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

The fourth industrial revolution is characterized by the integration of Artificial Intelligence and the Internet of Things paradigm into factories. Nowadays, more than a decade after this industrial revolution began, the maintenance approach remained unchanged in most industrial applications. The primary factor impeding the advancement of the maintenance approach is the significant expense associated with implementing Condition-Based or Predictive maintenance strategies, coupled with a lack of knowledge about the modelling or behaviour of a failing system.

In most facilities, maintenance continues to be performed according to a predefined schedule. An optimization of this approach involves intervening in the system only when necessary, which requires the knowledge of when a system is malfunctioning. Fault Detection and Anomaly Detection enable the triggering of an event when a known fault occurs or when a new, unfamiliar behaviour emerges in the maintained system. Predictive Maintenance aims to predict the Remaining Useful Life of a system. A quite novel frontier is the direct implementation of Predictive Maintenance within the maintained system, using the principles of Edge Computing.

In this work, a modular and general-purpose framework is proposed to ease the implementation into different systems. It is developed following an Unsupervised Machine Learning approach to overcome the lack of models. The Machine Learning core of the framework is based on the features extracted from the data gathered from sensors. The proposed framework operates in real-time, continuously assessing the health of the system. Firstly, it has been developed to be executed and tested on a PC using various Unsupervised Machine Learning algorithms, implemented in the Python language. The algorithms that appeared to maximize the performance-resources ratio were deployed on a microcontroller using the C language. The proposed solution includes all the tools necessary in the data pipeline. With this structure, the framework can be easily set up on a machine and extended to an arbitrary configuration of sensors and features. The PC implementation underwent testing using various Unsupervised algorithms on publicly available datasets, while the edge implementation was tested through laboratory experiments.

Both the test on datasets and the experimental results showed that the proposed framework is able to detect anomalies and give an estimate of the future degradation evolution of the system.

Table of Contents

List of Tables

List of Figures

Glossary

agent Software agent, a computer program that performs various actions continuously and autonomously on behalf of an individual or an organization. For example, a software agent may archive various computer files or retrieve electronic messages on a regular schedule. Such simple tasks barely begin to tap the potential uses of software agents, however. [1] 85

centroid The center of a cluster. It's the point that minimizes the sum of the distances between itself and all the points in the cluster. From a physical point of view, it's the center of mass of the cluster, if all the points of the cluster are treated as equal point masses. 33–35, 37, 38, 40, 42, 43

cluster In a set of data points, a cluster is a subset of the former that are more similar to each other than to the rest of the data points. This is a broad definition that leaves to the algorithm applied to perform the clustering the freedom to define what "similar" means. 34, 50

commissioning to bring something newly produced into working condition. 75, 76

Condition Based Maintenance Preventive maintenance including a combination of condition monitoring, inspection, testing, analysis, and ensuing maintenance actions. *Note: Condition monitoring and/or inspection and/or testing may be scheduled, on request, or continuous [2].* 9

Corrective Maintenance Maintenance carried out after fault recognition and intended to put an item into a state in which it can perform a required function [2]. 9

Deep Learning a class of algorithms which are based on artificial neural networks optimized to work with unstructured data such as images, voice, videos and text [3]. xv, 12

edge computing "Edge computing is an emerging computing paradigm which refers to a range of networks and devices at or near the user. Edge is about processing data closer to where it's being generated, enabling processing at greater speeds and volumes, leading to greater action-led results in real time."[4] 3, 4, 71, 74, 87, 99, 113, 114, 121

Feature Agent a software agent that extracts the features from the data polled by the Field Agent and stores them in a database in a suitable formatted way. xv, 76

Field Agent a software agent polls the data from the field and stores them in a database in a suitable formatted way. xii, xv, 76, 86

heuristic "any device, be it a program, rule, piece of knowledge, etc., which one is not entirely confident will be useful in providing a practical solution, but which one has reason to believe will be useful, and which is added to a problem-solving system in expectation that on average the performance will improve."[5] 34, 37, 58

hyperparameter hyperparameter is a parameter of a learning algorithm (not of the model). As such, it is not affected by the learning algorithm itself; it must be set prior to training and remains constant during training. 30, 31

JavaScript Object Notation is an open standard file format, and data interchange format, that uses human-readable text to store and transmit data objects consisting of attribute-value pairs and array data types. xiv, xvi

