

Package ‘tspredict’

August 20, 2025

Title Time Series Prediction with Integrated Tuning

Version 1.2.737

Description Time series prediction is a critical task in data analysis, requiring not only the selection of appropriate models, but also suitable data preprocessing and tuning strategies. TSPredIT (Time Series Prediction with Integrated Tuning) is a framework that provides a seamless integration of data preprocessing, decomposition, model training, hyperparameter optimization, and evaluation.

Unlike other frameworks, TSPredIT emphasizes the co-optimization of both preprocessing and modeling steps, improving predictive performance. It supports a variety of statistical and machine learning models, filtering techniques, outlier detection, data augmentation, and ensemble strategies.

More information is available in Salles et al. <[doi:10.1007/978-3-662-68014-8_2](https://doi.org/10.1007/978-3-662-68014-8_2)>.

License MIT + file LICENSE

URL <https://cefet-rj-dal.github.io/tspredict/>,
<https://github.com/cefet-rj-dal/tspredict>

BugReports <https://github.com/cefet-rj-dal/tspredict/issues>

Encoding UTF-8

RoxygenNote 7.3.2

Depends R (>= 4.1.0)

Imports stats, DescTools, e1071, elmNNRcpp, FNN, forecast, hht, KFAS,
mFilter, nnet, randomForest, wavelets, dplyr, daltoolbox

NeedsCompilation no

Author Eduardo Ogasawara [aut, ths, cre] (ORCID:
<<https://orcid.org/0000-0002-0466-0626>>),
Cristiane Gea [aut],
Diego Carvalho [ctb],
Diogo Santos [aut],
Eduardo Bezerra [ctb],
Esther Pacitti [ctb],
Fabio Porto [ctb],
Fernando Alexandrino [aut],
Rebecca Salles [aut],

Vitoria Birindiba [aut],
CEFET/RJ [cph]

Maintainer Eduardo Ogasawara <eogasawara@ieee.org>

Repository CRAN

Date/Publication 2025-08-20 18:50:01 UTC

Contents

adjust_ts_data	3
do_fit	4
do_predict	4
fertilizers	5
MSE.ts	5
R2.ts	6
select_hyper.ts_tune	6
sMAPE.ts	7
tsd	7
ts_arima	8
ts_aug_awareness	8
ts_aug_awaresmooth	9
ts_aug_flip	10
ts_aug_jitter	11
ts_aug_none	11
ts_aug_shrink	12
ts_aug_stretch	13
ts_aug_wormhole	13
ts_data	14
ts_elm	15
ts_fil_ema	16
ts_fil_emd	17
ts_fil_fft	17
ts_fil_hp	18
ts_fil_kalman	19
ts_fil_lowess	20
ts_fil_ma	21
ts_fil_none	21
ts_fil_qes	22
ts_fil_recursive	23
ts_fil_remd	23
ts_fil_seas_adj	24
ts_fil_ses	25
ts_fil_smooth	26
ts_fil_spline	26
ts_fil_wavelet	27
ts_fil_winsor	28
ts_head	28
ts_integtune	29

adjust_ts_data

3

ts_knn

30

ts_mlp

31

ts_norm_an

32

ts_norm_diff

33

ts_norm_ean

34

ts_norm_gminmax

35

ts_norm_none

35

ts_norm_swminmax

36

ts_projection

37

ts_reg

38

ts_regsw

38

ts_rf

39

ts_sample

40

ts_svm

41

ts_tune

42

[.ts_data

43

Index

45

<i>adjust_ts_data</i>	<i>Adjust ts_data</i>
-----------------------	-----------------------

Description

Converts a dataset to a ts_data object

Usage

`adjust_ts_data(data)`

Arguments

data dataset

Value

returns an adjusted ts_data

do_fit	<i>Fit Time Series Model</i>
--------	------------------------------

Description

The actual time series model fitting. This method should be override by descendants.

Usage

```
do_fit(obj, x, y = NULL)
```

Arguments

obj	an object representing the model or algorithm to be fitted
x	a matrix or data.frame containing the input features for training the model
y	a vector or matrix containing the output values to be predicted by the model

Value

returns a fitted object

do_predict	<i>Predict Time Series Model</i>
------------	----------------------------------

Description

The actual time series model prediction. This method should be override by descendants.

Usage

```
do_predict(obj, x)
```

Arguments

obj	an object representing the fitted model or algorithm
x	a matrix or data.frame containing the input features for making predictions

Value

returns the predicted values

fertilizers	<i>Fertilizers (Regression)</i>
-------------	---------------------------------

Description

List of Brazilian fertilizers consumption of N, P2O5, K2O.

- brazil_n: nitrogen consumption from 1961 to 2020.
- brazil_p2o5: phosphate consumption from 1961 to 2020.
- brazil_k2o: potash consumption from 1961 to 2020.

Usage

```
data(fertilizers)
```

Format

list of fertilizers' time series.

Source

This dataset was obtained from the MASS library.

References

International Fertilizer Association (IFA): <http://www.fertilizer.org>.

Examples

```
data(fertilizers)
head(fertilizers$brazil_n)
```

MSE.ts	<i>MSE</i>
--------	------------

Description

Compute the mean squared error (MSE) between actual values and forecasts of a time series

Usage

```
MSE.ts(actual, prediction)
```

Arguments

actual	real observations
prediction	predicted observations

Value

returns a number, which is the calculated MSE

R2.ts	R2
-------	----

Description

Compute the R-squared (R2) between actual values and forecasts of a time series

Usage

```
R2.ts(actual, prediction)
```

Arguments

actual	real observations
prediction	predicted observations

Value

returns a number, which is the calculated R2

select_hyper.ts_tune	<i>Select Optimal Hyperparameters for Time Series Models</i>
----------------------	--

Description

Identifies the optimal hyperparameters by minimizing the error from a dataset of hyperparameters. The function selects the hyperparameter configuration that results in the lowest average error. It wraps the dplyr library.

