Package 'tspredit'

February 11, 2026

Title Time Series Prediction with Integrated Tuning

Version 1.2.767

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>.

License MIT + file LICENSE

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

BugReports https://github.com/cefet-rj-dal/tspredit/issues

Encoding UTF-8

RoxygenNote 7.3.3

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 2026-02-11 08:00:02 UTC

Contents

`adjust_ts_data` *Adjust* `ts_data`

Description

Convert a compatible dataset to a `ts_data` object by setting column names, class, and the `sw` attribute consistently.

Usage

```
adjust_ts_data(data)
```

Arguments

data	Matrix or data.frame to adjust.

Value

An adjusted ts_data.

---

bioenergy *FAOSTAT Bioenergy Database*

Description

Bioenergy data from FAOSTAT. Data Type: Bioenergy consumption and production. Category: Environment. Creation Date 2024.

Usage

```
data(bioenergy)
```

Format

A list of time series.

Details

Series are named as <country>_<bio_consumption|bio_production> and contain annual values.

Source

FAOSTAT Bioenergy Database

References

FAO 2024. FAOSTAT Bioenergy, FAO, Rome, Italy. ; United Nations Statistics Division (UNSD), 2011; International Recommendations for Energy Statistics (IRES).

Examples

```
# Load bioenergy list and plot one series
data(bioenergy)
# bioenergy <- loadfulldata(bioenergy)
series <- bioenergy[[1]]
ts.plot(series, ylab = "TJ", xlab = "Year", main = "Bioenergy example")
```

CATS *CATS Time Series Competition*

Description

Univariate time series from the CATS (Competition on Artificial Time Series) benchmark. Data Type: Artificial time series with missing blocks. Category: Benchmark. Observations: 5,000 (4,900 known, 100 missing). The dataset contains five non-consecutive blocks of 20 missing values each. Competitors were asked to predict these 100 unknown points, and performance was evaluated using MSE (E1 for all unknowns and E2 for the first 80 points).

Usage

```
data(CATS)
```

Format

A data frame with five columns and 980 rows. Each column represents a known segment of the time series.

Details

The CATS benchmark contains artificial series with five nonconsecutive missing blocks of 20 points each. Models must impute or forecast the missing blocks; evaluation typically uses MSE over all missing points.

Source

CATS Time Series Competition

References

Lendasse, A., Oja, E., Simula, O., Verleysen, M., et al. (2004). *Time Series Prediction Competition: The CATS Benchmark*. In IJCNN'2004 - International Joint Conference on Neural Networks. Lendasse, A., Oja, E., Simula, O., Verleysen, M. (2007). *Time Series Prediction Competition: The CATS Benchmark*. Neurocomputing, 70(13-15), 2325–2329.

Examples

```
# Load CATS dataset
data(CATS)
# CATS <- loadfulldata(CATS)
```

climate *FAOSTAT Temperature Change on Land*

Description

Statistics of surface temperature anomalies on land, based on NASA-GISS GISTEMP data. Data Type: Temperature Anomalies. Category: Environment. Creation Date 2024.

Usage

```
data(climate)
```

Format

A list of time series.

Source

NASA-GISS GISTEMP

References

FAO, 2024. FAOSTAT Land, Inputs and Sustainability; Climate Change Indicators; Temperature change on land. GISTEMP Team, 2024: GISS Surface Temperature Analysis. NASA Goddard Institute for Space Studies. Hansen, J. et al., 1981–2019: Multiple foundational studies on global temperature analysis.

Examples

```
# Load climate list and plot one series
data(climate)
# climate <- loadfulldata(climate)
series <- climate[[1]]
ts.plot(series, ylab = "Temperature change (°C)", xlab = "Year",
        main = "Temperature change on land")
```

do_fit *Fit Time Series Model*

Description

Generic for fitting a time series model. Descendants should implement `do_fit.<class>`.

Usage

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

Arguments

obj	Model object to be fitted.
x	Matrix or data.frame with input features.
y	Vector or matrix with target values.

Value

A fitted object (same class as obj).

---

do_predict *Predict Time Series Model*

Description

Generic for predicting with a fitted time series model. Descendants should implement do_predict.<class>.

Usage

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

Arguments

obj	Fitted model object.
x	Matrix or data.frame with input features to predict.

Value

Numeric vector with predicted values.

---

emissions *FAOSTAT Emissions Totals*

Description

National and global estimates of greenhouse gas (GHG) emissions. Data Type: Greenhouse gas emissions. Category: Environment. Creation Date 2023.

Usage

```
data(emissions)
```

Format

A list of time series.

Source

FAOSTAT Emissions Totals.

References

FAO, 2023. FAOSTAT Climate Change: Agrifood systems emissions, Emissions Totals. IPCC Guidelines and Reports: 1996, 2000, 2006, 2014, 2019. PRIMAP-hist dataset v2.4.2: Gütschow et al., 2023.

Examples

```
# Load emissions list and plot one series
data(emissions)
# emissions <- loadfulldata(emissions)
series <- emissions[[1]]
ts.plot(series, ylab = "kt CO2e", xlab = "Year", main = "Emissions example (CH4/N2O)")
```

EUNITE.Loads *EUNITE Competition – Half-Hourly Electrical Loads*

Description

Half-hourly electrical load time series from the EUNITE forecasting competition. Data Type: Electrical load measurements. Category: Benchmark. Observations: 730 days, 48 intervals per day. This dataset contains univariate time series with half-hour resolution covering 1997–1998. It was used to forecast daily maximum loads in January 1999. Competitors were evaluated using MAPE and MAXIMAL prediction errors. Regressors such as temperature and calendar variables were also provided.

Usage

```
data(EUNITE.Loads)
```

Format

A data frame with 730 rows and 48 numeric columns. Each column corresponds to one half-hour interval, from 00:00 to 24:00.

Details

The EUNITE competition focused on forecasting maximum daily electrical loads for January 1999 using half-hourly load profiles and auxiliary regressors. Series are provided in a wide format with 48 half-hour intervals as columns.

Source

EUNITE Competition 2001 dataset (original competition website currently unavailable).

References

Chen, B.-J., Chang, M.-W., & Lin, C.-J. (2004). *Load forecasting using support vector machines: a study on EUNITE competition 2001*. IEEE Transactions on Power Systems, 19(4), 1821-1830.

Examples

```
# Load the dataset
data(EUNITE.Loads)
# EUNITE.Loads <- loadfulldata(EUNITE.Loads)

# Inspect the first few half-hourly columns (00:00 to 24:00 by 30 minutes)
head(names(EUNITE.Loads))

# Plot a single half-hour interval across days
ts.plot(EUNITE.Loads[["X24.00"]], ylab = "Load (MW)", xlab = "Day",
        main = "EUNITE: Half-hour interval 24:00")
```

EUNITE.Reg *EUNITE Competition – Regressors for Load Forecasting*

Description

Daily holiday and weekday indicators used as regressors in the EUNITE load forecasting competition. Data Type: Categorical indicators. Category: Benchmark. Observations: 730 (1997–1998). This dataset provides binary holiday flags and weekday identifiers to support the prediction of daily maximum electrical loads. It complements the datasets `EUNITE.Loads` and `EUNITE.Temp`. A test set with corresponding regressors for January 1999 is available.

Usage

```
data(EUNITE.Reg)
```

Format

A data frame with 730 rows and 3 columns:

Holiday Binary indicator (1 = holiday, 0 = regular day).

Weekday Integer encoding (1 = Sunday, ..., 7 = Saturday).

split Split into train and test

Details

Regressors complement the load profiles by providing daily-level covariates (e.g., holidays and weekdays), which are known to improve forecast accuracy when used with temperature.

Source

EUNITE Competition 2001 dataset (original competition website currently unavailable).

References

Chen, B.-J., Chang, M.-W., & Lin, C.-J. (2004). *Load forecasting using support vector machines: a study on EUNITE competition 2001.* IEEE Transactions on Power Systems, 19(4), 1821-1830.

