D4.3 Complex event detection module - Initial Version

Lead Partner:	ATHENA
Version:	Final
Dissemination Level:	PU
Work Package:	WP4
Due date:	31/03/2022
Submission Date:	31/03/2022

Abstract:

Deliverable 4.3 presents the initial version of the complex event detection methods, for handling the identification of complex patterns and events within RES time series. Using as starting point two diverse use cases (yaw misalignment detection and soiling detection), we have developed supervised and unsupervised methods for the identification of complex events to cover a wide range of event detection settings in RES time series, considering the type of the event (changepoint, time series segments) and the task (e.g. detect turbine yaw/pitch misalignment; detect significant PV panel cleaning events; detect soiling periods; detect the beginning of a potential soiling event). We demonstrate the prediction accuracy of the developed methods in an extended set of experiments. Further, we describe our ongoing work in exploiting motif discovery schemes for pattern detection in the examined RES use cases. Finally, we present preliminary implementations of our solutions on a scalable and parallelizable basis, demonstrating their efficiency on real-time event detection, in a large number of different input data streams.

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

Document Revision History

Date	Version	Author/Editor/Contributor	Summary of main changes / Status
25/02/2022	0.1.	Giorgos Giannopoulos, Ioannis Psarros	ToC
04/03/2022	0.2	Yannis Psarros, Alexandros Kalimeris, Panos Gidarakos	Contributions on method descriptions
11/03/2022	0.3	Yannis Psarros, Alexandros Kalimeris, Panos Gidarakos	Contributions on method descriptions and experimental evaluation
14/03/2022	0.4	Giorgos Giannopoulos, Manolis Terrovitis	Contributions on system design and KPIs
21/03/2022	0.5	Nguyen Thi Thao Ho	Contributions on MP and summarized data
23/03/2022	0.6	Yannis Psarros, Alexandros Kalimeris, Panos Gidarakos, Dimitirs Tsitsigkos, Giorgos Giannopoulos, Manolis Terrovitis	Version ready for internal review
29/03/2022	0.7	Seshu Tirupathi, Christos Tsiakaliaris	Internally reviewed version
31/03/2022	0.8	Yannis Psarros, Alexandros Kalimeris, Panos Gidarakos, Giorgos Giannopoulos, Nguyen Thi Thao Ho	Final version prepared
31/03/2022	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 4.2 develops complex event detection methods, comprising a twofold purpose: (a) provide methods that can identify a variety of events on RES time series and (b) render these methods applicable in a (nearly) real-time, distributed setting, where prediction/detection needs to be deployed continuously in (sliding windows of) parallelly incoming time series streams.

Deliverable 4.3 implements the initial version of the complex event detection methods, providing the tools to RES stakeholder to accurately and scalably handle two representative and significant use cases (yaw misalignment and soiling detection). Further, the implemented methods comprise the basis for providing a more generalizable toolkit of algorithms for pattern extraction and complex event detection. More specifically, this deliverable presents three sets of methods, as outlined next:

- **Changepoint detection/deviation detection**. This group of methods comprise an approach that jointly handles the tasks of changepoint detection, pattern extraction and behavior (event) detection on the power variable of a RES time series. The proposed approach is realized in the context of soiling detection, which includes individual sub-tasks that need to be handled: identifying changepoints that signify an effective washing event on the panels; identifying time series segments where a soiling event has occurred (offline-batch mode); detecting the beginning of a potential soiling event (online, real time mode). Our approach implements variations of different-purpose regression models for power approximation that are learnt on different areas of the time series. We note that our approach, although training regression models on predicting the expected power output, is considered unsupervised, since it does not require labels on soiling events as input. For the purposes of D4.3, training and assessment of our methods is performed in the soiling use case dataset provided by InAccess, as well as on an external, benchmark dataset provided by the National Renewable Energy Laboratory of USA (NREL)[1].
- **Pattern/behavior detection**. The aim of this group of methods is to be able to identify, either in an offline-batch setting, or in an online, real-time deployment, particular behaviors of a multivariate RES time series with respect to one of its variables. The proposed approach is realized in the context of yaw angle misalignment detection thus the behavior is defined through the different values of the yaw angle variable. Our approaches exploit regression algorithms that model different expected behaviors of the time series by learning on a (labelled) training set and predict the most probable behavior by aggregating approximations of the different models on newly incoming (test) data. For the purposes of D4.3, training and assessment of our methods is performed in the yaw misalignment (YM) dataset provided by ENGIE. Nevertheless, they are applicable, with proper configuration, tuning and adaptations, in other use cases settings (e.g. pitch misalignment detection, small gain detection, error detection in the wind parks setting), which comprise part of our future work in Tasks 4.1 and 4.2.
- **Motif-based pattern extraction and detection**. This group of methods continues part of our work documented in D4.1 “Pattern extraction methods - Initial Version”, by examining how the Matrix Profile suite of algorithms can be adapted or extended in order to specialize for handling the use cases of the project. In particular, in the current document we briefly report on: (a) our ongoing work on adapting the Matrix Profile computation process to be applicable directly on the summarized (modelled) time series representation of ModelarDB; (b) our initial

[1] https://datahub.duramat.org/dataset/data-for-validating-models-for-pv-module-performance

work on exploring Annotation Vector spaces, with respect to the time series values they utilize, that are suitable for focusing the motif extraction process specifically towards soiling events.

We note that, the work of Task 4.2 during the first period of the project emphasizes on effectiveness of the implemented methods. This consists in exploring and analyzing the RES use cases and the respective real-world datasets, obtaining insights and domain knowledge and utilizing it to implement methods that accurately solve the pattern extraction/event detection problems of the use cases. Having achieved this goal to a sufficient extent, our next steps will emphasize more on efficiency and scalability of the methods. Nevertheless, as demonstrated in Sections 4.4.3 and 5.4.4, there already exists the provision for scaling the implemented methods in millions of time series streams.

The evaluation of the two first groups of implemented methods (deviation detection and behavior detection) is performed in terms of (a) effectiveness, i.e. how accurately they detect the events considered in the two utilized use cases and (b) efficiency and scalability, i.e. how fast they can perform predictions in an online setting, where predictions need to be performed on continuously incoming windows of time series measurements, in parallel, from thousands of modules (either turbines or solar panels).

D4.3 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 “Use Cases and Requirements”). Section 3 presents a high-level architecture of the Complex Event Detection (CED) module of the MORE platform (see D5.2 for details). Section 4 describes the set of methods handling the **changepoint detection/deviation detection** tasks, while Section 5 describes the respective methods for the **pattern/behavior detection** tasks. Section 6 summarizes our progress on motif-based pattern extraction and detection, while Section 7 reports the current progress towards covering the respective functional requirements of the CED module (D5.2), as well as selected KPIs from the description of work. Section 8 concludes the deliverable.

2. Requirements of the initial version

In this section, we briefly discuss the requirements that guided the development of the initial methods for complex event detection. 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 complex event detection. 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 4.2 aims at four directions:

1. Identify frequently cooccurring types of patterns in time windows through a time series.
2. Develop both supervised and unsupervised methods for the identification of complex events, so as to cover a wide range of event detection settings in RES time series.
3. Extend the Matrix Profile (MP) suite of algorithms in order to adapt them in the RES setting, considering several directions: extension of the MPdist time series distance function so that the identification of motifs on new time series becomes more effective; exploit and adapt Annotation Vectors to utilize (MP) on supervised event detection settings on weakly annotated data; exploit identified motifs as training features in supervised settings for event detection.
4. Deploy sliding window approaches for event detection bases either on motifs, or on ML/DL approaches, that will consider elementary patterns within their processing.

The initial work of the task has focused mainly on points 1., 2. and 4., while contributions towards 3. have also been performed. In particular, considering point 1., and given the fact that this point has been initially handled via motif discovery in our previous work (see D4.1 “Pattern extraction methods - Initial Version”, Section 5.1), in the current deliverable we focused on the complementary aspect of regression-based patterns. In particular, patterns are now represented via regression models that aim to accurately approximate the power output of a module (also the angle of misalignment in the wind turbine setting specifically). By aggregating and counting the occurrence of individual pattern-representing models, we are able to detect an event/particular behavior on new incoming time series data in the turbine setting (see Section 0 for details).

Considering point 2., we have developed supervised, semi-supervised and unsupervised methods for event detection. Supervised methods have been deployed in the yaw angle misalignment use case, where the misalignment angles can be utilized as labels for training a model. Semi-supervised and unsupervised methods have been developed for the soiling use case, where soiling comprises a much more complex phenomenon, mostly empirically defined/identified, thus not adequate labeled data exist and semi-supervised or unsupervised methods need to be applied.

Considering point 4., all the developed methods support their deployment in sliding windows of incoming time series, for handling (nearly) real-time settings, where streams from many modules (turbines or panels) are being fed in parallel into the system.

Considering point 3., we have performed some preliminary work on adapting annotation vectors in order to identify complex events (considering several variable of the multidimensional time series) in the soiling use case, with rather encouraging results, as shown in Section 4.4. Further, a preliminary work on implementing a multidimensional distance function for matching motifs with time series segments has been documented in D4.1 (see D4.1, Section 5.1.2). This will comprise our basis for further extending the MPdist function in our next steps.

2.2. Link to the use cases

Following the same rationale with D4.1, our emphasis during the first period of the project regarding the development of complex event detection methods was on implementing methods for meaningfully and accurately solving real world RES problems, as prescribed by the use cases (D5.1). Thus, the most representative use cases from the two RES settings (yaw misalignment for wind parks, soiling of panels for solar parks) comprised our basis for method development and assessment. Next, we provide a brief description of each use case and explain how they drove the development of the initial pattern extraction methods. Note that more detailed descriptions can be found in D5.1 and D4.1.

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.

Our first approach handling the problem as a pattern extraction task was reported in D4.1 (Sections 5.1 and 5.2). In the current deliverable, we change the perspective, handling the problem as an event detection task, introducing the requirement to be able to deploy the developed methods in a real-time setting. Namely, our methods need to effectively and efficiently/scalably predict the yaw angle in constantly incoming windows of multidimensional time series, produced by a large number of different turbines in parallel.

In our setting, the multidimensional time series consists of a set of variables-measurements on the turbine, including wind speed, rotor speed, wind and nacelle direction and produced power. The desired event-behavior we aim to detect is the (absolute value of the) static angle between nacelle and wind direction, termed yaw misalignment angle, as described above. Given that, we consider two different approaches for modeling and solving the task.

- Directly use the yaw misalignment angle as a dependent variable and train regression models that exploit the remaining time series variables as independent variables to predict the former. The regression models are either (a) to be learned on a historical part of a turbine's time series that has available labelled data (trustworthy Lidar measurements on the angle) and be deployed (tested) on the newly incoming data of the same turbine or (b) to be learned on a turbine's time series and deployed on different turbines. This comprises the most intuitively straightforward approach, allowing to experiment with less variants, as described in detail in Section 5.
- Indirectly model the yaw misalignment angle by training regression models for approximating the power output (dependent variable) and assigning them to different angles. The prediction on newly incoming data is then performed by aggregation of the assigned angles of individual regression models that better approximate the power output on a time window of the new time series. This comprises a more elaborate modelling approach, that allows us to experiment with more variants, e.g., considering aggregation schemes.

We need to emphasize that the yaw misalignment use case comprises the motivation for designing and implementing the aforementioned approaches; other use cases in the wind turbine setting can

be handled with particular adaptations, reconfigurations or extensions. For example, the pitch misalignment use case (D5.1, Section 4.2) aims at detecting when the angle of a turbine's blade is misaligned and highly resembles the current setting. In this case, several straightforward adaptations need to be performed regarding the considered independent variables, the smoothing mechanisms and the model tuning process, so as to effectively adapt the current method. Considering the third wind turbine-related case, the small gain (D5.1, Section 4.3), the goal is to identify differences in the power output-gain, before and after a correction on the turbine. Since the implemented methods aim at learning-modelling specific behaviors, they can be adapted in proper comparison schemes that will compare behaviors before and after a correction and decide on the significance of the changed behavior with respect to the expected gain. Our ongoing work includes the aforementioned extensions and adaptations so as to be able to handle all wind turbine use cases under a unified framework.

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.

Soiling is on its own an inherently complex phenomenon. It consists in a gradual degradation of power output that, at some turn point, becomes rather costly for the park operator. Due to its gradual and, usually, subtle behavior it becomes rather challenging to detect, especially taking into account the normally expected variability of the solar panel's power output due to weather conditions and/or momentary outlier behaviors. On top of this, the complexity of the phenomenon further lies on the various patterns/events related to it. In particular, precipitation (rain) events may affect in a different way a soiling event, depending on their severity: isolated, low precipitation events might add to the phenomenon and more severe precipitation events will probably terminate a soiling event. To this end, we can break down the soiling phenomenon in three pattern/event detection tasks that need to be handled:

1. Changepoint detection, involving the identification/verification of whether 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. The specific task was initially studied in our previous work in D4.1 (Section 5.3), however, in our current work we performed several extensions, which are described in detail in Section 4.
2. Deviation detection, focusing on the identification of areas of the time series where a potential soiling event has taken place. Similarly, we significantly extended our previous work (D4.1 Section 5.4), providing a series of variants to handle the problem.
3. Real-time deviation detection, which focuses on the real time exploitation of the aforementioned methods on newly incoming streams from solar panels. Our goal there is to be able to constantly provide an estimate of the soiling loss (termed soiling derate), so as a park operator is able to take decisions regarding the need to manually wash a solar panel park. This final component of the proposed approach exploits models learned from the previous steps, in order to effectively provide this estimate.

The aforementioned components comprise a unified framework for pattern extraction and event detection, trained and assessed on the soiling use case. Methods provided by this framework are

either unsupervised or semi-supervised. Unsupervised methods train regression models on the power output of a panel, and they do not require labels on soiling events to be trained on, while semi-supervised methods exploit a minimal set of labelled data, namely the dates on which the solar panels were manually washed. Further, the rationale of training regression models for identifying changepoints and patterns can be adapted to other use cases, including the small gain detection in wind turbines and the PV tracker optimization in solar parks. These also comprise part of our future work on Tasks 4.1 and 4.2.

In parallel, both use cases motivate us to continue exploring the potential of motif discovery algorithms in the setting.

3. System design and components

In this section, we briefly present a high-level architecture of the Complex Event Detection module. For completeness, the upper part of Figure 1 presents a high-level description of the interconnections between the Pattern Extraction, Complex Event Detection and Visual Analytics modules, as introduced in D4.1. The lower part of the figure presents the currently implemented components of the framework, corresponding to behavior detection and deviation detection. The behavior detection component comprises an offline training phase and an online (real-time) deployment phase. The deviation detection component comprises an offline training phase and both offline and online deployment phases. Each component implements a series of method variants as enumerated below:

Deviation detection:

- Self-supervised changepoint detection. This refers to our method for detecting changepoints in one variable's behavior as follows: given a set of segments of the input time series, our method decides for each one of those segments whether it contains one changepoint. Without exploiting any information regarding parts of the time series where the variable in question behaves as expected, we provide a solution which essentially translates the original problem to a supervised one, based on the following observation: if there is a changepoint in a sufficiently short segment, then it should be that the target variable deviates from its expected behavior in the preceding (resp. succeeding) period of time.
- Semi-supervised changepoint detection. This also refers to a method for detecting changepoints in one variable's behavior, similar to the above. The main difference is that we assume minimal knowledge of the following type: we have a small number of points, not sufficiently large to define a proper training set, where the target variable is assumed to behave as expected. Our approach here is to define periods of time, parameterized by a user-defined length, which start from the given points and are sufficiently short to maintain no (or not many) changepoints. The resulting union of periods defines a training set for a model that aims to capture the expected behavior of our target variable. The resulting model is expected to accurately predict a time period following (resp. preceding) a changepoint, while it should poorly predict a time period preceding (resp. following) a changepoint.
- Self-supervised modelling. This component refers to regression models that capture the target variable's expected behavior. While no labelled points, i.e., points which are known to represent the expected behavior, are explicitly given, our approach uses the changepoints computed by the aforementioned components, to define training sets; a training set is a union of time periods starting from the detected changepoints.
- Semi-supervised modelling. This also includes regression models for the target variable's expected behavior. In comparison to the above, here, we additionally exploit a minimal set of representative points, which is not sufficiently large to define a training set.
- Deviation detection. Our models for the expected behavior aim to provide a tool for detecting periods where the target variable has a deviating behavior. This deviating behavior is typically slowly progressing, so it cannot be detected as a changepoint, since changepoints refer to rapid changes. This component provides a toolkit for analysing historical data and detecting deviating periods in a time series which is completely known when our models are deployed.
- Real-time deviation detection. Using our models for the expected behavior of a target variable, we are also able to make real-time deviation detection. Real-time deviation detection refers to a scenario where new data points are received, as a stream, and decisions about whether a deviating behavior is detected, should be made as soon as it is possible.

Behavior detection:

- Feature selection. The feature selection component is responsible for the detection of the most important feature variables. Automatic detection is implemented through an l1-norm penalized linear model (Lasso) as a feature selector. To be able to exploit domain knowledge, manual detection is also possible.
- Binning. This refers to partitioning the dataset into bins based on the values of an input variable and to train specific models on each bin. This leads to more expressive models, one for each bin, because of the different kind of dependencies that may occur in different value ranges of a specific variable.
- Model tuning. This component is responsible for tuning the hyper-parameters of each model. There are two main options for tuning: grid-search, which exhaustively tests all different configurations and keeps the best one and randomized-grid-search, which tests a random sample of configurations and selects the best.
- Direct modelling. This component refers to methods modelling the variable that represents the behavior of the time-series at the given point, as the dependent variable.
- Indirect modelling. This component includes solutions that work by modelling the behavior of the time-series during (labelled) periods and then assigning labels depending on how well those models approximate the behavior of the newly incoming data.
- Behavior detection. Refers to the detection of the behavior of newly incoming data based on the available labels and utilizing an aggregate of the predictions of the multiple models trained in different bins or regions of the training datasets.

Motif discovery:

- Motif extraction on summarized time series. This component aims at increasing the efficiency of the Matrix Profile based pattern extraction/event detection methods, by executing the calculations directly on the summarized time series data stored in ModelarDB.
- Motif extraction for deviation detection. This component examines an alternative to the main methods for deviation detection presented above. This alternative exploits the Annotation Vector scheme of the Matrix Profile in order to focus the motif search on time series areas that are more likely to present a gradual degradation of power (due to soiling), taking into account several variables of the multidimensional time series.

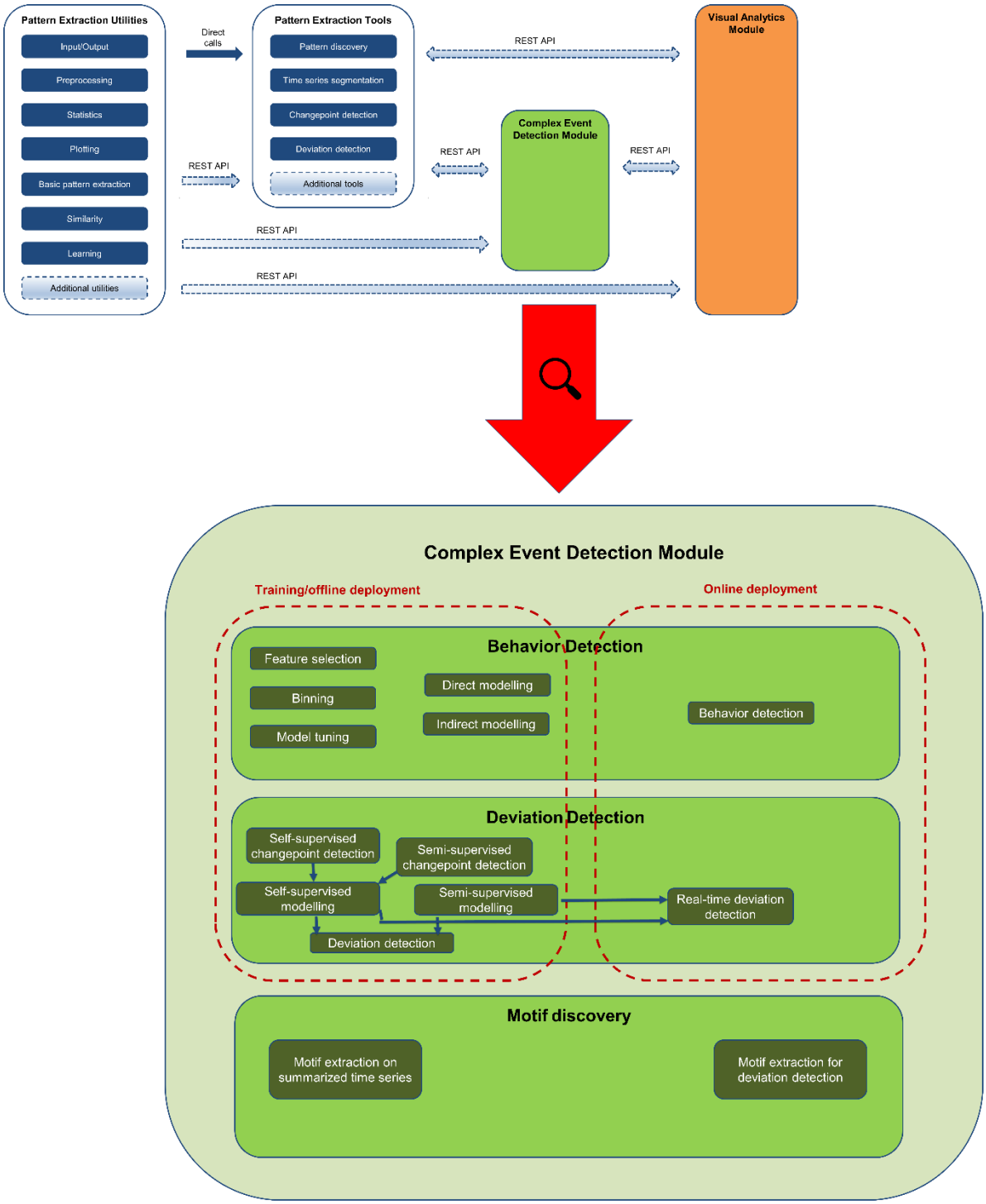

Figure 1: High level architecture of the Event Detection Module

4. Complex events: deviation detection

In this section, we present our methods for the detection of time periods, where one specific variable of the time series behaves significantly more differently than expected. The problem is motivated by the task of detecting soiling in solar panels, given only a time series of measurements of a minimal number of variables, which are considered reliable, e.g., power output, irradiance, module temperature, precipitation. A preliminary solution on this problem was proposed in D4.1. The proposed solution in D4.1 detects time periods in historical data, in the sense that the input time series is entirely given before our computations start. In this deliverable, while we keep some of the core ideas of the proposed solution of D4.1, we focus our attempts on building a framework which also supports real-time detection, meaning that the input is continuously supplemented by new data points as a stream, and the user must be able to extract information about the current instance in (near) real-time. To be able to make any decision regarding deviating behavior of a variable, it is sufficient to be able to make accurate predictions about the expected behavior of the given variable. Therefore, we focus on training accurate models that capture the expected behavior of the variable.

We propose solutions which are either semi-supervised, in the sense that they assume a small number of labelled data, not sufficiently large to train a model, or self-supervised, meaning that no labelled data are required, but the solution implicitly extracts labels so that it can treat part of the problem in a supervised manner.

