pyprocessta

Release v0.1.0+114.gf131d2d.dirty

Kevin Maik Jablonka

Feb 24, 2022

CONTENTS:

pyprocessta collects utilities for the analysis for timeseries data as they are collected in industrial processes. We used the tools to understand the solvent emissions from a post-combustion carbon capture plant.

CHAPTER

ONE

INSTALLATION

We recommend installing pyprocessta in a dedicated virtual environment or conda environment. Note that we tested the code on Python 3.8.

The latest version of pyprocessta can be installed from GitHub using

```
pip install git+https://github.com/kjappelbaum/pyprocessta.git
```

CHAPTER

TWO

PREPROCESSING

For basic preprocessing functions the `pyprocessta.preprocess` module can be used.

2.1 Aligning to dataframes

To align two dataframes, use

```
from pyprocessta.preprocess.align import align_two_dfs

aligned_dataframe = align_two_dfs(dataframe_a, dataframe_b)
```

2.2 Filtering and smoothing

To perform basic filtering operations you can use

```
from pyprocessta.preprocess.smooth import z_score_filter, exponential_window_smoothing

dataframe_no_spikes = z_score_filter(dataframe)
dataframe_smoothed = exponential_window_smoothing(dataframe)
```

2.3 Detrending

Often, it can be useful to remove trend components from time series data. One can distinguish stochastic and deterministic trend components, and we provide utilities to remove both

```
from pyprocessta.detrend import detrend_stochastic, detrend_linear_deterministc

dataframe_no_linear_trend = detrend_linear_deterministc(input_dataframe)
dataframe_no_stochastic_trend = detrend_stochastic(input_dataframe)
```

2.4 Resampling

For many applications it is important to have data sampled on a regular grid. To resample data onto such a grid you can use

```
from pyprocessta.resample import resample_regular

data_resampled = resample_regular(input_dataframe, interval='2min')
```

CHAPTER

THREE

EDA

3.1 Test for stationarity

One of the most important tests before modeling time series data is to check for stationarity (since many of the "simple" time series models assume stationarity).

```
from pyprocessta.eda.statistics import check_stationarity

test_results = check_stationarity(input_dataseries)
```

This will perform the Augmented-Dickey Fuller and Kwiatkowski–Phillips–Schmidt–Shin (KPSS).

3.2 Granger causality

One interesting analysis is to check for "correlations" between different timeseries. In timeseries speak, this means to look for Granger causality. To perform this analysis, you can use

```
from pyprocessta.eda.statistics import compute_granger_causality_matrix

causality_matrix = compute_granger_causality_matrix(input_dataframe)
```

The matrix can, for example, be plotted as heatmap and highlights the maximum "correlation" between two series (up to some maximum lag).

CHAPTER

FOUR

TRAINING A TCN MODEL

The Temporal convolutional neural network implementation uses the darts library. The only change is that we make it possible to also enable dropout for inference.

```
from pyprocessta.model.tcn import run_model, get_train_test_data, transform_data, get_
↪data

x_timeseries, y_timeseries = get_data(my_dataframe, targets=my_targets, features=my_
↪features_
train_tuple, test_tuple = get_train_test_data(x_timeseries, y_timeseries, split_date=
↪"2010-01-18 12:59:15")
train_tuple, test_tuples, transformers = transform_data(train_tuple, [test_tuple])

model = run_model(train_tuple)
```

CHAPTER

FIVE

CAUSAL IMPACT ANALYSIS

Causal impact analysis allows to estimate the effect of some intervention in the absence of a control experiment. For doing so, one builds a model of what the behavior of the system would be without the intervention. The approach used in the original causal impact paper uses Bayesian structured time series models which, simply speaking, model time series via two key equation: a state equation that connects a latent, unobserved, state to the observations and once equation that describes the transition between states. The model is then defined by a model for the state and transitions between the states (e.g., local level and seasonality). An efficient Python implementation of this is provided by the *tfcausalimpact* package. We provide a wrapper for this

```
from pyprocessta.causalimpact import run_causal_impact_analysis

ci = run_causal_impact_analysis(
    df=data,
    x_columns=["a", "b", "c"],
    intervention_column="a",
    y_column="e",
    start=[s_0, s_1],
    end=[e_0, e_1],
)
```

Where *ci* is an object.