Lead Time The interval between the detection of a novelty, and a malfunction happening. 109

likelihood function The likelihood function is a measure of how likely is it that a given parametric model would generate the observed data. 53, 54

linearly separable Two sets of data points in a two-dimensional space are said to be linearly separable when they can be completely separable by a single straight line. In general, two groups of data points are separable in a n-dimensional space if they can be separated by an (n-1)-dimensional hyperplane [6]. 51

Machine Learning Agent software agent that trains the models, evaluate the metrics on new data, and makes predictions about the future evolution of the metrics. It also interface itself with the operator. xvi, 76

MongoDB a source-available, cross-platform, document-oriented database program. vii, 80, 81, 95

NoSQL an approach to database management that can accommodate a wide variety of data models, including key-value, document, columnar and graph formats. A NoSQL database generally means that it is non-relational, distributed, flexible and scalable. [7] 80

On Line Maintenance Maintenance carried out on the item while it is operating and without impact on its performance. *Note: In this type of maintenance, it is important that all safety procedures are followed [2].* 9

On Site Maintenance Maintenance carried out at the location where the item is normally located [2]. 9

Pickle format The pickle module implements binary protocols for serializing and deserializing a Python object structure. "Pickling" is the process whereby a Python object hierarchy is converted into a byte stream, and "unpickling" is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy. 85, 89

Predictive Maintenance Condition-based maintenance carried out following a forecast derived from repeated analysis or known characteristics and evaluation of the significant parameters of the degradation of the item [2]. 9

Preventive Maintenance Maintenance carried out at predetermined intervals or according to prescribed criteria. Intended to reduce the probability of failure or the degradation of the functioning of an item [2]. 9

Remaining Useful Life The remaining time before system health falls below a defined failure threshold [8]. xvii, 16

snapshot an array of features that describe the state of a system in a specific time period. It's filled with any metric (time domain, frequency domain, etc.) x, 34, 35, 37–40, 42, 43, 46, 52, 102, 113

standardized a signal that has been transformed to have a zero mean and unit variance 47

Traditional ML algorithm learning from data to perform a task without being explicitly programmed, excluding deep learning algorithms. 12

Acronyms

ν-SVM One Class SVM v, x, 76, 107, 109

k-NN k-Nearest Neighbors 12

a.k.a. Also Known As 9, 42

ADC Analog to Digital Converter 87

AE Autoencoder 12

AIC Akaike Information Criterion x, 54, 104, 105

ANN Artificial Neural Network 12

ANOVA Analysis of Variance 127

ART Adaptive Resonance Theory 12

BGMM Bayesan Gaussian Mixture Model x, 56, 106, 109

BIC Bayesian Information Criterion x, 54, 104, 105

BPFI Ballpass frequency, inner race 14

BPFO Ballpass frequency, outer race 14

BSF Ball (roller) spin frequency 14

BSON Binary JavaScript Object Notation 80

CART Classification and Regression Tree 30

CBM Condition Based Maintenance 2, 3, 5, 11, 77

CEP Cepstral Editing Procedure 14

CLI Command Line Interface vii, 92, 100

CNC Computer Numerical Control ix, 74, 75

CNN Convolutional Neural Network 12

CSV Comma Separated Values 87

DBN Deep Belief Network 12

DBSCAN Density-Based Spatial Clustering of Applications with Noise iv, v, x, 5, 33, 47, 51, 52, 55, 56, 59, 76, 103–105, 107, 109, 121

DFT Discrete Fourier Transform 64–66, 68

DL Deep Learning vii, 12, 13

DLR Deep Reinforcement Learning 12

DMA Direct Memory Access 97

DT Decision Tree iii, 12, 17, 29–31

EM Expetaion Maximization 53, 54

FA Feature Agent vii, 76, 87, 102

FD Fault Detection ix, 2, 5, 14, 16, 17, 79, 83, 85, 88, 90, 110

FFT Fast Fourier Transform 64, 71, 72

FiA Field Agent 76, 87, 100, 102

FTF Fundamental train frequency (cage speed) 14

GAN Generative Adversarial Network 12

GD Gradient Descent iii, 24, 25

GMM Gaussian Mixture Model iv, v, x, 34, 54–56, 59, 76, 104–107, 109

HAL Hardware Abstraction Library 97

i.e. "id est" (that is) 76

IDE Integrated Development Environment 96

iForest Isolation Forest v, x, 34, 107–109, 121

IMS Center for Intelligent Maintenance Systems v, vii–x, 18, 61–65, 69–73, 99–109, 111–113

