D4.1 Pattern extraction methods - Initial Version

Lead Partner:	ATHENA
Version:	Final
Dissemination Level:	PU
Work Package:	WP4
Due date:	30/11/2021
Submission Date:	30/11/2021

Abstract:

Deliverable 4.1 presents the initial version of the pattern extraction methods, comprising a set of various utilities, as well as four initial tools for identifying patterns (segments), changepoints and areas of interest within RES time series. In particular, D4.1 presents utilities for performing preprocessing, statistics extraction, basic plotting, similarity comparisons, basic pattern extraction and learning actions on top of input time series. These utilities are exploited by more complete tools, for solving four initial pattern extraction tasks: (a) Interesting pattern discovery, (b) Time series segmentation, (c) Changepoint detection and (d) Deviation detection. Although the functionality of the four initial tools is demonstrated on top of two specific use cases, and the respective datasets on wind turbines and solar parks, the tools are designed in a configurable way, aiming to be applicable in various datasets and similar tasks.

ICT-H2020-51-2020 - Big Data technologies and extreme-scale analytics

Document Revision History

Date	Version	Author/Editor/Contributor	Summary of main changes / Status
28/09/2021	0.1.	Giorgos Giannopoulos	ToC
15/10/2021	0.2	Yannis Psarros, Alexandros Kalimeris, Panos Gidarakos	Contributions in pattern extraction utilities section
29/10/2021	0.3	Yannis Psarros, Alexandros Kalimeris, Panos Gidarakos	Contributions in pattern extraction tools section
05/11/2021	0.4	Giorgos Giannopoulos, Manolis Terrovitis	Contributions in requirements and system design sections
15/11/2021	0.5	Yannis Psarros, Alexandros Kalimeris, Panos Gidarakos, Giorgos Giannopoulos, Manolis Terrovitis	Version ready for internal review
23/11/2021	0.6	Nguyen Thi Thao Ho, Nicolas Girard	Internally reviewed version
29/11/2021	0.7	Yannis Psarros, Alexandros Kalimeris, Panos Gidarakos, Giorgos Giannopoulos	Final version prepared
30/11/2021	1.0	Giorgos Giannopoulos, George Papastefanatos, Manolis Terrovitis	Deliverable Submitted

Disclaimer

Copyright

1 Introduction

The goal of WP4 is to develop the methods and the tools that will support the cloud-based analytics functionality of the MORE platform, with emphasis on pattern extraction and detection, efficient querying and visualization. Task 4.1 develops pattern extraction methods for identifying interesting patterns (motifs), rare events (discords/anomalies), and changes or deviations (e.g. changepoints) from the expected behaviors of the time series. To this end, Task 4.1 exploits and extends the state of the art Matrix Profile[1] (MP) suite of algorithms, in combination with traditional time series analysis techniques and domain specific learning schemes. The goal of the task is to develop novel, scalable and effective algorithms and tools that will: (a) allow Renewable Energy Sources (RES) stakeholders to handle real-world problems in time series analysis and (b) be largely configurable and generalizable to similar time series tasks.

Deliverable 4.1 implements the initial version of the pattern extraction methods, focusing on forming the basis for the subsequent work in Tasks 4.1 and 4.2, as well as on providing initial input to Tasks 4.3 and 4.4, regarding the core functionalities that may utilize or build on. The work of Task 4.1/D4.1 starts from the study of the industrial use cases and the respective datasets on wind turbine and solar parks that were described in "Deliverable 5.1 Use Cases and Requirements" and takes into consideration the functional requirements that were defined in "Deliverable 5.2 Technical Platform Design". Based on these, our work is organized in two layers: (a) building an initial set of time series analysis utilities, regarding preprocessing, motif discovery, similarity comparisons, learning schemes, etc. that can be utilized within more complete pattern extraction processes, as well as in other tasks of WP4 (e.g. event detection, visualization, query engine) and (b) developing four initial tools that handle specific pattern extraction problems related to the prescribed RES use cases, but can also serve as more generic pattern extraction utilities in similar settings and datasets.

In particular, in this deliverable, the following four tools are presented: (a) **Pattern Discovery** tool, that focuses on identifying interesting patterns, defined as patterns that represent time series segments with different behaviors; (b) **Time series segmentation** tool, that learns its segmentation parameters from one or more initial, "golden truth" time series which is segmented on areas with different behaviors, and is able to segment a new input time series into areas with correspondingly different behaviors; (c) **Changepoint detection** tool, that examines a set of candidate time series segments that potentially contain changepoints and verifies/identifies which are more probable to actually contain a significant changepoint and (d) **Deviation detection** tool, that learns on a set of golden truth time series segments and tries to identify areas of the time series that present considerably deviating behavior.

D4.1 is organized as follows: Section 2 briefly presents the requirements that guided the development of our method, based on the Description of Work (DoW) of MORE and the use cases (D5.1). Section 3 presents a high-level architecture of the components comprising the initial version of the Pattern Extraction (PE) module of the MORE platform (see D5.2 for details). Section 4 describes the initial set of time series processing and analysis utilities, organized in submodules with specific functionalities. Section 0 describes the four initial pattern extraction tools, including development information and usage documentation. Section 6 presents a first-cut, qualitative assessment of the tools with respect to two of the prescribed use cases and reports the performed progress towards covering the respective functional requirements (D5.2). Section 7 concludes the deliverable.

[1] https://www.cs.ucr.edu/~eamonn/MatrixProfile.html

2 Requirements of the initial version

In this section, we briefly discuss the requirements that guided the development of the initial methods for pattern extraction. First, we enumerate a set of research goals that are prescribed in the DoW of MORE and regard the directions towards exploiting and extending state of the art solutions for pattern extraction. Then, we briefly enumerate two of the use cases from D5.1 that were examined at this stage of our work, towards building tools that can also solve the specific problems prescribed in them.

2.1 Research targets

According to the description of work of MORE, Task 4.1 aims at four complementary directions:

1. Extend the state of the art MP algorithms, building methods and tools, that can efficiently and effectively handle RES time series analytics problems.
2. Adapt the relevant to our setting MP profile algorithms so that they can directly be applied on summarized (modelled) time series, according to the models developed in WP2, in order to increase the efficiency and scalability of the pattern extraction methods.
3. Incorporate domain knowledge within the processing and the analysis of time series datasets, via Annotation Vector (AV) functions defined on top of the MP, to increase the effectiveness of the pattern extraction methods.
4. Develop learning schemes that will allow training on labelled, historical RES time series datasets in order to effectively identify patterns on new time series instances.

The initial work of the task has focused mainly on points 1. and 4., while contributions towards 3. have also been performed. In particular, the emphasis of our work was to study the use cases at hand and identify how we can exploit both traditional time series analysis techniques, as well as the MP algorithms in order to develop pattern extraction tools that can be of actual value for real-world RES analysis settings.

To this end, we developed extensions on the libraries of the MP, and built learning schemes that leveraged the baseline motif extraction functionality in order to solve more complex RES analytics problems (see Sections 5.1 and 5.2). Similarly, we developed preprocessing and analysis workflows and learning schemes on top of traditional time series analysis mechanisms to identify changepoints and deviations on RES time series (see Sections 5.3 and 5.4).

Additionally, we have currently integrated considerable domain knowledge (point 3.) from both wind turbine and solar panel stakeholder partners (Engie Laborelec and Inaccess respectively) into our methods, in the form of configurable preprocessing and pattern extraction functions (indicatively see Sections 4.2 and 4.5). Transforming and extending them as Annotation Vectors is a straightforward task, and part of our ongoing work.

With respect to point 2., we aim to develop methods that compute MP directly from data models obtained from ModelarDB (WP2), rather than from decompressed data. To this end, we have already made considerable progress by writing a wrapper to extract data models metadata from ModelarDB. We will study how to construct MP using data models in the next steps.

The aforementioned work has produced a set of core utilities and methods for pattern extraction that form the basis of Task 4.1, as well as WP4 in general. In our next steps, we are focusing on effectively handling point 2., i.e. integrating the developed methods with the summarization models of WP2, as well as on further extending our methods and building additional tools that will be able to handle various pattern extraction requirements of the RES field, including all prescribed use cases and datasets.

2.2 Link to the use cases

A significant initial step of our work focused on studying the defined use cases in D5.1 and exploring the available datasets provided by the use cases partners of the project on wind turbine and solar parks. Due to dataset provision limitations in the first months of the project, we focused on analysing two use cases, one for each RES: (a) Yaw misalignment in wind turbines and (b) Soiling of panels in solar parks. Next, we provide a brief description of each use case and explain how they drove the development of the initial pattern extraction methods.

2.2.1 Yaw misalignment

In this use case, the rotor axis of the turbine has formed an angle with the wind direction, leading to reduced performance. An angle of more than 5 degrees is considered significant in this use case and needs to be identified and corrected. The issue is that it is impossible to be able to constantly, accurately measure the angle of each turbine, since it requires specialized, costly equipment (Lidar) to be installed on every single turbine. Thus, the problem is currently handled by empirical monitoring of the performance of the turbines. In particular, the active power in relation with the relative angle that the nacelle makes with the wind is monitored. If there is a peak of production for a non-zero misalignment, this is an indication that there is a YM. However, this is not systematically performed because of the incertitude of the results provided by the several methodologies developed.

In the frame of this use case, we have been provided with a multidimensional time series measuring several variables (e.g. wind speed and direction, rotor rotation speed) including the active power of the turbine, for several turbines. Further, we have been provided labeled data for three of the turbines, regarding their measured yaw misalignment angle in specific time intervals, as presented in the table below.

Date of changes in 2018 | computed Yaw misalignment	Angle Before 02 August	Angle From 02 August	Angle From 04 October	Angle From 17 October	Angle From 24 October	Angle From 11 December and after
BEZ01	-6,7°	+4,8°	+9.4°	+/- 0°	-5.8°	+/- 0°
BEZ02	+2,7°	+2,7°	+4.9°	+/- 0°	-5.1°	+/- 0°
BEZ03	-2,2°	+3,4°	+7.5°	+/- 0°	-5.5°	+/- 0°

Table 1: Labels denoting changes in the yaw misalignment angle, in three turbines

This use case motivated us to examine how pattern extraction algorithms and, in particular motif discovery via the Matrix Profile suite of algorithms can be adapted/extended in the setting, in order to provide a more data driven solution. Consequently, we moved into two directions, as described next.

- Identify interesting patterns (motifs) that are potentially correlated mainly with aligned (or misaligned) areas of the time series. To identify them, we designed a two-step process, where a *model* is constructed (learnt) in one of the time series and then it is deployed in the remaining time series in order to assess its effectiveness. The model in this setting is comprised by one or more motifs that are with much higher frequency identified in only aligned (or misaligned) time series areas. If several parameterizations are examined and lead to specific motifs extracted from a “training” turbine and the same motifs are identified in a

"test" turbine, then the stakeholder might assume that the specific motifs (patterns) are significant for the problem of yaw misalignment and proceed to further studying them (their shapes, frequency and locations of appearance, etc).

- Directly identify segments of a time series that correspond to aligned or misaligned areas. To identify them, we exploit the semantic segmentation algorithm of the MP suite, which segments a time series so that the number of common motifs between different segments is minimized. Similarly, we extend this algorithm by defining a two-step scheme: at first, the optimal segmentation parameterization is identified on a "training" turbine, and then the segmentation model can be applied on new turbines to identify areas with different behaviors with respect to misalignment angle.

The two directions presented above led to the first two tools presented in this deliverable (see Sections 5.1 and 5.2). We need to emphasize that the yaw misalignment use case comprises the motivation for designing and implementing these two tools; the goal is that these tools can also be applicable in various similar settings and datasets.

2.2.2 Solar panel soiling

In this use case, solar panels are prone to gradually gathering soil, which leads to reduced performance. Soiling comprises a usually very slow and gradual phenomenon, thus making it difficult to detect. Similarly to the previous use case, it is rather costly to regularly inspect and wash all panels in all solar parks of an operator, while there is a trade-off that needs to be balanced between the actual power losses of a panel and the cost of its washing. Currently, the issue is handled by sample visual inspections and/or scheduled maintenance (i.e. washing twice a year). This is either a manual, highly inaccurate visual inspection, or a "blindly automated" scheduled maintenance. In both cases these are inherently inefficient processes.

In the frame of the use case, we have been provided with an extensive set of multidimensional time series measuring several variables (e.g. irradiance, temperature, humidity) including the active power of the solar panel, for several panels in several parks. Further, we have been provided labeled data for several hundreds of rain events and washing events, i.e. events that potentially correct/terminate a soiling event by cleaning the panel. These events (rains, washings) can be considered as candidate changepoints, since it is expected that the behavior (distribution) of the active power time series of the affected panel changes, as long as the event was effective (i.e. the rain was adequate, the washing was properly performed).

This use case motivated us to examine how time series analysis, in combination with machine learning schemes, can be used to identify patterns that help us: (a) verify that a candidate event was an actual changepoint, i.e. it indeed changed the behavior of the time series and (b) identify deviations from an expected behavior, i.e. identify areas in the time series where the active power has become unjustifiably lower, and thus soiling comprises a potential cause. Consequently, we implemented two individual tools, that handle individually the above two tasks, but can be combined as two steps of a soiling (gradual deviation) detection process, as described next.

