Master's Degree in Mechatronic Engineering

Politecnico di Torino

Master's Degree Thesis

UNSUPERVISED MACHINE LEARNING ALGORITHMS FOR EDGE NOVELTY DETECTION

Supervisors

Prof. Marcello CHIABERGE

Dott. Umberto ALBERTIN

Dott. Gianluca DARA

Candidate

Ariel PRIARONE

03 2024

Acknowledgements

ACKNOWLEDGMENTS

"HI"
Goofy, Google by Google

Abstract

Nowadays, the fourth industrial revolution is taking place, and it is characterized by the integration of Artificial Intelligence and the Internet of Things paradigm into factories. At the same time, in many industrial applications, the maintenance approach remained unchanged for decades. The main reason that holds back the evolution of the maintenance approach is the high costs of implementation of Condition-Based or Preventive maintenance strategies and a lack of knowledge about the modelling or behaviour of a failing system.

A framework that can be used for detecting a novelty behaviour (ND), a known fault (FD), and estimating the Remaining Useful Life (RUL) of the maintained system is proposed. To overcome the lack of models an Unsupervised Machine Learning (UML) approach is used and, on the other hand, to overcome the difficulties of implementation on different systems a modular and general-purpose framework has been developed.

A quite new frontier in the field of Predictive Maintenance is the implementation of the Artificial Intelligence application directly in the maintained system, this approach is called Edge Computing.

In this work, the framework has been developed to be run and tested on a PC using various Unsupervised Machine Learning algorithms, Using the Python language. The algorithms that appeared to maximize the performance-resources ratio have been then implemented in a microcontroller, using the C language. The edge implementation has been tested with laboratory experiments.

The proposed solution relies on the application of Machine Learning to features extracted by the time-domain signal by the framework itself. With this structure, the framework can be easily set up on a machine, read signals from sensors, extract features, and then train the models. The framework then switches to evaluation mode to perform ND, FD, and RUL predictions. After a Nd or FD event, the model can be refined performing a retraining using only the data that generated the event.

The considered features are settable for each sensor. The implementation of the feature extraction manages time-domain features (Mean, RMS, Standard Deviation, P2P, Skewness, Kurtosis) and frequency-domain features (The power of coefficients

of a Wavelet Packet Decomposition of any depth, or the FFT of the signal). The framework is designed to be easily extended to additional features and sensors.

The PC implementation is based on Software Agents that act autonomously on a common database. The following algorithms have been implemented to perform ND, FD, and RUL predictions: K-means, DBSCAN, Gaussian Mixture Models, One-Class Support Vector Machines, Isolation Forest and Local Outlier Factor. The K-means model was then chosen for the Edge implementation.

All the cited algorithms' performances have been compared on a dataset of bearings vibration published online by the Center for Intelligent Maintenance Systems (IMS) of the University of Cincinnati. The Edge implementation of the framework has been tested in laboratory experiments, using an accelerometer to measure the vibrations generated by an active shaker and, on a second set of tests, by a linear actuator.

The last refinement of the framework has been the implementation of additional feature scaling after the standardization, using a Random Forest model to refine the UML model. This mitigates the effect that noisy features have on the performance of the algorithms. This approach has been tested on the laboratory data.

Table of Contents

List of Tables

List of Figures

Chapter 1

Validation

This chapter is dedicated to the validation of the framework on real-world data. In **??**, the reference dataset [1] has been introduced. Firstly, the `python` implementation of the framework is validated on the IMS dataset several times with different configurations to show the flexibility of the framework, and try to find the best configuration for the dataset.

The first test in the IMS dataset is carried out with all the implemented machine learning models, then only the K-mean model is used in the following tests. In all tests an outlier filter has been implemented, so that the MLA will warn about the novelty behavior only if two consecutive snapshots are labeled as outliers. The number of consecutive snapshots is a parameter that can be adjusted in the framework settings.

Then, the edge computing implementation of the framework based on the K-means clustering is validated experimentally on a machine and with a laboratory shaker.

1.1 IMS dataset No.1 - Bearing 3x sensor

To start the validation, let's subdivide the test No.1 of the IMS dataset into *training* and *testing* datasets. The first 500 samples are used for training, and the remaining samples are used for testing.

For all the algorithms, the assumption about the system is that, even if the degradation is continuous, the system is surely healthy until 2003-11-07. The threshold for performing the ND is set conformingly to this assumption, for every model considered. Otherwise, the performance of any model could be artificially made as good as desired, by simply setting the threshold to a lower value.

The configuration file is set to use the data from the "bearing 3x" sensor, extracting all the time-domain and frequency-domain features described in **??**. The

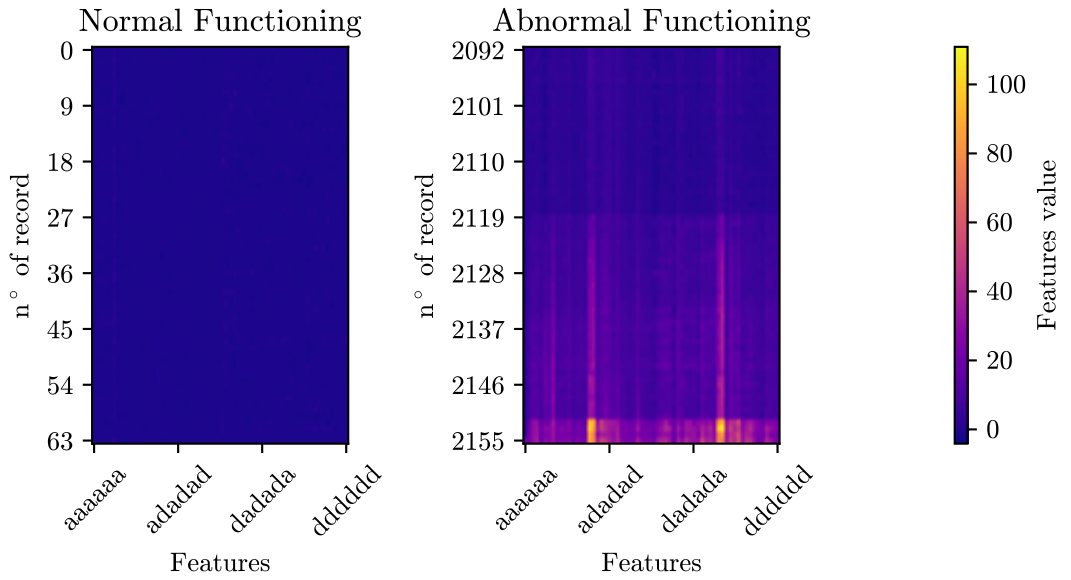

Figure 1.1: *Heatmap of the standardized features value for the test n°1 of IMS dataset*

training dataset is used to train the MLA to recognize the normal behavior of the bearing, and the testing dataset is used to validate the trained model. The **??** shows the parameters of test No.1 of the IMS dataset. For display purposes, the features are standardized, and the heatmap of the standardized features is shown in **figure 1.1** in normal and abnormal conditions.

The abstract version of the FiA has been used to extract the features from the dataset, creating all the snapshots in the set $\mathbf{S} = \{\boldsymbol{\mathcal{S}}_1, \boldsymbol{\mathcal{S}}_2, \ldots, \boldsymbol{\mathcal{S}}_{500}\}$. These snapshots are stored in the *unconsumed* collection of the database.

1.1.1 Training - K-means

Using the commands of the CLI, the training procedure has been launched:

```
C:/Users/JohnSmith/Code/framework> python ./MASTER.py
↪  run-feature-agent
C:/Users/JohnSmith/Code/framework> python ./MASTER.py
↪  run-machine-learning-agent novelty train
```

where the first command runs the FiA and the second one runs an "healthy" instance of the MLA in training mode. At this point, the MLA asks the user to move the snapshots from the *unconsumed* to the *healthy* collection, since the *healthy* collection is empty. After the confirmation, the MLA starts the training with a

different number of clusters, and outputs the scoring in the form of silhouette and inertia scores. The results are shown in **figure 1.2** and **figure 1.3**. The user can confirm that the best number of clusters is 2, as the silhouette score is the highest and the inertia score is at the POF point, or insert another number of clusters, remembering that it is best to overestimate the number of clusters to increase the system sensitivity, as discussed in **??**.

In this case, the number of clusters has been set to 2, so that the MLA saves the model trained with $n = 2$ into the database. Even if the feature space has high dimensionality, the agent plot to the user also a scatter plot of a subset of features of the training dataset, to have a visual feedback of the clustering, as shown in **figure 1.4**, where the points are the snapshots, the crosses are the centroids and the colors represent the assigned cluster. We can observe that selecting 2 as the number of clusters is adequate and that the projections of the clusters' shapes on some planes are not perfectly spherical but, at least, they are not too elongated. This is a good sign for the K-means algorithm, as discussed in **??**.

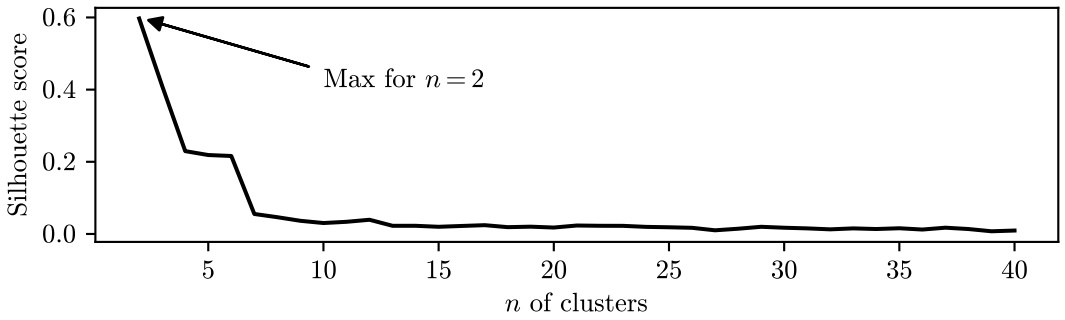

Figure 1.2: *Silhouette score for clustering the test n°1 of IMS dataset (K-means)*