Examples

```
# Load EUNITE regressors
data(EUNITE.Reg)
# EUNITE.Reg <- loadfulldata(EUNITE.Reg)

# Peek at the first rows
head(EUNITE.Reg)
```

EUNITE.Temp *EUNITE Competition – Average Daily Temperatures*

Description

Average daily temperatures collected for the EUNITE load-forecasting competition. Data Type: Meteorological measurements. Category: Benchmark. Observations: 1,461. The series covers 1995-1998 and was used as an exogenous regressor for predicting maximum daily electrical loads. Participants were asked to forecast January 1999 values.

Usage

```
data(EUNITE.Temp)
```

Format

A data frame with one numeric column and 1,461 rows (average daily temperature).

Details

Daily temperatures are commonly used as exogenous variables for load forecasting due to strong weather dependence. This series aligns with the period covered by EUNITE.Loads.

Source

EUNITE Competition 2001 dataset (original competition website currently unavailable).

References

Chen, B.-J., Chang, M.-W., & Lin, C.-J. (2004). *Load forecasting using support vector machines: a study on EUNITE competition 2001.* IEEE Transactions on Power Systems, 19(4), 1821-1830.

Examples

```
# Load daily temperature series
data(EUNITE.Temp)
# EUNITE.Temp <- loadfulldata(EUNITE.Temp)

# Plot temperature over time
ts.plot(EUNITE.Temp$Temperature, ylab = "Temperature (°C)", xlab = "Day",
        main = "EUNITE: Daily Temperature")
```

---

fertilizers *FAOSTAT Fertilizers by Nutrient*

Description

Statistics on agricultural use, production, and trade of chemical and mineral fertilizers. Data Type: Fertilizers use, production and trade. Category: Environment. Creation Date 2024.

Usage

```
data(fertilizers)
```

Format

A list of time series.

Source

FAOSTAT Fertilizers by Nutrient.

References

FAO, 2024. FAOSTAT: Fertilizers by Nutrient. FAO & UNSD (2017). System of Environmental-Economic Accounting for Agriculture, Forestry and Fisheries (SEEA AFF). UNSD (2017). Framework for the Development of Environment Statistics (FDES).

Examples

```
# Load fertilizers list and plot one series
data(fertilizers)
# fertilizers <- loadfulldata(fertilizers)
series <- fertilizers[[1]]
ts.plot(series, ylab = "tonnes", xlab = "Year", main = "Fertilizers example")
```

gdp *Gross Domestic Product and Agriculture Value Added*

Description

Summary of global and regional trends in GDP and agriculture value. Data Type: macroeconomic indicators. Category: Economy. Creation Date 2024.

Usage

```
data(gdp)
```

Format

list of time series.

Source

FAOSTAT Macro Indicators Database

References

FAO. 2024. Gross domestic product and agriculture value added 2013–2022 – Global and regional trends. FAOSTAT Analytical Briefs, No. 85. Rome. doi:10.4060/cd0763en

Examples

```
# Load GDP list and plot one series
data(gdp)
# gdp <- loadfulldata(gdp)
series <- gdp[[1]]
ts.plot(series, ylab = "US$", xlab = "Year", main = "GDP example")
```

ipeadata.d *Ipea Daily Macroeconomic Dataset*

Description

Daily economic time series from Ipea (Institute for Applied Economic Research, Brazil). Data Type: Macroeconomic indicators. Category: Public data. Observations: 901 to 8,154 per series, 12 series. This dataset contains the most requested time series provided by Ipea with daily frequency, including exchange rates, stock index, interest rates, imports and exports. The series span from 1962 to September 2017. Missing values were removed using `na.omit`. The last 30 observations are for test set.

Usage

```
data(ipeadata.d)
```

Format

A data frame with up to 8,154 rows and 12 columns. Each column corresponds to a different univariate daily time series.

Details

Contains daily macroeconomic indicators frequently used in empirical forecasting. Series are cleaned with `na.omit`.

Source

Ipea - Ipeadata Portal, section "Most Requested Series", filtered by frequency "Daily".

References

Ipea (2017). *Ipeadata – Macroeconomic and Regional Data*. Technical Report. http://www.ipeadata.gov.br

Examples

```
# Load Ipea daily dataset and plot the first series
data(ipeadata.d)
# ipeadata.d <- loadfulldata(ipeadata.d)
series <- ipeadata.d[[1]]
ts.plot(series, ylab = "Value", xlab = "Day", main = "Ipea daily example")
```

---

ipeadata.m *Ipea Monthly Macroeconomic Dataset*

Description

Monthly economic time series from Ipea (Institute for Applied Economic Research, Brazil). Data Type: Macroeconomic indicators. Category: Public data. Observations: 156 to 1019 per series, 23 series. This dataset contains the most requested time series provided by Ipea, including exchange rates, inflation indices, unemployment rates, interest rates, minimum wage, and GDP. The series span from 1930 to September 2017. Missing values were removed using `na.omit`. The last 12 observations are for testing set.

Usage

```
data(ipeadata.m)
```

Format

A data frame with up to 1019 rows and 23 columns. Each column corresponds to a different univariate monthly time series.

Details

Contains monthly macroeconomic indicators; the last 12 observations are intended as a test set.

Source

Ipea - Ipeadata Portal, section "Most Requested Series", filtered by frequency "Monthly".

References

Ipea (2017). *Ipeadata – Macroeconomic and Regional Data*. Technical Report. http://www.ipeadata.gov.br

Examples

```
# Load Ipea monthly dataset and plot the first series
data(ipeadata.m)
# ipeadata.m <- loadfulldata(ipeadata.m)
series <- ipeadata.m[[1]]
ts.plot(series, ylab = "Value", xlab = "Month", main = "Ipea monthly example")
```

---

loadfulldata *Load Full Dataset From Mini Data Object*

Description

Downloads and loads the full `.RData` object referenced by `attr(x, "url")` from a mini dataset object loaded from `data/`.

Usage

```
loadfulldata(x)
```

Arguments

`x`	A mini dataset object that contains `attr(x, "url")`.

Value

The full dataset object loaded from the remote `.RData` file.

m1 *M1 Competition Time Series*

Description

Time series data from the first Makridakis forecasting competition (M1), held in 1982. Data Type: Forecasting benchmark dataset. Category: Forecasting. Creation Date: 1982.

Usage

```
data(m1)
```

Format

A list of dataframes containing time series.

Details

Consolidated list with frequencies as keys (e.g., `monthly`, `quarterly`, `yearly`). Each element is a list of series. See Makridakis et al. (1982) for competition design and evaluation.

Source

The accuracy of extrapolation (time series) methods: Results of a forecasting competition

References

Makridakis et al. (1982). The accuracy of extrapolation (time series) methods: Results of a forecasting competition. Journal of Forecasting, 1(2), 111–153.

Examples

```
# Load consolidated M1 list
data(m1)
# m1 <- loadfulldata(m1)

# List available frequency keys
names(m1)

# Plot one series from a frequency bucket
series <- m1$monthly[[1]]
ts.plot(series, main = "M1 monthly series")
```

m3 *M3 Competition Time Series*

Description

Time series data from the third Makridakis forecasting competition (M3), held in 2000. Data Type: Forecasting benchmark dataset. Category: Forecasting. Creation Date: 2000.

Usage

```
data(m3)
```

Format

A list of lists containing time series.

Details

Consolidated list keyed by frequency (e.g., `monthly`, `other`, `quarterly`, `yearly`). Each holds a list of numeric vectors. See Makridakis & Hibon (2000) for competition results and implications.

Source

doi:10.1016/S01692070(00)000571

References

Makridakis and Hibon (2000). The M3-Competition: Results, conclusions and implications. International Journal of Forecasting, 16(4), 451–476.

