Package 'tsensembler'

August 28, 2017

Title Dynamic Ensembles for Time Series Forecasting

Version 0.0.2

Author Vitor Cerqueira [aut, cre],
Luis Torgo [ctb],
Carlos Soares [ctb]

Maintainer Vitor Cerqueira <cerqueira.vitormanuel@gmail.com>

Description A framework for dynamically combining forecasting models for time series forecasting predictive tasks. It leverages machine learning models from other packages to automatically combine expert advice using metalearning and other state-of-the-art forecasting combination approaches. The predictive methods receive a data matrix as input, representing an embedded time series, and return a predictive ensemble model. The ensemble use generic functions 'predict()' and 'forecast()' to forecast future values of the time series. Moreover, an ensemble can be updated using methods, such as 'update_weights()' or 'update_base_models()'. A complete description of the methods can be found in: Cerqueira, V., Torgo, L., Pinto, F., and Soares, C. ``Arbitrated Ensemble for Time Series Forecasting." to appear at: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. Springer International Publishing, 2017; and Cerqueira, V., Torgo, L., and Soares, C.: ``Arbitrated Ensemble for Solar Radiation Forecasting." International Work-Conference on Artificial Neural Networks. Springer, 2017 <doi:10.1007/978-3-319-59153-7_62>.

Imports xts, RcppRoll, methods, ranger, glmnet, earth, kernlab, Cubist, nnet, gbm, zoo, pls

Suggests testthat

License GPL (>= 2)

Encoding UTF-8

LazyData true

RoxygenNote 6.0.1

URL http://github.com/vcerqueira/tsensembler

NeedsCompilation no

Repository CRAN

Date/Publication 2017-08-28 12:20:49 UTC

R topics documented:

ADE *Arbitrated Dynamic Ensemble*

Description

Arbitrated Dynamic Ensemble (ADE) is an ensemble approach for adaptively combining forecasting models. A metalearning strategy is used that specializes base models across the time series. Each meta-learner is specifically designed to model how apt its base counterpart is to make a prediction for a given test example. This is accomplished by analysing how the error incurred by a given learning model relates to the characteristics of the data. At test time, the base-learners are weighted according to their degree of competence in the input observation, estimated by the predictions of the meta-learners.

Usage

```
ADE(form, data, specs, lambda = 50, omega = 0.5, select_best = FALSE)
```

Arguments

`form`	formula;
`data`	data to train the base models
`specs`	object of class `model_specs-class`. Contains the parameter setting information for training the base models;
`lambda`	window size. Number of observations to compute the recent performance of the base models, according to the committee ratio **omega**. Essentially, the top *omega* models are selected and weighted at each prediction instance, according to their performance in the last *lambda* observations. Defaults to 50 according to empirical experiments;
`omega`	committee ratio size. Essentially, the top *omega* * 100 percent of models are selected and weighted at each prediction instance, according to their performance in the last *lambda* observations. Defaults to .5 according to empirical experiments;
`select_best`	Logical. If true, at each prediction time, a single base model is picked to make a prediction. The picked model is the one that has the lowest loss prediction from the meta models. Defaults to FALSE;

References

Cerqueira, Vitor; Torgo, Luis; Pinto, Fabio; and Soares, Carlos. "Arbitrated Ensemble for Time Series Forecasting" to appear at: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. Springer International Publishing, 2017.

V. Cerqueira, L. Torgo, and C. Soares, “Arbitrated ensemble for solar radiation forecasting,” in International Work-Conference on Artificial Neural Networks. Springer, Cham, 2017, pp. 720–732

See Also

model_specs-class for setting up the ensemble parameters for an **ADE** model; forecast for the forecasting method that uses an **ADE** model for forecasting future values; predict for the method that predicts new held out observations; update_weights for the method used to update the weights of an **ADE** model between successive predict or forecast calls; update_ade_meta for updating (retraining) the meta models of an **ADE** model; update_base_models for the updating (retraining) the base models of an **ADE** ensemble (and respective weights); ade_hat-class for the object that results from predicting with an **ADE** model; and update_ade to update an ADE model, combining functions **update_base_models**, **update_meta_ade**, and **update_weights**.

Examples

```
specs <- model_specs(
  learner = c("bm_ppr", "bm_glm", "bm_mars"),
  learner_pars = list(
    bm_glm = list(alpha = c(0, .5, 1)),
    bm_svr = list(kernel = c("rbfdot", "polydot"),
                  C = c(1, 3)),
    bm_ppr = list(nterms = 4)
  )
)

data("water_consumption")
train <- embed_timeseries(water_consumption, 5)
train <- train[1:300, ] # toy size for checks

model <- ADE(target ~., train, specs)
```

ADE-class *Arbitrated Dynamic Ensemble*

Description

Arbitrated Dynamic Ensemble (ADE) is an ensemble approach for adaptively combining forecasting models. A metalearning strategy is used that specializes base models across the time series. Each meta-learner is specifically designed to model how apt its base counterpart is to make a prediction for a given test example. This is accomplished by analysing how the error incurred by a given learning model relates to the characteristics of the data. At test time, the base-learners are weighted according to their degree of competence in the input observation, estimated by the predictions of the meta-learners.

Slots

`base_ensemble` object of class base_ensemble-class. It contains the base models used that can be used for predicting new data or forecasting future values;

`meta_model` a list containing the meta models, one for each base model. The meta-models are random forests;

`form` formula;

`specs` object of class `model_specs-class`. Contains the parameter setting information for training the base models;

`lambda` window size. Number of observations to compute the recent performance of the base models, according to the committee ratio **omega**. Essentially, the top *omega* models are selected and weighted at each prediction instance, according to their performance in the last *lambda* observations. Defaults to 50 according to empirical experiments;

`omega` committee ratio size. Essentially, the top *omega* * 100 percent of models are selected and weighted at each prediction instance, according to their performance in the last *lambda* observations. Defaults to .5 according to empirical experiments;

`select_best` Logical. If true, at each prediction time, a single base model is picked to make a prediction. The picked model is the one that has the lowest loss prediction from the meta models. Defaults to FALSE;

`recent_series` the most recent `lambda` observations.

References

Cerqueira, Vitor; Torgo, Luis; Pinto, Fabio; and Soares, Carlos. "Arbitrated Ensemble for Time Series Forecasting" to appear at: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. Springer International Publishing, 2017.