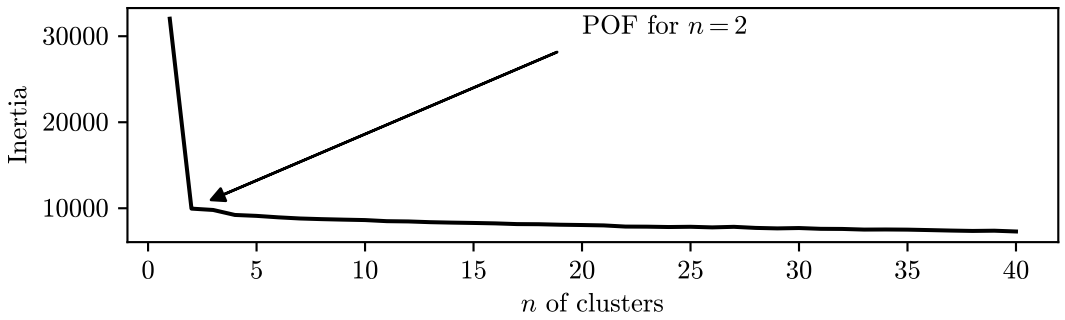

Figure 1.3: *Inertia score for clustering the test n°1 of IMS dataset (K-means)*

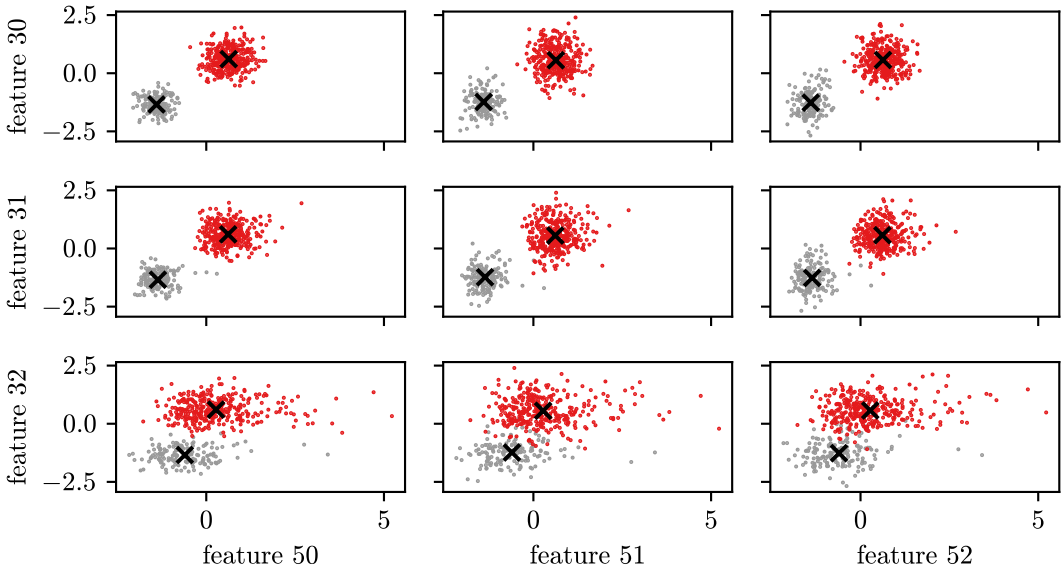

Figure 1.4: *Scatterplot of training snapshot for the test n°1 of IMS dataset*

1.1.2 ND Validation - K-means

Using the validation partition of the dataset, it is possible to set the MLA in *evaluate* mode. The FiA uses the validation partition and fills the *raw* collection with the time-series. The FA extract the features and continuously fill the *unconsumed* collection with the snapshots. The MLA evaluates the snapshots according to **??** and plots the result, as well as generating a warning if the novelty metric is greater than a certain threshold (in this case 50%, but it is configurable in the usual `.yaml` file). The results are shown in **figure 1.5**, where we can see that the framework detects the novelty quite early, at 2003-11-16 07:46, while the dataset authors, declared the test finished because of bearing defects (not catastrophic failures) at 2003-11-25 23:40. The comparison of the margin of early detection for different algorithms will be resumed later.

In **figure 1.6**, a detailed view of the ND metric becoming consistently greater than the threshold is shown.

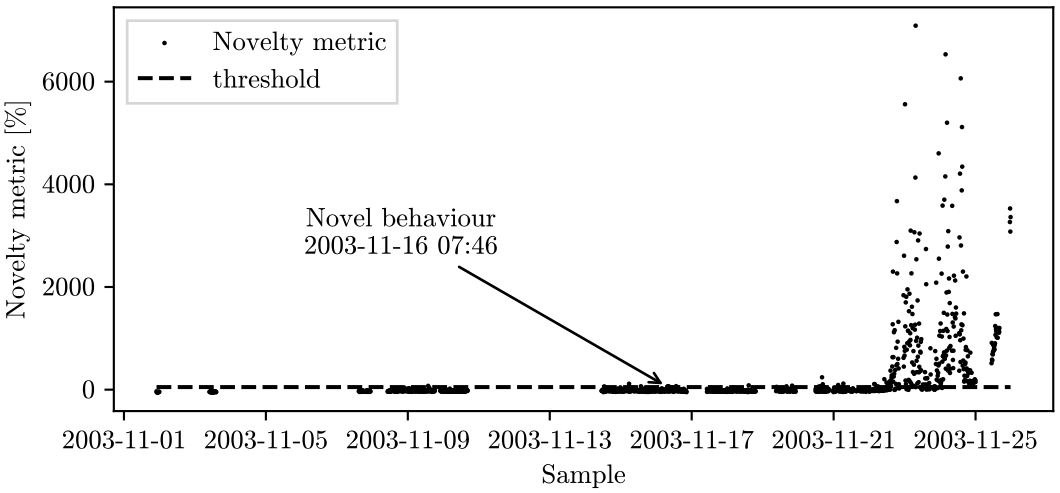

Figure 1.5: *Results of ND for the test n°1 of IMS dataset (K-means)*

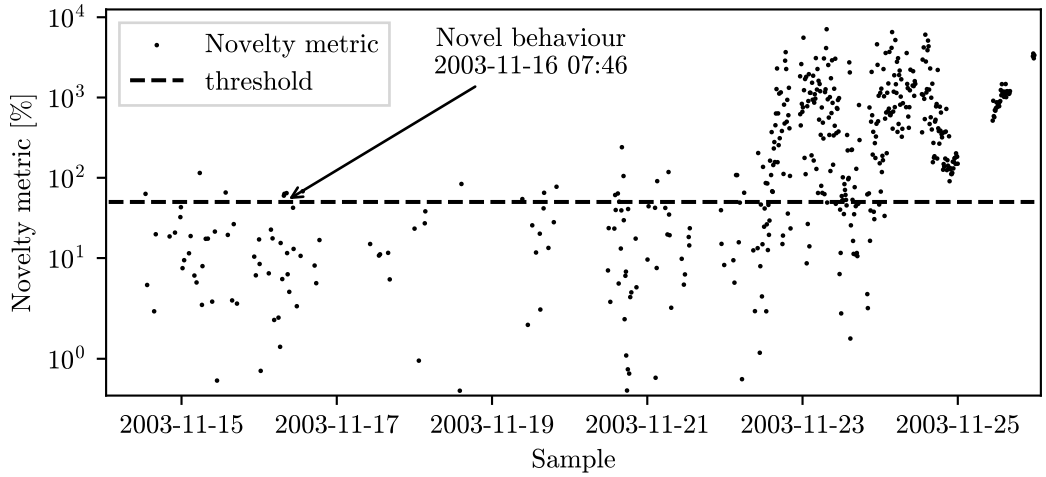

Figure 1.6: *Results of ND for the test n°1 of IMS dataset (K-means) - detailed view*

1.1.3 Training - DBSCAN

Using the same partition of the dataset as for the K-means training, we can train a DBSCAN model. In this case, the silhouette score has to be used to select a suitable value of the radius ε. As shown in **figure 1.7**, the optimal value is 8, which corresponds correctly to the generation of two clusters.

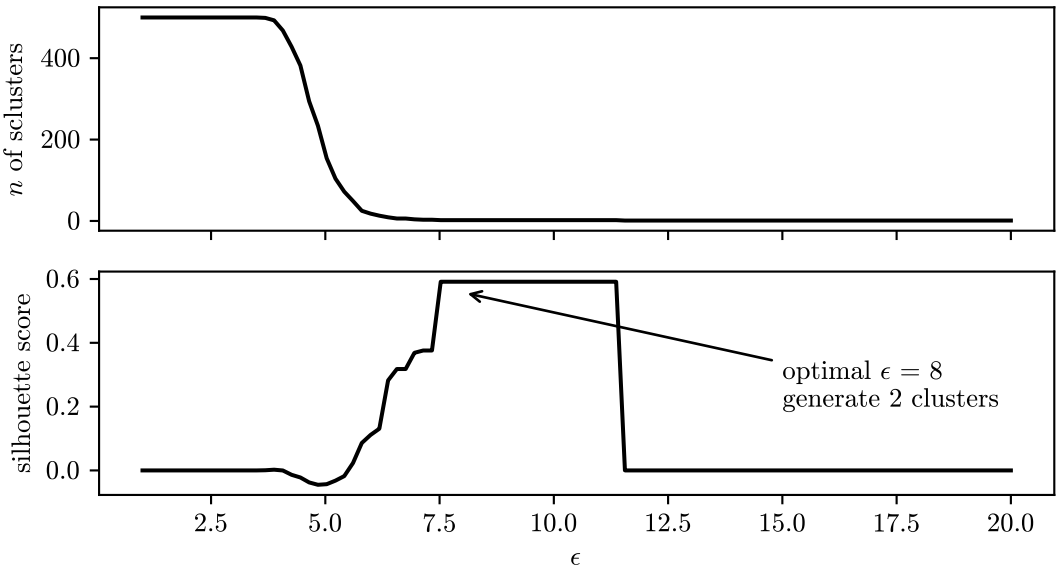

Figure 1.7: *Silhouette score for clustering the test n°1 of IMS dataset (DBSCAN)*

1.1.4 ND Validation - DBSCAN

As it has been done for the K-means, the validation partition of the dataset is now used for performing ND with the DBSCAN model, as described in **??**. The result is shown in **figure 1.8**, where we can see that the DBSCAN model detects the novelty at 2003-11-22 15:06, that is quite early, but not as early as the K-means model. This is because the metric generated by the DBSCAN model has a greater variance so, instead of increasing consistently, it overshoots the threshold quite before this time but fails to consistently stay above the threshold.

1.1.5 Training - GMM

Let's now try with the GMM model. The metric for selecting the number of clusters is now the BIC and the AIC, as shown in **figure 1.9**. The two metrics diverge but, as discussed in **??**, the AIC tends to perform better. In this case, minimizing the AIC leads to select 25 as the number of clusters, which is much more than what was selected with the K-means, but still a reasonable choice, also considering that the GMM is a soft clustering algorithm and that we are using the density as a metric to perform ND.