More formally, we consider as input a multi-variate time series, where one variable is flagged as the "target" variable, i.e., the variable whose deviating behavior we are looking to detect, and the other variables constitute the "feature" variables, meaning that the target variable depends on them. Moreover, we are given as input a set of segments of the input time series, denoted W, corresponding to time intervals where changepoints of the target variable may lie. We assume that all implicitly given changepoints are either positive or negative; we call a changepoint positive if it affects the target variable "positively" in the sense that the target variable switches from a non-expected behavior to its expected behavior, while we call a changepoint negative when the target variable switches from its expected behavior to a non-expected behavior. From now on, we assume that only positive changepoints are implicitly given, since the other case is symmetric. Apart from the segments which potentially contain positive changepoints, we are also (optionally) given a set of timestamps C, where we know with certainty that the target variable behaves as expected. One crucial parameter of our method is that of the minimal length of a time period which can be assumed that it is not long enough to affect the expected behavior. While changepoints describe rapid changes in the target variable's behavior, our ultimate task is to detect periods of slowly progressing deviating behavior. This deviating behavior can also be positive or negative (or both), depending on whether it can be described as an increase or decrease of the actual values, compared to the expected values of the target variable. For simplicity, and since the other case is symmetric, we present our methods for the case of negative deviation. Our ultimate goal in this task is to exploit all relevant information given as input to train accurate regression models capturing the expected behavior of our target variable.

In Section 4.1, we present our research contributions, in Section 4.2, we discuss details regarding the implementation of our methods, and in Section 4.3, we include use instructions. Section 4.4 presents experimental results on the effectiveness and scalability of the implemented methods.

4.1. Research contribution

In this section, we give a detailed description of our methods for deviation detection. In Section 4.1.1, we summarize our method for detecting changepoints, which has appeared in D4.1, and we discuss improvements introduced in this deliverable. In Section 4.1.2, we discuss the latest version of our

method for detecting time periods where significant deviation is observed, and in Section 4.1.3, we discuss aspects of our method related to supporting real-time detection.

4.1.1. Changepoints

We first summarize the changepoint detection tool, as described in D4.1. Changepoints are generally defined as rapid changes in one variable's behavior. Our changepoint detection tool aims to verify candidate changepoints in a time series. Given a set of segments of the input time series, corresponding to time periods where a changepoint may lie, our tool assigns scores to those segments, which represents the possibility of containing a changepoint. This is done by investigating each 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. 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.

In this deliverable, we consider one more method for verifying changepoints. We train a regression model on time periods succeeding the given timestamps C, where we know that the target variable behaves as expected. This is parameterized by the length of the window defining those training periods. The intuition is that during such a period of time, the target variable typically behaves as expected, therefore, the regression model captures the target variable's expected behavior. Now, in the case of verifying positive changepoints, we make predictions before and after each segment, and we assign higher scores to those segments, which satisfy the following: i) the prediction error for a time period succeeding the given segment is small, and ii) the prediction error for a time period preceding the given segment is large. Moreover, in this deliverable, we consider one additional metric to estimate the difference between predicted values and true values, namely the median error, which is sensitive to the sign of each individual difference, a property needed when dealing with changepoints or deviating periods that are either positive or negative.

The two methods, apart from being slightly different in concept, they also have different requirements regarding the accessible data. The first method is self-supervised, since it translates the original problem to a supervised one, based on the following observation: if there is a changepoint in a sufficiently short segment, then it should be that the target variable deviates from its expected behavior in the preceding (resp. succeeding) period of time. On the other hand, the second method is semi-supervised, because it exploits the minimal information provided in the form of C. A detailed description of both methods follows.

4.1.1.1. Method 1.

The method requires the following parameters: the length of the training period w_1, the length of the validation period w_2, the length of the test period w_3, and a threshold parameter $thrsh$. For each input segment defined by a time interval $[t, t']$, we fit a regression model in the time interval $[t - w_1 - w_2, t - w_2)$, we validate it in the time interval $[t - w_2, t)$ and we test it in the time interval $(t', t' + w_3]$.

For each timestamp j, let P_j be the true value of the target variable associated with j, and let $\overline{P_j}$ be the predicted value of the target variable associated with j. We compute the function $mape_0([t, t']) = \frac{mean(\{|P_j - \overline{P_j}| \mid j \in [t,t']|\})}{mean(\{|P_j| \mid j \in [t,t']|\})}$ on the validation interval and if the returned value is greater than 5% then we consider this event invalid and we discard it from further consideration. This

threshold aims to discard events that we are unable to classify with a good amount of certainty. Due to the nature of our task, the regression model is required to make very accurate predictions and detect power deviations at a very small scale. Our choice of 5% is the result of experimenting with real data, but smaller percentage values may be more appropriate for different regression models or time series of different granularity and/or large amounts of missing data.

The intuition now is that if there is a positive changepoint in $[t, t']$, then the regression model captures the non-expected behavior and it underestimates the target variable's values in $(t', t' + w_3]$. To compute the score of this potential positive changepoint, we first compute the median error on the validation interval, denoted $mede([t - w_2, t))$ and the median error on the test interval denoted $mede((t', t' + w_3])$. The score function is then defined as $CPscore1([t, t']) = \frac{mede((t', t'+w_3]) - mede([t-w_2, t))}{|(mede([t-w_2, t]))|}$, that is the percentage difference of the two median errors. Positive changepoints are then detected in all intervals $[t, t']$ with $CPscore1([t, t']) > thrsh$.

Pseudocode for Method 1 can be found in Algorithm 1.

```
def changepoints_detection1(TS, W, w_1, w_2, w_3, thrsh, target, feats):
    """
    TS: input time series
    W: intervals/segments to determine if they contain a changepoint
    w_1 = number of days defining the training set
    w_2 = number of days defining the validation set
    w_3 = number of days defining the test set
    thrsh = threshold on the score values determining that a segment
contains a changepoint
    target = name of the target variable
    feats = list of names of the feature variables
    """
    changepoints = []
    for each [t,t_prime] in W:
        TS_train = TS[t-w_1-w_2, t-w_2)
        TS_val = TS[t-w_2, t)
        TS_test = TS(t_prime, t_prime+w_3]
        M = fit_model(TS_train, target, feats) #fit regression model
        P_val = predict(M, TS_val) #validate model
        if mape_0(P, P_val, [t-w_2,t)) < 0.05: #validation error below 5%
            P_test = predict(M, TS_test) #predictions after the segment
            error2 = mede(P, _test,[t_prime,t_prime+w_3))#median error
            error1 = mede(P, P_val,[t-w_2,t)) #median error
            score = (error2-error1)/|error1|
            if score > thrsh:
                changepoints = changepoints.append([t,t_prime])
    return changepoints
```

Algorithm 1: Method 1 for the detection of cleaning events.

4.1.1.2. Method 2.

This method requires four input parameters: the length w_1 of a testing time period before an input segment, the length w_2 of a testing period following an input segment, the length w_3 of periods defining the training set, and a threshold parameter $thrsh$. Recall that we are given as input a set of timestamps C, on which, we know with certainty that the target variable behaves as expected. Parameter w_3 denotes the length of the time period following each given timestamp, during which the target variable is assumed to behave as expected. We train one regression model on the set of points defined by timestamps in the union of time intervals $[t, t + w_3]$, where t takes values in the set of input timestamps. This model aims to capture the target variable's expected behavior. For each $[t, t']$

in W , we use our model to make predictions on $W[t-w_1,t)$ and $(t',t'+w_2]$. If $mape_0\big((t',t'+w_2]\big)$ is greater than 5% then we consider this interval invalid and we discard if from further consideration. As in Method 1, this filtering step is to avoid considering events that our models fail to classify with a good amount of certainty.

The intuition now is that if $[t,t']$ contains a changepoint, then the target variable's behavior during $[t',t'+w_2]$ must resemble the expected behavior as predicted by our regression model. Similarly, if negative deviation is observed during $[t-w_1,t)$, then the induced $mede\big([t-w_1,t)\big)$ must be a negative number of substantial magnitude. Therefore, in case of a changepoint, the percentage difference of the two errors tends to be a small negative number. We compute as score the negated percentage difference of the two errors: $CPscore2([t,t'])=\frac{-\big(mede([t-w_1,t))-mede([t',t'+w_2))\big)}{|mede([t',t'+w_2))|}$. Positive changepoints are then detected in all intervals $[t,t']$with $CPscore2([t,t'])>thrsh.$

Pseudocode for Method 2 can be found in Algorithm 2.

```
def changepoints_detection2(TS, C, W, w_1, w_2, w_train, thrsh, target,
feats):
    """
    TS: input time series
    W: intervals/segments to determine if they contain a changepoint
    w_1 = number of days defining the test set before the segment
    w_2 = number of days defining the test set after the segment
    w_train = number of days of the periods defining the test set
    thrsh = threshold on the score values determining that a segment
contains a changepoint
    target = name of the target variable
    feats = list of names of the feature variables
    """
    TS_train = []
    for each [t,t_prime] in C:
        TS_train.append(TS[t_prime,t_prime+w])
    M = fit_model(TS_train, target, feats) #fit regression model
    changepoints = []
    for each [t,t_prime] in W:
        TS_before = TS[t-w_1, t)
        TS_after = TS(t_prime, t_prime+w_2]
        P_before = predict(M, TS_before) # test model before
        P_after = predict(M, TS_after) # test model after
        if mape_0(P, P_after, [t_prime,t+w2)) < 0.05: #validation error
below 5%
            error2 = mede(P, P_after,[t_prime,t_prime+w_2)) #median error
            error1 = mede(P, P_before,[t-w_1,t)) #median error
            score = -(error1-error2)/|error2|
            if score > thrsh:
                changepoints = changepoints.append([t,t_prime])
    return changepoints
```

Algorithm 2: Method 2 for the detection of cleaning events

4.1.2. Deviation detection in historical data

In this section, we present our approach for the detection of periods where a slowly progressing, negative deviating behavior of the target variable is observed. In particular, we assign scores to segments of the time series, evaluating with a high score time periods during which, the target variable observes a decreasing trend compared to what is expected. For each maximal time period that contains no changepoints, we first compute an estimate of the rate of progression of the

phenomenon. If the estimated rate of progression is negative, then the time period is suspected for deviating behavior. Then, the factor that quantifies the severity of the problem within a time period is an estimated value which takes into account all individual differences between actual values and expected values. Similarly to our changepoints detection methods discussed above, we fit a regression model that aims to capture the expected behavior of the target variable. To this purpose, we choose as training set, a set of input points with timestamps from one of the following three sets:

i) $\cup_{t \in C} [t, t+w]$,
ii) $\cup_{[t,t'] \in W_1} [t', t'+w]$,
iii) $\cup_{[t,t'] \in W_2} [t', t'+w]$,

where w is a user-defined parameter, W_1 is the set of segments returned by Method 1, and W_2 is the set of segments returned by Method 2. The training set i) leads to a model which is semi-supervised, since it relies on the minimal information provided in C. The same holds for iii), while ii) leads to a model which is either self-supervised or semi-supervised depending on whether C is provided as input. A more detailed discussion on the different input assumptions, made by the different model variants, is diverted to the end of this section.

Each of W_1, W_2 defines a different set of *stable* segments of the input time series which correspond to maximal time periods, non-intersecting W_1 or W_2, which do not contain any changepoint. In each stable segment, we consider the time series defined by the differences between real values of the target variable and predicted values, by one of the three different regression models above, and we compute a slope using the Theil–Sen method [S68,T92]. The Theil-Sen method is a way of fitting a line to a set of points, which is robust to outliers. The line is chosen by selecting the median slope over all lines defined by pairs of points. If the slope is negative, then the segment is suspected of deviating behavior. To each segment with a negative slope, we assign a score, quantifying deviation within that segment. For that purpose, we propose the following three score functions:

1. $lossmedian([t,t']) = 1 - median\left(\left\{\frac{P_j}{\overline{P}_j} \mid j \in [t,t']\right\}\right)$,
2. $lossmean([t,t']) = 1 - mean\left(\left\{\frac{P_j}{\overline{P}_j} \mid j \in [t,t']\right\}\right)$,
3. $lossaggregate([t,t']) = \sum_{j \in [t,t']} \left(1 - \frac{P_j}{\overline{P}_j}\right)$,

where t, t' are two timestamps, P_j is the true value corresponding to the target variable, for timestamp j, and $\overline{P}_j$ is the predicted value corresponding to the target variable, for timestamp j.

The function $lossmedian$ is based on the median ratio of true values divided by predicted values within the time interval under consideration. The choice of median, makes the score function robust to prediction outliers. However, this score function is sensitive to how the segmentation is produced, because segments with a long period of stable performance followed by a rapid decrease, are more likely to get assigned to a small score. While this is mitigated by computing the mean, as in $lossmean$, the resulting score function is now more sensitive to inaccurate predictions. Finally, $lossaggregate$ calculates the sum of modelled losses over all timestamps within the time interval under consideration. This score function evaluates preferably intervals with a large number of deviated predictions. An experimental comparison of the three loss functions can be found in Section 4.4.2.4.

The two different ways of segmenting the time series and the three different ways of obtaining a regression model for the expected target variable's behavior, yield different variants of our algorithm for the detection of deviating periods. Notice that Method 1 does not require having C as input. It simply relies on having a set of potential changepoints. Therefore, using W_1 to segment our time series, and $\cup_{[t,t'] \in W_1} [t', t'+w]$ to train our regression model, we obtain a variant that does not depend on having C.

4.1.3. Real-time deviation detection

Our approach for detecting deviating behavior of a target variable can be applied in a real-time scenario. In real-time scenarios, new measurements arrive as a stream, and we need to detect events as soon as they are detectable. The main assumption here is that sufficient training data are available, so that the two methods of Section 4.1.1 yield meaningful changepoints, which can be then used to define training sets for regression models that capture the expected behavior of the target variable, as discussed in Section 4.1.2. Having models which can accurately predict the target variable, is the main building block for real-time detection scenarios.

4.2. Implementation information

The deviation detection tool is implemented in Python, currently as a Jupyter[2] notebook linked to modules developed in the scope of D4.1-D4.3. For efficiently managing, preprocessing, and performing numerical operations on the data, we use pandas[3], scikit-learn[4] and numpy[5]. The notebook can be found at: https://github.com/MORE-EU/complex-event-detection/blob/main/notebooks/tools/deviation_detection_basic.ipynb. In particular, the notebook specializes our methods on the soiling use case.

Apart from methods developed in D4.1, new functions have been defined aiming to support the new functionalities introduced in this deliverable, and the corresponding experimental evaluation. A list of the new functions follows:

- **get_ts_line_and_slope:** Wrapper function computing the Theil-Sen regressor.
- **calc_changepoints_one_model**: Returns errors associated with changepoint detection in a given segment, as described in Section 4.1.1.2.
- **calc_changepoints_many_models**: Returns errors associated with changepoint detection in a given segment, as described in Section 4.1.1.1.
- **output_changepoints**: Given the output of one of the functions calc_changepoints_one_model, calc_changepoints_many_models, returns a dataframe containing all relevant information about changepoints.
- **plot_segment**: Given a segment of the input time series, it makes plots comparing the original curve of the target variable against the expected curve.
- **score_segment:** Returns three values corresponding to the three scoring functions defined in Section 4.1.2.

Our main technical contribution is to provide an implementation with multi-fold functionality, in the form of a Jupyter notebook. The notebook implements all different variants of methods described in Section 4.1. In particular, the training of the regression model on periods following the manual cleanings of the solar parks is in cell [8]. Then, detection of changepoints, following both Methods 1 and 2 can be found in cell [11]. Cell [12] and cell [13] implement training of regression models in periods following changepoints obtained by Method 1 and Method 2 respectively. Cell [16] implements the assigning of scores to segments, which are obtained by segmenting the time series using changepoints from Method 1. Scores use as reference one of the three models of the expected power output. The default option there, is the model trained after changepoints computed by Method 1, but the user can easily switch to any of the other two options. Similarly, cell [20] implements the

[2] https://jupyter.org/

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

[4] https://scikit-learn.org/stable/

[5] https://numpy.org/

assigning of scores to segments, which obtained by segmenting the time series using changepoints from Method 2. The default option for the reference model there, is the model trained after changepoints computed by Method 2, but the user can easily switch to any of the other two options.

4.3. User guide

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/.

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/complex-event-detection --recursive

One new folder under the name complex-event-detection will be created. Let <path> be the absolute path to that folder, i.e., the new folder is <path>/complex-event-detection.

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 - 1.3.3
- matplotlib - 3.4.2
- numpy - 1.20.1
- scikit_learn - 1.0.2

The above libraries are included in the requirements.txt file. 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 must select and run the following Jupyter notebook file:

<path>/ complex-event-detection/notebooks/tools/deviation_detection_basic.ipynb

4.3.2. Configuration options

In this section we list a set of parameters which allow the user to test different configurations that correspond to different variants of our methods.

- **w_train**: The parameter defining the length of training periods, as a number of days. Corresponds to variable w in Section 4.1.2. It can be found and changed in cell [8].
- **feats**: List of the names of feature variables for the regression model. It can be found in cell [11].
- **target**: The name of the target (dependent) variable for the regression model. It can be found in cell [11].
- **wa1**: Number of days defining the training set (before each segment investigated for changepoints), for Method 1. Corresponds to variable w_1 in Section 4.1.1.1. It can be found in cell [9].
- **wa2**: Number of days defining the validation set (before each segment investigated for changepoints), for Method 1. Corresponds to variable w_2 in Section 4.1.1.1. It can be found in cell [9].
- **wa3**: Number of days defining the test set (after each segment investigated for changepoints), for Method 1. Corresponds to variable w_3 in Section 4.1.1.1. It can be found in cell 9.
- **wb1**: Number of days defining the test set, before each segment investigated for changepoints, for Method 2. Corresponds to variable w_1 in Section 4.1.1.2. It can be found in cell [9].
- **wb2**: Number of days defining the test set, before each segment investigated for changepoints, for Method 2. Corresponds to variable w_2 in Section 4.1.1.2. It can be found in cell [9].
- **error_br_column**: Type of error used in estimating prediction error before each segment. The default option is the median error, which is used in calculating the score functions $CPscore1$, $CPscore2$, as described in Section 4.1.1. All options are:
    - 0: r squared error
    - 1: mean absolute error
    - 2: mean error
    - 3: mean absolute percentage error
    - 4: mean percentage error
    - 5: median error

  It can be found and changed in cell [9].
- **error_ar_column**: Type of error used in estimating prediction error before each segment. The default option is the median error, which is used in calculating the score functions $CPscore1$, $CPscore2$, as described in Section 4.1.1. All options are:
    - 0: r squared error
    - 1: mean absolute error
    - 2: mean error
    - 3: mean absolute percentage error
    - 4: mean percentage error
    - 5: median error

  It can be found and changed in cell [9].
- **thrsh**: Segments with scores larger than thrsh are classified as containing a changepoint. It can be found in cell [9].

- **ref_model**: This parameter specifies the model used in detecting segments with deviating behavior. This models the expected behavior of the target variable, and it can get assigned to any of the three following options:
    - model1: Model trained after changepoints computed by Method 1.
    - model2: Model trained after manual cleanings.
    - model3: Model trained after changepoints computed by Method 2.

  In cell [16], this parameter determines the execution of the method variant where the segmentation of the time series is obtained using changepoints from Method 1. In cell [20], this parameter determines the execution of the method variant where the segmentation of the time series is obtained using changepoints from Method 2.

4.3.3. Deployment instructions and options

The input time series is provided in the form of .csv file. This is defined in cell [2]. Moreover, in cell [2], the user can define the periods during which the solar panels were manually cleaned. In particular, dates_wash_start contains the exact dates on which manual cleanings were initiated, and dates_wash_stop contains exact dates on which manual cleanings were terminated. The input .csv file is expected to contain columns under the names defined in parameters target, and feats, but it is also expected to contain a column named 'precipitation'. Cell [14] calculates the modelled soiling ratio for each of the three model variants, and outputs a plot containing all of them. Cell [18] outputs the detected segments, for the method variant that uses changepoints obtained by Method 1 to segment the time series. Cell [22] outputs the detected segments, for the method variant that uses changepoints obtained by Method 2 to segment the time series.

4.4. Evaluation

We test our deviation detection methods in the soiling use case. In this context, the target variable is the power output of the PV modules, while the feature variables are irradiance and module temperature. The set of segments W which potentially contain changepoints, is the set of segments corresponding to raining periods or periods during which PV modules were manually cleaned. The set C containing timestamps where power output behaves as expected, is the set of timestamps corresponding to the ending dates of manual cleanings. In this context, changepoints are *cleaning events*, i.e., rains or manual cleanings which yield a positive effect in power output by effectively washing the panels. We employ both real and synthetic datasets and we study various aspects of performance and accuracy.

We compare our methods with a state-of-the-art method for estimating soiling losses, namely the stochastic rate and recovery (SRR) method [DMM18]. SRR uses an analytical formula to compute the expected power output, which is then used to compute a performance metric. The method first detects soiling intervals in a dataset, and then, based on the observed characteristics of each interval, estimates the total loss. Notice that SRR provides an overall estimate of soiling loss, while our focus lies on detecting intervals with severe soiling. However, the intermediate step of segmenting the time series into soiling intervals is present in both approaches. Therefore, we juxtapose the results of our changepoint detection/segmentation with those of SRR.

Section 4.4.1 contains a thorough description of the dataset which was mainly used for evaluation. In Section 4.4.2 we report our experimental results for the evaluation of effectiveness of our methods, and in Section 4.4.3 we discuss experimental results showing scalability of our methods.

4.4.1. Evaluation dataset

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 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).

To evaluate our methods, we use a dataset provided in [MAD+14], which contains a set of current-voltage (I-V) curves and associated meteorological data for PV modules representing all flat-plate PV technologies and for three different locations and climates for approximately one-year periods. For each location, we are given values for a normalized metric, called soiling derate, which compares the daily performance of a PV module to an identical PV module which is cleaned during daily maintenance. This is computed using measurements for short-circuit current and irradiance from these two identical PV modules (cleaned and not cleaned). Soiling derate is the result of dividing daily values of ampere-hours per kilowatt-hours per square meter Plane of Array (POA) irradiance for the not-cleaned PV module, by the corresponding values of the cleaned PV module. We emphasize that soiling derate measurements are only used for the evaluation of our methods and are not utilized as input in our method, nor in SRR. The time granularity is 5 minutes, and measurements are provided for all hours of daylight. The three locations are Cocoa, Florida, USA; Eugene, Oregon, USA; and Golden, Colorado, USA. PV modules in Cocoa and Eugene were cleaned when this was necessary in order to ensure that levels of soiling loss were maintained at a reasonable level. PV modules in Golden were not cleaned because frequent rains helped maintaining a reasonable level of soiling loss. Cocoa has a minimum soiling derate of 0.985, Eugene has a minimum soiling derate of 0.964, and Golden has a minimum soiling derate of 0.977. In our methods, we use measurements for the maximum power of the PV module in watts, the amount of solar irradiance in watts per square meter received on the PV module surface, the PV module back-surface temperature and the accumulated daily total precipitation. The dataset also provides dates on which all PV modules were cleaned. We apply our methods on one of the PV modules which were used to estimate the soiling derate, and in particular one which was not cleaned every day. We juxtapose our results with the soiling derate values. Our methods utilize Ridge Regression models. For those models, we use polynomial features of the 3rd degree and a regularization strength parameter $alpha = 10^{-4}$ during the fitting stages.

4.4.2. Evaluation of effectiveness

In this section, we evaluate our methods. We begin by evaluating our two variants for cleaning event detection and juxtaposing our results with those computed by SRR. We use as input, the time series measured at Eugene, because periods of declining performance are apparent, and because more significant soiling losses are observed. We note that PV modules at the Eugene site were cleaned on March 11, July 10, August 14, August 21, and August 26. As Figure 2 illustrates, right before the manual cleanings in August, a rapid drop in the performance is observed, mainly due to the fact that no significant precipitation is observed during July and August. Our second experiment offers a comparison between different ways of estimating the daily soiling ratio. We show that our models achieve better accuracy on predicting “clean” expected performance, comparing to the analogous model used in SRR. We proceed by evaluating our methods for detecting periods with severe soiling losses, on the same site. Then we compare the three score functions proposed in our method, by evaluating them on their ability to accurately rank segments with severe soiling. All of the aforementioned experiments can be found at https://github.com/MORE-EU/complex-event-detection/blob/main/notebooks/soiling_experiments/deviation_detection_labelled.ipynb.

We conclude with an experiment on a real-world scenario, where no labelled data are available. The notebook containing this experiment can be found at https://github.com/MORE-EU/complex-event-detection/blob/main/notebooks/soiling_experiments/deviation_detection_real.ipynb.

