Unsupervised Machine Learning Algorithms for Edge Novelty Detection

April 2024

Candidate: Ariel Priarone
Supervisor: Marcello Chiaberge **Co-supervisors:** Umberto Albertin, Gianluca Dara

I. INTRODUCTION

Predictive Maintenance (PM) and Novelty Detection (ND) are important topics in modern industrial engineering, aimed at proactively identifying equipment failures before they affect system functionality. Embracing these practices is crucial for reducing equipment downtime and optimizing maintenance efforts. PM aims to quantify and forecast the state of degradation of a system. A quite novel frontier is the direct implementation of PM algorithms within the maintained device, using the principles of Edge Computing.

A. Motivation

Despite the Fourth Industrial Revolution, the maintenance approach remained unchanged in many industrial applications. The primary factor impeding the advancement of the maintenance approach is the significant expense associated with implementing Condition Based (CB) or PM strategies, coupled with a lack of knowledge about the modelling or behaviour of a failing system.

According to a recent survey by the U.S. Department of Commerce, the top 25% of establishments relying on reactive maintenance were associated with 3.3 times more downtime than those in the bottom 25%.

B. Objective

The goal of this project is to design, develop and test a *degradation* based CB framework that performs ND, Fault Detection (FD) and PM, using one or several Unsupervised Machine Learning (UML) algorithms. The structure of the framework is thought to be modular and general-purpose to ease the implementation into different systems. It is developed following an unsupervised approach to overcome the common lack of physical models and labelled data of the maintained device. This framework has to be deployed for both PC and Edge Computing: the former developed in Python, the latter in C.

II. PROPOSED FRAMEWORK

The solution developed in this project is thought to be set up on a new device, and linked to the sensors of the most informative quantities of the system. In the first phase of commissioning, the framework collects the data, extracts the features and stores them. When the data collected is enough to characterize all the modes of operation of the maintained system, the UML model can be trained. Finally, the framework continues collecting new data, extracting the features, and evaluating the new data. The framework now produces a Novelty Metric (NM) that quantifies how abnormal the new data are. This phase can last indefinitely. When the NM overshoots a certain threshold, a warning is issued to the maintenance team. They can then decide to perform a maintenance action or to continue monitoring the system. If they declare the system as healthy, the framework can be retrained with the new data, to update the UML model. Otherwise, a second UML model can be trained to characterize the newly discovered fault and perform FD in the future.

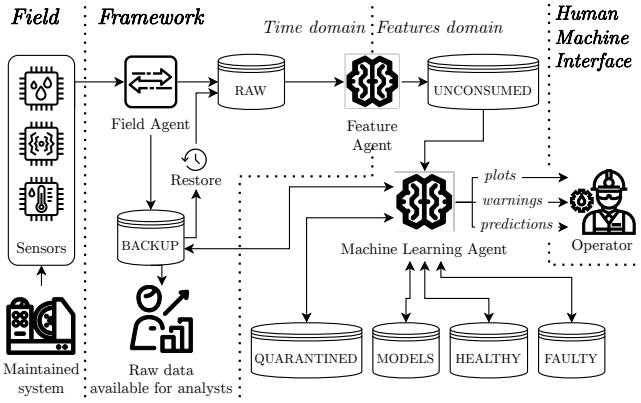

Fig. 1. The structure of the proposed framework

A. Software Agents

The proposed framework is based on software agents. Each agent is autonomous and performs a specific task. The developed agents, as shown in Fig. 1, are:

- **Field Agent (FiA)**: responsible for the synchronous sampling of the data.
- **Feature Agent (FA)**: extracts the features from the time series.
- **Machine Learning Agent (MLA)**: trains the UML algorithms and then performs ND, FD and PM. It reports the results to the user.

B. Database

All the Agents are connected to a common database. In the case of the PC implementation, MongoDB has been used. In the case of the Edge implementation, the data are stored directly in the microcontroller's memory. Regarding the structure shown in Fig. 1, the MongoDB database is composed of seven collections: *Raw* containing the time series, *Unconsumed* containing the features to be evaluated by the MLA, *Quarantined, Healthy* and *Faulty* containing the features that have been flagged as novelty, normal or faulty, respectively, and *Models* containing the UML models. The collection *Backup* is general purpose.