V. Cerqueira, L. Torgo, and C. Soares, "Arbitrated ensemble for solar radiation forecasting," in International Work-Conference on Artificial Neural Networks. Springer, Cham, 2017, pp. 720–732

See Also

`model_specs-class` for setting up the ensemble parameters for an **ADE** model; `forecast` for the forecasting method that uses an **ADE** model for forecasting future values; `predict` for the method that predicts new held out observations; `update_weights` for the method used to update the weights of an **ADE** model between successive predict or forecast calls; `update_ade_meta` for updating (retraining) the meta models of an **ADE** model; `update_base_models` for the updating (retraining) the base models of an **ADE** ensemble (and respective weights); `ade_hat-class` for the object that results from predicting with an **ADE** model; and `update_ade` to update an ADE model, combining functions **update_base_models**, **update_meta_ade**, and **update_weights**.

Examples

```
specs <- model_specs(
  learner = c("bm_ppr", "bm_glm", "bm_mars"),
  learner_pars = list(
    bm_glm = list(alpha = c(0, .5, 1)),
    bm_svr = list(kernel = c("rbfdot", "polydot"),
                  C = c(1, 3)),
    bm_ppr = list(nterms = 4)
  )
)

data("water_consumption")
train <- embed_timeseries(water_consumption, 5)
```

```
train <- train[1:300, ] # toy size for checks

model <- ADE(target ~., train, specs)
```

---

ade_hat *Predictions by an ADE ensemble*

Description

Predictions produced by a ADE-class object. It contains **y_hat**, the combined predictions, **Y_hat**, the predictions of each base model, **Y_committee**, the base models used for prediction at each time point, and **E_hat**, the loss predictions by each meta-model.

Usage

```
ade_hat(y_hat, Y_hat, Y_committee, E_hat)
```

Arguments

y_hat	combined predictions of the ensemble ADE. A numeric vector;
Y_hat	a matrix containing the predictions made by individual models;
Y_committee	a list describing the models selected for predictions at each time point (according to **lambda** and **omega**);
E_hat	predictions of error of each base model, estimated by their respective meta model associate;

See Also

ADE-class for generating an ADE ensemble.

Other ensemble predictions: ade_hat-class, dets_hat-class, dets_hat

---

ade_hat-class *Predictions by an ADE ensemble*

Description

Predictions produced by a ADE-class object. It contains **y_hat**, the combined predictions, **Y_hat**, the predictions of each base model, **Y_committee**, the base models used for prediction at each time point, and **E_hat**, the loss predictions by each meta-model.

Slots

`y_hat` combined predictions of the ensemble `ADE-class`. A numeric vector;

`Y_hat` a matrix containing the predictions made by individual models;

`Y_committee` a list describing the models selected for predictions at each time point (according to **lambda** and **omega**);

`E_hat` predictions of error of each base model, estimated by their respective meta model associate;

See Also

`ADE` for generating an ADE ensemble.

Other ensemble predictions: `ade_hat`, `dets_hat-class`, `dets_hat`

`base_ensemble` *base_ensemble*

Description

base_ensemble is a S4 class that contains the base models comprising the ensemble. Besides the base learning algorithms – `base_models` – base_ensemble class contains information about other meta-data used to compute predictions for new upcoming data.

Usage

```
base_ensemble(base_models, pre_weights, form, colnames)
```

Arguments

`base_models`	a list comprising the base models;
`pre_weights`	normalized relative weights of the base learners according to their performance on the available data;
`form`	formula;
`colnames`	names of the columns of the data used to train the **base_models**;

blocked_prequential *Prequential Procedure in Blocks*

Description

Prequential Procedure in Blocks

Usage

```
blocked_prequential(x, nfolds, FUN, .rbind = TRUE, ...)
```

Arguments

x	data to split into nfolds blocks;
nfolds	number of blocks to split data into;
FUN	to apply to train/test;
.rbind	logical. If TRUE, the results from FUN are **rbind**ed;
...	further parameters to FUN

See Also

intraining_estimations function to use as **FUN** parameter.

bm_cubist *Fit Cubist models (M5)*

Description

Learning a M5 model from training data Parameter setting can vary in **committees** and **neighbors** parameters.

Usage

```
bm_cubist(form, data, lpars)
```

Arguments

form	formula
data	training data for building the predictive model
lpars	a list containing the learning parameters

Details

See cubist for a comprehensive description.

Imports learning procedure from **Cubist** package.

See Also

other learning models: bm_mars; bm_ppr; bm_gbm; bm_glm; bm_gaussianprocess; bm_randomforest; bm_pls_pcr; bm_ffnn; bm_svr

Other base learning models: bm_ffnn, bm_gaussianprocess, bm_gbm, bm_glm, bm_mars, bm_pls_pcr, bm_ppr, bm_randomforest, bm_svr

---

bm_ffnn *Fit Feedforward Neural Networks models*

Description

Learning a Feedforward Neural Network model from training data.

Usage

```
bm_ffnn(form, data, lpars)
```

Arguments

form	formula
data	training data for building the predictive model
lpars	a list containing the learning parameters

Details

Parameter setting can vary in **size**, **maxit**, and **decay** parameters.

See nnet for a comprehensive description.

Imports learning procedure from **nnet** package.

See Also

other learning models: bm_mars; bm_ppr; bm_gbm; bm_glm; bm_cubist; bm_randomforest; bm_pls_pcr; bm_gaussianprocess; bm_svr

Other base learning models: bm_cubist, bm_gaussianprocess, bm_gbm, bm_glm, bm_mars, bm_pls_pcr, bm_ppr, bm_randomforest, bm_svr

bm_gaussianprocess *Fit Gaussian Process models*

Description

Learning a Gaussian Process model from training data. Parameter setting can vary in **kernel** and **tolerance**. See gausspr for a comprehensive description.

Usage

```
bm_gaussianprocess(form, data, lpars)
```

Arguments

form	formula
data	training data for building the predictive model
lpars	a list containing the learning parameters

Details

Imports learning procedure from **kernlab** package.

Value

A list containing Gaussian Processes models

See Also

other learning models: bm_mars; bm_ppr; bm_gbm; bm_glm; bm_cubist; bm_randomforest; bm_pls_pcr; bm_ffnn; bm_svr

Other base learning models: bm_cubist, bm_ffnn, bm_gbm, bm_glm, bm_mars, bm_pls_pcr, bm_ppr, bm_randomforest, bm_svr

bm_gbm *Fit Generalized Boosted Regression models*

Description

Learning a Boosted Tree Model from training data. Parameter setting can vary in **interaction.depth**, **n.trees**, and **shrinkage** parameters.

Usage

```
bm_gbm(form, data, lpars)
```

Arguments

form	formula
data	training data for building the predictive model
lpars	a list containing the learning parameters

Details

See gbm for a comprehensive description.

