A benchmark for context-dependent parametric anomaly detection in industrial time series

Mazen Alamir

A benchmark for context-dependent parametric anomaly detection in industrial time series

M. Alamir*

*Univ. Grenoble Alpes, CNRS, Grenoble INP, GIPSA-lab, 38000 Grenoble, France.
Emails: mazen.alamir@grenoble-inp.fr

***Abstract*—This paper introduces a publicly available dataset that can be used as a benchmark for anomaly detection in industrial time series and provides some initial set of performance results that can serve as a reference for future comparisons. The benchmark focuses on parametric uncertainties that are detectable only in specific sets of context while the context-related information is not present in the data. This strongly emulates the industrial real-life situations where equipments are used in many different non documented contexts which makes the normality characterization a quite difficult task. The targeted algorithms to be evaluated via the benchmark are those based on a one-class training on anomaly-free datasets where no instances of the anomalies are present which is again a commonly encountered situation in industry.**

***Index Terms*—Parametric Anomaly Detection; Industrial Time Series; Context-Dependent Anomalies; One-Class Methods.**

CONTENTS

I. INTRODUCTION

Anomaly detection is a hot topic in industry as it determines crucial issues such as machines availability, fatal break-down risk avoidance and optimization of maintenance operations. The last years witnessed a burst in Machine (and Deep) Learning-based attempts to address this problem in the specific industrial context [1], [2] which is characterized by 1) the rarity of recordings involving faulty instances 2) the difficulty to cover all possible future instances of anomalies and more importantly, 3) the fact that normality characterization heavily depends on the context of use which is not always documented in the datasets making cluster-based normality design quite difficult.

The first two specific features makes one-class training the only realistic option. Namely, normality should be characterized using only healthy samples of time series provided by the sensors. Based on the normality characterization and associated thresholds that are designed based on a given false alarm rate, the algorithm can be used to assess the health status for all newly coming streams of sensor values.

Nevertheless, the question that remains to answer is related to the a priori assessment of any anomaly detection algorithm in the specific case of industrial time series, namely:

RELEVANCE OF BENCHMARKS

> Given a *normality characterization* and an associated anomaly detection algorithm, how to assess the ability of the latter to detect unknown anomalies within the specific realm of (hidden context)-dependent industrial data as described above?

In many references and available benchmarks, the faulty instances are introduced by inserting slices of totally different classes of signals or by superimposing noise or time-dependent disturbances on the top of the raw healthy sensors recordings. This induces non physically justified (and hence rather unrealistic) instances of anomalies which might undermines the relevance of the conclusions/ranking that might be drawn from the evaluation outcome. The price of such irrelevance of benchmarks is the promotion of algorithms that are specifically designed and tailored to detect such unrealistic anomalies while being potentially quite bad in addressing the real-life problems of detecting anomalies in a specific use-case such as the industrial one.

In order for a benchmark to be compatible with industrial time series, it should incorporate some features that the latter exhibit. The benchmark introduced in the present paper incorporates the following features:

SPECIFICITY OF INDUSTRIAL TIME SERIES

1. Industrial time series obey some underlying (although possibly unknown) generally sparse relationships. These relationships represent physical laws[a].
2. These relationships might be context-dependent. Moreover, the information regarding the context is commonly absent in the learning data which prevent building context classifiers.
3. Many of equipment's anomalies materialize through changes in the parameters that are involved in the governing parsimonious relationships invoked in the first item above.
4. The representation of the different contexts in the training data is not balanced.

[a]The benchmark involves piece-wise polynomial relationships which are eligible to represent a wide range of situations if not all.

It is barely necessary to justify the **first** of the previously stated properties. Indeed, industrial equipments are engineered based on, at least qualitative, knowledge-based understanding of the principles that govern their behaviour. But physical laws are parsimonious in that they can be expressed through relationships involving a generally moderate number of coefficients[1].

As for the **second** of the three features invoked above, namely the fact that the relationships are context-dependent, a simple example is the relationship governing the longitudinal dynamic of a car involving the accelerator pedal's position, the acceleration and the traction force. Obviously, this relationship depends on the gear ratio which represents here the context-defining variable that might be absent from the recordings. More precisely, depending on the gear ratio, the relationship that represents the normal behaviour of this subsystem strongly differs. Another example is the relationships governing the braking system where the context might be represented by the road properties (snowy, wet, dry, slippy, hot, cold, etc.) which is sometimes not explicitly recorded for use in the training phase where the normality characterization is to be built.

As for the **third** feature, one can think of defaults representing abnormal friction values due to the deterioration of the quality of surfaces in contact, the defaults in a hydraulic valve that induces detuned relationships between the pressure's gradient and the resulting flow rate or a bias on some sensor that makes a physically consistent relationship no more valid to cite but few common examples. All these examples of faults can be *modelled* by the same nominal relationships with slightly *detuned* set of parameters. Consequently, the corresponding shift in parameters can be a physically consistent instance of parametric anomalies in industrial time series.

Finally, the unbalanced presence of the different contexts in datasets is very common as the data collection periods always correspond to limited time over which there is no reason for the different contexts to be present. It can even be quite unrealistic to ask for such balanced representation to be present in the working data as the operational partner who is responsible for the data collection task might be unaware of what we might call context within the anomaly detection realm in the first place.

The KAGGLE's benchmark dataset [3] made available by the author is presented in this paper and some performance results are shown which are obtained by a recently introduced parsimonious normality characterization algorithm [4]. Indeed, the dataset has been designed to meet the specificity of industrial time series as discussed above. The performance results provided hereafter can be used as baseline for future investigation regarding normality characterizations that would be specifically dedicated to industrial time series.

This paper is organised as follows. First of all the benchmark dataset is described in Section II together with the statement of the anomaly detection problem. Section III discusses the algorithm used to produce the baseline detection results and clearly defines the metrics used in evaluating their performances. The results are shown and discussed in Section IV before Section V summarizes the paper's contributions and give some hints for further investigations.

II. BENCHMARK'S DATASET AND PROBLEM STATEMENT

The publicly available KAGGLE dataset [3] provides 30 instances of the anomaly detection problem. Each instance of the problem is represented in a pickled[2] dataframe named

$$\texttt{pb\_i.pkl} \text{ for } i \in \{1, \ldots, 30\}.$$

The number of sensors involved in each instances as well as the number of different contexts are depicted in Fig. 1 where it can be seen that the number of sensors ranges from 2 to 12 while the number of different contexts is almost always equal to 4 except for two instances.

While a full information regarding the way the dataset is generated is not mandatory for its practical use as a benchmark dataset, it is worth to precisely state how the dataset has been generated for an informed evaluation of its specificity and hence a deeper understanding of the ability of different algorithms to detect the parametric anomalies it contains.

A. Some details regarding the generation of the dataset

The python package that served in the creation of the benchmark is made freely available [5] via the standard `pypi` package management repository. Therefore, it can be easily installed by the following command:

```
pip install pwp_bench
```

Moreover, a detailed description of the module can be found at the author's GITHUB public repository:

`https://mazenalamir.github.io/pwp_bench/`

[1]As an example, an industrial 4-axis manipulator robot can be described by less than a hundred of parameters. In spite of this, DNN models that are frequently used in deriving normality characterization might involve hundreds of thousands of active coefficients.

[2]A python format for serializing data content.

