EC5.25

Methodological Guideline for Time Series Anomaly Detection

Document reference number for ANR

contact@confiance-ai.fr | www.confiance.ai

Document reference: XXX

Contributors

	Name	Organisation
Responsible for the deliverable	Marc Nabhan	Air Liquide R&D
Co-authors	Martin Royer	IRT SystemX et Datashape, INRIA
	Mouhcine Mendil	IRT Saint Exupéry
	Olivier Antoni	CEA Grenoble
	Kevin Bleakley	INRIA Saclay
	Marielle Malfante	CEA Grenoble
	Laurent Chiasson Poirier	CEA Grenoble
	Andrés Troya-Galvis	Thales Alenia Space
	Christophe Gouguenheim	Thales Alenia Space
	Fred Ngole Mboula	CEA Saclay
	Kevin Pasini	IRT SystemX
Reviewers	Clément Arlotti	IRT SystemX
	Guillaume Chambaret	Air Liquide R&D

Document Control

Revision	Date	Commentary	Author
v1.0	13/10/2023	Document creation	Marc Nabhan
v1.1	11/12/2023	Stable version & Reviewing start	Marc Nabhan
v2.0	22/12/2023	Reviewed version	Marc Nabhan

Contents

A. Introduction and abstract

A.1. General introduction to trustworthy AI challenges

Trustworthiness in AI within critical systems (systems that can directly or indirectly affect human life and moral entities) is essential for its widespread adoption (by the industry, the decision makers, the general public, etc.) and poses the following significant challenges.

- First, how to design AI models, so that, by construction, they satisfy trustworthy properties (accuracy, robustness. . .).
- Secondly, how to characterize these AI models, for example to understand and explain their behavior and their adequacy to the operational domain.
- Then, how to implement and embed those AI models on hardware, by making them fit for the target without losing their trustworthy properties.
- Another question is, what methods of data engineering to apply in order to, among other topics, manage important volumes of data and adapt to the evolution of the operational domain.
- At system level, what verification and certification processes to consider specifically for AI-based systems.
- Finally, a federation of all these matters is necessary to build an end-to-end methodological approach, supported by a consistent engineering environment compatible with industrial practices.

These are the challenges, among others, that the Confiance.ai program addresses.

A.2. Context and scientific challenges

Data is everywhere. Whether it consists of sensors and logs, text or audio, the world we currently live in has data sources practically everywhere. This has let a lot of algorithms to emerge and take advantage of all this data. From expert systems all the way to neural networks, these AI algorithms leverage data to learn the problem at hand. Some of them even, e.g. machine learning and deep learning models, rely heavily on a big quantity of data to be able to solve the problem successfully.

However, what happens if these AI models are presented with data that are not so reliable in terms of quality? The problem being learned may include biases and misrepresentations, leading to incorrect outputs, false alarms and wrong consequences. The idea presented here then is to enhance the data quality, in the ways of abnormal behavior detection. By identifying anomalies in the dataset, we are able to understand what corresponds to nominal or abdnormal behavior within the dataset, and grasp its operational domain more effectively. This, in turn, can help us to take the appropriate actions in order to mitigate the detected anomalies, and reach a *clean* dataset to be used by the AI algorithms.

Thus, we present this methodological guideline which stems from the project EC5 of Confiance.ai, also named "Data and knowledge engineering for trusted AI". More specifically, the aim of this guideline is to present tools and methods that have been developed to tackle time series data and detect abnormal behavior.

The main Confiance.ai scientific challenges can be summed up as follows:

- **Measure the quality and representativeness of the data/knowledge of an operational domain**. Basically, we want to detect abnormal behavior related to time series data, which in turn helps in enhancing the overall data quality depending on the chosen use case and its operational domain.
- **Identify biases in data/knowledge leading to poor decisions**. In this case, abnormal behavior in data can often lead to wrong consequences, which incites to take incorrect actions. By detecting them, we can learn to understand, mitigate, and prevent them, leading to a better handling and comprehension of the data.
- **Methodology to exploit different sources (unlabelled data, knowledge, other domains, small data)**. Here, we will mainly present tools that are able to detect abnormal behavior for unlabelled time series, which is a recurrent problem encountered in industry. Other methodologies that can be used for validation purposes without the need of labels or ground truth are also showcased.

A.3. Rationale and document organization

In short, the rationale behind this document is to provide users an entry point to all these tools that have been developed within Confiance.ai. It serves as a comprehension guide on how to use each tool and methodology, in what context, and what potential results can be expected after a successful application. The document is organized as follows:

- This first chapter serves as an introduction and draws the roadmap to this methodological guideline, with a showcase of the context, rationale, scientific objectives, organization, target audience and usage.
- Then as Part I, chapters B to G will each in turn present one of the proposed methods for anomaly detection for time series data. Methods are sorted with respect to their capacity to learn from 1-dimensional series (Chapters B to F) or multiple dimensional series (Chapters D to G) – in other words, the methods of chapters B and C only handle univariate timeseries, the method of chapter G only handles multiple timeseries, and other methods can deal with both. The aim of each chapter is to provide, amongst others, the overall vision, context and added value of the corresponding methodology, identify its prerequisites and level of maturity, and also describe the procedures while providing explanatory diagrams.
- In Part II, chapters H, I and J will propose methodologies for the calibration, evaluation without labels and validation of generic anomaly detection procedures.
- Finally, Chapter K provides a summary of limitations and perspectives.

A.4. Target audience and how to use this document

The target audience for this document is mainly Data/AI researchers and scientists, and any user who is eager to tackle anomaly detection in time series and discover the various tools that were developed for that sake in this project. The document can be used to have a comprehensive understanding of the rationale behind each tool, and to be able to assess the potential applications and limitations depending on the use case at hand.

Part I

Anomaly Detection Tools

Contents for Part I

B. CNNDRAD – Self-supervised learning for Anomaly Detection using 1D-CNN

B.1. Component description

This component implements deep 1D-CNN architectures for the Anomaly Detection (AD) task on regularly sampled univariate time series. The name of the component *CNNDRAD* stands for *1D-CNN Data Reconstruction for Anomaly Detection*.

Please find here the id card and the repository of the component.

B.2. Context of use of the component

With respect to the End-to-End Approach, this component can be used in *Data Engineering* (section C.7):

- As a component that can be integrated into an AI-based system, the value contribution of the component lies in the detection of anomalies in batch or stream processing, for example in a supervision system.
- As an engineering tool, its contribution lies in the detection of anomalies in order to either annotate them to create new datasets, or to assess the quality of existing datasets to ensure that they do not contain any undesired anomalies for training Machine Learning models.

B.3. Prerequisites for using the component

This component is aimed at *Data Scientists* (responsible for providing quality datasets) in collaboration with *Machine Learning Engineers* (to optimize the parameters related to the training of the underlying Deep Learning model) and *Technical Experts* (expert knowledge is valuable to set the anomaly threshold and validate the detected anomalies).

Important insights from the time series can be drawn by inspecting local substructures. For anomaly detection, the size of the sliding window used for extracting such subsequences is a crucial parameter and using an inappropriate value may result in poor anomaly detections.

Basically, the component aims to detect anomalies in the *test data* which are defined as something different from what can be seen in the *train data*. Thus, having access to anomaly-free train data is the best way to learn a good representation of normal data in order to limit the number of false-positive anomaly detections in the test data. But if the *train data* contains very few anomalies, it is expected to have a low impact on the learned representation of normal data and on the quality of the detected anomalies in the test data. Practically speaking, the user can try to use the anomaly detector even if the number of anomalies in the train data is unknown. In that case, if the computed anomaly scores for a given test data are very high when anomalies are not expected, or if an expert declares the detected anomalies with the highest anomaly scores as false positive, the assumption regarding the normality of the train data should probably be reconsidered and the process should be repeated with other train data containing *a priori* fewer anomalies.

As context-level pre-requisites:

- To choose the appropriate sliding window size, the nature of the time series and the dominant period of the signal must be known.
- To learn a good representation of normal data, an example of time series with very few anomalies must be available.

As human-level pre-requisites:

- The user should be familiar with anomaly detection techniques for time series, in particular the use of a sliding window on the input data to generate windowed data samples on which the anomaly detection algorithm is applied.
- Although it is not required (as the user can use default parameters), some familiarity on the training of deep learning models should be desirable, especially for the choice of the component training parameters (batch size, number of epochs, train-validation split of the dataset) and for checking the training history (evolution of the loss and validation metric during training) of the underlying deep learning model.

B.4. Level of maturity of the component

The level of maturity of the component is 2:

- It has been validated against a set of 250 public datasets, for which the component used as a black box with default parameters was able to correctly predict more than three out of four hidden anomalies.
- It has been tested on the Air Liquide Use Case called *Efficiency Monitoring System (Data Quality)*, for which some of the detected anomalies that maximize the anomaly score were reviewed and validated by the Air Liquide expert.

B.5. Procedure for using the component

For installation and quick start, please refer to the users guidelines.

The overall picture of the described procedure can be illustrated as follows:

The procedure involves three different steps:

1. The first step consists in creating the anomaly detector once the size of the sliding window to be applied on the input data is known. This initializes the 1D-CNN model that will be used later to learn a representation of normal data. It is recommended to choose the size of the sliding window approximately equal to the number of points corresponding to the period of the data. If this period is unknown, the component provides the *compute_period* method to give an evaluation based on the autocorrelation peaks of the signal.
2. The second step consists in using anomaly-free data samples (called *train data*) and a set of predefined pretext tasks to learn a representation of normal data in a self-supervised way. This is achieved by using the *fit* method that outputs the trained 1D-CNN model that was initialized in the first step.
3. The third step is dedicated to detecting anomalies in the provided data samples (called *test data*). The *score_samples* method uses the pretrained 1D-CNN model to reconstruct windowed data samples and to output anomaly scores based on the normalized reconstruction scores that measure the distance between the reconstructed windowed data samples and the original ones, the normalization being done such that 99% of the windowed data samples used for training have a reconstruction score below one. These anomaly scores can

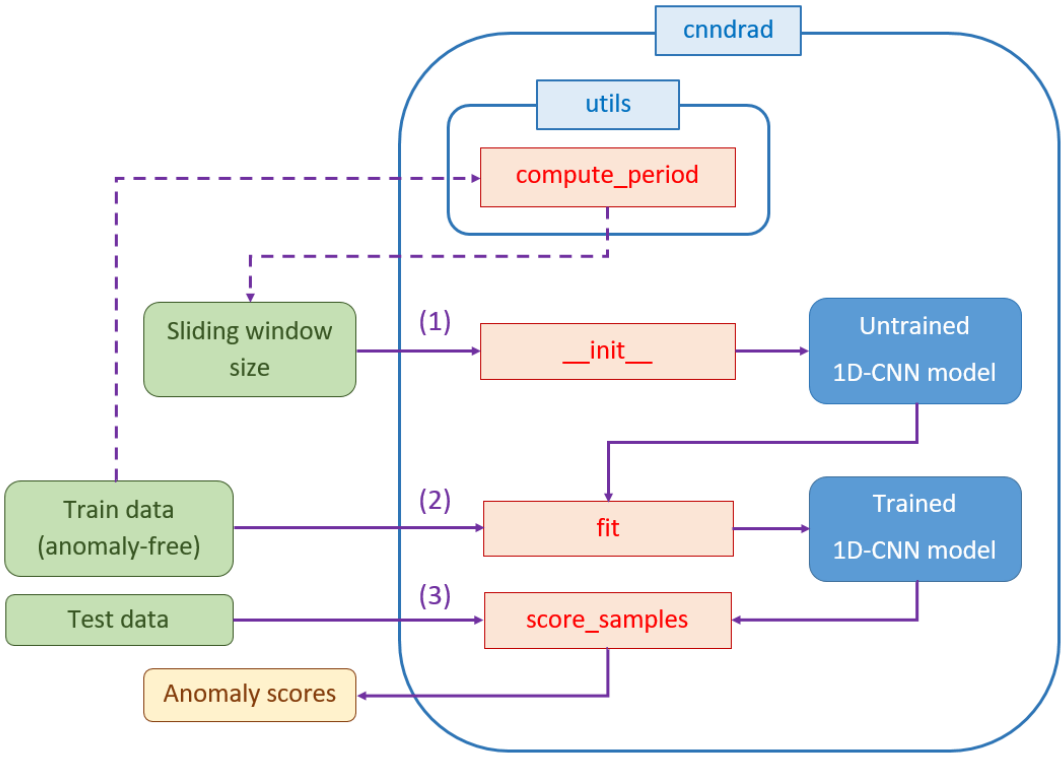

Figure B.1: A diagram overview of the procedure

finally be used by the user to report an anomaly whenever the value is greater than one, knowing that the higher this value, the greater the probability that there is an anomaly.

B.6. Demonstrators

In order to better present an operational description of the component, we deliver two comprehensive demonstrators, each one serving different and complementary purposes.

B.6.1 Library demonstrator

The library demonstrator is an example of use of the methods included in the component's library. This example is not related to any industrial use case provided within the project and can be run using the auto-executable Python script available here.

In this example, the dataset is created using two cosine functions with different periods and magnitudes. The *train data* corresponds to the first half of the dataset (from time 0 to 50), the *test data* to the second (from time 50 to 100). A synthetic anomaly is introduced into the *test data* by removing negative values for a short time interval (from time 80 to 81). The figure below shows the results available in the *score_samples.html* file after running the Python script.

B.6.2 Use-Case level demonstrator

For demonstration purpose, the component is applied to the Air Liquide Use Case called *Efficiency Monitoring System (Data Quality)* provided as part of the project, and the temperature

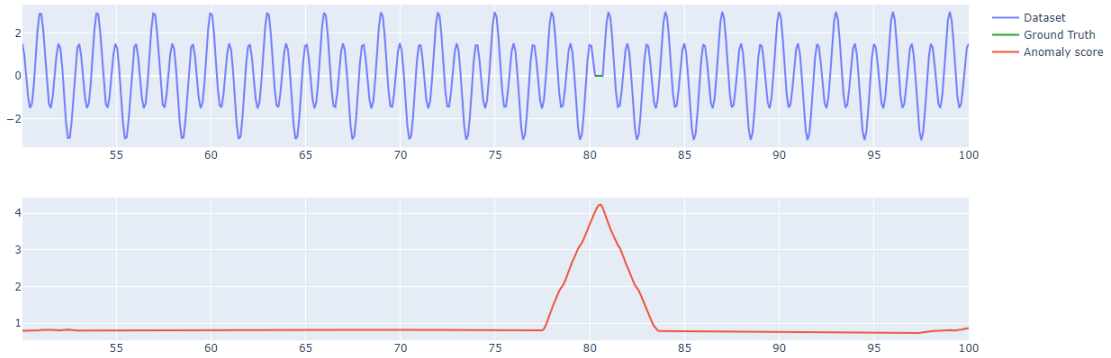

Figure B.2: Test data (top) and corresponding anomaly scores (bottom) calculated using the component's default parameters. Anomaly scores show a single peak centered around when the anomaly (green curve at top, labeled as *ground truth*) was introduced.

sensor *System_3-TI1223.PV* is selected. The dataset is created by resampling raw sensor data available over the full year 2020 at a rate of one sample per minute.

As no information is provided with the dataset, we do not know if the January data has very few anomalies and can be used as *train data*. We therefore apply the methodology explained in B.3. First the data from January is used as *train data* and other months data as *test data*. Although the detected anomalies are not bad with this choice, the anomaly scores turn out to be a little too high when no anomalies are expected, and we decide to move to February data. Using the data from February as *train data* has two positive effects: on the one hand, an anomaly in January is discovered, and on the other hand, the anomaly detection performance for other months of the year seems better. So, to detect anomalies over the year, the data from February is chosen as *train data*.

But after looking at the predicted anomaly scores, it comes out that a lot of anomalies and a change in the production regime occur in June, so the representation of the data learned previously can't be reasonably used to detect anomalies from July. A new anomaly detector is then trained to detect anomalies from July until December. As before, an anomaly is found in July, so the data from August is selected as *train data* to detect anomalies during the second half of the year.