Imports learning procedure from **gbm** package.

See Also

other learning models: bm_mars; bm_ppr; bm_gaussianprocess; bm_glm; bm_cubist; bm_randomforest; bm_pls_pcr; bm_ffnn; bm_svr

Other base learning models: bm_cubist, bm_ffnn, bm_gaussianprocess, bm_glm, bm_mars, bm_pls_pcr, bm_ppr, bm_randomforest, bm_svr

---

bm_glm *Fit Generalized Linear Models*

Description

Learning a Generalized Linear Model from training data. Parameter setting can vary in **alpha**. See glmnet for a comprehensive description.

Usage

```
bm_glm(form, data, lpars)
```

Arguments

form	formula
data	training data for building the predictive model
lpars	a list containing the learning parameters

Details

Imports learning procedure from **glmnet** package.

See Also

other learning models: bm_mars; bm_ppr; bm_gbm; bm_gaussianprocess; bm_cubist; bm_randomforest; bm_pls_pcr; bm_ffnn; bm_svr

Other base learning models: bm_cubist, bm_ffnn, bm_gaussianprocess, bm_gbm, bm_mars, bm_pls_pcr, bm_ppr, bm_randomforest, bm_svr

bm_mars *Fit Multivariate Adaptive Regression Splines models*

Description

Learning a Multivariate Adaptive Regression Splines model from training data.

Usage

```
bm_mars(form, data, lpars)
```

Arguments

form	formula
data	training data for building the predictive model
lpars	a list containing the learning parameters

Details

Parameter setting can vary in **nk**, **degree**, and **thresh** parameters.

See earth for a comprehensive description.

Imports learning procedure from **earth** package.

See Also

other learning models: bm_gaussianprocess; bm_ppr; bm_gbm; bm_glm; bm_cubist; bm_randomforest; bm_pls_pcr; bm_ffnn; bm_svr

Other base learning models: bm_cubist, bm_ffnn, bm_gaussianprocess, bm_gbm, bm_glm, bm_pls_pcr, bm_ppr, bm_randomforest, bm_svr

bm_pls_pcr *Fit PLS/PCR regression models*

Description

Learning aPartial Least Squares or Principal Components Regression from training data

Usage

```
bm_pls_pcr(form, data, lpars)
```

Arguments

form	formula
data	data to train the model
lpars	parameter setting: For this multivariate regression model the main parameter is "method". The available options are "kernelpls", "svdpc", "cppls", "widekernelpls", and "simpls"

Details

Parameter setting can vary in **method**

See mvr for a comprehensive description.

Imports learning procedure from **pls** package.

See Also

other learning models: bm_mars; bm_ppr; bm_gbm; bm_glm; bm_cubist; bm_randomforest; bm_gaussianprocess; bm_ffnn; bm_svr

Other base learning models: bm_cubist, bm_ffnn, bm_gaussianprocess, bm_gbm, bm_glm, bm_mars, bm_ppr, bm_randomforest, bm_svr

---

bm_ppr *Fit Projection Pursuit Regression models*

Description

Learning a Projection Pursuit Regression model from training data. Parameter setting can vary in **nterms** and **sm.method** parameters. See ppr for a comprehensive description.

Usage

```
bm_ppr(form, data, lpars)
```

Arguments

form	formula
data	training data for building the predictive model
lpars	a list containing the learning parameters

Details

Imports learning procedure from **stats** package.

See Also

other learning models: bm_mars; bm_gaussianprocess; bm_gbm; bm_glm; bm_cubist; bm_randomforest; bm_pls_pcr; bm_ffnn; bm_svr

Other base learning models: bm_cubist, bm_ffnn, bm_gaussianprocess, bm_gbm, bm_glm, bm_mars, bm_pls_pcr, bm_randomforest, bm_svr

---

bm_randomforest *Fit Random Forest models*

Description

Learning a Random Forest Model from training data. Parameter setting can vary in **num.trees** and **mtry** parameters.

Usage

```
bm_randomforest(form, data, lpars)
```

Arguments

form	formula
data	training data for building the predictive model
lpars	a list containing the learning parameters

Details

See ranger for a comprehensive description.

Imports learning procedure from **ranger** package.

See Also

other learning models: bm_mars; bm_ppr; bm_gbm; bm_glm; bm_cubist; bm_gaussianprocess; bm_pls_pcr; bm_ffnn; bm_svr

Other base learning models: bm_cubist, bm_ffnn, bm_gaussianprocess, bm_gbm, bm_glm, bm_mars, bm_pls_pcr, bm_ppr, bm_svr

bm_svr *Fit Support Vector Regression models*

Description

Learning a Support Vector Regression model from training data.

Usage

```
bm_svr(form, data, lpars)
```

Arguments

form	formula
data	training data for building the predictive model
lpars	a list containing the learning parameters

Details

Parameter setting can vary in **kernel**, **C**, and **epsilon** parameters.

See ksvm for a comprehensive description.

Imports learning procedure from **kernlab** package.

See Also

other learning models: bm_mars; bm_ppr; bm_gbm; bm_glm; bm_cubist; bm_randomforest; bm_pls_pcr; bm_ffnn; bm_gaussianprocess

Other base learning models: bm_cubist, bm_ffnn, bm_gaussianprocess, bm_gbm, bm_glm, bm_mars, bm_pls_pcr, bm_ppr, bm_randomforest

build_base_ensemble *Wrapper for creating an ensemble*

Description

Using the parameter specifications from model_specs-class, this function trains a set of regression models.

Usage

```
build_base_ensemble(form, data, specs)
```

Arguments

form	formula;
data	data.frame for training the predictive models;
specs	object of class model_specs-class. Contains the information about the parameter setting of the models to train.

Value

An S4 class with the following slots: **base_models**, a list containing the trained models; **pre_weights**, a numeric vector describing the weights of the base models according to their performance in the training data; and **colnames**, the column names of the data, used for reference.

Examples

```
data("water_consumption")
dataset <- embed_timeseries(water_consumption, 5)
specs <- model_specs(c("bm_ppr","bm_svr"), NULL)
M <- build_base_ensemble(target ~., dataset, specs)
```

DETS *Dynamic Ensemble for Time Series*

Description

A Dynamic Ensemble for Time Series (DETS). The DETS ensemble method we present settles on individually pre-trained models which are dynamically combined at run-time to make a prediction. The combination rule is reactive to changes in the environment, rendering an online combined model. The main properties of the ensemble are:

heterogeneity Heterogeneous ensembles are those comprised of different types of base learners. By employing models that follow different learning strategies, use different features and/or data observations we expect that individual learners will disagree with each other, introducing a natural diversity into the ensemble that helps in handling different dynamic regimes in a time series forecasting setting;

responsiveness We promote greater responsiveness of heterogeneous ensembles in time series tasks by making the aggregation of their members' predictions time-dependent. By tracking the loss of each learner over time, we weigh the predictions of individual learners according to their recent performance using a non-linear function. This strategy may be advantageous for better detecting regime changes and also to quickly adapt the ensemble to new regimes.

