Package: boostingDEA (via r-universe)

November 4, 2024

Type Package

Title A Boosting Approach to Data Envelopment Analysis

Version 0.1.0

Maintainer Maria D. Guillen <maria.guilleng@umh.es>

Description Includes functions to estimate production frontiers and make ideal output predictions in the Data Envelopment Analysis (DEA) context using both standard models from DEA and Free Disposal Hull (FDH) and boosting techniques. In particular, EATBoosting (Guillen et al., 2023 <doi:10.1016/j.eswa.2022.119134>) and MARSBoosting. Moreover, the package includes code for estimating several technical efficiency measures using different models such as the input and output-oriented radial measures, the input and output-oriented Russell measures, the Directional Distance Function (DDF), the Weighted Additive Measure (WAM) and the Slacks-Based Measure (SBM).

License AGPL (>= 3)

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

Imports Rglpk, dplyr, lpSolveAPI, stats, MLmetrics, methods

URL https://github.com/itsmeryguillen/boostingDEA

BugReports https://github.com/itsmeryguillen/boostingDEA/issues

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Depends R (>= 2.10)

Config/pak/sysreqs libglpk-dev

Repository https://mariaguilleng.r-universe.dev

RemoteUrl https://github.com/mariaguilleng/boostingdea

RemoteRef HEAD

RemoteSha acbe2c514984afe5a9f1bb24752864ab1268fc24

Contents

AddBF *Add a new pair of Basis Functions*

Description

This function adds the best pair of basis functions to the model

Usage

```
AddBF(data, x, y, ForwardModel, knots_list, Kp, minspan, Le, linpreds, err_min)
```

Arguments

data	data data.frame or matrix containing the variables in the model.
x	Column input indexes in data.
y	Column output indexes in data.
ForwardModel	list containing the set of basis functions and the B matrix.
knots_list	list containing the set of selected knots.
Kp	Maximum degree of interaction allowed.
minspan	integer. Minimum number of observations between knots. When minspan = 0, it is calculated as in Friedman's MARS paper section 3.8 with alpha = 0.05.
Le	integer Minimum number of observations before the first and after the final knot.
linpreds	logical. If TRUE, predictors can enter linearly
err_min	Minimum error in the split.

Value

A list containing the matrix of basis functions (B), a list of basis functions (BF), a list of selected knots (knots_list) and the minimum error (err_min).

banks *Taiwanese banks (in 2010)*

Description

The dataset consists of 31 banks operating in Taiwan.

Usage

```
data(banks)
```

Format

banks is a dataframe with 31 banks (rows) and 6 variables (outputs) named Financial.funds (deposits and borrowed funds in millions of TWD), Labor (number of employees), Physical.capital (net amount of fixed assets in millions of TWD), Finalcial.investments (financial assets, securities, and equity investments in millions of TWD), Loans (loans and discounts in millions of TWD) and Revenue (interests from financial investments and loans).

Source

The dataset has been extracted from the "Condition and Performance of Domestic Banks" published by the Central Bank of China (Taiwan) and the Taiwan Economic Journal (TEJ) for the year 2010. The "Condition and Performance of Domestic Banks" was downloaded from http://www.cbc.gov.tw/ct.asp?xItem=1062&ctN

References

Juo, J. C., Fu, T. T., Yu, M. M., & Lin, Y. H. (2015). Profit-oriented productivity change. Omega, 57, 176-187.

BBC_in *Linear programming model for radial input measure*

Description

This function predicts the expected output through a DEA model.

Usage

```
BBC_in(
  data,
  x,
  y,
  dataOriginal = data,
  xOriginal = x,
  yOriginal = y,
  FDH = FALSE
)
```

Arguments

data	data.frame or matrix containing the new variables in the model.
x	Vector. Column input indexes in data.
y	Vector. Column output indexes in data.
dataOriginal	data.frame or matrix containing the original variables used to create the model.
xOriginal	Vector. Column input indexes in original data.
yOriginal	Vector. Column output indexes in original data.
FDH	Binary decision variables

Value

matrix with the the predicted score

BBC_out *Linear programming model for radial output measure*

Description

This function predicts the expected output through a DEA model.