IOT Internet Of Things 8, 71

ISO International Organization for Standardization 6, 82

JSON JavaScript Object Notation vii, 80–82, 84, 85

LCM Least Common Multiple 66

LCSR Loop CurrentStep Response 11

LOF Local Outlier Factor v, x, 34, 58, 59, 76, 108, 109, 121

LR Linear Regressor 17

LS Least Squares 22, 79, 110

ML Machine Learining vii, xiii, 12, 13, 17, 20, 25, 29, 74–76, 85, 88, 109, 126

MLA Machine Learning Agent vii, ix, 76, 77, 88, 90, 99, 100, 102, 110–112, 126, 127

MLP Multilayered Perceptront 12

MSE Mean Squared Error 30

NASA National Aeronautics and Space Administration (USA) 61

ND Novelty Detection v–vii, ix, x, 2, 5, 15–17, 61, 71, 74, 76, 79, 81, 83, 88, 90, 99, 102–113, 118, 119, 121, 124, 128

OS Operating System 11

PC Personal Computer 5, 74, 95, 96, 113, 115, 122

PDF Probability Density Function 53, 55, 56

PdM Predictive Maintenance vii, 2–5, 11–13, 17, 77

PF Particle Filter 12

PM Proactive Maintenance vii, viii, 2, 9, 10

POF Pareto Optimal Front 38, 100

PvM Preventive Maintenance 1

PW Pre-Whitening 14

RF Random Forest iii, x, 17, 31, 126, 127

RM Reactive Maintenance vii, viii, 1, 9, 10

RMS Root Mean Squared 63

RNN Recurrent Neural Network 12

RTD resistance temperature detector 11

RUL Remaining Useful Life v, ix, x, 12, 16, 17, 75, 77, 80, 110–113

SGD Stochastic Gradient Descent 24

SOM Self-Organizing Map 12

SVM Support Vector Machine iii, iv, xiv, 12, 26–28, 34, 59, 60

TL Transfer Learning 12

UML Unsupervised Machine Learining 2, 4, 5, 33, 76, 81, 85

USA United States of America xvi

w.r.t. With Respect To 8, 9, 16, 43, 52, 63, 76, 79, 118, 122, 124

WPD Wavelet Packet Decomposition 71, 72

Symbols

$\boldsymbol{c}$ **Centroid** Point in the features space that represents a cluster. Ideally it is the center of mass of the cluster it represents. xviii, 34–40, 42, 45

$\boldsymbol{\mathcal{C}}$ **Cluster** A set of objects that are more similar to each other than to those in other clusters. 34, 36, 39, 40, 42, 43

$\boldsymbol{d}$ **Distance** Vector difference between two points in the features space. 40, 42–45, 49

$\mathbb{E}$ **Expected value** the arithmetic mean of the possible values a random variable can take, weighted by the probability of those outcomes 64

F **Feature number** The number of features that describe the state of a system. 53

$\boldsymbol{r}$ **Radius** Euclidean distance between the centroid $\boldsymbol{c}$ of a cluster and its farthest point. 42–45

$\boldsymbol{\mathcal{S}}$ **Snapshot** A set of features that describe the state of a system at a given time. viii, xviii, 34, 36, 38–43, 45, 46, 52, 58, 100

S Snapshots Set A set of snapshots $\boldsymbol{\mathcal{S}}$. 34, 36, 38, 100

Chapter 1

Introduction

1.1 Preface

Imagine being the owner of a piece of technology. It can be any physical device. Without maintenance, it is unavoidable that, after a certain amount of time of usage it will have some sort of malfunction. To overcome this problem, usually, maintenance work is planned and performed by a team of skilled technicians.

The simplest way of executing maintenance is to fix the device when it breaks. This is called *reactive maintenance* (RM). This has been done since forever, and it is still a very popular approach today, but it has numerous drawbacks, including extended downtime for repairs and logistical challenges associated with spare parts.