Usage

```
DETS(form, data, specs, lambda = 50, omega = 0.5)
```

Arguments

form	formula;
data	data frame to train the base models;
specs	object of class model_specs-class. Contains the parameter setting information for training the base models;
lambda	window size. Number of observations to compute the recent performance of the base models, according to the committee ratio **omega**. Essentially, the top *omega* models are selected and weighted at each prediction instance, according to their performance in the last *lambda* observations. Defaults to 50 according to empirical experiments;
omega	committee ratio size. Essentially, the top *omega* models are selected and weighted at each prediction instance, according to their performance in the last *lambda* observations. Defaults to .5 according to empirical experiments;

References

Cerqueira, Vitor; Torgo, Luis; Oliveira, Mariana, and Bernhard Pfahringer. "Dynamic and Heterogeneous Ensembles for Time Series Forecasting." Data Science and Advanced Analytics (DSAA), 2017 IEEE International Conference on. IEEE, 2017.

See Also

model_specs-class for setting up the ensemble parameters for an **DETS** model; forecast for the method that uses an **DETS** model for forecasting future values; predict for the method that predicts new held out observations; update_weights for the method used to update the weights of an **DETS** model between successive predict or forecast calls; update_base_models for the updating (retraining) the base models of an **DETS** ensemble (and respective weights); and dets_hat-class for the object that results from predicting with an **DETS** model.

Examples

```
specs <- model_specs(
 c("bm_ppr", "bm_svr"),
 list(bm_ppr = list(nterms = c(2, 4)),
      bm_svr = list(kernel = c("vanilladot", "polydot"), C = c(1,5)))
)

data("water_consumption");
train <- embed_timeseries(water_consumption, 5);

model <- DETS(target ~., train, specs, lambda = 30, omega = .2)
```

DETS-class *Dynamic Ensemble for Time Series*

Description

A Dynamic Ensemble for Time Series (DETS). The DETS ensemble method we present settles on individually pre-trained models which are dynamically combined at run-time to make a prediction. The combination rule is reactive to changes in the environment, rendering an online combined model. The main properties of the ensemble are:

heterogeneity Heterogeneous ensembles are those comprised of different types of base learners. By employing models that follow different learning strategies, use different features and/or data observations we expect that individual learners will disagree with each other, introducing a natural diversity into the ensemble that helps in handling different dynamic regimes in a time series forecasting setting;

responsiveness We promote greater responsiveness of heterogeneous ensembles in time series tasks by making the aggregation of their members' predictions time-dependent. By tracking the loss of each learner over time, we weigh the predictions of individual learners according to their recent performance using a non-linear function. This strategy may be advantageous for better detecting regime changes and also to quickly adapt the ensemble to new regimes.

Slots

`base_ensemble` object of class `base_ensemble-class`. It contains the base models used that can be used for predicting new data or forecasting future values;

`form` formula;

`specs` object of class `model_specs-class`. Contains the parameter setting information for training the base models;

`lambda` window size. Number of observations to compute the recent performance of the base models, according to the committee ratio **omega**. Essentially, the top *omega* models are selected and weighted at each prediction instance, according to their performance in the last *lambda* observations. Defaults to 50 according to empirical experiments;

`omega` committee ratio size. Essentially, the top *omega* models are selected and weighted at each prediction instance, according to their performance in the last *lambda* observations. Defaults to .5 according to empirical experiments;

`recent_series` the most recent `lambda` observations.

References

Cerqueira, Vitor; Torgo, Luis; Oliveira, Mariana, and Bernhard Pfahringer. "Dynamic and Heterogeneous Ensembles for Time Series Forecasting." Data Science and Advanced Analytics (DSAA), 2017 IEEE International Conference on. IEEE, 2017.

See Also

model_specs-class for setting up the ensemble parameters for an **DETS** model; forecast for the method that uses an **DETS** model for forecasting future values; predict for the method that predicts new held out observations; update_weights for the method used to update the weights of an **DETS** model between successive predict or forecast calls; update_base_models for the updating (retraining) the base models of an **DETS** ensemble (and respective weights); and dets_hat-class for the object that results from predicting with an **DETS** model.

Examples

```
specs <- model_specs(
 c("bm_ppr", "bm_svr"),
 list(bm_ppr = list(nterms = c(2, 4)),
      bm_svr = list(kernel = c("vanilladot", "polydot"), C = c(1,5)))
)

data("water_consumption")
train <- embed_timeseries(water_consumption, 5)

model <- DETS(target ~., train, specs, lambda = 30, omega = .2)
```

---

dets_hat *Predictions by an DETS ensemble*

Description

Predictions by an DETS ensemble

Usage

```
dets_hat(y_hat, Y_hat, Y_committee, W)
```

Arguments

y_hat	combined predictions of the ensemble DETS. A numeric vector;
Y_hat	a matrix containing the predictions made by individual models;
Y_committee	a list describing the models selected for predictions at each time point (according to **lambda** and **omega**);
W	a matrix with the weights of the base models at each prediction time.

Value

Set of results from predicting with a DETS ensemble

See Also

Other ensemble predictions: ade_hat-class, ade_hat, dets_hat-class

dets_hat-class *Predictions by an DETS ensemble*

Description

Predictions by an DETS ensemble

Slots

`y_hat` combined predictions of the ensemble `DETS-class`. A numeric vector;

`Y_hat` a matrix containing the predictions made by individual models;

`Y_committee` a list describing the models selected for predictions at each time point (according to **lambda** and **omega**);

`W` a matrix with the weights of the base models at each prediction time.

See Also

Other ensemble predictions: `ade_hat-class`, `ade_hat`, `dets_hat`

embed_timeseries *Embedding a Time Series*

Description

This function embeds a time series into an Euclidean space. This implementation is based on the function `embed` of **stats** package and has theoretical backgroung on reconstruction of attractors (see Takens, 1981). This shape transformation of the series allows for the use of any regression tool available to learn the time series. The assumption is that there are no long-term dependencies in the data.

Usage

```
embed_timeseries(timeseries, embedding.dimension)
```

Arguments

`timeseries`	a time series of class \"xts\".
`embedding.dimension`	an integer specifying the embedding dimension.