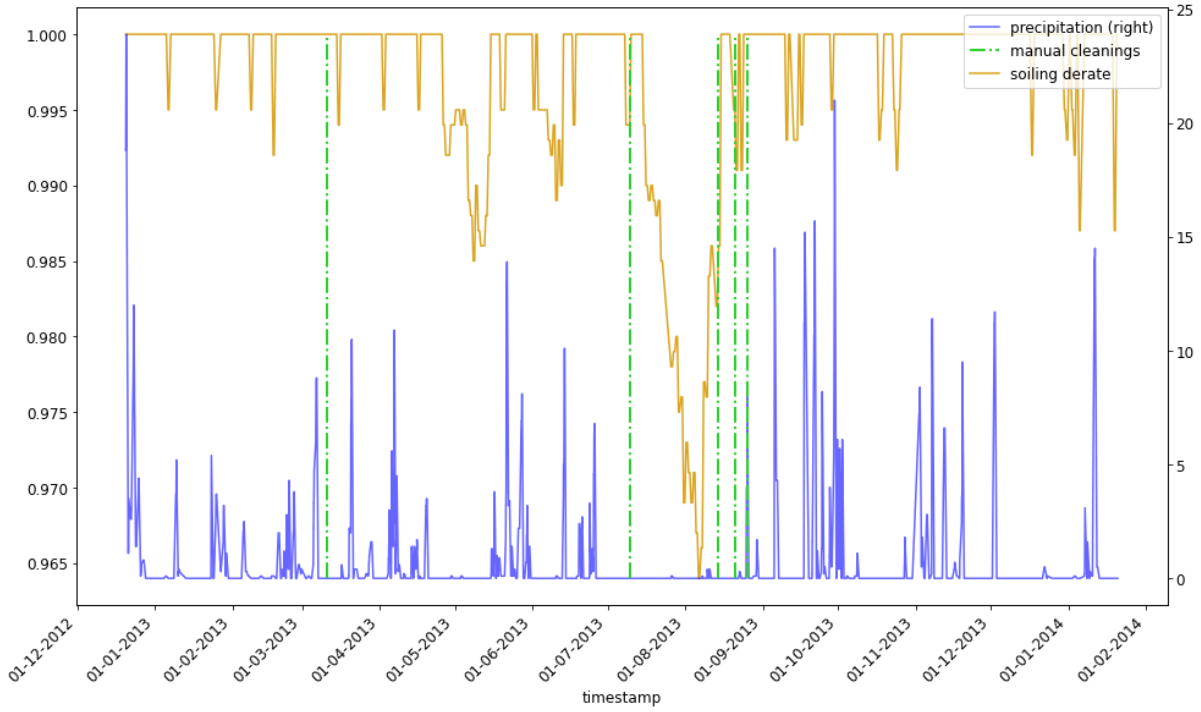

Figure 2 Soiling derate, precipitation and manual cleanings in the Eugene dataset.

4.4.2.1. Cleaning events

The first step in our method is the detection of cleaning events, i.e., changepoints in power output which partition the time series into soiling intervals. Similarly, SRR, before estimating the total loss, segments the time series into soiling intervals. This segmentation is important, since it provides a more detailed picture on how soiling progresses throughout time. We compare our output cleaning events with those detected by SRR, and we discuss differences of the two implementations. We apply Method 1 with parameters $w_1 = 10$ days, $w_2 = 5$ days, $w3 = 10$ days. The threshold parameter is $thrsh = 1$. More formally, the resulting set of cleaning events of Method 1 is $W_1 = \{[t, t'] \in W: CPscore1([t, t']) > 1\}$. We apply Method 2 with parameters $w_1 = 5$ days, $w_2 = 10$ days, $w_3 = 30$ days, $thrsh = 1$. More formally, the resulting set of cleaning events of Method 2 is $W_2 = \{[t, t'] \in W: CPscore2([t, t']) > 1\}$. Both of our methods operate on the dataset without any further preprocessing or filtering. We use the Python implementation of SRR, which is publicly available on Github[6]. In accordance with the discussion in [DMM18], SRR is employed on aggregate daily values calculated on measurements taken between 12:00 and 14:00, with irradiance greater than $500W/m^2$. SRR first computes a performance metric, as the ratio of realized to modeled PV energy yield, where modeled PV energy yield is estimated using standard formulas. Then, cleaning events are detected as positive shifts in a centered 14-day moving median of the performance metric. The output cleaning events consist only of those positive shifts that are outliers in magnitude.

Figure 3 shows the cleaning events detected by our Method 1 (blue solid vertical lines), the cleaning events detected by SRR (red dashed vertical lines) and the curve of soiling derate. We observe that our method is more sensitive in detecting cleaning events; examples are the shift lying in May 2013

[6] https://github.com/NREL/pv_soiling

and the shifts in January 2014. Figure 4 shows cleaning events detected by our Method 2, and cleaning events detected by SRR. Method 2 is less sensitive than Method 1 but it also detects the shift lying in May 2013, which is not detected by SRR. The good performance of our methods on detecting cleaning events is justified by the fact that our models capture the expected power output more accurately than the analytical formula used in SRR. This is further supported by the results of Section 4.4.2.2.

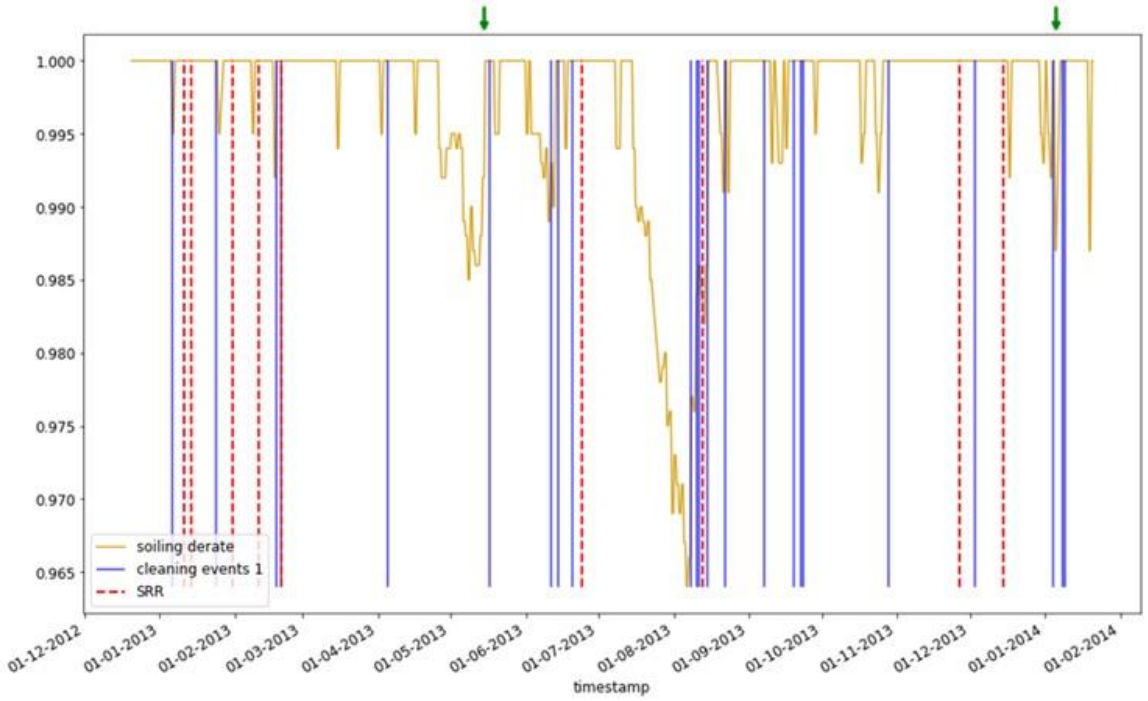

Figure 3 Cleaning events from our Method 1 and SRR. Green arrows indicate cleaning events not detected by SRR.

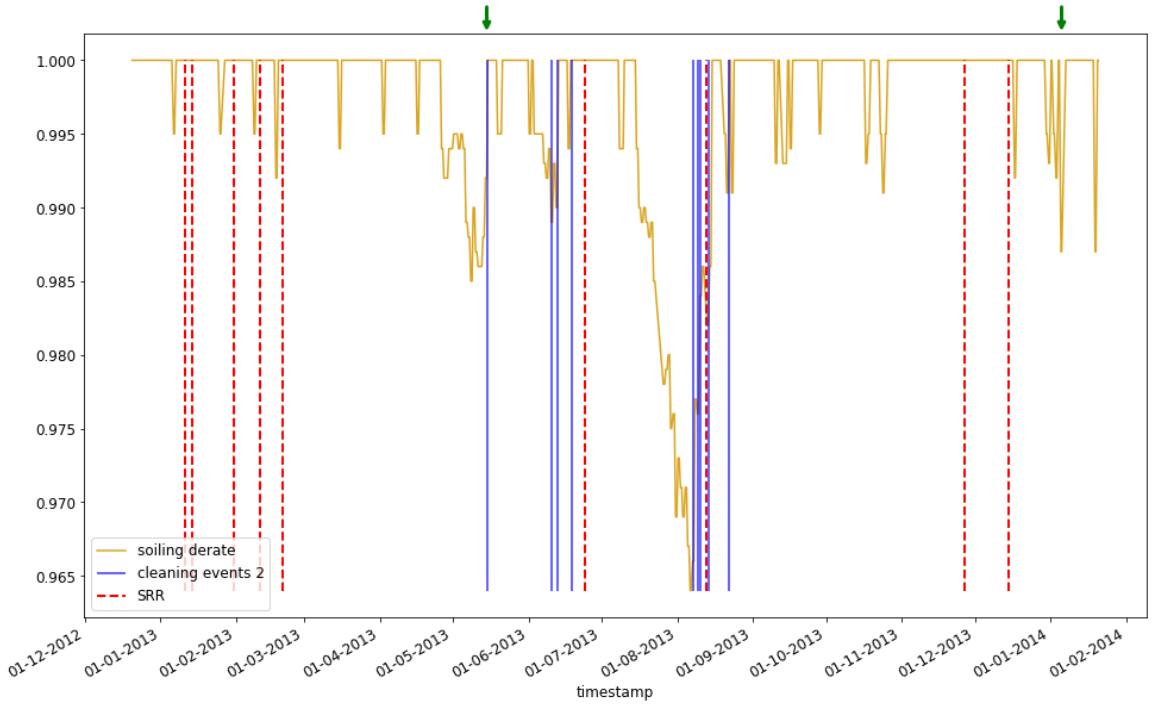

Figure 4 Cleaning events from our Method 2 and SRR. Green arrows indicate two of the cleaning events detected by Method 1. One is detected by Method 2, but none of them is detected by SRR.

4.4.2.2. Modelling soiling ratio

As described in Section 4.1.2, we can use the detected cleaning events to train regression models which represent the expected power output when the PV modules are clean. Using those models, we can estimate the soiling ratio, i.e., the ratio between the actual power output and the expected power output. Given a segmentation obtained by the output cleaning events of Method 1 or Method 2, executed with the same parameters as in Section 4.4.2.1, we train our regression model in periods of $w = 30$ days after the cleaning events or after the manual cleanings. We juxtapose our modelled soiling ratio with soiling derate and the performance metric used in SRR.

For each of our regression models, we compute a new time series by dividing the values of the true power output by the values predicted by the model, where we map negative values and values greater than one to zero and one, respectively. Then, we apply a rolling median with windows of one day. For SRR, we rely again on the publicly available implementation on Github[6]. We use as input aggregate daily values calculated on measurements taken between 12:00 and 14:00, with irradiance greater than 500W /m^2. We first compute the performance metric as the ratio of realized to modelled PV energy yield, where modelled PV energy yield is derived from a standard formula. Then, we perform a few processing steps as in [DMM18]: we normalize daily index to 95th percentile, we forward-fill the time series with a day scale limit (14 days), and we apply a rolling median with a window of 14 days. Due to the fact that forward-filling has a 14 days window limit, regions with missing values which are larger than 14 days will not be completely filled. Therefore, we additionally apply a forward/backward-filling operation to obtain a complete sequence. In Figure 5, we plot our modelled soiling ratio, for all three models discussed in Section 4.1.2, the soiling derate and the performance metric used in SRR. We also calculate the root-mean-square error (RMSE) comparing the soiling derate with each modelled ratio. We list these results in Table 1. It becomes evident, both from the RMSE values and from the visual inspection of the figure, that a better estimation of the soiling ratio can be derived by our models, compared to the model based on an analytical formula which is employed by SRR, in a setting where a soiling tendency needs to be detected, nearly real-time, on newly incoming data. Further, our two variants which use the results of Method 1 and Method 2 to define training sets, although performing the best, present slightly diverse behaviors, rendering each potentially preferable in diverse real-world settings, depending on the exact objective of a solar park operator. Specifically, using cleaning dates detected by Method 2 to define the training set provides the most accurate method in approximating soiling ratio, thus preferable in cases small to medium soiling events are tolerable by the operator, as long as “false alarms” are minimized. The model trained on periods after the cleaning events detected by Method 1, while slightly missing in accuracy, it is more sensitive in the detection of smaller (potential) soiling events, making it ideal in cases when event small soiling events need to be handled. On the other hand, we can see that the SRR derived estimation essentially predicts the majority of the considered time series period as soiling; a behavior with small practical use in a real-world deployment scenario.

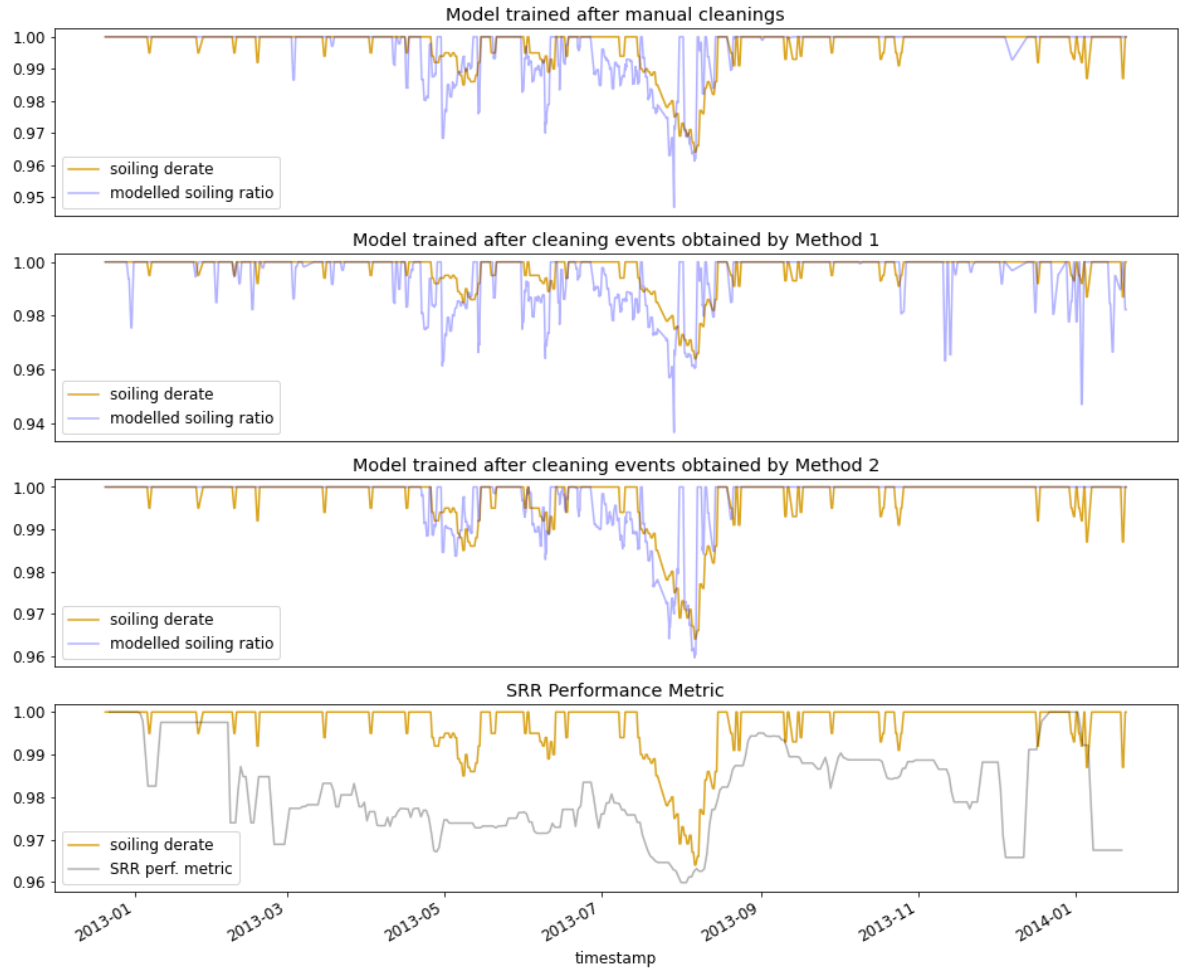

Figure 5 Soiling derate calculated from real measurements, the soiling ratio predicted by our models, and the performance metric used in SRR.

Model	**RMSE against soiling derate**
Expected power output calculated using regression model trained on periods of 30 days after manual cleaning dates.	0.009
Expected power output calculated using regression model trained on periods of 30 days after cleaning events obtained by Method 1.	0.007
Expected power output calculated using regression model trained on periods of 30 days after cleaning events obtained by Method 2.	0.005
Performance metric used in SRR.	0.018

Table 1 RMSE between ground truth soiling derate and modelled soiling ratio induced by our methods, and RMSE between ground truth soiling derate and performance metric used in SRR.

4.4.2.3. Periods with severe soiling

Next, we validate our method for the detection of periods with severe soiling. We consider both Method 1 and Method 2 for the time series segmentation, using the same parameter setting as described above. We train different models for representing the expected power output, as described in Section 4.1.2, and we apply the $lossmedian$ score function. We choose this score function as it works better in this case, a fact which is further supported by the discussion in Section 4.4.2.4. In particular, given a segmentation obtained by the output cleaning events W_1 of Method 1, we train our regression model in periods of $w = 30$ days after cleaning events W_1 or in periods of $w = 30$ days after manual cleanings of the panels. We output the intervals with non-negative score. Figure 6 illustrates the cleaning events W_1, and the output intervals of the two variants. Given a segmentation obtained by the output cleaning events W_2 of Method 2, we train our regression model in periods of $w = 30$ days after cleaning events W_2 or in periods of $w = 30$ days after manual cleanings of the panels. Figure 7 illustrates the cleaning events W_2, and the output intervals of the two variants. The results of Figure 6 indicate that training after cleaning events W_1 leads to a more sensitive model, as it also detects the last soiling interval which is not detected when the model is trained after manual cleanings C. The three drops in the performance between April 2013 and September 2013 are, however, detected by both variants. Figure 7 indicates that the two variants are equally sensitive, but the method detects only the two drops in the middle, because of the crude segmentation of the time series.

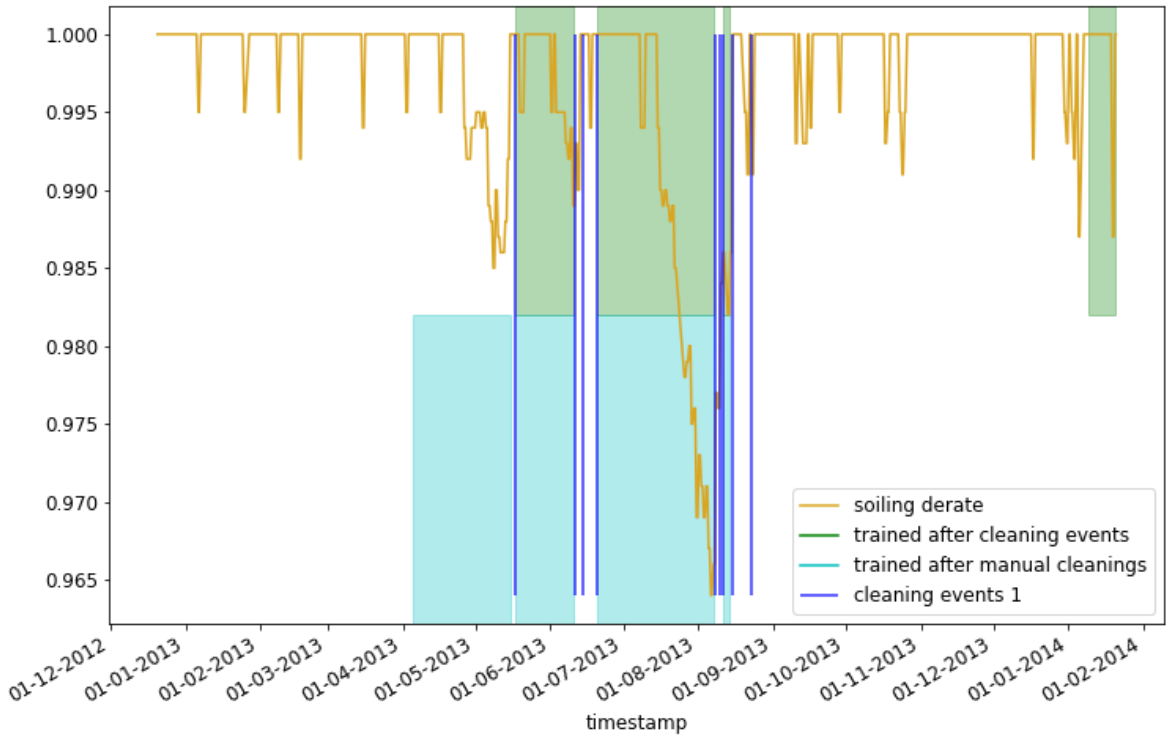

Figure 6 Intervals with severe soiling. Segmentation defined by cleaning events obtained by Method 1. Then, two variants are applied: the model is either trained after cleaning events obtained by Method 1, or after manual

cleanings.

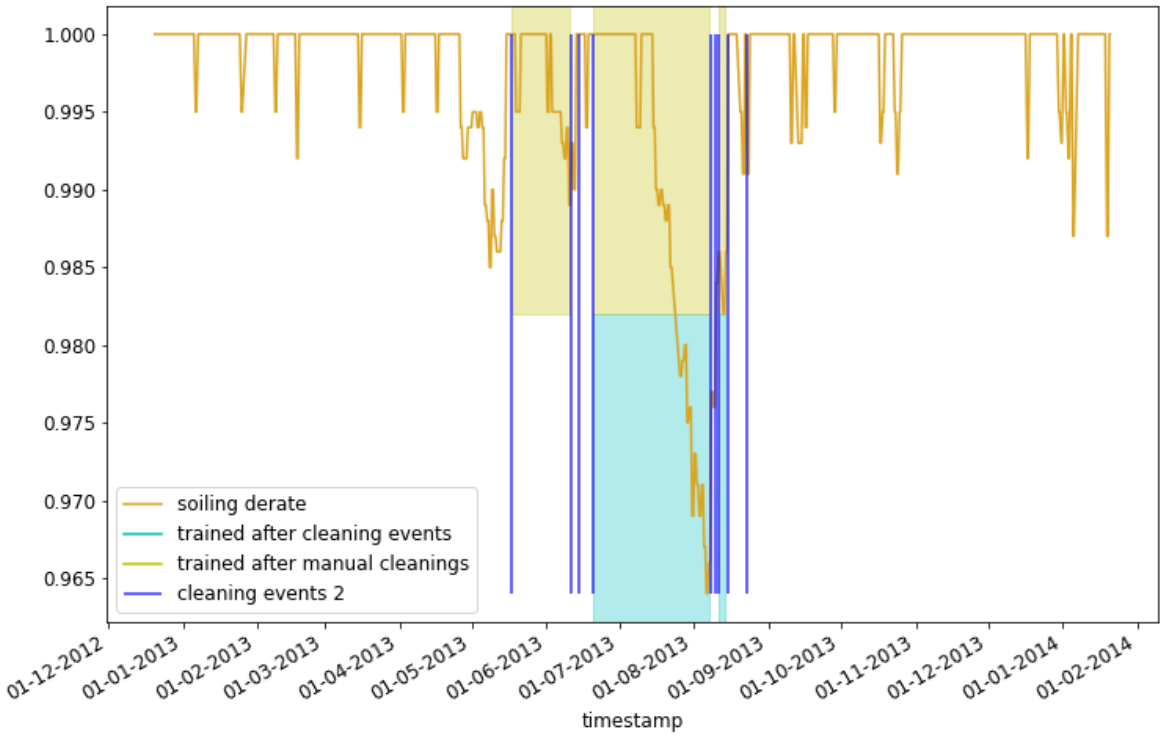

Figure 7 Intervals with severe soiling. Segmentation defined by cleaning events obtained by Method 2. Then, two variants are applied: the model is either trained after cleaning events obtained by Method 2, or after manual cleanings.