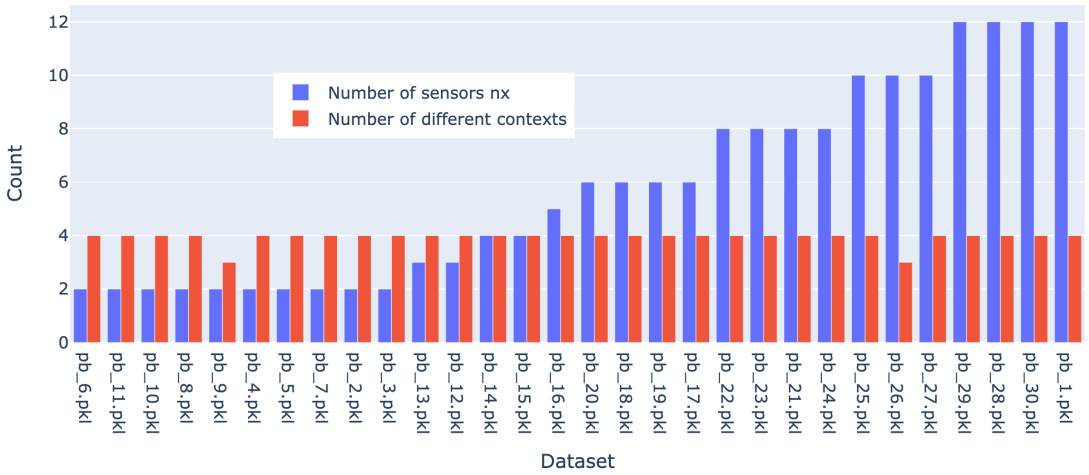

Fig. 1. The number of sensors (features) and the number of different contexts in each of the individual dataset inside the KAGGLE benchmark [3].

	x1	x2	x3	x4	x5	x6	idz	y	res	ytrue
0	-1.114	1.370	-1.363	-0.743	1.317	-1.079	1	-0.362	0.000	-0.362
1	1.275	0.374	-1.453	1.436	-1.422	0.491	1	-0.307	0.000	-0.307
2	-1.407	-0.657	-1.477	1.198	0.759	-0.774	1	-2.553	0.000	-2.553
3	-1.356	0.038	1.401	0.203	-1.069	0.122	1	-0.419	0.000	-0.419
4	-1.151	-0.494	-1.261	0.137	0.430	-1.107	1	-1.977	0.000	-1.977
5	1.052	0.775	-0.725	0.572	-1.489	1.350	1	-0.716	0.000	-0.716
6	1.124	-0.558	-1.431	-1.416	-1.389	1.284	1	-2.430	0.000	-2.430
7	1.366	0.830	1.426	-0.555	0.885	-1.223	1	-1.435	0.000	-1.435
8	-0.886	-0.388	1.288	0.517	-1.221	0.996	1	0.574	0.000	0.574
9	-1.249	-0.801	-1.285	0.444	0.672	-1.340	1	-2.986	0.000	-2.986
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
19984	-0.950	0.873	-1.035	-0.773	-0.621	0.728	3	1.437	0.050	1.387
19985	0.898	0.370	-1.246	0.710	0.838	0.078	3	0.077	0.354	0.432
19986	-0.677	-0.761	0.003	0.558	0.031	-1.354	3	-2.213	0.248	-1.966
19987	0.519	-0.606	0.217	0.009	-0.109	0.210	3	-0.022	0.011	-0.033
19988	-1.211	-1.419	-1.086	0.224	0.928	0.675	3	3.186	0.006	3.192
19989	-0.547	1.096	1.448	0.493	1.308	0.726	3	1.954	1.767	0.187
19990	-1.187	-0.365	0.141	0.668	-1.019	-0.688	3	-0.244	0.045	-0.200
19991	-1.232	0.555	-0.379	-0.425	-0.323	-0.925	3	-0.559	0.183	-0.376

Fig. 2. Truncated view of the first and the last rows of the dataframe contained in `pb_19.pkl`. Features are represented by the columns `xi`. `idz` is the context indicator (unknown to the algorithm). `y`, `ytrue` stand the detuned and the healthy labels respectively. `res` is the difference between `y` and `ytrue` (unknown to the algorithm).

More precisely, the KAGGLE benchmark dataset is an instantiation of what can be produced using the python package `pwp_bench`. Nevertheless, for the sake of completeness, the main guidelines used in `pwp_bench` are presented hereafter.

Fig. 2 shows an example of a dataframe[3]. The columns are defined as follows:

- ✓ The columns labelled as `xi` for $i \in \{1, \ldots, n_x\}$ (here $n_x = 6$) represents the features (sensors measurements).
- ✓ The column labelled by `idz` represents the context index. The number of contexts (number of values in the `idz` column) is almost always 4 except for two datasets where *only* three contexts are present (Fig. 1). The context is defined through the following steps which are repeated for the creation of each file `pb_i.pkl`:

[3]This corresponds to the file named `pb_19.pkl`.

1) First a randomly defined multi-variate polynomial in the features vector x is obtained which defines a scalar z that is a polynomial function of vector of features, namely:

$$z = P_{\text{context}}(x) := \sum_{i=1}^{n_m} \alpha_i \underbrace{\left[\prod_{j=1}^{n_x} x_j^{p_{ij}}\right]}_{\phi_i(x)} \tag{1}$$

where n_m is the number of monomials involved (randomly generated in the set $\{1, \ldots, 10\}$). Notice that the degree of the polynomial is defined by:

$$d\Big[P_{\text{context}}\Big] = \max\left\{\sum_{j=1}^{n_x} p_{ij} \quad i = 1, \ldots, n_m\right\} \tag{2}$$

The degree of the polynomial P_{context} is randomly generated inside the set $\{1, 2, 3\}$ for each of the files `pb_i.pkl` mentioned above.

2) The values of the scalar variable z defined by (2) is used to define the context associated to each row in the dataset. This is done by clustering the dataset based on the value of z so that a predefined number, say $n_c(=4)$ of sub-datasets is designated by partitioning the original full dataset. Once computed, this enables to create the context column, labelled by `idz` as shown in Fig. 2.

3) Once the `idz` column is created, the dataset is split into a number of blocks (greater than the number of context values) that are then randomly reordered so that sequences of different contexts, each appearing several times in the dataset inside timely consistent blocks are obtained emulating real-life situations.

   Examples of resulting context-induced features space's partitioning instances are shown in Fig. 3 for three files included in the dataset for which $n_x = 2$ enabling a graphical representation of the different context regions. Notice that in some cases, for a given context, the associated regions are not necessarily convex sets and might even be composed of two disjoint subsets in the features space.

✓ `ytrue` and `y` columns represent respectively the value of the label[4] y in the absence of anomalies and after introducing the anomaly to be detected. The value of `ytrue` and `y` are obtained using the following two steps which are repeated for each of the 30 files $\{\texttt{pb\_i.pkl}\}_{i=1}^{30}$:

1) For each subset of the data associated to the partition as defined above (through the values of the scalar z), a randomly generated polynomial, say $P_i^{(\texttt{idz})}$, in the features vector x is used to compute the value `ytrue` over the region indexed by `idz`. This is repeated for all the regions, namely for $\texttt{idz}\in\{1,\ldots,n_c\}$

2) A single region's index $j \in \{1,\ldots n_c\}$ is randomly selected to welcome a context-related fault, and parametric error is introduced in the second half of the dataset only over instances where $\texttt{idz} = j$. This is done by slightly detuning the coefficients of the polynomial $P_i^{(\texttt{idz})}$ and by recomputing the value of the detuned label `y` using the detuned polynomial while it is taken to be identical to `ytrue` on the remaining region indexed by $\texttt{idz}\neq j$. An example of the resulting time evolution of the different quantities is shown in Fig. 4 where $j = 2$ leading to a detuned values of y on the instances of $\texttt{idz} = j$ that are present in the second half of the data.

✓ Finally, the column labelled `res` is simply the difference between the last two columns. Fig. 5 shows a zoom over the region of Fig. 4 where the parametric anomaly is introduced in the second half of the dataset and only when the context is `idz=2`. The same is shown on Figures 6 and 7 where the parametric anomaly is introduced only on the occurrence of context `idz=3` in the second half of the experiment. Theses figures clearly suggest the difficulty of detecting the parametric anomalies of the benchmark as the value and the dynamics of the anomaly-induced behaviour remain quite similar in both amplitude and dynamics.

Notice also that in both cases, the size of the part of the dataset corresponding to the context for which the parametric anomaly is introduced is quite small which obviously means that the participation of this context to the definition of the normality is unbalanced making the detection even harder to perform. This unbalance regarding the representation of the different context in the training datasets is also common in the case of industrial time series as it has been mentioned when discussing the specificity of industrial use-case.

B. Normalized version of the benchmark

In the following results, a normalized version of the benchmark is also investigated. Indeed, in the absence of normalization, it might be possible by investigating the statistical properties of y to create clusters which might be highly correlated to the context index `idz`. In such situations which are not necessarily very frequent, the problem might be *simplified* by first creating the clusters and then fit a normality model for each of the resulting clusters.

In order to definitively avoid such potential simplification, it is possible to derive a normalized version of the benchmark in which, the y is normalized within each context so that the amplitudes of y become comparable over the different contexts making the clustering far from being an easy option. This is done in the following investigation by dividing the values of y inside each cluster by the value of the 95% quantile of $|y|$ inside the cluster in the training subset[5].