For all these reasons and in order to maximize the quality of the detected anomalies, the demonstration is divided in two parts as follows:

- In Demo #1, the data from February is used as *train data*, and other months data from January to May as *test data*.
- In Demo #2, the data from August is used as *train data*, and other months data from June to December as *test data*.

This demonstration can be run using the auto-executable Python script available here.

The figures below show the results available in the *score_samples.html* files for each month of the year after running the Python script. Some of the detected anomalies were reviewed by the Air Liquide expert (see explanations in parenthesis after the name of the month when available), who found that they are mainly due to process anomalies, changes in the production regime, or plant shutdowns.

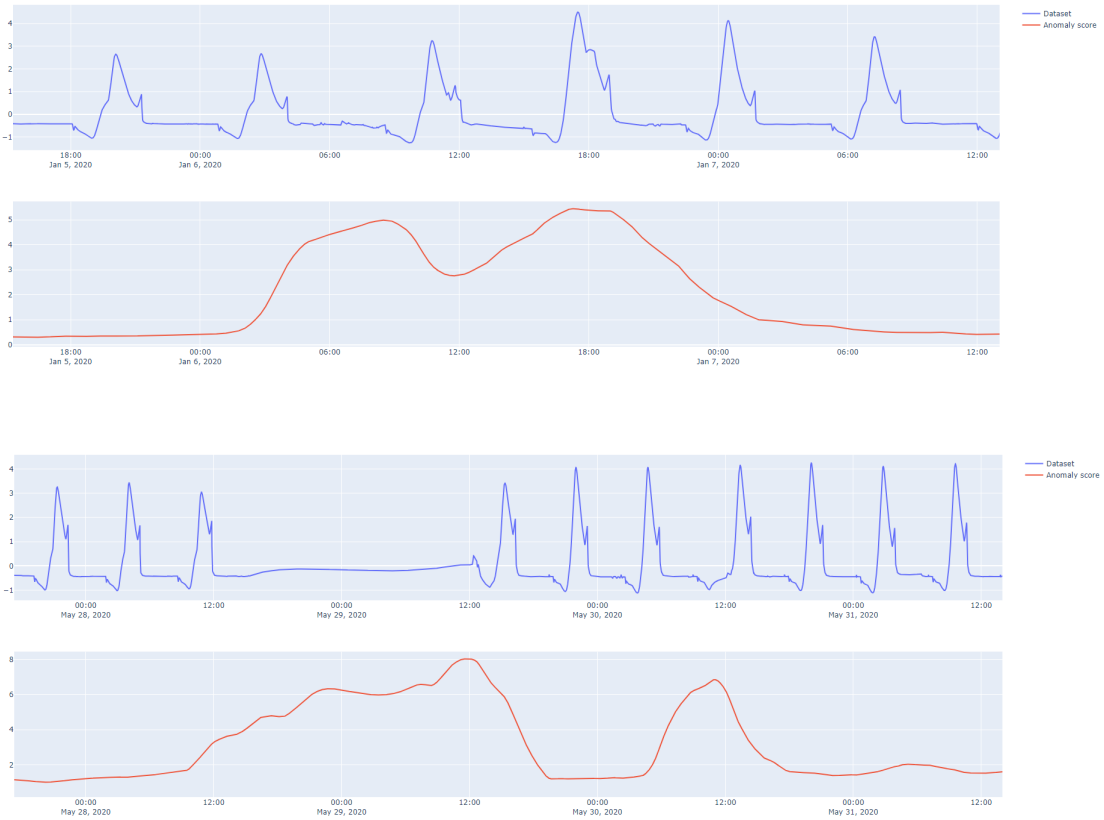

Figure B.3: Demo #1 - Test data (top) and corresponding anomaly scores (bottom) calculated using the component's default parameters for January and May (plant shutdown and process offset).

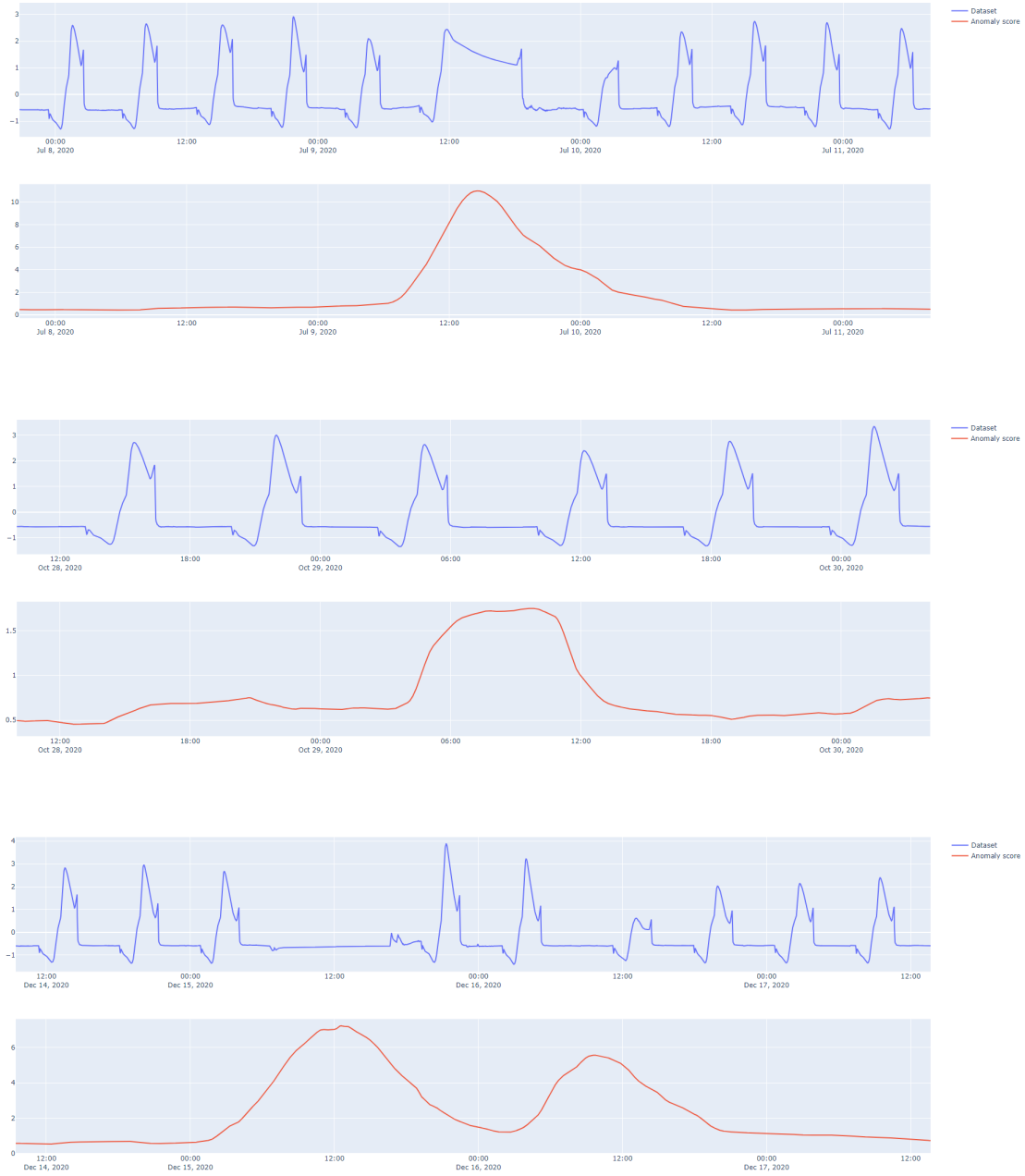

Figure B.4: Demo #2 - Test data (top) and corresponding anomaly scores (bottom) calculated using the component's default parameters for July (compressor change), October (process lag) and December.

C. Robust One Class Classification with the AnomaLip package

The AnomaLip package is a wrapper around the implementation of the robust one-class classification algorithm from Bethune et al. (2023), which was adapted to deal more easily with time-series datasets. In the following subsections we briefly recall the principles of the algorithm, and we describe at a high level of abstraction how to apply AnomaLip to a time-series anomaly detection problem. For a more technical description of the methodology please refer to the user documentation which can be found in AnomaLip Gitlab repository.

C.1. Robust one-class classification with 1-Lipschitz networks

A neural network F is said to be 1-Lipschitz if the following property is hold:

$$|F(x) - F(x')| \leq |x - x'|$$

The robust one-class classification algorithm exploits the properties of 1-Lipschitz networks (these properties are ensured by the utilization of the Deel-Lip library) to perform one-class classification by learning an adversary (and complementary) distribution of examples which it iteratively learns to classify against samples that actually come from the dataset of interest. In summary, the algorithm applies the following steps until convergence or a stop criterion is reached:

1. Generate a batch of random seeds (which will become adversarial samples)
2. Apply the Newton Raphson algorithm to *push* these samples towards the frontier of the current classifier
3. Optimize the classifier to separate a batch of real examples against the adversarial ones.

At the end of this procedure, we have two interesting properties:

- The score given by the classifier represents a distance to the classification frontier, which can be easily interpreted
- By applying the Newton Raphson algorithm on new samples without retraining the classifier, we have a generative model that can be used to provide counterfactual explanations.

C.2. Applying Anomalip to a time-series

The AnomaLip package follows a very generic anomaly detection pipeline as shown in figure C.1. From a raw time-series dataset, the only required step is to pre-process the data in order to normalize it so that it is centered around 0, which eases the optimization of the algorithm. This normalized dataset is passed to AnomaLip which takes care of further transforming the dataset by applying sliding windows, then trains the one-class classification algorithm. Once the model is trained, it can be used to score new samples. After a threshold has been selected (for example by considering the 5 percent quantile of the anomaly scores on the training dataset), we can apply it to define a set of anomalies. Furthermore, we can use AnomaLip to generate counterfactual explanations for these anomalies. These examples show how the algorithm would transform an

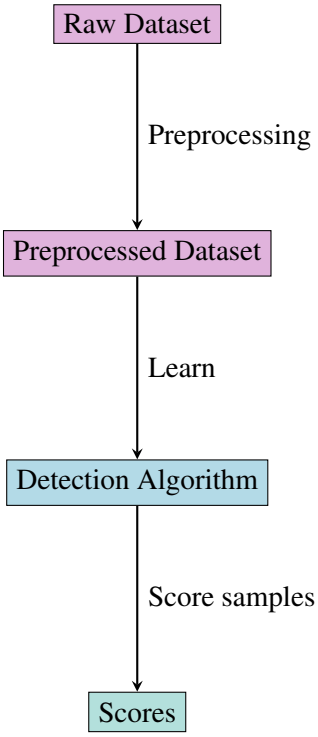

Figure C.1: High-level overview of an anomaly detection pipeline

anomaly sample so that it becomes normal. This can give valuable information during the result analysis phase, and give clues about the underlying behaviour of the analyzed time-series.

C.3. Conclusion and Recommendations

The Robust One-Class Classification algorithm was integrated into the Time-Series anomaly detection Action Sheet during Batch 3 as an exploratory activity. Following our first experiments on the AirLiquide EMS use-case, we reach the following conclusions:

- The algorithm is promising on univariate time-series. A qualitative evaluation of the results revealed the capacity to detect atypical patterns on each series. Quantitative evaluation using the available labels which where provided with the dataset are variable, but no expert analysis of the results was performed.
- The algorithm requires improvement for dealing with multi-variate series. Our first experiments applying the algorithm with multiple variables in parallel, where not concluding. We do not recommend to use the current version of the component on a multivariate problem. A combination of several univariate models may provide better results if needed.

Some improvement ideas have already emerged and may be the subject of future developpement of this component, such as better priors to generate the adversarial distribution, taking into account the time-dependent nature of the problem, or modifying slightly the architecture of the model to take into account multivariate series explicitely.

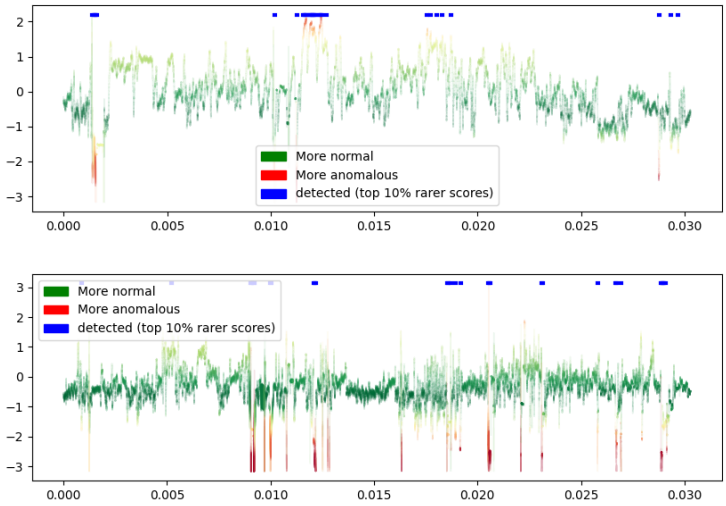

Figure C.2: Illustration of anomaly scores given by anomalip. Left on the training split (70%), right on the validation split (30%) of the System_3FIC1704A.PV signal of EMS dataset.

D. Kernel change-point detection

D.1. Context

With respect to the End-to-End Approach document, this component is concerned with sections C.7–C.13.

This component allows one to build and train a ML pipeline for the task of anomaly detection and post-hoc explainability.

D.2. Prerequisites

This component is aimed at data scientists, machine learning engineers, and technical experts (expert knowledge required to set, for instance, the maximum number of anomalies possible/probable in a given time series).

In order to properly use this component, one should be vaguely familiar with concepts such as anomaly detection, multivariate time series, statistical model selection, and kernels (the similarity function kind).

D.3. Maturity

For batch 3, this component is aiming for functional and technical maturity level 2. It has been tested on the Air Liquide Use Case.

D.4. Component overview

The kernel change-point detection component is wrapped around the Python **ruptures** package by Truong et al. (2020). In the following sections we briefly recall the principles of the algorithm, and we describe at a high level of abstraction how to apply it to a time-series anomaly detection problem. For a more technical description of the methodology please refer to the user documentation which can be found here. A demonstrator and toy data examples can also be found there.

In this component, we place ourselves in the kernel change-point detection setting introduced Harchaoui and Cappé (2007) and further developed by Garreau and Arlot (2018) and Arlot et al. (2018). This is a retrospective (i.e., offline) class of methods to detect where the data distribution changes in a (univariate or multivariate) time series; it is more general than anomaly detection—the latter can be seen as a special case.

The method works for both univariate and multivariate time series, and requires the prior choice of a kernel function, which essentially calculates *similarity measures* between data at different time points. This method outputs a set of change-points at which it believes data distribution changes occurred. The final number is determined by the model selection strategy described in

Arlot et al. (2018).

This component goes further than simply detecting change-points or anomalies. Since the data may involve thousands or more concurrent time series, detection of a change-point at time t does not necessarily help to pinpoint *which* of the individual time series were most implicated in the existence of a change-point at time t. We have therefore developed post-hoc methods to attribute individual time series to detected change-points, by ranking their "importance" and providing a visualization of this. Note that this post-hoc method and visualization only currently works for the linear kernel (not for other kernels such as the radial basis function kernel, etc.).

D.5. A very brief description of the theory

D.5.1 Kernel change-point detection

Let $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a positive semidefinite kernel, i.e., a measurable function $\mathcal{X} \times \mathcal{X} \to \mathbb{R}$ such that for any $x_1, \ldots, x_n \in \mathcal{X}$, the $n \times n$ matrix $(k(x_i, x_j))_{1 \leq i,j \leq n}$ is positive semidefinite. For $D \in \{1, \ldots, n+1\}$, we define the set of sequences of $D-1$ change-points by

$$\mathcal{T}_n^D := \left\{ (\tau_0, \ldots, \tau_D) \in \mathbb{N}^{D+1} \mid 0 = \tau_0 < \tau_1 < \ldots < \tau_D = n \right\},$$

where $\{\tau_1, \ldots, \tau_{D-1}\}$ are the change-points and τ_0 and τ_D are for notation only. We then measure the quality of any candidate segmentation $\tau \in \mathcal{T}_n^D$ of the sequence of random variables $X_1, \ldots, X_n$ with the kernel least squares criterion introduced by Harchaoui and Cappé (2007):

$$\widehat{\mathcal{R}}_n(\tau) = \frac{1}{n} \sum_{i=1}^{n} k(X_i, X_i) - \frac{1}{n} \sum_{\ell=1}^{D} \left[\frac{1}{\tau_\ell - \tau_{\ell-1}} \sum_{i=\tau_{\ell-1}+1}^{\tau_\ell} \sum_{i=\tau_{\ell-1}+1}^{\tau_\ell} k(X_i, X_j) \right]. \tag{D.1}$$

The "best" segmentation for this criterion—for a given D—is that which *minimizes* this criterion. In practice, we calculate the "best" segmentation for all values of D from 1 up to some predefined D_{max} (user knowledge required as to how many change-points one could realistically expect, currently hard-coded to be at most 50) and then run model selection to output the final "best" D (see Arlot et al. (2018)).