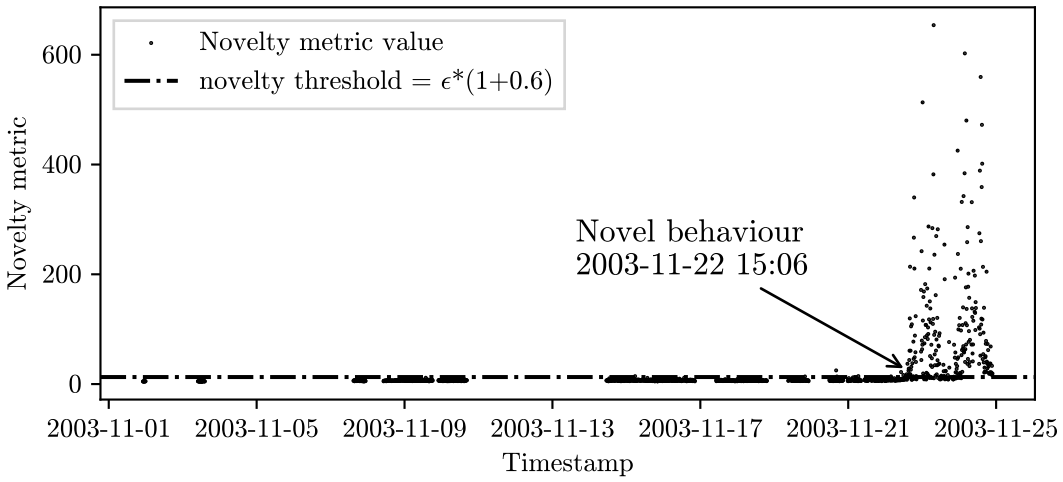

Figure 1.8: *Results of ND for the test n°1 of IMS dataset (DBSCAN)*

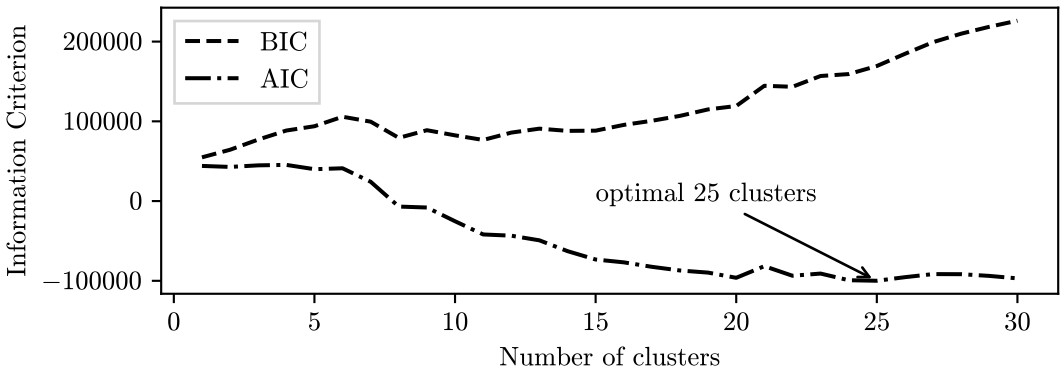

Figure 1.9: *BIC and AIC for clustering the test n°1 of IMS dataset (GMM)*

1.1.6 ND Validation - GMM

The validation partition of the dataset is now used for performing ND with the GMM model. The result is shown in **figure 1.10**, where we can see that the GMM model detects the novelty at 2003-11-22 03:47. The considerations about this result are the same as for the DBSCAN model, and in fact, the timestamp of the detection event is really close to the one obtained with DBSCAN. In **figure 1.10**, the metric (density value) appears in colored dots, as each color represents the cluster to which the snapshot has been assigned.

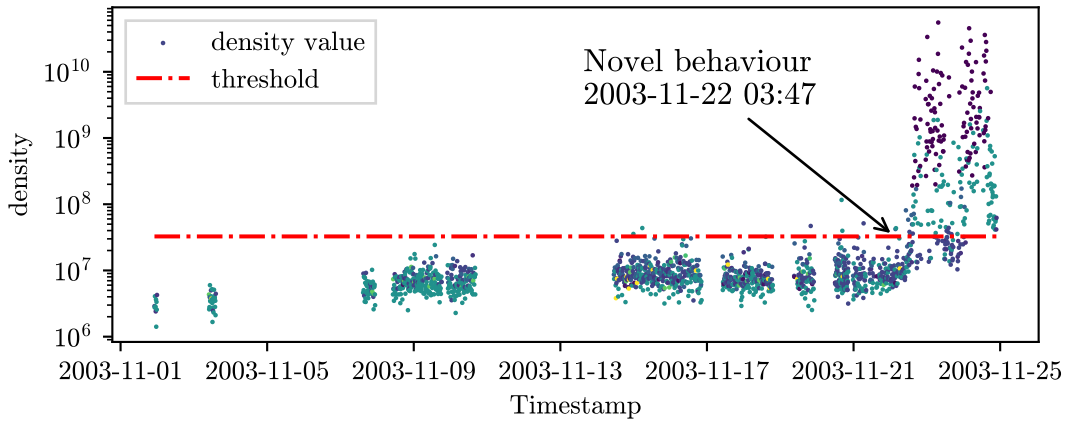

Figure 1.10: *Results of ND for the test n°1 of IMS dataset (GMM)*

1.1.7 ND Validation - Bayesian GMM

The other Gaussian model is the BGMM, since this approach is totally unsupervised, only the validation results are reported here. The result is shown in **figure 1.11**, where we can see that the BGMM model detects the novelty around the same time as the GMM model, at 2003-11-22 03:45.

In both GMM and BGMM the metric (density value) spans a lot of decades, so the plots are done on a logarithmic scale.

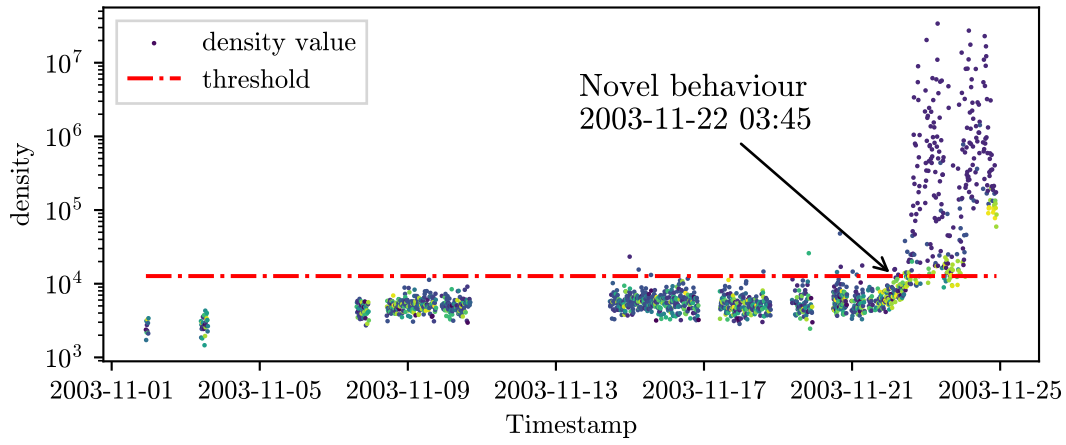

Figure 1.11: *Results of ND for the test n°1 of IMS dataset (BGMM)*

1.1.8 ND Validation - ν-SVM

The next algorithm to test is the ν-SVM. Again, this is totally unsupervised, so only the validation results are reported here. The novelty metric evolution over time is shown in **figure 1.12**, where we can see that the ν-SVM model detects the novelty at 2003-11-22 14:56, which is comparable with the DBSCAN and GMM models.

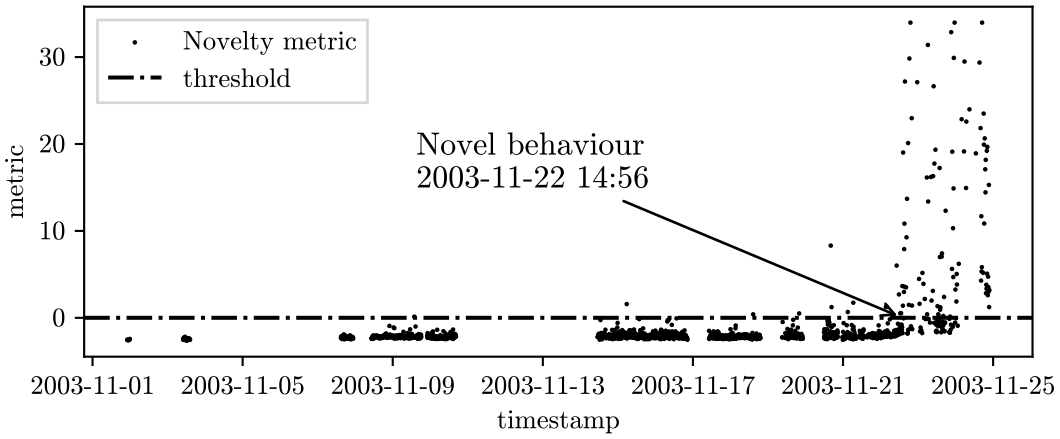

Figure 1.12: *Results of ND for the test n°1 of IMS dataset (ν-SVM)*

1.1.9 ND Validation - iForest

The second last technique to test is the one based on the iForest model. The result is shown in **figure 1.13**, where we can see that the iForest model detects the novelty at 2003-11-16 10:08:46, that is a good result comparable with the K-means model. The problem with the metric of the iForest is that it increases a lot the variance around the ND event, but the mean does not increase consistently, so a lot of snapshots are discarded as outliers, before the ND event.

This is, in my opinion, a promising approach. With these settings a lot of snapshots are discarded as outliers, but with a different outlier filter, based on the percentage of novelty samples in a window, the iForest model could perform even better.

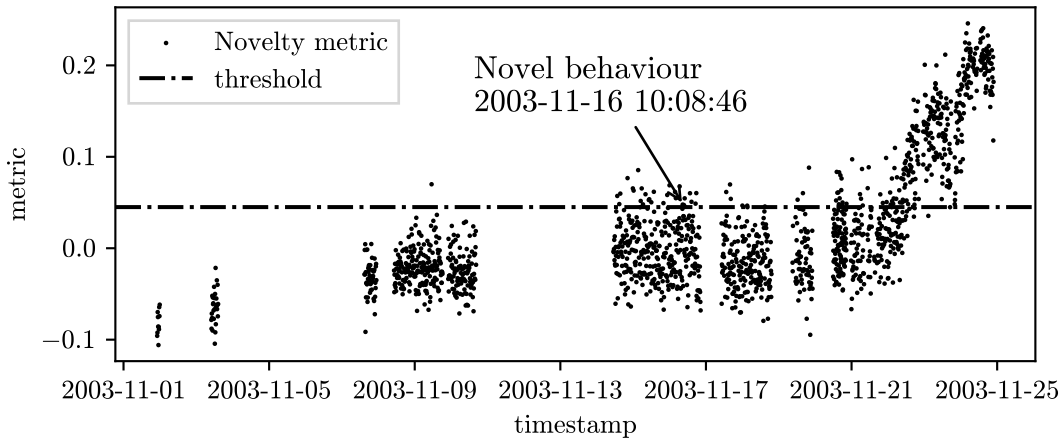

Figure 1.13: *Results of ND for the test n°1 of IMS dataset (iForest)*

1.1.10 ND Validation - LOF

The last algorithm to test is the LOF. The result is shown in **figure 1.14**, where we can see that the LOF model detects the novelty at 2003-11-16 07:49, which is a good result comparable with the K-means model. It doesn't have the same problem as the iForest, as there aren't as many discarded snapshots before the ND event.