The files associated to the resulting datasets are referred to hereafter by:

$$\texttt{pbn\_i.pkl} \qquad i \in \{1,\ldots,30\}$$

Fig. 8 shows an example of normalization where it can be observed that in the absence of normalization it would be possible to clearly distinguish clusters of context from the simple observation of the value of y.

In the forthcoming sections, results are shown for both versions of the benchmark to serve as reference in future investigation.

C. Statement of the anomaly detection problem

Based on the presentation above the anomaly detection problem can be stated as follows:

[4]The y-prediction error is supposed to be used as the residual for the anomaly detection algorithm.

[5]The function that performs the normalization is provided in the documentation of the `pwp_bench` package, see https://mazenalamir.github.io/pwp_bench/

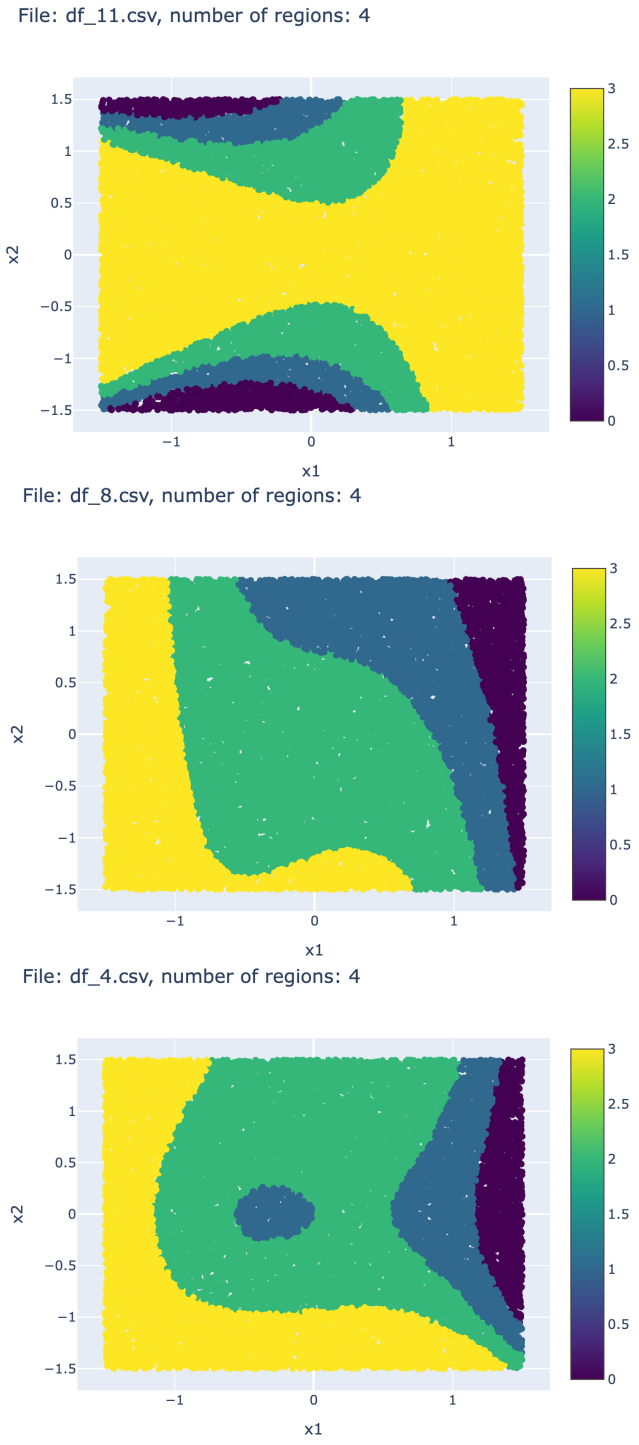

Fig. 3. Examples of partitioning-into-context regions in the features space where the label is defined by a specific multivariate polynomial over each region of context. Recall that while only two-dimensional features problems are amenable to such graphical representations, the `Kaggle` benchmark dataset contains datasets with the dimension of the vector of features ranging from 2 to 12.

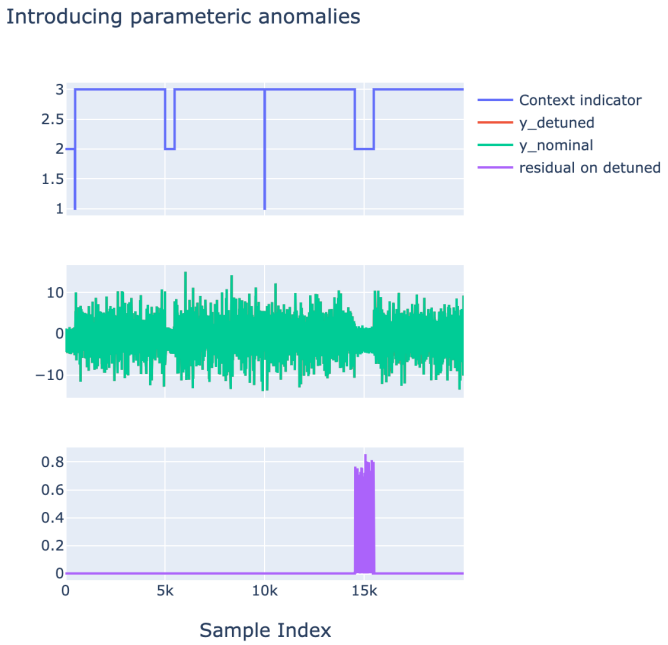

Fig. 4. Example of the time evolution of the columns content showing the context index `idz` (top), the healthy and detuned variable `ytrue` and `y` as well as the anomaly-induced difference between the two latter time series. The parametric anomaly is introduced for the context `idz`=2 in the second half of the dataset. A zoomed version of this figure is shown in Fig. 5.

Fig. 5. Zoomed view of Fig. 4 around the region where the anomaly is introduced showing the kind of error on the time evolution to be detected through normality characterization.

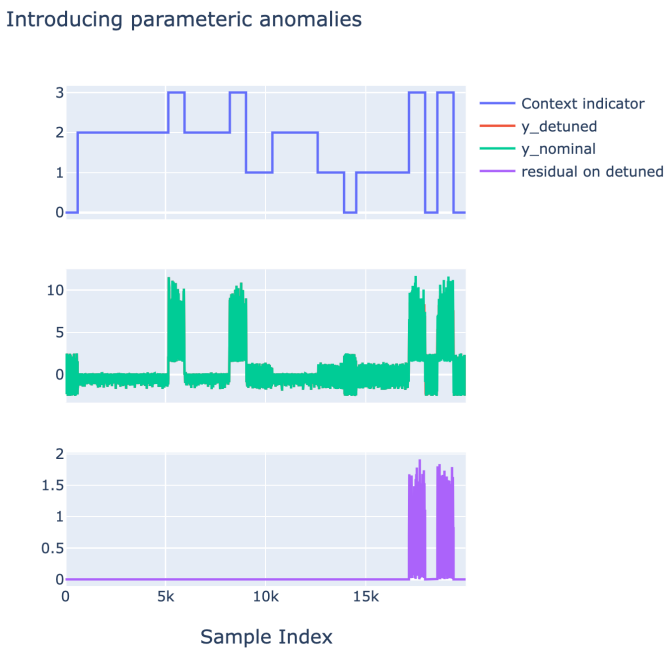

Fig. 6. Example of the time evolution of the columns content showing the context index `idz` (top), the healthy and detuned variable `ytrue` and `y` as well as the anomaly-induced difference between the two latter time series. The parametric anomaly is introduced for the context `idz`=3 in the second half of the dataset. A zoomed version of this figure is shown in Fig. 7.