The other family of maintenance techniques is called *preventive (or proactive) maintenance* (PvM). All those techniques share the property of being applied before the device shows a malfunction. This is a very broad category, and it includes a lot of different techniques, most of which will be described in **chapter 2**.

Back to the example, the owner of the device may want to avoid fixing it when it has already failed as much as possible, so the first improvement could be to perform maintenance periodically (on a schedule). This is called *predetermined maintenance*, and it is the simplest approach to PvM. It's basically the same approach that every motorist uses to minimize the risk of the vehicle breaking down in the middle of the road. The owner may want to take the technique a step further and seek some sort of guidance on deciding when to perform maintenance. A very intuitive approach is to simply ask the skilled technicians to inspect all the devices very often to try to understand if something is wrong without interfering with the normal operation of the device.

If a technician has enough experience, he may be able to detect the vast majority of the problems before they become critical. He may do that simply using his senses, for example by listening to the sound of the device, or by touching it to feel how much it vibrates, how hot is it and so on.

I would like to share, as a real-life example, what I witnessed during the commissioning of a new power plant. I remember that the commissioning manager (that is a chemist) was able to detect if the demineralization unit was working properly or not by simply tasting

the water it produced before the necessary instruments were installed.

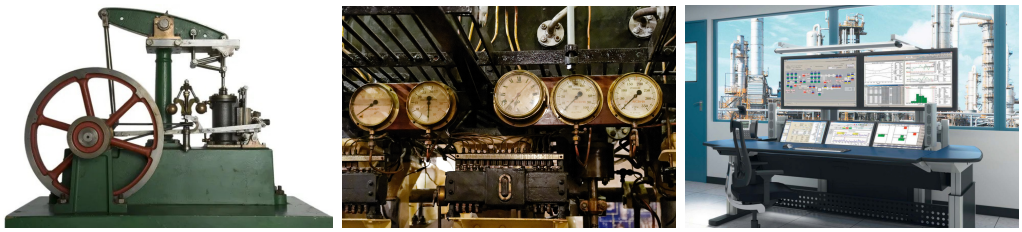
This naive approach can be enhanced by using a large number of sensors to monitor the most critical parameters of the device. In this case, the technician will train himself more on the data of the sensors, rather than on inspecting the device directly.

At this point, the next logical step would be to use the data from the sensor to train an algorithm, that will detect some patterns in the data that are not easily detectable by a human. This is called *condition-based maintenance* (CBM). This is the most common approach to PM today. If this algorithm is trained only on the data taken when the device is working properly, it would only be able to detect a novel behaviour, this is called *novelty detection* (ND). If the algorithm is trained on both the data taken when the device is working properly and the data taken when the device is malfunctioning in a specific way, it would be also able to detect the specific malfunction, this is called *fault detection* (FD).

One last improvement to the CBM approach is to use an algorithm that will also try to predict how much time is left before a critical malfunction will occur. This approach is referred to as *predictive maintenance* (PdM). This is the most advanced approach to PM today.

Both CBM and PdM could be done by using a model of the system that will predict the future behaviour based on the current state of the system. The problem with this approach is that it is very difficult to build a model that is accurate enough to be useful. Most of the time, in industrial applications, such a model is not available. A workaround to this problem is to apply an Unsupervised Machine Learning (UML) algorithm to the sensor data. This has the advantage of not needing a model of the system, but it has the drawback of being a black-box approach, so the parameters of the algorithm are not easily interpretable in a physical sense. Furthermore, the algorithm will be fairly good at detecting anomalies, but it will have some limitations in predicting the future behaviour of the system since it will likely be based on a forecast done interpolating the data of the past.

To visualize this evolution of maintenance approaches, let's have a look at **figure 1.1a**. This is a purely mechanical device, so the only way to detect a malfunction before it causes a failure is by trusting the "gut feeling" of the operator. Moving on, the device in **figure 1.1b** is also a steam engine, but it is equipped with some analogue gauges. In this case, it is possible to define some thresholds for the readings that are indicative of a malfunction and look at the time evolution of the values. The measure of the sensor is immediately displayed to the operator. Moving into nowadays, the device in **figure 1.1c** is a state of the Art control room. The data from the sensors are elaborated by a computer and displayed to the operator on screens. This allows the computer to run algorithms on the sensor data.