Usage

```
BBC_out(
  data,
  x,
  y,
  dataOriginal = data,
  xOriginal = x,
  yOriginal = y,
  FDH = FALSE
)
```

Arguments

data	data.frame or matrix containing the new variables in the model.
x	Vector. Column input indexes in data.
y	Vector. Column output indexes in data.
dataOriginal	data.frame or matrix containing the original variables used to create the model.
xOriginal	Vector. Column input indexes in original data.
yOriginal	Vector. Column output indexes in original data.
FDH	Binary decision variables

Value

matrix with the the predicted score

`bestEATBoost` *Tuning an EATBoost model*

Description

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

Usage

```
bestEATBoost(
  training,
  test,
  x,
  y,
  num.iterations,
  learning.rate,
  num.leaves,
  verbose = 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`.
`num.iterations`	Maximum number of iterations the algorithm will perform
`learning.rate`	Learning rate that control overfitting of the algorithm. Value must be in (0,1]
`num.leaves`	Maximum number of terminal leaves in each tree at each iteration
`verbose`	Controls the verbosity.

Value

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

`bestMARSBoost` *Tuning an MARSBoost model*

Description

This funcion computes the root mean squared error (RMSE) for a set of MARSBoost models built with a grid of given hyperparameters.

Usage

```
bestMARSBoost(
  training,
  test,
  x,
  y,
  num.iterations,
  learning.rate,
  num.terms,
  verbose = 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`.
`num.iterations`	Maximum number of iterations the algorithm will perform
`learning.rate`	Learning rate that control overfitting of the algorithm. Value must be in (0,1]
`num.terms`	Maximum number of reflected pairs created by the forward algorithm of MARS.
`verbose`	Controls the verbosity.

Value

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

CobbDouglas *Single Output Data Generation*

Description

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

Usage

```
CobbDouglas(N, nX)
```

Arguments

N	Sample size.
nX	Number of inputs. Possible values: 1, 2, 3, 4, 5,6, 9, 12 and 15.

Value

data.frame with the simulated data.

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.

CreateBF *Generate a new pair of Basis Functions*

Description

This function generates two new basis functions from a variable and a knot.

Usage

```
CreateBF(data, xi, knt, B, p)
```

Arguments

data	data.frame or matrix containing the variables in the model.
xi	integer. Variable index of the new basis function(s).
knt	Knot for creating the new basis function(s).
B	matrix of basis functions on which the new pair of functions is added.
p	integer. Parent basis function index.

Value

Matrix of basis function (B) updated with the new basis functions.

CreateCubicBF *Generate a new pair of Cubic Basis Functions*

Description

This function generates two new cubic basis functions from a variable and a knot previously created during MARS algorithm.

Usage

```
CreateCubicBF(data, xi, knt, B, side)
```

Arguments

data	data.frame or matrix containing the variables in the model.
xi	Variable index of the new basis function(s).
knt	Knots for creating the new basis function(s).
B	Matrix of basis functions.
side	Side of the basis function.

Value

Matrix of basis functions updated with the new basis functions.

DDF *Linear programming model for Directional Distance Function measure*

Description

This function predicts the expected output through a DEA model.

Usage

```
DDF(
  data,
  x,
  y,
  dataOriginal = data,
  xOriginal = x,
  yOriginal = y,
  FDH = FALSE,
  direction.vector
)
```

Arguments

data	data.frame or matrix containing the new variables in the model.
x	Vector. Column input indexes in data.
y	Vector. Column output indexes in data.
dataOriginal	data.frame or matrix containing the original variables used to create the model.
xOriginal	Vector. Column input indexes in original data.
yOriginal	Vector. Column output indexes in original data.
FDH	Binary decision variables
direction.vector	Direction vector. Valid values are: dmu (x_0, y_0), unit (unit vector), mean (mean values of each variable) and a user specific vector of the same length as the number of input and output variables

Value

matrix with the the predicted score

DEA *Data Envelope Analysis model*

Description

This function estimates a production frontier satisfying Data Envelope Analysis axioms using the radial output measure.

This function saves information about the DEA model.