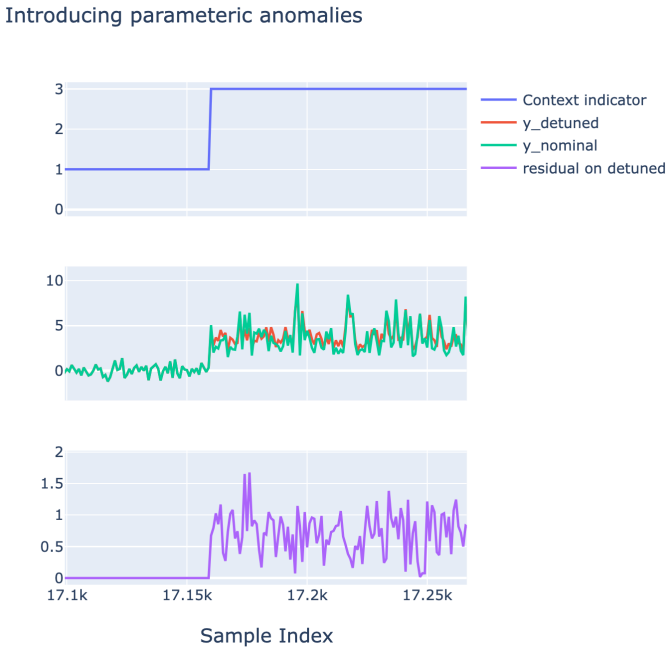

Fig. 7. Zoomed view of Fig. 6 around the region where the anomaly is introduced showing the kind of error on the time evolution to be detected through normality characterization.

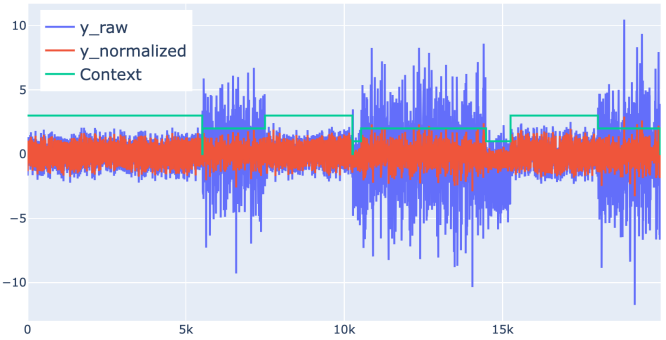

Fig. 8. Example of normalization performed on the file `pb_28.pkl` included in the benchmark dataset.

Statement of the anomaly detection problem.

For each dataset associated to one of the files contained in the benchmark, namely:

$$\texttt{pb\_i.pkl} \qquad i \in \{1, \ldots, 30\}$$

or their normalized versions:

$$\texttt{pbn\_i.pkl} \qquad i \in \{1, \ldots, 30\}$$

derive a normality characterization[a] using only the columns labelled by:

$$\texttt{x1}, \ldots, \texttt{x}_{n_x}, \texttt{y}$$

together with a threshold and use the resulting characterization to detect the anomaly introduced in the second half of each dataset.

[a]Possibly, but not necessarily, using the fact that it is the `y`-column that is impacted by the introduced parametric anomaly.

III. The Algorithms and the Evaluation Metrics

A. The normality characterization algorithm

The normality characterization and anomaly detection algorithm used in producing the baseline detection results for the benchmark have been recently introduced in the robotic context [4]. The idea is based on the identification, for each precision threshold `th`, of n_r polynomials $P_\kappa^{(\texttt{th})}, \kappa = 1, \ldots, n_r(\texttt{th})$ so that the the following residual can be defined:

$$e_{\texttt{th}}(x, y) := \min_{\kappa=1}^{n_r(\texttt{th})} \left| y - P_\kappa^{(\texttt{th})}(x) \right| \tag{3}$$

Notice that the number of polynomials $n_r(\texttt{th})$ as well as the polynomials $P_\kappa^{(\texttt{th})}, \kappa = 1, \ldots, n_r(\texttt{th})$ themselves, depend on the precision threshold `th`. The smaller `th` the higher is the number of polynomials involved in the definition (3) of the residual.

More precisely, the precision threshold `th` defined the precision reached by a polynomial to be accepted in the set of

polynomial involved in the relationship (3). This condition is expressed by:

$$\frac{\texttt{percentile}(e(\cdot),95)}{\texttt{percentile}(|y(\cdot)|,50)} \leq \texttt{th} \tag{4}$$

Table I shows some information regarding the implicit piecewise polynomial normality characterization that is obtained for the different files included in the original `Kaggle` dataset (without normalization). The same kind of numbers hold for the normalized dataset. The table shows the number of polynomial (#models=$n_r(\texttt{th})$), the total number of coefficients (#coeffs) and computation time (cpu in seconds) for different values of the precision threshold $\texttt{th}\in \{0.15, 0.2, 0.25, 0.3\}$ and for each of the different files including in the KAGGLE benchmark, namely `pb_i` for $i \in \{1, \ldots, 30\}$.

B. The bagging version of the residual

As the previously defined normality characterization is parametrized through the precision residual, a natural step is to come out with a *Bagging*-based[6] version of this characterization. This can be obtained by defining an averaged residual over all used values of the precision threshold, namely:

$$e := \frac{1}{\text{card}(\mathcal{T}_h)} \sum_{\texttt{th}\in\mathcal{T}_h} e_{\texttt{th}} \tag{5}$$

where $\mathcal{T}_h$ is the set of `th` values for which normality characterizations have been computed. As mentioned in the previous section, in the forthcoming results, this set is given by:

$$\mathcal{T}_h := \{0.15, 0.2, 0.25, 0.3\} \tag{6}$$

C. The evaluation metrics

Notice that the previous description enables to define the raw residual as an implicit difference between the value of the label and the closest predicted value among those provided by the different polynomials involved in the identified implicit relationship, evaluated at the vector of features x, namely $P_{\kappa}^{(\texttt{th})}(x)$.

But the total definition of the anomaly detection still involves two component:

- ✓ The definition of an averaging window's length, say $n_w \in \mathbb{N}$, which avoid a too nervous residual behaviour which is a common practice in anomaly detection algorithms. Once such a window's length n_w is adopted, a rolling averaging mechanism is used to deliver the filtered version of the residual:

$$\bar{e}_{\texttt{th}}(k) := \frac{1}{n_w}\left[\sum_{t=k-n_w+1}^{t=k} e_{\texttt{th}}(t)\right] \tag{7}$$

  or, in the bagging version:

$$\bar{e}(k) := \frac{1}{n_w}\left[\sum_{t=k-n_w+1}^{t=k} e(t)\right] \tag{8}$$

[6]In Machine Learning vocabulary, bagging refers to the technique of making several models predicting the same issue and to agglomerate their predictions to derive a single decision.

- ✓ The computation of a threshold on the resulting filtered residual beyond which, an alarm is raised. In the forthcoming results, this threshold is defined to be the 99% quantile of computed on the set of values $\bar{e}_{\texttt{th}}$ contained in the training set, denoted hereafter by $e_{\texttt{th}}|_{\text{train}}$.

  Therefore, a boolean indicator is created that can be labelled as the predicted anomaly indicator:

$$f_{\text{pred}} := \bar{e}_{\texttt{th}} > \texttt{percentile}(\bar{e}_{\texttt{th}}|_{\text{train}}, 99) \tag{9}$$

  or, in the bagging version:

$$f_{\text{pred}} := \bar{e} > \texttt{percentile}(\bar{e}|_{\text{train}}, 99) \tag{10}$$

  that can be compared to the true label:

$$f_{\text{true}} := \texttt{res} > 0 \tag{11}$$

  where `res` is the columns provided in the dataset.

Based on the pair $(f_{\text{predicted}}, f_{\text{true}})$, the confusion matrix can be computed and the associated detection rate and false alarm can be derived in the form of the tuple:

$$(\texttt{False Alarm}, \texttt{Detection Rate})$$

representing respectively, the ratio of health instances which are declared to be faulty and the ratio of the truly faulty instances that are detected as such. This tuple is reported as the performance summary in the resutls shown in the next section.

IV. RESULTS

A. Results with bagging

Fig. 9 shows the detection and the false alarm rate that are obtained for all the files contained in the benchmark as well as on their normalized versions.

An averaging window of $n_w = 200$ is used in order to smooth the decision and the 99% quantile is used to define the binary decision over the rolling window according to (9) and (10).

The results of Fig. 9 clearly show the excellent performances of the bagging-based solution since only one experiment show no detection capability of the model while for all the other files, almost perfect detection and a systematically small false alarm rates that averaged to 5% and never exceed 8%.

B. Performance results without bagging

The performance results of the individual models corresponding to the precision thresholds $\texttt{th}\in \{0.15, 0.2, 0.25, 0.3\}$ are shown respectively on Figs 10, 11, 12 and 13.