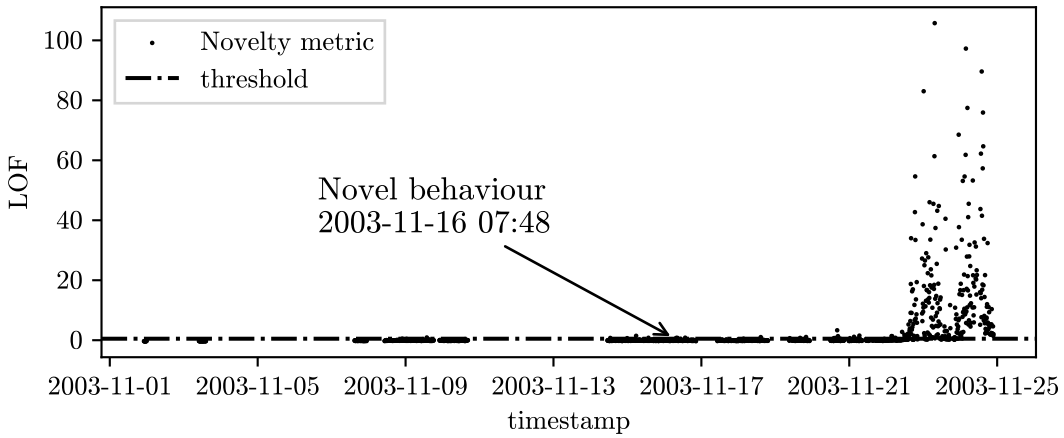

Figure 1.14: *Results of ND for the test n°1 of IMS dataset (LOF)*

1.1.11 Comparison of the results

Comparison between the models

Table 1.1: *Comparision of the results for the test n°1 of IMS dataset.*

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

In **table 1.1**, the results of all the previous tests are resumed, together with the result of performing ND without any machine learning algorithm, but just setting a threshold on the P2P value of the time-series, as it was previously shown in **??**. This last basic approach detects the novelty around the afternoon of 2003-11-22.

The ν-SVM and the DBSCAN models are not performing much better than not even using machine learning (at least on this dataset signal). The GMM and BGMM models are performing slightly better, but the margin is so low that the result may be biased by the threshold setting. The iForest, LOF and K-means models are performing better, they are all very close to detecting the novelty, around 14000 min = 9.7 days before the end of the test. The K-means model is the one performing the best, but just slightly better than the iForest and LOF models so, again, this small difference may not be significant. However, as discussed in the previous chapters, the K-means model will be used in the rest of the work, as it is also the most simple and interpretable model.

Comparison with another approach

As anticipated in the **??**, about the State of the Art, the signal of the same bearing (Bearing 3x) of this same test has been used in [2]. In their research, the authors used a different approach, based on regression, and obtained the result reported in **??**

Comment about the comparison

Every system that outputs a warning based on a trigger on a threshold is highly sensitive to the value of the threshold itself. This means that the comparison of the results is not straightforward, and quite opinable, because selecting a low threshold will make almost every system trigger earlier. The measure to take into consideration, in my opinion, is how many false positives are generated if the threshold is lowered, and how small the variance of the metric is. An high variance, on this dataset, means that the system is very sensitive while evaluating quite similar signals.

1.1.12 RUL Predictions validation - K-means

After the ND event, the MLA start predicting the future evolution of the novelty metric, and it superimpose the prediction curve to the same plot displayed to the user. The fitting procedure is the closed form solution of the LS problem applied to an exponential curve of **??**, as described in **??**. The samples used for the regression of the curve are the last 230 before the current one. This parameter of the framework is configurable in the `.yaml` file.

Some good predictions are shown in **figure 1.15**. The RUL is the difference between the intercept of the prediction curve and another threshold, higher than the one used for ND, and the current time. In the figure, the blue line is a prediction made just a few hours after the ND event (the vertical dashed line marks the time of the prediction). The same concept applies to the other predictions performed in later times.

In some circumstances, the novelty metric starts decreasing slightly, on average, as can be seen around 2003-11-21. In this case, if the novelty metric has this behavior for several snapshots (≈ 230), the fitted curve will be a decreasing exponential, as shown in **figure 1.16**.

If this situation occurs, the intercept with the threshold does not exist, and the RUL prediction fails, so the interpretation of the RUL is left to the user. In some cases, the defect in a system can "self-heal" (for example a crack in a bearing can be polished with the use [3]). If this behavior is possible for the system, this situation can be interpreted as a sign that the system is going to return to normality. Otherwise, the user can retain the previous RUL prediction as the RUL of the system.

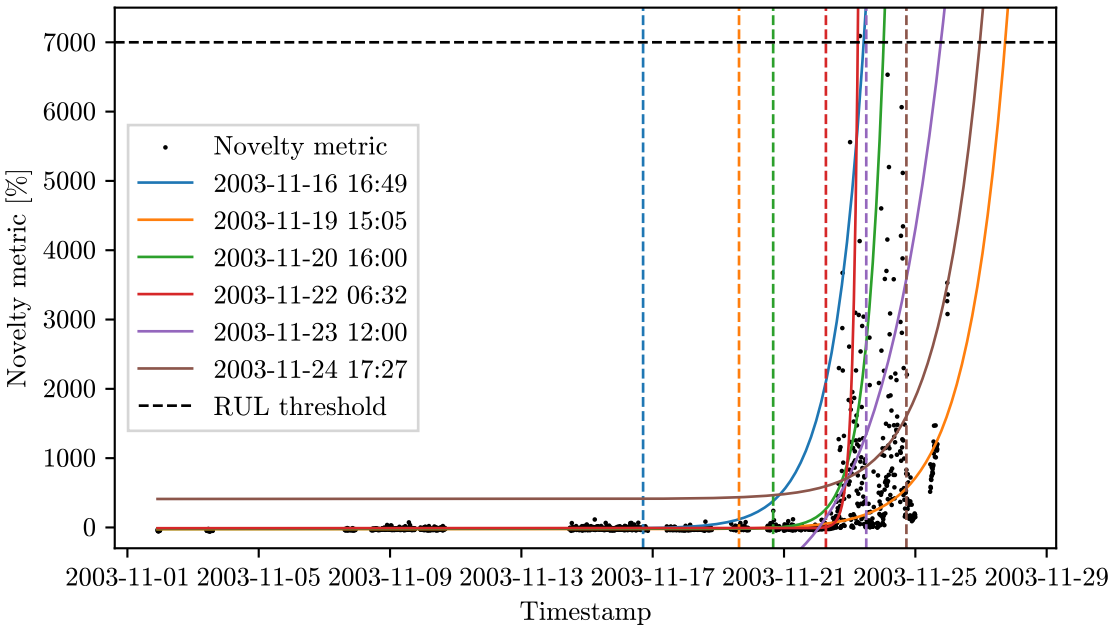

Figure 1.15: *RUL prediction at different instants after the ND event. The dashed lines are the instants of the predictions corresponding to the same color solid line prediction.*

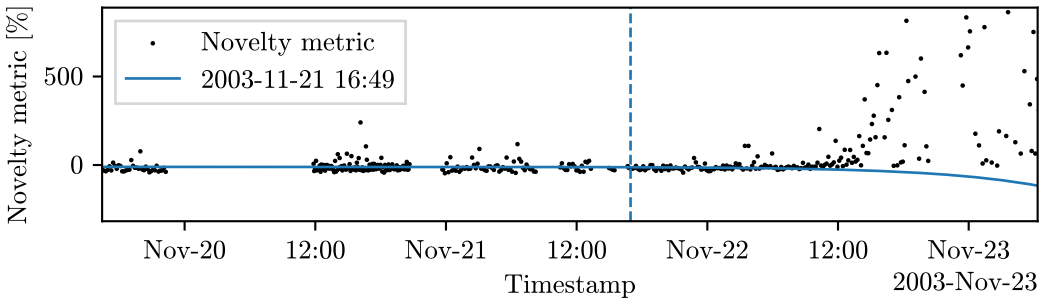

Figure 1.16: *Failed RUL prediction.*

1.1.13 Retraining, evaluating and predicting after ND event

If the user, after the ND event, performs an investigation that leads to the belief that the system is still healthy, the user can turn the MLA in *retrain* mode. In this case, the snapshots that are in the quarantine collection, are moved to the healthy collection (faulty if the instance is for FD and the investigation reveals that the fault is real). The model is then retrained with the new data with the same procedure used for the first training (silhouette and inertia scores are computed

and the user is asked to confirm the number of clusters).

Let's investigate what would have happened if the user declared the system healthy at 2003-11-23 00:00, in the previous scenario of "Bearing 3 x" signal in the IMS dataset. The MLA suggests that the best number of clusters is still two, so it has been retrained with the new data. The result of the updated model performing ND is shown in **figure 1.17**. The predictions of the RUL are shown in **figure 1.18**.

This test shows that in an increasingly decaying system retrained with data very close to the fault condition the MLA is able to detect the fault again. This comes at the cost of a later detection, and the first predictions after the ND event are not as good as the previous ones. Anyway, the RUL predictions still become more accurate as time passes, and the RUL prediction at the end of the test is still quite accurate (on the same day of the event).

Another consideration is about the RUL threshold: since the model has been retrained with "worse" data, the threshold for the RUL prediction should be set to a lower value, because now the clusters are either more in quantity, distorted or bigger, so it is unlikely that the novelty metric can still reach the same high values estimated before the retraining.

An intuition about why the sensitivity of the system is reduced after the retraining comes by examining the scatter plot of the snapshots in the feature space, shown in **figure 1.19**, where all the snapshots extracted from the dataset are displayed. The clusters are more elongated and much bigger. These shapes arise gradually from the original ones of **figure 1.4** so, by performing a retrain, both the effect of producing bigger clusters and one of the clusters being much more elongated play a role in reducing the sensitivity of the MLA.

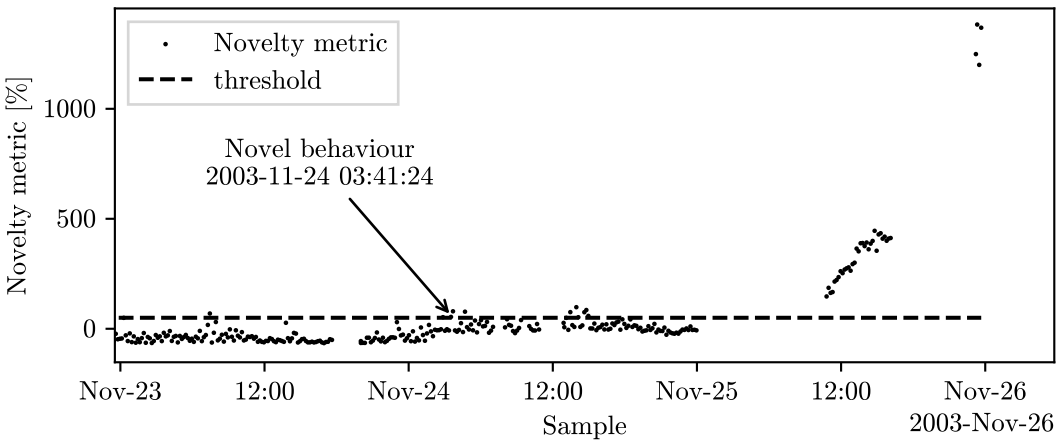

Figure 1.17: *Results of ND for the test n°1 of IMS dataset (K-means) - retrained model*

