

Package ‘ubair’

July 22, 2025

Title Effects of External Conditions on Air Quality

Version 1.1.0

Description Analyzes the impact of external conditions on air quality using counterfactual approaches, featuring methods for data preparation, modeling, and visualization.

License GPL (>= 3)

URL <https://gitlab.opencode.de/uba-ki-lab/ubair>

Depends R (>= 4.4.0),

Encoding UTF-8

RoxygenNote 7.3.2

Suggests testthat (>= 3.0.0), deepnet, fastshap, treeshap, shapviz,
knitr, rmarkdown

Config/testthat/edition 3

Imports rlang, data.table, dplyr, ggplot2, forecast, lubridate, tidyr,
yaml, ranger, lightgbm

LazyData true

LazyDataCompression xz

VignetteBuilder knitr

Note Note: The included dataset is licensed under ``DL-DE-BY-2.0." See
the dataset documentation for details.

NeedsCompilation no

Author Raphael Franke [aut],
Imke Voss [aut, cre]

Maintainer Imke Voss <imke.voss@uba.de>

Repository CRAN

Date/Publication 2025-01-27 18:30:01 UTC

Contents

calc_performance_metrics	2
calc_summary_statistics	3
clean_data	4
copy_default_params	5
detrend	6
estimate_effect_size	7
get_meteo_available	8
load_params	8
load_uba_data_from_dir	9
mock_env_data	10
plot_counterfactual	11
plot_station_measurements	12
prepare_data_for_modelling	13
rescale_predictions	14
retrend_predictions	15
run_counterfactual	16
run_dynamic_regression	17
run_fnn	18
run_lightgbm	20
run_rf	21
sample_data_DESN025	22
scale_data	23
split_data_counterfactual	24

Index	26
--------------	-----------

calc_performance_metrics

Calculates performance metrics of a business-as-usual model

Description

Model agnostic function to calculate a number of common performance metrics on the reference time window. Uses the true data value and the predictions prediction for this calculation. The coverage is calculated from the columns value, prediction_lower and prediction_upper. Removes dates in the effect and buffer range as the model is not expected to be performing correctly for these times. The incorrectness is precisely what we are using for estimating the effect.

Usage

```
calc_performance_metrics(predictions, date_effect_start = NULL, buffer = 0)
```

Arguments

predictions	data.table or data.frame with the following columns date Date of the observation. Needs to be comparable to date_effect_start element. value True observed value of the station prediction Predicted model output for the same time and station as value prediction_lower Lower end of the prediction interval prediction_upper Upper end of the prediction interval
date_effect_start	A date. Start date of the effect that is to be evaluated. The data from this point onwards is disregarded for calculating model performance
buffer	Integer. An additional buffer window before date_effect_start to account for uncertainty in the effect start point. Disregards additional buffer data points for model evaluation

Value

Named vector with performance metrics of the model

calc_summary_statistics

Calculates summary statistics for predictions and true values

Description

Helps with analyzing predictions by comparing them with the true values on a number of relevant summary statistics.

Usage

```
calc_summary_statistics(predictions, date_effect_start = NULL, buffer = 0)
```

Arguments

predictions	Data.table or data.frame with the following columns date Date of the observation. Needs to be comparable to date_effect_start element. value True observed value of the station prediction Predicted model output for the same time and station as value
date_effect_start	A date. Start date of the effect that is to be evaluated. The data from this point onwards is disregarded for calculating model performance
buffer	Integer. An additional buffer window before date_effect_start to account for uncertainty in the effect start point. Disregards additional buffer data points for model evaluation

Value

data.frame of summary statistics with columns true and prediction

clean_data

Clean and Optionally Aggregate Environmental Data

Description

Cleans a data table of environmental measurements by filtering for a specific station, removing duplicates, and optionally aggregating the data on a daily basis using the mean.

Usage

```
clean_data(env_data, station, aggregate_daily = FALSE)
```

Arguments

env_data	A data table in long format. Must include columns: Station Station identifier for the data. Komponente Measured environmental component e.g. temperature, NO2. Wert Measured value. date Timestamp as Date-Time object (YYYY-MM-DD HH:MM:SS format). Komponente_txt Textual description of the component.
station	Character. Name of the station to filter by.
aggregate_daily	Logical. If TRUE, aggregates data to daily mean values. Default is FALSE.

Details

Duplicate rows (by date, Komponente, and Station) are removed. A warning is issued if duplicates are found.