Value

An embedded time series

See Also

embed for the details of the embedding procedure.

Examples

```
## Not run:
require(xts)
ts <- as.xts(rnorm(100L), order.by = Sys.Date() + rnorm(100L))
embedded.ts <- embed.timeseries(ts, 20L)

## End(Not run)
```

---

erfc *Complementary Gaussian Error Function*

Description

Erfc stands for the Complementary Gaussian Error Function. This mathematical formula can be used as a squashing function. Consider x a numeric vector representing the squared error of base models in a given observation. By applying the erfc function on the error, the weight of a given model decays exponentially as its loss increases.

Usage

```
erfc(x, alpha = 2)
```

Arguments

x	A numeric vector. The default value for the parameter lambda presumes that x is in a 0–1 range. In the scope of this package, this is achieved using normalize function;
alpha	parameter used to control the flatness of the erfc curve. Defaults to 2.

Value

The complementary Gaussian error

References

Cerqueira, Vitor; Torgo, Luis; Oliveira, Mariana, and Bernhard Pfahringer. "Dynamic and Heterogeneous Ensembles for Time Series Forecasting." Data Science and Advanced Analytics (DSAA), 2017 IEEE International Conference on. IEEE, 2017.

Examples

```
## Not run:
  erfc(.1)
  erfc(c(.1, .7))

## End(Not run)
```

`forecast` *Forecasting using an ensemble predictive model*

Description

Generic function for forecasting future values of a time series from an `ADE-class` model or a `DETS-class` model.

Usage

```
forecast(object, h)

## S4 method for signature 'ADE'
forecast(object, h)

## S4 method for signature 'DETS'
forecast(object, h)
```

Arguments

`object`	predictive model object. A `ADE-class` or a `DETS-class` ensemble object;
`h`	steps to forecast

Note

the `forecast` generic in **tsensembler** assumes that the data is purely auto-regressive (no external variables), and that the target variable is the first column of the data provided. For a different data setup, the predict methods (`predict`) can be used (with successive calls with updates for multi-step forecasting).

See Also

`predict` for the predict method; `update_weights` for updating the weights of a model after forecasting; `update_base_models` for updating the base models of an ensemble.

Examples

```
specs <- model_specs(
 learner = c("bm_svr", "bm_glm", "bm_mars"),
 learner_pars = NULL
)

data("water_consumption")
dataset <- embed_timeseries(water_consumption, 5)
train <- dataset[1:500, ]

model <- DETS(target ~., train, specs)
model2 <- ADE(target ~., train, specs, lambda = 30)

next_vals_dets <- forecast(model, h = 2)
next_vals_ade <- forecast(model2, h = 2)
```

---

get_y *Get the response values from a data matrix*

Description

Given a formula and a data set, get_y function retrieves the response values.

Usage

```
get_y(data, form)
```

Arguments

data	data set with the response values;
form	formula

---

intraining_estimations *Out-of-bag loss estimations*

Description

A pipeline for retrieving out-of-bag loss estimations

Usage

```
intraining_estimations(train, test, form, specs)
```

Arguments

train	train set from the training set;
test	test set from the training set;
form	formula;
specs	object of class model_specs-class. Contains the specifications of the base models.

Value

A list containing two objects:

mloss loss of base models in **test**

oob out-of-bag test samples

See Also

Other out-of-bag functions: intraining_predictions

---

learning_base_models *Training the base models of an ensemble*

Description

This function uses *train* to build a set of predictive models, according to *specs*

Usage

```
learning_base_models(train, form, specs)
```

Arguments

train	training set to build the predictive models;
form	formula;
specs	object of class model_specs-class

Value

A series of predictive models (base_model), and the weights of the models computed in the training data (preweights).

See Also

build_base_ensemble.

Examples

```
data("water_consumption")
dataset <- embed_timeseries(water_consumption, 5)
specs <- model_specs(c("bm_ppr","bm_svr"), NULL)
M <- build_base_ensemble(target ~., dataset, specs)
```

`model_recent_performance`

Recent performance of models using EMASE

Description

This function computes **EMASE**, Erfc Moving Average Squared Error, to quantify the recent performance of the base models.

Usage

```
model_recent_performance(Y_hat, Y, lambda, omega, pre_weights)
```

Arguments

`Y_hat`	A `data.frame` containing the predictions of each base model;
`Y`	know true values from past data to compare the predictions to;
`lambda`	Window size. Number of periods to average over when computing **MASE**;
`omega`	Ratio of top models in the committee;
`pre_weights`	The initial weights of the models, computed in the available data during the learning phase;

Value

A list containing two objects:

model_scores The weights of the models in each time point

top_models Models in the committee in each time point

See Also

Other weighting base models: `EMASE`, `build_committee`, `get_top_models`, `model_weighting`, `select_best`

`model_specs` *Setup base learning models*

Description

This class sets up the base learning models and respective parameters setting to learn the ensemble.

Usage

```
model_specs(learner, learner_pars = NULL)
```

Arguments

`learner` character vector with the base learners to be trained. Currently available models are:

bm_gaussianprocess Gaussian Process models, from the **kernlab** package. See `gausspr` for a complete description and possible parametrization. See `bm_gaussianprocess` for the function implementation.

bm_ppr Projection Pursuit Regression models, from the **stats** package. See `ppr` for a complete description and possible parametrization. See `bm_ppr` for the function implementation.

bm_glm Generalized Linear Models, from the **glmnet** package. See `glmnet` for a complete description and possible parametrization. See `bm_glm` for the function implementation.

bm_gbm Generalized Boosted Regression models, from the **gbm** package. See `gbm` for a complete description and possible parametrization. See `bm_gbm` for the function implementation.

bm_randomforest Random Forest models, from the **ranger** package. See `ranger` for a complete description and possible parametrization. See `bm_randomforest` for the function implementation.

bm_cubist M5 tree models, from the **Cubist** package. See `cubist` for a complete description and possible parametrization. See `bm_cubist` for the function implementation.

bm_mars Multivariate Adaptive Regression Splines models, from the **earth** package. See `earth` for a complete description and possible parametrization. See `bm_mars` for the function implementation.

bm_svr Support Vector Regression models, from the **kernlab** package. See `ksvm` for a complete description and possible parametrization. See `bm_svr` for the function implementation.

bm_ffnn Feedforward Neural Network models, from the **nnet** package. See `nnet` for a complete description and possible parametrization. See `bm_ffnn` for the function implementation.

bm_pls_pcr Partial Least Regression and Principal Component Regression models, from the **pls** package. See `mvr` for a complete description and possible parametrization. See `bm_pls_pcr` for the function implementation.

learner_pars — a list with parameter setting for the **learner**. For each model, a inner list should be created with the specified parameters.