Usage

```
## S3 method for class 'ts_tune'
select_hyper(obj, hyperparameters)
```

Arguments

obj	a ts_tune object containing the model and tuning settings
hyperparameters	hyperparameters dataset

Value

returns the optimized key number of hyperparameters

`sMAPE.ts`*sMAPE*

Description

Compute the symmetric mean absolute percent error (sMAPE)

Usage

```
sMAPE.ts(actual, prediction)
```

Arguments

actual	real observations
prediction	predicted observations

Value

returns the sMAPE between the actual and prediction vectors

`tsd`*Time series example dataset*

Description

Synthetic dataset of sine function.

- x: correspond time from 0 to 10.
- y: dependent variable for time series modeling.

Usage

```
data(tsd)
```

Format

```
data.frame.
```

Source

This dataset was generated for examples.

Examples

```
data(tsd)
head(tsd)
```

ts_arima	ARIMA
----------	-------

Description

Creates a time series prediction object that uses the AutoRegressive Integrated Moving Average (ARIMA). It wraps the forecast library.

Usage

```
ts_arima()
```

Value

returns a ts_arima object.

Examples

```
library(daltoolbox)
data(tsd)
ts <- ts_data(tsd$y, 0)
ts_head(ts, 3)

samp <- ts_sample(ts, test_size = 5)
io_train <- ts_projection(samp$train)
io_test <- ts_projection(samp$test)

model <- ts_arima()
model <- fit(model, x=io_train$input, y=io_train$output)

prediction <- predict(model, x=io_test$input[1,], steps_ahead=5)
prediction <- as.vector(prediction)
output <- as.vector(io_test$output)

ev_test <- evaluate(model, output, prediction)
ev_test
```

ts_aug_awareness	Augmentation by awareness
------------------	---------------------------

Description

Time series data augmentation is a technique used to increase the size and diversity of a time series dataset by creating new instances of the original data through transformations or modifications. The goal is to improve the performance of machine learning models trained on time series data by reducing overfitting and improving generalization. Awareness reinforce recent data preferably.

Usage

```
ts_aug_awareness(factor = 1)
```

Arguments

factor increase factor for data augmentation

Value

a ts_aug_awareness object.

Examples

```
library(daltoolbox)
data(tsd)

#convert to sliding windows
xw <- ts_data(tsd$y, 10)

#data augmentation using awareness
augment <- ts_aug_awareness()
augment <- fit(augment, xw)
xa <- transform(augment, xw)
ts_head(xa)
```

ts_aug_awaresmooth	<i>Augmentation by awareness smooth</i>
--------------------	---

Description

Time series data augmentation is a technique used to increase the size and diversity of a time series dataset by creating new instances of the original data through transformations or modifications. The goal is to improve the performance of machine learning models trained on time series data by reducing overfitting and improving generalization. Awareness Smooth reinforce recent data preferably. It also smooths noise data.

Usage

```
ts_aug_awaresmooth(factor = 1)
```

Arguments

factor increase factor for data augmentation

Value

a ts_aug_awaresmooth object.

Examples

```
library(daltoolbox)
data(tsd)

#convert to sliding windows
xw <- ts_data(tsd$y, 10)

#data augmentation using awareness
augment <- ts_aug_awaresmooth()
augment <- fit(augment, xw)
xa <- transform(augment, xw)
ts_head(xa)
```

ts_aug_flip

*Augmentation by flip***Description**

Time series data augmentation is a technique used to increase the size and diversity of a time series dataset by creating new instances of the original data through transformations or modifications. The goal is to improve the performance of machine learning models trained on time series data by reducing overfitting and improving generalization. Flip mirror the sliding observations relative to the mean of the sliding windows.

Usage

```
ts_aug_flip()
```

Value

a ts_aug_flip object.

Examples

```
library(daltoolbox)
data(tsd)

#convert to sliding windows
xw <- ts_data(tsd$y, 10)

#data augmentation using flip
augment <- ts_aug_flip()
augment <- fit(augment, xw)
xa <- transform(augment, xw)
ts_head(xa)
```

ts_aug_jitter	<i>Augmentation by jitter</i>
---------------	-------------------------------

Description

Time series data augmentation is a technique used to increase the size and diversity of a time series dataset by creating new instances of the original data through transformations or modifications. The goal is to improve the performance of machine learning models trained on time series data by reducing overfitting and improving generalization. jitter adds random noise to each data point in the time series.

Usage

```
ts_aug_jitter()
```

Value

a ts_aug_jitter object.

Examples

```
library(daltoolbox)
data(tsd)

#convert to sliding windows
xw <- ts_data(tsd$y, 10)

#data augmentation using flip
augment <- ts_aug_jitter()
augment <- fit(augment, xw)
xa <- transform(augment, xw)
ts_head(xa)
```

ts_aug_none	<i>no augmentation</i>
-------------	------------------------

Description

Does not make data augmentation.

Usage

```
ts_aug_none()
```

Value

a ts_aug_none object.

Examples

```
library(daltoolbox)
data(tsd)

#convert to sliding windows
xw <- ts_data(tsd$y, 10)

#no data augmentation
augment <- ts_aug_none()
augment <- fit(augment, xw)
xa <- transform(augment, xw)
ts_head(xa)
```

ts_aug_shrink	<i>Augmentation by shrink</i>
---------------	-------------------------------

Description

Time series data augmentation is a technique used to increase the size and diversity of a time series dataset by creating new instances of the original data through transformations or modifications. The goal is to improve the performance of machine learning models trained on time series data by reducing overfitting and improving generalization. stretch does data augmentation by decreasing the volatility of the time series.