4.4.2.4. Ranking soiling periods

We evaluate the ranking of the detected intervals with respect to the different score functions proposed in Section 4.1.2. For different sets of cleaning events obtained by Method 1 and Method 2, we compare the ranked list of intervals (with a non-negative score) obtained by our methods, using regression models that are trained on periods of $w = 30$ days after the cleaning events, with the ranked list obtained by assigning the average soiling loss to each interval. The average soiling loss is calculated based on the soiling derate values. For the comparison, we use the Rank Biased Overlap (RBO) measure [WMZ10], a popular similarity measure for ranking of non-conjoint lists, which takes into account that different emphasis should be given on the different ranks of the list. RBO is parametrized by $p \in [0,1]$, which determines the weight associated with the top ranks. Table 2 lists RBO values for $p = 0.7,0.8$ and various parameters in our approach. Using a formula provided in [WMZ10], we are able to interpret our choices of p as follows: for $p = 0.7$, the top 3 ranks contribute approximately 84% to the RBO value, while for $p = 0.8$, the top 3 ranks contribute approximately 72% to the RBO value. We observe that the $lossmedian$ score yields considerably more accurate rankings, thus its adoption in the rest of the paper's experiments.

Method	p	$lossmedian$	$lossmean$	$lossaggregate$
CE1, $thrsh = 1$	0.7	0.298	0.081	0.088
CE1, $thrsh = 1$	0.8	0.328	0.124	0.148
CE2, $thrsh = 1$	0.7	0.279	0.279	0.068
CE2, $thrsh = 1$	0.8	0.288	0.288	0.120

CE1, $thrsh = 0.5$	0.7	0.295	0.265	0.030
CE1, $thrsh = 0.5$	0.8	0.322	0.290	0.073
CE2, $thrsh = 0.5$	0.7	0.286	0.135	0.044
CE2, $thrsh = 0.5$	0.8	0.302	0.188	0.093

Table 2 RBO values for various parameters and different score functions. CE1 denotes Method 1 and CE2 denotes Method 2 for the detection of cleaning events. $thrsh$ denotes the threshold on the score function defining cleaning events. For the ranking of intervals, the regression models are trained on periods of w=30 days after the cleaning events.

4.4.2.5. Real-world data

We now consider a real-world scenario, where no ground truth is available. We test our method on a dataset from a very different location and of different climate conditions, comprising measurements from a solar park located in Greece, and provided by InAccess. We are given values for power output, irradiance, module temperature and precipitation on a time granularity of 15 min for a period of approximately 7 years, and 15 dates of manual cleanings. We use our Method 1 for the detection of cleaning events, with parameters $w_1 = 10$ days, $w_2 = 5$ days, $w_3 = 10$ days, and $thrsh = 1$. For the detection of periods with soiling losses, we train our regression model on periods of $w = 30$ days after those cleaning events. We filter out rains with maximum precipitation of at most 0.8 and we consider time intervals lasting at least 10 days. Figure 8 illustrates the time interval with the highest $lossmedian$ value, as detected by our method. The time series (in blue) is defined by the difference between true power output and predicted power output. The orange dashed lines are the lines calculated by the Theil-Sen regression for the respective periods. We observe a decreasing trend in the time series for the detected period, as dictated by the slope of the line fitted by the Theil-Sen regression. This decreasing trend ends with a rain, illustrated by a light blue vertical span, which is detected by our method as a cleaning event. This example is an indication of the effectiveness and generalizability of the proposed method. Despite the lack of labels to be able to explicitly verify the result, the trend identified is consistent with soiling and it is verifiable through the effect of washing.

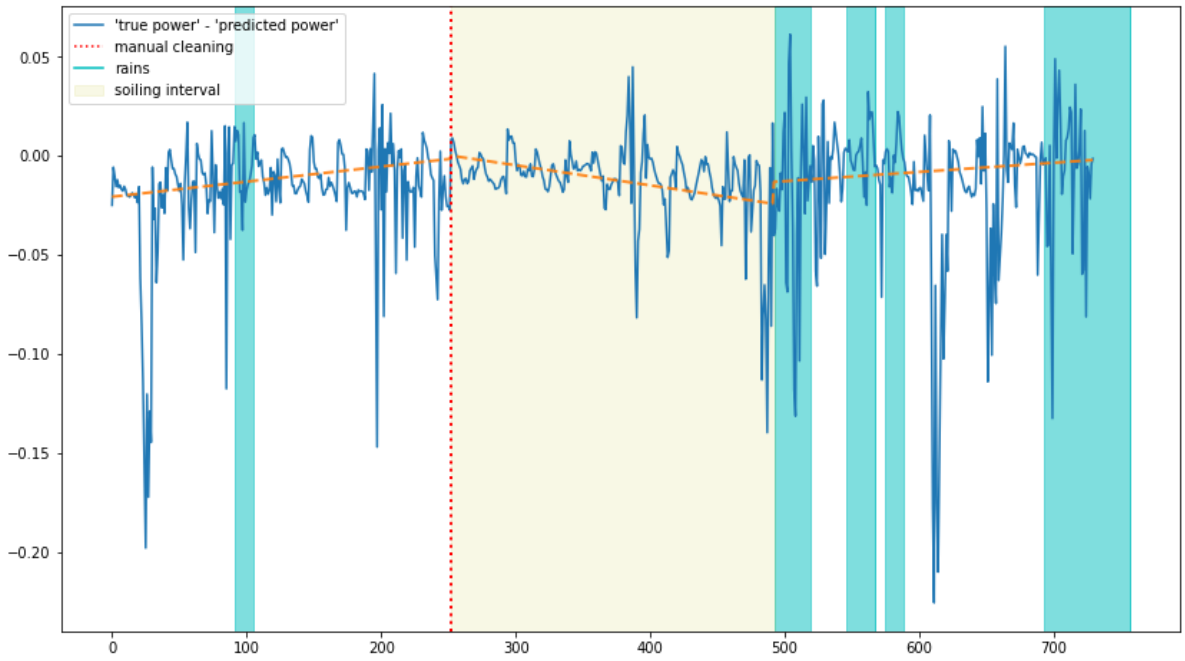

Figure 8 Detected soiling interval. The time series defined by the difference between true power output and predicted power output is depicted after the application of a rolling median of 5 days window size.

4.4.3. Evaluation of scalability

Our framework supports real-time detection, meaning that it supports a scenario where new data points are continuously arriving as a stream, and the user must be able to extract information about the current instance in (near) real-time. To claim efficiency of our methods in real-time detection, we conduct an experiment which simulates the real-time aspect as follows: given a large data stream as a regular time series, we process it in batches, where each batch of k points represents the last k points seen so far. Furthermore, to claim scalability of our approach, we adapt our methods to handle a scenario where multiple streams must be processed in parallel. In this consideration, decisions must be taken in real-time for multiple parks/sensors which transmit their measurements in a concurrent fashion.

In order to test our method on a large number of time series representing parallel streams, we construct a set of synthetic time series based on the Eugene dataset, which is discussed in Section 4.4.1. Given a segment of the Eugene dataset, we normalize it so that it has a granularity of 5 min in daily intervals between 07:00 and 17:00, we add Gaussian noise to each point and we compute its absolute value. Gaussian noise comes in the form of a random variable following the normal distribution with mean 0 and variance $\varepsilon \cdot \sigma^2$, where $\varepsilon > 0$ is a user defined parameter, and σ^2 is the variance of the variable/column in consideration. To create precipitation values, we follow the same approach, albeit conditioned on some user-defined probability p: with probability $1 - p$ a precipitation value is 0, and with probability p it is chosen by adding Gaussian noise to the respective precipitation value in the Eugene site, and then computing its absolute value. Repeating the above procedure several times, we create a synthetic dataset consisting of multiple time series which are then used as input in our methods.

Input time series are traversed in parallel; at each given instant we assume that a new batch of data points is revealed, in each one of the input time series. Data points are three-dimensional containing measurements of power output, irradiance and module temperature. Our task is to predict the expected power output for all points in all new batches, as this is the most basic operation needed for real-time detection. To simulate parallel computations by different nodes, we employ multi-core

parallel processing using Dask[7]. In particular, we use as input 100000 synthetic time series, each one representing a stream transmitted from a different source. The time granularity is 5 min, and each time series spans 70 days. In each day, we have measurements between 07:00 and 17:00. Overall, we have a sequence of 8470 points and we assume that a decision must be taken every 60 points, i.e., every 5 hours. Therefore, in each iteration we need to make predictions using a regression model in 100000 windows of length 60. This is implemented as a parallel computation by assigning batches of 200 windows to 30 cores, with maximum processor frequency at 3.7GHz, and available RAM at 256Gb. The average running time taken over all periods of 5 hours, for the standard sequential computation is 35.07 seconds, and the average running time of our parallel implementation is 10.34 seconds. The Jupyter notebook implementing this experiment is available at https://github.com/MORE-EU/complex-event-detection/blob/main/notebooks/parallelization_experiments/deliv_parallel_soiling.ipynb.

[7] https://dask.org/

5. Complex events: behavior detection

In this section, we present our methods for the detection and categorization of regions of time series that show a specific behavior. Given a labelled multivariate time series as our input, where each point is characterized by a label representing the current behavior of the time series, we aim to develop models that can learn those behaviors as well as distinguish between them, in order to accurately detect each one of those specific behaviors in either historical or in real-time incoming data. To do this we develop both direct and indirect methods that handle the aforementioned task; these methods work by utilizing regression algorithms that model the behaviors of the time series by learning on a labelled training dataset and then directly making predictions on the desired target variable that represents the behavior we examine (direct models) or aggregating approximations of several models on incoming (indirect models).

In addition, since the detection of the learned/modeled behaviors depend on the predictions of the various trained models, several evaluation schemes are considered, which aim to compare and aggregate the predictions of those models.

Furthermore, staying in line with the main focus of our project, that is solving real world RES problem scenarios, we apply and test our methods in the frame of the static Yaw Misalignment (YM) use case. For example, we utilize our direct models to directly predict the static YM of wind turbines on newly incoming test data using SCADA measurements as inputs. We also experiment with indirect models that approximate the power output of turbines in periods with specific YM values and then infer the YM of turbines in incoming data by comparing and aggregating the power output estimations we obtain from those models.

5.1. Research contribution

In this section, we give a detailed description of our methods for behavior detection. In Section 5.1.1, we summarize our machine learning workflow methods. In Sections 5.1.2, 5.1.3, we present our direct and indirect methods, explaining in-depth the way in which they operate during both training as well as test/prediction time.

5.1.1. Machine learning workflow

We begin by presenting a series of more general preprocessing, feature selection, and tuning procedures we have developed, which can be applied depending on the needs of the specific problem that one tries to tackle before moving on to the training and testing phases of our algorithms.

5.1.1.1. Preprocessing

Preprocessing is a crucial step in any data-driven machine learning pipeline, since the quality of the data that is used as an input to a machine learning algorithm directly correlates with the quality and the robustness a model trained on that data will exhibit. For this reason, in our tool, we offer a selection of preprocessing functionalities that the user can choose to apply before applying the machine learning methods. Those functionalities, aim to improve the performance of our models by removing outliers or other anomalies that are frequently present in real world datasets or by splitting the dataset into several subsets that allow us to train models that capture fine-grained nuances of the data.

We allow the user to choose a user-defined granularity in which the dataset will be resampled. Resampling is usually done to move the dataset to canonical lower-frequency intervals that remedy extreme fluctuations and noise that comes with data sampled by sensors with a very high frequency. For example, in our static Yaw Misalignment use case we resample the data from 2-second to 1-minute intervals; we also perform even more aggressive smoothing on some variables, however, the ideal

resampling frequency differs from dataset to dataset and from problem to problem, so we leave those numbers parametrizable by the users.

Our tool also allows the user to apply a series of domain-specific filters to remove any erroneous or redundant values found in the multi-dimensional time series. Methods for automatic outlier detection such as the Interquartile Method (IQR) and Local Outlier Factor (LOF) are also available. We provide multiple options for filtering and outlier removal, since users usually need to utilize a quite aggressive preprocessing pipeline, since the removal of noise, outliers and erroneous values is vital for the quality of models that are trained on real datasets.

Furthermore, inspired by [LJ21], we allow users to split the dataset into several bins depending on the values of an input variable and to train specific models on each bin. The size of the bins and the variable that will be used for the split is parametrizable by the user, the option of not splitting the dataset is also available. This functionality is important because different relationships between the variables are expressed in different value ranges of a specific variable e.g., in the context of static Yaw Misalignment, the power curve shows different behavior in different wind-speed ranges. Thus, the creation of bin-specific models would allow us to capture those differences and get more accurate results by utilizing the predictions of all those individual models.

We should also mention that we implement simple normalization operations such as the scaling of variables into the [0, 1] range with min-max scaling, to reduce the effects of large numerical differences between features before applying machine learning algorithms.

5.1.1.2. Feature selection

After passing the dataset through the previously mentioned preprocessing pipeline, a user can move on to perform an automatic feature selection step that lets us keep the most important features and remove the redundant ones. Since models that are penalized using the l1 norm yield sparse solutions by assigning zero coefficients to unimportant features, we use an l1-norm penalized linear model (Lasso) as a feature selector. More specifically, we utilize the LassoCV function from scikit-learn[8] to automatically find a good alpha regularization value and then, after training a Lasso model with that regularization parameter and all the features, we keep the features for which, the resulting coefficients are non-zero. Our tool also allows for manual selection of the variables that will be used as input features, which can be used for example by a domain expert when dealing with a specific task.

5.1.1.3. Model tuning

When working with multiple models that are trained on several bins of the dataset, that are of different size and that may contain observation values with significant differences between bins, it is unrealistic to find a specific set of algorithm hyper-parameters that will work reasonably well for each model. For this reason, we allow for the tuning of the hyper-parameters of each one of the models on the data of the respective bin. We assume that bins that result from similar constraints on the input variables, on different datasets, will yield similar bins of data. Depending on the model and the hyper-parameter search space that the user decides to utilize, there are two options for tuning: grid-search, which exhaustively tests all different combinations of the parameters and keeps the best performing combination and randomized-grid-search (used for models that are slower to train or for very large hyper-parameter spaces), which randomly tests a subset of all the possible combinations and selects the best. A user can parametrize the hyper-parameter search space depending on the specific problem he is trying to solve.

[8] https://scikit-learn.org

5.1.2. Direct Methods for behavior detection

In this section we will comprehensively describe our approaches that work by directly utilizing the variable that represents the behavior of the time series at the given point, as the dependent variable.

5.1.2.1. Training procedure

We begin by explaining the training procedure that we follow for our direct approaches. We define as $(x_1, \dots, x_n)$ the time series variables that were selected as the input features either by the feature selection algorithm or manually and y as our target variable which represents the behavior that we aim to detect. Using $(x_1, \dots, x_n)$ as the independent variables and y as the dependent one, k regression models are trained where k is the number of the bins that our dataset was split into. For each one of those k bins the model that corresponds to each bin is trained on randomly selected 80% of the data of the bin and then tested on the remaining 20% of the data to validate its generalization ability. Our training procedure allows for the models to be trained on several different datasets. This is critical because utilizing multiple datasets for training incorporates more possible behaviors that can be detected on newly seen data, consequently leading to better performance. The models are afterwards saved in the disk, so that evaluation/prediction methods can directly load and utilize them for detecting the learned behaviors in other datasets or in real-time incoming data. A pseudocode snippet that summarizes this method is available below (Algorithm 3).

```
def train_direct(D, B, fit_features, target_feature, scorer, model,
params):
  """
  D: input multivariate time series.
  B: set of indexes that index the separate bins D is split into.
  fit_features: the feature variables used as predictors
  target_feature: feature variable that we are trying to fit
  scorer: Score function used in grid search
  model: The selected regression model
  params: The sets of parameters to search
  """
  for bin_index in B:
    current_data = D[bin_index]
    train_data, test_data = train_test_split(current_data, 0.8, 0.2,
shuffle=True)
    hyper_params = perform_grid_search(train_data, fit_features,
target_feature, scorer, model, params)
    trained_model = fit_pipeline(df_train, fit_features, target_feature,
model, hyper_params)
    save(trained_model)
```

Algorithm 3: Training of direct method

5.1.2.2. Inference and evaluation

Here, the inference and evaluation procedures that we utilize to detect learned behaviors in new data will be discussed. The test dataset in which we want to detect the behaviors that our models have learned should be first handled in a similar manner as the training datasets regarding the preprocessing steps i.e., filtering, outlier removal, normalization, etc. Then, the trained models are loaded and for each bin b_i of the dataset the respective trained model m_i will be utilized to make predictions by taking the variables $(x_i', \dots, x_n')$ as inputs and yielding a prediction y_{pred} as its output. With the previous procedure we obtain a prediction y_{pred} for each point of the new dataset that characterizes its behavior. During our experimentation with real-world data, we observed that these per-point predictions often show very high-fluctuations that can be attributed to abnormal fluctuations in the input variables due to noise and sensor errors which have not been cleaned. For

this reason, a further post-processing smoothing step is adopted to additionally refine the results. This smoothing operation is done using a rolling median, whose window can be modified by the user depending on the specific task at hand, for example in the YM use case a 2-day window gives reasonable results. Finally, we evaluate the results of our method by calculating the Root Mean Squared Error (RMSE) and the Mean Absolute Error (MAE) between the ground truth values y_{gt} of the dataset and our predictions y_{pred}. The steps of the method we described in this section are also available as pseudocode in Algorithm 4.

```
def predict_direct(D, B, M):
  """
  D: input multivariate time series.
  B: set of indexes that index tha separate bins D is split into.
  M: set of trained models and metadata that is loaded
  """
  fit_features = M{'fit_features'}
  target_feature = M{'target_feature'}
  models = M{'models'}
  P = []
  for bin_no, bin_index in B:
    current_data = D[bin_index]
    m = model[bin_no]
    predictions = m.predict(current_data, fit_features, target_feature)
    P[bin_index] = predictions
  P = smoothen(P)
```

Algorithm 4: Prediction method with direct models

5.1.3. Indirect Methods for behavior detection

In this section we will comprehensively describe our approaches that work indirectly, by modeling the behavior of the time series during (labeled) periods and then assigning labels depending on how well those models approximate the behavior of the newly incoming data. Several aggregation schemes for inference and evaluation will also be discussed.

5.1.3.1. Training procedure

Regarding the training scheme utilized by our indirect methods, we should start by providing some definitions. Let $L = \{l_1, \dots, lc\}$ be a discrete set of c labels that describe the desired behavior of our system at each time point t_i. Also, for each time point t_i , we have a set of independent variables $(x_1, \dots, x_n)$ and a dependent variable y which is usually a variable whose value distribution is likely to be affected if it lies in regions that belong to different labels l_i. Thus, for each t_i we have a tuple $((x_1, \dots, x_n, y),\ l_i)$. Additionally, let $B = \{b_1, \dots, b_k\}$ be a set of k bins that our dataset has been split into (if $k = 1$ this devolves to a no-bins approach). Based on the different labels l_i , we split the dataset in c regions r_i each one corresponding to all the points that belong to l_i . For each region r_i and for each different bin b_j in each region, a regression model $m(l_i, b_j)$ is trained that utilizes $(x_1, \dots, x_n)$ as independent and y as the dependent variable, resulting in a set of $k \times c$ models. Similarly, to the previous method, all those models are trained on 80% of the training dataset (shuffled) and then tested on the remaining 20% to validate their performance. Afterwards the models are saved to disk. Regions of new time series are labeled according to the label (or a combination thereof) of the trained models that approximate them more precisely, as we will see in the following sections. Pseudocode for the training procedure employed by our indirect methods can be found in Algorithm 5.

```
def train_indirect(D, B, R, fit_features, target_feature, scorer, model,
params):
```

```
  """
  D: input multivariate time series.
  B: set of indexes that index tha separate bins D is split into.
  R: set of indexes that index labeled regions
  fit_features: the feature variables used as predictors
  target_feature: feature variable that we are trying to fit
  scorer: Score function used in grid search
  model: The selected regression model
  params: The sets of parameters to search
  """
  for r_index in R:
  current_data = D[r_index]
    for bin_index in B:
      current_data = current_data[bin_index]
      train_data, test_data = train_test_split(current_data, 0.8, 0.2,
shuffle=True)
      hyper_params = perform_grid_search(train_data, fit_features,
target_feature, scorer, model, params)
      trained_model = fit_pipeline(df_train, fit_features,
target_feature, model, hyper_params)
      save(trained_model)
```

Algorithm 5: Pseudocode for the training scheme of the indirect method

5.1.3.2. Indirect inference and evaluation: naïve method

The first method that we use to indirectly detect learned behaviors in a new (test) dataset, is by calculating a fitting score (that can be user defined i.e., MAPE) of each trained model on each bin, averaging the scores for each model over all bins, and finally assigning the label of the models with the best fitting average score to the region. More formally, we consider a part of the test dataset W, that can be a window of incoming data or a region in historical data. After preprocessing and binning the data of W in the same manner as we did in the training datasets, we proceed to make estimations in each bin b_j using models that correspond to all labels i.e., $m(l_1, b_j)$, $m(l_2, b_j)$, and so on and we get the average score over all bins for models that belong to the label l:

$$avg\_score(l) = \frac{1}{k}\sum_{j=1}^{k}\left(score\left(Y_{m(l,b_j)}, Y_W\right)\right)$$

where $Y_{m(l,b_j)}$ and Y_w are the predicted and the true values of the target variable respectively. Then we assign the label of the best avg_score to the region W:

$$label(W) = argmin_{l \in L}\left(avg\_score(l)\right)$$

This procedure can be applied to more regions W to cover the whole historical dataset if we consider that case. We call this method "naive", because it assigns the label of the best scoring models without considering the other available labels and how close the approximation of models belonging to those was, this yields a result that limited to the discrete set of initial labels L. Moreover, this inference method can easily work with either numerical or categorical labels. Lastly, the performance of our inference method is evaluated using the RMSE and MAE between the predicted and the true labels of each point.

```
def predict_indirect_naive(D, B, R, M):
"""
D: input multivariate time series.
B: set of indexes that index tha separate bins D is split into.
R: set of indexes that index labeled regions
M: set of trained models and metadata that is loaded
"""
fit_features = M{'fit_features'}
target_feature = M{'target_feature'}
labels = M{'labels'}
models = M{'models'}
evaluation_scores = {}
for r_index in R:
    current_data = D[r_index]
    for l in labels:
        models = models[l]
        bin_scores = []
        for bin_no, bin_index in B:
            current_data = D[bin_index]
            m = models[bin_no]
            predictions = m.predict(current_data, fit_features,
target_feature)
            score = score(predictions, current_data[target_feature])
            bin_scores.append(score)
        evaluation_scores[r_index][l] = mean(bin_scores)

for r_index in R:
    selected_label = argmin(evaluation_scores[r_index])
    assign_label(D[r_index], selected_label)
```