- Changepoint detection (verification) focuses on verifying that specific given events (rains and washings) function as changepoints, i.e. induce a considerable change in the distribution of the active power time series of a solar panel, with respect to the rest measured variables. To do so, we define a train/validation/test procedure on top of each event, in order to approximate an "expected" time series behaviour after the event and compare it with the actual one. The final decision as to whether an event qualifies as changepoint is taken based on two errors: one measuring the quality of approximation (before the event) and one measuring the deviation between actual and expected behaviour (after the event).

- Deviation detection focuses on identifying areas of the time series where the lower than expected active power cannot be justified by other variables and, thus, can be attributed to soiling. To do so, we exploit verified changepoints output from the previous tools, to train regression models on periods where the solar panels are considered clean with high confidence. These models represent an ideal behaviour of the active power time series and are compared, via sliding windows, with the actual active power of the panel. When a lower (and relatively consistently degrading) actual active power, compared to the expected one is observed, a potential soiling effect is identified.

The two aforementioned tools are presented in detail in Sections 5.3 and 5.4. Similarly, the two tools are not strictly tied to the soiling use case; on the contrary, our aim is to make them as generic and configurable as possible. In particular, our next steps include assessing and tuning the specific tools also for another use case of the project, the small gain one, described in D5.2.

3 System design and components

In this section, we discuss the interconnection of the various implemented components. We initiate our discussion with a high-level description of the implemented utilities, which aim to support the Pattern Extraction module but can also be utilized in other tasks of WP4 (e.g. event detection, visualization, query engine). We then proceed to a high-level description of our current tools, and we discuss how these are built on top of the aforementioned utilities.

Figure 1 presents the high-level architecture of the PE module, which comprises the Pattern Extraction utilities and the Pattern Extraction tools. Dark blue boxes and arrows present the currently implemented functionality, while dashed ones the planned/ongoing functionality. For completeness, we also present the planned interactions with the other two directly relevant modules of WP4, the Visual Analytics module and the Complex Event Detection module.

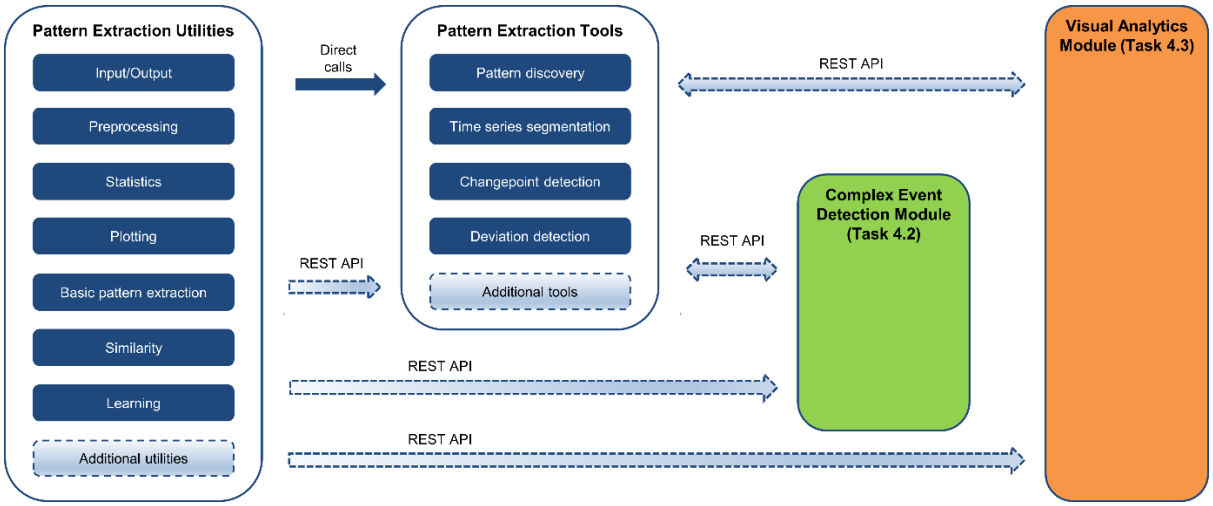

Figure 1: High level architecture of the Pattern Extraction module and its basic interactions with other WP4 modules

PE utilities are organized as submodules of the PE module. Each submodule corresponds to a different set of utilities, each one serving a common purpose. The *Input/Output submodule* contains all those utilities that are responsible for loading and storing of the data. The *Preprocessing submodule* contains utilities for the preprocessing of the input data, e.g. filtering of the data, filling out empty values, adding random noise. The *Statistics submodule* implements utilities for the computation of useful statistics either directly on the input data or in intermediate computations, e.g. error functions used in learning procedures. The *Plotting submodule* implements various utilities on plotting; this may concern plots illustrating the final output or plots with auxiliary information aiming to assist the user in optimizing the parameters in certain operations. The *Basic pattern extraction submodule* implements utilities for computing an MP index and post-processing an MP index in order to extract useful information on the data. The *Similarity submodule* contains utilities that implement nearest neighbor searching queries and distance computations. Finally, the *Learning submodule* implements functions which are used in learning tasks, e.g. fitting a regression model or extracting errors, and it relies on utilities from the Statistics submodule. All design decisions which led to the current status of the PE module are taken so that utilities are sufficiently generic and extendable, in order to satisfy requirements in the subsequent work in Tasks 4.1 and 4.2 (also Tasks 4.3, 4.4). In addition, we note that the PE module will be finalized as a complete API, in the strictly technical sense, in our direct next steps within Tasks 4.1 and 4.2.

Next, we enumerate the implemented tools and we discuss their interconnections with the PE utilities.

- Pattern discovery. The patterns we are currently examining are motifs, that is pairs (and more generically, groups) of sub-sequences of the time series that are very similar to each other. We are given a time series divided into a sequence of discrete segments, where each one is assigned to one of two classes, denoting a different behaviour (distribution). We remind that in the specific use case scenario that we are examining, the time series is segmented into regions that belong to one of two classes: “Yaw alignment” or “Yaw misalignment” (see also Section 2.2.1). The goal is to detect patterns which correspond to distinguishable characteristics of one of the classes. Our approach is based on the MP framework; in particular extracting information from an MP index. The tool relies on utilities from the following submodules: Input/Output, Preprocessing, Plotting, Basic pattern extraction, Similarity.
- Time series segmentation. The task here is to segment a time series. We assume that we are given a labelled time series with known changepoints and we employ an MP-based algorithm for semantic segmentation of time series, that computes changepoints in a “training” fashion where we optimize all parameters involved. Then, we use the optimal configuration to segment other time series. This tool relies on utilities from the following submodules: Input/Output, Preprocessing, Plotting, Basic pattern extraction.
- Changepoint detection (verification). Given a set of segments of the input time series, containing candidate changepoints, the task here is to verify whether each of those segments contains an actual changepoint. The main building block of our solution is a regression model. This tool relies on utilities from the Preprocessing submodule and the Learning submodule, which also depends on the Statistics submodule.
- Deviation detection. Given a time series and a set of segments indicating the “desired” behavior in that time series, the task is to detect segments where the behavior deviates much from the desired. Our approach is based on fitting a regression model to learn the desired behavior and then testing it on candidate segments. This tool relies on utilities from the Preprocessing submodule and the Learning submodule, which also depends on the Statistics submodule.

4 Pattern extraction utilities

In this section, we provide a thorough description of the implemented utilities which aim to fulfill the technical specification requirements of the Pattern Extraction module, as described in Deliverable D5.2. Those utilities include the computation of an index (in particular, the matrix profile) for the input time series, running variations of motif discovery algorithms in order to identify different types of patterns, including motifs, discords and changepoints, input/output utilities which assist in storing or loading the output of the aforementioned processing steps, preprocessing and filtering utilities, utilities for implementing learning schemes on the data and utilities for plotting the output or depicting intermediate computations.

Next, the implemented utilities, are organized and presented as part of the individual submodules of the PE module, as descripted in Section 3. The code is available on GitHub[2] accompanied by its documentation[3]. We note that the following include both re-utilization and extension of state of the art libraries, as well as development of new utilities, mainly regarding Sections 4.5-4.7. A more detailed description of our core development contributions is provided in Section 5.

4.1 Input/Output

4.1.1 Description

This group of utilities includes all functions which are responsible for loading and storing time series to the disk. Moreover, it includes functions for loading/storing necessary metadata which are produced by any of the methods of this module and can be further processed and, in particular, functions that load/save a matrix profile from/to the disk.

4.1.2 Implementation details

The functions implementing all relevant utilities are members of a python module, which can be found on GitHub in the following URL: https://github.com/MORE-EU/more-pattern-extraction/blob/main/modules/io.py. Our implementation builds on widely used libraries for numerical computations and basic data processing operations such as NumPy[4], Pandas[5], Matplotlib[6], h5py[7].

4.1.3 Main functions

- **load_df**: Loads a parquet file to a pandas DataFrame. Returns this pandas DataFrame.
- **load_mp**: Loads the Univariate/Multivariate Matrix profile which was saved from create_mp function (see Section 4.5.3) in a .npz file.
- **save_mdmp_as_h5**: Saves a multidimensional matrix profile as a pair of hdf5 files. Input is based on the output of the mstump algorithm[8].
- **load_mdmp_from_h5**: Loads a multidimensional matrix profile that has been saved as a pair of hdf5 files.

[2] https://github.com/MORE-EU/more-pattern-extraction/tree/main/modules

[3] https://more-eu.github.io/more-pattern-extraction/#documentation

[4] https://numpy.org/

[5] https://pandas.pydata.org/

[6] https://matplotlib.org/

[7] http://www.h5py.org/

[8] https://stumpy.readthedocs.io/en/latest/api.html#mstump

- **save_results**: Saves the results of a specific run in the directory specified by the results_dir and sub_dir_name parameters. The results contain figures that are created with an adaptation of the matrix profile foundation visualize function[9].

4.2 Preprocessing

4.2.1 Description

In this section, we discuss utilities regarding the handling and processing of the input data. Such functions handle filtering, resampling, adding noise, normalization and filling out missing values.

4.2.2 Implementation details

The functions implementing all relevant utilities are members of a python module, which can be found on Github in the following URL: https://github.com/MORE-EU/more-pattern-extraction/blob/main/modules/preprocessing.py. Our implementation builds on widely used libraries for numerical computations and basic data processing operations such as NumPy, Pandas, scikit-learn[10].

4.2.3 Main functions

- **change_granularity**: Given as input a time series and a parameter indicating the desired granularity, this function changes the granularity of the input time series and outputs the result. This is implemented as follows. The user is able to choose whether the output is allowed to have empty entries or not. In either case, we first resample the time series, and if empty entries are allowed, we return the outcome. If empty entries are not allowed, we apply interpolation in a memory-efficient manner by breaking the time series into chunks of a user-defined size. This aims to balance the memory usage between the memory-efficient resampling method and a potentially more costly interpolation method.
- **filter_col**: Given as input a pandas DataFrame, a name of a column, and an interval, the function outputs a DataFrame consisting of all rows in the DataFrame, whose values in the input column, belong to the input interval.
- **filter_dates**: Removes rows of the pandas DataFrame that are not in a given interval.
- **normalize:** This function transforms an input pandas DataFrame by rescaling values to the range [0,1]. This function works either for the whole DataFrame or for a specific column through the MinMaxScaler.
- **add_noise_to_series**: Given as input a pandas DataFrame, this function adds noise independently to each element and returns a new DataFrame. Uniform noise is applied.
- **add_noise_to_series_md**: Adds uniform noise to a multidimensional time series that is given as a pandas DataFrame.
- **filter_df**: Given as input a pandas DataFrame and multiple names of columns associated with intervals, this function outputs a DataFrame consisting of all rows in the DataFrame, whose values in the input columns, belong to the corresponding intervals.
- **chunker**: Dividing a DataFrame (or other array-like format) into pieces for better handling of RAM.

[9] https://matrixprofile.docs.matrixprofile.org/_modules/matrixprofile/visualize.html#visualize

[10] https://scikit-learn.org/stable/index.html

- **chunk_interpolate**: Data interpolation in a memory-efficient manner. Makes use of chunker, and interpolates each chunk individually before combining the results. This step is crucial for the complete data interpolation in very large datasets.
- **filter_dispersed**: Given as input a pandas DataFrame, a window size and a real-valued threshold, the function iteratively checks for each rolling window whether the ratio of the variance over the mean, i.e. the index of dispersion, for each column, is below the given threshold, and if this is the case it retains the last row of the window.
- **scale_df**: Scales each column of a pandas DataFrame to the [0, 1] range performing the min max scaling.

4.3 Statistics

4.3.1 Description

In this section, we discuss utilities regarding the computation of useful statistics about the dataset. This includes utilities that implement variations of standard error measures. Those measures are widely used in comparing time series, as well as ordered sequences of values in general and can be used, for example, in evaluating the performance of a regression model. It also includes helper functions used for ranking time series based on their error metrics (performance) and computing statistical measures such as the coefficient of multiple correlation.

4.3.2 Implementation details

The functions that technically implement all the utilities mentioned in this section are a part of a python module that can be found in the following URL: https://github.com/MORE-EU/more-pattern-extraction/blob/main/modules/statistics.py. Our implementations are built on already existing widely popular libraries for numerical operations, data processing and analysis such as numpy, pandas, and scikit-learn.