The performance results are still quite decent as the number of *failures* never exceeds four files among the thirty available ones. Notice also that these files are very frequently different among the solutions which explains the high success of the bagging-based solution. Indeed, when accidentally one of the solution does not detect the anomaly in a file, there is quite high probability that the anomaly is detected by the other individual models. This validates the very simple principle of bagging in machine learning. Notice however that this

File	th=0.15			th=0.2			th=0.25			th=0.3		
	#models	#coeffs	cpu (s)	#models	#coeffs	cpu (s)	#models	#coeffs	cpu (s)	#models	#coeffs	cpu (s)
df_14	27	562	15.2	7	144	8.7	7	140	5.9	6	155	7.4
df_28	2	8	12.2	2	8	11.3	2	8	12.2	2	48	12.6
df_29	2	7	9.1	3	85	10.1	2	47	9.3	2	52	9.4
df_15	7	188	9.9	4	9	4.8	8	194	8.9	9	249	7.2
df_17	2	23	4.5	2	20	4.4	2	12	4.5	2	30	4.7
df_16	4	104	6.3	15	482	12.1	20	569	9.8	9	277	10.5
df_12	5	91	5.4	4	83	5.0	4	85	5.1	5	100	5.1
df_13	20	295	8.8	5	94	6.1	4	75	5.1	5	110	5.9
df_9	7	63	6.6	4	54	4.1	5	48	4.4	3	37	3.2
df_11	7	88	5.8	6	61	5.6	4	52	4.1	3	32	3.1
df_10	8	89	7.8	4	49	3.7	4	44	4.0	4	42	3.8
df_8	5	48	4.4	4	34	3.8	4	39	3.9	3	35	3.1
df_5	5	57	4.3	6	54	3.3	3	34	3.1	2	26	2.2
df_21	3	82	7.3	3	51	7.3	3	29	7.2	3	50	7.5
df_20	6	188	12.7	6	186	8.2	6	149	12.3	5	154	11.3
df_4	7	72	6.0	5	59	5.0	4	47	4.2	4	50	4.1
df_22	6	231	8.8	3	18	7.2	3	77	7.2	3	37	7.3
df_6	4	53	3.7	3	35	3.1	3	32	2.6	2	25	2.4
df_7	3	35	3.1	3	36	3.0	4	42	3.3	4	32	3.6
df_23	3	54	9.3	3	100	9.1	3	94	9.1	3	16	9.4
df_27	7	238	28.8	10	435	31.5	5	222	19.7	3	107	11.9
df_3	7	87	5.7	6	61	5.3	4	51	4.2	4	47	4.2
df_2	4	25	3.8	3	24	2.9	4	32	3.2	3	28	3.1
df_26	2	51	7.0	2	49	7.3	2	52	7.1	2	51	7.2
df_18	2	13	4.3	2	37	4.4	2	55	4.5	2	19	4.5
df_30	2	9	9.9	2	8	9.4	3	62	16.1	2	45	9.9
df_24	3	93	9.2	3	51	9.2	3	57	9.4	3	55	9.2
df_25	2	47	7.6	2	48	7.1	2	54	7.0	2	48	6.7
df_1	3	56	15.9	3	67	15.4	3	65	14.8	3	82	15.4
df_19	17	648	18.6	9	336	14.1	7	290	13.2	4	109	8.7

TABLE I

CARDINALITY DASHBOARD: NUMBER OF POLYNOMIAL (#MODELS), TOTAL NUMBER OF COEFFICIENTS (#COEFFS) AND COMPUTATION TIME (CPU) FOR DIFFERENT VALUES OF THE PRECISION THRESHOLD $\text{TH} \in \{0.15, 0.2, 0.25, 0.3\}$ FOR THE DIFFERENT FILES INCLUDING IN THE KAGGLE BENCHMARK.

positive bagging mechanism does not materialize for the file `pb_15.csv` since the majority (three) among the four *voters* does not detect the anomaly in this case.

C. Example of the time evolution inside files

In order to better appreciate the difficulty of the anomaly detection task and to better understand the specificity of the underlying benchmark which is the main subject of the paper, Figs. 19-23 show examples of the temporal evolution of the default-free and the detuned labels together with the context indicator `idz`. This is shown for five of the thirty datasets for the sake of illustration.

These examples show how subtle is the introduced differences to be detected and sometimes how small is the portion of the training data that concerns the context over which the fault is introduced in the second half of the dataset.

V. CONCLUSION & FUTURE WORKS

In this paper a benchmark is presented for one class algorithms for parametric anomaly detection in industrial time series.

The dataset is publicly available on the KAGGLE data repository. The benchmark provides examples that meet some features that are described in this paper and which are commonly encountered in industrial datasets. The benchmark dataset has been generated via a publicly available python module, which is discussed in the paper and that can be used to generate more examples if needed.

Beside describing and discussing the quality of the benchmark dataset, the paper proposes a baseline performance results that are obtained using a recently developed algorithm that is based on normality characterization via parsimonious multivariate polynomial relationships. The results show quite promising performance of the algorithm beyond the robotic example to which it has been initially dedicated.

Current investigation includes the generation of performance indicators for a bench of candidate solutions to the problem of parametric anomaly detection addressed in the paper.

Such a necessary comparison enables a more suitable basis for the specific problem under consideration as the available benchmark are not necessary compatible with the industrial time series use-case. This can be of a tremendous importance in selecting the appropriate algorithms for this very crucial domain.

REFERENCES

[1] L. Colombi, M. Vespa, N. Belletti, M. Brina, S. Dahdal, F. Tabanelli, F. Resca, E. Bellodi, M. Tortonesi, C. Stefanelli, and M. Vignoli,

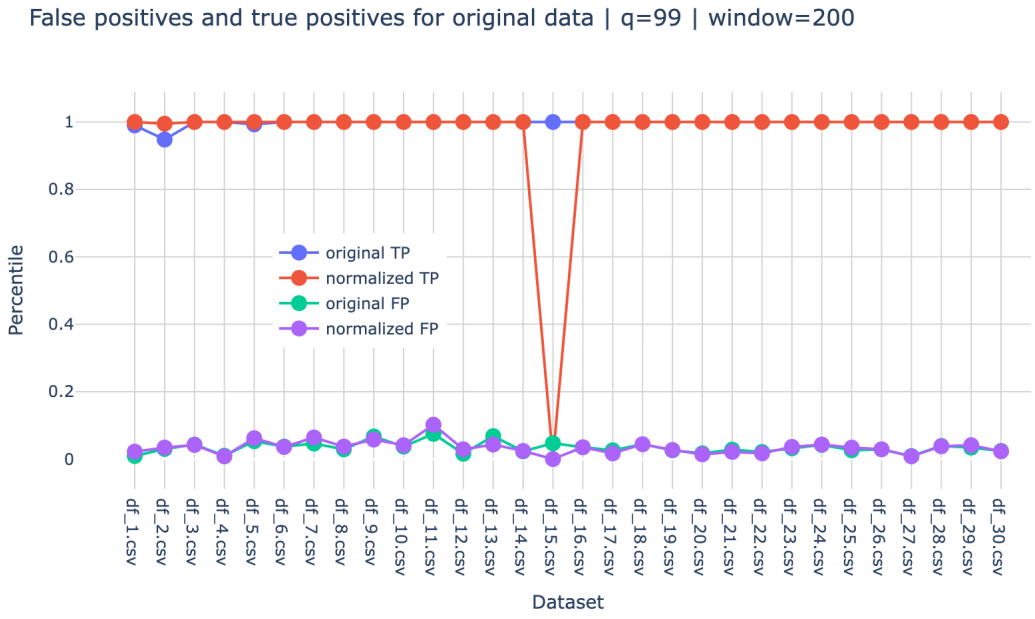

Fig. 9. Detection and false alarm rates of the bagging-based solution over the 30 files of the benchmark dataset.