Usage

```
DEA(data, x, y)

DEA_object(data, x, y, pred, score)
```

Arguments

data	data.frame or matrix containing the variables in the model.
x	Column input indexes in data.
y	Column output indexes in data.
pred	Output predictions using the BBC radial output measure
score	Efficiency score using the BBC radial output measure

Value

A DEA object.

A DEA object.

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.leaves)
```

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.
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

```
EAT(data, x, y, numStop = 5, max.leaves, 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.
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 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 but enhances some of its disadvantages such as the overfitting problem or the underestimation of technical inefficiency.

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.

EATBoost *Gradient Tree Boosting*

Description

This function estimates a production frontier satisfying some classical production theory axioms, such as monotonicity and determinictiness, which is based upon the adaptation of the machine learning technique known as Gradient Tree Boosting

This function saves information about the EATBoost model

Usage

```
EATBoost(data, x, y, num.iterations, num.leaves, learning.rate)

EATBoost_object(
  data,
  x,
  y,
  num.iterations,
  num.leaves,
  learning.rate,
  EAT.models,
  f0,
  prediction
)
```

Arguments

data	data.frame or matrix containing the variables in the model.
x	Column input indexes in data.
y	Column output indexes in data.
num.iterations	Maximum number of iterations the algorithm will perform
num.leaves	Maximum number of terminal leaves in each tree at each iteration.
learning.rate	Learning rate that control overfitting of the algorithm. Value must be in (0,1]
EAT.models	List of the EAT models created in each iterations
f0	Initial predictions of the model (they correspond to maximum value of each output variable)
prediction	Final predictions of the original data

Value

A EATBoost object.

A EATBoost object.

EAT_object *Create a EAT object*

Description

This function saves information about the Efficiency Analysis Trees model.

Usage

```
EAT_object(data, x, y, rownames, numStop, max.leaves, na.rm, tree)
```

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	Minimum number of observations in a node for a split to be attempted.
max.leaves	Depth of the tree.
na.rm	logical. If TRUE, NA rows are omitted. If FALSE, an error occurs in case of NA rows.
tree	list containing the nodes of the Efficiency Analysis Trees pruned model.

Value

An EAT object.

---

efficiency *Calculate efficiency scores*

Description

Calculates the efficiency score corresponding to the given model using the given measure

Usage

```
efficiency(
  model,
  measure = "rad.out",
  data,
  x,
  y,
  heuristic = TRUE,
  direction.vector = NULL,
  weights = NULL
)
```

Arguments

model	Model object for which efficiency score is computed. Valid classes are: DEA, FDH, EATBoost and MARSBoost.
measure	Efficiency measure used. Valid measures are: rad.out, rad.in
data	data.frame or matrix containing the new variables in the model.
x	Vector. Column input indexes in data.
y	Vector. Column output indexes in data.

heuristic	Only used if model is EATBoost. This indicates whether the heuristic or the exact approach is used.
direction.vector	Only used when measure is DDF.Direction vector. Valid values are: dmu (x_0, y_0), unit (unit vector), mean (mean values of each variable) and a user specific vector of the same length as the number of input and output variables
weights	Only used when measure is WAM. Weights. Valid values are: MIP (Measure of Inefficiency Proportions), RAM (Range Adjusted Measure), BAM (Bounded Adjusted Measure), normalized (normalized weighted additive model) and a user specific vector of the same length as the number of input and output variables

Value

matrix with the the predicted score

ERG *Enhanced Russell Graph measure*

Description

This function predicts the expected output through a DEA model.

Usage

```
ERG(data, x, y, dataOriginal = data, xOriginal = x, yOriginal = y, FDH = FALSE)
```

Arguments

data	data.frame or matrix containing the new variables in the model.
x	Vector. Column input indexes in data.
y	Vector. Column output indexes in data.
dataOriginal	data.frame or matrix containing the original variables used to create the model.
xOriginal	Vector. Column input indexes in original data.
yOriginal	Vector. Column output indexes in original data.
FDH	Binary decision variables

Value

matrix with the the predicted score

EstimCoeffsForward — *Estimate Coefficients in Multivariate Adaptive Frontier Splines during Forward Procedure.*

Description

This function solves a Quadratic Programming Problem to obtain a set of coefficients.