Examples

```
# Load consolidated M3 list and plot one monthly series
data(m3)
# m3 <- loadfulldata(m3)
series <- m3$monthly$M1
ts.plot(series, main = "M3 monthly series: M1")
```

m4 *M4 Competition Time Series*

Description

Time series data from the fourth Makridakis forecasting competition (M4), held in 2018. Data Type: Forecasting benchmark dataset. Category: Forecasting. Creation Date: 2018.

Usage

```
data(m4)
```

Format

A list of lists containing time series.

Details

Consolidated list keyed by frequency (e.g., `daily`, `hourly`, `monthly`, ...). Each holds a list of numeric vectors. See Makridakis et al. (2020) for an overview of M4 findings.

Source

M4 Competition - GitHub

References

Makridakis et al. (2020). The M4 Competition: Results, findings, conclusion and way forward. International Journal of Forecasting, 36(1), 54–74.

Examples

```
# Load consolidated M4 list and plot one available series
data(m4)
# m4 <- loadfulldata(m4)
freq_name <- names(m4)[1]
series_name <- names(m4[[freq_name]])[1]
series <- m4[[freq_name]][[series_name]]
ts.plot(series, main = paste("M4", freq_name, "series:", series_name))
```

MSE.ts *MSE*

Description

Compute mean squared error (MSE) between actual and predicted values.

Usage

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

Arguments

actual	Numeric vector of observed values.
prediction	Numeric vector of predicted values.

Details

MSE = mean((actual - prediction)^2).

Value

Numeric scalar with the MSE.

NN3 *NN3 Time Series Competition - Dataset A*

Description

Monthly time series from the NN3 forecasting competition. Data Type: Empirical business time series. Category: Benchmark. Observations: 50 to 126 per series, 111 series. The dataset contains 111 univariate monthly time series from real business processes. Each series has between 50 and 126 observations. Participants were asked to forecast the next 18 values, and performance was evaluated using the mean sMAPE across all series.

Usage

```
data(NN3)
```

Format

A data frame with up to 126 rows and 111 columns. Each column corresponds to a different univariate monthly time series.

Details

NN3 comprises monthly business time series with varying lengths. Forecast accuracy is typically evaluated using sMAPE across a fixed holdout horizon.

Source

NN3 Time Series Forecasting Competition

References

Crone, S.F., Hibon, M., & Nikolopoulos, K. (2011). *Advances in forecasting with neural networks? Empirical evidence from the NN3 competition on time series prediction*. International Journal of Forecasting, 27(3), 635–660. NN3 Competition (2007). http://www.neural-forecasting-competition.com/NN3/index.htm

Examples

```
# Load NN3 dataset
data(NN3)
# NN3 <- loadfulldata(NN3)

# Select one series by name and plot
series <- NN3[["NN3_111"]]
ts.plot(series, ylab = "Value", xlab = "Month", main = "NN3 example series")
```

NN5 *NN5 Time Series Competition*

Description

Daily time series from the NN5 forecasting competition. Data Type: ATM withdrawal amounts. Category: Benchmark. Observations: 735 per series, 111 series. The dataset contains 111 univariate time series representing daily cash withdrawals from ATMs in England. Each series includes 735 observations and may contain missing values and multiple seasonal patterns. Participants were asked to forecast the next 56 values for each series, and performance was evaluated using the mean sMAPE across all series.

Usage

```
data(NN5)
```

Format

A data frame with 735 rows and 111 columns. Each column corresponds to a different univariate daily time series.

Details

NN5 consists of daily ATM withdrawal amounts with complex multiple seasonalities and occasional missing values. Forecasts are evaluated via sMAPE on a 56-day horizon.

Source

NN5 Time Series Forecasting Competition

References

Crone, S.F. (2008). *Results of the NN5 Time Series Forecasting Competition*. IEEE WCCI 2008, Hong Kong. NN5 Competition (2008). http://www.neural-forecasting-competition.com/NN5/index.htm

Examples

```
# Load NN5 dataset
data(NN5)
# NN5 <- loadfulldata(NN5)

# Select one series and plot
series <- NN5[["NN5.111"]]
ts.plot(series, ylab = "Withdrawals", xlab = "Day", main = "NN5 example series")
```

---

pesticides *Pesticides Use Statistics*

Description

Statistics on the use of major pesticide groups and relevant chemical families. Data Type: pesticides use. Category: Environments. Creation Date 2024.

Usage

```
data(pesticides)
```

Format

A list of time series.

Details

Series are named by country with `_pesticides` suffix; values are annual usage amounts.

Source

Pesticides Use Database

References

FAO. 2024. FAOSTAT: Pesticides Use. RP_e_README_Domain_Information_2024. FAOSTAT Pesticides Use Database

Examples

```
# Load pesticides list and plot one series
data(pesticides)
# pesticides <- loadfulldata(pesticides)
series <- pesticides[[1]]
ts.plot(series, ylab = "tonnes", xlab = "Year", main = "Pesticides example")
```

`R2.ts` *R2*

Description

Compute coefficient of determination (R-squared).

Usage

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

Arguments

`actual`	Numeric vector of observed values.
`prediction`	Numeric vector of predicted values.

Value

Numeric scalar with R-squared.

`SantaFe.A` *Santa Fe Time Series Competition - Series A*

Description

Univariate time series A from the Santa Fe Time Series Competition. Data Type: Laser-generated nonlinear time series. Category: Benchmark. Observations: 1,100. This benchmark dataset consists of a low-dimensional nonlinear and stationary series recorded from a Far-Infrared-Laser in a chaotic regime. Competitors were asked to predict the last 100 observations, and performance was evaluated using NMSE.

Usage

```
data(SantaFe.A)
```

Format

A data frame with one column and 1,100 rows, containing numeric time series values.

Details

Series A is a classic nonlinear laser dataset used to assess forecasting methods under chaotic dynamics.

Source

Santa Fe Time Series Competition dataset (original archive URL unavailable).

References

Weigend, A.S. (1993). *Time Series Prediction: Forecasting the Future and Understanding the Past*. Reading, MA: Westview Press.

Examples

```
# Load Santa Fe A series and plot
data(SantaFe.A)
# SantaFe.A <- loadfulldata(SantaFe.A)
series <- SantaFe.A$V1
ts.plot(series, ylab = "Value", xlab = "Index", main = "Santa Fe A")
```

SantaFe.D *Santa Fe Time Series Competition - Series D*

Description

Univariate time series D from the Santa Fe Time Series Competition. Data Type: Simulated nonlinear time series. Category: Benchmark. Observations: 100,500. This benchmark dataset is composed of a four-dimensional nonlinear and non-stationary series. Competitors were asked to predict the last 500 observations, and performance was evaluated using NMSE.

Usage

```
data(SantaFe.D)
```

Format

A data frame with one column and 100,500 rows, containing numeric time series values.

Source

Santa Fe Time Series Competition dataset (original archive URL unavailable).

References

Weigend, A.S. (1993). *Time Series Prediction: Forecasting the Future and Understanding the Past*. Reading, MA: Westview Press.

Examples

```
# Load Santa Fe D series and plot a subset
data(SantaFe.D)
# SantaFe.D <- loadfulldata(SantaFe.D)
series <- SantaFe.D$V1
ts.plot(series[1:2000], ylab = "Value", xlab = "Index", main = "Santa Fe D (first 2000)")
```

---

select_hyper.ts_tune *Select Optimal Hyperparameters for Time Series Models*

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 symmetric mean absolute percent error (sMAPE).

Usage

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

Arguments

actual	Numeric vector of observed values.
prediction	Numeric vector of predicted values.

Details

sMAPE = mean(|a - p| / ((|a| + |p|)/2)), excluding zero denominators.

Value

Numeric scalar with the sMAPE.

References

- S. Makridakis and M. Hibon (2000). The M3-Competition: results, conclusions and implications. International Journal of Forecasting, 16(4).

stocks *IBOVESPA's 50 Most Traded Stocks*

Description

Historical daily data for the 50 most traded stocks in B3 (IBOVESPA), including opening, high, low, and closing prices, as well as trading volume. Data Type: Financial Time Series. Category: Finance. Creation Date: 2025.