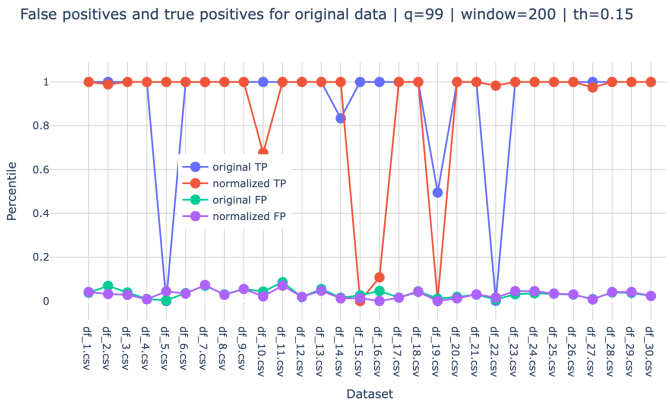

Fig. 10. Detection and false alarm rates of the solution associated to the precision threshold $\texttt{th} = 0.15$ over the 30 files of the benchmark dataset.

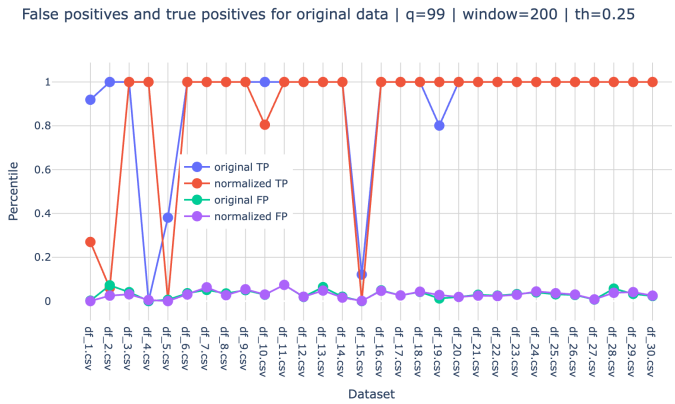

Fig. 12. Detection and false alarm rates of the solution associated to the precision threshold $\texttt{th} = 0.25$ over the 30 files of the benchmark dataset.

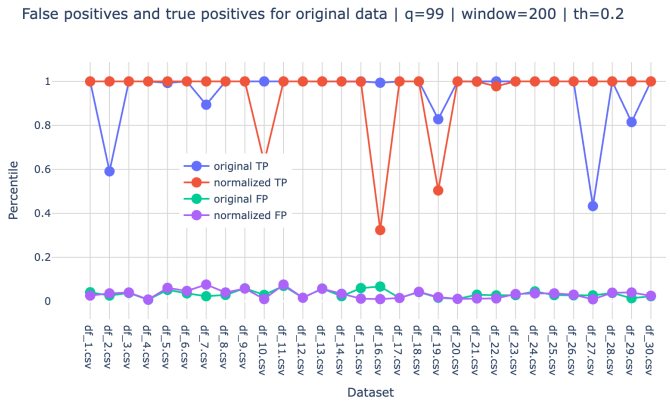

Fig. 11. Detection and false alarm rates of the solution associated to the precision threshold $\texttt{th} = 0.2$ over the 30 files of the benchmark dataset.

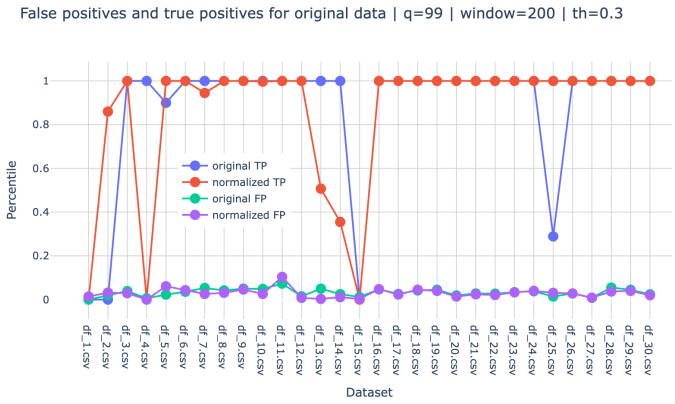

Fig. 13. Detection and false alarm rates of the solution associated to the precision threshold $\texttt{th} = 0.3$ over the 30 files of the benchmark dataset.

Bagging-based detector	original (FP/TP)	normalized (FP/TP)
df_1.csv	(0.009, 0.990)	(0.022, 1.000)
df_2.csv	(0.030, 0.948)	(0.034, 0.995)
df_3.csv	(0.043, 1.000)	(0.042, 1.000)
df_4.csv	(0.010, 1.000)	(0.008, 1.000)
df_5.csv	(0.053, 0.993)	(0.062, 1.000)
df_6.csv	(0.037, 1.000)	(0.036, 1.000)
df_7.csv	(0.046, 1.000)	(0.064, 1.000)
df_8.csv	(0.029, 1.000)	(0.037, 1.000)
df_9.csv	(0.067, 1.000)	(0.058, 1.000)
df_10.csv	(0.037, 1.000)	(0.041, 1.000)
df_11.csv	(0.075, 1.000)	(0.102, 1.000)
df_12.csv	(0.016, 1.000)	(0.029, 1.000)
df_13.csv	(0.068, 1.000)	(0.043, 1.000)
df_14.csv	(0.023, 1.000)	(0.024, 1.000)
df_15.csv	(0.047, 1.000)	(0.000, 0.000)
df_16.csv	(0.035, 1.000)	(0.035, 1.000)
df_17.csv	(0.026, 1.000)	(0.017, 1.000)
df_18.csv	(0.044, 1.000)	(0.044, 1.000)
df_19.csv	(0.026, 1.000)	(0.027, 1.000)
df_20.csv	(0.017, 1.000)	(0.014, 1.000)
df_21.csv	(0.028, 1.000)	(0.021, 1.000)
df_22.csv	(0.021, 1.000)	(0.017, 1.000)
df_23.csv	(0.032, 1.000)	(0.036, 1.000)
df_24.csv	(0.042, 1.000)	(0.043, 1.000)
df_25.csv	(0.026, 1.000)	(0.034, 1.000)
df_26.csv	(0.029, 1.000)	(0.029, 1.000)
df_27.csv	(0.009, 1.000)	(0.008, 1.000)
df_28.csv	(0.039, 1.000)	(0.038, 1.000)
df_29.csv	(0.034, 1.000)	(0.041, 1.000)
df_30.csv	(0.024, 1.000)	(0.023, 1.000)

Fig. 14. Precise results regarding the detection and the false alarm rates for the bagging-based solution. These results are showing graphically in Fig. 9.

($\texttt{th} = 0.2$)-based detector	original (FP/TP)	normalized (FP/TP)
df_1.csv	(0.041, 1.000)	(0.026, 1.000)
df_2.csv	(0.026, 0.591)	(0.036, 1.000)
df_3.csv	(0.038, 1.000)	(0.040, 1.000)
df_4.csv	(0.007, 1.000)	(0.008, 1.000)
df_5.csv	(0.052, 0.993)	(0.061, 1.000)
df_6.csv	(0.036, 1.000)	(0.047, 1.000)
df_7.csv	(0.023, 0.894)	(0.076, 1.000)
df_8.csv	(0.029, 1.000)	(0.040, 1.000)
df_9.csv	(0.059, 1.000)	(0.058, 1.000)
df_10.csv	(0.029, 1.000)	(0.010, 0.633)
df_11.csv	(0.070, 1.000)	(0.076, 1.000)
df_12.csv	(0.016, 1.000)	(0.016, 1.000)
df_13.csv	(0.058, 1.000)	(0.057, 1.000)
df_14.csv	(0.023, 1.000)	(0.034, 1.000)
df_15.csv	(0.060, 1.000)	(0.012, 0.999)
df_16.csv	(0.067, 0.994)	(0.010, 0.324)
df_17.csv	(0.015, 1.000)	(0.015, 1.000)
df_18.csv	(0.042, 1.000)	(0.043, 1.000)
df_19.csv	(0.016, 0.828)	(0.019, 0.504)
df_20.csv	(0.011, 1.000)	(0.011, 1.000)
df_21.csv	(0.030, 1.000)	(0.013, 0.999)
df_22.csv	(0.027, 1.000)	(0.013, 0.978)
df_23.csv	(0.028, 1.000)	(0.033, 1.000)
df_24.csv	(0.045, 1.000)	(0.036, 1.000)
df_25.csv	(0.028, 1.000)	(0.036, 1.000)
df_26.csv	(0.027, 1.000)	(0.030, 1.000)
df_27.csv	(0.027, 0.433)	(0.009, 1.000)
df_28.csv	(0.038, 1.000)	(0.039, 1.000)
df_29.csv	(0.014, 0.815)	(0.040, 1.000)
df_30.csv	(0.023, 1.000)	(0.025, 1.000)

Fig. 16. Precise results regarding the detection and the false alarm rates for the solution associated to the precision threshold $\texttt{th}$=0.2. These results are showing graphically in Fig. 11.