Usage

```
ts_aug_shrink(scale_factor = 0.8)
```

Arguments

scale_factor for shrink

Value

a ts_aug_shrink object.

Examples

```
library(daltoolbox)
data(tsd)

#convert to sliding windows
xw <- ts_data(tsd$y, 10)

#data augmentation using flip
augment <- ts_aug_shrink()
augment <- fit(augment, xw)
xa <- transform(augment, xw)
ts_head(xa)
```

ts_aug_stretch	<i>Augmentation by stretch</i>
----------------	--------------------------------

Description

Time series data augmentation is a technique used to increase the size and diversity of a time series dataset by creating new instances of the original data through transformations or modifications. The goal is to improve the performance of machine learning models trained on time series data by reducing overfitting and improving generalization. stretch does data augmentation by increasing the volatility of the time series.

Usage

```
ts_aug_stretch(scale_factor = 1.2)
```

Arguments

scale_factor for stretch

Value

a ts_aug_stretch object.

Examples

```
library(daltoolbox)
data(tsd)

#convert to sliding windows
xw <- ts_data(tsd$y, 10)

#data augmentation using flip
augment <- ts_aug_stretch()
augment <- fit(augment, xw)
xa <- transform(augment, xw)
ts_head(xa)
```

ts_aug_wormhole	<i>Augmentation by wormhole</i>
-----------------	---------------------------------

Description

Time series data augmentation is a technique used to increase the size and diversity of a time series dataset by creating new instances of the original data through transformations or modifications. The goal is to improve the performance of machine learning models trained on time series data by reducing overfitting and improving generalization. Wormhole does data augmentation by removing lagged terms and adding old terms.

Usage

```
ts_aug_wormhole()
```

Value

a ts_aug_wormhole object.

Examples

```
library(daltoolbox)
data(tsd)

#convert to sliding windows
xw <- ts_data(tsd$y, 10)

#data augmentation using flip
augment <- ts_aug_wormhole()
augment <- fit(augment, xw)
xa <- transform(augment, xw)
ts_head(xa)
```

ts_data

ts_data

Description

Time series data structure used in DAL Toolbox. It receives a vector (representing a time series) or a matrix y (representing a sliding windows). Internal ts_data is matrix of sliding windows with size sw. If sw equals to zero, it store a time series as a single matrix column.

Usage

```
ts_data(y, sw = 1)
```

Arguments

y	output variable
sw	integer: sliding window size.

Value

returns a ts_data object.

Examples

```
data(tsd)
head(tsd)

data <- ts_data(tsd$y)
ts_head(data)

data10 <- ts_data(tsd$y, 10)
ts_head(data10)
```

ts_elm	<i>ELM</i>
--------	------------

Description

Creates a time series prediction object that uses the Extreme Learning Machine (ELM). It wraps the elmNNRcpp library.

Usage

```
ts_elm(preprocess = NA, input_size = NA, nhid = NA, actfun = "purelin")
```

Arguments

preprocess	normalization
input_size	input size for machine learning model
nhid	ensemble size
actfun	defines the type to use, possible values: 'sig', 'radbas', 'tribas', 'relu', 'purelin' (default).

Value

returns a ts_elm object.

Examples

```
library(daltoolbox)
data(tsd)
ts <- ts_data(tsd$y, 10)
ts_head(ts, 3)

samp <- ts_sample(ts, test_size = 5)
io_train <- ts_projection(samp$train)
io_test <- ts_projection(samp$test)

model <- ts_elm(ts_norm_gminmax(), input_size=4, nhid=3, actfun="purelin")
model <- fit(model, x=io_train$input, y=io_train$output)
```

```
prediction <- predict(model, x=io_test$input[1,], steps_ahead=5)
prediction <- as.vector(prediction)
output <- as.vector(io_test$output)

ev_test <- evaluate(model, output, prediction)
ev_test
```

ts_fil_ema

Time Series Exponential Moving Average

Description

Used to smooth out fluctuations, while giving more weight to recent observations. Particularly useful when the data has a trend or seasonality component.

Usage

```
ts_fil_ema(ema = 3)
```

Arguments

ema exponential moving average size

Value

a ts_fil_ema object.

Examples

```
# time series with noise
library(daltoolbox)
data(tsd)
tsd$y[9] <- 2*tsd$y[9]

# filter
filter <- ts_fil_ema(ema = 3)
filter <- fit(filter, tsd$y)
y <- transform(filter, tsd$y)

# plot
plot_ts_pred(y=tsd$y, yadj=y)
```

`ts_fil_emd`*EMD Filter*

Description

EMD Filter

Usage

```
ts_fil_emd(noise = 0.1, trials = 5)
```

Arguments

<code>noise</code>	<code>noise</code>
<code>trials</code>	<code>trials</code>

Valuea `ts_fil_emd` object.**Examples**

```
# time series with noise
library(daltoolbox)
data(tsd)
tsd$y[9] <- 2*tsd$y[9]

# filter
filter <- ts_fil_emd()
filter <- fit(filter, tsd$y)
y <- transform(filter, tsd$y)

# plot
plot_ts_pred(y=tsd$y, yadj=y)
```

`ts_fil_fft`*FFT Filter*

Description

FFT Filter

Usage

```
ts_fil_fft()
```

Value

a ts_fil_fft object.