Value

A data.table:

- If aggregate_daily = TRUE: Contains columns for station, component, day, year, and the daily mean value of the measurements.
- If aggregate_daily = FALSE: Contains cleaned data with duplicates removed.

Examples

```
# Example data
env_data <- data.table::data.table(
  Station = c("DENW094", "DENW094", "DENW006", "DENW094"),
  Komponente = c("NO2", "O3", "NO2", "NO2"),
  Wert = c(45, 30, 50, 40),
  date = as.POSIXct(c(
    "2023-01-01 08:00:00", "2023-01-01 09:00:00",
    "2023-01-01 08:00:00", "2023-01-02 08:00:00"
  )),
  Komponente_txt = c(
    "Nitrogen Dioxide", "Ozone", "Nitrogen Dioxide", "Nitrogen Dioxide"
  )
)

# Clean data for StationA without aggregation
cleaned_data <- clean_data(env_data, station = "DENW094", aggregate_daily = FALSE)
print(cleaned_data)
```

copy_default_params	<i>Copy Default Parameters File</i>
---------------------	-------------------------------------

Description

Copies the default params.yaml file, included with the package, to a specified destination directory. This is useful for initializing parameter files for custom edits.

Usage

```
copy_default_params(dest_dir)
```

Arguments

dest_dir Character. The path to the directory where the params.yaml file will be copied.

Details

The params.yaml file contains default model parameters for various configurations such as LightGBM, dynamic regression, and others. See the [load_params\(\)](#) documentation for an example of the file's structure.

Value

Nothing is returned. A message is displayed upon successful copying.

Examples

```
copy_default_params(tempdir())
```

detrend	<i>Removes trend from data</i>
---------	--------------------------------

Description

Takes a list of train and application data as prepared by `split_data_counterfactual()` and removes a polynomial, exponential or cubic spline trend function. Trend is obtained only from train data. Use as part of preprocessing before training a model based on decision trees, i.e. random forest and lightgbm. For the other methods it may be helpful but they are generally able to deal with trends themselves. Therefore we recommend to try out different versions and guide decisions using the model evaluation metrics from `calc_performance_metrics()`.

Usage

```
detrend(split_data, mode = "linear", num_splines = 5, log_transform = FALSE)
```

Arguments

<code>split_data</code>	List of two named dataframes called train and apply
<code>mode</code>	String which defines type of trend is present. Options are "linear", "quadratic", "exponential", "spline", "none". "none" returns original data
<code>num_splines</code>	Defines the number of cubic splines if mode="spline". Choose num_splines=1 for cubic polynomial trend. If mode!="spline", this parameter is ignored
<code>log_transform</code>	If TRUE, use a log-transformation before detrending to ensure positivity of all predictions in the rest of the pipeline. A exp transformation is necessary during retrending to return to the solution space. Use only in combination with log_transform parameter in <code>retrend_predictions()</code>

Details

Apply `retrend_predictions()` to predictions to return to the original data units.

Value

List of 3 elements. 2 dataframes: detrended train, apply and the trend function

Examples

```
data(mock_env_data)
split_data <- list(
  train = mock_env_data[1:80, ],
  apply = mock_env_data[81:100, ]
)
detrended_list <- detrend(split_data, mode = "linear")
detrended_train <- detrended_list$train
detrended_apply <- detrended_list$apply
trend <- detrended_list$model
```

estimate_effect_size	<i>Estimates size of the external effect</i>
----------------------	--

Description

Calculates an estimate for the absolute and relative effect size of the external effect. The absolute effect is the difference between the model bias in the reference time and the effect time windows. The relative effect is the absolute effect divided by the mean true value in the reference window.

Usage

```
estimate_effect_size(df, date_effect_start, buffer = 0, verbose = FALSE)
```

Arguments

df	Data.table or data.frame with the following columns date Date of the observation. Needs to be comparable to date_effect_start element. value True observed value of the station prediction Predicted model output for the same time and station as value
date_effect_start	A date. Start date of the effect that is to be evaluated. The data from this point onward is disregarded for calculating model performance.
buffer	Integer. An additional buffer window before date_effect_start to account for uncertainty in the effect start point. Disregards additional buffer data points for model evaluation
verbose	Prints an explanation of the results if TRUE

Details

Note: Since the bias of the model is an average over predictions and true values, it is important, that the effect window is specified correctly. Imagine a scenario like a fire which strongly affects the outcome for one hour and is gone the next hour. If we use a two week effect window, the estimated effect will be $14 \times 24 = 336$ times smaller compared to using a 1-hour effect window. Generally, we advise against studying very short effects (single hour or single day). The variability of results will be too large to learn anything meaningful.