($\texttt{th} = 0.15$)-based detector	original (FP/TP)	normalized (FP/TP)
df_1.csv	(0.038, 1.000)	(0.041, 1.000)
df_2.csv	(0.069, 1.000)	(0.032, 0.989)
df_3.csv	(0.039, 1.000)	(0.028, 1.000)
df_4.csv	(0.009, 1.000)	(0.008, 1.000)
df_5.csv	(0.003, 0.000)	(0.044, 1.000)
df_6.csv	(0.036, 1.000)	(0.034, 1.000)
df_7.csv	(0.070, 1.000)	(0.073, 1.000)
df_8.csv	(0.030, 1.000)	(0.028, 1.000)
df_9.csv	(0.055, 1.000)	(0.055, 1.000)
df_10.csv	(0.043, 1.000)	(0.021, 0.676)
df_11.csv	(0.086, 1.000)	(0.070, 1.000)
df_12.csv	(0.018, 1.000)	(0.019, 1.000)
df_13.csv	(0.055, 1.000)	(0.047, 1.000)
df_14.csv	(0.015, 0.834)	(0.012, 1.000)
df_15.csv	(0.026, 1.000)	(0.013, 0.000)
df_16.csv	(0.046, 1.000)	(0.000, 0.108)
df_17.csv	(0.016, 1.000)	(0.015, 1.000)
df_18.csv	(0.044, 1.000)	(0.042, 1.000)
df_19.csv	(0.011, 0.495)	(0.000, 0.005)
df_20.csv	(0.019, 1.000)	(0.012, 1.000)
df_21.csv	(0.029, 1.000)	(0.030, 1.000)
df_22.csv	(0.007, 0.001)	(0.016, 0.983)
df_23.csv	(0.031, 1.000)	(0.045, 1.000)
df_24.csv	(0.035, 1.000)	(0.045, 1.000)
df_25.csv	(0.033, 1.000)	(0.034, 1.000)
df_26.csv	(0.029, 1.000)	(0.030, 1.000)
df_27.csv	(0.009, 1.000)	(0.007, 0.975)
df_28.csv	(0.039, 1.000)	(0.041, 1.000)
df_29.csv	(0.037, 1.000)	(0.041, 1.000)
df_30.csv	(0.023, 1.000)	(0.023, 1.000)

Fig. 15. Precise results regarding the detection and the false alarm rates for the solution associated to the precision threshold $\texttt{th}$=0.15. These results are showing graphically in Fig. 10.

($\texttt{th} = 0.25$)-based detector	original (FP/TP)	normalized (FP/TP)
df_1.csv	(0.002, 0.919)	(0.000, 0.270)
df_2.csv	(0.072, 1.000)	(0.025, 0.060)
df_3.csv	(0.041, 1.000)	(0.031, 1.000)
df_4.csv	(0.001, 0.000)	(0.005, 1.000)
df_5.csv	(0.006, 0.381)	(0.000, 0.002)
df_6.csv	(0.036, 1.000)	(0.030, 1.000)
df_7.csv	(0.051, 1.000)	(0.063, 1.000)
df_8.csv	(0.035, 1.000)	(0.027, 1.000)
df_9.csv	(0.050, 1.000)	(0.054, 1.000)
df_10.csv	(0.029, 1.000)	(0.030, 0.805)
df_11.csv	(0.074, 1.000)	(0.074, 1.000)
df_12.csv	(0.019, 1.000)	(0.020, 1.000)
df_13.csv	(0.064, 1.000)	(0.048, 1.000)
df_14.csv	(0.020, 1.000)	(0.016, 1.000)
df_15.csv	(0.000, 0.121)	(0.001, 0.000)
df_16.csv	(0.049, 1.000)	(0.047, 1.000)
df_17.csv	(0.026, 1.000)	(0.026, 1.000)
df_18.csv	(0.042, 1.000)	(0.043, 1.000)
df_19.csv	(0.012, 0.801)	(0.028, 1.000)
df_20.csv	(0.019, 1.000)	(0.019, 1.000)
df_21.csv	(0.029, 1.000)	(0.025, 1.000)
df_22.csv	(0.025, 1.000)	(0.023, 1.000)
df_23.csv	(0.032, 1.000)	(0.029, 1.000)
df_24.csv	(0.040, 1.000)	(0.044, 1.000)
df_25.csv	(0.031, 1.000)	(0.036, 1.000)
df_26.csv	(0.028, 1.000)	(0.030, 1.000)
df_27.csv	(0.007, 1.000)	(0.008, 1.000)
df_28.csv	(0.057, 1.000)	(0.038, 1.000)
df_29.csv	(0.033, 1.000)	(0.041, 1.000)
df_30.csv	(0.023, 1.000)	(0.025, 1.000)

Fig. 17. Precise results regarding the detection and the false alarm rates for the solution associated to the precision threshold $\texttt{th}$=0.25. These results are showing graphically in Fig. 12.

(th = 0.3)-based detector	original (FP/TP)	normalized (FP/TP)
df_1.csv	(0.000, 0.000)	(0.014, 0.000)
df_2.csv	(0.021, 0.000)	(0.032, 0.860)
df_3.csv	(0.039, 1.000)	(0.029, 1.000)
df_4.csv	(0.006, 1.000)	(0.000, 0.000)
df_5.csv	(0.023, 0.900)	(0.061, 1.000)
df_6.csv	(0.036, 1.000)	(0.043, 1.000)
df_7.csv	(0.053, 1.000)	(0.026, 0.945)
df_8.csv	(0.042, 1.000)	(0.031, 1.000)
df_9.csv	(0.050, 1.000)	(0.046, 1.000)
df_10.csv	(0.049, 1.000)	(0.026, 0.997)
df_11.csv	(0.073, 1.000)	(0.104, 1.000)
df_12.csv	(0.015, 1.000)	(0.008, 1.000)
df_13.csv	(0.050, 1.000)	(0.003, 0.507)
df_14.csv	(0.025, 1.000)	(0.011, 0.355)
df_15.csv	(0.012, 0.000)	(0.003, 0.000)
df_16.csv	(0.047, 1.000)	(0.048, 1.000)
df_17.csv	(0.026, 1.000)	(0.023, 1.000)
df_18.csv	(0.042, 1.000)	(0.045, 1.000)
df_19.csv	(0.045, 1.000)	(0.040, 1.000)
df_20.csv	(0.019, 1.000)	(0.014, 1.000)
df_21.csv	(0.028, 1.000)	(0.024, 1.000)
df_22.csv	(0.027, 1.000)	(0.021, 1.000)
df_23.csv	(0.034, 1.000)	(0.033, 1.000)
df_24.csv	(0.038, 1.000)	(0.040, 1.000)
df_25.csv	(0.014, 0.289)	(0.031, 1.000)
df_26.csv	(0.028, 1.000)	(0.028, 1.000)
df_27.csv	(0.008, 1.000)	(0.009, 1.000)
df_28.csv	(0.055, 1.000)	(0.037, 1.000)
df_29.csv	(0.045, 1.000)	(0.040, 1.000)
df_30.csv	(0.024, 1.000)	(0.020, 1.000)

Fig. 18. Precise results regarding the detection and the false alarm rates for the solution associated to the precision threshold th=0.3. These results are showing graphically in Fig. 13.

"Embedding models for multivariate time series anomaly detection in industry 5.0," *Data science and engineering*, 2025.

[2] M. Jung, H. Jang, W. Kwon, J. Seo, S. Park, B. Park, J. Park, D. Yu, and S. Lee, "A comparative study of customized algorithms for anomaly detection in industry-specific power data," *Energies (Basel)*, vol. 18, no. 14, pp. 3720–, 2025.

[3] M. Alamir, "Parametric anomaly detection in time-series," 2025. [Online]. Available: https://www.kaggle.com/dsv/13462796

[4] M. Alamir and S. Clavel, "An algorithm for sparse piece-wise polynomial residual generation for anomalies detection in industrial manipulator robots," *Control engineering practice*, vol. 165, no. December 2025, pp. 106 545–, 2025.