Examples

```
# time series with noise
library(daltoolbox)
data(tsd)
tsd$y[9] <- 2*tsd$y[9]

# filter
filter <- ts_fil_fft()
filter <- fit(filter, tsd$y)
y <- transform(filter, tsd$y)

# plot
plot_ts_pred(y=tsd$y, yadj=y)
```

ts_fil_hp

Hodrick-Prescott Filter

Description

This filter eliminates the cyclical component of the series, performs smoothing on it, making it more sensitive to long-term fluctuations. Each observation is decomposed into a cyclical and a growth component.

Usage

```
ts_fil_hp(lambda = 100, preserve = 0.9)
```

Arguments

lambda	It is the smoothing parameter of the Hodrick-Prescott filter. $\text{Lambda} = 100 * (\text{frequency})^2$ Correspondence between frequency and lambda values annual => frequency = 1 // lambda = 100 quarterly => frequency = 4 // lambda = 1600 monthly => frequency = 12 // lambda = 14400 weekly => frequency = 52 // lambda = 270400 daily (7 days a week) => frequency = 365 // lambda = 13322500 daily (5 days a week) => frequency = 252 // lambda = 6812100
preserve	value between 0 and 1. Balance the composition of observations and applied filter. Values close to 1 preserve original values. Values close to 0 adopts HP filter values.

Value

a ts_fil_hp object.

Examples

```
# time series with noise
library(daltoolbox)
data(tsd)
tsd$y[9] <- 2*tsd$y[9]

# filter
filter <- ts_fil_hp(lambda = 100*(26)^2) #frequency assumed to be 26
filter <- fit(filter, tsd$y)
y <- transform(filter, tsd$y)

# plot
plot_ts_pred(y=tsd$y, yadj=y)
```

ts_fil_kalman

*Kalman Filter***Description**

The Kalman filter is an estimation algorithm that produces estimates of certain variables based on imprecise measurements to provide a prediction of the future state of the system. It wraps KFAS package.

Usage

```
ts_fil_kalman(H = 0.1, Q = 1)
```

Arguments

- | | |
|---|---|
| H | variance or covariance matrix of the measurement noise. This noise pertains to the relationship between the true system state and actual observations. Measurement noise is added to the measurement equation to account for uncertainties or errors associated with real observations. The higher this value, the higher the level of uncertainty in the observations. |
| Q | variance or covariance matrix of the process noise. This noise follows a zero-mean Gaussian distribution. It is added to the equation to account for uncertainties or unmodeled disturbances in the state evolution. The higher this value, the greater the uncertainty in the state transition process. |

Value

a ts_fil_kalman object.

Examples

```
# time series with noise
library(daltoolbox)
data(tsd)
tsd$y[9] <- 2*tsd$y[9]

# filter
filter <- ts_fil_kalman()
filter <- fit(filter, tsd$y)
y <- transform(filter, tsd$y)

# plot
plot_ts_pred(y=tsd$y, yadj=y)
```

ts_fil_lowess	<i>Lowess Smoothing</i>
---------------	-------------------------

Description

It is a smoothing method that preserves the primary trend of the original observations and is used to remove noise and spikes in a way that allows data reconstruction and smoothing.

Usage

```
ts_fil_lowess(f = 0.2)
```

Arguments

f smoothing parameter. The larger this value, the smoother the series will be. This provides the proportion of points on the plot that influence the smoothing.

Value

a ts_fil_lowess object.

Examples

```
# time series with noise
library(daltoolbox)
data(tsd)
tsd$y[9] <- 2*tsd$y[9]

# filter
filter <- ts_fil_lowess(f = 0.2)
filter <- fit(filter, tsd$y)
y <- transform(filter, tsd$y)

# plot
plot_ts_pred(y=tsd$y, yadj=y)
```

ts_fil_ma	<i>Time Series Moving Average</i>
-----------	-----------------------------------

Description

Used to smooth out fluctuations and reduce noise in a time series.

Usage

```
ts_fil_ma(ma = 3)
```

Arguments

ma	moving average size
----	---------------------

Value

a ts_fil_ma object.

Examples

```
# time series with noise
library(daltoolbox)
data(tsd)
tsd$y[9] <- 2*tsd$y[9]

# filter
filter <- ts_fil_ma(3)
filter <- fit(filter, tsd$y)
y <- transform(filter, tsd$y)

# plot
plot_ts_pred(y=tsd$y, yadj=y)
```

ts_fil_none	<i>no filter</i>
-------------	------------------

Description

Does not make data filter

Usage

```
ts_fil_none()
```

Value

a ts_fil_none object.

Examples

```
# time series with noise
library(daltoolbox)
data(tsd)
tsd$y[9] <- 2*tsd$y[9]

# filter
filter <- ts_fil_none()
filter <- fit(filter, tsd$y)
y <- transform(filter, tsd$y)

# plot
plot_ts_pred(y=tsd$y, yadj=y)
```

ts_fil_qes

*Quadratic Exponential Smoothing***Description**

This code implements quadratic exponential smoothing on a time series. Quadratic exponential smoothing is a smoothing technique that includes components of both trend and seasonality in time series forecasting.

Usage

```
ts_fil_qes(gamma = FALSE)
```

Arguments

gamma If TRUE, enables the gamma seasonality component.

Value

a ts_fil_qes obj.

Examples

```
# time series with noise
library(daltoolbox)
data(tsd)
tsd$y[9] <- 2*tsd$y[9]

# filter
filter <- ts_fil_qes()
filter <- fit(filter, tsd$y)
y <- transform(filter, tsd$y)

# plot
plot_ts_pred(y=tsd$y, yadj=y)
```

ts_fil_recursive	<i>Recursive Filter</i>
------------------	-------------------------

Description

Applies linear filtering to a univariate time series or to each series within a multivariate time series. It is useful for outlier detection, and the calculation is done recursively. This recursive calculation has the effect of reducing autocorrelation among observations, so that for each detected outlier, the filter is recalculated until there are no more outliers in the residuals.