4.3.3 Main functions

- **mape**: This function computes the Mean Absolute Percentage Error measure between two given time series, with a slight variation to make it robust to values that are zero. This error measure is quite practical as it gives us a scale-independent and easily interpreted value.
- **mpe**: Similarly, this function computes the Mean Percentage Error between the 2 given time series. This function is useful when we are not interested only in the absolute error, but we want to examine if one time series has larger or smaller values than another one.
- **score**: Given two time series, one representing the true observations and one representing the predictions of a model, this function computes a set of values that measure how well the predicted time series matches the actual time series. The error measures it computes are the MAPE and MPE with the utilization of the above functions, as well as: R-squared, Mean Absolute Error and Mean Error.
- **get_top_deviations**: Given a NxM matrix that each one of the N rows contains M different scores of how well the Nth time series segment fits a model, this function finds the indices of the top most deviant segments according to the measure specified, for example MAPE, MPE etc.
- **multicorrelation**: Given a dataset of concurrently evolving time series (in the form of a pandas DataFrame) and specifying the target variable name corresponding to a column in that DataFrame, this function returns the multi-correlation index of the target column with all the other columns apart from the target.

4.4 Plotting

4.4.1 Description

In this section, we present functions that we use for plotting our results. We implement functions for plotting the matrix profile of a time series or the segmentation of a time series. We also include functions for plotting auxiliary information, e.g. plotting a graph that assists in selecting the best number of dimensions for the multivariate matrix profile, and helper functions for switching between different data formats.

4.4.2 Implementation details

The functions implementing all relevant utilities are members of a python module, which can be found on Github in the following URL: https://github.com/MORE-EU/more-pattern-extraction/blob/main/modules/plots. Our implementation builds on widely used libraries for numerical computations and basic data processing operations such as Matplotlib[11], NumPy, Pandas, Seaborn[12] and DTW[13].

4.4.3 Main functions

- **plot_knee**: Plots the minimum value of the matrix profile for each dimension. This plot is used to visually look for a 'knee' or 'elbow' that can be used to find the optimal number of dimensions to use.
- **predefined_labels**: Given a DataFrame and a list of sorted dates we find the indexes of those dates in our DataFrame.
- **convert_labels_to_dt**: Converting time series indexes to datetime indexes.
- **get_fixed_dates**: For sorted date data we get an array of indexes for a given list of dates.
- **plot_profile**: Plot the graph of a specific column for comparison and the corresponding Univariate Profile. A black arrow is pointing to the top motif in our set.
- **plot_profile_md**: Plot the multidimensional matrix profile. A black arrow is pointing to the top motif in our set.
- **plot_segmentation**: Plotting of the changepoints that we precomputed from segment_ts. The result consists of multiple graphs, one for each configuration. Later we use the Dynamic Time Warping distance in order to find a generalized distance between a list of dates and the precomputed change points. In the end, we save the ones that have the minimum distance up to a number the user wishes.

4.5 Basic pattern extraction

4.5.1 Description

In this section, we discuss the implementation of functions related to pattern extraction. This includes functions for computing a matrix profile, but also functions related to the post-processing of a matrix profile. The latter refers to functions for discovering motifs/discords or segmenting a time series.

[11] https://matplotlib.org/

[12] https://seaborn.pydata.org/

[13] https://pypi.org/project/dtw/#:~:text=Dtw%20is%20a%20Python%20Module%20for%20computing%20Dynamic,2.7-3.6%20and%20is%20distributed%20under%20the%20GPLv3%20license.

4.5.2 Implementation details

The functions implementing all relevant utilities are members of a python module, which can be found on Github in the following URL: https://github.com/MORE-EU/more-pattern-extraction/blob/main/modules/patterns.py. Our implementation builds on modules for matrix profile computation and processing such as pandas, NumPy, Matrixprofile[14] and Stumpy[15].

4.5.3 Main functions

- **pick_subspace_columns**: Given a multidimensional time series as a pandas Dataframe, keeps only the columns that have been used for the creation of the k-dimensional matrix profile.
- **to_mpf**: Converts a matrix profile to a form which is compatible with the matrix profile foundation library[16].
- **compute_mp_av**: Corrects a matrix profile by applying the complexity annotation vector[17].
- **pattern_loc**: Given a time series characterized by regions belonging to two different labels and a pattern i.e. a subsequence of consecutive measurements in the time series, the function returns the label name of the region that the pattern is contained in. If the pattern does not entirely lie in one of the labelled areas, then the function returns “both”.
- **calc_cost**: Given a time series characterized by regions belonging to two different labels and a pattern i.e. a subsequence of consecutive measurements in the time series, assign a cost to the pattern based on whether the majority of its occurrences are observed in regions of a time series that are annotated with the same binary label. The cost takes into account a possible difference in the total lengths of the segments. Return the label name of the region that the pattern is contained in, as well as the normalized number of occurrences.
- **calculate_motif_stats**: Calculates some useful statistics for the motifs found.
- **calculate_nn_stats**: Calculates some useful statistics for a pattern based on its nearest neighbors. That pattern is supposed to be found in another time series and is examined based on its neighbors on the current time series.
- **create_mp**: Wrapper function for computing a matrix profile. Given a time series in the form of a dataframe, the function computes the matrix profile and stores it in a user defined path. This works for univariate or multivariate DataFrames also with or without a Dask parallelization.
- **segment_ts**: Given a timeseries in the form of a dataframe, and a parameter indicating the number of change points, the function employs the FLUSS[18] algorithm to compute points that segment our timeseries for the observed behavioural change.

4.6 Similarity

4.6.1 Description

In this section, we are going to go over utilities related to similarity search refer to functions which are responsible for performing proximity queries to a dataset. Queries that can be handled include finding the k-nearest neighbors of a given vector or time series with respect to a standard distance function (e.g., Euclidean distance), in a pre-defined set of vectors/time series. One example of such a query is

[14] https://matrixprofile.docs.matrixprofile.org/

[15] https://stumpy.readthedocs.io/en/latest/

[16] https://matrixprofile.docs.matrixprofile.org/api.html#matrixprofile-utils-empty-mp

[17] https://matrixprofile.docs.matrixprofile.org/api.html#matrixprofile-transform-apply-av

[18] https://stumpy.readthedocs.io/en/latest/api.html#fluss

computing the top k-nearest neighbors of a given subsequence in a multivariate time series, among all (or a subset of) subsequences in the same time series.

4.6.2 Implementation details

The functions implementing all relevant utilities are members of a python module, which can be found on Github in the following URL: https://github.com/MORE-EU/more-pattern-extraction/blob/main/modules/similarity.py. The functions are built on functionalities found in the stumpy[19] and matrix-profile-foundation[20] libraries as well as some more generic functionalities of the well-known numpy[4]. Since we wanted to work with multidimensional time series as well, we extended some of the matrix-profile-foundation functions. The extensions can be found here: The implementation can be found here: https://github.com/MORE-EU/matrixprofile/blob/master/matrixprofile/algorithms/top_k_motifs.py. It should also be mentioned that the distances between the time series are calculated based on the multidimensional distance profile, as described at [YKK17].

4.6.3 Main functions

- **get_top_k_motifs**: Given a matrix profile, a matrix profile index, the window size and the DataFrame that contains a multidimensional timeseries, this function finds the top k motifs in the timeseries, as well as neighbours that are within the range *radius*min_matrix_profile_value* of each of the top k motifs.
- **find_neighbors**: Given a "query" time series of length w and a target time series of arbitrary length, this function searches for similar patterns in the target time series. Patterns with a distance less than *radius*min_dist* from the query are considered similar (neighbours) where radius is a user defined parameter. This function supports multidimensional queries and time series. The distance is calculated based on the multidimensional distance profile as previously mentioned. This function is implemented based on the univariate approaches of the matrix profile foundation library.
- **pairwise_dist**: Calculates the distance between two time series sequences. The distance is calculated based on the multidimensional distance profile. This function allows for the comparison of univariate and multidimensional sequences.

4.7 Learning

4.7.1 Description

In this set of utilities, we implement functions that build upon standard machine learning algorithms. In particular, functions contained in this module, handle tasks related to approximating a time series by means of a regression model. Those functions are needed, for example, when we are trying to discover trends or changepoints.

4.7.2 Implementation details

The functions implementing all relevant utilities are members of a python module, which can be found on Github in the following URL: https://github.com/MORE-EU/more-pattern-extraction/blob/main/modules/learning.py. Our implementation builds on widely used libraries for

[19] https://github.com/TDAmeritrade/stumpy

[20] https://github.com/matrix-profile-foundation/matrixprofile

numerical computations and basic data processing operations, namely NumPy, Pandas and scikit-learn.

4.7.3 Main Functions

- **predict**: Applies a regression model to predict values of a dependent variable for a given pandas DataFrame and given features, given as columns of the DataFrame.
- **fit_linear_model**: Fits a regression model on a given pandas DataFrame, and returns the model, the predicted values and the training error values. Applies Ridge Regression with polynomial features. The training error values returned by the function are the following: the coefficient of determination "R-squared", the mean absolute error, the mean error, the mean absolute percentage error, and the mean percentage error.
- **get_line_and_slope**: Fits a line on the 2-dimensional graph of a regular time series, defined by a sequence of real values. Returns information about the line.
- **train_on_reference_points**: Trains a regression model on a training set defined by segments of time series which is given as a pandas DataFrame. These segments are defined by a set of starting points and a parameter indicating their duration. In each segment, one subset of points is randomly chosen as the training set and the remaining points are used as part of the validation set.
- **predict_on_sliding_windows**: Given a regression model, predicts values on a sliding window in a pandas DataFrame and outputs a list of error values (one list of error values for each window), the list of predictions and the list of windows.
- **changepoint_scores**: Given as input a pandas DataFrame and a reference interval where a changepoint may lie, trains a regression model in a time window before the reference interval, validates the model in a window before the reference interval and tests the model in a window after the reference interval. Returns the predicted values for each of the three intervals, and associated error values.

5 Pattern extraction tools

The Pattern Extraction module (PE) is responsible for analyzing the RES time series data and extracting meaningful patterns in the form of motifs, discords, changepoints, etc. The module will incorporate the following core functionalities: (a) Calculating indexes (e.g., the Matrix Profile) on top of the time series that holds distance information between all different parts of a time series, considering specific but parameterizable pattern lengths; (b) Running variations of motif discovery algorithms in order to identify different types of patterns, including motifs, discords and changepoints; (c) Deploying traditional time series analytics, as well as ML algorithms (where meaningful) and jointly exploiting with motif discovery algorithms in the pattern extraction process; (d) Storing the output of the aforementioned processing (indexes, patterns) in the Patterns Repository. For the development process, several python libraries that calculate the Matrix Profile and perform further processing/analytics on it will be used. In particular: STUMPY[15], MatrixProfile[14] and, SCAMP[21] and pyscamp[22].

Next, we present the initial four pattern extraction tools developed in the frame of D4.1. The tools are available on GitHub[23] in the form of Jupyter notebooks.

We note that the development of the PE module is a continuous process through the course of the project. Thus, what we report in this deliverable is a snapshot of the implemented tools at the time of submission of the deliverable. Although the reported notebooks will be maintained in our GitHub repository for reference and consistency with D4.1, updated versions of them will be published alongside, during the next months, incorporating improvements and extensions of the implemented methods. Further, different variations of the tools may be developed, in order to be able to solve similar RES use cases. All the aforementioned extensions will be reported in "D4.2 Pattern extraction methods - Final Version".

5.1 Pattern discovery

5.1.1 Description

The pattern discovery tool aims to detect interesting patterns in time series with labeled time intervals. In particular, we are given a time series which is divided into a sequence of discrete segments, and each one is assigned to one of two classes. The goal is to detect patterns which correspond to distinguishable characteristics of one of the classes. To discover such patterns, we employ the MP framework. We compute the matrix profile to find the top-k motifs and, for each one of those motifs, we compute all neighboring patterns lying within its nearest neighbor distance, multiplied by a user-defined factor. For each pattern, among the top-k motifs, we calculate a cost based on the number of its near neighbors, which are observed in segments of either label: if all near neighbors are annotated with the same label then the cost is 0, while if occurrences of either label are equally observed then the cost is 1. The intuition is that patterns that are representative of their respective class, should mostly appear in the time regions corresponding to that class. Thus, the lower the value of the cost function the more likely for a pattern to contain meaningful information for the behavior of the labeled segment.

[21] https://github.com/zpzim/SCAMP

[22] https://pypi.org/project/pyscamp/

[23] https://github.com/MORE-EU/more-pattern-extraction/tree/main/notebooks

5.1.2 Implementation information

The pattern discovery tool is implemented in Python, currently as a Jupyter notebook linked to modules developed in the scope of Section 4. The notebook can be found in https://github.com/MORE-EU/more-pattern-extraction/blob/main/notebooks/interesting_patterns.ipynb. For the discovery of the various patterns, the Matrix Profile framework has been employed. More specifically, we used the STUMPY[19] library for the calculation of the multidimensional Matrix Profile and then after arranging its output in the appropriate format, we used the matrix-profile-foundation[20] library along with some extensions to support some functionalities on multidimensional time series such as the extraction of the top-k motifs and nearest neighbors. Our extensions can be found at our fork[24] . In greater detail, at[25] we extended the function top_k_motifs to work in a multidimensional setting. It can calculate the top-k motifs as well as a user defined number of their nearest neighbors using the multidimensional distance profile that is described at [YKK17].