Check each implementation to see the possible variations of parameters (also examplified below).

Examples

```
# A PPR model and a GLM model with default parameters
model_specs(learner = c("bm_ppr", "bm_glm"), learner_pars = NULL)


# A PPR model and a SVR model. The listed parameters are combined
# with a cartesian product.
# With these specifications an ensemble with 6 predictive base
# models will be created. Two PPR models, one with 2 nterms
# and another with 4; and 4 SVR models, combining the kernel
# and C parameters.
specs <- model_specs(
 c("bm_ppr", "bm_svr"),
 list(bm_ppr = list(nterms = c(2, 4)),
      bm_svr = list(kernel = c("vanilladot", "polydot"), C = c(1,5)))
)

# All parameters currently available (parameter values can differ)
model_specs(
 learner = c("bm_ppr", "bm_svr", "bm_randomforest",
             "bm_gaussianprocess", "bm_cubist", "bm_glm",
             "bm_gbm", "bm_pls_pcr", "bm_ffnn", "bm_mars"
         ),
 learner_pars = list(
    bm_ppr = list(
       nterms = c(2,4),
       sm.method = "supsmu"
     ),
    bm_svr = list(
       kernel = "rbfdot",
       C = c(1,5),
       epsilon = .01
     ),
    bm_glm = list(
       alpha = c(1, 0)
     ),
    bm_randomforest = list(
       num.trees = 500
     ),
    bm_gbm = list(
       interaction.depth = 1,
       shrinkage = c(.01, .005),
       n.trees = c(100)
     ),
    bm_mars = list(
       nk = 15,
```

```
      degree = 3,
      thresh = .001
    ),
    bm_ffnn = list(
      size = 30,
      decay = .01
    ),
    bm_pls_pcr = list(
      method = c("kernelpls", "simpls", "cppls")
    ),
    bm_gaussianprocess = list(
      kernel = "vanilladot",
      tol = .01
    ),
    bm_cubist = list(
      committees = 50,
      neighbors = 0
    )
  )
)
```

`model_specs-class` *Setup base learning models*

Description

This class sets up the base learning models and respective parameters setting to learn the ensemble.

Slots

`learner` character vector with the base learners to be trained. Currently available models are:

- **bm_gaussianprocess** Gaussian Process models, from the **kernlab** package. See `gausspr` for a complete description and possible parametrization. See `bm_gaussianprocess` for the function implementation.
- **bm_ppr** Projection Pursuit Regression models, from the **stats** package. See `ppr` for a complete description and possible parametrization. See `bm_ppr` for the function implementation.
- **bm_glm** Generalized Linear Models, from the **glmnet** package. See `glmnet` for a complete description and possible parametrization. See `bm_glm` for the function implementation.
- **bm_gbm** Generalized Boosted Regression models, from the **gbm** package. See `gbm` for a complete description and possible parametrization. See `bm_gbm` for the function implementation.
- **bm_randomforest** Random Forest models, from the **ranger** package. See `ranger` for a complete description and possible parametrization. See `bm_randomforest` for the function implementation.
- **bm_cubist** M5 tree models, from the **Cubist** package. See `cubist` for a complete description and possible parametrization. See `bm_cubist` for the function implementation.

bm_mars Multivariate Adaptive Regression Splines models, from the **earth** package. See earth for a complete description and possible parametrization. See bm_mars for the function implementation.

bm_svr Support Vector Regression models, from the **kernlab** package. See ksvm for a complete description and possible parametrization. See bm_svr for the function implementation.

bm_ffnn Feedforward Neural Network models, from the **nnet** package. See nnet for a complete description and possible parametrization. See bm_ffnn for the function implementation.

bm_pls_pcr Partial Least Regression and Principal Component Regression models, from the **pls** package. See mvr for a complete description and possible parametrization. See bm_pls_pcr for the function implementation.

`learner_pars` a list with parameter setting for the **learner**. For each model, a inner list should be created with the specified parameters.

Check each implementation to see the possible variations of parameters (also examplified below).

Examples

```
# A PPR model and a GLM model with default parameters
model_specs(learner = c("bm_ppr", "bm_glm"), learner_pars = NULL)


# A PPR model and a SVR model. The listed parameters are combined
# with a cartesian product.
# With these specifications an ensemble with 6 predictive base
# models will be created. Two PPR models, one with 2 nterms
# and another with 4; and 4 SVR models, combining the kernel
# and C parameters.
specs <- model_specs(
 c("bm_ppr", "bm_svr"),
 list(bm_ppr = list(nterms = c(2, 4)),
      bm_svr = list(kernel = c("vanilladot", "polydot"), C = c(1,5)))
)

# All parameters currently available (parameter values can differ)
model_specs(
 learner = c("bm_ppr", "bm_svr", "bm_randomforest",
             "bm_gaussianprocess", "bm_cubist", "bm_glm",
             "bm_gbm", "bm_pls_pcr", "bm_ffnn", "bm_mars"
         ),
 learner_pars = list(
    bm_ppr = list(
       nterms = c(2,4),
       sm.method = "supsmu"
     ),
    bm_svr = list(
       kernel = "rbfdot",
       C = c(1,5),
       epsilon = .01
     ),
```

```
    bm_glm = list(
       alpha = c(1, 0)
     ),
    bm_randomforest = list(
       num.trees = 500
     ),
    bm_gbm = list(
       interaction.depth = 1,
       shrinkage = c(.01, .005),
       n.trees = c(100)
     ),
    bm_mars = list(
       nk = 15,
       degree = 3,
       thresh = .001
     ),
    bm_ffnn = list(
       size = 30,
       decay = .01
     ),
    bm_pls_pcr = list(
       method = c("kernelpls", "simpls", "cppls")
     ),
    bm_gaussianprocess = list(
       kernel = "vanilladot",
       tol = .01
     ),
    bm_cubist = list(
       committees = 50,
       neighbors = 0
     )
  )
)
```

`model_weighting` *Model weighting*

Description

This is an utility function that takes the raw error of models and scales them into a 0-1 range according to one of three strategies:

Usage

```
model_weighting(x, trans = "softmax", ...)
```

Arguments

x	A object describing the loss of each base model
trans	Character value describing the transformation type. The available options are **softmax**, **linear** and **erfc**. The softmax and erfc provide a non-linear transformation where the weights decay exponentially as the relative loss of a given model increases (with respect to all available models). The linear transformation is a simple normalization of values using the max-min method.
...	Further arguments to `normalize` and `proportion` functions \(na.rm = TRUE\)

Details

erfc using the complementary Gaussian error function

softmax using a softmax function

linear A simple normalization using max-min method

These tranformations culminate into the final weights of the models.