Usage

```
ts_fil_recursive(filter)
```

Arguments

filter	smoothing parameter. The larger the value, the greater the smoothing. The smaller the value, the less smoothing, and the resulting series shape is more similar to the original series.
--------	---

Value

a ts_fil_recursive object.

Examples

```
# time series with noise
library(daltoolbox)
data(tsd)
tsd$y[9] <- 2*tsd$y[9]

# filter
filter <- ts_fil_recursive(filter = 0.05)
filter <- fit(filter, tsd$y)
y <- transform(filter, tsd$y)

# plot
plot_ts_pred(y=tsd$y, yadj=y)
```

ts_fil_remd	<i>EMD Filter</i>
-------------	-------------------

Description

EMD Filter

Usage

```
ts_fil_remd(noise = 0.1, trials = 5)
```

Arguments

noise	noise
trials	trials

Value

a ts_fil_remd object.

Examples

```
# time series with noise
library(daltoolbox)
data(tsd)
tsd$y[9] <- 2*tsd$y[9]

# filter
filter <- ts_fil_remd()
filter <- fit(filter, tsd$y)
y <- transform(filter, tsd$y)

# plot
plot_ts_pred(y=tsd$y, yadj=y)
```

ts_fil_seas_adj	<i>Seasonal Adjustment</i>
-----------------	----------------------------

Description

Removes the seasonal component from the time series without affecting the other components.

Usage

```
ts_fil_seas_adj(frequency = NULL)
```

Arguments

frequency	Frequency of the time series. It is an optional parameter. It can be configured when the frequency of the time series is known.
-----------	---

Value

a ts_fil_seas_adj object.

Examples

```
# time series with noise
library(daltoolbox)
data(tsd)
tsd$y[9] <- 2*tsd$y[9]

# filter
filter <- ts_fil_seas_adj(frequency = 26)
filter <- fit(filter, tsd$y)
y <- transform(filter, tsd$y)

# plot
plot_ts_pred(y=tsd$y, yadj=y)
```

ts_fil_ses

*Simple Exponential Smoothing***Description**

This code implements simple exponential smoothing on a time series. Simple exponential smoothing is a smoothing technique that can include or exclude trend and seasonality components in time series forecasting, depending on the specified parameters.

Usage

```
ts_fil_ses(gamma = FALSE)
```

Arguments

gamma If TRUE, enables the gamma seasonality component.

Value

a ts_fil_ses obj.

Examples

```
# time series with noise
library(daltoolbox)
data(tsd)
tsd$y[9] <- 2*tsd$y[9]

# filter
filter <- ts_fil_ses()
filter <- fit(filter, tsd$y)
y <- transform(filter, tsd$y)

# plot
plot_ts_pred(y=tsd$y, yadj=y)
```

ts_fil_smooth

Time Series Smooth

Description

Used to remove or reduce randomness (noise).

Usage

```
ts_fil_smooth()
```

Value

a ts_fil_smooth object.

Examples

```
# time series with noise
library(daltoolbox)
data(tsd)
tsd$y[9] <- 2*tsd$y[9]

# filter
filter <- ts_fil_smooth()
filter <- fit(filter, tsd$y)
y <- transform(filter, tsd$y)

# plot
plot_ts_pred(y=tsd$y, yadj=y)
```

ts_fil_spline

Smoothing Splines

Description

Fits a cubic smoothing spline to a time series.

Usage

```
ts_fil_spline(spar = NULL)
```

Arguments

spar smoothing parameter. When spar is specified, the coefficient of the integral of the squared second derivative in the fitting criterion (penalized log-likelihood) is a monotone function of spar. #'@return a ts_fil_spline object.

Examples

```
# time series with noise
library(daltoolbox)
data(tsd)
tsd$y[9] <- 2*tsd$y[9]

# filter
filter <- ts_fil_spline(spar = 0.5)
filter <- fit(filter, tsd$y)
y <- transform(filter, tsd$y)

# plot
plot_ts_pred(y=tsd$y, yadj=y)
```

ts_fil_wavelet

*Wavelet Filter***Description**

Wavelet Filter

Usage

```
ts_fil_wavelet(filter = "haar")
```

Arguments

filter Availables wavelet filters: haar, d4, la8, bl14, c6

Value

a ts_fil_wavelet object.

Examples

```
# time series with noise
library(daltoolbox)
data(tsd)
tsd$y[9] <- 2*tsd$y[9]

# filter
filter <- ts_fil_wavelet()
filter <- fit(filter, tsd$y)
y <- transform(filter, tsd$y)

# plot
plot_ts_pred(y=tsd$y, yadj=y)
```

ts_fil_winsor	<i>Winsorization of Time Series</i>
---------------	-------------------------------------

Description

This code implements the Winsorization technique on a time series. Winsorization is a statistical method used to handle extreme values in a time series by replacing them with values closer to the center of the distribution.

Usage

```
ts_fil_winsor()
```

Value

a ts_fil_winsor obj.