Usage

```
EstimCoeffsForward(B, y)
```

Arguments

B	matrix of basis functions.
y	Output vector in data.

Value

vector with the coefficients estimated.

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.

FDH *Free Disposal Hull model*

Description

This function estimates a production frontier satisfying Free Disposal HUll axioms using the radial output measure.

This function saves information about the FDH model.

Usage

```
FDH(data, x, y)

FDH_object(data, x, y, pred, score)
```

Arguments

data	data.frame or matrix containing the variables in the model.
x	Column input indexes in data.
y	Column output indexes in data.
pred	Output predictions using the BBC radial output measure
score	Efficiency score using the BBC radial output measure

Value

A FDH object.

A FDH object.

get.a.EATBoost *Get* EATBoost *leaves supports*

Description

Calculates the inferior corner of the leaves supports of a EATBoost model.

Usage

```
get.a.EATBoost(EATBoost_model)
```

Arguments

EATBoost_model	Model from class EATBoost from which the data are obtained

Value

data.frame with the leave supports

get.a.trees *Get the inferior corner of the leave support from all trees of* EATBoost

Description

Calculates the inferior corner of the support of all leave nodes of every tree created in the EATBoost model

Usage

```
get.a.trees(EATBoost_model)
```

Arguments

EATBoost_model	Model from class EATBoost from which the data are obtained

Value

list of matrix. The length of the list is equal to the num.iterations of the EATBoost_model. Each matrix corresponds to a tree, where the number of columns is the number of input variables and the number of rows to the number of leaves

get.b.trees *Get the superior corner of the leave support from all trees of* EATBoost

Description

Calculates the superior corner of the support of all leave nodes of every tree created in the EATBoost model

Usage

```
get.b.trees(EATBoost_model)
```

Arguments

EATBoost_model	Model from class EATBoost from which the data are obtained

Value

list of matrix. The length of the list is equal to the num.iterations of the EATBoost_model. Each matrix corresponds to a tree, where the number of columns is the number of input variables and the number of rows to the number of leaves

get.intersection.a *Get intersection between two leaves supports*

Description

Calculates the intersection between two leave nodes from different trees of a EATBoost model.

Usage

```
get.intersection.a(comb_a_actual, comb_b_actual)
```

Arguments

comb_a_actual	Inferior corner of first leave support
comb_b_actual	Superior corner of first leave support

Value

vector with the intersection. NULL if intersection is not valid.

isFinalNode *Is Final Node*

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.

MARSAdapted *Adapted Multivariate Adaptive Frontier Splines*

Description

Create an adapted version of Multivariate Adaptive Regression Splines (MARS) model to estimate a production frontier satisfying some classical production theory axioms, such as monotonicity and concavity.

Usage

```
MARSAdapted(
  data,
  x,
  y,
  nterms,
  Kp = 1,
  d = 2,
  err_red = 0.01,
  minspan = 0,
  endspan = 0,
  linpreds = 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.
nterms	Maximum number of reflected pairs created by the forward algorithm of MARS.
Kp	Maximum degree of interaction allowed. Default is 1.
d	Generalized Cross Validation (GCV) penalty per knot. Default is 2. If it is set to -1, GCV = RSS / n.
err_red	Minimum reduced error rate for the addition of two new basis functions. Default is 0.01.
minspan	Minimum number of observations between knots. When minspan = 0 (default), it is calculated as in Friedman's MARS paper section 3.8 with alpha = 0.05.
endspan	Minimum number of observations before the first and after the final knot. When endspan = 0 (default), it is calculated as in Friedman's MARS paper section 3.8 with alpha = 0.05.
linpreds	logical. If TRUE, predictors can enter linearly
na.rm	logical. If TRUE, NA rows are omitted.

Value

An AdaptedMARS object.

MARSAdaptedSmooth *Smoothing (Forward) Multivariate Adaptive Frontier Splines*

Description

This function smoothes the Forward MARS predictor.

Usage