Value

A list with two numbers: Absolute and relative estimated effect size.

get_meteo_available	<i>Get Available Meteorological Components</i>
---------------------	--

Description

Identifies unique meteorological components from the provided environmental data, filtering only those that match the predefined UBA naming conventions. These components include "GLO", "LDR", "RFE", "TMP", "WIG", "WIR", "WIND_U", and "WIND_V".

Usage

```
get_meteo_available(env_data)
```

Arguments

env_data	Data table containing environmental data. Must contain column "Komponente"
----------	--

Value

A vector of available meteorological components.

Examples

```
# Example environmental data
env_data <- data.table::data.table(
  Komponente = c("TMP", "NO2", "GLO", "WIR"),
  Wert = c(25, 40, 300, 50),
  date = as.POSIXct(c(
    "2023-01-01 08:00:00", "2023-01-01 09:00:00",
    "2023-01-01 10:00:00", "2023-01-01 11:00:00"
  ))
)
# Get available meteorological components
meteo_components <- get_meteo_available(env_data)
print(meteo_components)
```

load_params	<i>Load Parameters from YAML File</i>
-------------	---------------------------------------

Description

Reads a YAML file containing model parameters, including station settings, variables, and configurations for various models. If no file path is provided, the function defaults to loading `params.yaml` from the package's `extdata` directory.

Usage

```
load_params(filepath = NULL)
```

Arguments

filepath Character. Path to the YAML file. If NULL, the function will attempt to load the default params.yaml provided in the package.

Details

The YAML file should define parameters in a structured format, such as:

```
target: 'N02'

lightgbm:
  nrounds: 200
  eta: 0.03
  num_leaves: 32

dynamic_regression:
  ntrain: 8760

random_forest:
  num.trees: 300
  max.depth: 10

meteo_variables:
- GLO
- TMP
```

Value

A list containing the parameters loaded from the YAML file.

Examples

```
params <- load_params()
```

```
load_uba_data_from_dir
```

Load UBA Data from Directory

Description

This function loads data from CSV files in the specified directory. It supports two formats:

Usage

```
load_uba_data_from_dir(data_dir)
```

Arguments

`data_dir` Character. Path to the directory containing .csv files.

Details

1. "inv": Files must contain the following columns:
 - Station, Komponente, Datum, Uhrzeit, Wert.
2. "24Spalten": Files must contain:
 - Station, Komponente, Datum, and columns Wert01, ..., Wert24.

File names should include "inv" or "24Spalten" to indicate their format. The function scans recursively for .csv files in subdirectories and combines the data into a single `data.table` in long format. Files that are not in the expected format will be ignored.

Value

A `data.table` containing the loaded data in long format. Returns an error if no valid files are found or the resulting dataset is empty.

mock_env_data

Mock Environmental Data

Description

A small dataset of environmental variables created for testing and examples. This dataset includes hourly observations with random values for meteorological and temporal variables.

Usage

```
mock_env_data
```

Format

A data frame with 100 rows and 12 variables:

date POSIXct. Date and time of the observation (hourly increments).

value Numeric. Randomly generated target variable.

GLO Numeric. Global radiation in W/m² (random values between 0 and 1000).

TMP Numeric. Temperature in °C (random values between -10 and 35).

RFE Numeric. Rainfall in mm (random values between 0 and 50).

WIG Numeric. Wind speed in m/s (random values between 0 and 20).

WIR Numeric. Wind direction in degrees (random values between 0 and 360).
LDR Numeric. Longwave downward radiation in W/m² (random values between 0 and 500).
day_julian Integer. Julian day of the year, ranging from 1 to 10.
weekday Integer. Day of the week, ranging from 1 (Monday) to 7 (Sunday).
hour Integer. Hour of the day, ranging from 0 to 23.
date_unix Numeric. UNIX timestamp (seconds since 1970-01-01 00:00:00 UTC).

Source

Generated within the package for example purposes.

Examples

```
data(mock_env_data)
head(mock_env_data)
```

plot_counterfactual	<i>Prepare Plot Data and Plot Counterfactuals</i>
---------------------	---

Description

Smooths the predictions using a rolling mean, prepares the data for plotting, and generates the counterfactual plot for the application window. Data before the red box are reference window, red box is buffer and values after black, dotted line are effect window.