D.5.2 Post-hoc attribution of individual time series to detected change-points

Once the set of change-points is output using the above method, we go through this set—one by one—and output, for the change-point at time t:

- a plot showing the global criterion minimized above on the interval with left boundary the previous change-point (or 0) and with right boundary the next change-point (or end time point). The minimum of this curve is where the change-point was detected.
- superimposed onto this plot, the individual contribution of each time series to this global criterion. Since by default the linear kernel was used, the global criterion turns out to be the *sum* of the individual criteria! Thus we can visually see which individual time series are related to the detection of a change-point.
- A ranked list in terms of "importance" of the individual time series in the existence of this change-point.

Below in Figure D.1 is a plot showing what this looks like in practice.

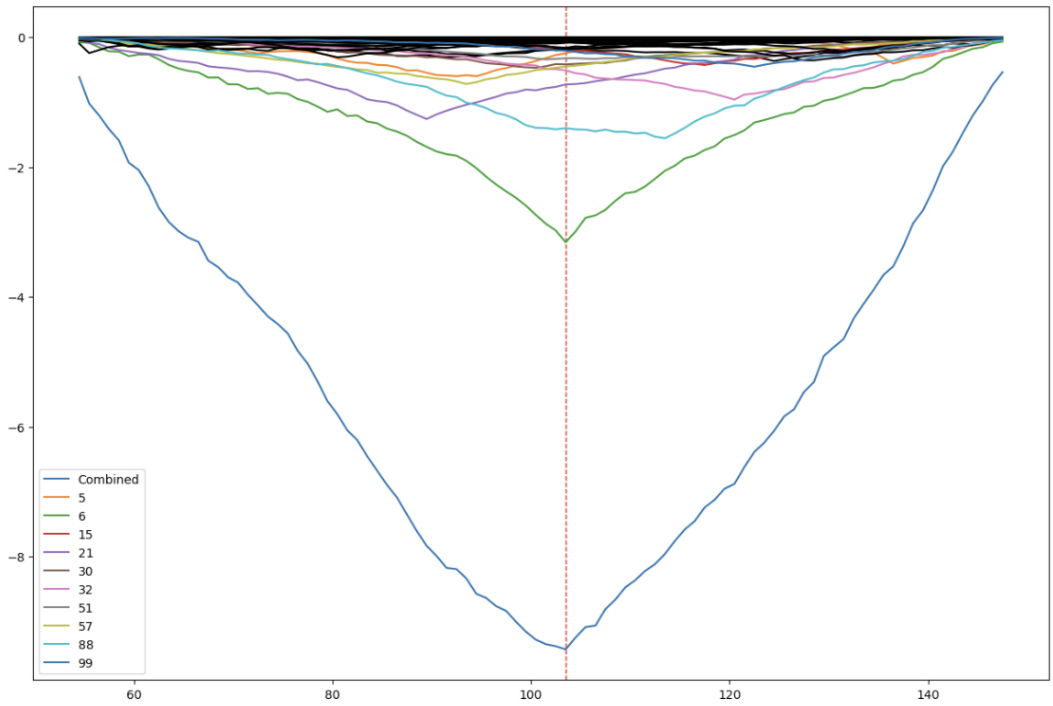

Figure D.1: Post-hoc change-point attribution to individual time series

One sees that the change-point was detected just after $t = 100$ (blue curve, minimum) and that the individual time series which contributed the most to this minimum value were the green, blue, purple, and pink ones, in that order. Note that this kind of visualization also helps us to note that out of these four time series, only the green one actually would have detected the change-point here; the others actually had minima elsewhere. Thus, the visual plots are tools which complements the ranked list of individual time series.

D.6. Prerequisites, parameters, and other details

To run this module, one requires:

- a pre-processed input array of size (number of time points) $\times$ (number of concurrent time series).
- the choice of a kernel function; however, the default is **linear** and while the change-point detection will function correctly with other kernel choices, the post-hoc steps to attribute individual time series to detected change-points will not be meaningful in the current version of this component.

D.7. Flow-chart

The kernel change-point detection module follows a generic anomaly detection and attribution pipeline as shown in Figure D.2 and as described above.

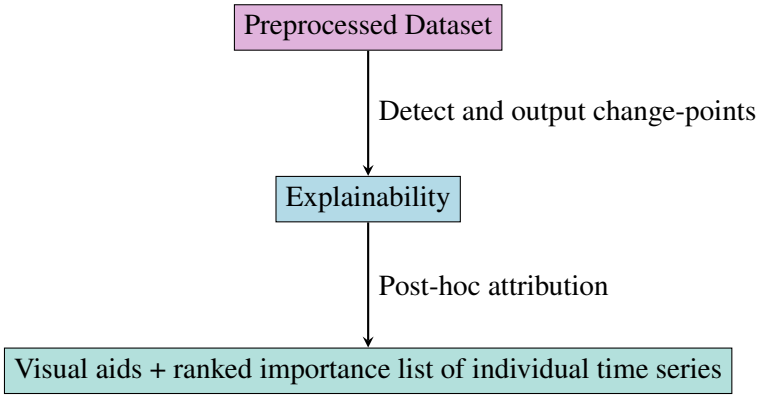

Figure D.2: High-level overview of the kernel change-point detection and attribution pipeline

D.8. Conclusion and recommendations

This component should run without difficulty on any input array of size (number of time points) $\times$ (number of concurrent time series) with no missing values. This array will probably be preprocessed elsewhere (remove or impute missing values, interpolate to fixed time grid, normalized) using expert knowledge. For the sake of simplicity, this module does no preprocessing at all.

No other parameters are *required* but various default paramaters can be modified (with expert knowledge) with help of the user documentation found here.

The choice of the kernel is by default linear, but other kernels can be chosen instead for the change-point detection step (see the user documentation linked to above). Note however that the post-hoc individual time-series attribution to detected change-points (in the multi-time-series setting) currently only makes sense if the kernel is linear. Do not use any output from the post-hoc step to make conclusions if you have not used the linear kernel.

We hope in the future to find ways to have other kernels in the post-hoc step with meaningful and intuitive outputs for attributing individual time series to detected change-points.

E. Sparsity based anomaly detection

E.1. Context of the method

This guideline fits into the context of C.7 "Data Engineering" and, more specifically, C.7.7 Evaluate data trustworthiness, according to the End-to-End Approach document.
More specifically, it concerns a method for detecting anomalies in univariate or multivariate monomodal time series, i.e. those involving the same numerical or physical quantity. It is desirable when there is little a priori regularity in the time series studied, and when these are highly non-stationary, even under normal operating conditions.

E.2. Prerequisite for using the method

Use of the method requires a moderate level of familiarity with signal processing and machine learning, particularly unsupervised learning, to understand the parameters relating to the expressiveness of the learned model and calibrate them judiciously. On the other hand, domain knowledge can guide the choice of certain parameters, such as sliding window size, but also the selection and grouping of time series to be analyzed.
As for data, time series must be regularly sampled, with the same sampling step in the multivariate case. The method does not require a label, nor does it require anomaly-free sequences for training.

E.3. Level of maturity of the method

Aimed maturity level: 3.

- being tested on Naval Group UC
- batch 2 version of the algorithm was tested on Air Liquid Efficiency Monitoring System (Data Quality) UC.

E.4. Procedure

For installation and use of the component, please refer to the user manual. The procedure involves the following steps:

1. pre-processing: variable selection, resampling, imputation of missing values, and any other transformation specific to the use case.
2. training is carried out in two stages: detection and vector representation of salient temporal events using an appropriate autoencoder; modeling of the distribution of detected temporal events. The first part of training in particular can be done from scratch or from a previous model.
3. inference: extract temporal events from a univariate or multivariate sequence to be analyzed using the model, where appropriate, assign these events to modes previously identified in the training data, calculate an anomaly score univariately or jointly.

An unsupervised variable selection method is available independently of the anomaly selection component here.

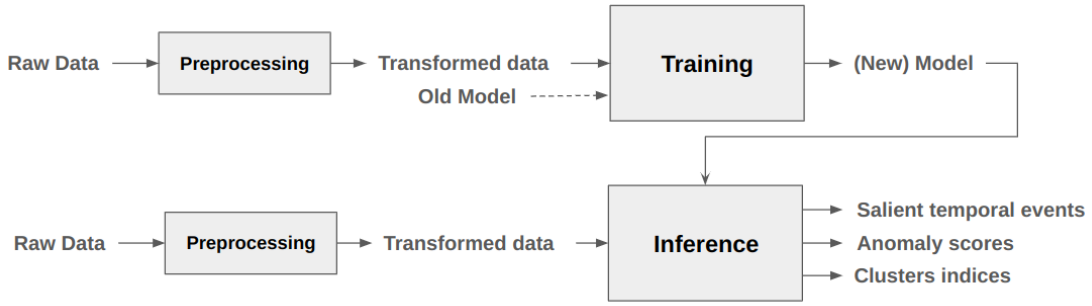

Figure E.1: A diagram overview of the Procedure

The training step is summarized in figure E.2. The aim is to build a vector representation of salient temporal events to make them comparable.
To do this, the process first involves breaking down the signal into spectral components, each with reduced temporal variability. Each spectral component is then represented as a sparse linear combination of learned sliding time patterns.
The architecture used to obtain these patterns and their sparse activation's magnitudes is illustrated in figure E.3.
At each instant, the vector of pattern activations gives the temporal structure of the signal in a window the size of the patterns and therefore provides a vector representation of the temporal events. This idea is illustrated in figure E.4 which shows, superimposed, the time series of the pattern activation magnitudes in a given signal. The events selected for the anomaly search are those whose vector embeddings have non-negligible norms (see the boxes in figure E.4).
The second part of the training involves modelling the distribution of temporal events sparse codes to derive anomaly scores as illustrated in figure E.5. This can be done using parametric models such as mixtures of Gaussian distributions, in which case the anomaly score refers to the eccentricity of each sparse code with respect to the mean of the model component to which it is assigned.
Instead, we have adopted a non-parametric approach based on optimal transport. Its principle is as follows. Optimal transport can be used to construct a mapping of an empirical distribution onto itself, or in a multivariate case, mappings of several empirical distributions onto their barycentre in the sense of optimal transport. Abnormal vectors are expected to have higher transport costs than others, given their isolation.
We observe empirically that these transport costs follow an exponential distribution whose parameter can be estimated. The corresponding cumulative distribution function provides an anomaly score. This approach has the advantage of not making any assumptions about a probability distribution whose vectors have been sampled.

E.5. Demonstrators

In this section, we illustrate how the method works using synthetic examples or examples taken from use cases.

E.5.1 Library Demonstrator

The model is trained on a non-stationary time series containing breakpoints (see figure E.6).

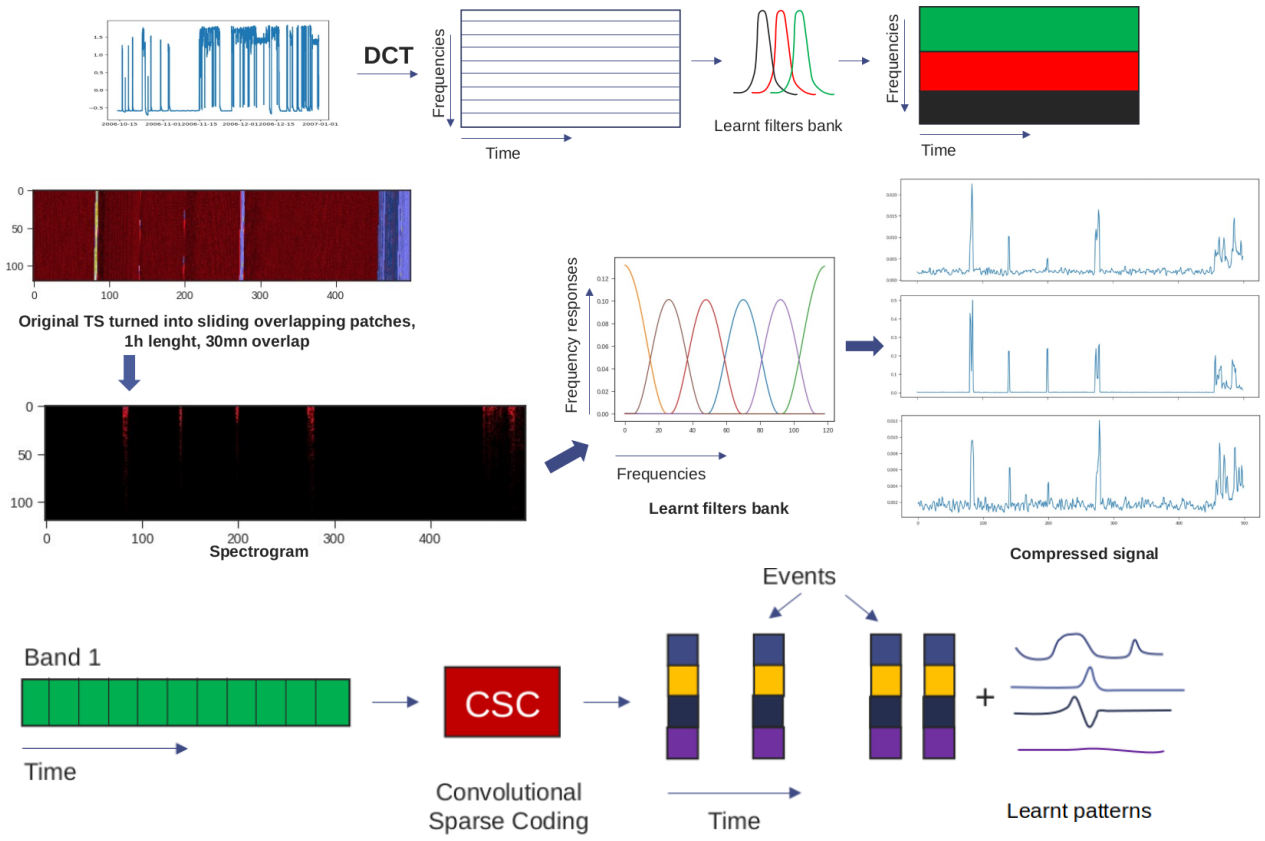

Figure E.2: Time events representation

The time series of anomaly scores obtained is shown in figure E.7 where we can see that the strongest peaks correspond to breakpoints.
By setting the detection threshold at 0.9, we obtain the orange segments in figure E.8 which contain the anomalies in the analysed signal.
The method is therefore capable of detecting anomalies in a non-stationary time series and the training is robust to the presence of anomalies in the training data.

E.5.2 Use-case level Demonstrator

This component was applied to the Naval Group use case data.
Examples of sequences with the highest anomaly scores are shown in Figures E.9.