Other extensions of the matrixprofile library include the extension of the function make_complexity_av that is available here[26], which is used to compute the "complexity" annotation vector from the matrixprofile library in a multidimensional fashion. Lastly, we have extended some visualization[27] functionalities of the matrixprofile foundation library that can work for multidimensional time series.

Under the scope of this tool, several other functions have been developed, including: (a) The find_neighbors[28] function that can be used to perform a similarity search and find the nearest neighbors of a given query that exist in another given time series, using the multidimensional Distance Profile; (b) A function that calculates the pairwise distance of two time series using the multidimensional distance profile[29]. Several other functionalities for calculating and reporting statistics and metrics for patterns have also been implemented and can be found at[30]. Finally, for efficiently managing, preprocessing, and performing numerical operations on the data, the pandas, scikit-learn and numpy libraries have been used. For the purposes of demonstrating a prototype version of this tool as well as during the development process, jupyter-notebooks[31] have also been used.

In general, the main contribution of this tool can be summarized as the combination of all the above functionalities for the introduction of a step-by-step method that helps detect "meaningful" patterns.

That method can be briefly described by the following steps:

1. Calculation of the multidimensional matrix profile.
2. Computation of the top-k motifs in a multidimensional fashion as well as their neighbours.
3. Characterization of those top-k motifs with a cost that is associated with how skewed the distribution of their neighbours is, over the labelled regions of the time series.
4. Validation of the results in other similar time series.

[24] https://github.com/MORE-EU/matrixprofile

[25] https://github.com/MORE-EU/matrixprofile/blob/master/matrixprofile/algorithms/top_k_motifs.py#L316

[26] https://github.com/MORE-EU/matrixprofile/blob/master/matrixprofile/transform.py#L135

[27] https://github.com/MORE-EU/matrixprofile/blob/master/matrixprofile/visualize.py

[28] https://github.com/MORE-EU/more-pattern-extraction/blob/main/modules/similarity.py#L31

[29] https://github.com/MORE-EU/more-pattern-extraction/blob/main/modules/similarity.py#L106

[30] https://github.com/MORE-EU/more-pattern-extraction/blob/main/modules/statistics.py

[31] https://jupyter.org/

5.1.3 User guide

5.1.3.1 Installation

Python 3.8.5+ is required. For a guide on how to install python, one can visit https://www.python.org/.

In order to check the version of python installed in the system, one can run:

$ python -- version

One should start by downloading the latest version of our source code. If git is installed, this step can be implemented as follows:

$ git clone https://github.com/MORE-EU/more-pattern-extraction --recursive

One new folder under the name more-pattern-extraction will be created. Let <path> be the absolute path to that folder, i.e., the new folder is <path>/more-pattern-extraction.

It is preferable to use a virtual environment, which will host all necessary libraries, as follows:

$ python3 -m venv <path/to/new/virtualenv/>

$ source <path/to/new/virtualenv/>/bin/activate

Now, with the virtual environment activated one can proceed to install the required libraries.

The required libraries are:

- pandas - 1.3.4
- h5py - 2.10.0
- matplotlib - 3.4.3
- numba - 0.53.1
- numpy - 1.21.1
- scikit_learn- 1.0
- pytest-timeit==0.3.0

The above libraries are included in the requirements.txt file. All libraries (including those needed for the rest of the tools to work) can be installed with the following command:

$ pip install –r pip_requirements.txt

An extended version of the matrixprofile[Error! Bookmark not defined.] library is also required for this tool. The l ibrary is included as a sub-module in our github repository[24] and can be installed as follows:

First, navigate to the root of the repository with:

$ cd <path>/more_pattern_extraction/

Then, navigate to the sub-module folder:

$ cd matrixprofile/

Finally install the matrix profile library from the sub-module source files:

$ pip install -e .

In addition, jupyter-notebook must also be installed by executing the following command:

$ pip install pip install notebook

For a more detailed guide on how to install jupyter-notebook and other Jupyter services, one can visit https://jupyter.org/install.html.

We should also note that if one decided to use a virtual environment, they have to install jupyter-notebook, according to the instructions, inside the activated virtual environment.

Now in order to run the notebook implementing this tool, one first needs to start jupyter-notebook (Note, if a virtual environment is used start the notebook inside the activated environment):

$ jupyter-notebook

This will open a new tab in the default internet browser where one has to select and run the following Jupyter notebook file:

<path>/more-pattern-extraction/notebooks/interesting_patterns.ipynb

5.1.3.2 Configuration options

For the smooth execution of this notebook's functions, a list of configuration options is declared. These fields must be the following:

- *granularity*: The constant time interval between two consecutive observations
- *start*: The start of a datetime interval
- *end*: The end of a datetime interval
- *k*: We extract this parameter by plot_knee which plots the minimum value of the matrix profile for each dimension. This plot is used to visually look for a 'knee' or 'elbow' that can be used to find the optimal number of dimensions to use
- *m*: The subsequence window size is used for finding patterns lengths (minutes, days etc.) in our DataFrames. An integer. It is mostly used in the creation of the profile.
- *ezones*: A list of exclusion zones to exclude near or trivial motifs.
- *radii*: The radii variable is used to associate a neighbor by checking if the neighbor's distance is less than or equal to dist * radii
- *topk*: Number of top motifs user wishes to find.
- *max_neighbors*: The number of neighbors user wishes the motif to have.
- *m1*: a datetime filter.
- *segment_labels*: Filtering of the neighborhood's regions.

5.1.3.3 Deployment instructions and options

In this notebook the user will read a .csv or .parquet file in order to create a pandas DataFrame containing the time series as we can see in the cell [2] of the notebook. Providing only the relative or absolute path to the file.

Then the multivariate matrix profile is saved as a .h5 file. More specifically the matrix profile distances and indices are saved to the disk in a user defined path found in cell [4]. Afterwards, we get the subspace of dimensions in which the k-dimensional matrix profile was computed. We perform motif search and save visual, statistical results [8], [9]. Then we validate our results in similar datasets contained in DataFrames [10]. In cells [11-15], the user loads the .csv files that summarize the statistics obtained from the previous cells and then queries for the closest matches of the top patterns on the new datasets used for validation. Finally, the results of the validation are written in .csv format to the disk.

5.2 Time series segmentation

5.2.1 Description

The time series segmentation tool aims to discover evident behavioral changes. In particular, we are given a labeled time series for training and at least one time series where we would like to apply our model. The task is to discover points which split the time series into areas with different behavior. In the current setting, similarly to the previous tool, we consider two classes/areas with different behavior: *aligned* and *misaligned*. To discover such points, we employ the matrix profile framework and in particular the FLUSS algorithm [GDY+17] for time series segmentation in a multi-dimensional way. We use the input labelled time series to optimize all parameters associated with the FLUSS algorithm. This "training" step is implemented by exhaustively enumerating a suitable set of configurations. We keep the configuration that minimizes the Dynamic Time Warping distance between the change points returned by FLUSS and the ground truth changepoints. Then we advance to the deployment step by employing the extracted model over similar unlabeled time series and assess whether areas with different behaviors can also be identified on them.

5.2.2 Implementation information

The time series segmentation tool is implemented in Python, currently as a Jupyter notebook linked to modules developed in the scope of Section 4. The notebook can be found in:

https://github.com/MORE-EU/more-pattern-extraction/blob/main/notebooks/semantic_segmentation.ipynb. Our main technical contribution is the adaptation of the FLUSS algorithm, that performs semantic segmentation on a time series, into a learning scheme that is able to learn an appropriate configuration of the algorithm for the specific task. This way, the algorithms can learn on the behavior of, e.g. a turbine and be able to segment new turbines into aligned/misaligned areas.

In general, the main contribution of this tool can be summarized as the combination of all the above functionalities for the introduction of a step-by-step method that helps behavioral changes. That method can be briefly described by the following steps:

1. Calculation of the multidimensional matrix profile.
2. Computation of the Fluss algorithm for the semantic segmentation of the time series taking into account several variables of the multidimensional time series.
3. Characterization of the change points returned by step 2 with a distance cost that is associated with how far those points laying from fixed points (labelled regions) and selection of the optimal semantic segmentation configuration regarding this cost.
4. Deployment and validation of the effectiveness of the learned segmentation model in other time series.

5.2.3 User guide

5.2.3.1 Installation

Python 3.8.5+ must be installed in the system. For a guide on how to install python, one can visit https://www.python.org/. In addition, jupyter-notebook must also be installed. For a guide on how to install jupyter-notebook, one can visit https://jupyter.org/install.html.

The following libraries are required:

- Pandas - 1.3.4
- Argparse - 3.2
- Matplotlib - 3.4.3
- Matrixprofile - 1.1.10

- Numpy - 1.21.1
- Seaborn - 0.11.2
- Stumpy - 1.09.02
- Dtw-Python – 1.01.10

The above libraries are included in the pip_requirements.txt file. All libraries (including those needed for the rest of the tools to work) can be installed with the following command:

$ pip install –r pip_requirements.txt

In order to check the version of python installed in the system, one can run:

$ python --version

It is sometimes preferable to use a virtual environment, which will host all necessary libraries, as follows:

$ virtualenv -p `which python` <path/to/new/virtualenv/>

$ source <path/to/new/virtualenv/>/bin/activate

In addition, jupyter-notebook must also be installed. Now, one can install all required libraries as described above. The next step consists of downloading the last version of our source code. If git[32] is installed, this step can be implemented as follows:

$ git clone https://github.com/MORE-EU/more-pattern-extraction

A new folder under the name more-pattern-extraction will be created. Let <path> be the absolute path to that folder, i.e. the new folder is <path>/more-pattern-extraction. Now in order to run the notebook implementing this tool, one first needs to start jupyter-notebook:

$ jupyter-notebook

This will open a new tab in the default internet browser where one has to select and run the following Jupyter notebook file:

<path>/more-pattern-extraction/notebooks/semantic_segmentation.ipynb

5.2.3.2 Configuration options

For the smooth execution of this notebook's functions, a list of configuration options is declared. These fields must be the following:

- *motif_len*: The subsequence window size is used for finding patterns lengths (minutes, days etc.) in our DataFrames. An integer. It is mostly used in the creation of the profile. [5]
- *granularity*: The constant time interval between two consecutive observations. [4]
- *L*: The subsequence length that is set roughly to be one period length. This is likely to be the same value as the motif_len, used to compute the matrix profile and matrix profile index. L is a list of integers. The L is a factor which excludes change point detection from the 'edges' of the DataFrame. [9]

[32] https://github.com/git-guides/install-git

- regions: Number of segments that the user is going to divide the space. This has to do with the sensitivity of the segment_ts algorithm. The more the segments the user wants to divide the more sensitive will perform in order to find useful locations of changing behavior. [9]
- *excl_factor*: It replaces the Arc Curve with 1 given by segment_ts depending of the size of L multiplied with an exclusion Factor. [9]
- *top_seg*: Number of the best plots user wants to save. [10]
- *k_optimal*: We extract this parameter by plot_knee which plots the minimum value of the matrix profile for each dimension. This plot is used to visually look for a 'knee' or 'elbow' that can be used to find the optimal number of dimensions to use. [9]
- *fixed_dates*: this an input but it can be defined by the user A list of sorted dates. It should be useful for comparing change point behavior between output points from algorithms and label user defined data. [10]

5.2.3.3 Deployment instructions and options

In this notebook the user will read a .csv/ .parquet file in order to create a pandas DataFrame/timeseries as we can see in [2]. Providing only a "path". In the next step we preprocess our time series [4]. Then a multi-dimensional Time Series is created and saved as a .npz file [5]. In our case we have loaded a precomputed one [6]. Then the optimal variables are extracted in order to be used for multidimensional search [8] of those points [9]. Later a user desired number of best plots of semantic segmentation tool are saved to the disk as .png files [10], as well as the fitted segmentation on similar DataFrames [14].

5.3 Changepoint detection

5.3.1 Description

The changepoint detection tool aims to verify candidate changepoints in a time series. Given a set of segments of the input time series, corresponding to periods of time where a changepoint may have occurred, our tool essentially ranks those segments with respect to the possibility of containing a changepoint. This can be seen as a classification task, where segments are classified with respect to whether they contain a changepoint or not. To solve this, we investigate each individual segment as follows: i) we fit a regression model on the data contained in a time interval before the assumed changepoint, ii) we validate it on a time interval before the assumed changepoint, and iii) we make predictions on a time interval after the assumed changepoint. If the validation error is small and the predicted values after the given segment deviate significantly from the real ones, then we conclude that one changepoint lies in the given segment. The intuition is that if the validation error is small, then this means that the regression model has successfully learned the behavior of the time series (w.r.t. to a given target variable) before the assumed changepoint. Hence, if one changepoint lies in the given segment, it learnt the model should fail on predicting values after that segment. To estimate the deviation between the predicted and the real values, we use standard metrics such as i) the mean absolute percentage error, ii) the mean absolute error, iii) the mean error, iv) the mean percentage error, and v) the coefficient of determination "R squared". Metrics iii) and iv) are especially useful when we are specifically looking for changepoints with either positive or negative effect.

5.3.2 Implementation information

The changepoint detection tool is implemented in Python, currently as a Jupyter notebook linked to modules developed in the scope of Section 4. The notebook can be found in https://github.com/MORE-EU/more-pattern-extraction/blob/main/notebooks/changepoint_detection.ipynb.