Usage

```
plot_counterfactual(
  predictions,
  params,
  window_size = 14,
  date_effect_start = NULL,
  buffer = 0,
  plot_pred_interval = TRUE
)
```

Arguments

predictions	The data.table containing the predictions (hourly)
params	Parameters for plotting, including the target variable.
window_size	The window size for the rolling mean (default is 14 days).
date_effect_start	A date. Start date of the effect that is to be evaluated. The data from this point onwards is disregarded for calculating model performance

buffer	Integer. An additional, optional buffer window before date_effect_start to account for uncertainty in the effect start point. Disregards additional buffer data points for model evaluation. Use buffer=0 for no buffer.
plot_pred_interval	Boolean. If TRUE, shows a grey band of the prediction interval.

Details

The optional grey ribbon is a prediction interval for the hourly values. The interpretation for a 90% prediction interval (to be defined in alpha parameter of `run_counterfactual()`) is that 90% of the true hourly values (not the rolled means) lie within the grey band. This might be helpful for getting an idea of the variance of the data and predictions.

Value

A ggplot object with the counterfactual plot. Can be adjusted further, e.g. set limits for the y-axis for better visualisation.

plot_station_measurements

Descriptive plot of daily time series data

Description

This function produces descriptive time-series plots with smoothing for the meteorological and potential target variables that were measured at a station.

Usage

```
plot_station_measurements(
  env_data,
  variables,
  years = NULL,
  smoothing_factor = 1
)
```

Arguments

env_data	A data table of measurements of one air quality measurement station. The data should contain the following columns: Station Station identifier where the data was collected. Komponente The environmental component being measured (e.g., temperature, NO2). Wert The measured value of the component. date The timestamp for the observation, formatted as a Date-Time object in the format "YYYY-MM-DD HH:MM:SS" (e.g., "2010-01-01 07:00:00").
----------	---

	Komponente_txt A textual description or label for the component.
variables	list of variables to plot. Must be in <code>env_data\$Komponente</code> . Meteorological variables can be obtained from <code>params.yaml</code> .
years	Optional. A numeric vector, list, or a range specifying the years to restrict the plotted data. You can provide: • A single year: <code>years = 2020</code> • A numeric vector of years: <code>years = c(2019, 2020, 2021)</code> • A range of years: <code>years = 2019:2021</code> If not provided, data for all available years will be used.
smoothing_factor	A number that defines the magnitude of smoothing. Default is 1. Smaller numbers correspond to less smoothing, larger numbers to more.

Value

A ggplot object. This object contains:

- A time-series line plot for each variable in `variables`.
- Smoothed lines, with smoothing defined by `smoothing_factor`.

Examples

```
library(data.table)
env_data <- data.table(
  Station = "Station_1",
  Komponente = rep(c("TMP", "NO2"), length.out = 100),
  Wert = rnorm(100, mean = 20, sd = 5),
  date = rep(seq.POSIXt(as.POSIXct("2022-01-01"), , "hour", 50), each = 2),
  year = 2022,
  Komponente_txt = rep(c("Temperature", "NO2"), length.out = 100)
)
plot <- plot_station_measurements(env_data, variables = c("TMP", "NO2"))
```

```
prepare_data_for_modelling
```

Prepare Data for Training a model

Description

Prepares environmental data by filtering for relevant components, converting the data to a wide format, and adding temporal features. Should be called before [split_data_counterfactual\(\)](#)

Usage

```
prepare_data_for_modelling(env_data, params)
```

Arguments

env_data	A data table in long format. Must include the following columns: Station Station identifier for the data. Komponente The environmental component being measured (e.g., temperature, NO2). Wert The measured value of the component. date Timestamp as POSIXct object in YYYY-MM-DD HH:MM:SS format. Komponente_txt A textual description of the component.
params	A list of modelling parameters loaded from params.yaml. Must include: meteo_variables A vector of meteorological variable names. target The name of the target variable.

Value

A data.table in wide format, with columns: date, one column per component, and temporal features like date_unix, day_julian, weekday, and hour.