Algorithm 6: Pseudocode for the naïve inference method

5.1.3.3. Indirect inference and evaluation: frequency-based aggregation

We aim to tackle the inference part of our indirect method with another more sophisticated method that can take into account more than one label for each bin of the dataset. For each point in bin b_j of the region W in question, we make an approximation using all the available trained models $m(l_i, b_j)$ and we assign to it the label of the model that gives the closest approximation for that point. This way, we get frequencies f_{l_i} of how many points were labeled as l_i in the bucket b_j of region W. To obtain the label of the bucket b_i we take a weighted average of the labels with their frequencies as weights:

$$label(b_j) = \frac{\sum_{i \le v \le c} f_{l_i} \cdot l_i}{\sum_{i \le v \le c} f_{l_i}}$$

and then to get the label that characterizes W:

$$label(W) = \frac{1}{k} \sum_{j=1}^{k} label(b_j)$$

It should also be mentioned that the number of the most frequent labels in each bin can be defined by the user to better fit the problem at hand. Besides, if the number of buckets $k = 1$ then the approach is equivalent to no splitting of the data if a user decides to do so. As with all other methods MAE and RMSE are utilized to further evaluate the performance of the model at test-time.

```
def predict_indirect_freq(D, B, R, M):
"""
D: input multivariate time series.
B: set of indexes that index tha separate bins D is split into.
R: set of indexes that index labeled regions
M: set of trained models and metadata that is loaded
"""
fit_features = M{'fit_features'}
target_feature = M{'target_feature'}
labels = M{'labels'}
models = M{'models'}
evaluation_scores = {}
for r_index in R:
    current_data = D[r_index]
    bin_labels = []
    for bin_no, bin_index in B:
        current_data = D[bin_index]
        closest_diff = [inf] * len(current_data)
        closest_label = [nan] * len(current_data)
        for l in labels:
            models = models[l]
            m = models[bin_no]
            predictions = m.predict(current_data, fit_features,
target_feature)
            diffs = abs(predictions-current_data[target_feature])
            for i in range(len(diffs))
                if diffs[i] < closest_diff[i]:
                    closest_diff[i] = diffs[i]
                    closest_label[i] = l
        freqs = {}
        for l in labels:
            for i in range(len(current_label))
                freqs[l] = count(current_labels, l)
        weighted_avg = average(labels, weights=freqs) # can be modified
to only use top v freqs
        bin_labels.append(weighted_avg)
    region_label = mean(bin_labels)
    assign_label(D[r_index], region_label)
```

Algorithm 7: Pseudocode for the frequency-based aggregation inference method

5.1.3.4. Indirect inference and evaluation: measure-based aggregation

Another method that has been developed to detect and label specific behaviors of time series using our indirect models is a similarity/dissimilarity measure-based method (we will continue with similarity in this section for simplicity). In a way that resembles the previous method 5.1.3.3, we use each model $m(l_i, b_j)$ to make predictions in each bin b_j of the region of interest W. We then calculate a goodness-of-fit score for each one of those models i.e., the coefficient of determination R^2. Next, we normalize those scores to sum to 1 utilizing the $l1$-norm and we use those as the weights for each bin as follows:

$$label(b_j) = \frac{\sum_{i \le v \le c} s_{m(l_i,b_j)} \cdot l_i}{\sum_{i \le v \le c} s_{m(l_i,b_j)}}$$

where $s_{m(l_i,b_j)}$, is the normalized goodness-of-fit score for the respective model. To get the final label for W we again perform a mean operation over the labels of all the bins:

$$label(W) = \frac{1}{k}\sum_{j=1}^{k} label(b_j)$$

Moreover, this method can be easily parametrized to use a variety of different similarity or dissimilarity measures that score how well a regression model's estimations fit the target variable.

```
def predict_indirect_measure(D, B, R, M, measure):
"""
D: input multivariate time series.
B: set of indexes that index tha separate bins D is split into.
R: set of indexes that index labeled regions
M: set of trained models and metadata that is loaded
mreasure: goodnes-of-fit measure.
"""
fit_features = M{'fit_features'}
target_feature = M{'target_feature'}
labels = M{'labels'}
models = M{'models'}
evaluation_scores = {}
for r_index in R:
    current_data = D[r_index]
    bin_labels = []
    for bin_no, bin_index in B:
        current_data = D[bin_index]
        measure_scores = {}
        for l in labels:
            models = models[l]
            m = models[bin_no]
            predictions = m.predict(current_data, fit_features,
target_feature)
            measure_scores[l] = score(predictions,
current_data[target_feature], measure)
        norm_scores = normalize_L1(measure_scores)
        weighted_avg = average(labels, weights=norm_scores) # can be
modified to only use top v scores
        bin_labels.append(weighted_avg)
    region_label = mean(bin_labels)
    assign_label(D[r_index], region_label)
```

Algorithm 8: Pseudocode for the measure -based aggregation inference method

5.2. Implementation Information

The behavior detection tools are implemented in Python, currently as a Jupyter notebook. For efficiently managing, preprocessing, and performing numerical operations on the data, we use pandas[3], scikit-learn[4], numpy[5]. The notebooks can be found in https://github.com/MORE-EU/complex-event-detection/tree/main/notebooks/tools. In particular, the notebooks specialize our Training methods on the Yaw misalignment use case. In the next lines we are going to explicitly explain our functions and parameters we use in our methods.

For the matter of efficiency, we will discuss about our Direct model, which we train and evaluate, since the same functions are also utilized in the Indirect model.

Apart from methods developed in D4.1, new functions have been defined aiming to support the new functionalities introduced in this deliverable, and the corresponding experimental evaluation. A list of the new functions follows:

- **outliers_IQR:** The interquartile range identifies outliers by creating a net. Any values that fall outside of this net are considered outliers.
- **split_to_bins**: Divides our dataset into subsets which a characteristic that a user is seeking. Returns True/False if the characteristic exists in each subset.
- **lasso_selection**: Returns the most important features from our given dataset. We are using the Lasso linear model and select the best model over a cross validation estimator, the best variables are selected from the importance of the models' weights.
- **perform_grid_search**: Returns the best parameters of the estimator for a user input model from a list of user defined parameters space by a cross validation search.
- **fit_pipeline:** Returns the model, the predicted target value and metric scores for a user defined pipeline model with the results of the perform_grid_search algorithm

Our main technical contribution is to provide an implementation of our method, in the form of a Jupyter notebook. The notebook implements all different variants of the Training methods described in Section 5.1. In particular, the preprocessing step can be found in the cell [3]. Then, we use a scaling algorithm in order to fix the different values of the dataset in cell [4]. In Cell [6] we are splitting our dataset into wind bins; this step is not mandatory but rather a user defined step. Cell [7] implements the feature selection algorithm, which is fully parametrizable and allows to the user to use the features she wishes. In cell [8] the user can define the regression methods as well as the parameter search space. In the direct model the Default is the Random Forest algorithm and the Indirect model uses the Ridge regression model. Both regressors and parameters are easily defined by our users. Lastly in cell [9] the training stage of our model is taking place.

5.3. User guide

5.3.1. 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/complex-event-detection --recursive

One new folder under the name complex-event-detection will be created. Let <path> be the absolute path to that folder, i.e., the new folder is <path>/ complex-event-detection.

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 - 1.3.3
- matplotlib - 3.4.2
- numpy - 1.20.1
- scikit_learn - 1.0.2

The above libraries are included in the requirements.txt file. 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 notebooks:

The files are located at:

<path>/ complex-event-detection/notebooks/tools/

- **Direct_model_training.ipynb**
- **Direct_model_Evaluation_method.ipynb**
- **Indirect_model_training.ipynb.ipynb**
- **Indirect_model_Evaluation_Naive.ipynb**

5.3.2. Configuration options

In this section we list a set of parameters which allow the user to test different configurations that correspond to different variants of our methods. Our methods follow the same steps. An explanation for which method we are using is stated when needed. We also need to state that those options apply for the training procedure. We prompt the user to keep the same options on the evaluation procedure.

- **path**: The path of the dataset to be read. It can be found in cell [2].
- **granularity**: Frequency (most of the times time) which the user can resample the time series. It can be found in cell [3].
- **col**: The desired column to filter in our time series. It can be found in cell [3].
- **less_than**: Filtering the column dropping values below that price. It can be found in cell [3].
- **more_than**: Filtering the column dropping values above that price. It can be found in the cell [3].
- **start**: Date that signifies the start of the interval. It can be found in cell [3].
- **end**: Date that signifies the end of the interval. It can be found in cell [3].
- **bin_size**: Size of the bins the user prefers to split the dataset. It can be found in cell [6].
- **min_speed**: Minimum value of our bin can reach. It can be found in cell [6].
- **max_speed**: Maximum value of our bin can reach. It can be found in cell [6].
- **bin_feature**: A variable name that the user wishes to apply the split_to_bins. It can be found in cell [6].

- **features**: Feature variables to be used in lasso_selection. It can be found in cell [7].
- **target**: Target variable to be used for lasso_selection. It can be found in cell [7].
- **alphas**: List of alphas where to compute the models in lasso_selection. The Default value is None. It can be found in cell [7].
- **model**: User Defined Regressor that will be used in the Pipeline. In the Direct model the Default is Random Forest Regressor. In the Indirect model the Default is Ridge with Polynomial Features. It can be found in cell [8].
- **pipeline**: Takes as an input the previous model parameters. It can be found in cell [8].
- **parameters**: User parametrized list of parameters in order to be used in the training of the model. It can be found in cell [8].
- **feats, target, scorer, model, params**: Features to be fitted, target value, score function, model already defined, the parameters from above, used in perform_grid_search algorithm correspondingly. It can be found in cell [9] For the direct model, in cell [10] for the Indirect method.
- **feats, target, pipeline, params**: Features to be fitted, target value, score function, model already defined, the hyper parameters which were returned from perform_grid_search algorithm to be used in fit_pipeline algorithm. It can be found in cell [9] For the direct model, in cell [10] for the Indirect method.

5.3.3. Deployment instructions and options

In this section we present deployment instructions for the training procedure of the Direct and the Indirect model as well as instructions for a Naive evaluation method. The user can find all the variants of our Evaluation methods at https://github.com/MORE-EU/complex-event-detection/tree/main/notebooks/yaw_misalignment_experiments.

Direct model: The input time series is provided in the form of a .parquet file. This is defined in cell [2]. Moreover, in cell [3], the user can define the change of granularity, and thresholds for our preprocessing filtering steps. In cell [4] the user can observe the ground truth labels over a time interval for our turbines as well as the smoothed Dynamic Yaw misalignment. In the next cell [5], we scale our data frames and save them to a .pickle file for later usage. Then we split our data frames in wind speed bins in a user-friendly procedure, in cell [6] (this step is optional) and in cell [7] we perform a feature selection by using the Lasso algorithm, which can be also parametrized by the user, who can define manually the features she needs to use without the Lasso algorithm. For the two last steps, in cell [8], the user can define the desired regression methods and set a relevant parameter search space she wishes. Finally, in cell [9] the training takes place, in which for each turbine we apply the wind bins and for each bin we split our dataset into a training set and a test set, and we perform a grid search in order to calculate the hyper parameters of our models with a user defined choice of grid search algorithm, and we fit our model and make predictions in order to calculate the meanAPE. We save our models and our selected features in a .pickle file and we print run times, train and test errors.

Evaluation method for Direct model: The input time series is provided in the form of a .parquet file. This is defined in cell [2]. Moreover, in cell [3], the user can define the change of granularity, thresholds for filtering, although we recommend to evaluate the new dataset in the same settings as of the training. In cell [4] the user can observe the ground truth labels over a time interval for our turbines as well as the smoothed Dynamic Yaw misalignment. Then we split our data frames in wind speed bins in a user-friendly procedure cell [6], a step which is not necessary unless it took place in the training procedure, and cell [7] we estimate the yaw angle using per-point predictions of the trained model. In the next step we plot the results of our model on the new dataset [8]. In [9] we present the previous plots in a user defined smoothing average of two days. In [10] we illustrate a boxplot of predicted yaw angle over ground truth angles for all the different wind bins and in [11] the analogous boxplots for

the median values of those angles. In the last step, in cell [12], for the medians of the yaw angles we plot again the static yaw misalignment.

Indirect model: The input time series is provided in the form of a .parquet file. This is defined in cell [2]. Moreover, in cell [3], the user can define the change of granularity, and thresholds for our preprocessing filtering steps. In the next cell [4], we scale our data frames and save them to a .pickle file for later usage. In the cell [5], the user can observe the ground truth labels over a time interval for our turbines as well as the smoothed Dynamic Yaw misalignment. Then we split our data frames in wind speed bins in a user-friendly procedure cell [6] (this is an optional step) and in [7] we perform a feature selection by using the Lasso algorithm, this is a user defined procedure also, where the user is also allowed to define manually the features she needs to use. For the two last steps, in cell [8], the user can define the desired regression methods and set a relevant parameter search space she wishes and in [9], the training takes place, in which for each turbine we apply the wind bins. Then for each static misalignment value we train our active power model. For each bin we split our dataset into a training set and a test set, and we perform a grid search in order to calculate the hyper parameters of our models with a user defined choice of grid search algorithm. Finally, we fit our model and make predictions in order to calculate the meanAPE. We save our models and our selected features in a .pickle file and we print run times, train, and test errors. In cell [10], we include box plots of the power in comparison with the Yaw misalignment angle for each user defined wind speed bin from the previous step [6].

Naive Evaluation method for Indirect model: The input time series is provided in the form of a .parquet file. This is defined in cell [2]. Moreover, in cell [3], the user can define the change of granularity, and thresholds for our preprocessing filtering steps. We recommend evaluating the new dataset in the same settings as of the training. In cell [4], the user can observe the ground truth labels over a time interval for our turbines as well as the smoothed Dynamic Yaw misalignment. Then, we split our data frames in wind speed bins in a user-friendly procedure cell [5]. This step is not necessary if it didn't take place in the training procedure. In cell [6], we estimate the yaw angle using the most fitting pre-trained model label. In the next step we plot the results of our model on the new dataset [7] and we illustrate some boxplots of power over those angles for all the different wind bins [8]. For the two last steps, in cells [8][9] the user can define the desired smoothing she wants to predict in sliding windows. In our example, we use one day and ten days windows correspondingly.

5.4. Evaluation

In the frame of the Yaw Misalignment use case, we have been provided with several multidimensional time series measuring a set of 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 Table 3: Labels denoting changes in the yaw misalignment angle, in three turbinesTable 3. The given static misalignment angles were measured using LiDAR. We also note that during our experimentation only the absolute value of the static yaw misalignment angles is considered, since the degradation of the turbine's power production is mainly correlated to the magnitude of the misalignment and not the sign.

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

BEBEZ01	-6,7°	+4,8°	+9.4°	+/- 0°	-5.8°	+/- 0°
BEBEZ02	+2,7°	+2,7°	+4.9°	+/- 0°	-5.1°	+/- 0°
BEBEZ03	-2,2°	+3,4°	+7.5°	+/- 0°	-5.5°	+/- 0°

Table 3: Labels denoting changes in the yaw misalignment angle, in three turbines

5.4.1. Preprocessing

Providing a dataset that contains as little outliers, errors, and noise as possible is very important for any data-driven machine learning method. For this reason, before training our models we pass the dataset through a preprocessing pipeline. We keep values of the time series for wind speed, pitch angle, rotor speed, active power, nacelle direction, and wind direction. The input time series have a time granularity of 2 seconds. To decrease the effect of instantaneous instabilities and noise, we resample the input time series to a granularity of 1 min. Then, we calculate a new variable, the dynamic yaw misalignment, using the nacelle direction and wind direction variables. Afterwards, we apply a 60-minutes rolling mean on the dynamic yaw misalignment variable to smoothen out any extreme present. We remove outliers by means of the interquartile range (IQR) method, and by exploiting domain knowledge. More specifically, we apply a series of filtering operations, we keep values of wind speed between 5-11m/s because in that range there is no curtailment effects on the power curve. In addition, we keep datapoints with pitch angle between 2 and -2 degrees; this is done to remove points that are affected by the pitch misalignment phenomenon, thus keeping points where yaw misalignment is the main contributing factor to any inefficiencies. Finally, we keep values of rotor speed between 8 and 15 rpm and remove datapoints with active power very close to or less than zero. We also keep only the part of the time series which is labelled and a small portion of 1 month before the first and after the last LiDAR measurement. We should also note that for some of the experiments only a subset of the previously mentioned variables is finally utilized, as a result of a Lasso-based feature selection method (see Section 5.1.1.2).

5.4.2. Preliminary Analysis

We explored the previously mentioned wind park datasets in-depth, to gain a better understanding and insight on how exactly the power output behaves at different measured wind speeds and static yaw misalignment angles. Indicatively, in Figure 9 we can see a collection of boxplots that show the observed active power of the wind turbines at different static yaw misalignment values. The boxplots are created for several wind speed ranges to get an overview of whether and how this behavior is different at different wind speeds. The data used for that figure were collected from two turbines BEBEZE01 and BEBEZE03 of the ones provided to us by Engie; BEBEZE02 also shows a behavior that heavily resembles those.

An important insight that we obtain by examining Figure 9 is that there is a quite significant drop in the measured active power variable when the yaw misalignment deviates from 0° even when the misalignment is quite low i.e., approx. 2°. Then as the absolute value of the misalignment angle increases, we observe a steady negative trend that is not as steep as when the turbine first leaves its aligned state. This suggests that there is a notable decline in turbine power efficiency even for low (absolute) static yaw misalignment values and thus, to get the most power out of the wind turbine, corrections would be desirable as soon as the turbine leaves the “aligned” state of misalignment angles near 0°.

Moreover, in the bins that represent data from higher wind speed ranges, we observe a higher in-bucket variance regarding the yaw-misalignment to active power relationship, where points that belong to a higher misalignment angle show a higher median active power. However, this variance could also be partially attributed to the lower number of available datapoints at those bins after the

various filters and outlier removal methods we apply. Nevertheless, it serves as an indication for the need to further examine the individual behaviors of wind turbines in different wind buckets, in relation to the yaw misalignment task.

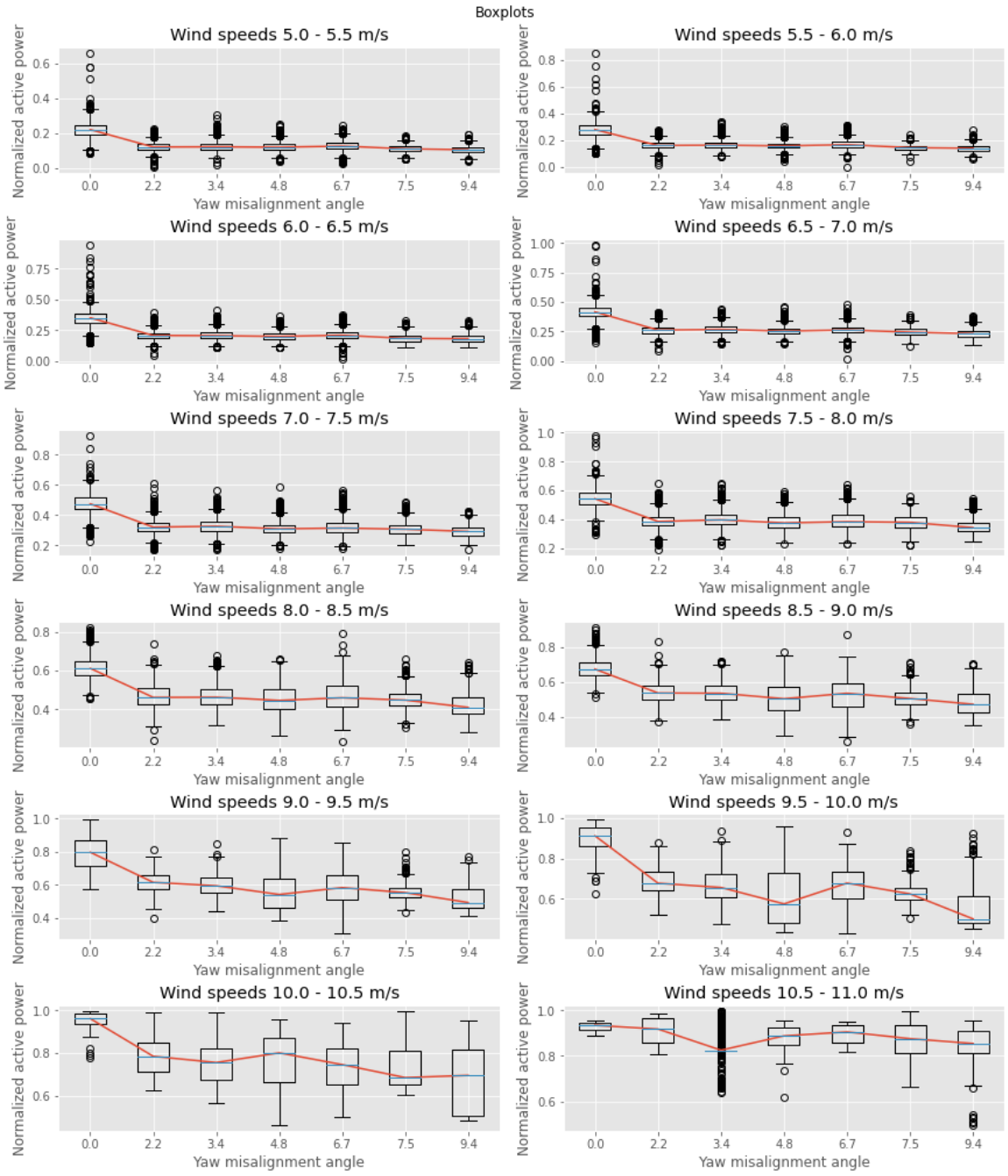

Figure 9: Collection of boxplots which compare the (absolute) yaw misalignment angle to the normalized active power for several wind speed bins (Turbines: BEBEZE01, BEBEZE03).

5.4.3. Evaluation of effectiveness

To evaluate the effectiveness of our methods we conducted a rigorous set of experiments on real-world, labelled wind turbine datasets (Table 3: Labels denoting changes in the yaw misalignment angle, in three turbinesTable 3). In our experiments we compare the predicted static yaw

misalignment obtained by our models to the ground truth given to us by the labels. We consider a wide variety of training as well as prediction and evaluation settings. For the evaluation of the effectiveness, we utilize standard error measures such as RMSE, MAE and MAE and we also provide visual comparisons for a more intuitive insight.

5.4.3.1. Effectiveness of Direct Methods

We begin by analyzing the effectiveness of our direct methods, i.e. methods that rely on directly working with the yaw misalignment measurements as the dependent variable of our regression models (described in Section 5.1.2). We examine different training settings regarding the binning of the dataset, and we also work with two different combinations of turbines used for our training dataset. One training variant splits the dataset into bins based on wind speed (1 m/s intervals) and the other performs no binning operations. This aims to identify the effects of handling the dataset with or without binning, on the effectiveness of the algorithms. The following combinations of training/evaluation are examined: 1) train on BEBEZE01 and BEBEZE02, then evaluate on BEBEZE03 and 2) train on BEBEZE01 and BEBEZE03, then evaluate on BEBEZE02. This is done to further verify the generalization ability of our models.

During the training process we utilize a train-test (80%-20%) split to validate the correctness of each model and to also identify any possible overfitting issues and the parameters of each model are selected through grid search. In every setting that we described in the previous paragraph, the measured MAPE in the training sets ranged between 3-7% and between 7-14% on the validation sets. Those values indicate that our models can learn the relationships present in the data and that they do not overfit, since a small increase of MAPE is expected when the model works in a validation set. Moving on to the evaluation on new turbines, in Figure 10 and Figure 11, we present the results of our model when predicting the static yaw misalignment on a new turbine that was not considered by the model during training. In Figure 10, we can visually observe the performance of our direct model when the binning technique is used. The RMSE and MAE is also calculated in each case, and we report a RMSE = 1.55 and MAE = 1.28 when evaluating on BEBEZE03 (left image in Figure 10) and RMSE = 1.55 and MAE = 1.40 when evaluating on BEBEZE02 (right image in Figure 10).