Usage

```
data(stocks)
```

Format

A list of dataframes containing time series.

Details

Each entry is a data frame with columns `date`, `open`, `high`, `low`, `close`, and `volume`.

Source

B3

References

B3 - Brasil, Bolsa, Balcão. 2025. Historical stock trading data. B3 Official Website

Examples

```
# Load stocks list and plot closing prices for a ticker (if present)
data(stocks)
# stocks <- loadfulldata(stocks)
if ("VALE3" %in% names(stocks)) {
  series <- stocks$VALE3$close
  ts.plot(series, ylab = "Close", xlab = "Index", main = "VALE3 close price")
}
```

---

tsd *Time series example dataset*

Description

Synthetic dataset based on a 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

```
# Load dataset and preview the first rows
data(tsd)
head(tsd)
```

ts_arima *ARIMA*

Description

Create a time series prediction object based on the AutoRegressive Integrated Moving Average (ARIMA) family.

This constructor sets up an S3 time series regressor that leverages the `forecast` package to automatically select orders via `auto.arima` and provide one-step and multi-step forecasts.

Usage

```
ts_arima()
```

Details

ARIMA models combine autoregressive (AR), differencing (I), and moving average (MA) components to model temporal dependence in a univariate time series. The `fit()` method uses `forecast::auto.arima()` to select orders using information criteria, and `predict()` supports both a single one-step-ahead over a horizon (rolling) and direct multi-step forecasting.

Assumptions include (after differencing) approximate stationarity and homoskedastic residuals. Always inspect residual diagnostics for adequacy.

Value

A `ts_arima` object (S3), which inherits from `ts_reg`.

References

- G. E. P. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung (2015). Time Series Analysis: Forecasting and Control. Wiley.
- R. J. Hyndman and Y. Khandakar (2008). Automatic time series forecasting: The forecast package for R. Journal of Statistical Software, 27(3), 1–22. doi:10.18637/jss.v027.i03

Examples

```
# Example: rolling-origin evaluation with multi-step prediction
# Load package and dataset
library(daltoolbox)
data(tsd)

# 1) Wrap the raw vector as `ts_data` without sliding windows
ts <- ts_data(tsd$y, 0)
ts_head(ts, 3)

# 2) Split into train/test using the last 5 observations as test
samp <- ts_sample(ts, test_size = 5)
io_train <- ts_projection(samp$train)
```

```
io_test <- ts_projection(samp$test)

# 3) Fit ARIMA via auto.arima
model <- ts_arima()
model <- fit(model, x = io_train$input, y = io_train$output)

# 4) Predict 5 steps ahead from the most recent observed point
prediction <- predict(model, x = io_test$input[1,], steps_ahead = 5)
prediction <- as.vector(prediction)
output <- as.vector(io_test$output)

# 5) Evaluate forecast accuracy
ev_test <- evaluate(model, output, prediction)
ev_test
```

`ts_aug_awareness` *Augmentation by Awareness*

Description

Bias the augmentation to emphasize more recent points in each window (recency awareness), increasing their contribution to the augmented sample.

Usage

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

Arguments

`factor`	Numeric factor controlling the recency weighting.

Value

A `ts_aug_awareness` object.

References

- Q. Wen et al. (2021). Time Series Data Augmentation for Deep Learning: A Survey. IJCAI Workshop on Time Series.

Examples

```
# Recency-aware augmentation over sliding windows
# Load package and example dataset
library(daltoolbox)
data(tsd)

# Convert to 10-lag sliding windows and preview
xw <- ts_data(tsd$y, 10)
ts_head(xw)
```

```
# Apply awareness augmentation (bias toward recent rows)
augment <- ts_aug_awareness()
augment <- fit(augment, xw)
xa <- transform(augment, xw)
ts_head(xa)
```

`ts_aug_awaresmooth` *Augmentation by Awareness Smooth*

Description

Recency-aware augmentation that also progressively smooths noise before applying the weighting, producing cleaner augmented samples.

Usage

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

Arguments

`factor`	Numeric factor controlling the recency weighting.

Value

A `ts_aug_awaresmooth` object.

References

- Q. Wen et al. (2021). Time Series Data Augmentation for Deep Learning: A Survey. IJCAI Workshop on Time Series.

Examples

```
# Recency-aware augmentation with progressive smoothing
# Load package and example dataset
library(daltoolbox)
data(tsd)

# Convert to 10-lag sliding windows and preview
xw <- ts_data(tsd$y, 10)
ts_head(xw)

# Apply awareness+smooth augmentation and inspect result
augment <- ts_aug_awaresmooth()
augment <- fit(augment, xw)
xa <- transform(augment, xw)
ts_head(xa)
```

ts_aug_flip *Augmentation by Flip*

Description

Time series augmentation by mirroring sliding-window observations around their mean to increase diversity and reduce overfitting.

Usage

```
ts_aug_flip()
```

Details

This transformation preserves the window mean while flipping the deviations, effectively generating a symmetric variant of the local pattern.

Value

A `ts_aug_flip` object.

References

- Q. Wen et al. (2021). Time Series Data Augmentation for Deep Learning: A Survey. IJCAI Workshop on Time Series.

Examples

```
# Flip augmentation around the window mean
# Load package and example dataset
library(daltoolbox)
data(tsd)

# Convert to sliding windows and preview
xw <- ts_data(tsd$y, 10)
ts_head(xw)

# Apply flip augmentation and inspect augmented windows
augment <- ts_aug_flip()
augment <- fit(augment, xw)
xa <- transform(augment, xw)
ts_head(xa)
```

ts_aug_jitter *Augmentation by Jitter*

Description

Time series augmentation by adding low-amplitude random noise to each point to increase robustness and reduce overfitting.

Usage

```
ts_aug_jitter()
```

Details

Noise scale is estimated from within-window deviations.

Value

A ts_aug_jitter object.

References

- J. T. Um et al. (2017). Data augmentation of wearable sensor data for Parkinson's disease monitoring using convolutional neural networks.
- Q. Wen et al. (2021). Time Series Data Augmentation for Deep Learning: A Survey. IJCAI Workshop on Time Series.

Examples

```
# Jitter augmentation with noise estimated from windows
# Load package and example dataset
library(daltoolbox)
data(tsd)

# Convert to sliding windows and preview
xw <- ts_data(tsd$y, 10)
ts_head(xw)

# Apply jitter (adds small noise; keeps target column unchanged)
augment <- ts_aug_jitter()
augment <- fit(augment, xw)
xa <- transform(augment, xw)
ts_head(xa)
```

ts_aug_none *No Augmentation*

Description

Identity augmentation that returns the original windows while preserving the augmentation interface and indices.

Usage

```
ts_aug_none()
```

Value

A ts_aug_none object.

Examples

```
# Identity augmentation (no changes to windows)
# Load package and example dataset
library(daltoolbox)
data(tsd)

# Convert to sliding windows and preview
xw <- ts_data(tsd$y, 10)
ts_head(xw)

# No augmentation; returns the same windows with indices preserved
augment <- ts_aug_none()
augment <- fit(augment, xw)
xa <- transform(augment, xw)
ts_head(xa)
```

ts_aug_shrink *Augmentation by Shrink*

Description

Decrease within-window deviation magnitude by a scaling factor to generate lower-variance variants while preserving the mean.

Usage

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

Arguments

scale_factor	Numeric factor used to scale deviations.

Value

A ts_aug_shrink object.

References

- Q. Wen et al. (2021). Time Series Data Augmentation for Deep Learning: A Survey. IJCAI Workshop on Time Series.

Examples

```
# Shrink augmentation reduces within-window deviations
# Load package and example dataset
library(daltoolbox)
data(tsd)

# Convert to sliding windows and preview
xw <- ts_data(tsd$y, 10)
ts_head(xw)

# Apply shrink augmentation and inspect augmented windows
augment <- ts_aug_shrink()
augment <- fit(augment, xw)
xa <- transform(augment, xw)
ts_head(xa)
```

---

ts_aug_stretch *Augmentation by Stretch*

Description

Increase within-window deviation magnitude by a scaling factor to produce higher-variance variants.