Examples

```
env_data <- data.table::data.table(
  Station = c("StationA", "StationA", "StationA"),
  Komponente = c("NO2", "TMP", "NO2"),
  Wert = c(50, 20, 40),
  date = as.POSIXct(c("2023-01-01 10:00:00", "2023-01-01 11:00:00", "2023-01-02 12:00:00"))
)
params <- list(meteo_variables = c("TMP"), target = "NO2")
prepared_data <- prepare_data_for_modelling(env_data, params)
print(prepared_data)
```

rescale_predictions	<i>Rescale predictions to original scale.</i>
---------------------	---

Description

This function rescales the predicted values (prediction, prediction_lower, prediction_upper). The scaling is reversed using the means and standard deviations that were saved from the training data. It is the inverse function to [scale_data\(\)](#) and should be used only in combination.

Usage

```
rescale_predictions(scale_result, dt_predictions)
```

Arguments

- `scale_result` A list object returned by `scale_data()`, containing the means and standard deviations used for scaling.
- `dt_predictions` A data frame containing the predictions, including columns `prediction`, `prediction_lower`, `prediction_upper`.

Value

A data frame with the predictions and numeric columns rescaled back to their original scale.

Examples

```
data(mock_env_data)
scale_res <- scale_data(
  train_data = mock_env_data[1:80, ],
  apply_data = mock_env_data[81:100, ]
)
params <- load_params()
res <- run_lightgbm(
  train = scale_res$train, test = scale_res$apply,
  params$lightgbm, alpha = 0.9, calc_shaps = FALSE
)
dt_predictions <- res$dt_predictions
rescaled_predictions <- rescale_predictions(scale_res, dt_predictions)
```

`retrend_predictions` *Restors the trend in the prediction*

Description

Takes a dataframe of predictions as returned by any of the 'run_model' functions and restores a trend which was previously removed via `detrend()`. This is necessary for the predictions and the true values to have the same units. The function is basically the inverse function to `detrend()` and should only be used in combination with it.

Usage

```
retrend_predictions(dt_predictions, trend, log_transform = FALSE)
```

Arguments

- `dt_predictions` Dataframe of predictions with columns `value`, `prediction`, `prediction_lower`, `prediction_upper`
- `trend` lm object generated by `detrend()`
- `log_transform` Returns values to solution space, if they have been log transformed during de-trending. Use only in combination with `log_transform` parameter in `detrend` function.

Value

Retrended dataframe with same structure as `dt_predictions` which is returned by any of the `run_model()` functions.

Examples

```
data(mock_env_data)
split_data <- list(
  train = mock_env_data[1:80, ],
  apply = mock_env_data[81:100, ]
)
params <- load_params()
detrended_list <- detrend(split_data,
  mode = "linear"
)
trend <- detrended_list$model
detrended_train <- detrended_list$train
detrended_apply <- detrended_list$apply
result <- run_lightgbm(
  train = detrended_train,
  test = detrended_apply,
  model_params = params$lightgbm,
  alpha = 0.9,
  calc_shaps = FALSE
)
retrended_predictions <- retrend_predictions(result$dt_predictions, trend)
```

<code>run_counterfactual</code>	<i>Full counterfactual simulation run</i>
---------------------------------	---

Description

Chains detrending, training of a selected model, prediction and retrending together for ease of use. See documentation of individual functions for details.

Usage

```
run_counterfactual(
  split_data,
  params,
  detrending_function = "none",
  model_type = "rf",
  alpha = 0.9,
  log_transform = FALSE,
  calc_shaps = FALSE
)
```

Arguments

split_data	List of two named dataframes called train and apply
params	A list of parameters that define the following: meteo_variables A character vector specifying the names of the meteorological variables used as inputs. model A list of hyperparameters for training the chosen model. Name of this list and its parameters depend on the chosen models. See run_dynamic_regression(), run_lightgbm(), run_rf() and run_fnn() functions for details
detrending_function	String which defines type of trend to remove. Options are "linear", "quadratic", "exponential", "spline", "none". See detrend() and retrend_predictions() for details.
model_type	String to decide which model to use. Current options random forest "rf", gradient boosted decision trees "lightgbm", "dynamic_regression" and feedforward neural network "fnn"
alpha	Confidence level of the prediction interval between 0 and 1.
log_transform	If TRUE, uses log transformation during detrending and retrending. For details see detrend() documentation
calc_shaps	Boolean value. If TRUE, calculate SHAP values for the method used and format them so they can be visualised with shapviz:sv_importance() and shapviz:sv_dependence() . The SHAP values are generated for a subset (or all, depending on the size of the dataset) of the test data.