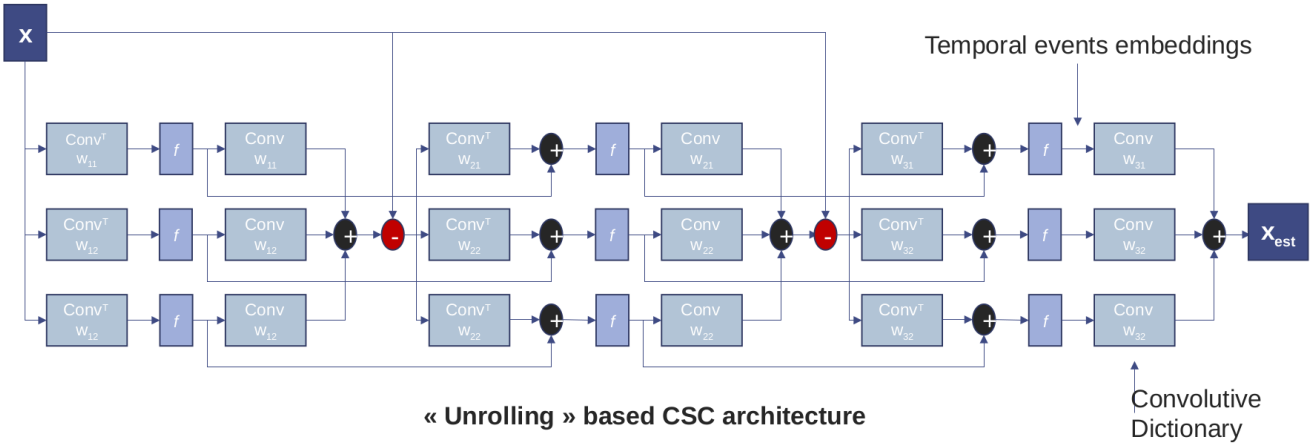

Figure E.3: Convolutive Sparse Coding architecture

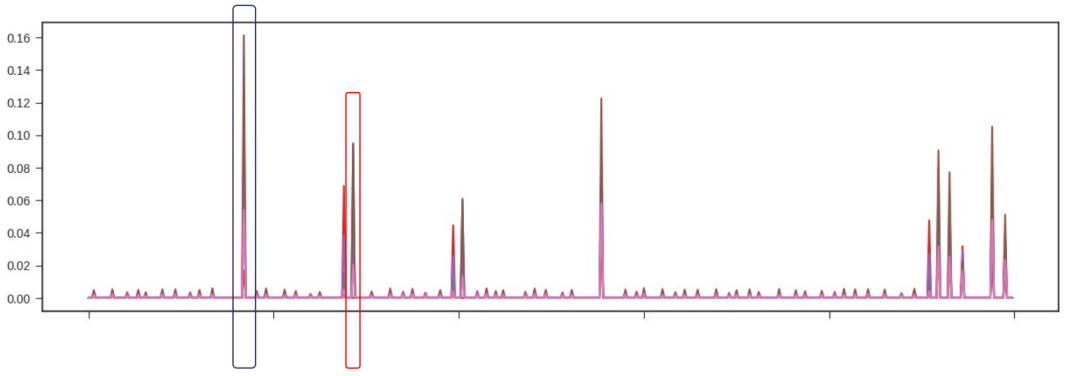

Figure E.4: Sparse Codes

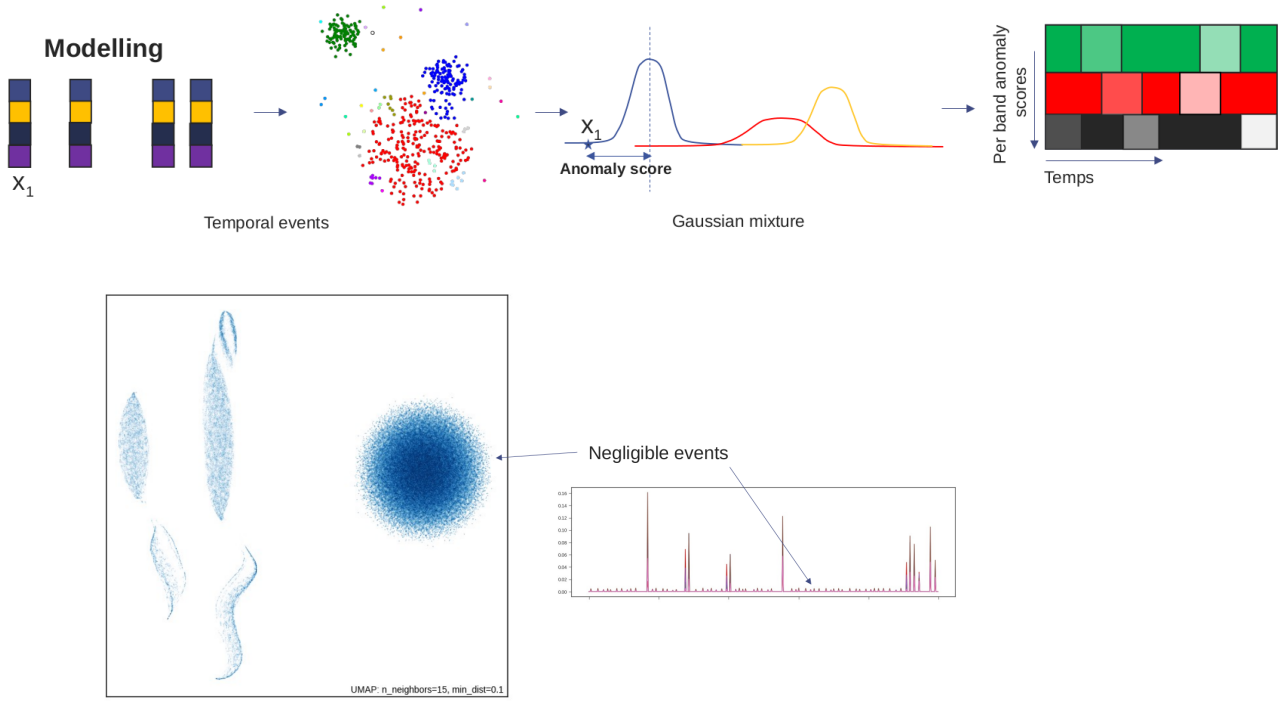

Figure E.5: Modelling sparse codes

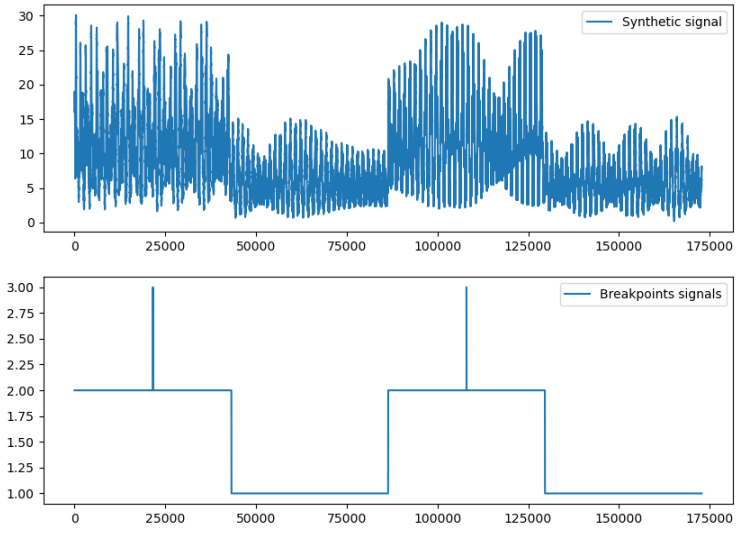

Figure E.6: Synthetic signal and breakpoints.

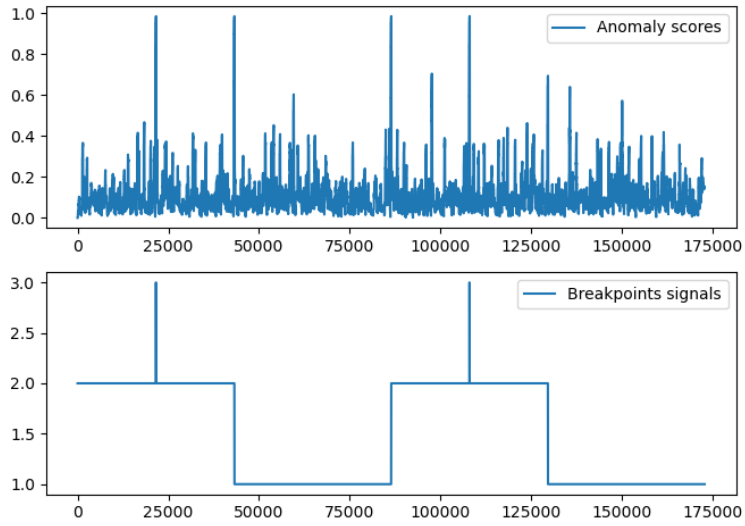

Figure E.7: Anomaly scores.

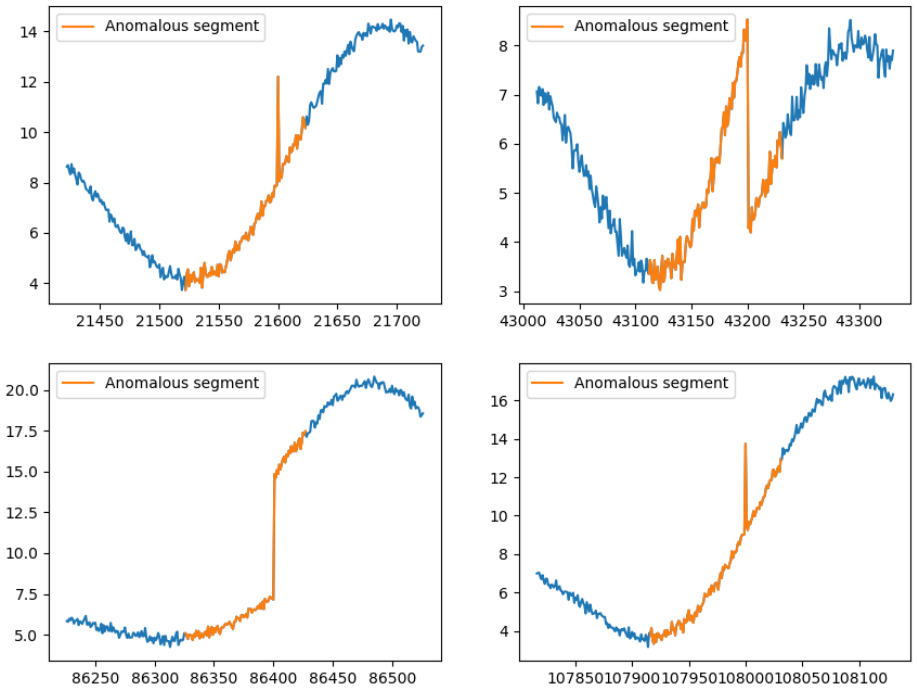

Figure E.8: Anomalous segments.

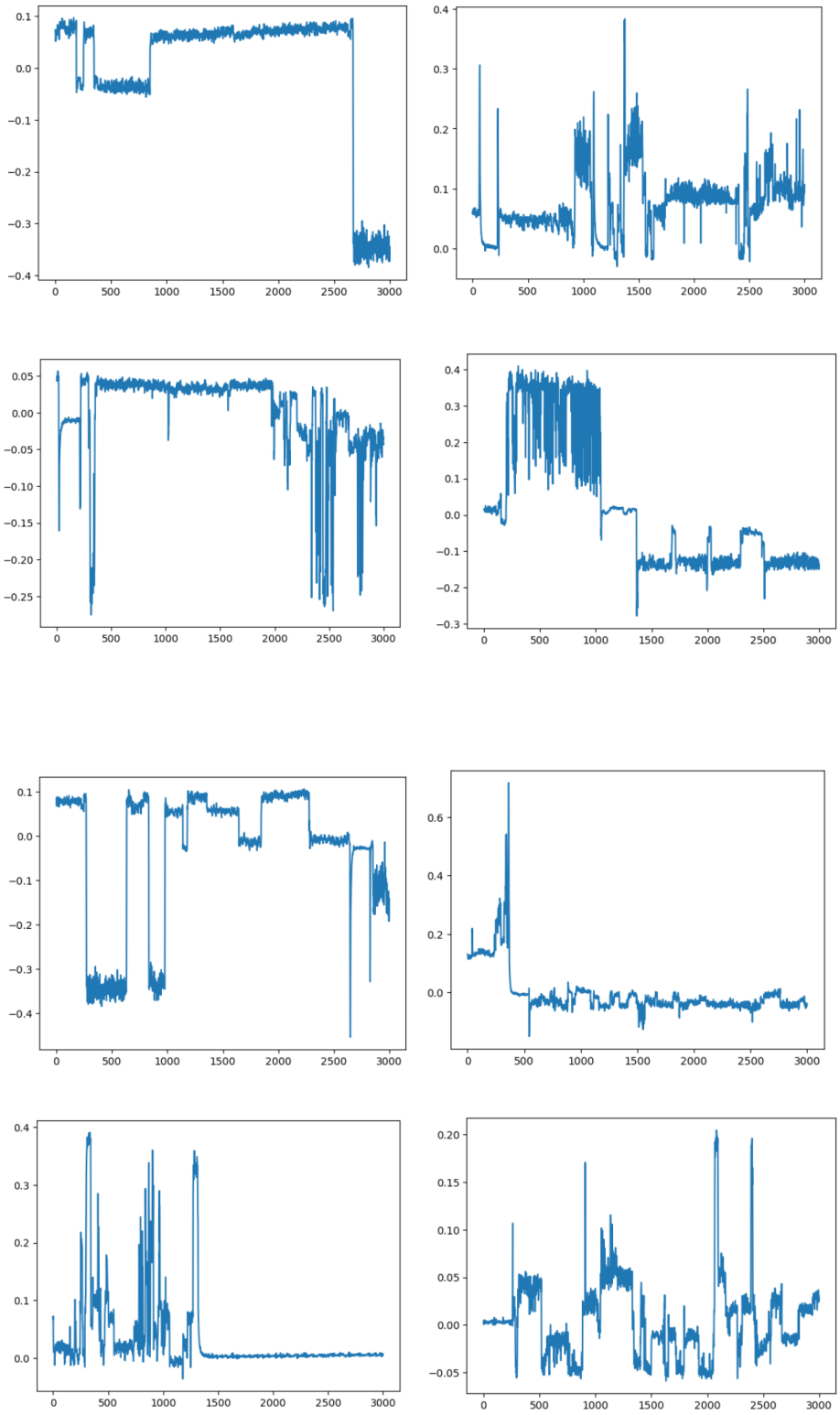

Figure E.9: Anomalous sequences from Naval Group UC.

F. UQmodels for times series – Contextual deviation analysis

F.1. Component description

"UQmodels for times series" is a library under development aiming to perform times series analysis through forecasting models with Uncertainty Quantification (UQ). One of the library's focuses is to identify the different uses of uncertainty quantification techniques within a data science chain based on machine learning/deep learning models.
This library mainly aims to provide tools and form modeling pipeline to :

- ML-Task modeling: make regression/forecast of a quantity of interest from features using ML or Deep learning.
- ML-UQ modeling: assess model decision uncertainty that may be due to irreducible variability or local model unreliability.
- Post-processing : Valorize Modeling outputs (prediction and UQ) to produce complementary KPIs :
  - UQ-KPI : At the inference step, that provide insight about margins of errors of the prevision and model reliability indicators.
  - ANOM-KPI : After real-time observation by computing anomaly score based on the difference between observation and expected values, considering uncertainty to better characterize abnormal state of the system.

Therefore, the main functionality of the library is on the modeling step of data science, through the implementation of an UQModel wrapper that can combine several implementations of forecasting models, UQestimators, and post-processor to form a mathematical processing chain. Such UQ-model can perform time series monitoring as forecasting augmented by uncertainty KPIs (as error margin or model reliability) or anomaly detection based on contextual deviation score that assesses difference between an observation and an expected value, taking into account the uncertainties of the data and predictive models.

For anomaly detection purposes, we can manipulate UQModels, using an ML or Deep learning UQestimators in charge of forecast and UQ estimation, combined with a specific post-processor in charge of producing an anomaly score from the forecast, UQEstimation, and real observation to produce a contextual deviation score.

In addition to the modeling and post-processing parts, the library also implements minor functionalities (in the form of wrappers) designed to facilitate the formalization of the pre-processing and evaluation step. The whole view of the library aim is provided F.1.