Our main technical contribution in this tool is the implementation of a function that provides a way to extract information on the possibility that a certain segment of the input time series contains one changepoint. We refer to function changepoint_scores[33]. The function essentially defines a training set of points lying before the assumed changepoint, a validation set lying between the training set and the assumed changepoint and a test set lying after the assumed changepoint. The training set is then used to train a regression model, which is later validated using the validation set. The intuition is that if the model has small validation error, then the behavior of the target variable before the changepoint has been correctly learned. Hence, if the model has small validation error and large test error, then it should be the case that there exists a changepoint. The function returns threefold information: one part that contains the prediction and a vector of errors for the training set, a second part that contains the prediction and a vector of errors for the validation set and a third part containing the prediction and a vector of errors for the test set. We stress two positive aspects in the design of this function. First, it simply relies on the existence of a good regression model. This is a problem which has been widely studied in the literature, and depending on the context and the type of data, different algorithms may suit better. This leaves space for further improving and finetuning our code in the future. Second, the information returned by the function allows for further post-processing in order to evaluate the input segments. Hence, while the function returns a set of important error values, there are many different ways of evaluating the segment, based on them.

In particular, in our current implementation, in order to detect segments which are more likely to contain a changepoint, we first filter them with respect to their associated validation error; only segments with error below a user-defined threshold survive this step. For this filtering step, we employ the mean absolute percentage error, which is defined as the mean of absolute percentage differences between predictions and observations. The goal here is to ensure that the regression model has successfully learned the behavior of the target variable before the changepoint. Then, we use the validation and the test error returned by the changepoint_scores function to rank all remaining segments. In particular, we compute the percentage difference between the mean error during testing and the mean error during validation. Formally, we compute (*mean testing error – mean validation error)/(absolute value of mean validation error).* This score function has the property that it is sensitive to whether the changepoint affects the target variable positively or negatively. Segments which contain changepoints that positively affect the target variable are the ones that are more likely to achieve a high score. This is a desired property in the specific scenario studied in our notebook.

We use pandas and NumPy for basic handling and numerical operation of the data. We rely on scikit-learn for utilities related to regression, and we use matplotlib for visualization purposes.

5.3.3 User guide

5.3.3.1 Installation

Python 3.8.5+ must be installed in the system. For a guide on how to install python, one can visit https://www.python.org/. The following libraries are required:

- matplotlib==3.4.3
- pandas==1.3.4
- NumPy==1.21.1
- scikit_learn==1.0

The above libraries are included in the pip_requirements.txt file. All libraries (including those needed for the rest of the tools to work) can be installed with the following command:

[33] https://github.com/MORE-EU/more-pattern-extraction/blob/main/modules/learning.py

$ pip install –r pip_requirements.txt

In order to check the version of python installed in the system, one can run:

$ python --version

It is sometimes preferable to use a virtual environment, which will host all necessary libraries, as follows:

$ virtualenv -p `which python` <path/to/new/virtualenv/>

$ source <path/to/new/virtualenv/>/bin/activate

In addition, jupyter-notebook must also be installed. Now, one can install all required libraries as described above. The next step consists of downloading the last version of our source code. If git[34] is installed, this step can be implemented as follows:

$ git clone https://github.com/MORE-EU/more-pattern-extraction

A new folder under the name more-pattern-extraction will be created. Let <path> be the absolute path to that folder, i.e. the new folder is <path>/more-pattern-extraction. Now in order to run the notebook implementing this tool, one first needs to start jupyter-notebook:

$ jupyter-notebook

This will open a new tab in the default internet browser where one has to select and run the following Jupyter notebook file:

<path>/more-pattern-extraction/notebooks/changepoint_detection.ipynb

5.3.3.2 Configuration options

We present a list of user-defined parameters for our tool. In the current version, those parameters can be easily modified by means of changing a variable assignment in the notebook. We plan to integrate a more interactive, visual way of assigning values to those parameters, in a future version. A list of parameters, along with instructions on how to assign new values to them follows:

- *w1*: The number of days defining the training set. This variable can be reassigned in cell [10].
- *w2*: The number of days defining the validation set. This variable can be reassigned in cell [10].
- *w3*: The number of days defining the test set. This variable can be reassigned in cell [10].
- feats: A list of names of columns of the input pandas DataFrame, corresponding to the feature variables. Normally, those variables should be highly correlated with the target variable (see next item). Used in regression. This variable can be reassigned in cell [11].
- *target*: A name of a column of the input pandas DataFrame corresponding to the dependent variable, i.e. the input variable whose behavioral changes are being detected. Used in regression. This variable can be reassigned in cell [11].
- *val_error_column*: Validation error which will be used in the final evaluation of a segment, i.e. 0=r_squared, 1=mae, 2=me, 3=mape, 4=mpe. This variable can be reassigned in cell [13].
- *test_error_column*: Test error which will be used in the final evaluation of a segment, i.e. 0=r_squared, 1=mae, 2=me, 3=mape, 4=mpe. This variable can be reassigned in cell [14].

[34] https://github.com/git-guides/install-git

- *thrsh*: Threshold on the validation error. Only segments with validation error below that threshold will be considered. This variable can be reassigned in cell [13].
- *no_segments*: The number of higher ranked segments which will be printed. This variable can be reassigned in cell [14].

5.3.3.3 Deployment instructions and options

The tool supports input timeseries in .csv file format. In particular the input time series will be read as a (multi)variate pandas dataframe. This can be seen in cells [2] and [3], where the user can define the path where the input .csv file is located. The input segments which will be evaluated as to whether they contain a changepoint or not are then read in cell [5], extracted in cell [6], and filtered in cell [8]. In the particular use case demonstrated in our notebook, those segments are exported from a different file and stored in two lists: one list containing the dates where such a segment starts, and one list containing each date where the corresponding segment ends. Those two lists are named dates_rain_start and dates_rain_stop and can be found and modified in cell [6]. Next, those lists are filtered and produce two new lists named dates_rain_start_filtered and dates_rain_stop_filtered. This step can be found and modified in cell [8]. Finally, the output is printed in cells [15] and [16].

5.4 Deviation detection

5.4.1 Description

The deviation detection tool aims to detect segments of a time series where the behavior of a given target variable deviates from that of the "expected" or the "desired". We are given as input a time series, along with a set of reference points and a target variable. The reference points aim to define reference segments, i.e. segments of the time series where the target variable behaves as desired. For example, reference points may be the starting points of reference segments, whose length may be unknown or requires further investigation. In other words, the target variable certainly behaves "desirably" for a period right after the given points. Our approach on this task is based on utilizing regression machine learning models. We fit a regression model on the segments around (before or after depending on the individual use case scenario) the reference points, aiming to capture the "target" behavior of the target variable. Then, we compare the values of the trained model with the real values in a sliding window passing through the whole time series and detect the segments that deviate the most. To implement this comparison, and rank the various segments, we use a diverse set of standard metrics: i) the mean absolute percentage error, ii) the mean absolute error, iii) the mean error, iv) the mean percentage error, and v) the coefficient of determination "R squared".

5.4.2 Implementation information

The deviation detection tool is implemented in Python, currently as a Jupyter notebook linked to modules developed in the scope of Section 4. The notebook can be found in: https://github.com/MORE-EU/more-pattern-extraction/blob/main/notebooks/deviation_detection.ipynb. For learning the desired behavior of the time series, so that we can afterwards detect deviations from it, linear regression functionalities from scikit-learn have been utilized. For the purpose of training, validating and actually performing inference over the time series several functions have been implemented such as train_on_reference_points[35] and predict_on_sliding_windows[36] that, along with other functions we implemented as well as imported ones from scikit-learn, implement the training and predicting

[35] https://github.com/MORE-EU/more-pattern-extraction/blob/main/modules/learning.py#L91

[36] https://github.com/MORE-EU/more-pattern-extraction/blob/main/modules/learning.py#L137

functionalities. The predict_on_sliding_windows function returns scores of how well a prediction fits the actual time series and those results can afterwards be passed through the get_top_deviations[37] function that calculates the n (user defined number) most deviant segments depending on the score the user specifies. Additionally, for efficiently managing, preprocessing, and performing numerical operations on the data, the pandas, scikit-learn and numpy libraries have been used. Additionally, for visualizing the time series and some results, the matplotlib library has been used. For the purposes of demonstrating a prototype version of this tool as well as during the development process, the jupyter-notebooks have also been deployed.

Our main technical contribution is the formulation of a generally applicable method that highlights segments of a time series that differ from a "desirable" learned model. Assuming that there are available reference points to regions where the time series in question exhibits the desired behavior, we implement functionalities that can learn and then validate their performance on those segments. In greater detail, in the train_on_reference_points function, we gather all the points in a user-defined window around all the reference points and we train our model on 80% of that data and keep the rest 20% to validate how well the model learned the behavior around those segments. Then, using the model that we have confirmed that is performing sufficiently well on the validation set, we slide a parametrizable window along the time series. On the individual windows that emerge from the previous process, we predict what the time series should be, according to the output of the model. Finally, if the actual measurements of the time series differ significantly from the ones that come from the model, then that segment is considered as a possible segment where deviation from the desired behavior is observed. It should also be noted that our method follows a modular design, meaning that the model used for predicting can easily be replaced with another to effectively adapt to a specific use case. Similarly, the scores returned from predict_on_sliding_windows can be post-processed according to the needs of each use case in order to get satisfactory results. On a final note, we also implement a functionality (get_line_and_slope[38]) that fits a line on the time series produced by subtracting the predicted from the actual values of the time series in the detected segments, smoothed by a rolling mean. The slope of that line can be employed as an indicator of how rapidly and in what direction the actual time series deviates from the model that represents the desired behavior.

In order to rank segments with respect to how much they deviate from the desired behavior, we use the mean percentage error which is returned from predict_on_sliding_windows and it is defined as the mean, over all points in a given window, percentage of difference of the predicted value from the true value. This is an error function which is sensitive to the sign of the average difference. This is desirable in the use case that we handle in our current implementation, because we are specifically looking to detect time periods where the real values of the target variable are generally lower than desired. Moreover, we are also using the slope returned by get_line_and_slope, which fits a line on the points produced by subtracting the predicted from the real values. This slope aims to capture the rate of change of difference between the real and the predicted values, which is also desirable in scenarios where deviation is expected to occur in a trending fashion like in the current use case. The resulting score function is simply the product *mean percentage error, min(0 slope), which implicitly filters out all segments with positive slope*.

5.4.3 User guide

5.4.3.1 Installation

Python 3.8.5+ is required. For a guide on how to install python, one can visit https://www.python.org/.

[37] https://github.com/MORE-EU/more-pattern-extraction/blob/main/modules/statistics.py#L67

[38] https://github.com/MORE-EU/more-pattern-extraction/blob/main/modules/learning.py#L69

In order to check the version of python installed in the system, one can run:

$ python -- version

One should start by downloading the latest version of our source code. If git is installed, this step can be implemented as follows:

$ git clone https://github.com/MORE-EU/more-pattern-extraction --recursive

One new folder under the name more-pattern-extraction will be created. Let <path> be the absolute path to that folder, i.e., the new folder is <path>/more-pattern-extraction.

It is preferable to use a virtual environment, which will host all necessary libraries, as follows:

$ python3 -m venv <path/to/new/virtualenv/>

$ source <path/to/new/virtualenv/>/bin/activate

Now, with the virtual environment activated one can proceed to install the required libraries.

The required libraries are:

- pandas - 1.3.4
- h5py - 2.10.0
- matplotlib - 3.4.3
- numba - 0.53.1
- numpy - 1.21.1
- scikit_learn- 1.0

The above libraries are included in the requirements.txt file. All libraries (including those needed for the rest of the tools to work) can be installed with the following command:

$ pip install –r pip_requirements.txt

An extended version of the matrixprofile[Error! Bookmark not defined.] library is also required for this tool. The l ibrary is included as a sub-module in our github repository[24] and can be installed as follows:

First, navigate to the root of the repository with:

$ cd <path>/more_pattern_extraction/

Then, navigate to the sub-module folder:

$ cd matrixprofile/

Finally install the matrix profile library from the sub-module source files:

$ pip install -e .

In addition, jupyter-notebook must also be installed by executing the following command:

$ pip install pip install notebook

We should also note that if one decided to use a virtual environment, they have to install jupyter-notebook, according to the instructions, inside the activated virtual environment.

Now in order to run the notebook implementing this tool, one first needs to start jupyter-notebook (Note, if a virtual environment is used start the notebook inside the activated environment):

$ jupyter-notebook

This will open a new tab in the default internet browser where one has to select and run the following Jupyter notebook file:

<path>/more-pattern-extraction/notebooks/deviation_detection.ipynb

5.4.3.2 Configuration options

- *start*: The start of the hourly window we extract from each day
- *end*: The start of the hourly window we extract from each day i.e.,
- *w_train*: The number of days defining the training set
- *ref_points*: A list containing the starting date of each segment where the model is trained.
- *feats*: A list of names of columns of df corresponding to the feature variables.
- *target*: A name of a column of df corresponding to the dependent variable.
- *window*: The size of the sliding window, as a number of days.
- *step*: The size of the step, as a number of days.

5.4.3.3 Deployment instructions and options

The tool requires the operation of a .csv file. This input provides a (multi)variate pandas dataframe.