Value

Data frame of predictions, model and importance

Examples

```
data(mock_env_data)
split_data <- list(
  train = mock_env_data[1:80, ],
  apply = mock_env_data[81:100, ]
)
params <- load_params()
res <- run_counterfactual(split_data, params, detrending_function = "linear")
prediction <- res$retrended_predictions
random_forest_model <- res$model
```

run_dynamic_regression

Run the dynamic regression model

Description

This function trains a dynamic regression model with fourier transformed temporal features and meteorological variables as external regressors on the specified training dataset and makes predictions on the test dataset in a counterfactual scenario. This is referred to as a dynamic regression model in [Forecasting: Principles and Practise, Chapter 10 - Dynamic regression models](#)

Usage

```
run_dynamic_regression(train, test, params, alpha, calc_shaps)
```

Arguments

train	Dataframe of train data as returned by the split_data_counterfactual() function.
test	Dataframe of test data as returned by the split_data_counterfactual() function.
params	list of hyperparameters to use in dynamic_regression call. Only uses ntrain to specify the number of data points to use for training. Default is 8760 which results in 1 year of hourly data
alpha	Confidence level of the prediction interval between 0 and 1.
calc_shaps	Boolean value. If TRUE, calculate SHAP values for the method used and format them so they can be visualised with shapviz:sv_importance() and shapviz:sv_dependence() . The SHAP values are generated for a subset (or all, depending on the size of the dataset) of the test data.

Details

Note: Runs the dynamic regression model for individualised use with own data pipeline. Otherwise use [run_counterfactual\(\)](#) to call this function.

Value

Data frame of predictions and model

run_fnn	<i>Train a Feedforward Neural Network (FNN) in a Counterfactual Scenario.</i>
---------	---

Description

Trains a feedforward neural network (FNN) model on the specified training dataset and makes predictions on the test dataset in a counterfactual scenario. The model uses meteorological variables and sin/cosine-transformed features. Scales the data before training and rescales predictions, as the model does not converge with unscaled data.

Usage

```
run_fnn(train, test, params, calc_shaps)
```

Arguments

train	A data frame or tibble containing the training dataset, including the target variable (value) and meteorological variables specified in <code>params\$meteo_variables</code> .
test	A data frame or tibble containing the test dataset on which predictions will be made, using the same meteorological variables as in the training dataset.
params	A list of parameters that define the following: meteo_variables A character vector specifying the names of the meteorological variables used as inputs. fnn A list of hyperparameters for training the feedforward neural network, including: • <code>activation_fun</code>: The activation function for the hidden layers (e.g., "sigmoid", "tanh"). • <code>momentum</code>: The momentum factor for training. • <code>learningrate_scale</code>: Factor for adjusting learning rate. • <code>output_fun</code>: The activation function for the output layer • <code>batchsize</code>: The size of the batches during training. • <code>hidden_dropout</code>: Dropout rate for the hidden layers to prevent overfitting. • <code>visible_dropout</code>: Dropout rate for the input layer. • <code>hidden_layers</code>: A vector specifying the number of neurons in each hidden layer. • <code>num_epochs</code>: Number of epochs (iterations) for training. • <code>learning_rate</code>: Initial learning rate.
calc_shaps	Boolean value. If TRUE, calculate SHAP values for the method used and format them so they can be visualised with <code>shapviz:sv_importance()</code> and <code>shapviz:sv_dependence()</code> . The SHAP values are generated for a subset (or all, depending on the size of the dataset) of the test data.

Details

This function provides flexibility for users with their own data pipelines or workflows. For a simplified pipeline, consider using `run_counterfactual()`.

Experiment with hyperparameters such as `learning_rate`, `batchsize`, `hidden_layers`, and `num_epochs` to improve performance.

Warning: Using many or large hidden layers in combination with a high number of epochs can lead to long training times.

Value

A list with three elements:

`dt_predictions` A data frame containing the test data along with the predicted values:

prediction The predicted values from the FNN model.

prediction_lower The same predicted values, as no quantile model is available yet for FNN.

prediction_upper The same predicted values, as no quantile model is available yet for FNN.

model The trained FNN model object from the `deepnet::nn.train()` function.

importance SHAP importance values (if `calc_shaps = TRUE`). Otherwise, NULL.

Examples

```
data(mock_env_data)
params <- load_params()
res <- run_fnn(
  train = mock_env_data[1:80, ],
  test = mock_env_data[81:100, ], params,
  calc_shaps = FALSE
)
```

run_lightgbm

Run gradient boosting model with lightgbm

Description

This function trains a gradient boosting model (lightgbm) on the specified training dataset and makes predictions on the test dataset in a counterfactual scenario. The model uses meteorological variables and temporal features.