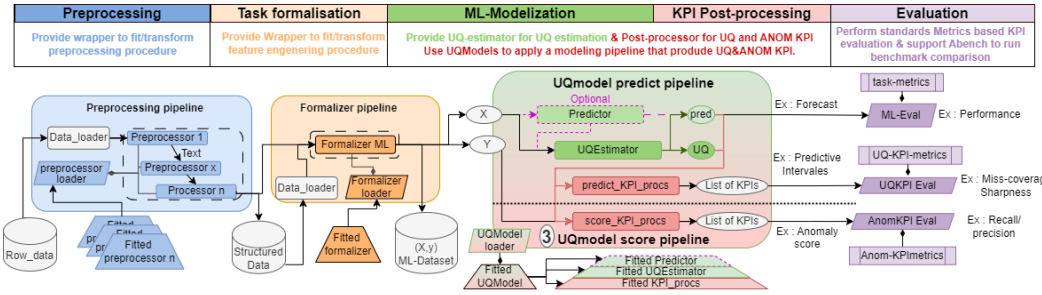

Figure F.1: Tools chain view

Please find here the repository of the component.

F.2. Context of use of the component

With respect to the End-to-End Approach, this component can be used in *Data Engineering* (section C.7):

- As a component that can be integrated into an AI-based system, the value contribution of the component lies in both "model uncertainty quantification" and "model based anomaly detection" in batch or stream processing, for example in a supervision system.
- As an engineering tool, its contribution lies also in both "model based data uncertainty quantification" and "model based anomaly detection", to feed in a data-qualification procedure some statistical analysis of data by both UQ and Anomaly indicator. Such indicators can help to identify outliers, contextual data-representativeness issues, or local abnormal noised subsets, to assess the quality of existing datasets before training Machine Learning models.

F.3. Prerequisites for using the component

The component is a library designed to be used by a data scientist. It then requires knowing how to design a processing chain using standard ML-tools as scikit-learn library, It also requires expert knowledge to translate an industrial problem into a machine learning dataset. The data preprocessing step is not covered by the library, since it requires expert knowledge of data structure, data semantics, and relevant representation according to the operational issues to solve. Then, more specifically, the data scientist has to :

- Be able to preprocess raw data collected into structured data that could be valorized through mathematical processing.
- Be able to provide a well-defined machine learning Regression/Forecast task with useful explanatory features with :
  - Targets having interest to be monitored for system state characterisation
  - Features useful for ML-modeling, which means that provide a rich data representation informing about the system state through contextual information, historical/lagged values.
- Be able to interpret the relevance of results :
  - Through qualitative analysis of results
  - By being able to apply the evaluation guidelines provided to analyze model KPI relevance.

However, the library (Although the use of specialized preprocessing libraries is recommended) can provide some naive processing, and help to execute a pre-designed processing chain (see demonstrator use cases).

F.4. Level of maturity of the component

The component is currently at a level of functional maturity: 2 and technical maturity: 2, and aims to achieve a level 3 for functional and technical maturity after an in-progress application to the Use case Naval group Anomaly detection.

We identify three main functionalities (linked to the UQ-KPIs) provided:

- For forecasting with error margins, This is a functionally mature task that has been successfully applied to public synthetics and datasets and on the Air Liquide forecasting and Air Liquide EMS Data quality use cases.
- For the construction of anomaly scores taking uncertainty into account. This is a functionally mature task that has been successfully applied to synthetic data, IRT-systemX datasets, and the Air Liquide EMS Data quality use case. In this later case, only qualitative evaluation has been done due to the lack of ground truth.
- For the KPI that qualify model reliability (OOD) at inference based on epistemic uncertainty, This is an upstream exploratory task linked to OOD detection, which presents technical and methodological difficulties and therefore has low functional maturity. However, encouraging results have been achieved on the EMS forecasting data set, and this functionality is currently being analyzed for the other two use cases.

Plus, if we consider the initiative of harmonization of preprocessing/ modeling/ and evaluation pipeline as an additional methodology, the contribution is currently being evaluated through the standardization of UQmodels demonstrators for the 3 times series use cases of confiance.AI.

F.5. Procedure for using the component

After pre-processing the data and carrying out the features-engineering, the use of Uqmodels is composed of 4 steps :

1. Firstly, we have to instantiate a UQModels object that realizes the modeling and post-processing pipeline by :
    - Optionally provide a predictor (or choose an UQEstimator that also predicts in addition to UQ estimation).
    - Choose an UQEstimator and set its parameters (or use default ones)
    - If wanted, choose predict-KPI-processors and set their parameters (or use default ones)
    - If wanted, choose score-KPI-processors and set their parameters (or use default ones)
2. Then, call the fit method to fit successively the whole modeling pipeline.
3. Then call predict (resp. score) method to recover model prediction and predict-KPIs (resp. only the score-KPIs).
4. Finally, ensure the modeling relevance by performing an evaluation step using metrics that evaluate prediction, measures, and provided KPI.

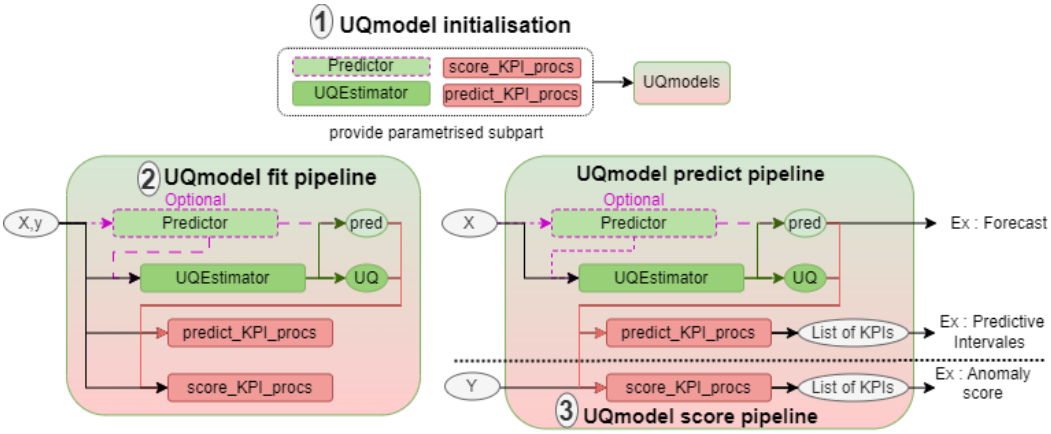

Figure F.2: Illustrated of UQModels pipeline

For advanced use, it is also possible to implement customs UQEstimators, KPI-processors, or UQmodels pipeline. This was achieved by creating specific UQmodels to treat the EMS data quality use case as a multisource problem (linked to the consideration of multiple sensors). With this in mind, the library is trying to implement reusable modular building blocks within new UQmodel pipelines.

F.6. Demonstrators

F.6.1 Library demonstrator

A basic tutorial is provided here here. It shows how to instantiate and apply a UQModel wrapper, that uses an RF-UQEstimators combined with a Gaussian-Predictive-Interval-Processor, a Model-reliability-Indicator-Processor and a Contextual-deviation-anomaly-Processor to a synthetic dataset of multivariate forecasting.

```python
# UQEstimator &  hyper-parameters
UQEstimator_initializer = RF_UQEstimator
RF =  RandomForestRegressor(min_samples_leaf=5,n_estimators=50,max_depth=20,ccp_alpha=0.0001,max_samples=0.7)
UQEstimator_parameters = {'estimator':RF,'var_min':0.002,'type_UQ':'var_A&E','rescale':True}

# PostProcesseur that compute Predictive intervals with alpha confidence lvl
PIs_proc = UQProc.NormalPIs_processor(list_alpha=[0.025,0.975])

# PostProcesseur that compute an epistemics lvl score
Elvl_proc = UQProc.Epistemicscorelvl_processor()

# PostProcesseur that compute anomaly score with theorical ratio of 1%
Anom_proc = UQProc.Anomscore_processor(Anom_parameters={'alpha':0.01})

# UQMODELs
RF_UQModel = UQModel(UQEstimator_initializer,UQEstimator_parameters,
                     name='UQModels',
                     predictor=None,
                     list_predict_KPI_processors=[UQ_proc,PIs_proc,Elvl_proc],
                     list_score_KPI_processors=[Anom_proc])

 # Application

# Learning phase
RF_UQModel.fit(X[train],y[train])
# Application in forecast
pred,(KPI_PIs,KPI-Elvl) = RF_UQModel.predict(X)
# Application in Anomaly score
KPI_anom = RF_UQModel.score(X,y)
```

Figure F.3: Code implementation of a UQModel

Such UQModel pipeline performs multivariate forecasting and provides 2 complementary KPI at the inference step: a Predictive envelope composed of 4 quantiles (sigma and 2sigma corresponding quantile) based on the UQ estimation and model reliability score that give insight about the data point where the model is the least confident. Then at the score step (after the observation), UQModel provides a contextual deviation score by the dimension that expresses the statistical abnormality according to the deviation (observation minus prevision) and the UQ measures. We observe that the score expresses a high anomaly where an anomaly was injected. As data are generated according to a gaussian law, there is also 'nominal' outliers that are not considered as abnormal by the ground truth (such as not injected) but may also have a high anomaly score.

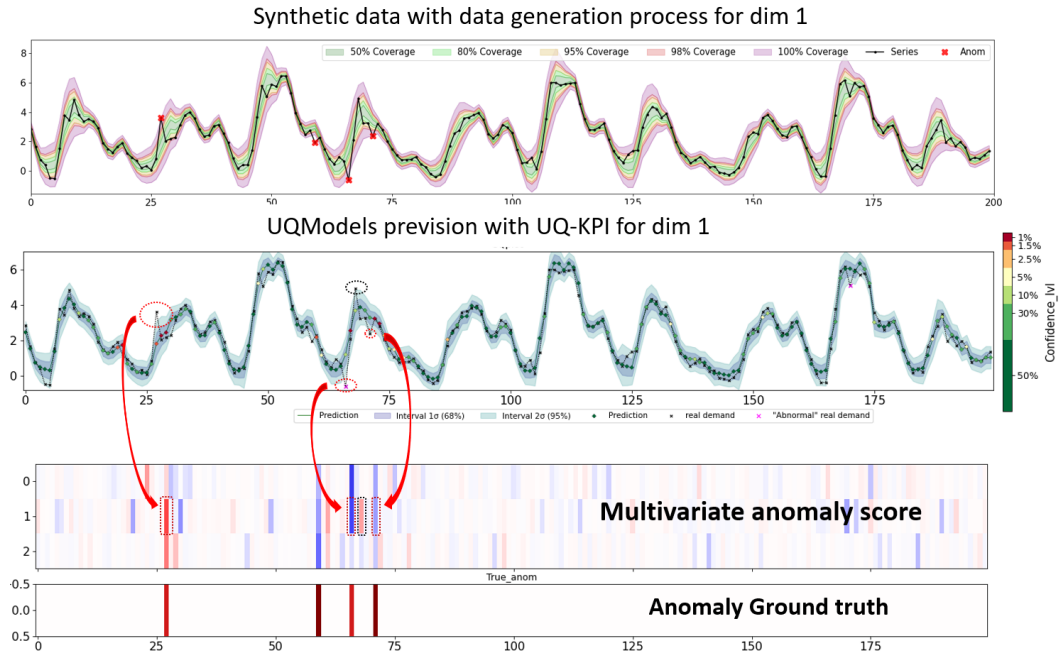

Figure F.4: Illustrated results.

F.6.2 Use-Case level demonstrator

Figure F.5 briefly summarizes what have been done for anomaly detection purpose with UQ-MOdels work:

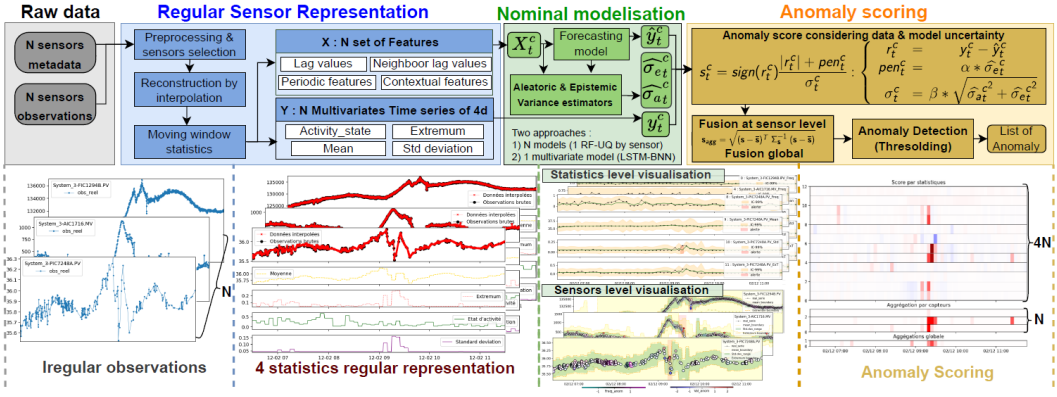

Figure F.5: Illustrated Work pipeline

There are two objectives to this demonstrator: the first one is to implement anomaly detection based on a deviation score normalized by an uncertainty measure linked to ML-modeling. The second is to produce a processing chain, from the raw data to the KPI of interest (the anomaly score), using the various wrappers designed in UQModel library.
We can therefore first distinguish a pre-processing and feature engineering stage carried out with naive processing, and representation was naively designed (due to some lack of operational knowledge) for the use case and then encapsulated in the Models wrappers as if they were code supplied outside the library.

Then the modeling stage consists of experimenting with two 2 UQModels based on two different types of Estimators (a set of random forests and a multivariate LSTM neural network) using the same post-processing to assess if the UQModel wrapper is indeed generic.
More in detail: the first part of the work will concern extracting features from an irregular time series. Naive work was set up to extract by sliding average some standard statistics in the form of a regular multivariate series, whose dimensions correspond to information channels on the irregular observation series of sensors.

The second part of the preprocessing work is related to the selection (filtering) of the sensors that seem relevant to the study, which would make it possible to decrease the dimension of the data by removing the redundant information, or sensors whose signal is too poor in information.

The third part of the work will consist of building a representation (set of features) based on lag features and various extracted information (Features engineering) to build a rich representation that allows learning algorithms to perform fine modeling.

The fourth part of the work consists of the core of the modeling aimed here, to extract a nominal behavior (mean and dynamic variance) to allow, through a contextually normalized prediction residual, to quantify the deviation of the observations to the normal behavior. Relevant anomalies can be detected by thresholds of the anomaly scores using empirical hypothesis (determined during training phase) or theoretical hypothesis (Gaussian assumption).

G. TDAAD – Topological Data Analysis for Anomaly Detection

This component is a set of methods based on the research field of Topological Data Analysis (TDA) used for the task of Anomaly Detection (AD) on time series data. Find here the component's id card and code repository.
Use these methods for the task of detecting abnormal behaviour in a dataset, a type of problem belonging to *unsupervised learning*. The methods target systems with multiple sensors/data source e.g. multiple/multivariate time series, it is not designed to handle analysis of univariate time series or other types of data (images, graphs, etc).

G.1. Context

With respect to the End-to-End Approach document, this component is concerned with sections C.7 "Data Engineering", C.8 "Development of ML Model"...
This component allows to build and train a ML pipeline for the task of anomaly detection with a certain number of hyperparameters.

G.2. Prerequisites

In order to properly use this component, one should be familiar with concepts such as ML pipelines, Unsupervised Learning, Anomaly Detection and Multivariate Time Series. Notions in TDA should help, even though this component can be understood in the broader context of *representation learning*.

G.3. Maturity

On batch 3 this component is aiming for functional and technical maturity levels 3.