In our work, we used the causal impact framework with TCN models with Monte-Carlo dropout uncertainty estimates. You can find the code for this in the *paper* directory.

CHAPTER

SIX

THE PYPROCESSTA API REFERENCE

6.1 Preprocessing

Sometimes, different kinds of measurements are sampled at different intervals. This module provides utilities to combine such data. We will always operate on pandas dataframes with datatime indexing

pyprocessta.preprocess.align.**align_two_dfs**(*df_a*, *df_b*, *interpolation='linear'*)

Alignes to dataframes with datatimeindex Resamples both dataframes on the dataframe with the lowest frequency timestep. The first timepoint in the new dataframe will be the later one of the first observations of the dataframes.

https://stackoverflow.com/questions/47148446/pandas-resample-interpolate-is-producing-nans https://stackoverflow.com/questions/66967998/pandas-interpolation-giving-odd-results

Parameters

- **df_a** (*pd.DataFrame*) – Dataframe
- **df_b** (*pd.DataFrame*) – Dataframe
- **interpolation** (*Union[str, int], optional*) – Interpolation method. If you provide an integer, spline interpolation of that order will be used. Defaults to "linear".

Returns merged dataframe

Return type pd.DataFrame

This module contains basic data cleaning functions

pyprocessta.preprocess.clean.**drop_duplicated_indices**(*df*)

If one concatenates dataframes there might be duplicated indices. This can lead to problems, e.g., in interpolation steps. One easy solution can be to just drop the duplicated row

Parameters **df** (*Union[pd.Series, pd.DataFrame]*) – Input data

Returns Data without duplicated indices

Return type Union[pd.Series, pd.DataFrame]

In some time series there is a trend component that does not interest us, e.g., because we have domain knowledge that this trend is due to another phenomenon like instrument drift. In this case, we might want to remove the trend component for furhter modeling. The same is the case for the variance. If the variance increases over time, one might want to remove this effect using a Box-Cox transformation [1]

References: [1] https://otexts.com/fpp2/transformations.html#mathematical-transformations

pyprocessta.preprocess.detrend.**detrend_linear_deterministc**(*data*)

Removes a deterministic linear trend from a series. Note that we assume that the data is sampled on a regular grid and we estimate the trend as

np.arange(len(series) * (series.iloc[end] - series.iloc[start]) / (end - start)

)

Parameters **data** (*Union[pd.Series, pd.DataFrame]*) – Data to detrend. In case of dataframes we detrend every column separately.

Returns Detrended data

Return type Union[pd.Series, pd.DataFrame]

pyprocessta.preprocess.detrend.**detrend_stochastic**(*data*)

Detrends time series data using the difference method y_t - y_{t-1}. This is useful to remove stochastic trends (random walk with trend).

Parameters **data** (*Union[pd.Series, pd.DataFrame]*) – Time series data to detrend

Returns Differenced data

Return type Union[pd.Series, pd.DataFrame]

Often, data is not sampled on a regular grid. This module provides to regularize such data

pyprocessta.preprocess.resample.**resample_regular**(*df*, *interval='10min'*, *interpolation='linear'*, *start_time=None*)

Resamples the dataframe at a desired interval.

Parameters

- **df** (*pd.DataFrame*) – input dataframne
- **interval** (*str, optional*) – Resampling intervall. Defaults to "10min".
- **interpolation** (*Union[str, int], optional*) – Interpolation method. If you provide an integer, spline interpolation of that order will be used. Defaults to "linear".

Returns Output data.