```
MARSAdaptedSmooth(data, nX, knots, y)
```

Arguments

data	data.frame or matrix containing the variables in the model.
nX	number of inputs in data.
knots	data.frame containing knots from Forward MARS.
y	output indexes in data.

Value

List containing the set of knots from backward (knots), the new cubic knots (cubic_knots) and the set of coefficients (alpha).

MARSAdapted_object *Create an MARSAdapted object*

Description

This function saves information about the adapted Multivariate Adaptive Frontier Splines model.

Usage

```
MARSAdapted_object(
  data,
  x,
  y,
  rownames,
  nterms,
  Kp,
  d,
  err_red,
```

```
  minspan,
  endspan,
  na.rm,
  MARS.Forward,
  MARS.Forward.Smooth
)
```

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.
nterms	Maximum number of terms created by the forward algorithm .
Kp	Maximum degree of interaction allowed. Default is 1.
d	Generalized Cross Validation (GCV) penalty per knot. Default is 2. If set to -1, GCV = RSS / n.
err_red	Minimum reduced error rate for the addition of two new basis functions. Default is 0.01.
minspan	Minimum number of observations between knots. When minspan = 0 (default), it is calculated as in Friedman's MARS paper section 3.8 with alpha = 0.05.
endspan	Minimum number of observations before the first and after the final knot. When endspan = 0 (default), it is calculated as in Friedman's MARS paper section 3.8 with alpha = 0.05.
na.rm	logical. If TRUE, NA rows are omitted.
MARS.Forward	The Multivariate Adaptive Frontier Splines model after applying the forward algorithm without the smoothing procedures
MARS.Forward.Smooth	The Multivariate Adaptive Frontier Splines model after applying the forward algorithm after applying the smoothing procedure

Value

A MARSAdapted object.

MARSBoost *LS-Boosting with adapted Multivariate Adaptive Frontier Splines (MARS)*

Description

This function estimates a production frontier satisfying some classical production theory axioms, such as monotonicity and concavity, which is based upon the adaptation of the machine learning technique known as LS-boosting using adapted Multivariate Adaptive Regression Splines (MARS) as base learners.

This function saves information about the LS-Boosted Multivariate Adaptive Frontier Splines model.

Usage

```
MARSBoost(data, x, y, num.iterations, num.terms, learning.rate)

MARSBoost_object(
  data,
  x,
  y,
  num.iterations,
  learning.rate,
  num.terms,
  MARS.models,
  f0,
  prediction,
  prediction.smooth
)
```

Arguments

`data`	`data.frame` or `matrix` containing the variables in the model.
`x`	Column input indexes in `data`.
`y`	Column output indexes in `data`.
`num.iterations`	Maximum number of iterations the algorithm will perform
`num.terms`	Maximum number of reflected pairs created by the forward algorithm of MARS.
`learning.rate`	Learning rate that control overfitting of the algorithm. Value must be in (0,1]
`MARS.models`	List of the adapted forward MARS models created in each iterations
`f0`	Initial predictions of the model (they correspond to maximum value of each output variable)
`prediction`	Final predictions of the original data without applying the smoothing procedure
`prediction.smooth`	Final predictions of the original data after applying the smoothing procedure

Value

A `MARSBoost` object.

A `MARSBoost` object.

---

mse *Mean Squared Error*

Description

This function computes the mean squared error between two numeric vectors.

Usage

```
mse(y, yPred)
```

Arguments

y	Vector of actual data.
yPred	Vector of predicted values.

Value

Mean Squared Error.

---

mse_tree *Mean Squared Error*

Description

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

Usage

```
mse_tree(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.

posIdNode *Position of the node*

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.DEA *Model Prediction for DEA*

Description

This function predicts the expected output by a DEA object.

Usage

```
## S3 method for class 'DEA'
predict(object, newdata, x, y, ...)
```

Arguments

object	A DEA object.
newdata	data.frame. Set of input variables to predict on.
x	Inputs index.
y	Outputs index.
...	further arguments passed to or from other methods.

Value

data.frame with the predicted values. Valid measures are: rad.out.

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

Description

This function predicts the expected output by an EAT object.