Usage

```
run_lightgbm(train, test, model_params, alpha, calc_shaps)
```

Arguments

train	Dataframe of train data as returned by the split_data_counterfactual() function.
test	Dataframe of test data as returned by the split_data_counterfactual() function.
model_params	list of hyperparameters to use in <code>lgb.train</code> call. See lightgbm:lgb.train() params argument for details.
alpha	Confidence level of the prediction interval between 0 and 1.
calc_shaps	Boolean value. If TRUE, calculate SHAP values for the method used and format them so they can be visualised with shapviz:sv_importance() and shapviz:sv_dependence() . The SHAP values are generated for a subset (or all, depending on the size of the dataset) of the test data.

Details

Note: Runs the gradient boosting model for individualised use with own data pipeline. Otherwise use [run_counterfactual\(\)](#) to call this function.

Value

List with data frame of predictions and model

Examples

```
data(mock_env_data)
split_data <- list(
  train = mock_env_data[1:80, ],
  apply = mock_env_data[81:100, ]
)
params <- load_params()
variables <- c("day_julian", "weekday", "hour", params$meteo_variables)
res <- run_lightgbm(
  train = mock_env_data[1:80, ],
  test = mock_env_data[81:100, ], params$lightgbm, alpha = 0.9,
  calc_shaps = FALSE
)
prediction <- res$dt_predictions
model <- res$model
```

run_rf

Run random forest model with ranger

Description

This function trains a random forest model (ranger) on the specified training dataset and makes predictions on the test dataset in a counterfactual scenario. The model uses meteorological variables and temporal features.

Usage

```
run_rf(train, test, model_params, alpha, calc_shaps)
```

Arguments

train	Dataframe of train data as returned by the split_data_counterfactual() function.
test	Dataframe of test data as returned by the split_data_counterfactual() function.
model_params	list of hyperparameters to use in ranger call. See ranger:ranger() for options.
alpha	Confidence level of the prediction interval between 0 and 1.

calc_shaps Boolean value. If TRUE, calculate SHAP values for the method used and format them so they can be visualised with `shapviz:sv_importance()` and `shapviz:sv_dependence()`. The SHAP values are generated for a subset (or all, depending on the size of the dataset) of the test data.

Details

Note: Runs the random forest model for individualised use with own data pipeline. Otherwise use `run_counterfactual()` to call this function.

Value

List with data frame of predictions and model

sample_data_DESN025	<i>Environmental Data for Modelling from station DESN025 in Leipzig-Mitte.</i>
---------------------	--

Description

This dataset contains environmental measurements from the Leipzig Mitte station provided by the Sächsisches Landesamt für Umwelt, Landwirtschaft und Geologie (LfULG). Alterations in the data: Codes for incorrect values have been removed.

Usage

```
sample_data_DESN025
```

Format

A data table with the following columns:

Station Station identifier where the data was collected.

Komponente The environmental component being measured (e.g., temperature, NO2).

Wert The measured value of the component.

date The timestamp for the observation, formatted as a Date-Time object in the format "YYYY-MM-DD HH:MM:SS" (e.g., "2010-01-01 07:00:00").

Komponente_txt A textual description or label for the component.

The dataset is structured in a long format and is prepared for further transformation into a wide format for modelling.

Details

The dataset is licensed under the "Data Licence Germany – attribution – version 2.0 (DL-DE-BY-2.0)". (1) Any use will be permitted provided it fulfils the requirements of this "Data licence Germany – attribution – Version 2.0".

The data and meta-data provided may, for commercial and non-commercial use, in particular

- be copied, printed, presented, altered, processed and transmitted to third parties;
- be merged with own data and with the data of others and be combined to form new and independent datasets;
- be integrated in internal and external business processes, products and applications in public and non-public electronic networks.

(2) The user must ensure that the source note contains the following information:

- the name of the provider,
- the annotation "Data licence Germany – attribution – Version 2.0" or "dl-de/by-2-0" referring to the licence text available at www.govdata.de/dl-de/by-2-0, and
- a reference to the dataset (URI).

This applies only if the entity keeping the data provides the pieces of information 1-3 for the source note.

(3) Changes, editing, new designs or other amendments must be marked as such in the source note.