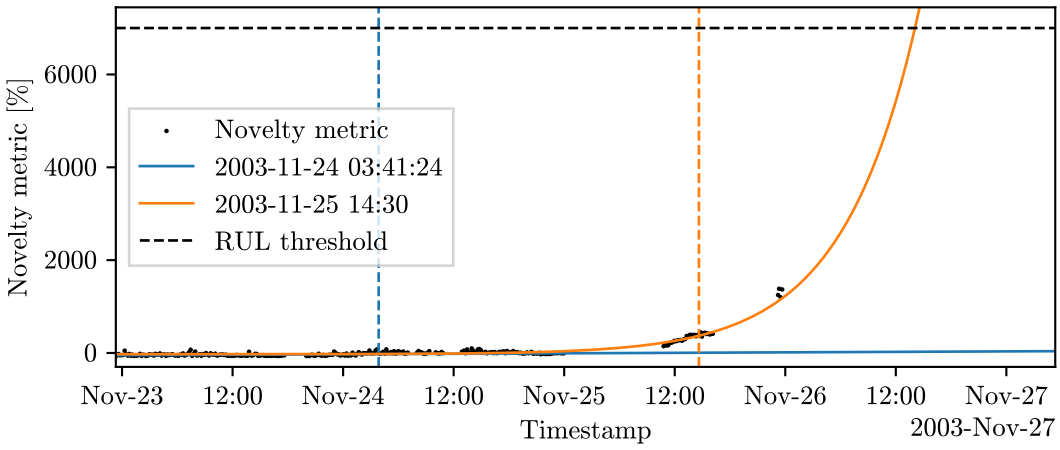

Figure 1.18: *RUL prediction at different instants after the ND event. The dashed lines are the instants of the predictions corresponding to the same color solid line prediction.*

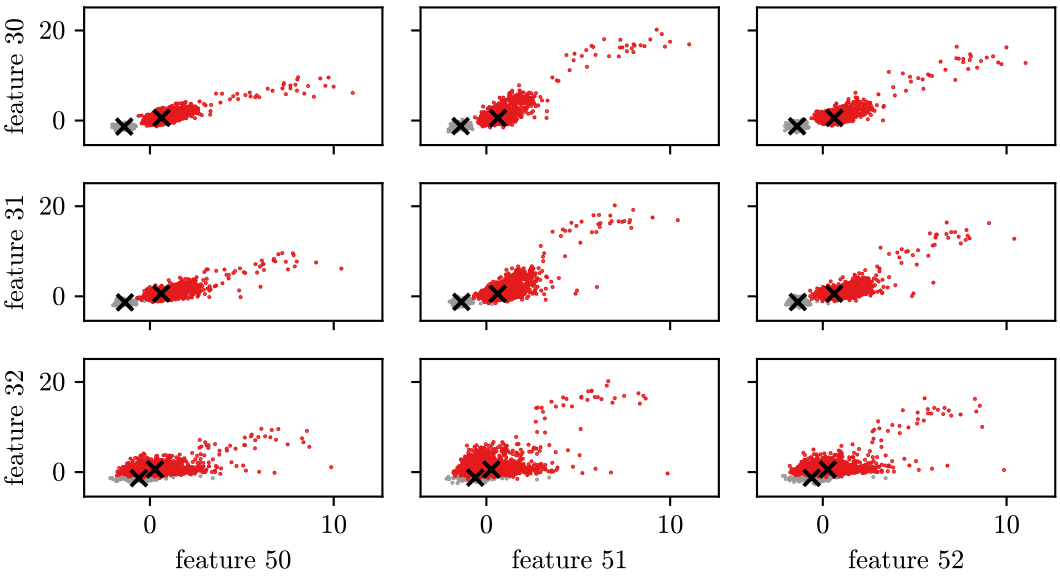

Figure 1.19: *Scatterplot of all the snapshot for the test n°1 of IMS dataset*

1.2 IMS dataset No.1 - All sensors

In the previous section, an extensive test of the framework has been performed on a single signal from the dataset (Bearing 3 x). So the warning given by the MLA was

indicating a problem in a specific component of the maintained system. Let's now test a configuration that takes into account all the signals of the dataset, so all eight signals from the four bearings are used for feature extraction. This configuration should be able to detect a generic novel behavior of the system or, better, a situation in which the system is abnormal as a whole (the signals may be normal but the combination of them may be abnormal). In this case, the configuration file has been set to use all the time-domain and all the frequency-domain features, and the MLA has been trained with the same procedure as before.

1.3 IMS dataset No.2 - Bearing 3x sensor

1.4 IMS dataset No.3 - Bearing 3x sensor

1.5 Experiments on a laboratory shaker - Test 1

After the PC implementation of the framework has been tested widely on the IMS dataset, the edge computing implementation had to be validated experimentally. The first test was done with a laboratory shaker, which is basically a powerful active speaker with a really wide band that can be attached with a bolt to a structure, to vibrate it.

In this case, an accelerometer, whose key specifications are shown in **table 1.2**, was used to capture the vibration signal. The accelerometer was attached to the shaker, with a custom 3D-printed fixture. This first test has the scope of checking the capability of the edge computing implementation to detect a new low amplitude harmonic in the signal. The signal is generated as a `.wav` file and fed to the shaker by a player. Both the input of the shaker and the output of the accelerometer were monitored with a digital oscilloscope. The setup is shown in **figure 1.20**.

Table 1.2: *Specifications of the ADXL335 Accelerometer*

Parameter	**Value**
Supply Voltage	1.8V to 3.6V
Sensing Range	±3g
Sensitivity	300 mV/g
Bandwidth	0.5 Hz to 1600 Hz
Output Type	Analog
Output Voltage Range	0V to V_{CC}
Operating Temperature	-40°C to +85°C
Package	3mm × 5mm × 1mm

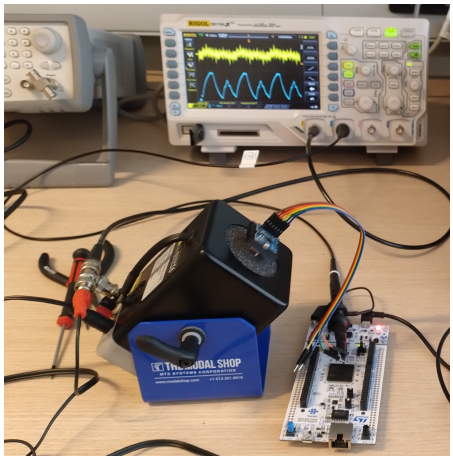

Figure 1.20: *Setup of the shaker tests.*

Table 1.3: *Harmonic coefficients for the shaker test. Wave 1 and Wave 2 are training signals, and Harmonic Injection is the signal to be detected.*

Signal Name	**Harmonic frequency** [Hz]						**Amplitude** $[\text{mV}]_{pp}$
	30	70	100	300	800	1400	
Wave 1	0.1	1.0	1.0	-	1.0	1.0	1000
Wave 2	0.1	0.8	1.0	-	3.0	0.6	1000
Harmonic Injection	0.1	1.0	1.0	0.1	1.0	1.0	1000

1.5.1 Training and evaluating

The framework was firstly set to gather the data, extract the features according to the configuration, and send the data to the PC for training. The training signals are two waves with different harmonic content and the test signal is very similar to one used for training, except for the presence of an additional harmonic with a small amplitude. The train and test signals harmonic content is reported in **table 1.3**. The amplitude of the vibration has been tuned at each test to ensure that the microcontroller was reading a signal of 1V_{pp}. The amplitude of the signals has been kept constant to test the capability of the framework to detect the frequency content of the signal in the feature extraction phase. The waveform of the test signals is shown with the one of one training signal in **figure 1.21**, to show the similarity of the two signals.

The setting of the framework can be resumed as follows:

- 67 features extracted from the signal ($2^6 = 64$ features from the wavelet decomposition, mean, P2P, and RMS);

- 110 samples for training for each signal.
- sampling frequency of 5kHz, for 1s of acquisition.

After the data gathering was completed, the training was done on the PC, as usual. The silhouette score correctly suggested 2 as the best number of clusters to be used. The PC part of the framework outputs the `model.h` file directly in the embedded project folder, so just a new upload of the firmware was needed to test the detection. The microcontroller was then set in *evaluation* mode and both the two training signals and the test signal were fed to the shaker.

1.5.2 Results

The result of the novelty detection is shown in **figure 1.22**. The result is consistent with the expected outcome, as the training signals produced a negative novelty metric, while the test signal produced a positive (and quite high) novelty metric. The spectrum of the signals used is shown graphically in **figure 1.23**.