(a) *early steam engine [9]* **(b)** *steam engine [10]* **(c)** *control room [11]*

Figure 1.1: *Evolution of machinery*

The last comment is about where to run the algorithm. The most common approach is to run it on a server, that is not part of the device itself. This has the advantage of not having constraints on how much computational power is needed, because it's usually feasible to add more computational power to a computer that is not located on the device itself. The main drawback is that the data has to be transmitted to the server. This may not be feasible in some cases, for example for a mobile device, or for a device that is located in a remote area with no internet connection. In this case, the algorithm has to be run near the device itself, usually on a microcontroller that will perform an action when the algorithm detects a malfunction. This approach is called *edge computing*. Using a microcontroller also has the advantage of using very little power, which is critical for battery-powered devices.

1.2 Motivation

Take, for example, a survey published by the U.S. Department of Commerce in 2020, the maintenance expenditure of the manufacturing industry in the USA was $57.3 billion, while the losses due to preventable maintenance issues amounted to $119.1 billion. The top 25% of those establishments relying on reactive maintenance were associated with 3.3 times more downtime than those in the bottom 25% [12]. This means that the economic impact of failures exceeds the cost of maintenance by a factor of 2.1, and this is concentrated in those sectors that rely more on reactive maintenance. The aforementioned emphasizes the importance of developing a general purpose PdM system that can be applied in a large variety of systems.

In the article [13], the authors divide the CBM strategies in five categories:

- *experience based* predictions are based on the experience and knowledge outside or within the company. It is based on little scattered data on average conditions. The only requirement is that the experience of the experts is quantified and used;

- *reliability statistics* predictions are based on the statistical analysis of the failure data without considering the specific system. These methods also estimate the life of an average component operating under historically average conditions;

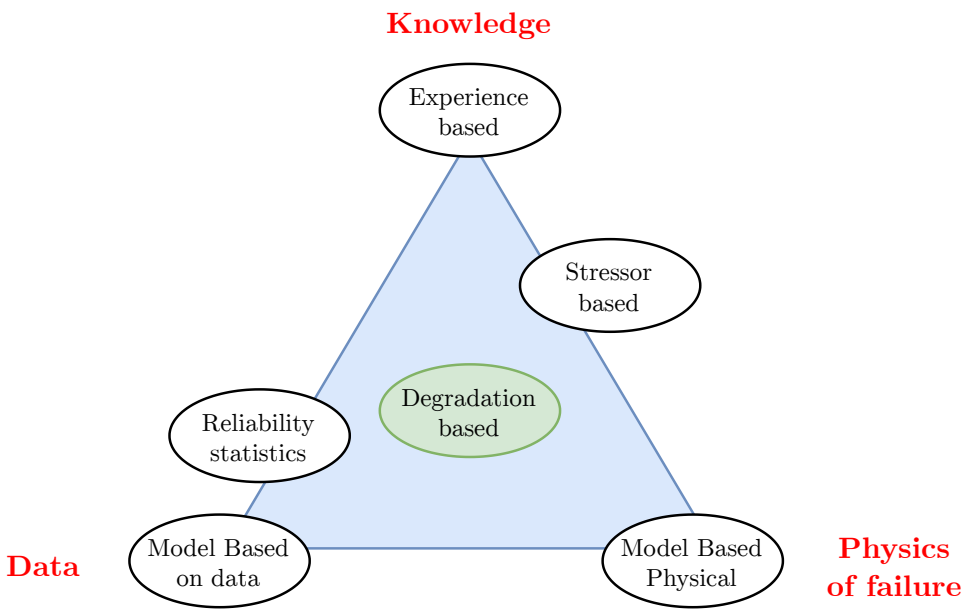


Figure 1.2: *Maintenance triangle*

- *stressor based* predictions can be considered an extension of the reliability statistics methods. The difference is that they also consider a measure of the stress (humidity, temperature, etc.) that the component is exposed to;
- *degradation based* predictions are based on the extrapolation of the degradation of the component;
- *model based* predictions are based on the knowledge of the physics of the system. The assumption is that the degradation can be computed instead of measured. They can be based on a physical model or a data-driven model.