For more information on the license, visit <https://www.govdata.de/dl-de/by-2-0>.

Source

Sächsisches Landesamt für Umwelt, Landwirtschaft und Geologie (LfULG).

Examples

```
data(sample_data_DESN025)
params <- load_params()
dt_prepared <- prepare_data_for_modelling(sample_data_DESN025, params)
```

scale_data

Standardize Training and Application Data

Description

This function standardizes numeric columns of the `train_data` and applies the same scaling (mean and standard deviation) to the corresponding columns in `apply_data`. It returns the standardized data along with the scaling parameters (means and standard deviations). This is particularly important for neural network approaches as they tend to be numerically unstable and deteriorate otherwise.

Usage

```
scale_data(train_data, apply_data)
```

Arguments

train_data	A data frame containing the training dataset to be standardized. It must contain numeric columns.
apply_data	A data frame containing the dataset to which the scaling from train_data will be applied.

Value

A list containing the following elements:

train	The standardized training data.
apply	The apply_data scaled using the means and standard deviations from the train_data.
means	The means of the numeric columns in train_data.
sds	The standard deviations of the numeric columns in train_data.

Examples

```
data(mock_env_data)
detrended_list <- list(
  train = mock_env_data[1:80, ],
  apply = mock_env_data[81:100, ]
)
scale_result <- scale_data(
  train_data = detrended_list$train,
  apply_data = detrended_list$apply
)
scaled_train <- scale_result$train
scaled_apply <- scale_result$apply
```

```
split_data_counterfactual
```

Split Data into Training and Application Datasets

Description

Splits prepared data into training and application datasets based on specified date ranges for a business-as-usual scenario. Data before application_start and after application_end is used as training data, while data within the date range is used for application.

Usage

```
split_data_counterfactual(dt_prepared, application_start, application_end)
```

Arguments

dt_prepared The prepared data table.

application_start The start date(date object) for the application period of the business-as-usual simulation. This coincides with the start of the reference window. Can be created by e.g. `lubridate::ymd("20191201")`

application_end The end date(date object) for the application period of the business-as-usual simulation. This coincides with the end of the effect window. Can be created by e.g. `lubridate::ymd("20191201")`

Value

A list with two elements:

train Data outside the application period.

apply Data within the application period.

Examples

```
dt_prepared <- data.table::data.table(  
  date = as.Date(c("2023-01-01", "2023-01-05", "2023-01-10")),  
  value = c(50, 60, 70)  
)  
result <- split_data_counterfactual(  
  dt_prepared,  
  application_start = as.Date("2023-01-03"),  
  application_end = as.Date("2023-01-08")  
)  
print(result$train)  
print(result$apply)
```

Index

- * **datasets**
 - mock_env_data, [10](#)
 - sample_data_DESN025, [22](#)
- calc_performance_metrics, [2](#)
- calc_performance_metrics(), [6](#)
- calc_summary_statistics, [3](#)
- clean_data, [4](#)
- copy_default_params, [5](#)
- detrend, [6](#)
- detrend(), [15](#), [17](#)
- estimate_effect_size, [7](#)
- get_meteo_available, [8](#)
- lightgbm:lgb.train(), [20](#)
- load_params, [8](#)
- load_params(), [5](#)
- load_uba_data_from_dir, [9](#)
- mock_env_data, [10](#)
- plot_counterfactual, [11](#)
- plot_station_measurements, [12](#)
- prepare_data_for_modelling, [13](#)
- ranger:ranger(), [21](#)
- rescale_predictions, [14](#)
- retrend_predictions, [15](#)
- retrend_predictions(), [6](#), [17](#)
- run_counterfactual, [16](#)
- run_counterfactual(), [12](#), [18](#), [19](#), [21](#), [22](#)
- run_dynamic_regression, [17](#)
- run_dynamic_regression(), [17](#)
- run_fnn, [18](#)
- run_fnn(), [17](#)
- run_lightgbm, [20](#)
- run_lightgbm(), [17](#)
- run_rf, [21](#)
- run_rf(), [17](#)
- sample_data_DESN025, [22](#)
- scale_data, [23](#)
- scale_data(), [14](#), [15](#)
- shapviz:sv_dependence(), [17–20](#), [22](#)
- shapviz:sv_importance(), [17–20](#), [22](#)
- split_data_counterfactual, [24](#)
- split_data_counterfactual(), [6](#), [13](#), [18](#), [20](#), [21](#)