C. Multiple Instances

As shown in Fig. 2, the framework can be implemented in multiple instances. This is useful to better isolate the location of the anomaly in a complex system. The larger the set of sensors that a single instance of the framework is connected

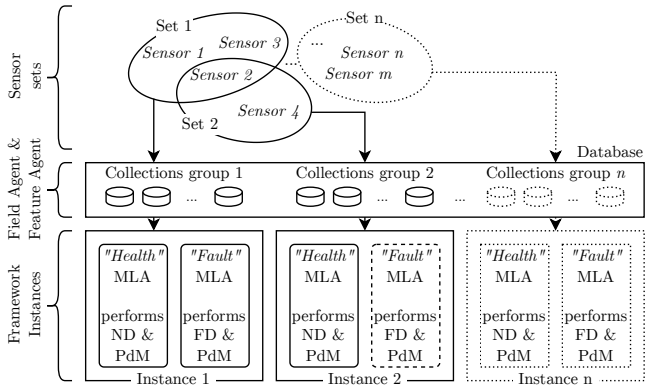

Fig. 2. Multiple instances implementation of the framework

to, the more difficult it is to isolate the anomaly. On the other hand, configuring a large group of sensors allows the detection of complex anomalies.

D. Unsupervised Machine Learning Models

The considered UML implemented in the framework are: *K-means*, *DBSCAN*, *Gaussian Mixture Model* (GMM), *Isolation Forest* (IF), *Local Outlier Factor* (LOF), *One-Class Support Vector Machine* (ν-SVM).

K-means and DBSCAN are traditionally clustering algorithms, so a custom NM has been developed for this project. In the case of the K-means, the NM is the distance of a new instance in the feature space from the closest centroid, normalized by the cluster radius. For the DBSCAN, the NM is the distance to the new instance from the closest instance in a cluster, normalized by the neighborhood radius.

The other models are already commonly used for ND, so the NM has been linked to the "score" provided by the library functions.

If the UML is trained with faulty data to perform FD, the NM measures "how not faulty" the new data are. In this case, the value is transformed into a Fault Metric (FM) with the function $\mathrm{FM} = -\ln(\mathrm{NM} + 1)$ to preserve the coherence.

III. FEATURE EXTRACTION

The framework is developed to acquire time series data from an arbitrary configuration of sensors. The FiA provides time series records according to a predefined cycle or synchronized with the device operation. It ensures that the sampling process is synchronized with the correct sampling frequency. Once a time series is available, the FA extracts the features from the time-domain data. Every time series is linked to a specific set of features to be extracted, configurable in the `.config` file of the framework.

A. Feature set

The considered features are divided into two categories:

- **Time domain features**: Mean, Standard deviation, Peak-to-peak value (P2P), Root Mean Square value (RMS), Skewness and Kurtosis.
- **Frequency domain features**: Energy of the Wavelet Packet Decomposition (WPD) coefficients, Fast Fourier Transform (FFT) coefficients. The WPD is based on the PyWavelets[1] library, for Python, and on the Wavelib[2] library, for C.

[1] `https://github.com/PyWavelets/pywt.git`
[2] `https://github.com/rafat/wavelib.git`

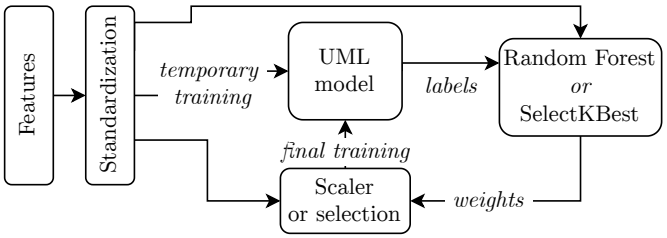

Fig. 3. Feature scaling and selection procedure

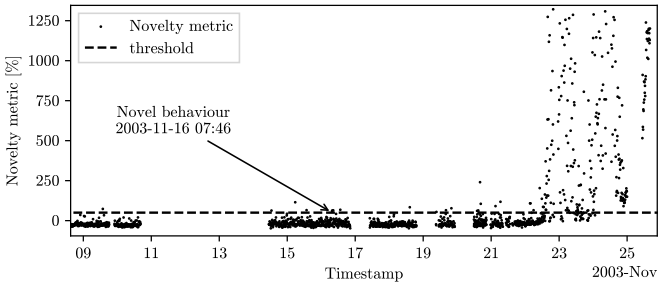

Fig. 4. Results of ND on the IMS dataset

The time domain features are computed in the corrected form for sampled data. In the frequency domain, the WPD is preferred in this work, because it reduces the dimensionality of the feature space. The features are standardized along the training dataset, so that the mean and standard deviation are 0 and 1, respectively. This is done to ease the training of the UML algorithms.