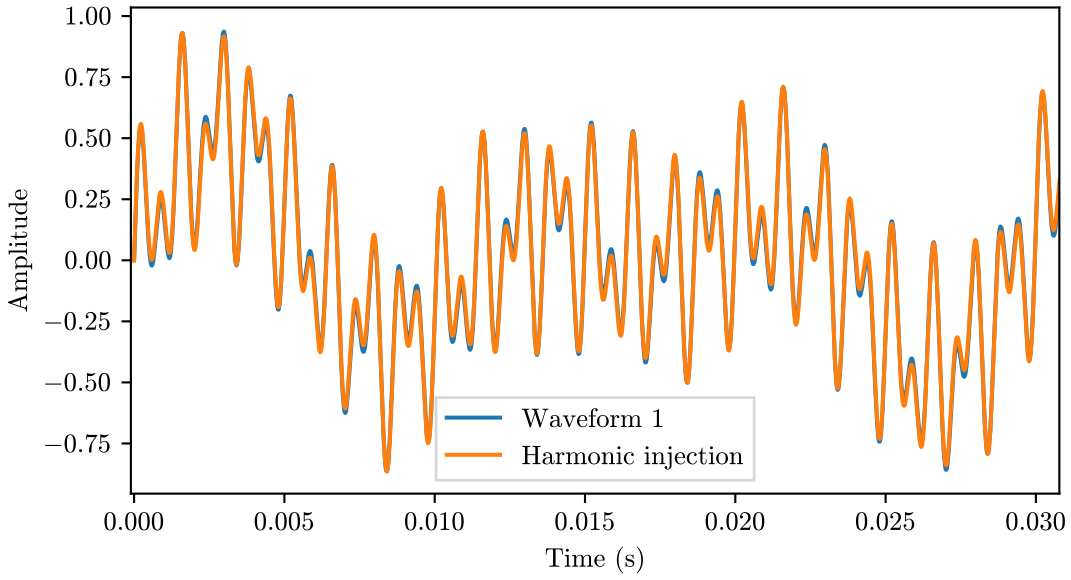

Figure 1.21: *Waveform comparison of the shaker test.*

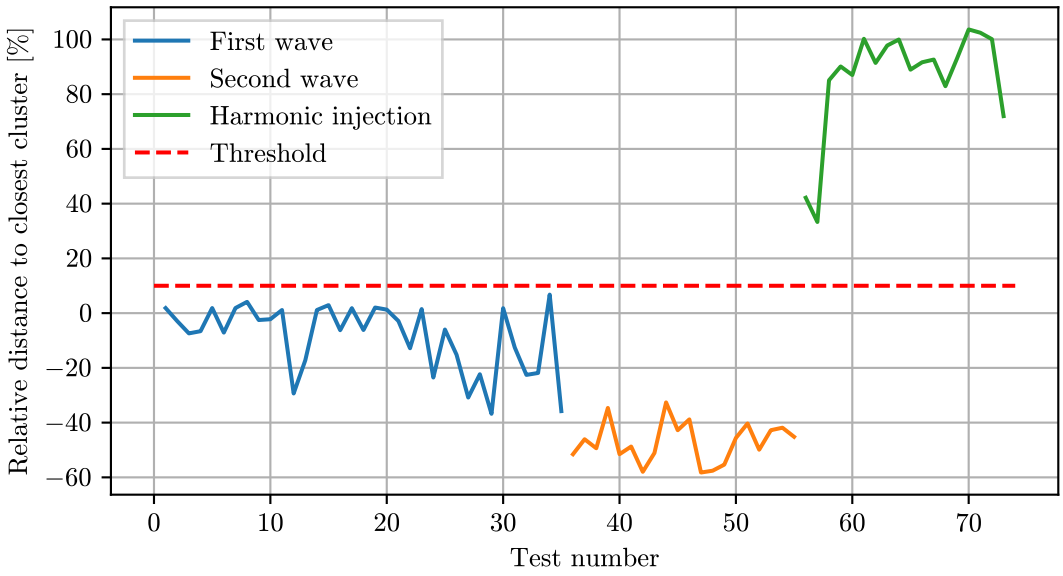

Figure 1.22: *Novelty detection result*

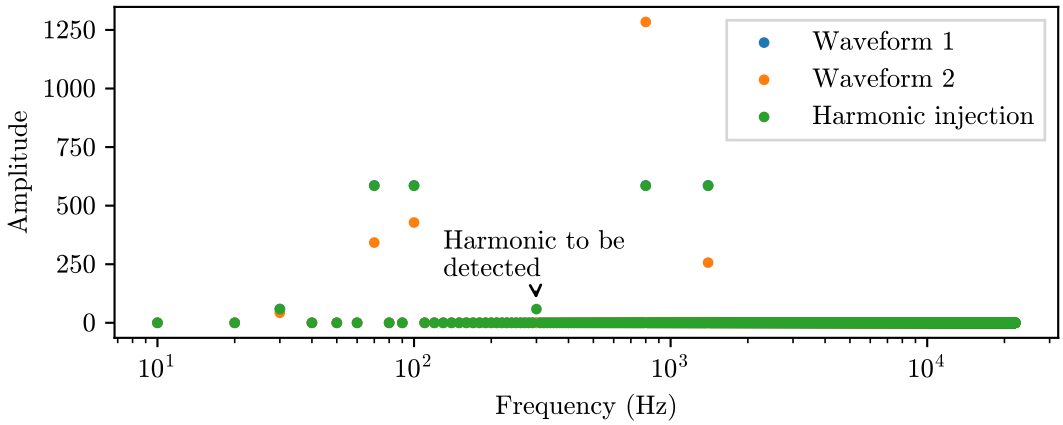

Figure 1.23: *Spectrum of the waveforms.*

1.6 Experiments on a laboratory shaker - Test 2

In the previous section, the first test on the shaker was presented. The test has shown the capability of the framework to detect unknown harmonics. A second test was done to evaluate the capability to detect time-domain variations and the effect of reducing the frequency resolution.

1.6.1 Training and evaluating

This new configuration has been set to use only 4 frequency-domain features and the same 3 time-domain features of the previous test, for a total of 7 features. The signal used for training and testing is resumed in **table 1.4**. The set is composed of the same signal at different amplitudes used for training and testing, plus another signal with different frequency content but the same amplitude as a training signal used for testing.

The training has been carried out in the same way as the previous test, the training of the K-means model has been done with 4 clusters, and loaded on the microcontroller.

Table 1.4: *Parameters of the second shaker test.*

Harmonic frequency [Hz]					**Amplitude** [mV_{pp}]	**No. of snapshots**	
10	30	60	70	100		Train	Test
-	0.1	-	1.0	1.0	580	100	10
-	0.1	-	1.0	1.0	1000	100	10
-	0.1	-	1.0	1.0	1980	100	10
-	0.1	-	1.0	1.0	1540	100	10
-	0.1	-	1.0	1.0	2000	-	20
-	0.1	-	1.0	1.0	0	-	10
-	0.1	-	1.0	1.0	800	-	10
-	0.1	-	1.0	1.0	200	-	10
-	0.1	-	1.0	1.0	1220	-	10
1.0	1.0	0.1	-	-	1540	-	10

1.6.2 Results

The result of the novelty detection is shown in **figure 1.24**. The first 4 lines have been correctly identified as normal, as they were in fact a repetition of the training signals. Then the purple and cyan line in the figure is the same training signal, but 20 mV higher in amplitude w.r.t. the training signal. The novelty metric overshoots the threshold in 5 samples out of 20. An increase of 2% in amplitude generates the ND event 25% of the times can be observed with this signal.

The brown, grey and light-green lines are the same signal, but with a bigger difference in amplitude w.r.t. the training signal. All the snapshots of these signals correctly generated a novelty metric above the threshold. The blue line is the signal with a different frequency content, and it has been correctly identified as a novelty event, this is the confirmation that even with just 4 frequency bins, the wavelet

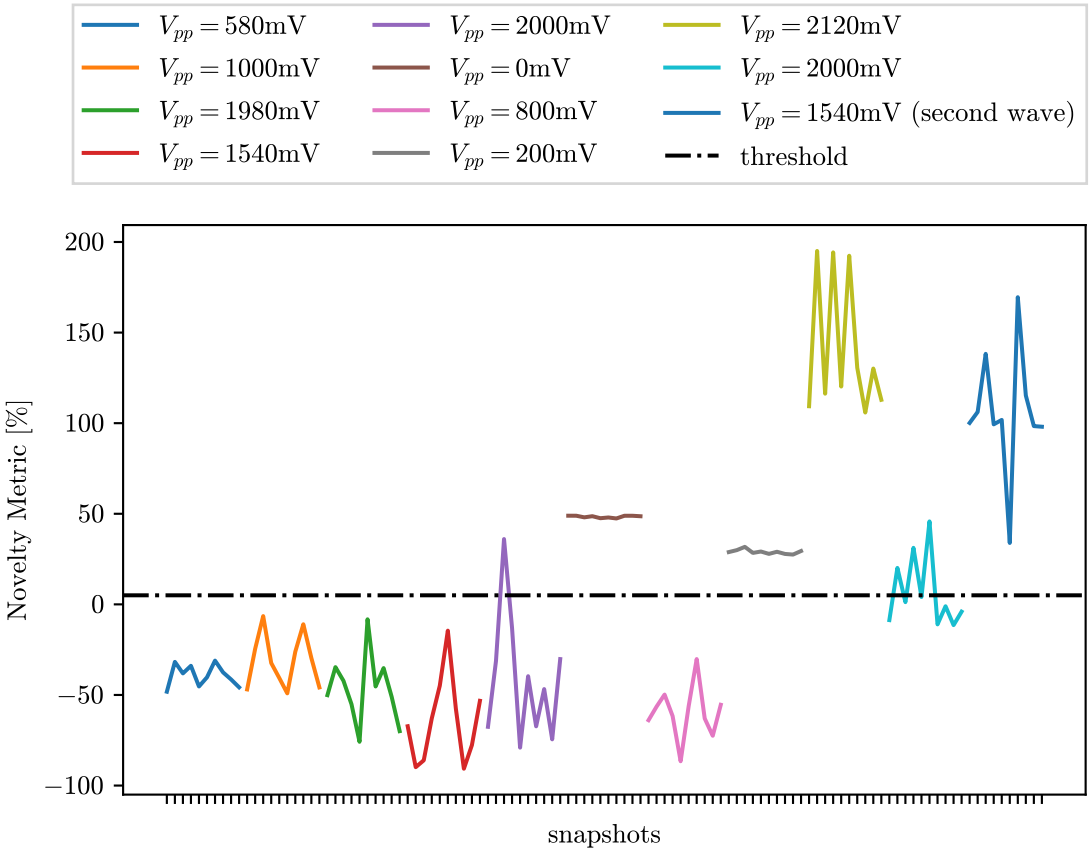

Figure 1.24: *Novelty detection result*

decomposition is still generating features that are informative.

The pink line is the test signal with amplitude of 800mV. It's evident that the novelty metric is below the threshold, and the signal has been classified as normal even if it is not in the training dataset. Let's investigate how this happened. The first consideration is that the 800mV amplitude is quite tight to both the 1000mV and 580mV signals used for training. Moreover, in this case, the total number of features is just 7. This allows plotting all the features against each other, to see why the ND event has not been detected. In **figure 1.25** the scatter plot of the features is shown. It's evident that, in this environment, even performing the standardization of the features, the clusters are still very elongated, resembling almost a line. To fit an elongated cluster in a hypersphere, it is inevitable that in some sections, the hypersphere will not closely surround the cluster, leaving "space" for false negative results. Another problem is that the k-means algorithm tends to split long clusters. In the figure, the red dots are the false negative results, and the gray shades are the hypersphere projection on the considered features plane. The

black dots are the training data. The effect of the elongated clusters is particularly evident in the plot of the "Feature 3" against "Feature 2", where the red dots are in between two clusters, that are modeled as one. On the other hand, looking at the plot of "Feature 1" against "Feature 4", a very long cluster has been split in two. This is an example of exploiting the limitations of the k-means algorithm anticipated in **??**. For completness, in **figure 1.25**, also the true positive results are shown, as magenta dots.

