Package: tspredit (via r-universe)

August 29, 2024

Title Time Series Prediction Integrated Tuning

Version 1.0.777

Description Prediction is one of the most important activities while working with time series. There are many alternative ways to model the time series. Finding the right one is challenging to model them. Most data-driven models (either statistical or machine learning) demand tuning. Setting them right is mandatory for good predictions. It is even more complex since time series prediction also demands choosing a data pre-processing that complies with the chosen model. Many time series frameworks have features to build and tune models. The package differs as it provides a framework that seamlessly integrates tuning data pre-processing activities with the building of models. The package provides functions for defining and conducting time series prediction, including data pre(post)processing, decomposition, tuning, modeling, prediction, and accuracy assessment. More information is available at Izau et al. <doi:10.5753/sbbd.2022.224330>.

License MIT + file LICENSE

URL `https://github.com/cefet-rj-dal/daltoolbox`,
`https://cefet-rj-dal.github.io/daltoolbox/`

Encoding UTF-8

RoxygenNote 7.3.2

Imports dplyr, stats, forecast, mFilter, DescTools, hht, wavelets, KFAS, daltoolbox

Config/reticulate list(packages = list(list(package = ``scipy"), list(package = ``torch"), list(package = ``pandas"), list(package = ``numpy"), list(package = ``matplotlib"), list(package = ``scikit-learn")))

NeedsCompilation no

Author Eduardo Ogasawara [aut, ths, cre] (<`https://orcid.org/0000-0002-4466-0626`>), Cristiane Gea [aut], Diogo Santos [aut], Rebecca Salles [aut], Vitoria Birindiba [aut], Carla Pacheco [aut], Eduardo Bezerra [aut], Esther Pacitti [aut], Fabio Porto [aut], Federal Center for Technological Education of Rio de Janeiro (CEFET/RJ) [cph]

Maintainer Eduardo Ogasawara <eogasawara@ieee.org>

Depends R (>= 3.5.0)

Date/Publication 2024-07-29 14:20:02 UTC

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

RemoteUrl https://github.com/cran/tspredit

RemoteRef HEAD

RemoteSha c570cf9d883ddcd07d4cb76f1fddf56c34603dbb

Contents

fertilizers — *Fertilizers (Regression)*

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)
```

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(sin_data)

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

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

---

ts_aug_awaresmooth *Augmentation by awareness smooth*

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(sin_data)

#convert to sliding windows
xw <- ts_data(sin_data$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(sin_data)

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

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

ts_aug_jitter — *Augmentation by jitter*

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(sin_data)

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

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

ts_aug_none — *no augmentation*

Description

Does not make data augmentation.

Usage

```
ts_aug_none()
```

Value

a `ts_aug_none` object.

Examples

```
library(daltoolbox)
data(sin_data)

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

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

---

ts_aug_shrink *Augmentation by shrink*

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(sin_data)

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

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

ts_aug_stretch *Augmentation by stretch*

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(sin_data)

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

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

ts_aug_wormhole *Augmentation by wormhole*

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(sin_data)

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

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

---

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(sin_data)
sin_data$y[9] <- 2*sin_data$y[9]

# filter
```

```
filter <- ts_fil_ema(ema = 3)
filter <- fit(filter, sin_data$y)
y <- transform(filter, sin_data$y)

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

---

ts_fil_emd *EMD Filter*

Description

EMD Filter

Usage

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

Arguments

noise	noise
trials	trials

Value

a ts_fil_emd object.

Examples

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

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

# plot
plot_ts_pred(y=sin_data$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(sin_data)
sin_data$y[9] <- 2*sin_data$y[9]

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

# plot
plot_ts_pred(y=sin_data$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. Lambda = 100*(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(sin_data)
sin_data$y[9] <- 2*sin_data$y[9]

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

# plot
plot_ts_pred(y=sin_data$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(sin_data)
sin_data$y[9] <- 2*sin_data$y[9]

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

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

`ts_fil_lowess` *Lowess Smoothing*

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(sin_data)
sin_data$y[9] <- 2*sin_data$y[9]

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

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

`ts_fil_ma` *Time Series Moving Average*

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(sin_data)
sin_data$y[9] <- 2*sin_data$y[9]

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

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

---

ts_fil_none *no filter*

Description

Does not make data filter

Usage

```
ts_fil_none()
```

Value

a `ts_fil_none` object.

Examples

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

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

# plot
plot_ts_pred(y=sin_data$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(sin_data)
sin_data$y[9] <- 2*sin_data$y[9]

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

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

`ts_fil_recursive` *Recursive Filter*

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(sin_data)
sin_data$y[9] <- 2*sin_data$y[9]

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

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

---

ts_fil_remd *EMD Filter*

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(sin_data)
sin_data$y[9] <- 2*sin_data$y[9]

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

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

ts_fil_seas_adj *Seasonal Adjustment*

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(sin_data)
sin_data$y[9] <- 2*sin_data$y[9]

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

# plot
plot_ts_pred(y=sin_data$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(sin_data)
sin_data$y[9] <- 2*sin_data$y[9]

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

# plot
plot_ts_pred(y=sin_data$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(sin_data)
sin_data$y[9] <- 2*sin_data$y[9]

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

# plot
plot_ts_pred(y=sin_data$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(sin_data)
sin_data$y[9] <- 2*sin_data$y[9]

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

# plot
plot_ts_pred(y=sin_data$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(sin_data)
sin_data$y[9] <- 2*sin_data$y[9]

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

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

---

ts_fil_winsor — *Winsorization of Time Series*

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(li = 0.05)
```

Arguments

`li`	Defines the Winsorization limit (proportion of values to be trimmed)

Value

a `ts_fil_winsor` obj.

Examples

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

# filter
filter <- ts_fil_winsor()
filter <- fit(filter, sin_data$y)
```

```
y <- transform(filter, sin_data$y)

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

ts_maintune *Time Series Tune*

Description

Time Series Tune

Usage

```
ts_maintune(
  input_size,
  base_model,
  folds = 10,
  preprocess = list(daltoolbox::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
preprocess	list of preprocessing methods
augment	data augmentation method

Value

a ts_maintune object.

Examples

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

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

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

# Generic model tunning
```

```
model <- fit(tune, x=io_train$input, y=io_train$output, ranges)

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_none` *no normalization*

Description

Does not make data normalization.

Usage

```
ts_norm_none()
```

Value

a `ts_norm_none` object.

Examples

```
library(daltoolbox)
data(sin_data)

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

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

Index