Usage

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

Arguments

scale_factor	Numeric factor used to scale deviations.

Value

A ts_aug_stretch object.

References

- Q. Wen et al. (2021). Time Series Data Augmentation for Deep Learning: A Survey. IJCAI Workshop on Time Series.

Examples

```
# Stretch augmentation increases within-window deviations
# Load package and example dataset
library(daltoolbox)
data(tsd)

# Convert to sliding windows and preview
xw <- ts_data(tsd$y, 10)
ts_head(xw)

# Apply stretch augmentation and inspect augmented windows
augment <- ts_aug_stretch()
augment <- fit(augment, xw)
xa <- transform(augment, xw)
ts_head(xa)
```

---

ts_aug_wormhole *Augmentation by Wormhole*

Description

Generate augmented windows by selectively replacing lag terms with older lagged values, creating plausible alternative trajectories.

Usage

```
ts_aug_wormhole()
```

Details

This combinatorial replacement preserves overall scale while introducing temporal permutations of lag content.

Value

A `ts_aug_wormhole` object.

References

- Q. Wen et al. (2021). Time Series Data Augmentation for Deep Learning: A Survey. IJCAI Workshop on Time Series.

Examples

```
# Wormhole augmentation replaces some lags with older values
# Load package and example dataset
library(daltoolbox)
data(tsd)
```

```
# Convert to sliding windows and preview
xw <- ts_data(tsd$y, 10)
ts_head(xw)

# Apply wormhole augmentation and inspect augmented windows
augment <- ts_aug_wormhole()
augment <- fit(augment, xw)
xa <- transform(augment, xw)
ts_head(xa)
```

`ts_data` *ts_data*

Description

Construct a time series data object used throughout the DAL Toolbox.

Accepts either a vector (raw time series) or a matrix/data.frame already organized in sliding windows. Internally, a `ts_data` is stored as a matrix with `sw` lag columns named `t{lag}` (e.g., `t9, t8, ..., t0`). When `sw` is zero or one, the series is stored as a single column (`t0`).

Usage

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

Arguments

`y`	Numeric vector or matrix-like. Time series values or sliding windows.
`sw`	Integer. Sliding-window size (number of lag columns).

Value

A `ts_data` object (matrix with attributes and column names).

Examples

```
# Example: building sliding windows
data(tsd)
head(tsd)

# 1) Single-column ts_data (no windows)
data <- ts_data(tsd$y)
ts_head(data)

# 2) 10-lag sliding windows (t9 ... t0)
data10 <- ts_data(tsd$y, 10)
ts_head(data10)
```

ts_elm *ELM*

Description

Create a time series prediction object that uses Extreme Learning Machine (ELM) regression.

It wraps the `elmNNRcpp` package to train single-hidden-layer networks with randomly initialized hidden weights and closed-form output weights.

Usage

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

Arguments

`preprocess`	Normalization preprocessor (e.g., `ts_norm_gminmax()`).
`input_size`	Integer. Number of lagged inputs used by the model.
`nhid`	Integer. Hidden layer size.
`actfun`	Character. One of 'sig', 'radbas', 'tribas', 'relu', 'purelin'.

Details

ELMs are efficient to train and can perform well with appropriate hidden size and activation choice. Consider normalizing inputs and tuning `nhid` and the activation function.

Value

A `ts_elm` object (S3) inheriting from `ts_regsw`.

References

- G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew (2006). Extreme Learning Machine: Theory and Applications. Neurocomputing, 70(1–3), 489–501.

Examples

```
# Example: ELM with sliding-window inputs
# Load package and toy dataset
library(daltoolbox)
data(tsd)

# Create sliding windows of length 10 (t9 ... t0)
ts <- ts_data(tsd$y, 10)
ts_head(ts, 3)

# Split last 5 rows as test set
samp <- ts_sample(ts, test_size = 5)
# Project to inputs (X) and outputs (y)
```

```
io_train <- ts_projection(samp$train)
io_test <- ts_projection(samp$test)

# Define ELM with global min-max normalization and fit
model <- ts_elm(ts_norm_gminmax(), input_size = 4, nhid = 3, actfun = "purelin")
model <- fit(model, x = io_train$input, y = io_train$output)

# Forecast 5 steps ahead starting from the last known window
prediction <- predict(model, x = io_test$input[1,], steps_ahead = 5)
prediction <- as.vector(prediction)
output <- as.vector(io_test$output)

# Evaluate forecast error on the test horizon
ev_test <- evaluate(model, output, prediction)
ev_test
```

`ts_fil_ema` *Exponential Moving Average (EMA)*

Description

Smooth a series by exponentially decaying weights that give more importance to recent observations.

Usage

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

Arguments

`ema`	exponential moving average size

Details

EMA is related to simple exponential smoothing; it reacts faster to level changes than a simple moving average while reducing noise.

Value

A `ts_fil_ema` object.

References

- C. C. Holt (1957). Forecasting trends and seasonals by exponentially weighted moving averages. O.N.R. Research Memorandum.

Examples

```
# Exponential moving average smoothing on a noisy series
# Load package and example data
library(daltoolbox)
data(tsd)

# Inject an outlier to illustrate smoothing effect
tsd$y[9] <- 2 * tsd$y[9]

# Define EMA filter, fit and transform the series
filter <- ts_fil_ema(ema = 3)
filter <- fit(filter, tsd$y)
y <- transform(filter, tsd$y)

# Compare original vs smoothed series
plot_ts_pred(y = tsd$y, yadj = y)
```

---

ts_fil_emd *EMD Filter*

Description

Empirical Mode Decomposition (EMD) filter that decomposes a signal into intrinsic mode functions (IMFs) and reconstructs a smoothed component.

Usage

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

Arguments

noise	noise
trials	trials

Value

A ts_fil_emd object.

References

- N. E. Huang et al. (1998). The Empirical Mode Decomposition and the Hilbert Spectrum for nonlinear and non-stationary time series analysis. Proceedings of the Royal Society A.

Examples

```
# EMD-based smoothing: remove first IMF as noise
# Load package and example data
library(daltoolbox)
data(tsd)
tsd$y[9] <- 2 * tsd$y[9]  # inject an outlier

# Fit EMD filter and reconstruct without the first (noisiest) IMF
filter <- ts_fil_emd()
filter <- fit(filter, tsd$y)
y <- transform(filter, tsd$y)

# Compare original vs smoothed series
plot_ts_pred(y = tsd$y, yadj = y)
```

---

ts_fil_fft *FFT Filter*

Description

Frequency-domain smoothing using the Fast Fourier Transform (FFT) to attenuate high-frequency components.

Usage

```
ts_fil_fft()
```

Details

The implementation estimates a cutoff based on spectral statistics and reconstructs the series from dominant frequencies.

Value

A `ts_fil_fft` object.

References

- J. W. Cooley and J. W. Tukey (1965). An algorithm for the machine calculation of complex Fourier series. Math. Comput.

Examples

```
# Frequency-domain smoothing via FFT cutoff
# Load package and example data
library(daltoolbox)
data(tsd)
tsd$y[9] <- 2 * tsd$y[9]  # inject an outlier
```

```
# Fit FFT-based filter and reconstruct without high frequencies
filter <- ts_fil_fft()
filter <- fit(filter, tsd$y)
y <- transform(filter, tsd$y)

# Compare original vs frequency-smoothed series
plot_ts_pred(y = tsd$y, yadj = y)
```

`ts_fil_hp` *Hodrick-Prescott Filter*

Description

Decompose a series into trend and cyclical components using the Hodrick–Prescott (HP) filter and optionally blend with the original series.

This filter removes short-term fluctuations by penalizing changes in the growth rate of the trend 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.