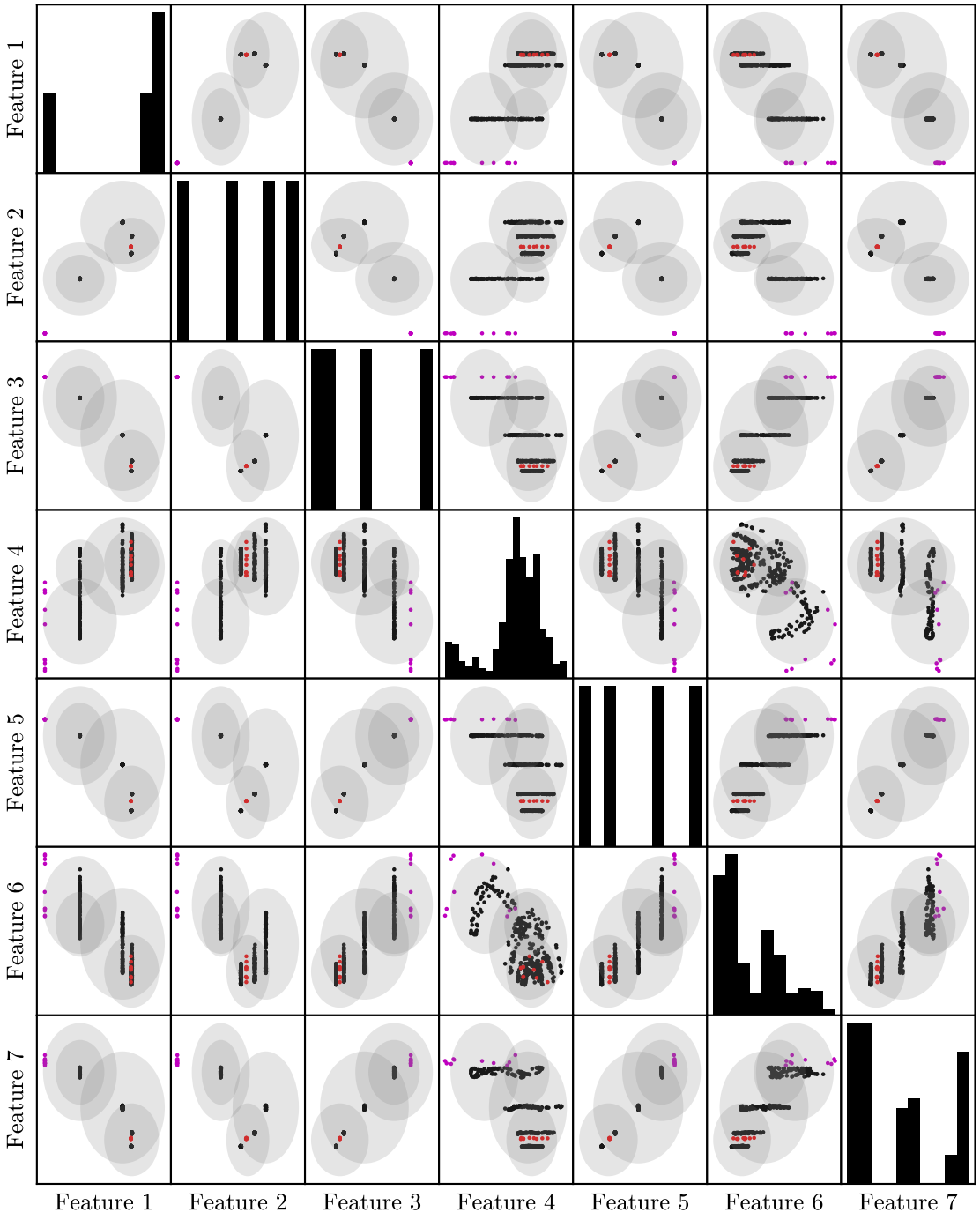

Figure 1.25: *False Negative and True Positive results. On the diagonal, there is a histogram of the feature values. The off-diagonal plots are the scatter plots of the features. The shades are the projection of the clusters on the considered plane. (Red: False Negative, Magenta: True Positive, Black: training data)*

1.6.3 Possible improvements

The environment of this test is very challenging for the k-means algorithm. As discussed in **??**, there are algorithms that are not affected by the clusters' shapes. The candidate algorithms that may perform better in this situation are the LOF, the iForest and DBSCAN. A future work could be to implement these algorithms in the edge computing framework, despite being more demanding in computational power and memory, and test them in this environment.

As proof of concept, the LOF implementation in `python` has been used to perform ND on the same dataset used in this section in edge computing. The results are reported in **??**. The LOF algorithm has been able to correctly identify all the ND events, even the signals with just 20mV variation from the training dataset, and the 800mV signal that was problematic for the K-means. The LOF, however, generated a false positive on the 580mV signal. This false positive may be avoided by increasing the threshold, but this would also increase the false negative rate.

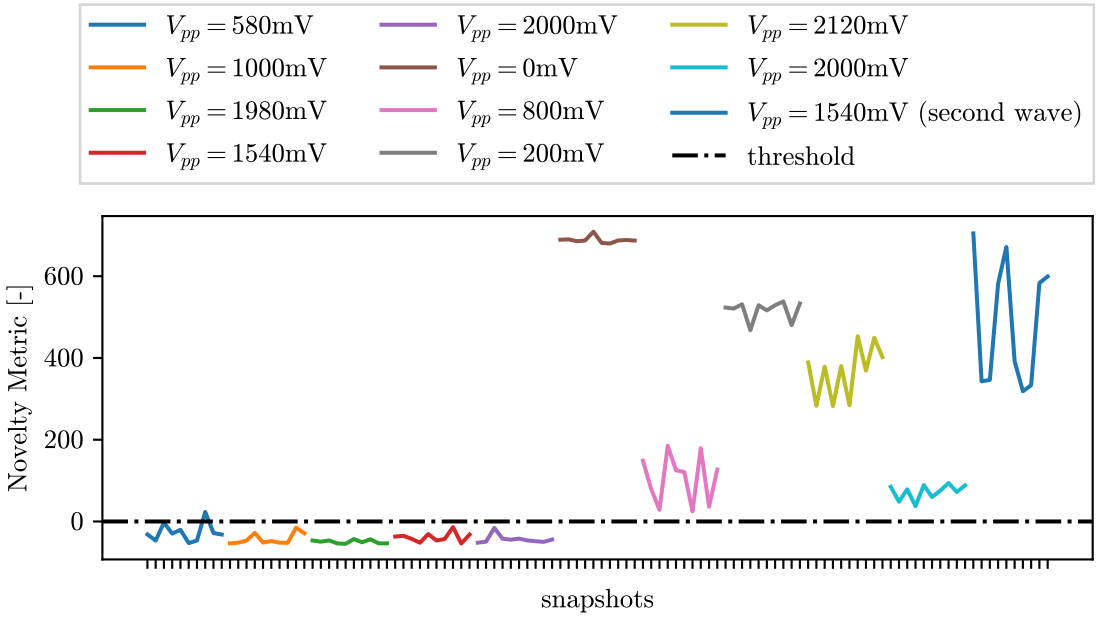

Figure 1.26: *LOF novelty detection result*

1.7 Experimental validation on a linear axis

The experimental validation reported in the previous **section 1.5** and **section 1.6** was carried out in a well-controlled environment with a shaker that was able to generate vibrations according to specific references. To further test the framework, a real-world application is considered in this section. The setup consists of a machine equipped with a linear axis, that is used to move a platform. On the moving platform the same accelerometer described in **table 1.2** has been attached using a custom 3D-printed fixture.

The test consists of defining a set of movements to be actuated by the platform, the accelerometer is used to capture the characteristics of each movement. As done previously, some movement profiles are used for training and others for testing. The position reference is shown in **figure 1.27**, and the parameters of the profiles are resumed in **table 1.5**.

Figure 1.27: *Position reference for the linear axis test.*

Table 1.5: *Harmonic coefficients for the shaker test.*

Profile N.	**Speed** [ms^{-1}]	**Acceleration** [ms^{-2}]	**Jerk** [s]
1	0.8	6	0.02
2	0.4	3	0.02
3	0.4	6	0.02
4	0.6	8	0.02

1.7.1 Training

To perform the training, a loop has been implemented on the PC that manages the axis movements. The script cyclically actuates the axis to follow the reference profile and asks the microcontroller to start the acquisition of the accelerometer data at the beginning of movement. The received features are then stored in a file, and the process is repeated for each profile. The sampling frequency of the microcontroller is 5kHz, for a total of 6000 samples per profile.

Although not useful for the training, the microcontroller has been set not only to transmit the features to the PC but also the time-series, for visualization purposes. The time-series of the training set are shown in **figure 1.28**, and the features are shown in **figure 1.29**.

In the time-series set it is possible to see some outliers, for example, there is a record in which profile 1 started being actuated by the axis with a delay

w.r.t. the others. Profile 4, instead, has some outliers due to the axis sometimes overshooting the reference position. These outlier are caused by the axis control, and the investigation about why does it happen is out of the scope of this work.

The training set contains 100 snapshots for each profile, for a total of 400 snapshots. The K-means model is then trained for $n = 5$ clusters, according to the silhouette criterion.

As done previously, the training is performed with the user confirming the correct number of clusters. And updating the model into the microcontroller.

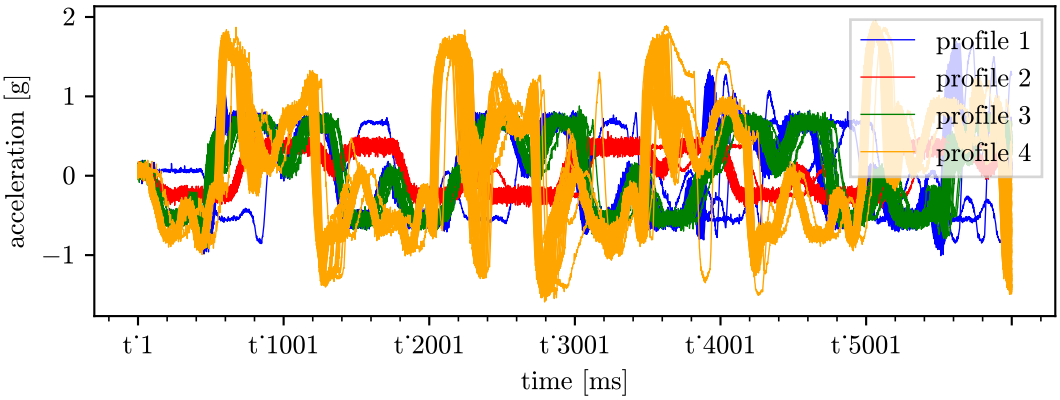

Figure 1.28: *Timeseries of the training set*

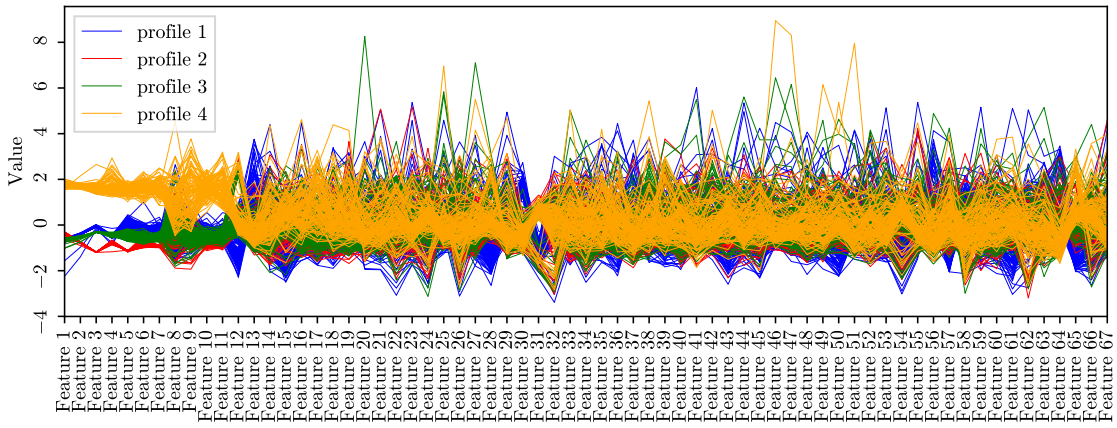

Figure 1.29: *features of the training set*

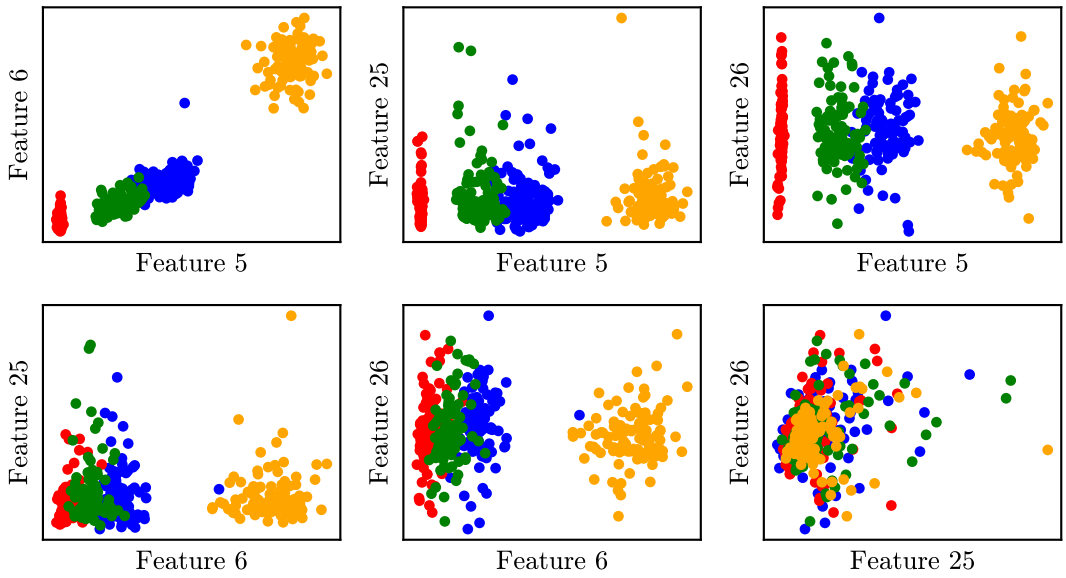

Figure 1.30: *Visualization of the separation between profiles in the feature space*