The authors of [13] also propose a distribution of these categories in a triangle, as shown in **figure 1.2**. Each vertex represents a requisite for the implementation, and the distance of the method to the vertex represents how dependent it is on that requisite. To obtain a general purpose PdM framework, the *degradation based* approach seems the most promising, since it is the least dependent on a specific requisite.

Moreover, to enhance even more the general purpose nature of the framework, a UML model can be chosen. This choice confines the complexity of the work in the development of the framework itself, speeding up the implementation of the framework on a machine about which little is known, plus, it can be implemented on a new machine without a deep knowledge of the UML algorithms themself.

In the paper [14], it is pointed out that edge computing implementations, not only enable the use of the framework for special applications but also enhance the cybersecurity of facilities that may not need an edge implementation due to technical limitations.

1.3 Objective of the thesis

The goal of this thesis is to design, code and test a *degradation based* CBM framework that will perform ND, FD and PdM, using one or several UML algorithms. This framework is thought to be general purpose, needing just to receive time-series data from sensors, and to be set in training or evaluation mode. This system will be implemented twice:

- coding in `python` and running it on a PC;
- coding in `C` and running it on a microcontroller.

The development of the framework will be done modular and configurable, so that it will be easily adaptable to different use cases. Things like the number of sensors, the sampling frequency, and the number and types of features to extract will be easily configurable in a single file. The framework will be also designed to be easily expandable, so that new features can be added simply by developing a method that appends the new feature to the feature vector, without the need to modify the rest of the framework.

To do that, first, a real bearing vibration dataset published by [15] will be analyzed to decide how to preprocess the data, which features to extract and which algorithm to use. The candidate algorithms described in **chapter 4** will be applied to the dataset and the results will be compared. These algorithms are **K-means**, **DBSCAN**, **Gaussian Mixture Model**, **Isolation Forest**, **Local Outlier Factor** and **One-Class support vector machine**.

All of the tests will be carried out trying to follow the most unsupervised approach possible, so the only information used to train the algorithm will be the data itself. Every user input needed by the algorithm will be chosen using an easily automatable method, to preserve the unsupervised nature of the framework that will be developed. For example, user inputs are the number of clusters in K-means, or the radius in the DBSCAN.

At the end of the analysis, considering the trade-off between the performance of the algorithm, the computational cost and the simplicity, K-means will be chosen as the algorithm to implement in a real-time framework, developed in `python`. Then, the data of the dataset will be polled from a database at regular intervals and fed to the framework to simulate a real sensor polling the signal directly from the machine and evaluate the real-time performance of the implementation.

After that, a version of the framework with simplified architecture was implemented in `C` and tested on a `STM32F767ZI` microcontroller board.

1.4 Notations

In this thesis, the following notations are used:

- bold lowercase letters ($\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$) are used to denote vectors;
- italic lowercase letters (a, b, c) are used to denote scalars;
- bold uppercase letters ($\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$) are used to denote matrices;
- $\boldsymbol{a}_{i(j)}$ is the jth element of the vector $\boldsymbol{a}_i$;
- $||\boldsymbol{a}||_2$ is the l^2-norm of the vector $\boldsymbol{a}$, defined as $\sqrt{\sum_{i=1}^{n} a_i^2}$, denoted also as $\|a\|$ for simplicity;
- $||\boldsymbol{a}||_n$ is the generic l^n-norm of the vector $\boldsymbol{a}$, defined as $\sqrt[n]{\sum_{i=1}^{n} |a_i|^n}$;
- parenthesis encapsulating an index are used to condense descriptions, for example "$\boldsymbol{a}_{i(j)}$ **is the force applied to the ith (jth) step**" means that $\boldsymbol{a}_i$ is the force applied to the ith step and $\boldsymbol{a}_j$ is the force applied to the jth step;
- date and times are presented in the ISO 8601 format [16]. for example `YYYY-MM-DDThh:mm:ss`, where `T` is the date-time separator;

No distinction in notation is made between a vector in the physical sense (applied to a point, with a direction, and a magnitude) and a vector in the mathematical sense (a generic number $\in \mathbb{R}^n$).

As regards the flowcharts, the following symbols shown in **table 1.1** are used.