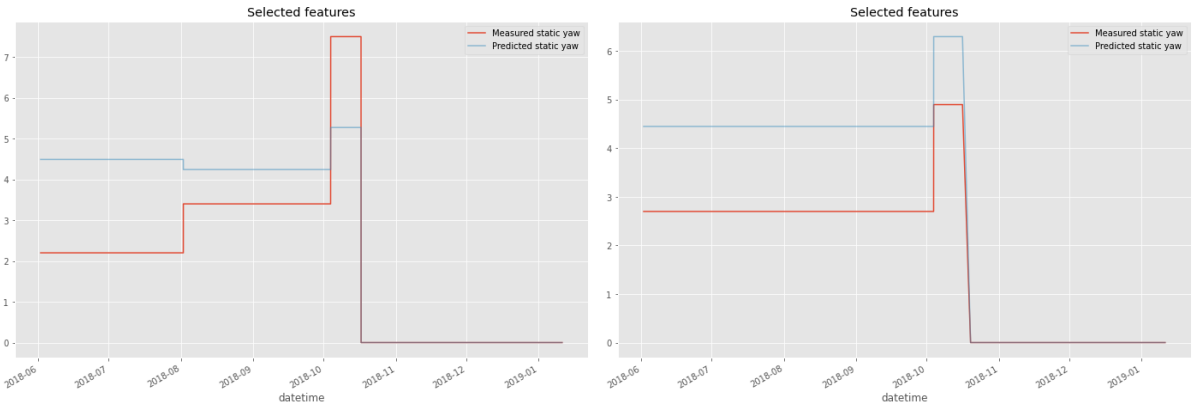

Figure 10: Predicted vs Actual yaw misalignment on labelled regions. Predictions are made on BEBEZE03 (left) and BEBEZE02 (right), the dataset was split into bins (1m/s wind speed intervals)

We conducted the same experiments without binning our datasets and the results are showcased in Figure 11. The resulting plots are very similar to the previous case and the reported RMSE and MAE are 1.55 and 1.26 when evaluating on BEBEZE03 (left image in Figure 11) and 1.54 and 1.41 when evaluating on BEBEZE02 (right image in Figure 11) respectively.

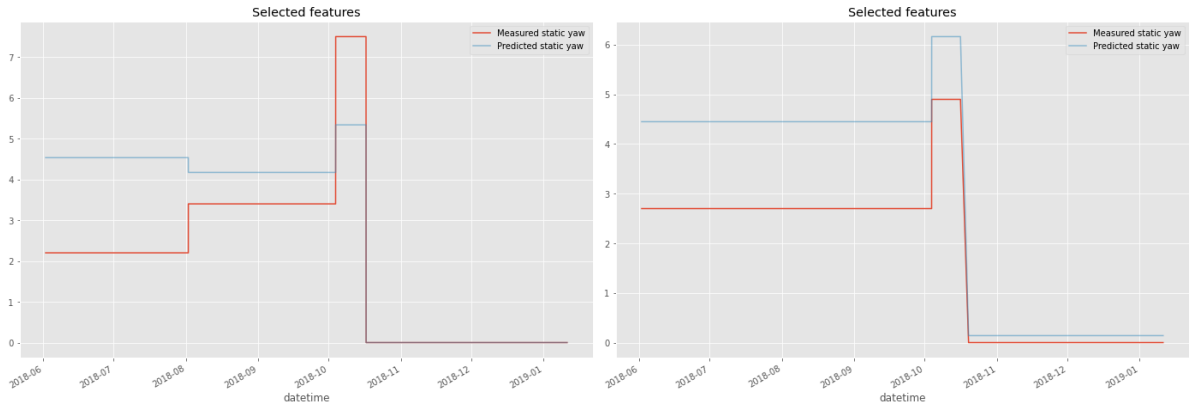

Figure 11: Predicted vs Actual yaw misalignment on labelled regions. Predictions are made on BEBEZE03 (left) and BEBEZE02 (right), the datasets were not split into bins

From the figures and the numerical results, we presented above, we can deduce that our direct models make accurate predictions that deviate approximately 1.5° on average. Furthermore, our methods are very accurate on predicting the 0° angle and distinguish between that and other misalignment angles. This is important because the largest drop-off in active power in our data is observed when the turbine deviates from the aligned state of 0° and that deviation can be accurately detected for correction. The predictions for other misalignment values (2.2° to 7.5°) are also quite reasonable with the predicted values closely following the actual ones in each region with slight errors. Also, the binning of the dataset in the direct case does not seem to increase the performance of the models, since very minor differences were observed between the results when working with binned and not binned data. Lastly, all the experiments are available as Jupyter notebooks at https://github.com/MORE-EU/complex-event-detection/tree/main/notebooks/yaw_misalignment_experiments/direct_methods.

5.4.3.2. Effectiveness of Indirect Methods

In this subsection, we provide a thorough analysis of the effectiveness of our indirect methods. In greater detail, these methods utilize algorithms that model the active power of a turbine during periods of different static yaw misalignment and then infer the misalignment depending on which specific model approximates the active power more accurately in new data. Once more, we examine a wide variety of different training settings regarding the usage of different sets of features as our inputs among others. More specifically, we examine: 1) using only the wind speed as a feature, 2) using only the features that are selected by the Lasso method, and 3) using all the available features. For all experimental settings, we also consider two cases, binning the data into bins based on wind speed or keeping them as they are. As in the previous section, the following combinations of training/evaluation are examined: 1) train on BEBEZE01 and BEBEZE02, then evaluate on BEBEZE03 and 2) train on BEBEZE01 and BEBEZE03, then evaluate on BEBEZE02. This is done to further verify the generalization ability of our models.

During the training process we utilize a train-test (80%-20%) split to validate the correctness of each model and to also identify any possible overfitting issues. Also, every model is tuned using grid search. During training we measure the MAPE of each model in the training and then in the test set and we get: 1) 9%-13% MAPE in the training and 8%-13% in the test set when using only wind speed to train and split into wind bins. Without the wind bins method, we score 9%-14% MAPE in the training set and 8%-14% in the test set, 2) 2%-4% MAPE in the training and 2%-5% in the test set when using only the selected features and split into wind bins. Without the wind bins method, we score 2%-6% MAPE in the training set and 2%-6% in the test set, and 3) 2%-4% MAPE in the training and 2%-5% in the test set when using all the available features and split into wind bins. Without the wind bins method, we score 2%-6% MAPE in the training set and 2%-6% in the test set. As it is clearly seen from observing the previous MAPEs, there is a significant reduction in error when using the selected or all the features

compared to only the wind speed. In the following paragraphs we will present figures that illustrate the predicted yaw misalignment of our indirect models and compare them to the actual values. Those figures will also be accompanied by numerical error measurements such as RMSE and MAE. Lastly, all the experiments are available as Jupyter notebooks at https://github.com/MORE-EU/complex-event-detection/tree/main/notebooks/yaw_misalignment_experiments/indirect_methods.

We initiate the experimental evaluation of our indirect methods by focusing on the frequency-based aggregation method where all feature variables are being used. Figure 12 illustrates a comparison between the actual static yaw misalignment angle and the corresponding prediction, using the top 2 most frequent labels in each bin. The reported RMSE and MAE are 1.49 and 1.42 when evaluating on BEBEZE03 (left image in Figure 12) and 3.19 and 3.02 when evaluating on BEBEZE02 (right image in Figure 12) respectively. In Figure 13 we make the same comparison, in the case that the top one most frequent label from each bin is being used. The reported RMSE and MAE are 1.18 and 0.75 when evaluating on BEBEZE03 (left image in Figure 13) and 2.76 and 2.49 when evaluating on BEBEZE02 (right image in Figure 13) respectively. In Figure 14 all bins are being used and the reported RMSE and MAE are 1.88 and 1.84 when evaluating on BEBEZE03 and 2.62 and 2.68 when evaluating on BEBEZE02.

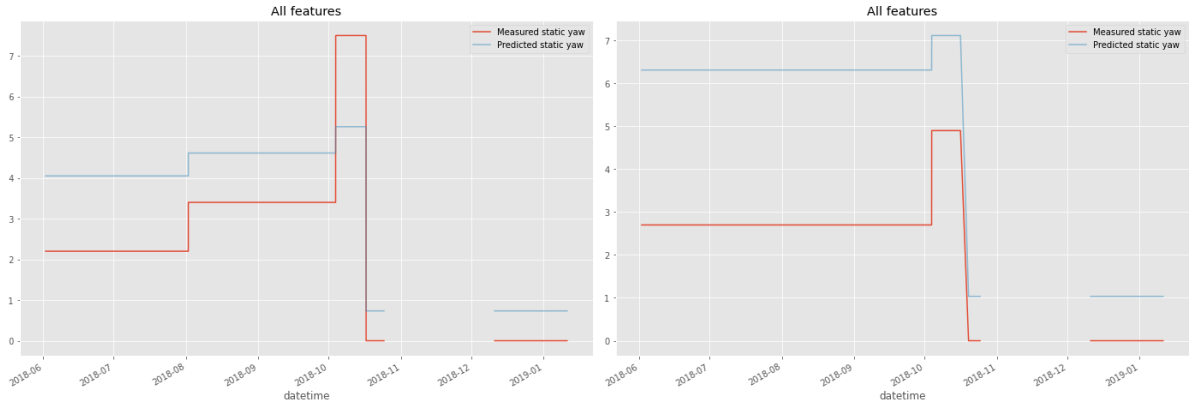

Figure 12 Predictions are made on BEBEZE03 (left) and BEBEZE02 (right). Frequency-based aggregation using top 2 frequent labels per bin. All features are used.

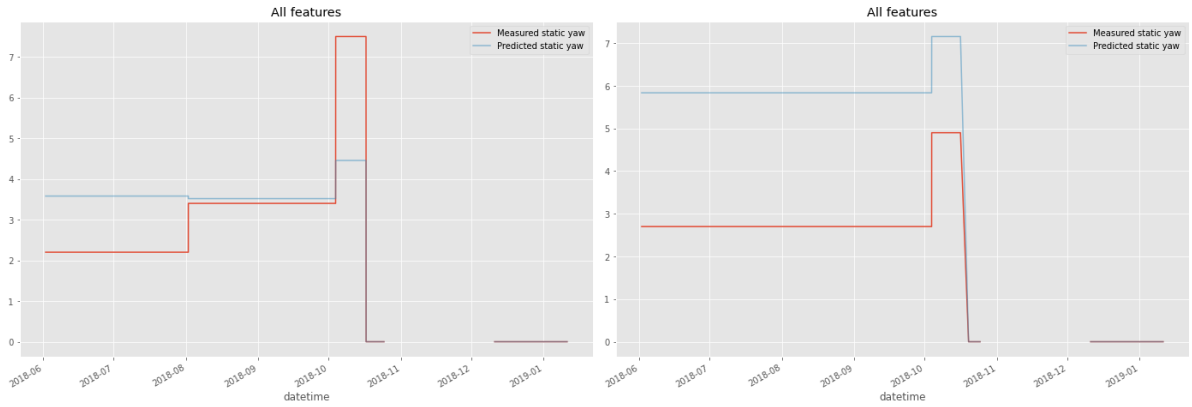

Figure 13 Predictions are made on BEBEZE03 (left) and BEBEZE02 (right). Frequency-based aggregation using top 1 frequent label per bin. All features are used.

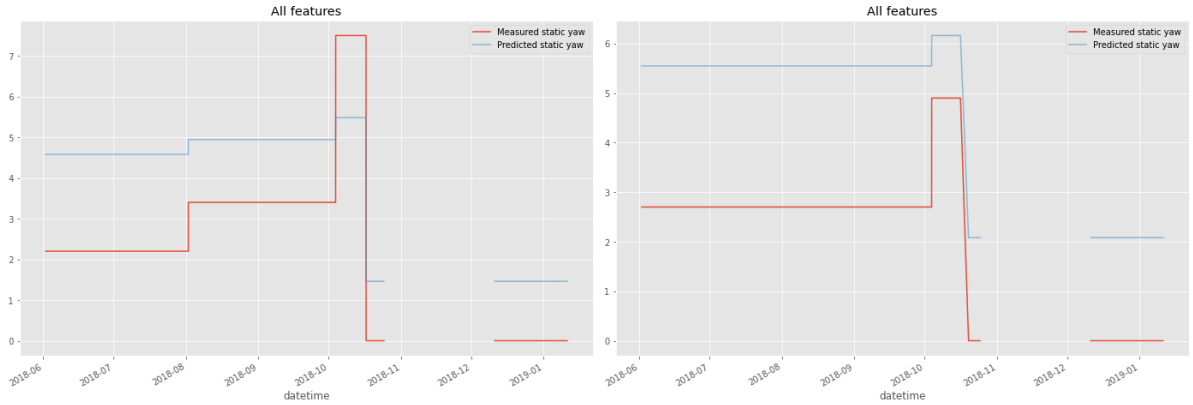

Figure 14 Predictions are made on BEBEZE03 (left) and BEBEZE02 (right). Frequency-based aggregation using all labels per bin. All features are used.

In the second set of experiments, we investigate the case where feature variables are selected as described in Section 5.1.1.2. In Figure 15, we compare the actual static yaw misalignment angle and the corresponding prediction, using the top 2 most frequent labels in each bin. The reported RMSE and MAE are 1.49 and 1.42 when evaluating on BEBEZE03 and 2.91 and 2.76 when evaluating on BEBEZE02 respectively. In Figure 16 we make the same comparison, in the case that the top one most frequent label from each bin is being used. The reported RMSE and MAE are 1.77 and 1.55 when evaluating on BEBEZE03 and 3.11 and 2.80 when evaluating on BEBEZE02 respectively. In Figure 17 all bins are being used and the reported RMSE and MAE are 1.95 and 1.90 when evaluating on BEBEZE03 and 2.63 and 2.55 when evaluating on BEBEZE02.

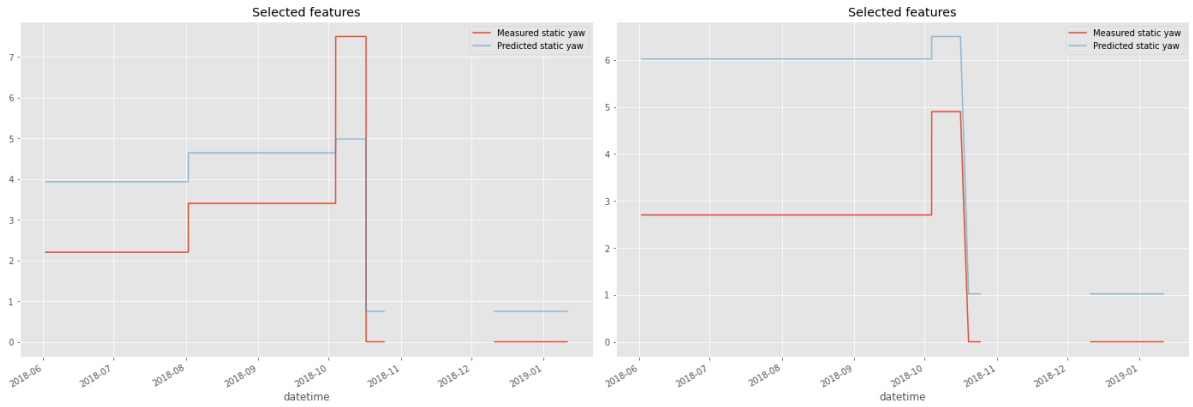

Figure 15 Predictions are made on BEBEZE03 (left) and BEBEZE02 (right). Frequency-based aggregation using top 2 frequent labels per bin. Selected features are used.

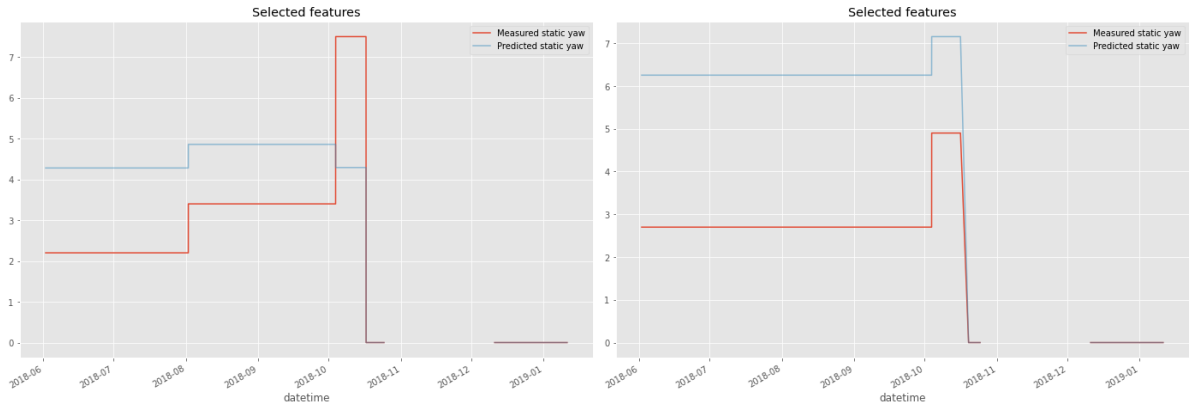

Figure 16 Predictions are made on BEBEZE03 (left) and BEBEZE02 (right). Frequency-based aggregation using top 1 frequent label per bin. Selected features are used.

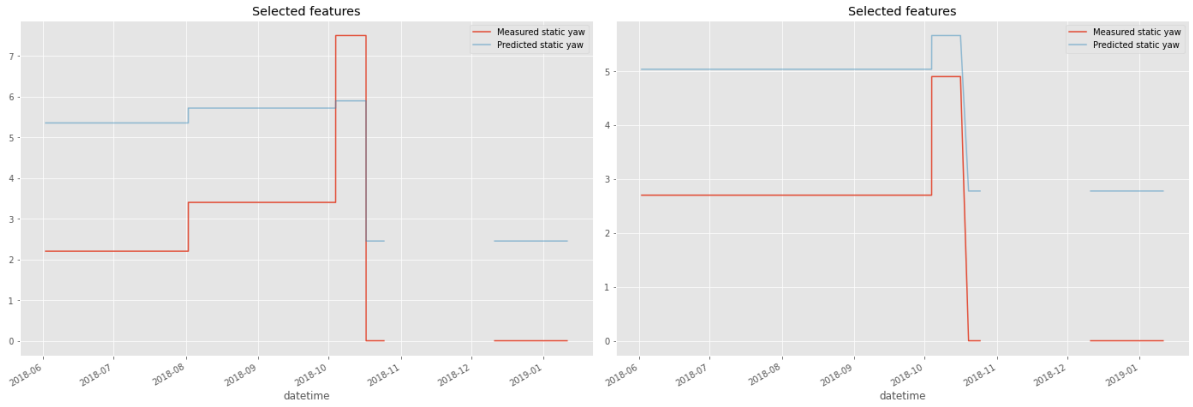

Figure 17 Predictions are made on BEBEZE03 (left) and BEBEZE02 (right). Frequency-based aggregation using all labels per bin. Selected features are used.

We observe similar behaviors in the plots of this section to the plots of Section 5.4.3.1. Once again, our methods are very accurate on predicting the 0° angle and distinguish between that and other misalignment angles. The predictions for other misalignment values are also reasonably accurate in the majority of the cases. Of course, it becomes evident even from the visual inspection of the figures that Direct methods are slightly more effective in identifying the misalignment angle. Nevertheless, we believe that both methods can have their merits and further experimentation both in the same setting (by including more independent variables and more labelled turbines upon provision) and in the similar setting of pitch misalignment are part of our next steps in Task 4.2.

5.4.4. Evaluation of scalability

To evaluate our methods in a real-time scenario, we proceed as in Section 4.4.3, by simulating the real-time aspect. Given a large data stream in the form of a static time series, we process it in batches, where each batch of k points represents the last k points seen so far. To claim scalability of, we adapt our methods to handle a scenario where multiple streams must be processed in parallel.

Given a segment of input time series, we normalize it so that it has a granularity of 1 min, and we add Gaussian noise to each point, and we compute its absolute value. Gaussian noise is defined by random variables following the normal distribution with mean 0 and variance $\varepsilon \cdot \sigma^2$, where $\varepsilon > 0$ is a user defined parameter, and σ^2 is the variance of the variable/column in consideration.

Input time series are traversed in parallel; at each given instant we assume that a new batch of data points is revealed, in each one of the input time series. Data points contain measurements of wind speed, pitch angle, rotor speed, active power, nacelle direction, and wind direction. Our task is to predict the yaw misalignment angle corresponding to all new batches of points, as this is the most basic operation needed for real-time detection. To simulate parallel computations by different nodes, we employ multi-core parallel processing using Dask. In particular, we use as input 20000 synthetic time series, each one representing a stream transmitted from a different source. The time granularity is 1 min, and each time series spans 20 days. We assume that a decision must be taken every 1440 points, i.e., every 1 day. For each day we make predictions using a regression model in 20000 windows of length 1440. This is implemented as a parallel computation by assigning batches of 334 windows to 30 cores (2 batches per core), with maximum processor frequency at 3.7GHz, and available RAM at 256Gb. The average running time taken over all days, for the standard sequential computation is 753.59 seconds, and the average running time of our parallel implementation is 67.67 seconds. The Jupyter notebook implementing this experiment is available at https://github.com/MORE-EU/complex-event-detection/tree/main/notebooks/parallelization_experiments.

6. Complex events: motif discovery

In this section, we briefly report on our ongoing work on motif-based pattern extraction and detection. This currently comprises two strands of work: Matrix profile Computation on summarized data and Annotation Vector and motif-based detection of soiling events, as presented next.

6.1. Computing Matrix Profile using Segment View

Next, we present our initial exploration on how to compute Matrix Profile (MP) using segment view, i.e., data models provided by ModelarDB. We first briefly describe how the original Matrix Profile works, and then discuss how we can intervene in that process to construct MP directly using data models.

6.1.1. Matrix Profile

Matrix Profile is a data structure designed to enable other algorithms built on top of it to perform different data mining tasks such as anomaly detection and motif discovery. The MP data structure has two main components: a distance profile and a profile index. The distance profile stores the minimum Euclidean distance between a query and a time series, and the profile index stores the index of the first nearest-neighbor, i.e., the location of the most similar sub-sequence in the time series to the query.

Generally, to compute Matrix Profile, a sliding window approach is used. Consider a time series **X** of length n, and a query **q** of length m. The MP computation algorithm on **X** and **q** works as follows:

- A window that has the same length as the query **q** is slided over the time series **X**, where the Euclidean distance between the windowed sub-sequence and the query is computed. The distance calculations occur (n - m + 1) times for each query **q**.
- The distance profile is updated with the minimal values. Note that in this step, an exclusion zone to prevent trivial matches is set; this is usually set as the half of the window size m both before and after the current window index. The values in the exclusion zone are ignored when computing the minimum distance and the nearest-neighbor index.
- The final step is to compute the profile index by identifying and updating the index of the first nearest-neighbor of the query.

Finding Motifs and Discords using Matrix Profile: A motif is a repeated pattern in a time series, and a discord is an anomaly. Using the computed Matrix Profile data structure, it is straightforward to find the top-K motifs or discords. As Matrix Profile stores the distances in Euclidean space, it means that a sub-sequence that has distance close to 0 to the query is the most similar, comprising a motif. In contrast, a sub-sequence with a distance much greater than 0 is the most dissimilar sub-sequence w.r.t the query, i.e., the discord. Therefore, extracting the smallest distances from distance profile gives the motifs, and the largest distances from distance profile gives the discords.

6.1.2. Euclidean Distance using Segment View

6.1.2.1. Representing time series as polynomial function

Let f(t) and g(t) be the polynomial functions of degree 0, 1, or 2. We have:

$$f(t)= at^2+bt+c$$

$$g(t)= mt^2+nt+p$$

Depending on the degree of f(t) and g(t), we can set the corresponding coefficients to 0. For example, if f(t) has degree 1, we set a=0. If g(t) has degree 0, i.e., constant function, we set m=n=0.

Without loss of generality, consider two time series represented by f(t) and g(t):

$X = \{x_1, x_2, ..., x_k\} = \{f(t), t=1, 2, 3, ..., k\}$

$Y = \{y_1, y_2, ..., y_k\} = \{g(t), t= 1, 2, 3, ..., k\}$

Note that any function $\{\phi(t), t=i, i+1, ...,i+k\}$ can always be represented by an equivalent function {h(t), t = 1, 2,..., k} by using the transformation $h(t) = \phi(t+i)$.