1.7.2 Testing

The microcontroller is then set in *evaluate* mode and the ND is performed on a loop that repeats the movement of profile 2. The result of this first model (Model 1) is shown in **figure 1.31**. We can see that the model immediately falsely detects a novelty, despite profile 2 being part of the training set. The figure also shows a clearer view of the evolution of the novelty score, obtained by applying a moving average filter on the last 5 values of the novelty metric.

Let's investigate why this model gives almost all false positive results. Analyzing the features of the training set (**figure 1.29**), we can see that the first features, up to ≈ 12, are grouped by profile, but most of the remaining features are not, this arises the suspect that these features may not be significative of the movement, but maybe just representing noise. This is likely also because the necessary standardization procedure ensures that each feature will have unitary standard deviation, regardless of the magnitude of the feature itself w.r.t. the others. This may cause "noise" features to be amplified in the model. Moreover, it's clear that, being most of the features not significant, the model is not able to distinguish between the profiles.

In **figure 1.30**, a better visualization of the problem is presented. Features 5 and 6 are significative of the specific movement as we can see in the scatter plot, because the data points are grouped by profile. The features 25 and 26, instead, are not significative, as the data points are not grouped by profile.

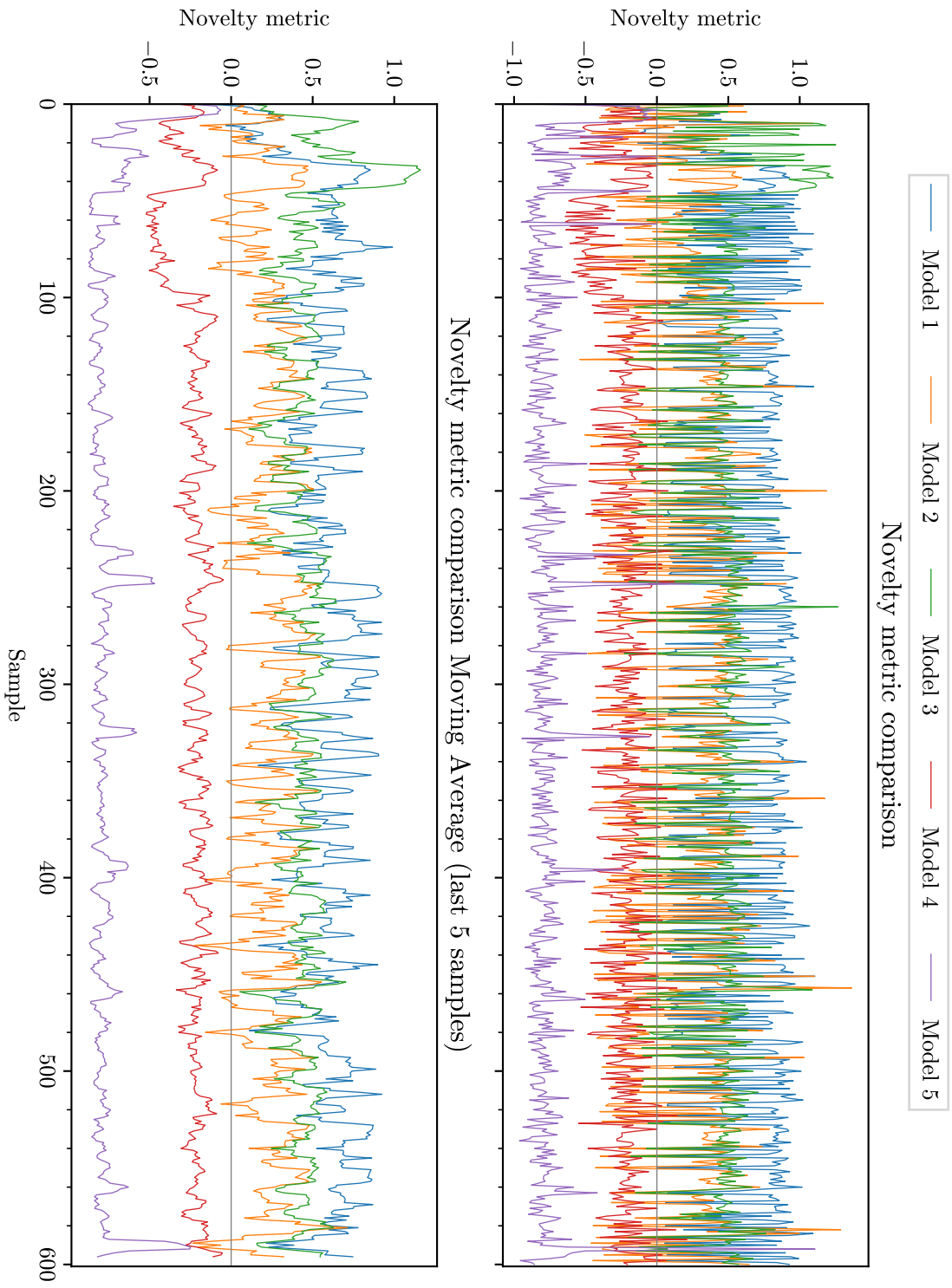

Figure 1.31: *Novelty detection on profile 2.*

1.7.3 Feature scaling

To address the problem of the non-significative features, a feature scaling method is proposed. The idea is to scale the features in such a way that the most significant features will have a higher weight in the model. To do that, a naive approach could be to visually select the important features (by eye it's evident that are the first few) and apply a small scaling factor to the others.

This approach goes against the principle of the framework being fully automatic to train. To address this problem, an unsupervised and automatic method is proposed. The idea is to scale the features in such a way that the most significant features will have a higher weight in the model. To do that, it's possible to exploit the fact that, at this point, the K-means model is already trained and the training procedure provided labels for the dataset. It's now possible to apply a *supervised* ML algorithm in a way that is transparent to the user, so the whole procedure remains *unsupervised*.

The framework is then extended to include the possibility to apply feature scaling. The two possibilities are to use the Random Forest classifier or the SelectKBest algorithm to provide the weights. Since this scaling will affect also the future snapshots, a new train of the MLA is necessary. The new model is then trained with the same training set, but with the scaled features. This approach is illustrated in **figure 1.32**.

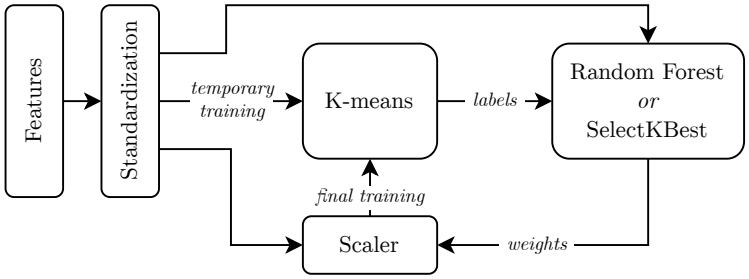

Figure 1.32: *Feature scaling procedure.*

This scaling technique is used with several configuration for both scaling the feature, and reducing the number of features, discarding the unnecessary ones. The settings of each tuned model are resumed in **table 1.6**. These models are describe later in this section.

Table 1.6: *Tuned embedded models parameters*

Model	Feature Scaling		Feature Subset	Training Snapshots (per each profile)				N of clusters
	SelectKBest	Random Forest		1	2	3	4	
Model 1				100	100	100	100	5
Model 2		✓		100	100	100	100	3
Model 3	✓			100	100	100	100	3
Model 4		✓		100	200	100	100	5
Model 5		✓	✓	100	200	100	100	6

Random Forest

The Random Forest classification algorithm gives a measure of the importance of each feature, and this measure can be used to scale the features. The RF algorithm is trained on the training set, with the labels provided by the K-means model.

The feature importance is based on the idea that that the Gini impurity is reduced at a split in each decision tree of the forest. The more the impurity is reduced, the more important the feature used for that split is [4]. This concept, averaged over all splits of all the trees, gives a measure of the importance of each feature.

For our purpose, the inportance is then normalized to have a maximum value of 1, so the most important feature will remain unscaled.

SelectKBest

Another tool for analyzing the importance of features is the SelectKBest algorithm. It is available in the `scikit-learn` library and is used to select the k most important features or alternatively, to output a score for each feature, based on ANOVA statistics.

Results

scrivere qui i barplot dei pesi

Bibliography

[1] J. Lee, H. Qiu, G. Yu, J. Lin, and Rexnord Technical Services. «Bearing Data Set». In: *IMS, University of Cincinnati* (2007). Available at: https://www.nasa.gov/intelligent-systems-division/discovery-and-systems-health/pcoe/pcoe-data-set-repository/ (cit. on p. 1).

[2] Umberto Albertin, Giuseppe Pedone, Matilde Brossa, Giovanni Squillero, and Marcello Chiaberge. «A Real-Time Novelty Recognition Framework Based on Machine Learning for Fault Detection». In: *Algorithms* 16.2 (2023). ISSN: 1999-4893. DOI: 10.3390/a16020061. URL: https://www.mdpi.com/1999-4893/16/2/61 (cit. on p. 11).

[3] Hai Qiu, Jay Lee, Jing Lin, and Gang Yu. «Wavelet filter-based weak signature detection method and its application on rolling element bearing prognostics». In: *Journal of Sound and Vibration* 289.4 (2006), pp. 1066–1090. ISSN: 0022-460X. DOI: https://doi.org/10.1016/j.jsv.2005.03.007. URL: https://www.sciencedirect.com/science/article/pii/S0022460X0500221X (cit. on p. 12).

[4] Stefano Nembrini, Inke R König, and Marvin N Wright. «The revival of the Gini importance?» eng. In: *Bioinformatics* 34.21 (Nov. 2018), pp. 3711–3718. ISSN: 1367-4811. DOI: 10.1093/bioinformatics/bty373. URL: https://doi.org/10.1093/bioinformatics/bty373 (cit. on p. 30).