Table 1.1: Symbols used in the flowcharts

Symbol	Name	Usage
	terminator	start or stop the process
	stored data	save some data
	data	elaborate some data
	actions	perform automated actions
	document	read or write a docuument

Continued on next page

Table 1.1: Symbols used in the flowcharts (Continued)

	database	perform an action on a database
	display	report, plot or display a value
	manual input	request an input from the user
	manual operation	request the user to do something
	or	join flow line
	predefined process	run a programmed process

Chapter 2

State of the Art

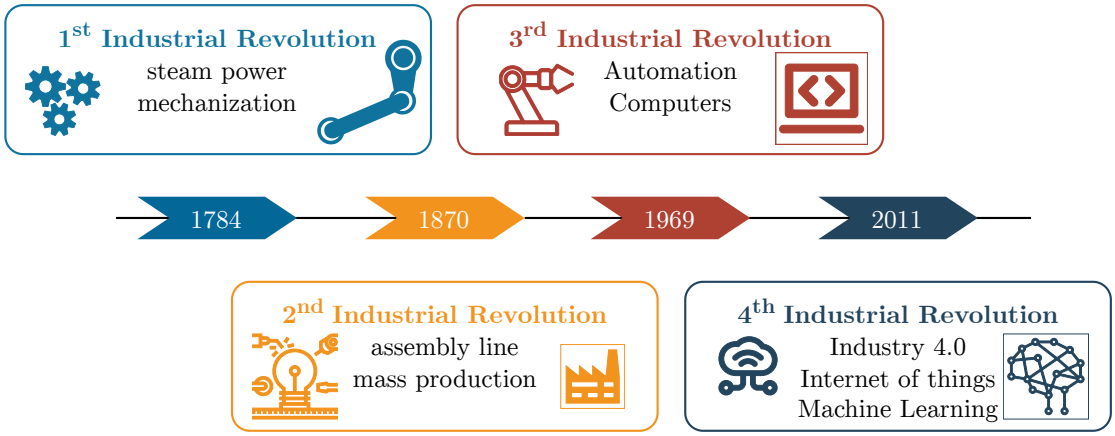


Figure 2.1: *Indusstrial revolutions*

The invention of the modern steam engine in the 18^{th} century, marked the beginning of the first industrial revolution. The second industrial revolution, in the 19^{th} century, was characterized by the introduction of mass production and the assembly line. The introduction of computers and automation in factories, in the 20^{th} century, enabled the third industrial revolution. Nowadays, we are currently living in the 4^{th} industrial revolution, that embraces the industry 4.0 vision. State-of-the-art industries have small decentralized smart networks that make decisions autonomously. This is possible thanks to the *Internet of Things* (IOT), smart sensors and actuators, and *Big Data* analysis (**figure 2.1**). The data to be monitored varies w.r.t. the field of application. The most common are [17]:

- Vibration Analysis - Efficient method for detecting issues in rotating equipment.
- Acoustic Analysis - Detects or monitors cracks in pipes and other structures.
- Lubrication Oils Analysis - Analyzes particles in oils to assess component wear.

- Particle Analysis in Working Environment - Applied to equipment operating in fluid environments.
- Corrosive Analysis - Ultrasound measurements to determine corrosion in various structures.
- Thermal Analysis - Identifies overheating in mechanical and electrical systems.
- Performance Analysis - Efficient technique for pinpointing operational problems in the system.