Examples

```
# time series with noise
library(daltoolbox)
data(tsd)
tsd$y[9] <- 2*tsd$y[9]

# filter
filter <- ts_fil_winsor()
filter <- fit(filter, tsd$y)
y <- transform(filter, tsd$y)

# plot
plot_ts_pred(y=tsd$y, yadj=y)
```

ts_head	<i>Extract the First Observations from a ts_data Object</i>
---------	---

Description

Returns the first n observations from a ts_data

Usage

```
ts_head(x, n = 6L, ...)
```

Arguments

x	ts_data object
n	number of rows to return
...	optional arguments

Value

returns the first n observations of a ts_data

Examples

```
data(tsd)
data10 <- ts_data(tsd$y, 10)
ts_head(data10)
```

ts_integtune	<i>Time Series Integrated Tune</i>
--------------	------------------------------------

Description

Time Series Tune

Usage

```
ts_integtune(  
  input_size,  
  base_model,  
  folds = 10,  
  ranges = NULL,  
  preprocess = list(ts_norm_gminmax()),  
  augment = list(ts_aug_none())  
)
```

Arguments

input_size	input size for machine learning model
base_model	base model for tuning
folds	number of folds for cross-validation
ranges	a list of hyperparameter ranges to explore
preprocess	list of preprocessing methods
augment	data augmentation method

Value

a ts_integtune object.

Examples

```

library(daltoolbox)
data(tsd)
ts <- ts_data(tsd$y, 10)

samp <- ts_sample(ts, test_size = 5)
io_train <- ts_projection(samp$train)
io_test <- ts_projection(samp$test)

tune <- ts_integtune(input_size=c(3:5), base_model = ts_elm(),
  ranges = list(nhid = 1:5, actfun=c('purelin')),
  preprocess = list(ts_norm_gminmax()))

# Generic model tuning
model <- fit(tune, x=io_train$input, y=io_train$output)

prediction <- predict(model, x=io_test$input[1,], steps_ahead=5)
prediction <- as.vector(prediction)
output <- as.vector(io_test$output)

ev_test <- evaluate(model, output, prediction)
ev_test

```

ts_knn

*KNN time series prediction***Description**

Creates a prediction object that uses the K-Nearest Neighbors (KNN) method for time series regression

Usage

```
ts_knn(preprocess = NA, input_size = NA, k = NA)
```

Arguments

preprocess	normalization
input_size	input size for machine learning model
k	number of k neighbors

Value

returns a ts_knn object.

Examples

```

library(daltoolbox)
data(tsd)
ts <- ts_data(tsd$y, 10)
ts_head(ts, 3)

samp <- ts_sample(ts, test_size = 5)
io_train <- ts_projection(samp$train)
io_test <- ts_projection(samp$test)

model <- ts_knn(ts_norm_gminmax(), input_size=4, k=3)
model <- fit(model, x=io_train$input, y=io_train$output)

prediction <- predict(model, x=io_test$input[1,], steps_ahead=5)
prediction <- as.vector(prediction)
output <- as.vector(io_test$output)

ev_test <- evaluate(model, output, prediction)
ev_test

```

ts_mlp

*MLP***Description**

Creates a time series prediction object that uses the Multilayer Perceptron (MLP). It wraps the nnet library.

Usage

```
ts_mlp(preprocess = NA, input_size = NA, size = NA, decay = 0.01, maxit = 1000)
```

Arguments

preprocess	normalization
input_size	input size for machine learning model
size	number of neurons inside hidden layer
decay	decay parameter for MLP
maxit	maximum number of iterations

Value

returns a ts_mlp object.

Examples

```

library(daltoolbox)
data(tsd)
ts <- ts_data(tsd$y, 10)
ts_head(ts, 3)

samp <- ts_sample(ts, test_size = 5)
io_train <- ts_projection(samp$train)
io_test <- ts_projection(samp$test)

model <- ts_mlp(ts_norm_gminmax(), input_size=4, size=4, decay=0)
model <- fit(model, x=io_train$input, y=io_train$output)

prediction <- predict(model, x=io_test$input[1,], steps_ahead=5)
prediction <- as.vector(prediction)
output <- as.vector(io_test$output)

ev_test <- evaluate(model, output, prediction)
ev_test

```

ts_norm_an

*Time Series Adaptive Normalization***Description**

Transform data to a common scale while taking into account the changes in the statistical properties of the data over time.

Usage

```
ts_norm_an(outliers = outliers_boxplot(), nw = 0)
```

Arguments

outliers	Indicate outliers transformation class. NULL can avoid outliers removal.
nw	integer: window size.

Value

returns a ts_norm_an object.

Examples

```

# time series to normalize
library(daltoolbox)
data(tsd)

# convert to sliding windows
ts <- ts_data(tsd$y, 10)

```



```
ts_head(ts, 3)
summary(ts[,10])

# normalization
preproc <- ts_norm_an()
preproc <- fit(preproc, ts)
tst <- transform(preproc, ts)
ts_head(tst, 3)
summary(tst[,10])
```

ts_norm_diff

Time Series Diff

Description

This function calculates the difference between the values of a time series.

Usage

```
ts_norm_diff(outliers = outliers_boxplot())
```

Arguments

outliers Indicate outliers transformation class. NULL can avoid outliers removal.

Value

returns a ts_norm_diff object.

Examples

```
# time series to normalize
library(daltoolbox)
data(tsd)

# convert to sliding windows
ts <- ts_data(tsd$y, 10)
ts_head(ts, 3)
summary(ts[,10])

# normalization
preproc <- ts_norm_diff()
preproc <- fit(preproc, ts)
tst <- transform(preproc, ts)
ts_head(tst, 3)
summary(tst[,9])
```

ts_norm_ean	<i>Time Series Adaptive Normalization (Exponential Moving Average - EMA)</i>
-------------	--

Description

Creates a normalization object for time series data using an Exponential Moving Average (EMA) method. This normalization approach adapts to changes in the time series and optionally removes outliers.

Usage

```
ts_norm_ean(outliers = outliers_boxplot(), nw = 0)
```

Arguments

outliers	Indicate outliers transformation class. NULL can avoid outliers removal.
nw	windows size

Value

returns a ts_norm_ean object.