Usage

```
## 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 predicted values.

predict.EATBoost *Model prediction for EATBoost algorithm*

Description

This function predicts the expected output by a EATBoost object.

Usage

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

Arguments

object	A EATBoost 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 predicted values.

predict.FDH *Model Prediction for FDH*

Description

This function predicts the expected output by a FDH object.

Usage

```
## S3 method for class 'FDH'
predict(object, newdata, x, y, ...)
```

Arguments

object	A FDH object.
newdata	data.frame. Set of input variables to predict on.
x	Inputs index.
y	Outputs index.
...	further arguments passed to or from other methods.

Value

data.frame with the predicted values. Valid measures are: rad.out.

predict.MARSAdapted *Model Prediction for Adapted Multivariate Adaptive Frontier Splines.*

Description

This function predicts the expected output by a MARS object.

Usage

```
## S3 method for class 'MARSAdapted'
predict(object, newdata, x, class = 1, ...)
```

Arguments

object	A MARSAdapted object.
newdata	data.frame. Set of input variables to predict on.
x	Inputs index.
class	Model for prediction. 1 MARS Boost without smoothing procedure.2 MARS Boost with smoothing procedure..
...	further arguments passed to or from other methods.

Value

`data.frame` with the predicted values.

---

predict.MARSBoost *Model Prediction for Boosted Multivariate Adaptive Frontier Splines*

Description

This function predicts the expected output by a `MARSBoost` object.

Usage

```
## S3 method for class 'MARSBoost'
predict(object, newdata, x, class = 1, ...)
```

Arguments

`object`	A `MARSBoost` object.
`newdata`	`data.frame`. Set of input variables to predict on.
`x`	Inputs index.
`class`	Model for prediction. `1` MARS Boost without smoothing. `2` MARS Boost with smoothing.
`...`	further arguments passed to or from other methods.

Value

`data.frame` with the predicted values.

---

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 Pre-processing for Multivariate Adaptive Frontier Splines.*

Description

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

Usage

```
preProcess(data, x, y, 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.
na.rm	logical If TRUE, NA rows are omitted.

Value

It returns a data.frame in the required format.

Russell_in *Linear programming model for Russell input measure*

Description

This function predicts the expected output through a DEA model.

Usage

```
Russell_in(
  data,
  x,
  y,
  dataOriginal = data,
  xOriginal = x,
  yOriginal = y,
  FDH = FALSE
)
```

Arguments

data	data.frame or matrix containing the new variables in the model.
x	Vector. Column input indexes in data.
y	Vector. Column output indexes in data.
dataOriginal	data.frame or matrix containing the original variables used to create the model.
xOriginal	Vector. Column input indexes in original data.
yOriginal	Vector. Column output indexes in original data.
FDH	Binary decision variables

Value

matrix with the the predicted score

Russell_out *Linear programming model for Russell output measure*

Description

This function predicts the expected output through a DEA model.

Usage

```
Russell_out(
  data,
  x,
  y,
  dataOriginal = data,
  xOriginal = x,
  yOriginal = y,
  FDH = FALSE
)
```

Arguments

data	data.frame or matrix containing the new variables in the model.
x	Vector. Column input indexes in data.
y	Vector. Column output indexes in data.
dataOriginal	data.frame or matrix containing the original variables used to create the model.
xOriginal	Vector. Column input indexes in original data.
yOriginal	Vector. Column output indexes in original data.
FDH	Binary decision variables

Value

matrix with the the predicted score

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)
```

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.

WAM *Linear programming model for Weighted Additive Model*

Description

This function predicts the expected output through a DEA model.

Usage

```
WAM(
  data,
  x,
  y,
  dataOriginal = data,
  xOriginal = x,
  yOriginal = y,
  FDH = FALSE,
  weights
)
```

Arguments

data	data.frame or matrix containing the new variables in the model.
x	Vector. Column input indexes in data.
y	Vector. Column output indexes in data.
dataOriginal	data.frame or matrix containing the original variables used to create the model.
xOriginal	Vector. Column input indexes in original data.
yOriginal	Vector. Column output indexes in original data.
FDH	Binary decision variables
weights	Weights. Valid values are: MIP (Measure of Inefficiency Proportions), RAM (Range Adjusted Measure), BAM (Bounded Adjusted Measure), normalized (normalized weighted additive model) and a user specific vector of the same length as the number of input and output variables

Value

matrix with the the predicted score

Index