6.1.2.2. Computing Euclidean distance between two time series using approximated functions

Given the two time series **X** and **Y** represented by f(t) and g(t), the Euclidean distance between **X** and **Y** is computed as

$$\begin{aligned} D(X,Y) &= \sqrt{(x_1-y_1)^2+(x_2-y_2)^2+...+(x_k-y_k)^2} \\ &= \sqrt{(f_1(t)-g_1(t))^2+(f_2(t)-g_2(t))^2+...+(f_k(t)-g_k(t))^2} \\ &= \sqrt{(f(1)-g(1))^2+(f(2)-g(2))^2+...+(f(k)-g(k))^2} \\ &= \sqrt{\sum_{i=1}^{k}(f(i)-g(i))^2} \end{aligned}$$

We have:

$$\begin{aligned} (f(i)-g(i))^2 &= ((ai^2+bi+c)-(mi^2+ni+p))^2 \\ &= ((a-m)i^2+(b-n)i+(c-p))^2 \\ &= (a-m)^2i^4+2(a-m)(b-n)i^3+((b-n)^2+2(a-m)(c-p))i^2+2(b-n)(c-p)i+(c-p)^2 \end{aligned}$$

Let: $\alpha_4 = (a-m)^2$, $\alpha_3 = 2(a-m)(b-n)$, $\alpha_2 = (b-n)^2 + 2(a-m)(c-p)$, and $\alpha_1 = 2(b-n)(c-p)$, we have the Euclidean distance of **X** and **Y** is:

$$\begin{aligned} D(X,Y) &= \sqrt{\sum_{i=1}^{k}(f(i)-g(i)^2)} \\ &= \sqrt{\alpha_4\sum_{i=1}^{k}i^4+\alpha_3\sum_{i=1}^{k}i^3+\alpha_2\sum_{i=1}^{k}i^2+\alpha_1\sum_{i=1}^{k}i+k(c-p)^2} \end{aligned}$$

where:

$$s_4 = \sum_{i=1}^{k}i^4 = \frac{k(k+1)(2k+1)(3k^2+3k-1)}{30}$$

$$s_3 = \sum_{i=1}^{k}i^3 = \frac{k^2(k+1)^2}{4}$$

$$s_2 = \sum_{i=1}^{k}i^2 = \frac{k(k+1)(2k+1)}{6}$$

$$s_1 = \sum_{i=1}^{k}i = \frac{k(k+1)}{2}$$

6.1.2.3. Computing z-Normalized Euclidean distance between two time series using approximated functions

One of the implementations of Matrix Profile uses z-normalized Euclidean distance to compute distance profile. For this reason, we derive the z-Normalized Euclidean distance formula between two polynomial functions.

Consider two time series X and Y, represented by two polynomial functions f(x) and g(x) as the above section. The z-Normalized Euclidean distance between X and Y is:

$$\begin{aligned}\hat{D}(X,Y) &= \sqrt{\sum_{i=1}^{k}(\hat{x}_i - \hat{y}_i)^2} \\ &= \sqrt{\sum_{i=1}^{k}(\frac{x_i - \mu_x}{\sigma_x} - \frac{y_i - \mu_y}{\sigma_y})^2} \\ &= \sqrt{\sum_{i=1}^{k}(\frac{f(i) - \mu_x}{\sigma_x} - \frac{g(i) - \mu_y}{\sigma_y})^2}\end{aligned}$$

where μ_x and σ_x are the mean and the standard deviation of the time series X, and μ_y and σ_y are the mean and the standard deviation of the time series Y, respectively.

The mean and the standard deviation of X are computed as:

$$\begin{aligned}\mu_x &= \frac{1}{k}\sum_{i=1}^{k} f(i) \\ &= \frac{1}{k}\sum_{i=1}^{k}(ai^2 + bi + c) \\ &= \frac{1}{k}(as_2 + bs + kc)\end{aligned}$$

$$\begin{aligned}\sigma_x &= \sqrt{\frac{1}{k}\sum_{i=1}^{k}(f(i) - \mu_x)^2} \\ &= \sqrt{\frac{1}{k}\sum_{i=1}^{k}(ai^2 + bi + c - \mu_x)^2} \\ &= \sqrt{\frac{1}{k}\sum_{i=1}^{k}[a^2i^4 + 2abi^3 + (b^2 + 2ac - 2a\mu_x)i^2 + 2(c - \mu_x)bi + (c - 2\mu_x)^2]} \\ &= \sqrt{\frac{1}{k}[a^2s_4 + 2abs_3 + (b^2 + 2ac - 2a\mu_x)s_2 + 2(c - \mu_x)bs_1 + k(c - 2\mu_x)^2]}\end{aligned}$$

Similarly, we compute the mean and the standard deviation of Y as:

$$\mu_y = \frac{1}{k}\sum_{i=1}^{k} g(i)$$
$$= \frac{1}{k}(ms_2 + ns + kp)$$

$$\sigma_y = \sqrt{\frac{1}{k}\sum_{i=1}^{k}(g(i) - \mu_y)^2}$$
$$= \sqrt{\frac{1}{k}[m^2 s_4 + 2mns_3 + (n^2 + 2mp - 2m\mu_y)s_2 + 2(p - \mu_y)ns_1 + k(p - 2\mu_y)^2]}$$

Next, we compute z-Normalized Euclidean distance between X and Y. Let: $L = \sigma_x\mu_y - \sigma_y\mu_x$, $A = \sigma_y a - \sigma_x m$, $B = \sigma_y b - \sigma_x n$, and $C = \sigma_y c - \sigma_x p$. We have:

$$\hat{D}(X,Y) = \sqrt{\sum_{i=1}^{k}(\frac{f(i) - \mu_x}{\sigma_x} - \frac{g(i) - \mu_y}{\sigma_y})^2}$$
$$= \frac{1}{\sigma_x\sigma_y}\sqrt{A^2 s_4 + 2ABs_3 + (B^2 + 2AC + 2AL)s_2 + 2B(C + L)s_1 + k(C + L)^2}$$

6.1.3. Matrix Profile Construction using Segment View

Based on the Matrix Profile procedure and the Euclidean distance computed using polynomial functions presented in the previous sections, it is straightforward on how we can intervene in the process to construct the Matrix Profile directly from data models. More specifically, the intervention occurs at the step of computing the distance profile, in which the Euclidean distance computation is performed on the segment view rather than on the time series. Once the distance profile is built, the profile index will be constructed following the original MP algorithm. The mining algorithms that built on top of this MP data structure to find motifs and discords will be the same without any changes. Below, we provide the pseudo code on how to construct the distance profile using the segment view.

Consider the time series **X** = {x_1, x_2, ..., x_n} of length n, and a query q = {q_1, q_2,..., q_m} of length m. Let **F** = {f_i, i=1,...,k} be a set of polynomial functions used to approximate **X**. Each function f_i in F has length l_i. The below algorithm outlines how to construct the distance profile using the segment view.

Algorithm 1: Distance profile using segment view

Input: $\mathcal{F} = \{f_i, i = 1, ..., k\}$: set of polynomial functions used to approximate time series X,
n: length of the time series X,
q: query,
m: length of query q

Output: d: distance profile

$s_i \leftarrow 0$; /* start index of a window */
$e_i \leftarrow s_i + m$; /* end index of a window */
$w \leftarrow getWindow(s_i, e_i)$; /* get the first window at index 0 of length m */
while $e_i < n$ **do**
  $f \leftarrow getPolyFunction(\mathcal{F}, [s_i, e_i])$; /* get functions f of segment $[s_i, e_i]$ */
  $ed \leftarrow computeEuclideanDistance(f, q)$; /* compute Euclidean distance between f and q */
  if $ed < getDistance(d, s_i)$ **then**
    $d \leftarrow d.insert(ed, s_i)$; /* update distance profile with distance ed */
  s_{i++};
  $e_i \leftarrow s_i + m$;
  $w \leftarrow getWindow(s_i, e_i)$;

In the description above, we demonstrated how we can compute the Euclidean distance between two polynomial functions, and provided the pseudo code to apply this computation to construct the distance profile directly from data models used for lossy compression in ModelarDB. In our next steps, we will test and assess the presented method on the project's datasets, and further explore other directions to construct the Matrix Profile, including using Fourier transform and convolution. We will also study the lossless compression and the method to compute MP using this compression type.

6.2. Motif-based soiling detection

In this section we present our ongoing work on adapting the Matrix Profile, and in particular, exploring and identifying composite Annotation Vectors, in order to effectively identify complex events of specific type. In our setting, complex events are gradual deviations (degradations) of a particular variable of a multidimensional time series; specifically, we examine soiling events that are characterized by such a behavior on the power output variable.

6.2.1. Method overview

As described above a Matrix Profile is an index of the time series build to measure minimum distances defined on subsequences of length m of the time series. The MP maintains, for each position in the time series (having excluded the last m – 1 points), (n - m + 1) minimum distances to the rest of the time series segments. Small distances in the MP denote motifs, i.e., similar patterns, while large distances denote discords, i.e., anomalies/outliers. While MP comprises a widely used toolbox of algorithms for pattern extraction, which are proven to effectively handle time series pattern extraction problems in a variety of tasks, it is evident that it requires a considerable amount of domain specific parameterization in order to be able to detect, complex, RES related events, as the ones handled in the project. Considering our use case example, soiling events are not expected to be very short, very similar, very frequently happening events, as would be a more suitable pattern for a vanilla MP motif discovery algorithm.

Annotation Vectors (AV) are a valuable tool of the MP, which allows to weight the significance of different areas of a MP index (and correspondingly of the time series itself), with respect to the motif discovery process. This way, the motif discovery algorithm can essentially emphasize on areas of the

time series with specific behavior; the behavior can be defined with respect to various variables of the multidimensional time series.

In our setting, AV are exploited as follows. Given a labeled multivariate time series as our input, where each point is characterized by a label presenting a percentage of how much soiling has occurred on a PV panel, aim to develop an annotation time series (AV), in order to weight out intervals during which our panel was in a clean state i.e., behaving optimal. Doing so, will allow the Matrix Profile to detect motifs, mostly on areas of the time series that present a power degradation, most probably due to soiling. By exploring a large number of motif discovery configuration, with respect to the granularity of the input time series and the length of the searched pattern-motif, we cross-check which configurations discover motifs that fall into areas of the time series that correspond to soiling, according to the available labels. This way, we try to learn the behavior of a dirty panel via the optimal motifs that are selected, so we can easily later distinguish a clean from a dirty one in either historical data or in a real-time incoming data scenario. The implemented workflow comprises three steps, as described next.

6.2.1.1. Preprocessing

Preprocessing comprises a significant step, since the quality and granularity of the input data can considerably affect the quality and the robustness of the output. For this reason, we implement and explore a set of preprocessing functionalities that the user can choose to apply before proceeding to motif discovery.

- We allow the user to resample the dataset to a desired granularity level. In particular, in the examined use case, we know that soiling is a slow-in-time procedure, so observing a PV panel in a daily basis would be a reasonable indicator, so we end up resampling the data from 15-minutes to 1-day intervals, and also perform smoothing via a moving average on 7 days. Nevertheless, in the implemented workflow, the aforementioned choices are parameterizable for the end user to experiment with.
- We allow the user to apply methods for outlier detection, such as the Interquartile Method (IQR) and Local Outlier Factor (LOF).
- We implement simple normalization operations such as the scaling of variables into the [0, 1] range with min-max scaling, to reduce the effects of large numerical differences between features before applying the Matrix Profile algorithms.

6.2.1.2. Annotation Vectors Creation

Immediately after passing the dataset through the preprocessing pipeline, a set of different Annotation Vectors are defined, aiming to capture correlations of different time series variables with the soiling phenomenon, based on the obtained domain knowledge on the task. Each of these AVs is a time series, of equal length with the PV module multivariate time series, of real numbers in the range of [0,1], which denote the importance of different areas of the time series with respect to potential soiling events. This importance is directly reflected in the motif discovery processing, e.g., a value of 1 in the AV means that any motif starting at that time interval is potentially very important and must be conserved whereas a 0 means that the computed motif can and should be ignored. Of course, any values between 1 and 0 is possible to exist and the definition of those values could simply mean that larger values correspond to more important motifs than motifs assigned to lower values. As a result, annotations vectors allow users to ignore insignificant patterns in a time series. We have currently defined and experimented with three AVs, as described next:

- **PI ratio AV**: We form a ratio of Power over Irradiance, where the power is the raw output power which our PV panel provides in the time measured and irradiance is the rate at which solar energy falls onto a surface of the panel. Under optimal conditions, the PV module is expected to absorb the maximum of the solar energy which is provided daily.

Underperformance, i.e., ratio under the maximum ratio value recorded, implies a potential soiling phenomenon. Given that, the annotation vector is defined as: $\frac{\max PI_{ratio} - PI_{ratio}}{\max PI_{ratio} - \min PI_{ratio}}$.

- **PT ratio AV**: We form a ratio of Power over Module Temperature, where the module temperature is the PV panels back surface temperature, measured in the centre of the cell near the actual PV module centre. The intuition for this vector comes from the idea that soiling effects are mostly dirt or sand on the panel cells. This could mean that the panel cannot absorb the optimal power even with optimal irradiance due of its soiled surface. This behavior becomes suspicious when the module temperature is high but the absorbing power is not distributed in a same way. Given that, the annotation vector is defined as: $\frac{\max PT_{ratio} - PT_{ratio}}{\max PT_{ratio} - \min PT_{ratio}}$.

The constructed AVs are used to adapt (correct) the computed Matrix Profiles, according to the following formula:

$$CMP = MP + (1 - AV) \cdot \max(MP)$$

Furthermore, our workflow supports the configurable creation of Matrix Profiles over various pattern-motif lengths, allowing the exploration of optimal motif lengths to represent soiling phenomena.

6.2.1.3. Detection of Motifs

In the last steps, we implement a configurable algorithm for the selection of the desirable number of motif types, as well as the maximum/minimum instances each motif type can have. Lastly, we return the graphical illustration for all the variations the user chose along with a frame which holds important statistical information of the aforementioned procedure.

6.2.2. Implementation information

The motif discovery tool is implemented in Python, currently as a Jupyter notebook linked to modules developed in the scope of D4.1. For efficiently managing, preprocessing, and performing numerical operations on the data, we use pandas[3], scikit-learn[4], numpy[5] and stumpy library. The notebook can be found at https://github.com/MORE-EU/complex-event-detection/tree/main/notebooks/soiling_motif_experiments/Motifs_Detection_Experiments. In particular, the notebook specializes our method on the soiling use case.

Apart from methods developed in D4.1, new functions have been defined aiming to support the new functionalities introduced in this deliverable, and the corresponding experimental evaluation. A list of the new functions follows:

- **pi_ratio_av:** An annotation vector function.
- **pt_ratio_av**: An annotation vector function.
- **ci_ratio_av:** An annotation vector function.
- **clean_motifs**: Returns the indices and the distance of the computed motifs associated with the previous annotation vectors after preprocessing.
- **Ranker**: Given the output of the function clean_motifs, and the data frames of the rains/washing intervals, returns a dictionary containing all relevant information about motifs plus the ranking of the computed motifs.
- **Get_corrected_matrix_profile:** Returns the corrected/weighted matrix profile which is created from a user defined/already implemented Annotation Vector
- **motif_plot:** Returns the plots of each motif type for each pattern length
- **summary_motifs:** Returns a dataframe of statistical summaries of the motifs

Our main technical contribution is to provide an implementation of our method, in the form of a Jupyter notebook. The notebook implements all different variants of the method described in Section

6.1. In particular, the detection of the raining periods following the manual cleanings of the solar parks is in cell [3]. Then, creation of a dictionary of different matrix profiles can be found in cell [4]. Cell [5] is the creation of the annotation vectors and computation of the corrected matrix profile. Cell [8] implements the computation of the motifs, as the ranking of those and the creation of the data frame output which holds all the relevant information by using the aforementioned methods. Lastly in cell [9] the user can observe a graphical illustration of those motifs over the different pattern length periods and types of motifs that our method computed along with relevant graphs from our dataset variables.

6.2.3. User guide

6.2.3.1. 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/complex-event-detection --recursive

One new folder under the name complex-event-detection will be created. Let <path> be the absolute path to that folder, i.e., the new folder is <path>/ complex-event-detection.

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 - 1.3.3
- matplotlib - 3.4.2
- numpy - 1.20.1
- stumpy - 1.8.0

The above libraries are included in the requirements.txt file. 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>/ complex-event-detection/notebooks/soiling_motif_experiments/Motifs_Detection_Experiments

6.2.3.2. Configuration options

In this section we list a set of parameters which allow the user to test different configurations that correspond to different variants of our methods.

- **presi**: The parameter of filtering significant rains with more than 'presi' precipitation. It can be found in cell [3]
- **dates_wash_start**: List of the date of which a cleaning in the park started. It can be found in cell [3].
- **dates_wash_stop**: List of the date of which a cleaning in the park stopped. It can be found in cell [3].
- **days_dict**: Dictionary with the lengths of the different days which the user creates the matrix profile It can be found in cell [4].
- **mps**: Dictionary which keeps all the computed matrix profiles over the days_dict. The user can choose the variable which prefers to calculate the one-dimensional profile. The default is set to calculate the power output of the panel. It can be found in the cell [4].
- **corrected**: The corrected matrix profile with respect to an annotation vector which implements the clean_motifs. It can be found in cell [7].
- **mot**: The arithmetic values that were used in the days_dict in order to loop over all the different motifs lengths. It can be found in cell [7].
- **max_motifs**: The maximum number of motifs to return. It can be found in cell [7].
- **min_nei**: The minimum number of similar matches a subsequence needs to have in order to be considered a motif. A subsequence must have at least one similar match in order to be considered a motif i.e, the Default value is set to 1. It can be found and changed in cell [7].
- **max_di**: For a candidate motif, *M*, and a non-trivial subsequence, *S*, *max_di* is the maximum distance allowed between *M* and *S* so that *M* is considered a match of *Q*. If *max_di* is a function, then it must be a function that accepts a single parameter, *D*, which is the distance profile between *M* and *the time series*. If None, this defaults to *np.nanmax([np.nanmean(D) - 2.0 * np.nanstd(D), np.nanmin(D)]).* It can be found and changed in cell [7].
- **cut**: The largest matrix profile value (distance) that a candidate motif is allowed to have. It plays the same role as max_di the distance function considers the matrix profile of the time series. It can be found in cell [7].
- **max_matc**: The maximum number of similar matches of a motif to be returned. The resulting matches are sorted by distance, so a value of *D* means that the indices of the most similar *D* subsequences is returned. If *None*, all matches within *max_di* of the motif representative will be returned. It can be found and changed in cell [7].

In cell [5] the procedure of the annotation vector creation and calculation of the new corrected matrix profile is taking place, which is a fully user parametrized procedure where the user can define its own vectors and transfer them in a matrix profile setting just by following the same steps.

6.2.3.3. Deployment instructions and options

The input time series is provided in the form of .csv file. This is defined in cell [2]. Moreover, in cell [2], the user can define the change of granularity and the smoothing of the panel's dataset.

In the next cell [3], the user can define the periods during which the solar panels were manually cleaned and the filtering of the rains which occurred. In particular, dates_wash_start contains the exact dates on which manual cleanings were initiated, and dates_wash_stop contains exact dates on which manual cleanings were terminated with respect to the notation dates_rain_start, dates_rain_stop will implement the same procedure for the rains. Both are going to be saved in a data frame notation. The input .csv file is expected to contain columns under the names defined as our notebook model set. Cell [7] calculates the dictionary and the ranking of the calculated motifs and outputs/saves a .csv file of all the important statistical analysis of those motifs and a set of plots containing all of them.

6.2.4. Evaluation

Next, we present an indicative set of motifs that are discovered in the soiling use case. We use the same dataset with one described in Section 4.4.1, which comprises a multivariate PV modules time series, along with a time series representing the soiling derate of the module, serving as labeled data.

We first present, in Figure 18, the labeled data on soiling derate, which clearly indicate which areas of the PV modules time series are experiencing a soiling phenomenon and at what extent.

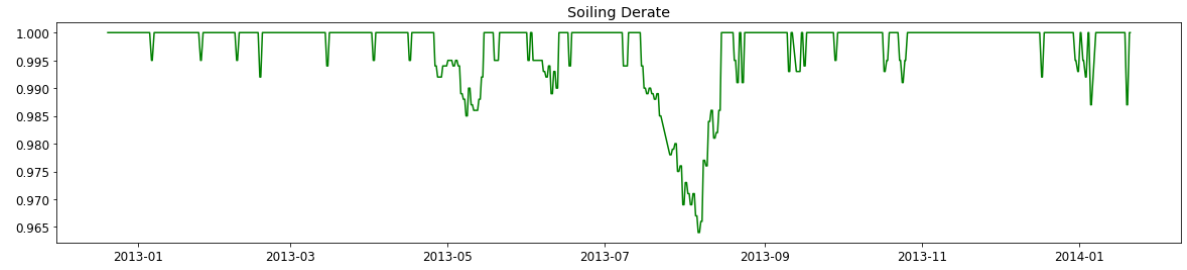

Figure 18: Labelled data on soiling degradation (derate)

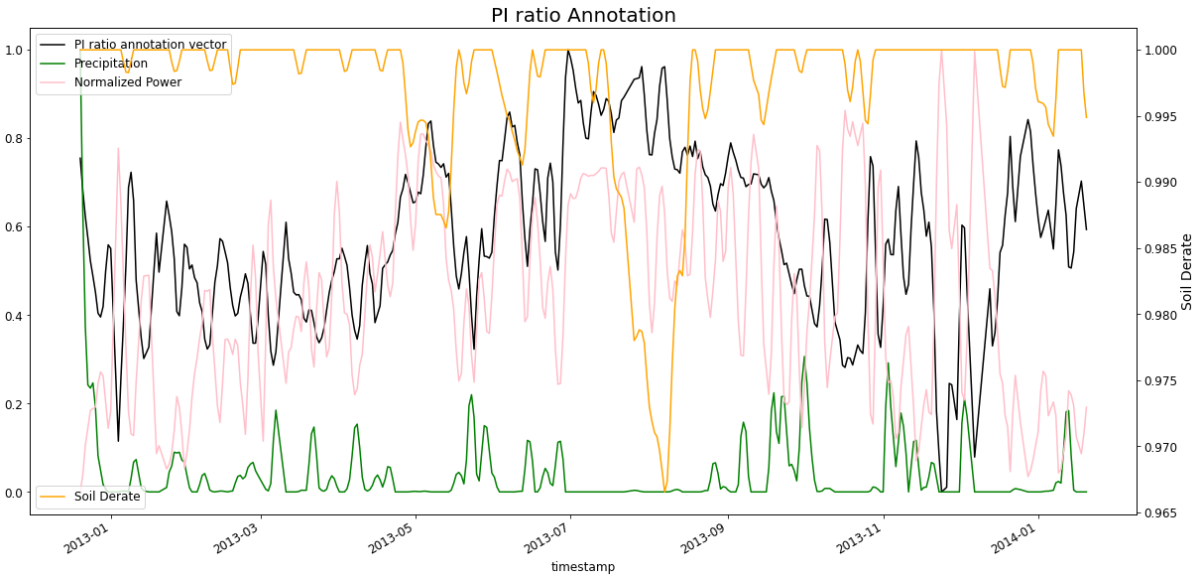

Figure 19 demonstrates (in black color) the PI AV, along with several variables on of the time series, as well as the soiling derate (in yellow and in different scale, so that the soiling events are clearly demonstrated). From this figure, one can observe a tendency of peaks in the PI AV when soiling events take place.