Examples

```
# time series to normalize
library(daltoolbox)
data(tsd)

# convert to sliding windows
ts <- ts_data(tsd$y, 10)
ts_head(ts, 3)
summary(ts[,10])

# normalization
preproc <- ts_norm_ean()
preproc <- fit(preproc, ts)
tst <- transform(preproc, ts)
ts_head(tst, 3)
summary(tst[,10])
```

ts_norm_gminmax	<i>Time Series Global Min-Max</i>
-----------------	-----------------------------------

Description

Rescales data, so the minimum value is mapped to 0 and the maximum value is mapped to 1.

Usage

```
ts_norm_gminmax(outliers = outliers_boxplot())
```

Arguments

outliers Indicate outliers transformation class. NULL can avoid outliers removal.

Value

returns a ts_norm_gminmax object.

Examples

```
# time series to normalize
library(daltoolbox)
data(tsd)

# convert to sliding windows
ts <- ts_data(tsd$y, 10)
ts_head(ts, 3)
summary(ts[,10])

# normalization
preproc <- ts_norm_gminmax()
preproc <- fit(preproc, ts)
tst <- transform(preproc, ts)
ts_head(tst, 3)
summary(tst[,10])
```

ts_norm_none	<i>no normalization</i>
--------------	-------------------------

Description

Does not make data normalization.

Usage

```
ts_norm_none()
```

Value

a ts_norm_none object.

Examples

```
library(daltoolbox)
data(tsd)

#convert to sliding windows
xw <- ts_data(tsd$y, 10)

#no data normalization
normalize <- ts_norm_none()
normalize <- fit(normalize, xw)
xa <- transform(normalize, xw)
ts_head(xa)
```

ts_norm_swminmax

Time Series Sliding Window Min-Max

Description

The ts_norm_swminmax function creates an object for normalizing a time series based on the "sliding window min-max scaling" method

Usage

```
ts_norm_swminmax(outliers = outliers_boxplot())
```

Arguments

outliers Indicate outliers transformation class. NULL can avoid outliers removal.

Value

returns a ts_norm_swminmax object.

Examples

```
# time series to normalize
library(daltoolbox)
data(tsd)

# convert to sliding windows
ts <- ts_data(tsd$y, 10)
ts_head(ts, 3)
summary(ts[,10])

# normalization
```

```
preproc <- ts_norm_swminmax()
preproc <- fit(preproc, ts)
tst <- transform(preproc, ts)
ts_head(tst, 3)
summary(tst[,10])
```

ts_projection*Time Series Projection*

Description

Separates a `ts_data` object into input and output components for time series analysis. This function is useful for preparing data for modeling, where the input and output variables are extracted from a time series dataset.

Usage

```
ts_projection(ts)
```

Arguments

`ts` matrix or data.frame containing the time series.

Value

returns a `ts_projection` object.

Examples

```
#setting up a ts_data
data(tsd)
ts <- ts_data(tsd$y, 10)

io <- ts_projection(ts)

#input data
ts_head(io$input)

#output data
ts_head(io$output)
```

 ts_reg

TSReg
Description

Time Series Regression directly from time series Ancestral class for non-sliding windows implementation.

Usage

```
ts_reg()
```

Value

returns ts_reg object

Examples

```
#This is an abstract class.
```

 ts_regsw

TSRegSW
Description

Time Series Regression from Sliding Windows. Ancestral class for Machine Learning Implementation.

Usage

```
ts_regsw(preprocess = NA, input_size = NA)
```

Arguments

preprocess	normalization
input_size	input size for machine learning model

Value

returns a ts_regsw object

Examples

```
#This is an abstract class.
```

ts_rf

*Random Forest***Description**

Creates a time series prediction object that uses the Random Forest. It wraps the randomForest library.

Usage

```
ts_rf(preprocess = NA, input_size = NA, nodesize = 1, ntree = 10, mtry = NULL)
```

Arguments

preprocess	normalization
input_size	input size for machine learning model
nodesize	node size
ntree	number of trees
mtry	number of attributes to build tree

Value

returns a ts_rf object.

Examples

```
library(daltoolbox)
data(tsd)
ts <- ts_data(tsd$y, 10)
ts_head(ts, 3)

samp <- ts_sample(ts, test_size = 5)
io_train <- ts_projection(samp$train)
io_test <- ts_projection(samp$test)

model <- ts_rf(ts_norm_gminmax(), input_size=4, nodesize=3, ntree=50)
model <- fit(model, x=io_train$input, y=io_train$output)

prediction <- predict(model, x=io_test$input[1,], steps_ahead=5)
prediction <- as.vector(prediction)
output <- as.vector(io_test$output)

ev_test <- evaluate(model, output, prediction)
ev_test
```

`ts_sample`*Time Series Sample*

Description

Separates the `ts_data` into training and test. It separates the test size from the last observations minus an offset. The offset is important to allow replication under different recent origins. The data for train uses the number of rows of a `ts_data` minus the test size and offset.

Usage

```
ts_sample(ts, test_size = 1, offset = 0)
```

Arguments

<code>ts</code>	time series.
<code>test_size</code>	integer: size of test data (default = 1).
<code>offset</code>	integer: starting point (default = 0).

Value

returns a list with the two samples

Examples

```
#setting up a ts_data
data(tsd)
ts <- ts_data(tsd$y, 10)

#separating into train and test
test_size <- 3
samp <- ts_sample(ts, test_size)

#first five rows from training data
ts_head(samp$train, 5)

#last five rows from training data
ts_head(samp$train[-c(1:(nrow(samp$train)-5)),])

#testing data
ts_head(samp$test)
```


ts_svm

SVM

Description

Creates a time series prediction object that uses the Support Vector Machine (SVM). It wraps the e1071 library.