G.4. Component overview

Here is a functional diagram of the tdaad anomaly detection scheme:
`tdaad` provides machine learning algorithms for analyzing timeseries data through the lense of Topological Data Analysis, and deriving anomaly scores.
The targeted input is an object `X` representing a *multiple time series* with variables columns and timestamps lines. We use the term *multiple time series* to describe a set of univariate timeseries that describe a system or object. Note that the package does not handle analysis of a single univariate timeseries.
The main idea of this package is to analyze time series with topological methods. It is done in three essential steps:

1. cut the timeseries into chunks using a sliding window algorithm,
2. represent each timeseries window with topological features,

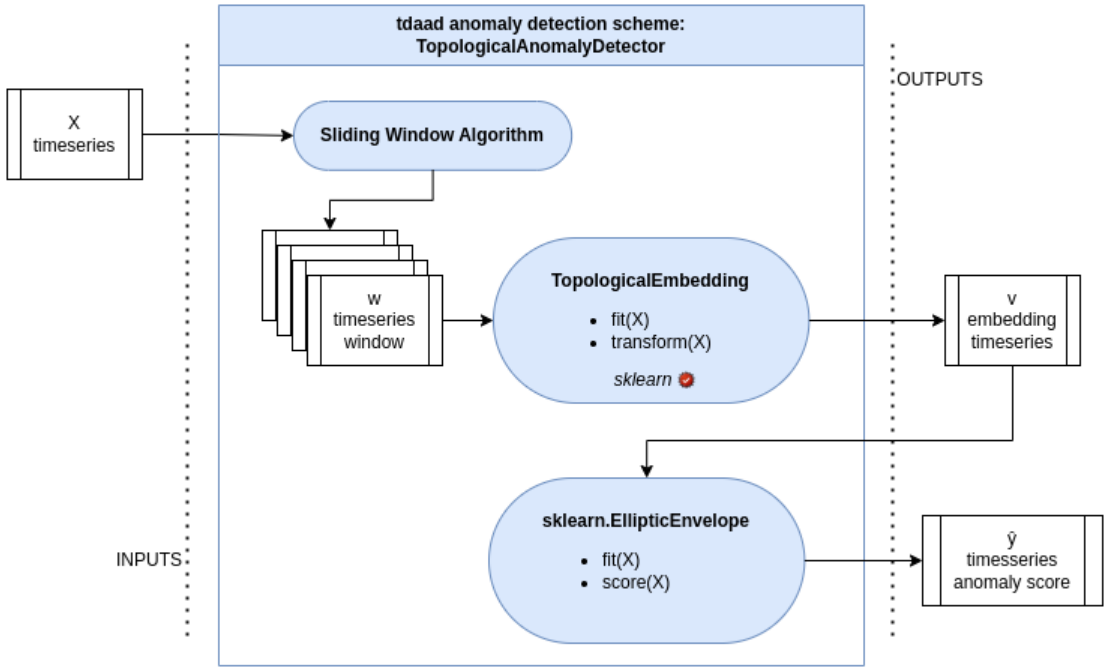

Figure G.1: Functional overview

3. estimate the empirical covariance of those topological features to derive an anomaly detection procedure.

The combination of steps 1. and 2. are performed by an object called `TopologicalEmbedding`, and the method can be understood as a representation learning method.
Step 3. is a standard step of anomaly detection procedure based on vectorized data.
The combined result is the `TopologicalAnomalyDetector` object.

G.4.1 Main features: `TopologicalEmbedding` and `TopologicalAnomalyDetector`

As the package is based upon, inspired by and compatible with well-known `scikit-learn` python library, the representation learning and anomaly detection learning are performed in the most standard way.

```
from tdaad.topological_embedding import TopologicalEmbedding
embedding = TopologicalEmbedding().fit_transform(X)
```

Based on this representation, we use an elliptic envelope algorithm: we derive an empirical mean and empirical covariance and the associated Mahalanobis distance, then use this distance as an anomaly score and derive a decision function. All of this is performed within the `TopologicalAnomalyDetector` object, so that to perform an anomaly detection process one only needs the following:

```
from tdaad.anomaly_detectors import TopologicalAnomalyDetector
detector = TopologicalAnomalyDetector().fit(X)
anomaly_scores = detector.score_samples(X)
```

G.4.2 Inputs

For now the component is designed to handle inputs `X` in the form of a DataFrame with variables as columns and timestamps as lines.

G.4.3 Main outputs

The main output is an `anomaly_score` from the `TopologicalAnomalyDetector` object, in the form of a univariate `pandas.DataFrame` with timestamps as indices.
Another key output is the embedding result `transform` of the `TopologicalEmbedding` object, that results in a multivariate `pandas.DataFrame` with timestamps as indices.
For this component's complete guidelines please refer to this page, with corresponding diagrams, activities and software description.

Part II

Anomaly Detection Calibration, Evaluation and Validation

Contents for Part II

H. Conformal Anomaly Detection

H.1. Description of the method

The method we discuss, called *Conformal Anomaly Detection* (CAD), is a statistical approach for calibrating anomaly detectors that learn from data. The benefit of such method is to provide theoretical guarantees for controlling the *False Detection Rate* (FDR) within a level of risk α (e.g. 5%) chosen by the user. In other words, CAD enables to upper-bound false alarms with a rate selected by the operator.

H.2. Context

With respect to the End-to-End Approach, this guideline fits into the two contexts, depending on how the CAD framework is used:

- Context "C.7: Data Engineering", more specifically "C.7.7: Evaluate Data trustworthiness". The component can be used for anomaly detection in order to assess the quality of a process or a dataset, ensuring trustworthiness and conformity with the system requirements.
- Context "C.8: Development of ML Model", more specifically "C.8.5: Optimization of trained ML Model". The component can be used to improve an anomaly detection model by optimizing its hyper-parameters (e.g., detection threshold).

H.2.1 Background

CAD is a post-processing approach for anomaly detectors based on the concept of *Conformal Prediction* (CP)[1]. We will not delve into the theory of CP, as its understanding is not necessary to comprehend the procedure of CAD. We refer the reader to confiance.ai reports EC3N5 and EC4N20 for further details about the theory of CP.

Consider a trained anomaly detector $\hat{f}$ that predicts if a data point X is anomalous based on an anomaly score $\hat{s}$ –or more generally any score function that evaluates the strangeness or inadequacy of a data point– such that:

$$\hat{f}(X) = \begin{cases} \text{not anomaly} & \text{if } \hat{s}(X) \leq M_{\text{threshold}} \\ \text{anomaly} & \text{if } \hat{s}(X) > M_{\text{threshold}} \end{cases} \tag{H.1}$$

where $M_{\text{threshold}}$ is the anomaly score threshold, that is the lowest anomaly scores associated to a data point to be considered anomalous. It is a critical parameter in anomaly detection, as it determines which data points are flagged as anomalies. Choosing an appropriate threshold is essential for balancing the sensitivity and specificity of the anomaly detector[2]:

[1] CP is an *Uncertainty Quantification* (UQ) framework for making predictions with guaranteed coverage. We extensively discussed this topic in previous deliverable. For further details, we recommend the reader to refer to confiance.ai reports EC3N5 and EC4N20.

[2] Sensitivity is the proportion of true anomalies that are correctly detected. Specificity is the proportion of normal data points that are correctly identified as normal.

- A lower threshold will result in a higher sensitivity, but also a lower specificity. This means that the anomaly detector will be more likely to detect true anomalies, but it will also be more likely to flag normal data points as anomalies. This can lead to a large number of false positives, which can be costly and time-consuming to investigate.
- A higher threshold will result in a lower sensitivity, but also a higher specificity. This means that the anomaly detector will be less likely to detect true anomalies, but it will also be less likely to flag normal data points as anomalies. This can lead to a large number of false negatives, which can be dangerous, as they can lead to missed opportunities or even disasters.

Setting the optimal anomaly score threshold is challenging, especially for unsupervised anomaly detection. It typically requires labeled data or expert knowledge. CAD is a distribution-free and model-agnostic approach that can calibrate the anomaly score threshold to control the *False Detection Rate* (FDR), which is the upper bound on the false alarm rate under a user-specified limit. Such control comes with a mathematical guarantee.

In other words, CAD allows users to set a threshold for how many false positives they are willing to tolerate, and then CAD will automatically adjust the anomaly score threshold to guarantee that the FDR does not exceed that limit. This is very useful for unsupervised anomaly detection, as it eliminates the need for labeled data or expert knowledge.

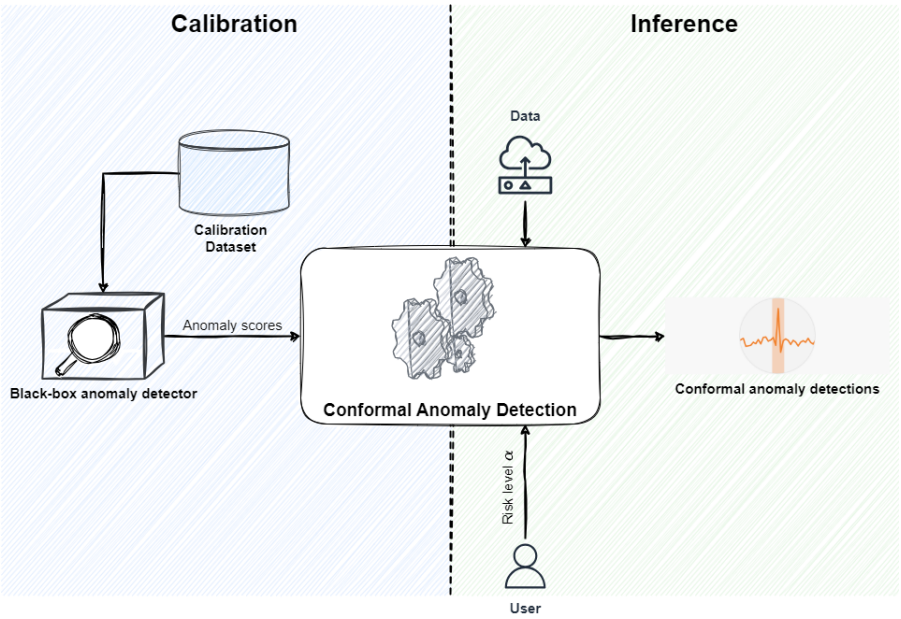

Figure H.1: Conformal Anomaly detection

H.3. Scope

CAD is a distribution-free and model-agnostic framework:

- Distribution-free means that CAD does not require any assumptions about the underlying distribution of the data.
- Model-agnostic means that CAD can calibrate any *Machine Learning* (ML) model (non-parametric, random forest, neural network, boosting, ...) from different domains (Time series, computer vision, NLP, ...).

Generally, CAD can be used with any type of data, including numerical data, categorical data, and time series data. It is particularly well-suited for detecting anomalies in complex data sets where it is difficult to define a precise model of normal behavior.

H.4. Procedure

CAD hinges on a lightweight post-processing step, which is both computationally efficient and imposes no additional requirements other than the availability of a holdout dataset. Such approach can be used with any anomaly detector that produces an anomaly score $\hat{s}(x)$ for each data point x. Additionally, we need a new dataset of n samples, called the calibration dataset, which will be used to evaluate the calibrated anomaly threshold. The calibration dataset should not have been used during the training phase. The overall procedure is illustrated in Figure H.1. The detailed steps are provided in Algorithm 1.

Algorithm 1 Conformal Anomaly Detection (CAD)

Require: score function $\hat{s}$, user-specified risk α, calibration set $X_1, \ldots, X_n$ and test point X_{n+1}.
return Anomaly indicator $\hat{\mathscr{C}}(X_{n+1}) \in \{0,1\}$.

Calibration:

for $i \leftarrow 1$ **to** n **do**
  $s_i \leftarrow \hat{s}(X_i)$
end for
$\hat{q} \leftarrow \text{quantile}\left(s_1, s_2, \ldots, s_n; \frac{\lceil (n+1)(1-\alpha) \rceil}{n}\right)$

Inference:

$s_{n+1} \leftarrow \hat{s}(X_{n+1})$
if $s_{n+1} \leq \hat{q}$ **then**
  $\hat{\mathscr{C}}(X_{n+1}) \leftarrow 0$ (inlier)
else
  $\hat{C}(X_{n+1}) \leftarrow 1$ (anomaly)
end if

CAD is a general algorithm for postprocessing anomaly scores based on the framework of CP.

H.5. Guarantees

The choice of calibration dataset is of utmost importance in CAD. The FDR control guarantee for anomaly detection is valid when the calibration examples and the test point are i.i.d.

Theorem 1 *If the calibration examples $X_1, \ldots, X_n$ and the test point X_{n+1} are i.i.d., then the probability of false alarm of the conformal anomaly detector $\hat{\mathscr{C}}$ is smaller than the user-defined risk level α (Laxhammar (2014)).*

In practice, it is sufficient for the anomaly scores estimated on the calibration and the test sets to be i.i.d to guarantee the FDR control.

There are two explanations for the cause of a conformal anomaly: 1) it may correspond to a rare or previously unseen, yet normal, example that happens with probability at most α, i.e. a false

alarm 2) If not, it is a true anomaly in the sense that it was not generated according to the same distribution as the calibration data.

In general, the calibration dataset should include only a small number of anomalous data points (anomaly ratio in the calibration dataset should be at most α). In this setting, the guarantee of controlling the FDR holds in practice. It is noteworthy to mention a rule of thumb for selecting the calibration dataset size n. Roughly speaking, Angelopoulos and Bates (2021) recommend choosing a calibration set of size n = 1000 is sufficient for most purposes. It is also important to note that the calibration dataset should not be used to train the anomaly detector. Otherwise, the anomaly detector will be overfitted to the calibration dataset and will not be able to generalize to new data accurately.

H.6. Implementation

- CAD is implemented in the open-source library **puncc**[3] under the anomaly detection module. Comprehensive documentation is available online.
- The application of CAD on benchmarks (numenta[4] and toy datasets) and on the EMS use-case are available at `https://git.irt-systemx.fr/confianceai/ec_5/ec5_as3/cad`.

H.7. Maturity

On batch 3, the component **puncc** is aiming for functional and technical maturity levels 3.

H.8. Contributions

- Implementation of CAD within the open-source library **puncc**.
- Application of CAD on top of classic anomaly detection algorithms on benchmark datasets.
- Application of CAD on top of classic anomaly detection algorithms for EMS data.
- Use of CAD to calibrate the anomaly scores produced by an anomaly detector developed in confiance.ai for EMS data. This allows to control the false discovery rate (FDR) of the anomaly detector.
- General guidelines about CAD and insights into how to use it effectively for EMS data.

H.9. Demonstrator

The demonstrator compiles CAD applications on academic benchmarks and the *Efficiency Monitoring System* (EMS) use-case. Specifically for EMS, we illustrate the calibration of several anomaly detection models on times series derived from factory sensors. Figure H.2 shows the example of conformalizing an isolation forest to reduce its false anomaly rate.

[3] `https://github.com/deel-ai/puncc`.
[4] `https://www.numenta.com/resources/htm/numenta-anomoly-benchmark/`

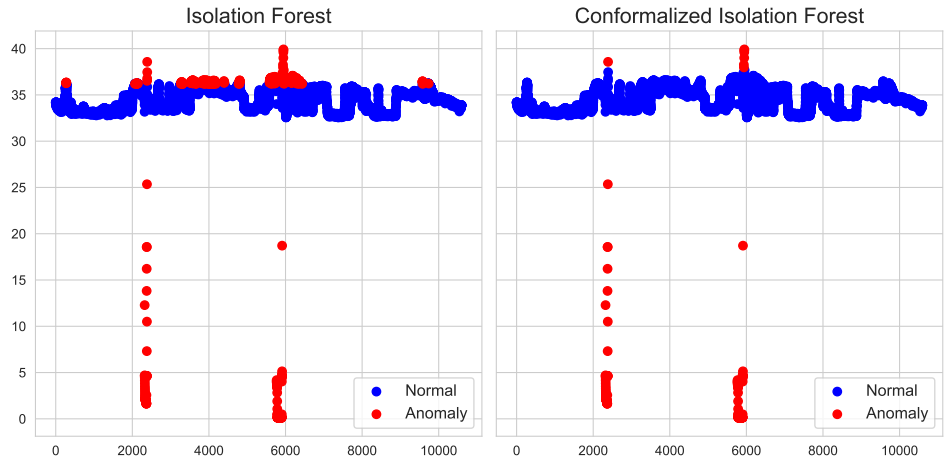

Figure H.2: Results of using CAD on an unsupervised anomaly detector. The isolation forest algorithm is trained then applied on a time series derived from a sensor within the UC Air Liquid EMS. The rate of alarms drops from 9% to 2% when calibrating the anomaly detection threshold using CAD.