B. Scaling and selection

Despite the standardization, during the experimental validation, it has been observed that some features are more informative than others. To reduce the impact of the less informative features, an optional feature scaling step can be performed. The scaling is done by multiplying the features by a weight array. The weights can be computed by performing a Random Forest training or using the SelectKBest library method. Alternatively, the weights can be used to remove the less informative features, reducing the dimensionality of the feature space. This procedure is shown in Fig. 3.

IV. VALIDATION

A. On public datasets

The PC implementation of the framework has been thoroughly tested on a publicly available dataset provided by the Center for Intelligent Maintenance Systems (IMS). The dataset contains time series of the vibration of four bearings, for a total of three separate "run to failure" experiences. The dataset provides information about the location and type of defects observed after each test.

TABLE I
COMPARISON OF THE RESULTS FOR THE TEST N°1 OF IMS DATASET.

Algorithm	**ND event**	**LT** [min]	**LT** [days]
K-means	2003-11-16 07:46	**13913**	**9.6**
DBSCAN	2003-11-22 15:06	4833	3.3
GMM	2003-11-22 03:47	5513	3.8
BGMM	2003-11-22 03:45	5514	3.8
ν-SVM	2003-11-22 14:56	4844	3.3
IF	2003-11-16 10:08	13771	9.6
LOF	2003-11-16 07:48	13912	9.6
P2P without any ML	2003-11-22 16:06	4774	3.3

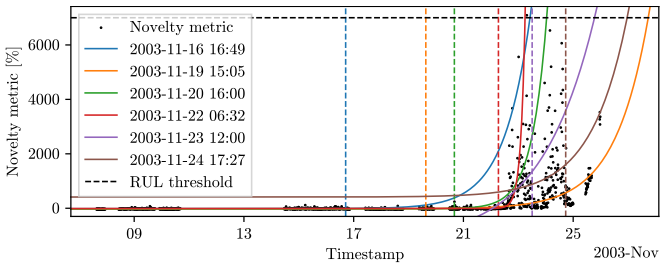

Fig. 5. Results of RUL predictions on the IMS dataset

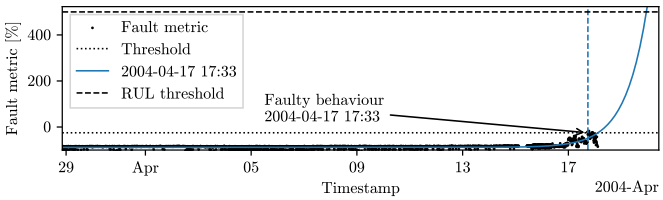

Fig. 6. Results of FD on the IMS dataset

The ND capability of the framework has been tested using all the UML models. The lead time (LT) elapsed between the ND event and the actual fault is used to compare the models. The results are compared in table I. The evolution of the NM, using a K-means model on the "Bearing 3x" signal of the test N°1, is shown in Fig. 4.

To perform PM, the framework has to estimate the Remaining Useful Life (RUL) of the system. The RUL is predicted computing when a fitted curve crosses a certain threshold. The type of curve to fit is configurable, in this work $y = a \cdot e^{b \cdot x} + c$ has been used. The results of the RUL predictions are shown in Fig. 5.

Since the second and third tests of the IMS dataset share the same kind of fault, the framework has been trained to perform FD with the faulty data of the second test and then evaluated on the third. Analogously to the ND, after the FM crosses a certain threshold, a warning is issued and RUL predictions are performed. The results of the FD are shown in Fig. 6.