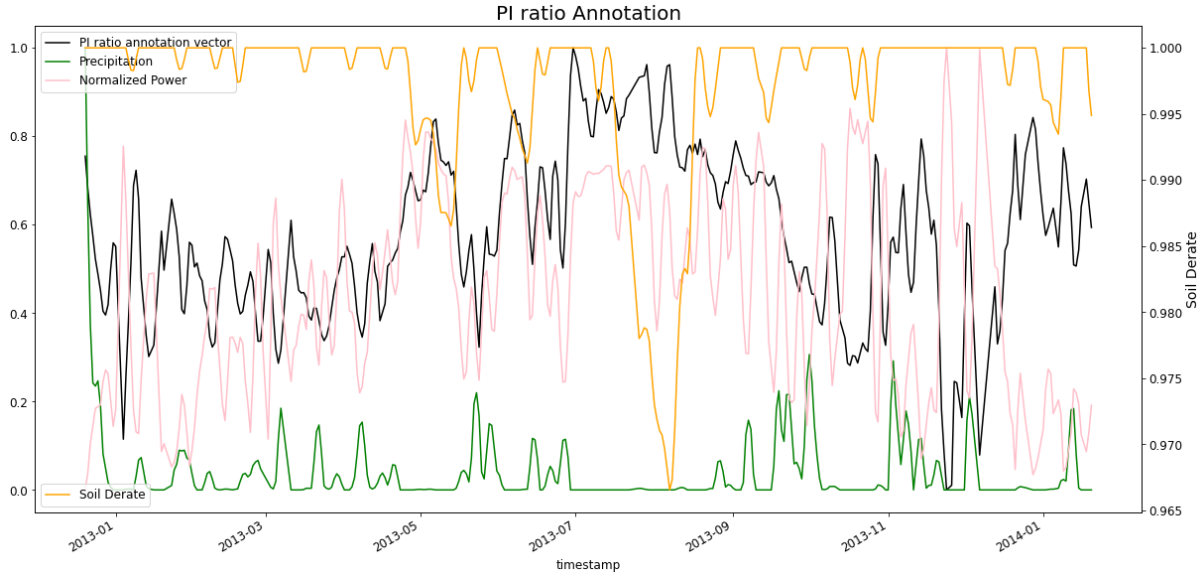

Figure 19: Annotation Vector on Power to Irradiance (PI) ratio. The soiling derate is shown in yellow and in different scale, so that the soiling events are clearly demonstrated.

Figures Figure 20 and Figure 21 present two different motif types that are detected by using the PI Annotation Vector on the Matrix Profiles constructed for 20-days and 30-days motif lengths respectively. We can observe that both motif types are able to detect several of the soiling phenomena (denoted by the soiling derate with blue lines), with the second motif type (20-days length) being more sensitive in the identification of soiling.

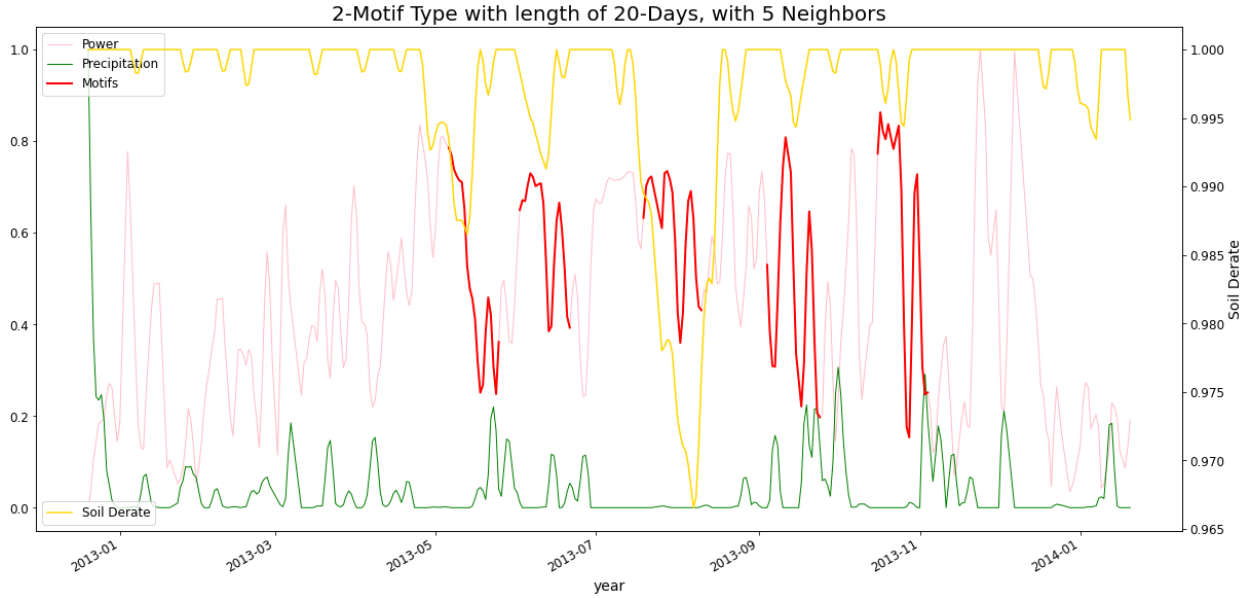

Figure 20: Representative motif instances detected using the MP and the PI AV. The soiling derate is shown in yellow and in different scale.

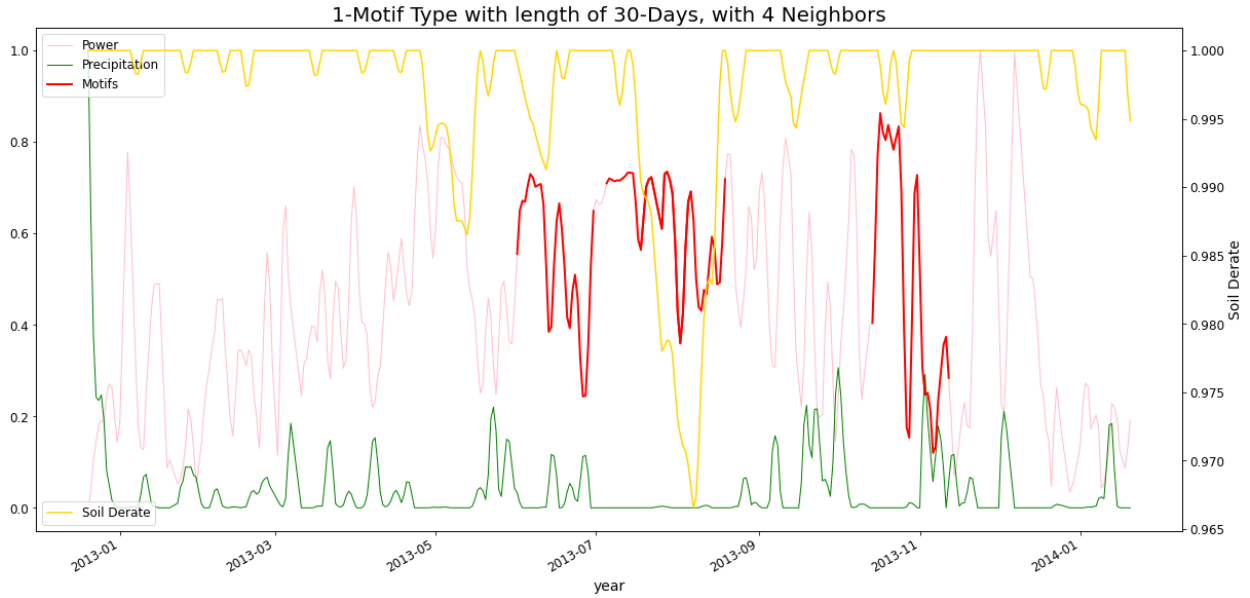

Figure 21: Representative motif instances detected using the MP and the PI AV. The soiling derate is shown in yellow and in different scale.

Similarly, Figure 22 demonstrates (in black color) the PT AV, along with several variables on of the time series, as well as the soiling derate (in blue). A similar tendency of peaks in the PT AV when soiling events take place is also observed here.

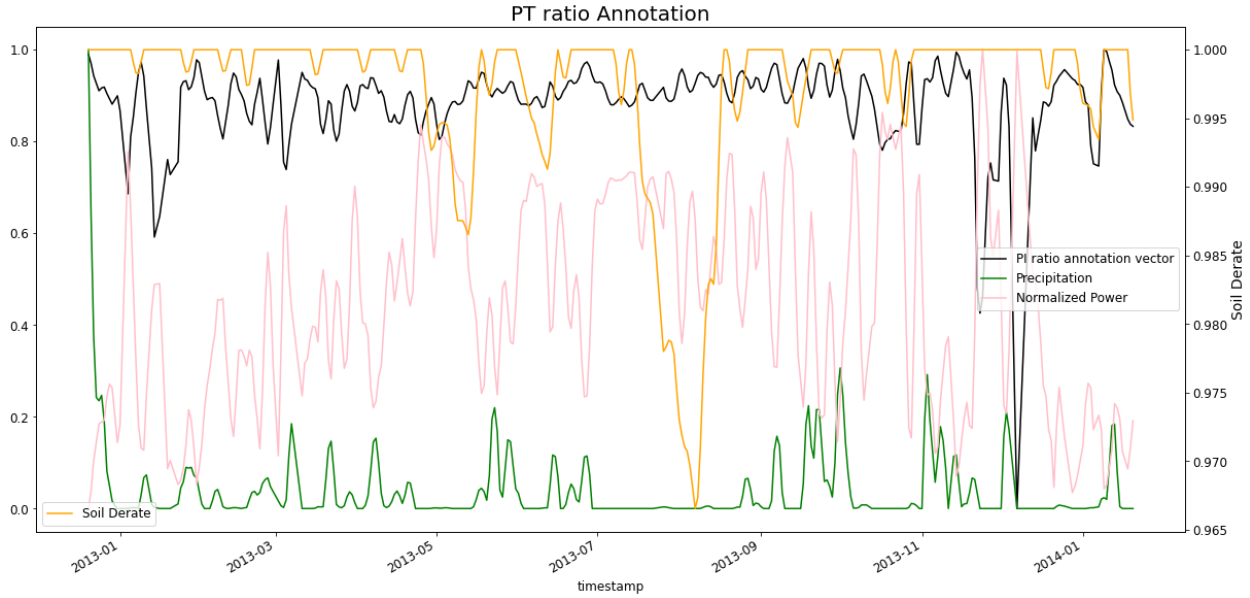

Figure 22: Annotation Vector on Power to Temperature (PT) ratio. The soiling derate is shown in yellow and in different scale.

Figures Figure 23 and Figure 24 present two different motif types that are detected by using the PT Annotation Vector on the Matrix Profiles constructed for 15-days and 20-days motif lengths respectively. Similarly, both motif types are able to detect several of the soiling phenomena (denoted by the soiling derate with blue lines), with the second motif type (20-days length) being more sensitive

in the identification of soiling. In general, the PT AV seems to produce more sensitive motifs with respect to soiling identification.

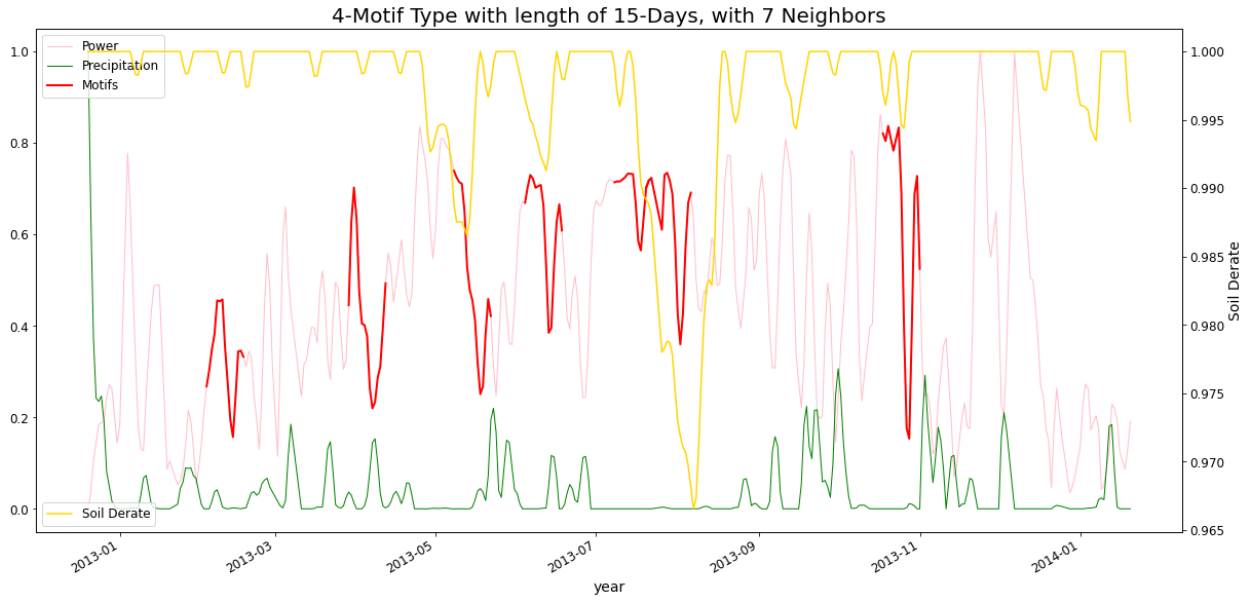

Figure 23: Representative motif instances detected using the MP and the PT AV. The soiling derate is shown in yellow and in different scale.

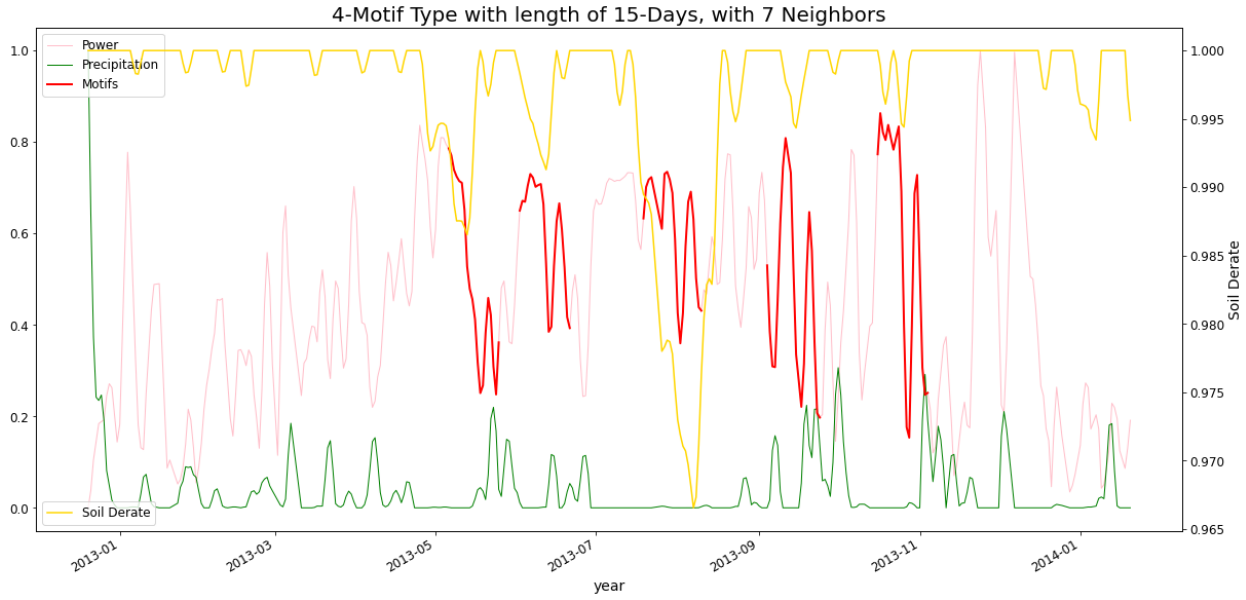

Figure 24: Representative motif instances detected using the MP and the PT AV. The soiling derate is shown in yellow and in different scale.

Overall, the above examples demonstrate a concrete potential of the particular approach on identifying soiling events. Nevertheless, a more in-depth analysis and more complete exploration of more composite Annotation Vectors is required in order to reach to more definite conclusions. Further, it appears that the so far assessed configurations are problematic in identifying the rightmost, mild soiling events, requiring thus further examination with respect to the proper Annotation Vectors that will allow their identification too.

7. Progress on functional specifications and KPIs

In this section, we enumerate the functional specifications regarding the Complex Event Detection module (CED) 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. Finally, we report on KPIs from the description of work of MORE that are related to the work presented in this deliverable.

For each functional specification, its implementation status is provided, considering two stages:

- **Implemented** denotes a fully functional implementation of the respective specification, with or without potential improvements to be planned for next releases.
- **Pending** denotes that the particular specification is planned to be implemented in an upcoming release.

Functional specification	Implementation status – Initial version	Comments
FS-CED-01 Update existing time series indexes	Pending	-
FS-CED-02 Identify significant patterns/events on a univariate/multivariate time series in nearly-real time	Implemented (see Sections 4.1, 6.2)	Potential additions and extensions (e.g. use of the Matrix profile index) in the final version.
FS-CED-03 Identify significant anomalies on a univariate/multivariate time series in nearly-real time	Implemented (see Sections 4.1, 5.1)	Potential additions and extensions (e.g. use of the Matrix profile index) in the final version.
FS-CED-04 Identify changes in the time series behavior of a univariate/multivariate time series in nearly-real time	Implemented (see Sections 4.1, 5.1, 6.2)	Potential additions and extensions (e.g. use of the Matrix profile index) in the final version.
FS-CED-05 Identify trends in a univariate/multivariate time series in nearly-real time	Implemented (see Sections 4.1.2, 4.1.3)	Potential additions and extensions (e.g. supporting multiple variables) in the final version.
FS-CED-06 Identify frequently cooccurring patterns on a univariate/multivariate time series	Pending	-
FS-CED-07 Automatically identify optimal Matrix Profile parameters	Pending	-
FS-CED-08 Automatically identify optimal Matrix Profile parameters	Pending	-

FS-CED-09 Identify patterns on a univariate/multivariate time series based on sparsely supervised data	Implemented (see Subsection 4.1.2)	Semi-supervised variations of the method for changepoint/deviation detection.

Table 4: Functional specifications and implementation status for the Complex Event detection module

Shortly, out of 9 prescribed functional specifications, 5 are fully implemented and 4 are pending. Of course, several of the already implemented specifications will be constantly revisited as Task 4.2 proceeds, updating, improving and extending the currently implemented functionality.

Regarding KPIs from MORE's description of work, Table 5 presents the progress on selected KPIs that were deemed relevant to the current progress of Task 4.2.

Key Performance Indicator	Description (copied from DoW)	Progress
Patterns detected in real time	**Before:** Complex patterns based on motifs cannot be detected by TSMS systems **After:** Patterns will be detected in subsecond time for thousands of streams (that might be related to each pattern, the system will be able to scale to millions and billions of streams)	Significant progress has been performed in detecting both motif- and regression-based patterns on RES time series. The implemented methods are able to accurately detect such patterns on two of the project's use cases, on respective real-world datasets. Initial experiments demonstrate large scaling potential on (nearly) real-time deployment.
Number of detected events	**Before:** No detection **After:** Previously mined events will be detected in real time	The presented methods are able to exploit both trained models and precomputed patterns to detect new events in real-time.
Accuracy of prediction/classification	**Before:** TSMS do not offer prediction/classification tools, domain specific systems might vary on the accuracy and depend highly on expert knowledge **After:** Beyond the state-of-the-art accuracy of ML and pattern based methods, incorporated to MORE platform	We demonstrate almost excellent accuracy in detecting significant yaw misalignment in turbine time series. We demonstrate that our soiling detection method outperforms a state-of-the-art method of the recent literature.
Detected malfunctions	**Before:** Only significant power failures can be detected **After:** Power degradations will be detected in real-time and attributed to specific causes	Both behavior detection and behavior deviation detection methods are demonstrated to be able to detect power degradations that are quite hard to detect in the

		current use cases (yaw misalignment and soiling).

Table 5: Progress on relevant KPIs

As summarized in Table 5, we have made significant progress in various aspects of performance, which are of key importance to our ultimate objectives in Task 4.2. We provide strong evidence of scalability for our pattern detection methods in real-time scenarios, and we claim accuracy of our methods by juxtaposing them to the state-of-the-art. Moreover, we are able to detect events or malfunctions and attribute them to real causes, in scenarios where this was not possible.

8. Conclusion

Deliverable 4.3 presents the initial version of the complex event detection methods of the MORE platform. Our preliminary set of methods cover a large part of the functional specifications for the Complex Event detection module, defined in D5.2. Further, they are motivated and furthermore tested in the prescribed use cases of D5.1, demonstrating encouraging effectiveness and scalability results in handling two of the use cases.

In particular, Deliverable 4.3 includes a series of methods for the detection of complex events and, specifically, the detection of particular behaviors or behavior deviations with respect to certain variables in multidimensional time series. The first set of methods is motivated by problems occurred in the soiling use case, such as the detection of changepoints and the detection of periods with slowly progressing deviating behavior. The second set of methods is motivated by the yaw misalignment detection use case in wind turbines. These methods exploit a limited set of labeled data, in the form of multivariate time series of measurements for which the yaw misalignment angle has been accurately measured by dedicated equipment (LIDAR).

Our methods are tested on accuracy, using labelled data as ground-truth when available, and on scalability by applying them on large, real-time, instances. In the soiling use case, our methods can accurately predict the soiling ratio, as supported by our experimental comparison with a state-of-the-art approach and by evaluating with labelled data provided by NREL. Moreover, our methods can be efficiently applied in an online scenario where a large number of streams of measurements must be analyzed simultaneously. This latter claim is also experimentally tested, supporting our claims of scalability. In the scope of methods aiming to treat the yaw misalignment use case, almost all proposed variations for behavior detection can very accurately identify when the yaw angle deviates from 0 degrees in the yaw misalignment setting, which is of high real-world value, since our analysis shows that major drop in power output is observed even for very small angle values.

Our ongoing work includes improving and extending our current methods with a focus on further exploring the applicability of the matrix profile within our current solutions. This is in line with the functional specifications defined in D5.2 and the progress made so far. Furthermore, in the scope of developing generic tools which are applicable in various use cases, we aim to design query procedures which are both expressive and user-friendly. One idea in this direction is to design a method for detecting patterns in a time series, where patterns are identified by a sequence of simple predicates.

9. References

[DMM18] Michael G. Deceglie, Leonardo Micheli, and Matthew Muller. 2018. Quantifying Soiling Loss Directly From PV Yield. IEEE Journal of Photovoltaics 8, 2 (2018), 547–551. https://doi.org/10.1109/JPHOTOV.2017.2784682

[LJ21] Gao, Linyue & Hong, Jiarong. Data-driven yaw misalignment correction for utility-scale wind turbines. Journal of Renewable and Sustainable Energy 13, 063302 (2021); https://doi.org/10.1063/5.0056671

[MAD+14] Bill Marion, Allan Anderberg, Chris Deline, Joe del Cueto, Matt Muller, Greg Perrin, Jose Rodriguez, Steve Rummel, Timothy J. Silverman, Frank Vignola, Rich Kessler, Josh Peterson, Stephen Barkaszi, Mark Jacobs, Nick Riedel, Larry Pratt, and Bruce King. 2014. New data set for validating PV module performance models. In 2014 IEEE 40th Photovoltaic Specialist Conference (PVSC). 1362–1366. https://doi.org/10.1109/PVSC.2014.6925171

[S68] Pranab Kumar Sen. 1968. Estimates of the Regression Coefficient Based on Kendall's Tau. J. Amer. Statist. Assoc. 63, 324 (1968), 1379–1389. https://doi.org/10.1080/01621459.1968.10480934

[T92] Henri Theil. 1992. A Rank-Invariant Method of Linear and Polynomial Regression Analysis. Springer Netherlands, Dordrecht, 345–381. https://doi.org/10.1007/978-94-011-2546-8_20

[WMZ10] William Webber, Alistair Moffat, and Justin Zobel. 2010. A Similarity Measure for Indefinite Rankings. ACM Trans. Inf. Syst. 28, 4, Article 20 (nov 2010), 38 pages. https://doi.org/10.1145/1852102.1852106