The following code snippets shows how to easily wrap a blackbox anomaly detector using puncc and run the calibrated model. CAD enables to control the false alarm rate up to the user-defined value of 5%:

```
import numpy as np
from deel.puncc.anomaly_detection import SplitCAD

# Instantiate CAD on top of a pretrained anomaly detector
cad = SplitCAD(anomaly_predictor, train=False)

# Calibrate the threshold using a calibration dataset
cad.fit(z_calib=calib_dataset)

# We set the maximum false detection rate to 5%
alpha =0.05

# The method ‘predict‘ is called on the new data points
# to predict which are anomalous and which are not
cad_results =cad.predict(new_dataset, alpha=alpha)
cad_anomalies =new_dataset[cad_results]
cad_not_anomalies =new_dataset[np.invert(cad_results)]
```

H.10. Conclusion

Conformal Anomaly Detection (CAD) is a well-established and versatile anomaly detection method that is both theoretically sound and practically useful. It is free from the drawbacks of many other anomaly detection methods, such as the need for subjective anomaly threshold tuning and the assumption of a specific data distribution.
The primary advantage of employing CAD lies in its ability to fine-tune the anomaly score threshold, allowing for precise control over the *False Detection Rate* (FDR) up to a user-specified

threshold α. This process hinges on a lightweight post-processing step, which is both computationally efficient and imposes no additional requirements other than the availability of a calibration dataset. This particular attribute holds significant importance within the industrial context as it effectively mitigates the burden of incessant false alarms, thereby conserving the time and energy of system operators at a minimal cost.

Nonetheless, it is essential to recognize that an exclusive focus on minimizing the FDR may inadvertently lead to a different problem: an increase in false negatives. Failing to detect an anomaly can result in substantial financial losses or other forms of damage. Consequently, it is essential for future research to focus on achieving a balance between minimizing false alarms and reducing false negatives, aiming to strike a Pareto-efficient trade-off that optimizes both aspects.

I. EMMV – Excess-Mass and Mass-Volume Curves

This component is a set of two methods to evaluate the performance of an anomaly detection model without requiring labeled data.
Classical methods to assess the performance of the anomaly detection algorithm, such as confusion matrix, F1 score, precision, recall, or AUROC or AUPR, depend on the availability of labeled data. It is a problem to assess the performance of such an algorithm when ground truth is scarce or unavailable.
Excess-Mass (EM) and Mass-Volume (MV) curves are used to evaluate the performance of unsupervised anomaly detection algorithms without the need for labeled data. The curves are particularly applicable in unsupervised intrusion detection systems and are based on the assumption that anomalies occur in low probability regions of the data generating process. The EM and MV curves provide an alternative means to compare the performance of unsupervised anomaly detection algorithms when labeled data are scarce or unavailable.

I.1. Context

With respect to the End-to-End Approach document, this component is concerned with section C.9 “Evaluation of ML Model”.
This component allows to evaluate a ML model for anomaly detection using only unlabeled data.

I.2. Prerequisites

In order to properly use this component, one should be familiar with concepts such as Unsupervised Learning, Anomaly Detection and Multivariate Time Series.

I.3. Maturity

On batch 3 this component is aiming for functional and technical maturity levels 1.

I.4. Component overview

The component provides a means to evaluate the performance of unsupervised anomaly detection algorithms by assessing their ability to identify and rank anomalies in the data.
In more formal terms, given a time series with d features, we define an anomaly detector as a **scoring function** $s : \mathbb{R}^d -> \mathbb{R}$, such that the smaller s(x), the more abnormal x.
Under the assumption that anomalies are rare events, the goal of the scoring function s is therefore to evaluate the "distance" between the event x, and the "mean" of normal events. For this we use the **density level set f**, that is a region in the data space where the probability density function takes values above a certain threshold. It is a solution of a minimization problem that seeks to find a set with the minimum volume such that the probability of the set is greater than or equal to a specified level. In simpler terms, it represents regions in the data space where the density is relatively high, and it is used to identify areas with low probability, which are likely to contain anomalies.

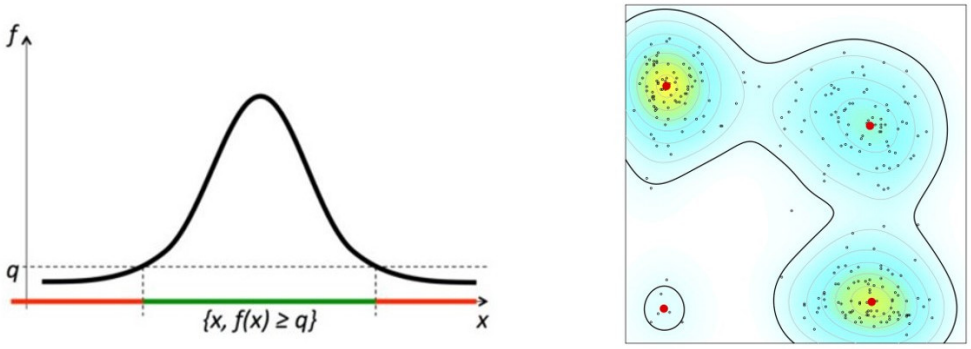

Figure I.1: Detecting an anomaly <=> estimating the density level set f. How do we do that when we don't know the distribution F ?

In other words, s is good if the preorder it induces is "close" to that of the density level set f of the dataset. How can we know that distance without knowing the distribution F of normal events (which we don't know since we don't have ground truth) ?
The key here is that s does not need to approximate f. It only needs to induce the same preorder. In other words, s is a good scoring function if: $s(x_1) > s(x_2) <=> f(x_1) > f(x_2)$
The goal therefore becomes to define a criterion C of proximity to the true density f. There exist two such criteria that have been proven effective to evaluate anomaly-detection algorithms:

- **Mass-Volume (MV) curve**: It provides a means to assess the performance of anomaly detection algorithms by plotting the function that maps the mass to the volume. This curve helps in visualizing how the volume of extreme regions in the data varies with the mass of these regions, thus aiding in the evaluation of the algorithm's ability to identify anomalies in the data. It represents the mass of the anomalies detected versus the volume of the region in which the anomalies are found i.e. measure of the density of the data distribution.
- **Excess-Mass (EM) Curve**: The EM curve is another criterion used for evaluating the performance of anomaly detection algorithms. It represents the excess-mass of the anomalies detected, i.e. the difference between the mass of the anomalies and the mass that would be expected if the anomalies were randomly distributed, versus the fraction of the total volume that the anomalies occupy i.e. measure of their spatial concentration.

The point is that the mass-volume and the excess-mass curves cannot be know without knowing the distribution F of normal events. However **they can be estimated** using known, unlabeled samples of the time series.

I.5. Usage

The component has been developed in python to calculate the MV and EM curves. It has been successfully tested on four open-license time series use-cases. It computes an objective score for the anomaly detection algorithm, and is very useful to rank two competing algorithms: if $MV_{s_1} > MV_(s_2)$, then s_1 is better at detecting anomalies that s_2 on the given (unlabeled) dataset. In order to use it, you need a time series anomaly detection dataset, not necessarily labeled (that is the point of the method), and two classifiers. Running the classifiers and the dataset through the EMMV method produces two sets of scores: two MV scores (one for classifier s_1, one for classifier s_2), and two EM scores. The scores should be consistent, meaning $MV_{s_1} >$

$MV_{(s_2)} \iff EM_{s_1} > EM_{(s_2)}$. This ordering allows one to compare two classifiers on a same unlabeled dataset. The absolute value of the scores (if only one classifier is available) also gives some indications on the quality of the classification.

J. Validation & test procedure for anomaly detectors

J.1. Introduction

This document presents and highlights various approaches for anomaly detection. We here approach anomaly detection from the validation point of view: how can we validate methods designed to find the unexpected?

We propose for this chapter to introduce the Open Set Recognition (OSR) formalism in section J.2 and to highlight the importance and the role of anomaly detection in view of this formalism. Section J.3 details the possible approaches to use for selecting the most appropriate anomaly detector when considering a given applicative usecase with its constraints and to estimate its performances in use. Section J.4 focuses more in depth in a category of metrics, namely internal metrics, and stresses its particular importance. Conlusion remarks in Section **??** complete the chapter.

J.2. Open Set Recognition formalism & Anomaly Detection

Open Set Recognition (OSR) is a context in machine learning where the input data in operation mode differs from the training Geng et al. (2020). It makes the distinction between:

- The data available at training time, which comes from a data acquisition campaign and can be formalized as the *measured space*. Considering the data from the classes point of view, OSR makes the distinction between:
  - Known Know Classes (KKC) which are the classes of interest (cats and dogs for a classification model targeting animals for instance), also referred as the *positive* classes.
  - Known Unknown Classes (KUC) which are classes of lesser interest from the applicative point of view, but still relevant to include in the model because they are part of the variety the model will encounter. It can be a *background* class, or an *other* class in the case for the animal recognition model. Those classes can also be referred as the *negative* classes.
- The data the model will face when being in use in the 'real world', at inference time. This dataset can be formalized as the *open space*. From the classes point of view, KKC and KUC are present in the open space. However the open space also contains two types of classes, both of which non representated in the measured space.
  - Unknown Known Classes (UKC) which are expected classes but data are not available at training time. For instance other types of animals.
  - Unknown Unknown Classes (UUC) which are completely unexpected classes.

Working under the open set assumption assumes a distinction between the measured space and the open space. On the contrary, the closed set presumes that the measured space is similar to the open space, and is fully representative of the real world diversity. Implicitly, most of AI models

are on a closed set assumption, leading to a difficulty when deploying the models for operational use. The following of this Section presents how anomaly detection can be used conjointly to classification models to work efficiently under an open set assumption. It is illustrated in Figure J.1

To tackle a classification problem under an open set assumption, two models can be used jointly:

- Supervised learning can be used to train the classic classification model M_c. This model is trained on a given number of classes N_C of the training set D_{train}. At inference time, it has the ability to distinguish one class from another.
- Semi-supervised learning, also known as one-class classifiers, can be used to train an anomaly detection model M_{AD}. This model is trained on one class only: the *nominal* or *normal* class. The training dataset D_{train} remains the same, the labels however are disregarded and replaced by *normal*. At inference time, M_{AD} has the ability to distinguish data between nominal mode, and an anomaly. If the data is normal, its class can be obtained with relative confidence by the classification model M_c. If it is detected as an anomaly, the classification results of M_C should be disregarded.

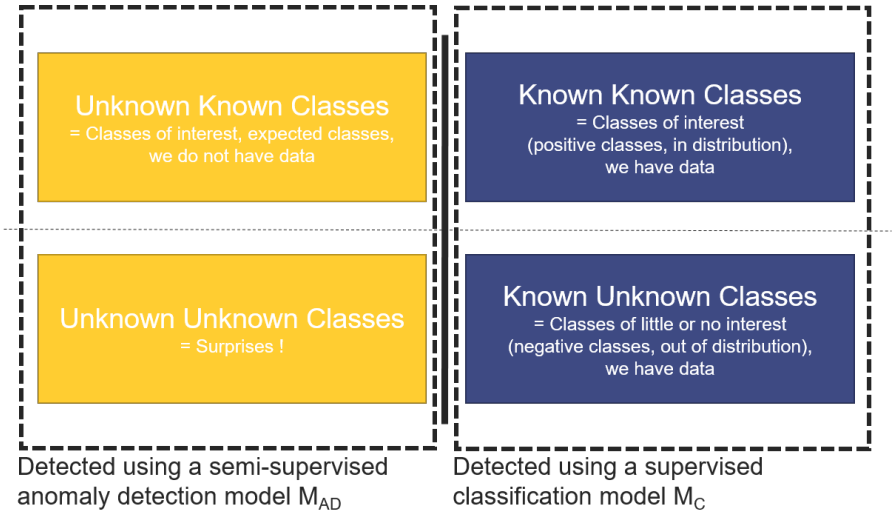

Figure J.1: Illustration of OSR context and the place of both classification and anomaly detection models.

J.3. Validation & test procedures: focus on anomaly detectors

Traditionally, the classification model M_c is trained as follow:

- Training using the dataset D_{train} and a supervised learning approach.
- Validation, a.k.a. selection of the set of hyper-parameters leading to the best model is done using the dataset $D_{validation}$ and an appropriate metric.
- The evaluation of the model capacities is done during the test step, using the dataset D_{test} and the same evaluation metric. D_{test} is used only once, when the final model has already been selected. The use of different subsets $D_{validation}$ and D_{test} ensures that the capacities of M_c are not over-estimated and limits the risk of over-fitting.

Typical metrics for the validation of classification models include the accuracy, or the balanced accuracy which is preferred if the dataset is imbalanced. Class by class precision and recall can also be used for a finer analysis, along side with confusion matrix. Such metrics are computed using labels, and can be referred to as *external metrics*.

For the anomaly detection model M_{AD}, the process is as follow:

- Training using the dataset D_{train} and a semi-supervised learning approach.
- Validation is much more complex, since it cannot be done classically on $D_{validate}$ using the traditional metrics: anomalies are not present in $D_{validate}$
- Similarly, the evaluation step is as complex: D_{test} does not contain any anomalies.

Anomaly detection is a binary classification problem where only on class is known at training, hence the use of semi-supervised or one-class classifier approaches. Model selection and model estimation cannot be performed in a similar fashion than M_C. The following of this chapter aims at summarizing the various options for both validation (model selection) and test (model performance evaluation) steps of an anomaly detection model trained using semi-supervised learning. In particular, two questions must be addressed: on which data? and with which metrics?
We address the issue depending on the data availability distinguishing two cases: with or without additional data in which some anomalies (UKC and/or UUC) could be present. We underline the fact that anomalies are both UKC (expected) and UUC (unexpected): validation and test procedures must select and measure the model M_{AD} ability to detect both categories.

With the existing datasets $D_{validation}$ and D_{test}

- If the full external metrics cannot be computed or both selection and testing of M_{AD}, the can be partially measured using the nominal data and labels present in $D_{validation}$ and D_{test}. In particular, the precision can be computed. This approach is computable for both $D_{validation}$ and D_{test} and gives information on the considered models. It is however not advised to only consider this approach for model selection for it would lead to an extremely biased model selection. We would advice to always compute such metrics, at least for informational purposes.
- Another set of metrics can be introduced and used: internal metrics, which are fully detailed in section J.4. In this section we simply summarize them as a set of metrics measuring some properties of a model, without the use of any label. A more extended view on this approach, including the specific properties they reflect, and the dataset on which they can and should be computed in presented in the specific section.
- A completely different approach is to perform model selection and testing on a proxy task, for which both steps can be evaluated. This approach is commonly used and based on the assumption that satisfactory performances on the proxy task lead to good capacities on the Anomaly Detection task. If easily computable, we would also advice checking those metrics, but keeping in mind their limitation.
- Finally, another common approach is to build a fake anomaly detection scenario from the known classes KKC and KUC: one or several known classes are arbitrarily chosen to be considered as anomalies, or fake anomalies can be artificially built. In this scenario, the considered classes become UKC and can be considered as part of a $D_{validationOpen}$. Processes applicable in this case are detailed in the more general of anomaly detection model selection and test when additional data including anomalies is available and detailed in the next paragraph. One specificity of this approach however, is its limitation to models used conjointly to classification models. They cannot be used if one tries to build an

Anomaly Detection model to be used by itself (as it the case with the AirLiquide usecase).

With additional data: $D_{validationOpen}$ and $D_{testOpen}$:

In addition to computing various metrics on $D_{validation}$ and D_{test} and if the applicative usecase allows it, further steps can be considered for M_{AD} selection and test. In particular, two extra data-sets of validation and testing would be used, both being more representative of the "real world". Practically speaking, such data-sets could be referred as $D_{validationOpen}$ and $D_{testOpen}$ and would come from a different data acquisition campaign, or a different period of time. Anomalies would be present in those data-sets. Depending on the availability of such data (regardless of the presence or absence of labels), various validation and testing procedures can be implemented. Before addressing those extra validation and test scenario, we draw the reader's attention to the label question in $D_{validationOpen}$ and $D_{testOpen}$. The most common case would be to have unlabeled datasets. In some specific cases however, some labels can be obtained: especially in the "fake scenario" previously exposed. We highlight than in thoses cases, such anomalies would be considered as UKC. M_{AD} ability to detect them is to be considered, but it should not be assumed as an ability to detect UUC which are, by definition, not expected.