[5] M. Alamir, "pwpbench: A python package for the creation of benchmarks problems for anomaly detection in multi-context industrial data." 2025. [Online]. Available: http://arxiv.org/abs/xxx

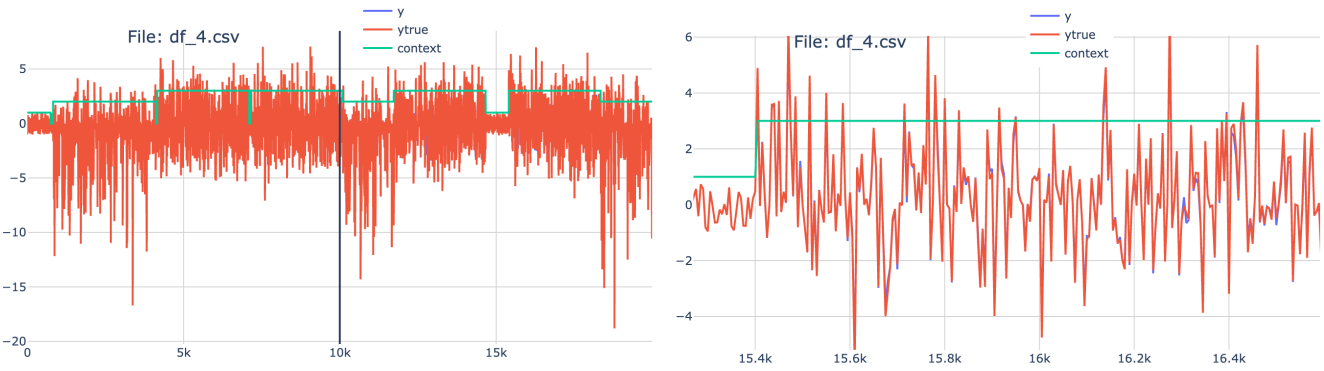

Fig. 19. File `df_4`: default introduced on context `idz` $= 3$ in the second half of the dataset. (left) the total time scope with the vertical solid black vertical line denoting the end of the training data. **Note how small is the difference between the original label and the fault induced detuned value.**

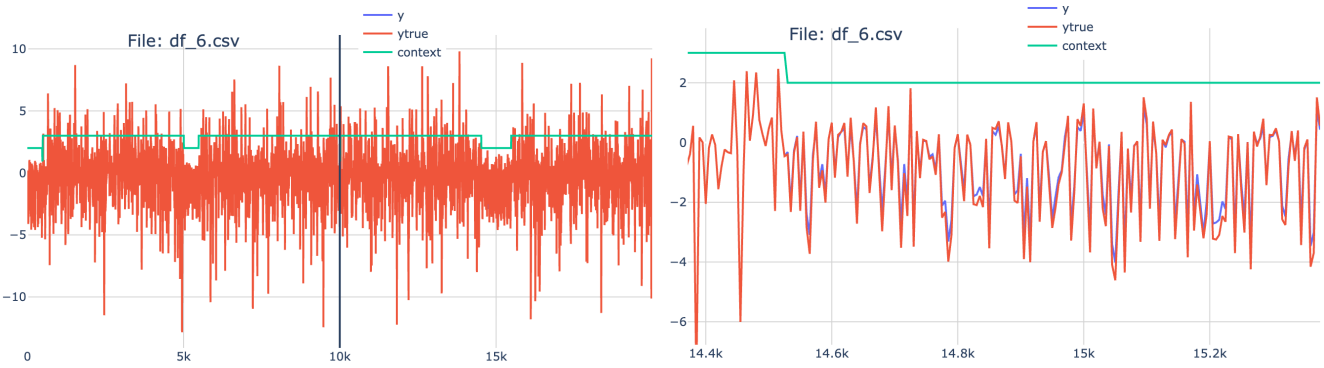

Fig. 20. File `df_6`: default introduced on context `idz` $= 2$ in the second half of the dataset. (left) the total time scope with the vertical solid black vertical line denoting the end of the training data. **Note how small is the portion of the training data during which the context labelled by `idz` $= 2$ over which the default is introduced in the second half of the data**.

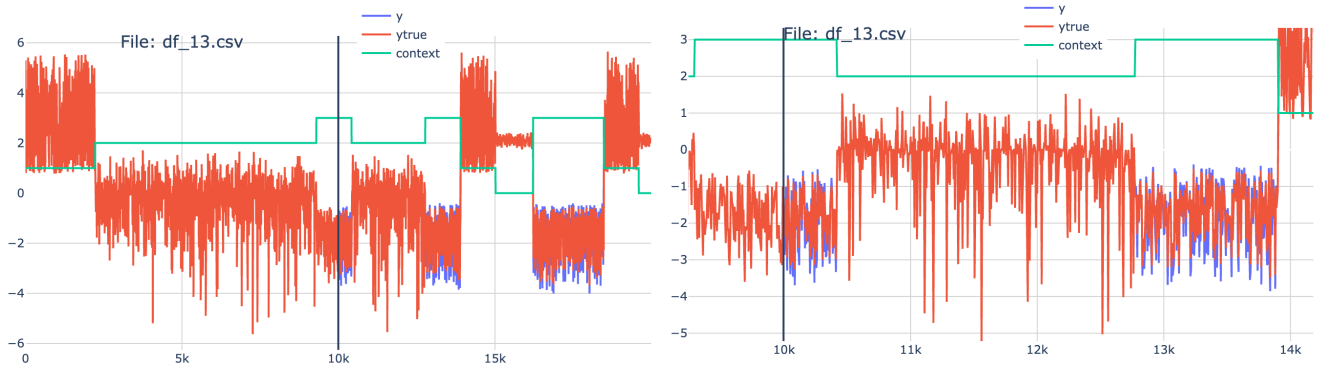

Fig. 21. File `df_13`: default introduced on context `idz` $= 3$ in the second half of the dataset. (left) the total time scope with the vertical solid black vertical line denoting the end of the training data. **Note how small is the portion of the training data during which the context labelled by `idz` $= 3$ over which the default is introduced in the second half of the data**.

Fig. 22. File `df_16`: default introduced on context `idz` $= 1$ in the second half of the dataset. (left) the total time scope with the vertical solid black vertical line denoting the end of the training data. **Note how small is the portion of the training data during which the context labelled by `idz` $= 1$ over which the default is introduced in the second half of the data**.

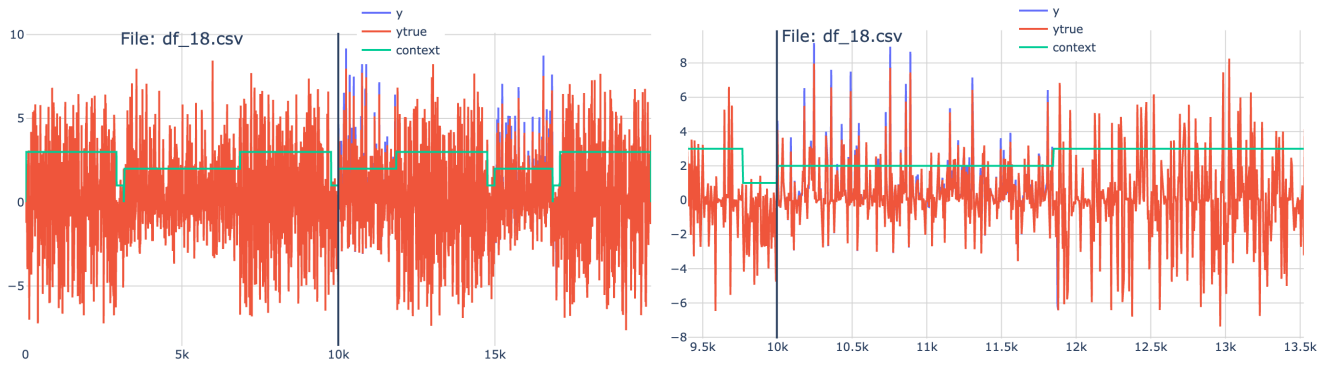

Fig. 23. File `df_16`: default introduced on context `idz` $= 1$ in the second half of the dataset. (left) the total time scope with the vertical solid black vertical line denoting the end of the training data. **Note how subtle is the introduced difference**.