Details

The filter strength is governed by `lambda = 100 * frequency^2`. Use `preserve` in (0, 1] to convex-combine the raw series and the HP trend.

Value

A `ts_fil_hp` object.

References

- R. J. Hodrick and E. C. Prescott (1997). Postwar U.S. business cycles: An empirical investigation. Journal of Money, Credit and Banking, 29(1).

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

Estimate a latent trend via a state-space model using the Kalman Filter (KF), wrapping the `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.

References

- R. E. Kalman (1960). A new approach to linear filtering and prediction problems. Journal of Basic Engineering, 82(1), 35–45.

Examples

```
# State-space smoothing with Kalman Filter (KF)
# Load package and example data
library(daltoolbox)
data(tsd)
tsd$y[9] <- 2 * tsd$y[9]  # inject an outlier

# Fit KF (H = obs noise, Q = process noise) and transform
filter <- ts_fil_kalman()
filter <- fit(filter, tsd$y)
y <- transform(filter, tsd$y)

# Plot original vs KF-smoothed series
plot_ts_pred(y = tsd$y, yadj = y)
```

---

ts_fil_lowess *LOWESS Smoothing*

Description

Locally Weighted Scatterplot Smoothing (LOWESS) fits local regressions to capture the primary trend while reducing noise and spikes.

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.

References

- W. S. Cleveland (1979). Robust locally weighted regression and smoothing scatterplots. Journal of the American Statistical Association.

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` *Moving Average (MA)*

Description

Smooth out fluctuations and reduce noise by averaging over a fixed-size rolling window.

Usage

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

Arguments

`ma`	moving average size

Details

Larger windows produce smoother series but may lag turning points.

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 *No Filter*

Description

Identity filter that returns the original series unchanged.

Usage

```
ts_fil_none()
```

Value

A ts_fil_none object.

Examples

```
# Identity filter (returns original series)
# Load package and example series
library(daltoolbox)
data(tsd)
tsd$y[9] <- 2 * tsd$y[9]  # inject an outlier for comparison

# Fit identity filter and transform (no change expected)
filter <- ts_fil_none()
filter <- fit(filter, tsd$y)
y <- transform(filter, tsd$y)

# Plot original vs (identical) filtered series
plot_ts_pred(y = tsd$y, yadj = y)
```

ts_fil_qes *Quadratic Exponential Smoothing*

Description

Double/triple exponential smoothing capturing level, trend, and optionally seasonality components.

Usage

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

Arguments

gamma	If TRUE, enables the gamma seasonality component.

Value

A ts_fil_qes object.

References

- P. R. Winters (1960). Forecasting sales by exponentially weighted moving averages. Management Science.

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 *Recursive Filter*

Description

Apply recursive linear filtering (ARMA-style recursion) to a univariate series or each column of a multivariate series. Useful for smoothing and mitigating autocorrelation.

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 *Robust EMD Filter*

Description

Ensemble/robust EMD-based denoising using CEEMD to separate noise-dominated IMFs and reconstruct the signal.

Usage

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

Arguments

noise	noise
trials	trials

Value

A ts_fil_remd object.

References

- Z. Wu and N. E. Huang (2009). Ensemble Empirical Mode Decomposition: a noise-assisted data analysis method. Advances in Adaptive Data Analysis.

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` *Seasonal Adjustment*

Description

Remove the seasonal component from a time series while preserving level and trend, using a state-space/BATS approach.

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.

References

- R. J. Hyndman and G. Athanasopoulos (2021). Forecasting: Principles and Practice (3rd ed). OTexts. (BATS/seasonal adjustment)

Examples

```
# Seasonal adjustment using BATS at known frequency
# Load package and example data
library(daltoolbox)
data(tsd)
tsd$y[9] <- 2 * tsd$y[9]  # inject an outlier (illustrative)

# Fit seasonal adjustment (set frequency if known) and transform
filter <- ts_fil_seas_adj(frequency = 26)
filter <- fit(filter, tsd$y)
y <- transform(filter, tsd$y)

# Plot original vs seasonally adjusted series
plot_ts_pred(y = tsd$y, yadj = y)
```

ts_fil_ses *Simple Exponential Smoothing*

Description

Exponential smoothing focused on the level component, with optional extensions to trend/seasonality via Holt–Winters variants.

Usage

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

Arguments

gamma	If TRUE, enables the gamma seasonality component.

Value

A ts_fil_ses object.

References

- R. G. Brown (1959). Statistical Forecasting for Inventory Control.

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

Remove or reduce randomness (noise) using a robust smoothing strategy that first mitigates outliers and then smooths residual variation.

Usage

```
ts_fil_smooth()
```

Value

A ts_fil_smooth object.

Examples

```
# Robust smoothing with iterative outlier mitigation
# Load package and example data
library(daltoolbox)
data(tsd)
tsd$y[9] <- 2 * tsd$y[9]  # inject an outlier

# Fit smoother and transform to reduce spikes/noise
filter <- ts_fil_smooth()
filter <- fit(filter, tsd$y)
y <- transform(filter, tsd$y)

# Compare original vs smoothed series
plot_ts_pred(y = tsd$y, yadj = y)
```

ts_fil_spline *Smoothing Splines*

Description

Fit a cubic smoothing spline to a time series for smooth trend extraction with a tunable roughness penalty.

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.

Value

A ts_fil_spline object.

References

- P. Craven and G. Wahba (1978). Smoothing noisy data with spline functions. Numerische Mathematik.

Examples

```
# Smoothing splines with adjustable roughness penalty
# Load package and example data
library(daltoolbox)
data(tsd)
tsd$y[9] <- 2 * tsd$y[9]  # inject an outlier

# Fit spline smoother (spar controls smoothness) and transform
filter <- ts_fil_spline(spar = 0.5)
filter <- fit(filter, tsd$y)
y <- transform(filter, tsd$y)

# Compare original vs smoothed series
plot_ts_pred(y = tsd$y, yadj = y)
```

---

ts_fil_wavelet *Wavelet Filter*

Description

Denoise a series using discrete wavelet transforms and selected wavelet families.

Usage

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

Arguments

filter	Available wavelet filters: 'haar', 'd4', 'la8', 'bl14', 'c6'.

Value

A ts_fil_wavelet object.

References

- S. Mallat (1989). A Theory for Multiresolution Signal Decomposition: The Wavelet Representation. IEEE Transactions on Pattern Analysis and Machine Intelligence.

Examples

```
# Denoising with discrete wavelets (optionally selecting best filter)
# Load package and example data
library(daltoolbox)
data(tsd)
tsd$y[9] <- 2 * tsd$y[9]  # inject an outlier

# Fit wavelet filter ("haar" by default; can pass a list to select best)
filter <- ts_fil_wavelet()
filter <- fit(filter, tsd$y)
y <- transform(filter, tsd$y)

# Compare original vs wavelet-denoised series
plot_ts_pred(y = tsd$y, yadj = y)
```

---

ts_fil_winsor *Winsorization of Time Series*

Description

Apply Winsorization to limit extreme values by replacing them with nearer order statistics, reducing the influence of outliers.

Usage

```
ts_fil_winsor()
```

Value

A `ts_fil_winsor` object.

References

- J. W. Tukey (1962). The future of data analysis. Annals of Mathematical Statistics. (Winsorization discussed in robust summaries.)

Examples

```
# Winsorization: cap extreme values to reduce outlier impact
# Load package and example data
library(daltoolbox)
data(tsd)
tsd$y[9] <- 2 * tsd$y[9]  # inject an outlier
```

```
# Fit Winsor filter and transform series
filter <- ts_fil_winsor()
filter <- fit(filter, tsd$y)
y <- transform(filter, tsd$y)

# Plot original vs Winsorized series
plot_ts_pred(y = tsd$y, yadj = y)
```

`ts_head` *Extract the First Observations from a* `ts_data` *Object*

Description

Return 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

The first n observations of a `ts_data` (as a matrix/data.frame).