In this notebook the user will read a .csv file in order to create a pandas dataframe/timeseries as we can see in the cell [2]. Providing either the relative or the absolute path to the file. Then by setting a list of sorted dates and an integer number of days that the user wishes as in cell [3], the tool will fit a model. After the training the user is going to predict on any possible sliding window of parametrizable size as in cell [4]. Then in cell [5] taking the scores that the user found from the previous step the user Is going to rank and plot the most indicative windows.

6 First cut assessment and implementation progress report

In this section, we provide a first cut, qualitative assessment of how the implemented tools perform in two of the examined use cases. We note that the goal of our initial work in Pattern Extraction has been on developing a core set of utilities and tools to build upon in our next steps, with emphasis on developing meaningful solutions with respect to the provided RES use cases and datasets. Given that, evaluating the efficiency and scalability of the presented methods was out of the scope of the current deliverable. Nevertheless, all the presented utilities and tools are built on top of robust, efficient and easily parallelizable libraries, either via GPU acceleration or via node distribution. The scalability and efficiency of the developed methods will be assessed in upcoming deliverables of Tasks 4.1 and 4.2.

Further, we enumerate the functional specifications of the MORE platform that regard the Pattern Extraction module and report which of them and to which extent are covered in the current, initial version of the methods.

6.1 Qualitative assessment of the pattern extraction tools

6.1.1 Pattern Discovery

In this section, we showcase an execution of our prototype pattern discovery tool. We focus on a real-world use case scenario on a wind turbine dataset that is provided by ENGIE-Laborelec. More specifically, our input consists of a multivariate time series, where the variables correspond to measurements on “active power”, “wind speed”, “wind direction”, etc. ENGIE has also provided some sparse LIDAR measurements made on the yaw of a set of turbines. Based on those measurements we label the different segments of the time series in our experiments.

Our goal is to discover multidimensional patterns in the time series that are indicative of a specific behavior of the time series. More specifically, in the ENGIE use case, we want to discover patterns that are indicative of Yaw Misalignment or Yaw Alignment regions in the time series. To this end, we employ the multidimensional Matrix Profile algorithm [YKK17] to find the top-k multidimensional motifs of the time series of one of the turbines (named BEBEZE01). Then, for each of those motifs we compute their nearest neighbors in the time series utilizing the multidimensional Distance Profile [YKK17]. Afterwards, we assign a cost to each motif depending on the distribution of its neighbors among the various labelled regions i.e., if the majority of the neighbors of a motif lie in a segment that is labelled as “Yaw Misalignment” then it is assigned a low cost and we assume that the pattern is commonly observed in such segments and it is characteristic of the behavior of the turbine when Yaw Misalignment is present.

Finally, to validate the generalization ability of our assumptions, we find the closest matches of a multidimensional pattern we extracted from the initial turbine (BEBEZE01) time series on two other turbines (BEBEZE02 and BEBEZE03) that we have labels available. If the matches of the pattern in the new turbines show a similar distribution, we can say that our initial assumption generalizes reasonably, and a motif (pattern) representing a particular behavior of the time series has been identified for further inspection. A high-level graphical presentation of that method can be seen at Figure 2.

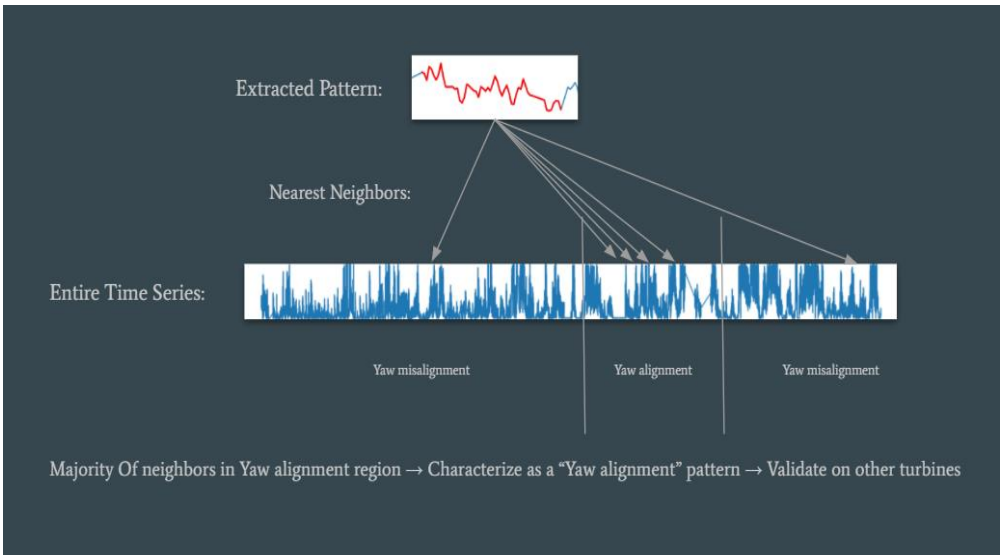

Figure 2: High-level graphical presentation of our method, showing a one-dimensional pattern for simplicity.

In the notebook that is available in our github repository[39], we have performed an experiment following the method that we described above. We first extract patterns from the BEBEZE01 turbine full dataset and then test the validity of our results in the dataset obtained from the BEBEZE02 turbine.

In Figure 3 we can see a table with useful information about the patterns our method has extracted from the BEBEZE01 turbine. At this point, we should remind the reader that a motif is a pair of subsequences of consecutive points of the timeseries that the distance between them is the lowest of any other pair. Additionally, a top k^{th} motif is the pair with the k^{th} lowest distance between them among all the possible pairs. In the table, we can see the column "k" that signifies the ranking of the motif for the respective setting of the experiment i.e., a pattern with k = 4 is the top 4^{th} motif pair for the particular setting. We can also observe the labelled region where the first and the second subsequence of the motif pair are observed. Next, the columns "aligned" and "misaligned" show us the number of the motif instances that fall in each labeled region. Furthermore, we have the "cost" column that characterizes each pattern depending on the distribution of its pair of sub-sequences as well as its neighbors in the labeled regions of the dataset, a lower cost denotes that a pattern is observed mostly in one of the two labelled regions, consequently meaning that it is more likely to be a conserved, meaningful pattern that characterizes the behavior of the respective labelled region. The last columns show some algorithm parameters such as the "exclusion zone" and the "window size", the last column indicates in what kind of region the majority of the motif instances fall into.

Moving on testing the validity of our results, we are going to focus on the pattern with the lowest cost from Figure 3. Initially, we performed a similarity search in order to find the closest matches of each one of the sub-sequences of the motif pair we focused on. Then we investigated and compared the distribution of those matches in BEBEZE02. In (Figure 4) we can get valuable insight on each one of the two sub-sequences extracted from BEBEZE01. More specifically, we can determine the distance between the closest match in BEBEZE02 and the pattern from BEBEZE01 as well as the average of the distances of the found matches in BEBEZE02 and the indices of each one of those matches. We can also examine the distribution of the closest matches and the respective "cost" value in BEBEZE02. In the last column, there is a Boolean value that confirms if the majority of the closest matches of the subsequence is in regions of the same label as the majority of the instances of the motif we examined in the initial time series (BEBEZE01). As seen in the table, both subsequences show a similar

[39] https://github.com/MORE-EU/more-pattern-extraction/blob/main/notebooks/interesting_patterns.ipynb

distribution between “aligned” and “misaligned” regions in BEBEZE02. The subsequence in the second row especially demonstrates a reasonable “cost” that is quite close to the one in BEBEZE01, reinforcing our assumption that an extracted pattern of this kind in one turbine can generalize in others.

	k	motif_subseq_1	motif_subseq_2	aligned	misaligned	cost	window_size	exclusion_zone	radius	motif indices	majority of motif instance occurences
0	4	misaligned	misaligned	11	90	0.665358	1440	0.250	1.5	[5692, 106079]	misaligned
1	5	misaligned	aligned	11	88	0.675373	1440	0.250	1.5	[53792, 274596]	misaligned
2	4	misaligned	aligned	3	22	0.714826	1440	0.500	1.5	[107932, 274225]	misaligned
3	1	misaligned	aligned	29	73	0.763250	1440	0.125	1.5	[138948, 154717]	aligned
4	5	misaligned	aligned	28	71	0.766704	1440	0.125	1.5	[221664, 259094]	aligned
5	8	misaligned	aligned	13	85	0.768342	1440	0.125	1.5	[123808, 239618]	misaligned
6	6	misaligned	misaligned	12	76	0.783479	1440	0.250	1.5	[39021, 182291]	misaligned
7	4	misaligned	misaligned	16	80	0.898542	1440	0.125	1.5	[5134, 121421]	misaligned
8	9	misaligned	misaligned	17	85	0.898542	1440	0.125	1.5	[19405, 87191]	misaligned
9	2	misaligned	misaligned	22	75	0.910552	1440	0.125	1.5	[5324, 140379]	aligned
10	3	misaligned	misaligned	22	77	0.923620	1440	0.125	1.5	[116595, 220186]	aligned
11	7	misaligned	misaligned	18	84	0.932789	1440	0.125	1.5	[178040, 202652]	misaligned
12	3	misaligned	misaligned	15	57	0.964605	1440	0.500	1.5	[5504, 105890]	aligned
13	10	misaligned	aligned	19	83	0.965723	1440	0.125	1.5	[53854, 274657]	misaligned
14	6	misaligned	misaligned	20	82	0.997417	1440	0.125	1.5	[5720, 106112]	misaligned

Figure 3:Table of statistics and information associated with the motifs extracted from BEBEZE01.

Moreover, for demonstrative purposes in Figure 5 we show a visual representation of the multi-dimensional motif pair that we extracted in BEBEZE01 and the closest matches of each subsequence of the pair in BEBEZE02. This way the reader can get a better intuition on what a multi-dimensional pattern looks like and how close the matches obtained from the simillarity search are.

	pattern_idx_in_ts1	motif_pair_dist_in_ts1	Closest neighbor dist in ts2	Average neighbor dist in ts2	NNs in ts2 indices	aligned	misaligned	exclusion_zone	cost	matching_majority
0	5692	35.507103	23.779344	43.687443	[5694, 25817, 50376, 140832, 45972, 20003, 905...	17	83	0.250	0.910338	True
1	106079	35.507103	24.319942	40.281058	[106080, 140757, 172445, 267441, 5729, 90651, ...	14	86	0.250	0.798079	True

Figure 4: Table of statistics and information of a pattern extracted from BEBEZE01 and the closest matches of each of its sub-sequences in BEBEZE02

A user with domain experience in the field of the specific use case can study the statistics in the above tables alongside with a visual representation of the extracted pattern. By utilizing those together with his expertise one can identify potentially useful patterns and proceed to analyse them further. For example, the specific motifs might bring up particular issues in the operation of a turbine, identified via the demonstrated/highlighted behaviour of the time series in the motif areas (areas) for particular variable of the time series. Further, the particular motifs can be utilized in event detection techniques (to be reported in D4.2) in order to nearly real-time identify events on a turbine, such as yaw misalignment, by identifying respective characteristic motifs in an incoming stream of turbine measurements.

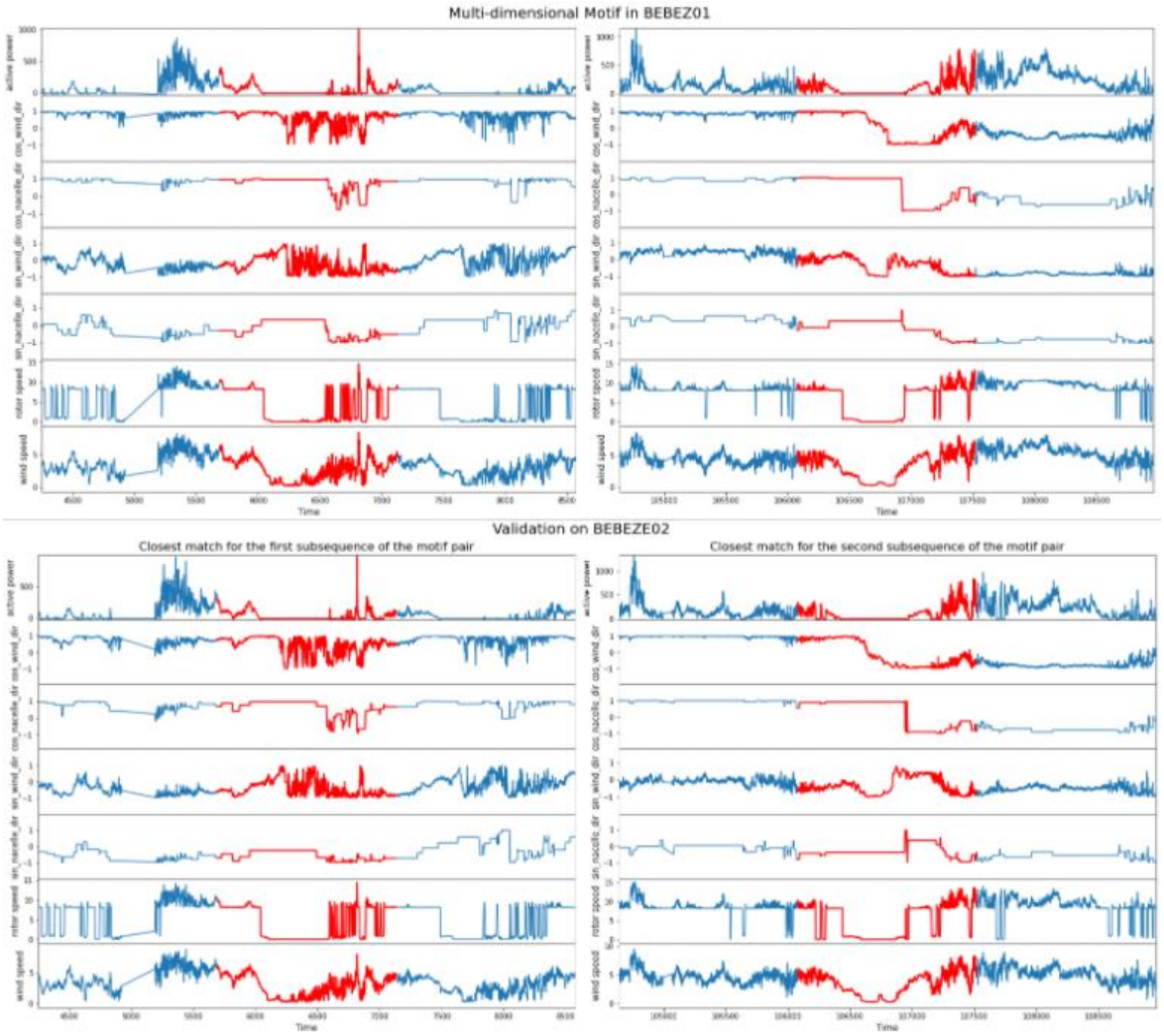

Figure 5: Extracted multidimensional pattern in BEBEZE01 and its closest matches in BEBEZE02.