Value

An object describing the weights of models

See Also

Other weighting base models: `EMASE`, `build_committee`, `get_top_models`, `model_recent_performance`, `select_best`

---

predict *Predicting new observations using an ensemble*

Description

Initially, the predictions of the base models are collected. Then, the predictions of the loss to be incurred by the base models **E_hat** (estimated by their associate meta models) are computed. The weights of the base models are then estimated according to **E_hat** and the committee of top models. The committee is built according to the *lambda* and *omega* parameters. Finally, the predictions are combined according to the weights and the committee setup.

Usage

```
## S4 method for signature 'ADE'
predict(object, newdata)

## S4 method for signature 'DETS'
predict(object, newdata)

## S4 method for signature 'base_ensemble'
predict(object, newdata)
```

Arguments

object	an object of class ADE-class;
newdata	new data to predict

Examples

```
###### Predicting with an ADE ensemble

specs <- model_specs(
 learner = c("bm_glm", "bm_mars"),
 learner_pars = NULL
)

data("water_consumption")
dataset <- embed_timeseries(water_consumption, 5)
train <- dataset[1:1000, ]
test <- dataset[1001:1500, ]

model <- ADE(target ~., train, specs)

preds <- predict(model, test)


## Not run:

###### Predicting with a DETS ensemble

specs <- model_specs(
 learner = c("bm_svr", "bm_glm", "bm_mars"),
 learner_pars = NULL
)

data("water_consumption")
dataset <- embed_timeseries(water_consumption, 5)
train <- dataset[1:700, ]
test <- dataset[701:1000, ]

model <- DETS(target ~., train, specs, lambda = 50, omega = .2)

preds <- predict(model, test)

## End(Not run)


## Not run:
###### Predicting with a base ensemble

model <- ADE(target ~., train, specs)

basepreds <- predict(model@base_ensemble, test)
```

```
## End(Not run)
```

`roll_mean_matrix` *Computing the rolling mean of the columns of a matrix*

Description

Computing the rolling mean of the columns of a matrix

Usage

```
roll_mean_matrix(x, lambda)
```

Arguments

`x`	a numeric data.frame;
`lambda`	periods to average over when computing the moving average.

`tsensembler` *Dynamic Ensembles for Time Series Forecasting*

Description

This package implements ensemble methods for time series forecasting tasks. Dynamically combining different forecasting models is a common approach to tackle these problems.

Details

The main methods in **tsensembler** are in `ADE-class` and `DETS-class`:

ADE Arbitrated Dynamic Ensemble (ADE) is an ensemble approach for dynamically combining forecasting models using a metalearning strategy called arbitrating. A meta model is trained for each base model in the ensemble. Each meta-learner is specifically designed to model the error of its associate across the time series. At forecasting time, the base models are weighted according to their degree of competence in the input observation, estimated by the predictions of the meta models

DETS Dynamic Ensemble for Time Series (DETS) is similar to **ADE** in the sense that it adaptively combines the base models in an ensemble for time series forecasting. DETS follows a more traditional approach for forecaster combination. It pre-trains a set of heterogeneous base models, and at run-time weights them dynamically according to recent performance. Like **ADE**, the ensemble includes a committee, which dynamically selects a subset of base models that are weighted with a non-linear function

The ensemble methods can be used to `predict` new observations or `forecast` future values of a time series. They can also be updated using generic functions (check see also section).

References

Cerqueira, Vitor; Torgo, Luis; Pinto, Fabio; and Soares, Carlos. "Arbitrated Ensemble for Time Series Forecasting" to appear at: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. Springer International Publishing, 2017.

V. Cerqueira, L. Torgo, and C. Soares, "Arbitrated ensemble for solar radiation forecasting," in International Work-Conference on Artificial Neural Networks. Springer, 2017, pp. 720–732

Cerqueira, Vitor; Torgo, Luis; Oliveira, Mariana, and Bernhard Pfahringer. "Dynamic and Heterogeneous Ensembles for Time Series Forecasting." Data Science and Advanced Analytics (DSAA), 2017 IEEE International Conference on. IEEE, 2017.

See Also

ADE-class for setting up an **ADE** model; and DETS-class for setting up an **DETS** model; see forecast to check the generic function for forecasting future values of a time series using an ensemble from *tsensembler*; see also update_weights and update_base_models to check the generic function for updating the predictive models in an ensemble.

Examples

```
## Not run:

data("water_consumption")
# embedding time series into a matrix
dataset <- embed_timeseries(water_consumption, 5)

# splitting data into train/test
train <- dataset[1:1000,]
test <- dataset[1001:1020, ]

# setting up base model parameters
specs <- model_specs(
  learner = c("bm_ppr","bm_glm","bm_svr","bm_mars"),
  learner_pars = list(
    bm_glm = list(alpha = c(0, .5, 1)),
    bm_svr = list(kernel = c("rbfdot", "polydot"),
                  C = c(1,3)),
    bm_ppr = list(nterms = 4)
  ))

# building the ensemble
model <- ADE(target ~., train, specs)


# forecast next value and update base and meta models
# every three points;
# in the other points, only the weights are updated
predictions <- numeric(nrow(test))
for (i in seq_along(predictions)) {
  predictions[i] <- predict(model, test[i, ])@y_hat
```

```
  if (i %% 3 == 0) {
    model <-
      update_base_models(model,
                         rbind.data.frame(train, test[seq_len(i), ]))

    model <- update_ade_meta(model, rbind.data.frame(train, test[seq_len(i), ]))
  }
  else
    model <- update_weights(model, test[i, ])
}

point_forecast <- forecast(model, h = 5)

# setting up an ensemble of support vector machines
specs2 <-
  model_specs(learner = c("bm_svr"),
              learner_pars = list(
                bm_svr = list(kernel = c("vanilladot", "polydot",
                                         "rbfdot"),
                              C = c(1,3,6))
              ))

model <- DETS(target ~., train, specs2)
preds <- predict(model, test)@y_hat


## End(Not run)
```

---

update_ade *Updating an ADE model*

Description

update_ade is a generic function that combines update_base_models, update_ade_meta, and update_weights.

Usage

```
update_ade(object, newdata)

## S4 method for signature 'ADE'
update_ade(object, newdata)
```

Arguments

object	a ADE-class object.

newdata — data used to update the ADE model. This should be the data used to initially train the models (training set), together with new observations (for example, validation set). Each model is retrained using newdata.