Usage

```
ts_svm(
  preprocess = NA,
  input_size = NA,
  kernel = "radial",
  epsilon = 0,
  cost = 10
)
```

Arguments

preprocess	normalization
input_size	input size for machine learning model
kernel	SVM kernel (linear, radial, polynomial, sigmoid)
epsilon	error threshold
cost	this parameter controls the trade-off between achieving a low error on the training data and minimizing the model complexity

Value

returns a ts_svm object.

Examples

```
library(daltoolbox)
data(tsd)
ts <- ts_data(tsd$y, 10)
ts_head(ts, 3)

samp <- ts_sample(ts, test_size = 5)
io_train <- ts_projection(samp$train)
io_test <- ts_projection(samp$test)

model <- ts_svm(ts_norm_gminmax(), input_size=4)
model <- fit(model, x=io_train$input, y=io_train$output)

prediction <- predict(model, x=io_test$input[1,], steps_ahead=5)
prediction <- as.vector(prediction)
```

```
output <- as.vector(io_test$output)

ev_test <- evaluate(model, output, prediction)
ev_test
```

ts_tune

Time Series Tune

Description

Creates a `ts_tune` object for tuning hyperparameters of a time series model. This function sets up a tuning process for the specified base model by exploring different configurations of hyperparameters using cross-validation.

Usage

```
ts_tune(input_size, base_model, folds = 10, ranges = NULL)
```

Arguments

<code>input_size</code>	input size for machine learning model
<code>base_model</code>	base model for tuning
<code>folds</code>	number of folds for cross-validation
<code>ranges</code>	a list of hyperparameter ranges to explore

Value

returns a `ts_tune` object

Examples

```
library(daltoolbox)
data(tsd)
ts <- ts_data(tsd$y, 10)
ts_head(ts, 3)

samp <- ts_sample(ts, test_size = 5)
io_train <- ts_projection(samp$train)
io_test <- ts_projection(samp$test)

tune <- ts_tune(input_size=c(3:5), base_model = ts_elm(ts_norm_gminmax()),
  ranges = list(nhid = 1:5, actfun=c('purelin')))

# Generic model tuning
model <- fit(tune, x=io_train$input, y=io_train$output)

prediction <- predict(model, x=io_test$input[1,], steps_ahead=5)
prediction <- as.vector(prediction)
```

```
output <- as.vector(io_test$output)

ev_test <- evaluate(model, output, prediction)
ev_test
```

[.ts_data

*Subset Extraction for Time Series Data***Description**

Extracts a subset of a time series object based on specified rows and columns. The function allows for flexible indexing and subsetting of time series data.

Usage

```
## S3 method for class 'ts_data'
x[i, j, ...]
```

Arguments

x	ts_data object
i	row i
j	column j
...	optional arguments

Value

returns a new ts_data object

Examples

```
data(tsd)
data10 <- ts_data(tsd$y, 10)
ts_head(data10)
#single line
data10[12,]

#range of lines
data10[12:13,]

#single column
data10[,1]

#range of columns
data10[,1:2]

#range of rows and columns
data10[12:13,1:2]
```

```
#single line and a range of columns  
#'data10[12,1:2]
```

```
#range of lines and a single column  
data10[12:13,1]
```

```
#single observation  
data10[12,1]
```

Index

- * **datasets**
 - fertilizers, [5](#)
 - tsd, [7](#)
- [.ts_data, [43](#)
- adjust_ts_data, [3](#)
- do_fit, [4](#)
- do_predict, [4](#)
- fertilizers, [5](#)
- MSE.ts, [5](#)
- R2.ts, [6](#)
- select_hyper.ts_tune, [6](#)
- sMAPE.ts, [7](#)
- ts_arima, [8](#)
- ts_aug_awareness, [8](#)
- ts_aug_awaresmooth, [9](#)
- ts_aug_flip, [10](#)
- ts_aug_jitter, [11](#)
- ts_aug_none, [11](#)
- ts_aug_shrink, [12](#)
- ts_aug_stretch, [13](#)
- ts_aug_wormhole, [13](#)
- ts_data, [14](#)
- ts_elm, [15](#)
- ts_fil_ema, [16](#)
- ts_fil_emd, [17](#)
- ts_fil_fft, [17](#)
- ts_fil_hp, [18](#)
- ts_fil_kalman, [19](#)
- ts_fil_lowess, [20](#)
- ts_fil_ma, [21](#)
- ts_fil_none, [21](#)
- ts_fil_qes, [22](#)
- ts_fil_recursive, [23](#)
- ts_fil_remd, [23](#)
- ts_fil_seas_adj, [24](#)
- ts_fil_ses, [25](#)
- ts_fil_smooth, [26](#)
- ts_fil_spline, [26](#)
- ts_fil_wavelet, [27](#)
- ts_fil_winsor, [28](#)
- ts_head, [28](#)
- ts_integtune, [29](#)
- ts_knn, [30](#)
- ts_mlp, [31](#)
- ts_norm_an, [32](#)
- ts_norm_diff, [33](#)
- ts_norm_ean, [34](#)
- ts_norm_gminmax, [35](#)
- ts_norm_none, [35](#)
- ts_norm_swminmax, [36](#)
- ts_projection, [37](#)
- ts_reg, [38](#)
- ts_regsw, [38](#)
- ts_rf, [39](#)
- ts_sample, [40](#)
- ts_svm, [41](#)
- ts_tune, [42](#)
- tsd, [7](#)