6.1.2 Time series segmentation

In this section, we showcase an execution of a prototype time series segmentation tool. We focus on a real-world use case scenario on a wind turbine dataset that is provided by ENGIE, described above in Section 6.1.1.

Our goal is to discover points in the time series that are indicative of a change in the behavior of the time series. More specifically, in the particular use case, we want to discover segments that are indicative of Yaw Misalignment or Yaw Alignment regions in the time series. For that purpose, we employ the multidimensional Matrix Profile algorithm [YKK17]. To identify them, we exploit the semantic segmentation algorithm of the Matrix Profile suite, which segments a time series so that the number of common motifs between different segments is minimized. Then, for those changepoints we compute their distance with the ground truth changepoints (labels) in the time series utilizing the Dynamic Time Warping (DTW) algorithm. This distance serves as a measure of cost or penalty for our model, providing an estimate of how far it is from making predictions as expected. We use this distance to optimize our model over a selected set of configurations.

We apply the first step of the tool, i.e. the training step, on the first available turbine, named BEBEZE01. There, we assess several configurations of the MP segmentation algorithm, w.r.t. the

produced DTW distance between the actual changepoints provided as labels and the predicted changepoints by the algorithm. In particular, the DTW distance and then we scale it between the measurements. In return, for each configuration, we receive the dates which segment our timeseries. Our model considers the best segment configuration the one that minimizes the Normalized Distance.

	L	Changepoints indexes	Changepoints Dates	Normalized Distance
0	18000	[1235560, 1145559, 1039863, 911129, 821128]	[2019-03-08, 2019-02-04, 2018-12-30, 2018-11-15, 2018-10-15]	0.166482
1	19000	[1172161, 1039863, 911129, 726766, 622161]	[2019-02-14, 2018-12-30, 2018-11-15, 2018-09-12, 2018-08-07]	0.051764
2	20000	[1172161, 1039863, 911129, 726766, 622161]	[2019-02-14, 2018-12-30, 2018-11-15, 2018-09-12, 2018-08-07]	0.051764
3	21000	[1172161, 1039863, 911129, 726766, 619033]	[2019-02-14, 2018-12-30, 2018-11-15, 2018-09-12, 2018-08-05]	0.060048
4	22000	[1172161, 1039863, 911129, 726766, 616739]	[2019-02-14, 2018-12-30, 2018-11-15, 2018-09-12, 2018-08-05]	0.066124
5	23000	[1172161, 1039863, 911129, 726766, 611765]	[2019-02-14, 2018-12-30, 2018-11-15, 2018-09-12, 2018-08-03]	0.079296
6	24000	[1172161, 1039863, 911129, 726766, 599138]	[2019-02-14, 2018-12-30, 2018-11-15, 2018-09-12, 2018-07-30]	0.112737

Figure 6: Normalized distances between actual and predicted changepoints for several configurations of the MP semantic segmentation algorithm, on the "training" dataset of turbine BEBEZ01

This first step provides as an optimal configuration, which provides the segmentation results depicted in Figure 7. We can see that the predicted changepoints fall quite close to the actual ones.

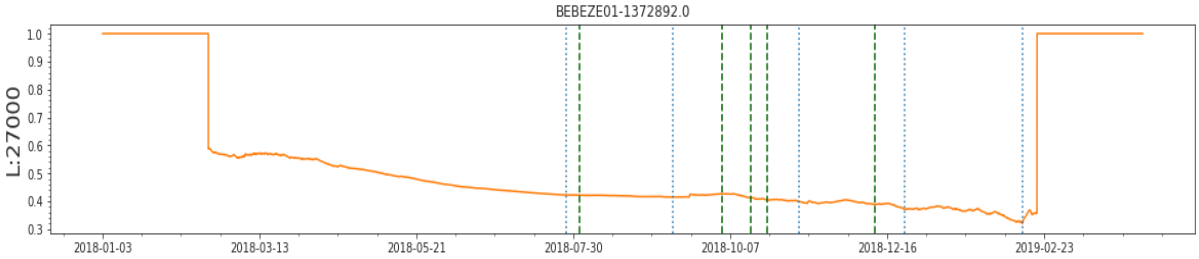

Figure 7: Segmentation model trained on turbine BEBEZE01. Green lines comprise the labelled data, functioning as actual segmentation points (or changepoints), while the blue lines comprise the model's segmentation predictions

As a next step, we assess the above selected model on the other two turbines plot those results (Figure 8 and Figure 9 respectively). We observe that the results share a similar behavior as we can see from the graphs below. Namely, the configuration learnt in the first turbine generalizes well to the other two turbines, predicting (some) changepoints that fall very close to the actual ones.

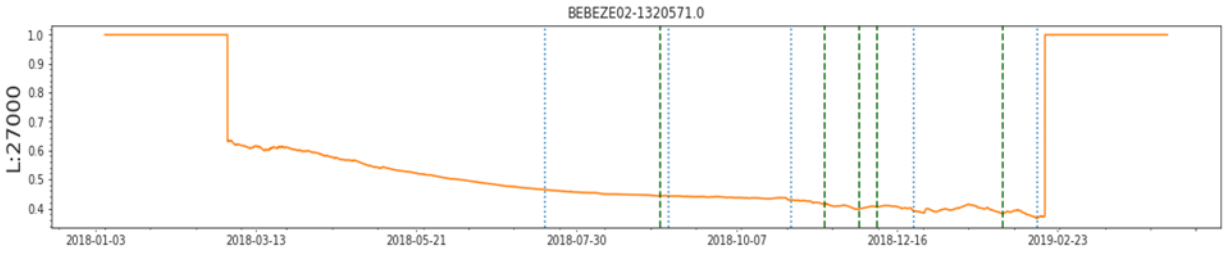

Figure 8: The best segmentation model (selected on BEBEZE01), evaluated on turbine BEBEZE02. We observe that several actual changepoints are approximated by the model's predictions also in BEBEZE02.

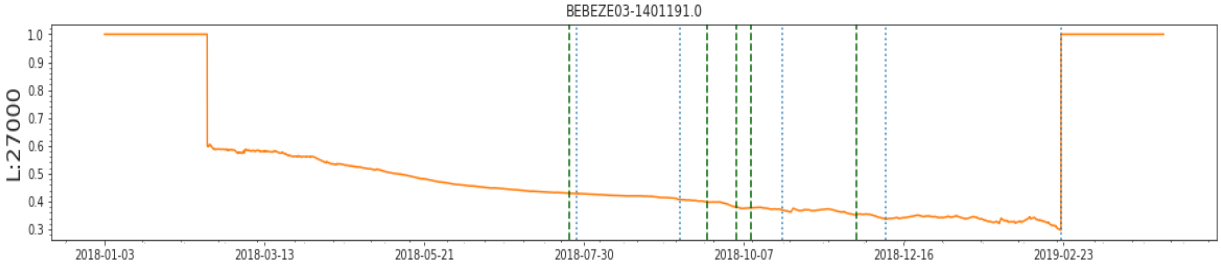

Figure 9: The best segmentation model (selected on BEBEZE01), evaluated on turbine BEBEZE03. We observe that several actual changepoints are approximated by the model's predictions also in BEBEZE03.

6.1.3 Changepoint detection

In order to present a complete execution of our changepoint detection tool, we focus on applying it to the special use case of detecting changepoints in the power output of solar panel arrays. The dataset is provided to us by INACCESS. Our input consists of a multivariate time series, where the interesting variables correspond to measurements on "power output", "module temperature" and "irradiance".

Power output is the dependent variable, i.e. the variable whose behavioral change is being detected, and module temperature and irradiance are the two feature variables used in regression. Moreover, we are given values of precipitation as input, and we use them to detect maximal periods of positive precipitation, i.e. rains. We use our tool to detect changepoints among all rains, and in particular we are interested in changepoints that positively affect power output, i.e. rains which have potentially cleaned the solar panels and this has led to an increase in power output.

	Score (Percentage difference of ME)	Starting date	Ending date	Max precipitation	Mean precipitation	Validation ME (true-pred) error	Test ME (true-pred) error
0	29.067368	2015-09-25 22:00:00	2015-09-26 09:15:00	0.86	0.47	0.000031	0.000930
1	6.958397	2014-11-07 12:15:00	2014-11-08 04:45:00	1.04	0.47	-0.000906	0.005397
2	6.535540	2020-10-01 21:45:00	2020-10-02 07:15:00	2.03	0.19	-0.000513	0.002842
3	3.952931	2014-10-07 09:00:00	2014-10-07 18:30:00	0.97	0.68	-0.003928	0.011600
4	2.784995	2016-10-27 12:30:00	2016-10-28 11:45:00	1.54	0.53	0.001187	0.004494
5	2.688800	2016-06-25 12:30:00	2016-06-26 02:30:00	0.89	0.40	-0.005955	0.010058
6	2.552649	2021-03-01 01:45:00	2021-03-02 00:00:00	0.90	0.17	-0.001694	0.002631
7	2.391768	2020-11-29 12:30:00	2020-11-29 20:30:00	1.85	0.51	-0.001442	0.002007
8	2.369552	2014-10-05 10:30:00	2014-10-06 04:30:00	0.94	0.39	-0.005513	0.007551
9	1.961907	2020-10-12 22:00:00	2020-10-13 10:00:00	4.29	0.40	-0.000895	0.000860

Figure 10: Output example. We use the mean validation error and the mean test error to rank rains with respect to their possibility of containing a changepoint.

Figure 10 shows a sorted list of rains. Highly ranked rains are the ones that we consider more probable of containing a changepoint. We rank rains by computing the percentage difference between the validation mean error and the test mean error. Recall that we validate our regression model in a time period before the assumed changepoint and we test it in a time period after the assumed changepoint. Therefore, a changepoint, which positively affects the power output, has a positive effect in the percentage difference between the two errors. For a complete discussion on the errors computed during validation and testing, which can be used in developing other ranking methods, e.g. in scenarios different from the one studied here, the reader is diverted to Section 5. Next, we focus on one of the

highly ranked rains, and we analyze our result with respect to that rain. We choose the rain with index 3 as this is an example with moderately high score value that is most likely not an outlier. Figure 11 illustrates the predicted and the true values of power output, for a time period containing the training set, the validation set and the test set for the rain with index 3 (as in the table of Figure 10. The training set contains all those points in the time period that starts 25 days before the beginning of the given rain and ends 10 days before the rain. This is followed by the time period defining the validation set. The test set then contains points in a time period of 10 days immediately after the end of the rain.

In the last two columns of Figure 10, we can see that during the validation period, the mean error is –0.003928 and during the test period the mean error is 0.011600. This means that our regression model successfully captures the behavior before the rain, and in particular it overestimates (on average) the power output during that period, but it significantly underestimates the corresponding values after the rain. We conclude that this rain has a positive effect on power output and it should be considered a changepoint.

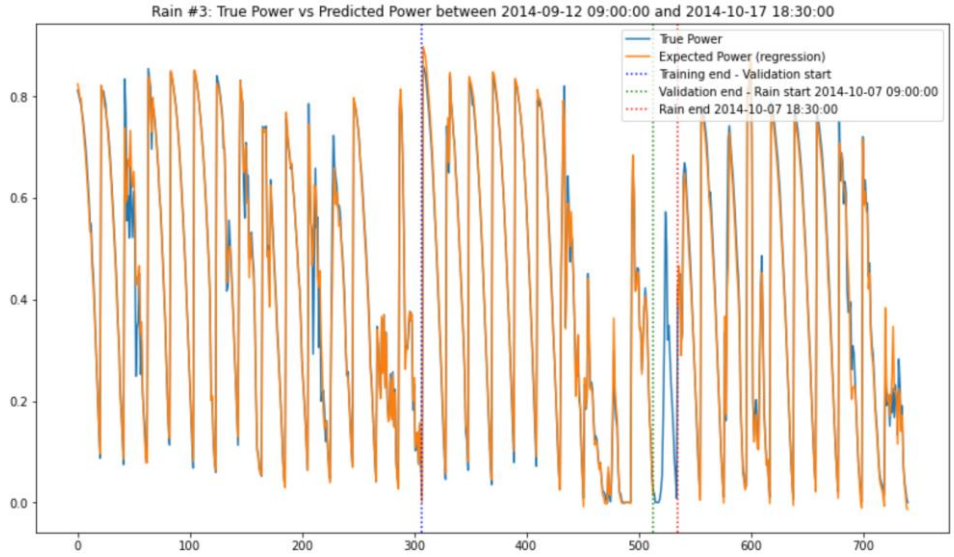

Figure 11: Regression on power output: real vs predicted values.