Return type pd.DataFrame

pyprocessta.preprocess.smooth.**exponential_window_smoothing**(*data*, *window_size*, *aggregation='mean'*)

Parameters

- **data** (*Union[pd.Series, pd.DataFrame]*) – Data to smoothen
- **window_size** (*int*) – size for the exponential window
- **aggregation** (*str, optional*) – Aggregation function. Defaults to "mean".

Returns Smoothned data

Return type Union[pd.Series, pd.DataFrame]

pyprocessta.preprocess.smooth.**z_score_filter**(*data*, *threshold=2*, *window=10*)

Replaces spikes (values > threshold * z_score) with the median of the window values before.

Parameters

- **data** (*Union[pd.Series, pd.DataFrame]*) – Series to despike
- **threshold** (*float, optional*) – Threshold on the z-score. Defaults to 2.
- **window** (*int, option*) – Window that is used for the median with which the spike value is replaced. This mean only looks back.

Returns Despiked series

Return type Union[pd.Series, pd.DataFrame]

6.2 EDA

pyprocessta.eda.statistics.**check_granger_causality**(*x*, *y*, *max_lag=20*, *add_constant=True*)

Check if series x is Granger causal for series y We reject the null hypothesis that x does *not* Granger cause y if the pvalues are below a desired size of the test.

Parameters

- **x** (*pd.Series*) – Time series.
- **y** (*pd.Series*) – Time series.
- **max_lag** (*int, optional*) – Maximum lag to use for the causality checks. Defaults to 20.
- **add_constant** (*bool, optional*) – [description]. Defaults to True.

Returns results dictionary

Return type dict

pyprocessta.eda.statistics.**check_stationarity**(*series*, *threshold=0.05*, *regression='c'*)

Performs the Augmented-Dickey fuller and Kwiatkowski-Phillips-Schmidt-Shin (KPSS) tests for stationarity.

Parameters

- **series** (*pd.Series*) – Time series data
- **threshold** (*float, optional*) – p-value thresholds for the statistical tests. Defaults to 0.05.
- **regression** (*str, optional*) – If regression="c" then the tests check for stationarity around a constant. For "ct" the test check for stationarity around a trend. Defaults to "c".

Returns Results dictionary with key "stationary" that has a bool as value

Return type dict

6.3 Causal impact analysis

Causal impact analysis uses machine learning to construct a counterfactual (what would the results have been without an intervention) which can be used to estimate an effect size without the need for a control group A good introduction is https://www.youtube.com/watch?v=GTgZfCltMm8 the original paper is https://storage.googleapis.com/pub-tools-public-publication-data/pdf/41854.pdf. The particular implementation we use was described in https://towardsdatascience.com/implementing-causal-impact-on-top-of-tensorflow-probability-c837ea18b126

One can use covariates to build the counterfactual but one needs to be careful that they are not changed by the intervention.

pyprocessta.causalimpact.**run_causal_impact_analysis**(*df*, *x_columns*, *intervention_column*, *y_column*, *start*, *end*, *p_value_threshold=0.05*)

Run the causal impact analysis. Here, we use all the x that are not related to the intervention variable.

Parameters

- **df** (*pd.DataFrame*) – Dataframe to run the analysis on
- **x_columns** (*List[str]*) – All column names that can potentially be used as covariates for the counterfactual model

- **intervention_column** (*str*) – Name of the column on which the intervention has been performed
- **y_column** (*str*) – Target column on which we want to understand the effect of the intervention
- **start** (*List*) – Two elements defining the pre-intervention interval
- **end** (*List*) – Two elements defining the post-intervention interval
- **p_value_threshold** (*float*) – H0 that x does not Granger cause y is rejected when p smaller this threshold. Defaults to 0.05.

Returns causalimpact object

Return type object

6.4 TCN

6.5 Utilities

pyprocessta.utils.**is_regular_grid**(*series*)

For many analyses it can be convenient to have the data on a regular grid. This function checks if this is the case.

Parameters **series** (*pd.Series*) – pd.Series of datetime

Returns [description]

Return type bool

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX