Number of variables randomly sampled as candidates at each split. The available options are:
  - "BRM": \( \frac{\text{in}}{3} \)
  - "DEA1": \( \frac{\text{t.obs}}{2} - \text{out} \)
  - "DEA2": \( \frac{\text{t.obs}}{3} - \text{out} \)
  - "DEA3": \( \text{t.obs} - 2 \times \text{out} \)
  - "DEA4": \( \min(\frac{\text{t.obs}}{\text{out}}, \frac{\text{t.obs}}{3} - \text{out}) \)
  - Any integer
- **na.rm**: 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 containing the individual EAT models.
- **error**: Out-of-Bag error at the forest.
- **OOB**: list containing Out-of-Bag set for each tree.

Examples

```r
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)
```
Create a RFEAT object

Description

This function saves information about the Random Forest for Efficiency Analysis Trees model.

Usage

```r
RFEAT_object(
  data,  # data.frame or matrix containing the variables in the model.
  x,    # Column input indexes in data.
  y,    # Column output indexes in data.
  rownames,  # string. Data rownames.
  numStop,  # Minimun number of observations in a node for a split to be attempted.
  m,    # Number of trees to be built.
  s_mtry,  # Select number of inputs in each split.
    # • "Breiman": in / 3
    # • "DEA1": (t.obs / 2) - out
    # • "DEA2": (t.obs / 3) - out
    # • "DEA3": t.obs - 2 * out
    # • "DEA4": min(t.obs / out, (t.obs / 3) - out)
  na.rm,  # logical. If TRUE, NA rows are omitted.
  forest,  # list containing the individual Efficiency Analysis Trees.
  error,  # Error in Random Forest for Efficiency Analysis Trees.
  OOB  # list containing the observations with which each tree has been trained.
)
```

Arguments

data  # data.frame or matrix containing the variables in the model.
x    # Column input indexes in data.
y    # Column output indexes in data.
rownames    # string. Data rownames.
numStop    # Minimun number of observations in a node for a split to be attempted.
m    # Number of trees to be built.
s_mtry    # Select number of inputs in each split.
  # • "Breiman": in / 3
  # • "DEA1": (t.obs / 2) - out
  # • "DEA2": (t.obs / 3) - out
  # • "DEA3": t.obs - 2 * out
  # • "DEA4": min(t.obs / out, (t.obs / 3) - out)
na.rm    # logical. If TRUE, NA rows are omitted.
forest    # list containing the individual Efficiency Analysis Trees.
error    # Error in Random Forest for Efficiency Analysis Trees.
OOB    # list containing the observations with which each tree has been trained.

Value

A RFEAT object.
**RF_predictor**  
*Random Forest + Efficiency Analysis Trees Predictor*

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

**Usage**
```r
RF_predictor(forest, xn)
```

**Arguments**
- `forest`: list containing the individual Efficiency Analysis Trees.
- `xn`: Row indexes in data.

**Value**
Vector of predictions.

---

**scores**  
*Pruning Scores*

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

**Usage**
```r
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.
**selectTk**

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

**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**

**Select Possible Inputs in Split.**

**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**

**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.

---

**split**

**Split node**

**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)
**split_forest**

**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**  
*Split Node in Random Forest EAT*

**Description**

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

**Usage**

```r
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  Trees for RCV

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 Data Generation

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 simulated data.
Description

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

Usage

Y1.sim(N, nX)

Arguments

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

Value

data.frame with simulated data.
Index

* datasets
  PISAindex, 34

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

EAT, 13
EAT_BCC_in, 15
EAT_BCC_out, 16
EAT_DDF, 16
EAT_frontier_levels, 17
EAT_leaf_stats, 18
EAT_object, 19
EAT_RSL_in, 20
EAT_RSL_out, 20
EAT_size, 21
EAT_WAM, 22
efficiencyCEAT, 22
efficiencyDensity, 24
efficiencyEAT, 25
efficiencyJitter, 26
efficiencyRFEAT, 27
estimEAT, 28

frontier, 29
generateLv, 30
imp_var_EAT, 30
imp_var_RFEAT, 31
isFinalNode, 31
layout, 32
M_Breiman, 33
mse, 32
mtry_inputSelection, 33
PISAindex, 34
plotEAT, 35
plotRFEAT, 35
posIdNode, 36
predict.EAT, 37
predict.RFEAT, 37
predictFDH, 38
predictor, 39
preProcess, 39
RandomEAT, 40
rankingEAT, 41
rankingRFEAT, 41
RBranch, 42
RCV, 43
RF_predictor, 46
RFEAT, 43
RFEAT_object, 45
scores, 46
select_mtry, 47
selectTk, 47
SERules, 48
split, 48
split_forest, 49
treesForRCV, 50
X2Y2.sim, 50
Y1.sim, 51