6.1.4 Deviation detection

Here, we present an execution of our prototype deviation detection tool. We focus on a real-world use case scenario on a solar panel dataset that is provided to us by INACCESS. More specifically, our input consists of a multivariate time series, where the variables correspond to measurements on “power output”, “module temperature” and “irradiance” among others.

Our aim is to learn the behavior that the solar panels exhibit during periods that they are clean with the utilization of a regression model and then detect windows of possible soiling by detecting deviation from that learned model. The target variable of the regression model is the "power output" and the variables used as features are "irradiance" and "module temperature". After training the model on the clean regions, we use it to predict how the panels should behave under specific environmental conditions ("irradiance", "module temperature") if they were clean.

We perform the previous process on windows spanning the entirety of the time series and calculate various metrics for each window such as the Mean Percentage Error, the Mean Absolute Percentage Error, the R-squared among others. We post-process the metrics, rank the windows and pick the ones

where we observe the actual power being significantly lower than the expected power (i.e. the power that we would expect if the panels were clean). A window that exhibits such a behavior is shown in Figure 12.

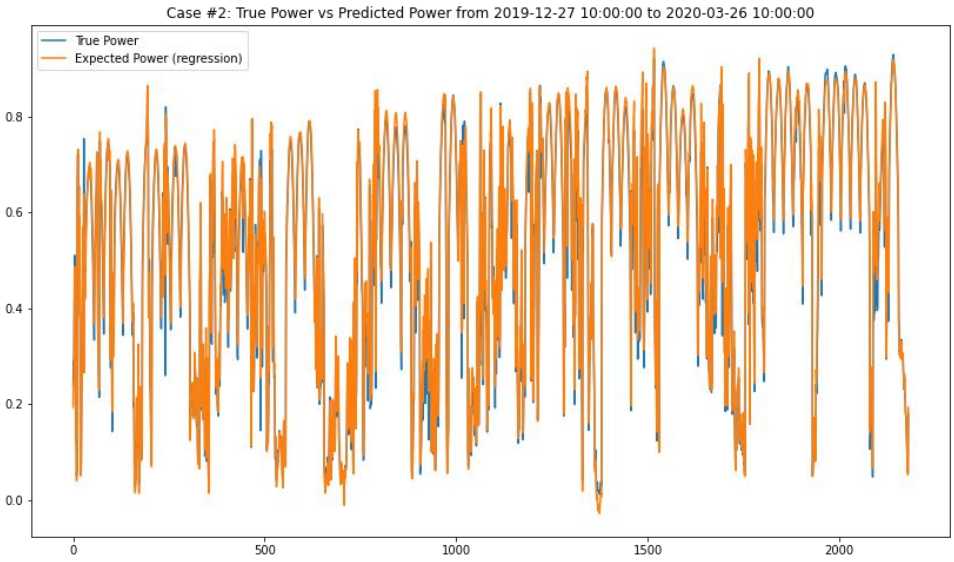

Figure 12: Plot of a window where the True Power output of the panels is on average lower than the Expected Power (learned on clean panels).

Since in this case we are trying to detect deviations that happen due to soiling that is a gradual process, we also consider the trend (Figure 13). To compute the trend, we use a derived time series that is the actual power minus the output of our model, smoothened by a rolling mean. On that time series we fit a line using the Ordinary Least Squares method and we assume that this line and consequently its slope approximates the trend. If we measure a small negative trend, it might be indicative of the gradual loss of power that happens due to soiling.

In Figure 14 we can see a table containing some results from our experiments. More specifically, in the columns “Window Starting Date”, “Window Ending Date” the user can see the beginning and the end of each extracted window. In the column “MPE we can see the Mean Percentage Error metric that we calculate in the window, that is the mean percentage of difference of the predicted value from the true value. The advantage of MPE is that is informative of whether the change occurs in the negative or in the positive direction. Next, in the column “Slope” one can observe the slope of the line we fit on the derived time series that represents the differences that denote the actual power minus the output of our model. Finally, in the first column we can see the score that we use to rank each window. We combine the values of MPE and slope with a product so that we can rank higher, windows, that show a negative trend and where the actual values are generally lower than the values of our model. To achieve this, we use the product $MPE \cdot \min(0,\ slope)$, this way we eliminate the positive slopes since we want to identify a negative trend in this use case and, by using that score, we guarantee that the windows assigned the highest values are the windows that exhibit the most negative slope and MPE. Users experienced in the particular domain of the problem can extract even more information i.e., they can tell if the values of the MPE and slope observed are expected in the detected windows and investigate further.

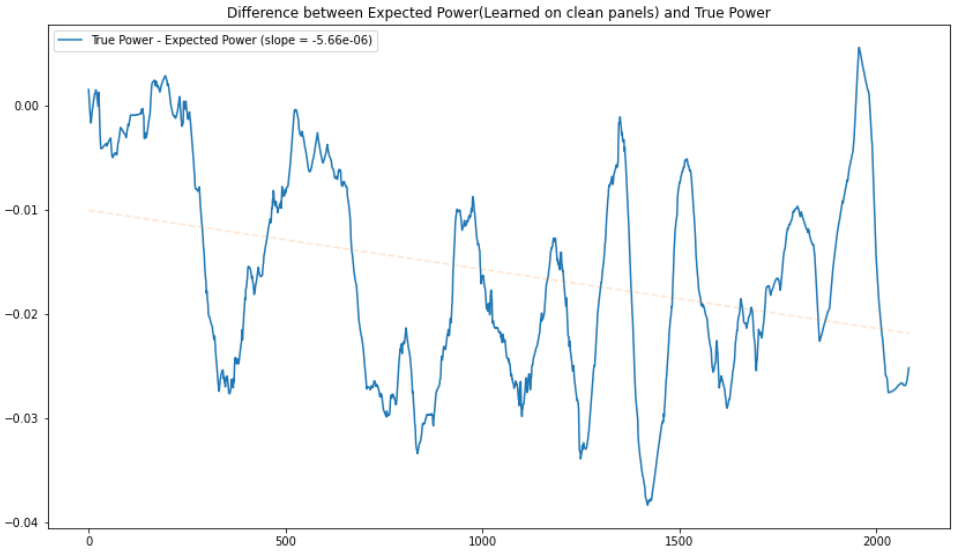

Figure 13: Plot of the derived time series True Power - Expected Power (Our model output) and the fitted line representing the trend.

	Score "(mpe * min(0, slope))"	Window Starting Date	Window Ending Date	MPE	Slope
0	3.865026e-07	2019-11-27	2020-02-25	-0.025145	-0.000015
1	1.641525e-07	2019-12-27	2020-03-26	-0.028984	-0.000006
2	1.459724e-07	2021-05-20	2021-08-18	-0.016539	-0.000009
3	6.990360e-08	2019-10-28	2020-01-26	-0.010406	-0.000007
4	6.854006e-08	2021-04-20	2021-07-19	-0.008370	-0.000008

Figure 14: Table with information associated with the top 5 windows found in this experiment.

6.2 Functional specifications implementation progress

In this section, we enumerate the functional specifications regarding the Pattern Extraction module (PE) of the more architecture that were defined in D5.2, which formally summarize the core functionality expected from the developed methods. Further, we briefly describe their implementation status in the current, initial version. For each functional specification, its implementation status is provided, considering three stages:

- **Implemented** denotes a fully functional implementation of the respective specification, with or without potential improvements to be planned for next releases.
- **Partially Implemented** denotes a partial implementation of the specification, meaning either that (i) a significant part of the specification is developed, however, the implementation is not yet functional or (ii) a significant subset of the related functionalities to the specification is implemented and fully functional, however, more individual functionalities still need to be implemented for the specification to be considered fully covered.
- **Pending** denotes that the particular specification is planned to be implemented in an upcoming release.

Functional specification	Implementation status – Initial version	Comments
FS-PE-01 Sample a time series in various time scales	Implemented (see subsection 4.2)	-
FS-PE-02 Preprocess/filter a time series	Implemented (see subsection 4.2)	Fully functional implementation – Potential extensions in the final version
FS-PE-03 Annotate a time series by using a custom user defined annotation function.	Pending	-
FS-PE-04 Annotate a time series by using a set of pre-implemented annotation functions for generic use on RES time series.	Partially Implemented (see subsection 4.5)	Currently partially supported by preprocess/filter functionalities
FS-PE-05 Automatically identify optimal Annotation Vector parameters	Pending	-
FS-PE-06 Normalize a time series	Implemented (see subsection 4.2)	-
FS-PE-07 Approximate a time series	Implemented (see subsection 4.7)	Fully functional implementation – Potential additions of approximation functions in the final version
FS-PE-08 Extract statistics on a time series	Implemented (see subsection 4.3)	Fully functional implementation – Potential additions of approximation functions in the final version
FS-PE-09 Compare two univariate time series	Implemented (see subsections 4.3, 4.6)	Fully functional implementation – Potential additions of comparison functions in the final version
FS-PE-10 Compare two multivariate time series	Implemented (see subsection 4.6)	Fully functional implementation – Potential additions of comparison functions in the final version
FS-PE-11 Identify the k-Nearest Neighbors of a univariate time series with respect to a standardized evaluation measure	Implemented (see subsection 4.6)	Mainly adaptation of the MP library to our setting
FS-PE-12	Implemented (see subsection 4.6)	Extensions of the MP library to our setting

Identify the k-Nearest Neighbors of a multivariate time series with respect to a standardized evaluation measure		
FS-PE-13 Index a univariate/multivariate time series via Matrix Profile	Implemented (see subsection 4.5)	Mainly adaptation of the MP library to our setting
FS-PE-14 Extract interesting patterns (motifs) from a univariate/multivariate time series	Implemented (see subsection 4.5)	Fully functional implementation – Potential extensions/variations in the final version
FS-PE-15 Extract interesting patterns (motifs) from a summarized univariate time series	Pending	-
FS-PE-16 Identify anomalies (discords) in a univariate/multivariate time series	Partially Implemented (see subsection 5.3)	Partially implemented by the changepoint detection/verification tool and the respective utilities. Additional utilities and tools to be implemented in the final version
FS-PE-17 Identify anomalies (discords) in a summarized univariate time series	Pending	-
FS-PE-18 Identify changepoints in a univariate/multivariate time series	Implemented (see subsection 5.3)	Fully functional implementation – Potential extensions in the final version
FS-PE-19 Identify trends in a univariate/multivariate time series	Partially Implemented (see subsection 5.4)	Partially supported in the deviation detections tool and the respective utilities. Additional utilities and tools to be implemented in the final version
FS-PE-20 Decompose a time series into Seasonal, Trend and Remainder components.	Pending	-
FS-PE-21 Segment a univariate/multivariate time series into regions of different behaviour.	Implemented (see subsection 5.2)	Fully functional implementation – Potential extensions in the final version

Table 2: Functional specifications and implementation status for the PE module

Shortly, out of 21 prescribed functional specifications, 13 are fully implemented and 3 are partially implemented, with only 5 being pending. Of course, several of the already implemented specifications will be constantly revisited as Task 4.1 proceeds, updating, improving and extending the currently implemented functionality.

7 Conclusion

Deliverable 4.1 presents the initial version of the pattern extraction methods of the MORE platform. These methods are a set of reusable utilities, as well as four initial tools in the form of notebooks. These initial methods cover a large part of the functional specifications for the Pattern Extraction module, defined in D5.2. Further, they are inline with the prescribed use cases of D5.1, demonstrating encouraging effectiveness results in handling two of the use cases.

The presented methods comprise the basis of work in Tasks 4.1 and 4.2. Our ongoing work includes the adaptation of several of the above functionality to be directly applicable on summarized time series, according to the ModelarDB models, as well as the extension and enhancement of existing utilities. Further, we aim to extend the existing set of tools, in order to cover all the prescribed use cases as well as variations of them. Finally, we are working on integrating the existing tools with the visualization facilities developed in Task 4.3, in order to facilitate their configuration and usage by RES stakeholders.

8 References

[GDY+17] S. Gharghabi, Y. Ding, C. M. Yeh, K. Kamgar, L. Ulanova and E. Keogh, "Matrix Profile VIII: Domain Agnostic Online Semantic Segmentation at Superhuman Performance Levels," 2017 IEEE International Conference on Data Mining (ICDM), 2017, pp. 117-126, doi: 10.1109/ICDM.2017.21.

[YKK17] Chin-Chia Michael Yeh, Nickolas Kavantzas, Eamonn J. Keogh:Matrix Profile VI: Meaningful Multidimensional Motif Discovery. ICDM 2017: 565-574