B. Laboratory tests

Since the validation on the IMS dataset has shown that the K-means model is both the simplest and the most effective, it has been deployed on the Edge implementation. In the first phase, the framework gathers the data, extracts the features and sends them to a PC. The UML model is trained on the PC and then sent back to the Edge. After, the microcontroller autonomously evaluates the status of the maintained system. The structure of the Edge implementation is shown in Fig. 7.

Extensive tests have been performed on a laboratory shaker, simulating the vibrations generated by a generic mechanical system. The tests have been carried out to evaluate the sensitivity to variations in both amplitude and frequency

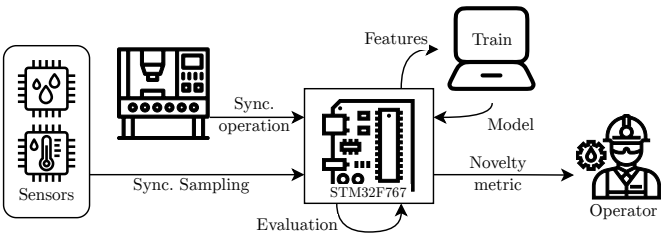

Fig. 7. Structure of the Edge implementation

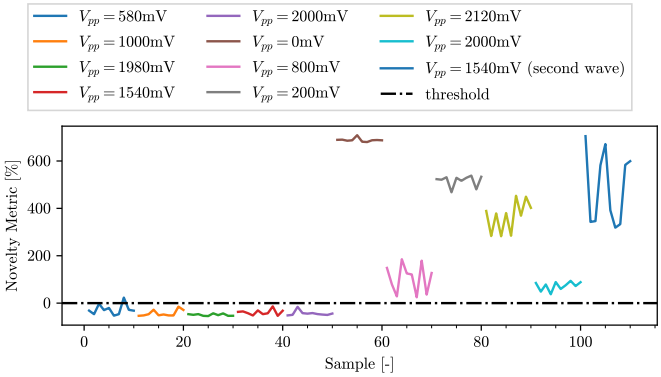

Fig. 8. Results of laboratory test on shaker vibrations

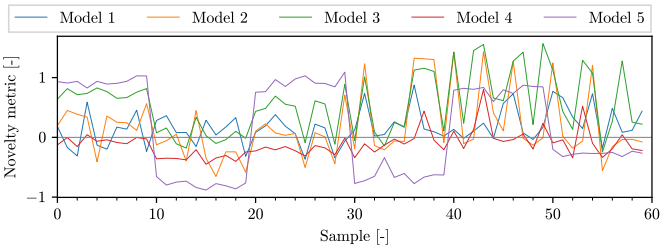

Fig. 9. Results of laboratory test on linear actuator acceleration profiles

content of the vibrations. In Fig. 8 the results of the ND are shown. The first five lines are novelty evaluations a repetition of the same signals used for training, and the remaining lines are the successful detections of the novelty.

A second series of tests has been performed mounting the accelerometer on a linear actuator. The axis of the accelerometer has been aligned with the direction of the movement of the actuator, to sense more the acceleration actuated rather than the vibrations. A set of predefined movement profiles has been used for training, while another set for testing. This series of tests exploited a high number of non-significant features. This is because the WPD gave high resolution on a wide range of frequency content, but the actuator excited only a few, and very low, frequencies. The remaining features were almost all noise, and the standardization procedure had the side effect of amplifying them to the same level as the significant ones. An effort has been made to fine-tune the model to reduce the impact of the noise (see Fig. 9, models 1 to 4). The best result, however, has been obtained by removing the less informative features and using a reduced feature space. The benefit of this approach is evident in Fig. 9, as “Model 5” shows a clear and sharp detection of the alternating pattern between known and unknown movement profiles.

V. CONCLUSION AND FUTURE WORK

The proposed solution has been proven effective in performing Novelty Detection in both public datasets and laboratory tests. The flexibility of the framework allows it to be easily adapted to different systems.

Some tests have shown that the standardization of the features is not always enough to make the clusters in the data space meaningful. In this case, in the PC implementation, Isolation Forest and Local Outlier Factor Models, being not cluster-based, have been tested and resulted as more effective. As future work, the deployment of these models on the Edge implementation would bring a significant improvement in the performance of the framework in some specific applications.