- If labels of the UKC are available, external metrics can be classically computed for both the validation and test of M_{AD} ability to detect the UKC. If computable, we would advice to take this metrics into consideration, along side with a complementary approach for dealing with the UUC.
- Without any labeling of $D_{validationOpen}$ and $D_{testOpen}$, the use of internal metric is advised for model selection (see SectionJ.4 for further detail). Contrary to the scenario using only $D_{validation}$ and D_{test}, using the additional datasets will allow a computation of the full set of internal metrics. Such metrics can also be used for testing the model with the limitation that they give additionnal information on the model, but do not estimate the anomaly detection binary classification performances.
- To overcome this limitation, one approach - applicable - is to consult an expert on the applicative usecase for review of both raised anomalies, and raised nominal data. A posteriori review of such data gives an estimate of the anomaly detection binary classification performances. We highlight the importance of reviewing both anomalies and nominal data to allow the computation of the full external metrics. If the review time is to important, a common approach is to review only a subset of the prediction.

The detail of internal metrics, the properties they evaluate and which can be computed depending on the availability of $D_{validationOpen}$ and $D_{testOpen}$ is reviewed in the next section. A schematic summarizing the validation and test process for both M_C and M_{AD} is proposed in Figure J.2.

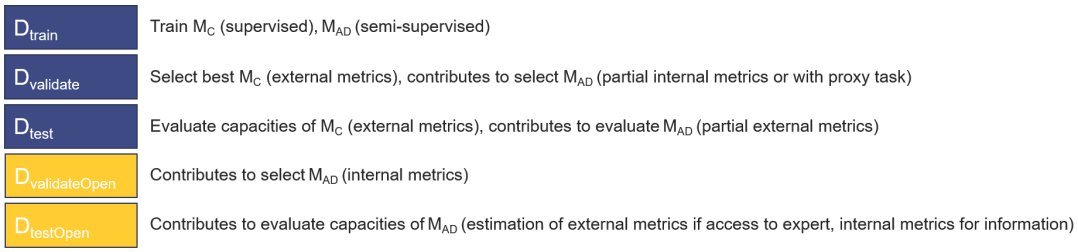

Figure J.2: Illustration of OSR context and the place of both classification and anomaly detection models.

J.4. Focus on internal metrics

Internal metrics are metrics computed to measure some aspects of a given model, without any labels. They were traditionally related the evaluation of unsupervised learning. We here summarize the main families of internal metrics present in the literature.

Consistency between models The general idea behind this set of metric is to compare one model in comparison with the other models being tested. Ma et al. (2021) presents them with the following question: *"Which model comes out as the main prediction tendency over all model tested?"*. If many models are tested, those metrics assume that the model following the global tendency will give the best result. Model giving isolated predictions would be less reliable. Those metrics are impacted by the other models being tested, which can be a drawback.

- Unsupervised Disentangled Ranking (UDR) Duan et al. (2019) compares predictions among models with similar hyper-parameters but different seeds.
- Model centrality (MC) Wan et al. (2021) compares prediction of various models using a uniform representation to define a global centroid.
- MC-HITS and MC supported by unsupervised outlier model ensemble are similar to classical MC but differ by the centroid positioning which is determined by iterative procedure. It aims at avoiding perturbations caused by singular inconsistent predictions.

Such metrics can be computed on $D_{validation}$ but with limited interest since anomalies are not present. They are more relevant with $D_{validationOpen}$, keeping in mind the limitation that they are impacted by the set of models considered (measure impacted by the experimental protocol).

Intrinsic properties of the models Such metrics are targeting properties of the models themselves and are therefore independent of the other models being considered. In particular, those metrics focus on compactness of the clusters, their separability, or a combination of both. The main metrics focusing on the compactness include is Root-Mean-Square Standard Deviation *RMSSD* Halkidi et al. (2001); Talia and Trunfio (2012) which represents the clusters compactness based on the summation of the standard deviation of each cluster and in each dimension. The value is normalized by the total effective number of point. The smaller the STD measure, the better the clustering result.
Separability of the model is usually measured using one of the two following metrics. R-squared *RS* Mickey (1997) compares the rate of difference between overall dispersion of the dataset points and the averaged dispersion within all clusters. The closer *RS* is to 1, the higher the dispersion. Modified Hubert statistic *H* also focuses only on the separation quality of the clustering by comparing two matrix, one of them representing the proximity of the point in all the data set and the second one represent the proximity inside each cluster. The bigger is H, the best is the clustering result.
Finally, a relatively large number of metrics combining both compactness and separability can be used. We here give a short description of those metrics:

- Dunn index *DI* which computes the ratio between the smallest distance between data of different clusters and the largest distance of two points in one single cluster among the number clusters detected Dunn (1974). A good clustering result will lead to a high value of Dunn Index.
- Xie-Beni score *XB* which is a ratio between the level of compaction of the data within the same cluster and an estimation the separation of the data from different clusters Xie and Beni (1991). The optimal cluster number is reached with the minimum *XB*.

- Clustering Validation index based on Nearest Neighbors *CVNN* which evaluates the inter-cluster separation based on points that govern the cluster frontier geometry Liu et al. (2013). The separability measure is here governed by the peripheral points in each cluster: the denser the frontier, the more important it becomes in the inter-cluster separation measurement. Points in the center of the cluster do not contributes to the separation metrics. Compactness is based on the average distance between points in the same cluster. The combination takes a form of the summation of the inter-cluster separation and the intra-cluster compactness after the normalization for both of them.
- Index I *I* is a ratio between separation measure and compactness measure Maulik and Bandyopadhyay (2002). Separation is evaluated as the maximum distance between clusters center. Compactness is computed as a sum of distance from points to their cluster centers in each clusters. The higher the I index, the better the clustering result.
- SD index *SD* Halkidi et al. (2000) represents the sum the average scattering and the total separation of clusters.
- Calinski-Harabasz coefficient *CH* is also known as a "variance ratio criterion" Caliński and Harabasz (1974). It was originally designed for identification of the optimal number of clusters of a cloud of point in a multi-dimensional Eucledian space. The optimal model is the one that maximize CH value.
- Silhouette index *Si* represents the clustering partition based on the dissimilarity of each instance to its cluster's instances and to its "neighbors cluster's" instances Talia and Trunfio (2012); Rousseeuw (1987). The higher the silhouette, the better the clustering. The normalization by the number of point in each cluster is introduce to keep significance of all clusters even with imbalanced configuration.
- Davies Bouldin index *DB* Davies and Bouldin (1979) measures the average of cluster's similarities. A combinaison of cluster i and j included among the number of clusters established are used to compute the ratio between internal compaction of both clusters i/j divided by the distance between the two clusters. A low DB measure indicate a good clustering result.
- IREOS also combines separability and compacity within a same metric but using an Support Vector Machine model to transform the input data Marques et al. (2020).
- Mass-volume and volume excess metrics express can also be used and are detailed in another chapter of this document.

Those metrics are computable from the compactness perspective with $D_{validation}$ and are therefore a valid approach for M_{AD} selection. For both separability and compacity estimation, $D_{validationOpen}$ is required.

Model complexity This last family of approaches is a way to evaluate the complexity of the considered model M_{AD}. A substitute model is trained with supervised learning to reproduce the the behavior M_{AD} :it is trained from $D_{validateOpen}$ and the associate labels produced by M_{AD}. External evaluation of of the proxy model is used to estimate how simple or complex is M_{AD}. The idea behind this family of models is to favor simple models, which generalize better (Okham's razor). Similarly, such metrics can be computed from $D_{validation}$ but with a limited impact. They are more representative if a $D_{validationOpen}$ is available.

J.5. Conclusion

We proposed in this chapter to look at anomaly detection algorithms with the Open Set Recognition formalism. It includes a distinction between expected anomalies (UKC) and unexpected anomalies (UUC). While selecting an anomaly detection model, it is capital to take both types into consideration. Several scenarios are presented to perform an optimal selection of anomaly detectors, with a highlight on the role of internal metrics to detect UUC.

K. Conclusion

In this methodological guideline document, we presented several tools for time series anomaly detection, as well as some methods for calibrating, evaluating and validating anomaly detectors. To aid the reader in navigating between all these chapters to find the best method that would be suitable to his problem, we briefly recap the main features of each method at its current development state at the end of batch 3.

K.1. Anomaly Detection Tools

1 – CNNDRAD
- Author: Olivier Antoni, CEA Grenoble
- Keywords: 1D-CNN, pretext tasks (data reconstruction)
- Dimensionnality: Univariate
- Framework: Self-supervised
- Anomalies: Segments
- Prerequisites: Sliding window size, regularly sampled time series, train data with very few anomalies
- Limitations: Train data should be representative of the operating regime for which anomalies will be detected in the test data

2 – Anomalip
- Author: Andrés Troya-Galvis, Thales Alenia Space
- Keywords: 1-Lipschitz, one-class classification
- Dimensionnality: Univariate
- Framework: Unsupervised
- Anomalies: Points
- Prerequisites: Zero-centered data normalization
- Limitations: Multivariate application

3 – Kernels
- Author: Kevin Bleakley, INRIA Saclay
- Keywords: Change-points detection
- Dimensionnality: Univariate, Multivariate
- Framework: Unsupervised, post-hoc interpretability
- Anomalies: Points
- Prerequisites: Pre-processed input array (number of time points) × (number of concurrent time series), choice of kernel function
- Limitations: Post-hoc interpretability is only possible if the kernel choice is linear

4 – Sparsity based
- Author: Fred Ngole Mboula, CEA Saclay
- Keywords: Multiscale patterns learning

- Dimensionnality: Univariate, Multivariate, Unimodal
- Framework: Unsupervised
- Anomalies: Points, segments
- Prerequisites: Sliding window size, regularly sampled time series
- Limitations: slow sensors drifts can not be detected.

5 – UQModels
- Author: Kevin Pasini, IRT SystemX
- Keywords: Contextual deviation analysis
- Dimensionnality: Multivariate, Multimodal
- Framework: Unsupervised
- Anomalies: Points, segments
- Prerequisites: Raw data preprocessing, Regression/Forecasting task definition
- Limitations:

6 – TDAAD
- Author: Martin Royer, IRT SystemX et Datashape, INRIA
- Keywords: Topological Data Analysis
- Dimensionnality: Multivariate
- Framework: Unsupervised
- Anomalies: Points, segments
- Prerequisites:
- Outputs: data embedding, sample score.
- Limitations: for d timeseries with n timestamps, computation in roughly $O(d^2 \times n)$.

K.2. Anomaly Detection Calibration, Evaluation and Validation

7 – CAD
- Author: Mouhcine Mendil, IRT Saint Exupéry
- Keywords: Conformal Prediction, Anomaly detection calibration, False Detection Rate, Distribution-free, Model-agnostic
- Prerequisites: Anomaly scores, Calibration dataset
- Limitations: Possible increase in false negatives

8 – EMMV
- Author: Christophe Gouguenheim, Thales Alenia Space
- Keywords: Excess-Mass and Mass-Volume Curves, Unsupervised anomaly detection evaluation, Little to no labeled data
- Prerequisites: Anomaly detector, two classifiers
- Limitations:

9 – OSR
- Author: Marielle Malfante, CEA Grenoble
- Keywords: Open Set Recognition, Validation & test procedure for anomaly detection
- Prerequisites: Semi-supervised anomaly detector for Unknown Known and Unknown Unknown classes, Supervised classifier for Known Known and Known Unknown classes

- Limitations: Getting acquainted with several scenarios of possible applications of this procedure

Bibliography

Angelopoulos, A. N. and Bates, S. (2021). A gentle introduction to conformal prediction and distribution-free uncertainty quantification. *arXiv preprint arXiv:2107.07511*.

Arlot, S., Celisse, A., and Harchaoui, Z. (2018). A kernel multiple change-point algorithm via model selection. *Journal of machine learning research*, 20(162).

Bethune, L., Novello, P., Boissin, T., Coiffier, G., Serrurier, M., Vincenot, Q., and Troya-Galvis, A. (2023). Robust one-class classification with signed distance function using 1-lipschitz neural networks.

Caliński, T. and Harabasz, J. (1974). A dendrite method for cluster analysis. *Communications in Statistics-theory and Methods*, 3(1):1–27.

Davies, D. L. and Bouldin, D. W. (1979). A cluster separation measure. *IEEE transactions on pattern analysis and machine intelligence*, (2):224–227.

Duan, S., Matthey, L., Saraiva, A., Watters, N., Burgess, C. P., Lerchner, A., and Higgins, I. (2019). Unsupervised model selection for variational disentangled representation learning. *arXiv preprint arXiv:1905.12614*.

Dunn, J. C. (1974). Well-separated clusters and optimal fuzzy partitions. *Journal of cybernetics*, 4(1):95–104.

Garreau, D. and Arlot, S. (2018). Consistent change-point detection with kernels. *Electronic Journal of Statistics*, 12(2).

Geng, C., Huang, S.-j., and Chen, S. (2020). Recent advances in open set recognition: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 43(10):3614–3631.

Halkidi, M., Batistakis, Y., and Vazirgiannis, M. (2001). On clustering validation techniques. *Journal of intelligent information systems*, 17:107–145.

Halkidi, M., Vazirgiannis, M., and Batistakis, Y. (2000). Quality scheme assessment in the clustering process. In *Principles of Data Mining and Knowledge Discovery: 4th European Conference, PKDD 2000 Lyon, France, September 13–16, 2000 Proceedings 4*, pages 265–276. Springer.

Harchaoui, Z. and Cappé, O. (2007). Retrospective mutiple change-point estimation with kernels. In *2007 IEEE/SP 14th Workshop on Statistical Signal Processing*, pages 768–772. IEEE.

Laxhammar, R. (2014). *Conformal anomaly detection: Detecting abnormal trajectories in surveillance applications*. PhD thesis, University of Skövde.

Liu, Y., Li, Z., Xiong, H., Gao, X., Wu, J., and Wu, S. (2013). Understanding and enhancement of internal clustering validation measures. *IEEE transactions on cybernetics*, 43(3):982–994.

Ma, M. Q., Zhao, Y., Zhang, X., and Akoglu, L. (2021). A large-scale study on unsupervised outlier model selection: Do internal strategies suffice? *arXiv preprint arXiv:2104.01422*.

Marques, H. O., Campello, R. J., Sander, J., and Zimek, A. (2020). Internal evaluation of unsupervised outlier detection. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 14(4):1–42.

Maulik, U. and Bandyopadhyay, S. (2002). Performance evaluation of some clustering algorithms and validity indices. *IEEE Transactions on pattern analysis and machine intelligence*, 24(12):1650–1654.

Mickey, R. M. (1997). Applied multivariate techniques. *Journal of the American Statistical Association*, 92(437):384.

Rousseeuw, P. J. (1987). Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65.

Talia, D. and Trunfio, P. (2012). *Service-oriented distributed knowledge discovery*. CRC Press.

Truong, C., Oudre, L., and Vayatis, N. (2020). Selective review of offline change point detection methods. *Signal Processing*, 167:107299.

Wan, Z., Mahajan, Y., Kang, B. W., Moore, T. J., and Cho, J.-H. (2021). A survey on centrality metrics and their network resilience analysis. *IEEE Access*, 9:104773–104819.

Xie, X. L. and Beni, G. (1991). A validity measure for fuzzy clustering. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 13(08):841–847.

Confiance ai

Title: Methodological Guideline for Time Series Anomaly Detection

Keywords: Time Series, Anomaly Detection, Data Quality

This methodological guideline presents several anomaly detection tools for time series, which have been developed and integrated within the Confiance.ai consortium. The aim of this document is to aid the reader in briefly understanding the possible usage of each component in order to direct the application depending on the use case at hand. The document is then completed with some methods for calibration, evaluation and validation of anomaly detectors, which were also explored to tackle objectives such as adjusting the False Detection Rate or having little to no labelled information in the dataset.

Our partners:

Air Liquide AIRBUS Atos cea Inria
NAVAL GROUP GROUPE RENAULT SAINT EXUPÉRY SAFRAN sopra steria
SystemX THALES Valeo