Examples

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

`ts_integtune` *Time Series Integrated Tune*

Description

Integrated tuning over input sizes, preprocessing, augmentation, and model hyperparameters for time series.

Usage

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

Arguments

input_size	Integer vector. Candidate input window sizes.
base_model	Base model object for tuning.
folds	Integer. Number of cross-validation folds.
ranges	Named list of hyperparameter ranges to explore.
preprocess	List of preprocessing objects to compare.
augment	List of augmentation objects to apply during training.

Value

A ts_integtune object.

References

Salles, R., Pacitti, E., Bezerra, E., Marques, C., Pacheco, C., Oliveira, C., Porto, F., Ogasawara, E. (2023). TSPredIT: Integrated Tuning of Data Preprocessing and Time Series Prediction Models. Lecture Notes in Computer Science.

Examples

```
# Integrated search over input size, preprocessing and model hyperparameters
library(daltoolbox)
data(tsd)

# Build windows and split into train/test, then project to (X, y)
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)

# Configure integrated tuning: ranges for input_size, ELM (nhid, actfun), and preprocessors
tune <- ts_integtune(
  input_size = 3:5,
  base_model = ts_elm(),
  ranges = list(nhid = 1:5, actfun = c('purelin')),
  preprocess = list(ts_norm_gminmax())
)
```

```
# Run search; augmentation (if provided) is applied during training internally
model <- fit(tune, x = io_train$input, y = io_train$output)

# Forecast and evaluate on the held-out window
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

Create a prediction object that uses the K-Nearest Neighbors regression for time series via sliding windows.

Usage

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

Arguments

preprocess	Normalization preprocessor (e.g., ts_norm_gminmax()).
input_size	Integer. Number of lagged inputs.
k	Integer. Number of neighbors.

Details

KNN regression predicts a value as the average (or weighted average) of the outputs of the k most similar windows in the training set. Similarity is computed in the feature space induced by lagged inputs. Consider normalization for distance-based methods.

Value

A ts_knn object (S3) inheriting from ts_regsw.

References

- T. M. Cover and P. E. Hart (1967). Nearest neighbor pattern classification. IEEE Transactions on Information Theory, 13(1), 21–27.

Examples

```
# Example: distance-based regression on sliding windows
# Load tools and example series
library(daltoolbox)
data(tsd)

# Build 10-lag windows and preview a few rows
ts <- ts_data(tsd$y, 10)
ts_head(ts, 3)

# Split end of series as test and project (X, y)
samp <- ts_sample(ts, test_size = 5)
io_train <- ts_projection(samp$train)
io_test <- ts_projection(samp$test)

# Define KNN regressor and fit (distance-based; normalization recommended)
model <- ts_knn(ts_norm_gminmax(), input_size = 4, k = 3)
model <- fit(model, x = io_train$input, y = io_train$output)

# Predict multiple steps ahead and evaluate
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

Create a time series prediction object based on a Multilayer Perceptron (MLP) regressor.

It wraps the `nnet` package to train a single-hidden-layer neural network on sliding-window inputs. Use `ts_regsw` utilities to project inputs/outputs.

Usage

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

Arguments

`preprocess`	Normalization preprocessor (e.g., `ts_norm_gminmax()`).
`input_size`	Integer. Number of lagged inputs used by the model.
`size`	Integer. Number of hidden neurons.
`decay`	Numeric. L2 weight decay (regularization) parameter.
`maxit`	Integer. Maximum number of training iterations.

Details

The MLP is a universal function approximator capable of learning non-linear mappings from lagged inputs to next-step values. For stability, consider normalizing inputs (e.g., `ts_norm_gminmax()`). Hidden size and weight decay control capacity and regularization respectively.

Value

A `ts_mlp` object (S3) inheriting from `ts_regsw`.

References

- D. E. Rumelhart, G. E. Hinton, and R. J. Williams (1986). Learning representations by back-propagating errors. Nature 323, 533–536.
- W. N. Venables and B. D. Ripley (2002). Modern Applied Statistics with S. Fourth Edition. Springer. (for the `nnet` package)

Examples

```
# Example: MLP on sliding windows with min-max normalization
# Load package and dataset
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)

# Prepare projection (X, y)
samp <- ts_sample(ts, test_size = 5)
io_train <- ts_projection(samp$train)
io_test <- ts_projection(samp$test)

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

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

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

ts_norm_an *Adaptive Normalization*

Description

Transform data to a common scale while adapting to changes in distribution over time (optionally over a trailing window).

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

A ts_norm_an object.

References

Ogasawara, E., Martinez, L. C., De Oliveira, D., Zimbrão, G., Pappa, G. L., Mattoso, M. (2010). Adaptive Normalization: A novel data normalization approach for non-stationary time series. Proceedings of the International Joint Conference on Neural Networks (IJCNN). doi:10.1109/IJCNN.2010.5596746

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 *First Differences*

Description

Transform a series by first differences to remove level and highlight changes; normalization is then applied to the differenced series.

Usage

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

Arguments

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

Value

A ts_norm_diff object.

References

Salles, R., Assis, L., Guedes, G., Bezerra, E., Porto, F., Ogasawara, E. (2017). A framework for benchmarking machine learning methods using linear models for univariate time series prediction. Proceedings of the International Joint Conference on Neural Networks (IJCNN). doi:10.1109/IJCNN.2017.7966139

Examples

```
# Differencing + global min-max normalization
# Load package and example data
library(daltoolbox)
data(tsd)

# Convert to sliding windows and preview raw last column
ts <- ts_data(tsd$y, 10)
ts_head(ts, 3)
summary(ts[,10])

# Fit differencing preprocessor and transform; note one fewer lag column
preproc <- ts_norm_diff()
preproc <- fit(preproc, ts)
tst <- transform(preproc, ts)
ts_head(tst, 3)
summary(tst[,9])
```

ts_norm_ean *Adaptive Normalization with EMA*

Description

Normalize a time series using exponentially weighted statistics that adapt to distributional changes, optionally after outlier mitigation.

Usage

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

Arguments

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

Value

A ts_norm_ean object.

References

Ogasawara, E., Martinez, L. C., De Oliveira, D., Zimbrão, G., Pappa, G. L., Mattoso, M. (2010). Adaptive Normalization: A novel data normalization approach for non-stationary time series. Proceedings of the International Joint Conference on Neural Networks (IJCNN). doi:10.1109/IJCNN.2010.5596746

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 *Global Min–Max Normalization*

Description

Rescale values so the global minimum maps to 0 and the global maximum maps to 1 over the training set.

Usage

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

Arguments

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

Details

The same scaling is applied to inputs and inverted on predictions via `inverse_transform`.

Value

A `ts_norm_gminmax` object.

References

Ogasawara, E., Murta, L., Zimbrão, G., Mattoso, M. (2009). Neural networks cartridges for data mining on time series. Proceedings of the International Joint Conference on Neural Networks (IJCNN). doi:10.1109/IJCNN.2009.5178615

Examples

```
# Global min-max normalization across the full training set
# Load package and example data
library(daltoolbox)
data(tsd)

# Build 10-lag windows and preview raw scale
ts <- ts_data(tsd$y, 10)
ts_head(ts, 3)
summary(ts[,10])

# Fit global min-max and transform; inspect post-scale values
preproc <- ts_norm_gminmax()
preproc <- fit(preproc, ts)
tst <- transform(preproc, ts)
ts_head(tst, 3)
summary(tst[,10])
```

ts_norm_none *No Normalization*

Description

Identity transform that leaves data unchanged but aligns with the pre/post-processing interface.

Usage

```
ts_norm_none()
```

Value

A ts_norm_none object.

Examples

```
# Identity normalization (no scaling applied)
# Load package and example data
library(daltoolbox)
data(tsd)

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

# No data normalization — transform returns inputs unchanged
normalize <- ts_norm_none()
normalize <- fit(normalize, xw)
xa <- transform(normalize, xw)
ts_head(xa)
```

ts_norm_swminmax *Sliding-Window Min–Max Normalization*

Description

Create an object for normalizing each window by its own min and max, preserving local contrast while standardizing scales.

Usage

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

Arguments

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

Value

A ts_norm_swminmax object.

References

Ogasawara, E., Murta, L., Zimbrão, G., Mattoso, M. (2009). Neural networks cartridges for data mining on time series. Proceedings of the International Joint Conference on Neural Networks (IJCNN). doi:10.1109/IJCNN.2009.5178615

Examples

```
# Per-window min-max normalization for sliding windows
# Load package and example data
library(daltoolbox)
data(tsd)

# Build 10-lag windows and preview raw scale
ts <- ts_data(tsd$y, 10)
ts_head(ts, 3)
summary(ts[,10])

# Fit per-window min-max and transform; inspect post-scale values
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

Split a ts_data (sliding windows) into input features and output targets for modeling.