See Also

ADE-class for building an ADE model; update_weights for updating the weights of the ensemble (without retraining the models); update_base_models for updating the base models of an ensemble; and update_ade_meta for updating the meta-models of an ADE model.

Other updating models: update_ade_meta, update_weights

Examples

```
specs <- model_specs(
 learner = c("bm_svr", "bm_glm", "bm_mars"),
 learner_pars = NULL
)

data("water_consumption")
dataset <- embed_timeseries(water_consumption, 5)
# toy size for checks
train <- dataset[1:300, ]
validation <- dataset[301:400, ]
test <- dataset[401:500, ]

model <- ADE(target ~., train, specs)

preds_val <- predict(model, validation)
model <- update_ade(model, rbind.data.frame(train, validation))

preds_test <- predict(model, test)
```

---

update_ade_meta *Updating the metalearning layer of an ADE model*

Description

The **update_ade_meta** function uses new information to update the meta models of an ADE-class ensemble. As input it receives a ADE-class model object class and a new dataset for updating the weights of the base models in the ensemble. This new data should have the same structure as the one used to build the ensemble. Updating the base models of the ensemble is done using the update_base_models function.

Usage

```
update_ade_meta(object, newdata)

## S4 method for signature 'ADE'
update_ade_meta(object, newdata)
```

Arguments

object	a ADE-class object.
newdata	data used to update the meta models. This should be the data used to initially train the meta-models (training set), together with new observations (for example, validation set). Each meta model is retrained using newdata.

See Also

ADE-class for building an ADE model; update_weights for updating the weights of the ensemble (without retraining the models); and update_base_models for updating the base models of an ensemble.

Other updating models: update_ade, update_weights

Examples

```
## Not run:
specs <- model_specs(
 learner = c("bm_svr", "bm_glm", "bm_mars"),
 learner_pars = NULL
)

data("water_consumption")
dataset <- embed_timeseries(water_consumption, 5)
train <- dataset[1:1000, ]
validation <- dataset[1001:1200, ]
test <- dataset[1201:1500, ]

model <- ADE(target ~., train, specs)

preds_val <- predict(model, validation)
model <- update_ade_meta(model, rbind.data.frame(train, validation))

preds_test <- predict(model, test)

## End(Not run)
```

update_base_models *Update the base models of an ensemble*

Description

This is a generic function for updating the base models comprising an ensemble.

Usage

```
update_base_models(object, newdata)

## S4 method for signature 'ADE'
update_base_models(object, newdata)

## S4 method for signature 'DETS'
update_base_models(object, newdata)
```

Arguments

object	an ensemble object, of class DETS-class or ADE-class;
newdata	new data used to update the models. Each base model is retrained, so newdata should be the past data used for initially training the models along with any further available observations.

Details

update_base_models function receives a model object and a new dataset for retraining the base models. This new data should have the same structure as the one used to build the ensemble.

See Also

ADE-class for the ADE model information, and DETS-class for the DETS model information; update_ade_meta for updating the meta models of an ADE ensemble. See update_weights for the method used to update the weights of the ensemble. Updating the weights only changes the information about the recent observations for computing the weights of the base models, while updating the model uses that information to retrain the models.

Examples

```
data("water_consumption")
dataset <- embed_timeseries(water_consumption, 5)
# toy size for checks execution time
train <- dataset[1:300,]
test <- dataset[301:305, ]

specs <- model_specs(c("bm_ppr","bm_glm","bm_mars"), NULL)

model <- ADE(target ~., train, specs)
```

```
predictions <- numeric(nrow(test))
for (i in seq_along(predictions)) {
  predictions[i] <- predict(model, test[i, ])@y_hat
  model <-
    update_base_models(model,
                       rbind.data.frame(train, test[seq_len(i), ]))
}

####

specs2 <- model_specs(c("bm_ppr","bm_randomforest","bm_svr"), NULL)

modeldets <- DETS(target ~., train, specs2)

predictions <- numeric(nrow(test))
# predict new data and update models every three points
# in the remaining points, the only the weights are updated
for (i in seq_along(predictions)) {
  predictions[i] <- predict(modeldets, test[i, ])@y_hat

  if (i %% 3 == 0)
    modeldets <-
      update_base_models(modeldets,
                         rbind.data.frame(train, test[seq_len(i), ]))
  else
    modeldets <- update_weights(modeldets, test[seq_len(i), ])
}
```

`update_weights` *Updating the weights of base models*

Description

Update the weights of base models of a `ADE-class` or `DETS-class` ensemble. This is accomplished by using computing the loss of the base models in new recent observations.

Usage

```
update_weights(object, newdata)

## S4 method for signature 'ADE'
update_weights(object, newdata)

## S4 method for signature 'DETS'
update_weights(object, newdata)
```

Arguments

object	a ADE-class or DETS-class model object;
newdata	new data used to update the most recent observations of the time series. At prediction time these observations are used to compute the weights of the base models

Note

Updating the weights of an ensemble is only necessary between different calls of the functions predict or forecast. Otherwise, if consecutive know observations are predicted (e.g. a validation/test set) the updating is automatically done internally.

See Also

update_weights for the weight updating method for an ADE model, and update_weights for the same method for a DETS model

Other updating models: update_ade_meta, update_ade

Examples

```
data("water_consumption")
dataset <- embed_timeseries(water_consumption, 5)

# toy size for checks
train <- dataset[1:300,]
test <- dataset[301:305, ]

specs <- model_specs(c("bm_ppr","bm_glm","bm_mars"), NULL)
## same with model <- DETS(target ~., train, specs)
model <- ADE(target ~., train, specs)

# if consecutive know observations are predicted (e.g. a validation/test set)
# the updating is automatically done internally.
predictions1 <- predict(model, test)@y_hat

# otherwise, the models need to be updated
predictions <- numeric(nrow(test))
# predict new data and update the weights of the model
for (i in seq_along(predictions)) {
  predictions[i] <- predict(model, test[i, ])@y_hat

  model <- update_weights(model, test[i, ])
}

all.equal(predictions1, predictions)
```

water_consumption *Water Consumption in Oporto city (Portugal) area.*

Description

A time series of classes xts and zoo containing the water consumption levels a specific delivery point at Oporto town, in Portugal.

Usage

```
water_consumption
```

Format

The time series has 1741 values from Jan, 2012 to Oct, 2016 in a daily granularity.

consumption consumption of water, raw value from sensor

Source

https://www.addp.pt/home.php

Index