Usage

```
ts_projection(ts)
```

Arguments

ts	Matrix or data.frame containing a ts_data representation.

Details

For a multi-column ts_data, returns all but the last column as inputs and the last column as the output. For a single-row matrix, returns ts_data-wrapped inputs/outputs preserving names and window size.

Value

A ts_projection object with two elements: $input and $output.

Examples

```
# Setting up a ts_data and projecting (X, y)
# Load example dataset and create windows
data(tsd)
ts <- ts_data(tsd$y, 10)

io <- ts_projection(ts)

# Input data (features)
ts_head(io$input)

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

---

ts_reg *TSReg*

Description

Base class for time series regression models that operate directly on time series (non-sliding-window specialization).

Usage

```
ts_reg()
```

Details

This class is intended to be subclassed by modeling backends that do not require the sliding-window interface. Methods such as fit(), predict(), and evaluate() dispatch on this class.

Value

A ts_reg object (S3) to be extended by concrete models.

Examples

```
# Abstract base class — instantiate concrete subclasses instead
# Examples: ts_mlp(), ts_rf(), ts_svm(), ts_arima()
```

ts_regsw *TSRegSW*

Description

Base class for time series regression models built on sliding-window representations.

Usage

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

Arguments

preprocess	Normalization preprocessor (e.g., ts_norm_gminmax()).
input_size	Integer. Number of lagged inputs per example.

Details

This class provides helpers to map ts_data matrices into the input window expected by ML backends and to apply pre/post processing (e.g., normalization) consistently during fit and predict.

Value

A ts_regsw object (S3) to be extended by concrete models.

Examples

```
# Abstract base class for sliding-window regressors
# Use concrete subclasses such as ts_mlp(), ts_rf(), ts_svm(), ts_elm()
```

ts_rf *Random Forest*

Description

Create a time series prediction object that uses Random Forest regression on sliding-window inputs.

It wraps the randomForest package to fit an ensemble of decision trees.

Usage

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

Arguments

preprocess	Normalization preprocessor (e.g., ts_norm_gminmax()).
input_size	Integer. Number of lagged inputs used by the model.
nodesize	Integer. Minimum terminal node size.
ntree	Integer. Number of trees in the forest.
mtry	Integer. Number of variables randomly sampled at each split.

Details

Random Forests reduce variance by averaging many decorrelated trees. For tabular sliding-window features, they can capture nonlinearities and interactions without heavy feature engineering. Consider normalizing inputs for comparability across windows and tuning mtry, ntree, and nodesize.

Value

A ts_rf object (S3) inheriting from ts_regsw.

References

- L. Breiman (2001). Random forests. Machine Learning, 45(1), 5–32.

Examples

```
# Example: sliding-window Random Forest
# Load tools and data
library(daltoolbox)
data(tsd)

# Turn series into 10-lag windows and preview
ts <- ts_data(tsd$y, 10)
ts_head(ts, 3)

# Train/test split and (X, y) projection
samp <- ts_sample(ts, test_size = 5)
io_train <- ts_projection(samp$train)
io_test <- ts_projection(samp$test)

# Define Random Forest and fit (tune ntree/mtry/nodesize as needed)
model <- ts_rf(ts_norm_gminmax(), input_size = 4, nodesize = 3, ntree = 50)
model <- fit(model, x = io_train$input, y = io_train$output)

# Forecast multiple steps and assess error
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

Split a `ts_data` into train and test sets.

Extracts `test_size` rows from the end (minus an optional `offset`) as the test set. The remaining initial rows form the training set. The `offset` is useful to reproduce experiments with different forecast origins.

Usage

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

Arguments

`ts`	A `ts_data` matrix.
`test_size`	Integer. Number of rows in the test split (default = 1).
`offset`	Integer. Offset from the end before the test split (default = 0).

Value

A list with `$train` and `$test` (both `ts_data`).

Examples

```
# Setting up a ts_data and making a temporal split
# Load example dataset and build windows
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

Create a time series prediction object that uses Support Vector Regression (SVR) on sliding-window inputs.

It wraps the `e1071` package to fit epsilon-insensitive regression with linear, radial, polynomial, or sigmoid kernels.

Usage

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

Arguments

`preprocess`	Normalization preprocessor (e.g., `ts_norm_gminmax()`).
`input_size`	Integer. Number of lagged inputs used by the model.
`kernel`	Character. One of 'linear', 'radial', 'polynomial', 'sigmoid'.
`epsilon`	Numeric. Epsilon-insensitive loss width.
`cost`	Numeric. Regularization parameter controlling margin violations.

Details

SVR aims to find a function with at most epsilon deviation from each training point while being as flat as possible. The `cost` parameter controls the trade-off between margin width and violations; `epsilon` controls the insensitivity tube width. RBF kernels often work well for nonlinear series; tune `cost`, `epsilon`, and kernel hyperparameters.

Value

A `ts_svm` object (S3) inheriting from `ts_regsw`.

References

- C. Cortes and V. Vapnik (1995). Support-Vector Networks. Machine Learning, 20, 273–297.

Examples

```
# Example: SVR with min-max normalization
# Load package and dataset
library(daltoolbox)
data(tsd)

# Create sliding windows and preview
ts <- ts_data(tsd$y, 10)
ts_head(ts, 3)

# Temporal split and (X, y) projection
samp <- ts_sample(ts, test_size = 5)
io_train <- ts_projection(samp$train)
io_test <- ts_projection(samp$test)

# Define SVM regressor and fit to training data
model <- ts_svm(ts_norm_gminmax(), input_size = 4)
model <- fit(model, x = io_train$input, y = io_train$output)

# Multi-step forecast and evaluation
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

Create a `ts_tune` object for hyperparameter tuning of a time series model.

Sets up a cross-validated search over hyperparameter ranges and input sizes for a base model. Results include the evaluated configurations and the selected best configuration.

Usage

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

Arguments

`input_size`	Integer vector. Candidate input window sizes.
`base_model`	Base model object to tune (e.g., `ts_mlp()`).
`folds`	Integer. Number of cross-validation folds.
`ranges`	Named list of hyperparameter ranges to explore.

Value

A ts_tune object.

References

- R. Kohavi (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. IJCAI.
- Salles, R., Pacitti, E., Bezerra, E., Marques, C., Pacheco, C., Oliveira, C., Porto, F., Ogasawara, E. (2023). TSPredIT: Integrated Tuning of Data Preprocessing and Time Series Prediction Models. Lecture Notes in Computer Science.

Examples

```
# Example: grid search over input_size and ELM hyperparameters
# Load library and example data
library(daltoolbox)
data(tsd)

# Prepare 10-lag windows and split into train/test
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)

# Define tuning: vary input_size and ELM hyperparameters (nhid, actfun)
tune <- ts_tune(
  input_size = 3:5,
  base_model = ts_elm(ts_norm_gminmax()),
  ranges = list(nhid = 1:5, actfun = c('purelin'))
)

# Run CV-based search and get the best fitted model
model <- fit(tune, x = io_train$input, y = io_train$output)

# Forecast and evaluate on the held-out horizon
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

A new ts_data object with preserved metadata and column names.

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