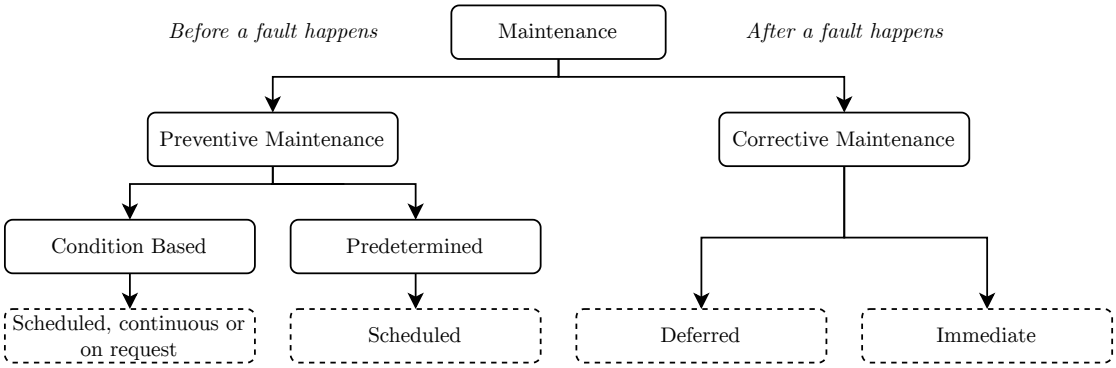


Figure 2.2: *Standard terminology for industrial maintenance [18]*

Standard terminology A standard terminology used for industrial maintenance is provided by the European committee for standardization with the standard `EN13306:2018` [2]. The terminology is summarized in **figure 2.2**. The most advanced maintenance technique family is **Condition Based Maintenance**. This category includes the most modern **Predictive Maintenance**. Note that the definition does not imply that the "monitoring" of the system must be continuous, it may also be scheduled or not even scheduled and performed both manually or by a program.

The standard also defines what **On Line Maintenance** and **On Site Maintenance**. All these definitions are reported in the glossary

RM vs PM As anticipated in the introduction, the two main approaches are **reactive maintenance** (a.k.a. Corrective Maintenance), which restores system functionality, and **proactive maintenance** (a.k.a. Preventive Maintenance), which preserves system functionality [19].

The former approach leads to very high downtime w.r.t. the latter [12]. For all the time a system is down, the company forfeits the opportunity to make a profit. This is called *lost opportunity cost*. In reality, the total costs of downtime are even higher, because there are other costs associated with labor overhead and materials [20]. The second approach optimizes both the pre-repair and post-repair logistics and, acting before the failure, can reduce also the total downtime. A qualitative diagram of these benefits is

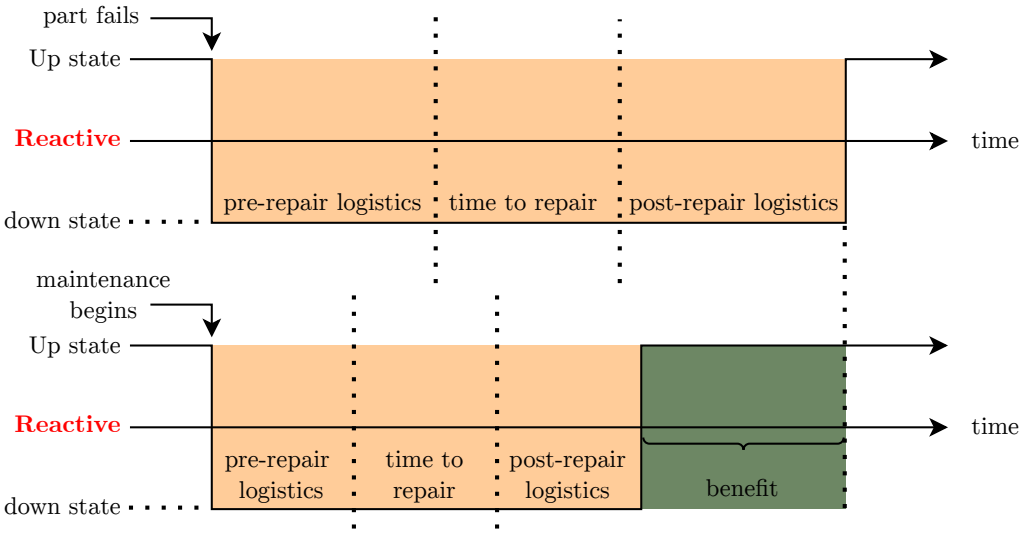


Figure 2.3: *Downtime comparison (RM and PM)*

shown in **figure 2.3**. Other than the downtime, a more complete comparison between the advantages and disadvantages of the two approaches is shown in **table 2.1**.

Table 2.1: *Advantages and disadvantages of RM and PM maintenance [20]*

Maintenance	Advantages	Disadvantages
Reactive	low setup costs easy to setup	unscheduled downtime increases labour costs unoptimized resources increases manufacturing costs
Proactive	increases system availability minimizes logistical downtime reduces unscheduled downtime decreases costs • optimizes parts • optimizes labour maintenance is planned optimizes logistical support	high setup costs savings not seen immediately not feasible for all equipment

Passive vs active maintenance PdM techniques can be divided also into *passive* and *active*. The former uses existing sensors or adds new sensors to the system and these data are just analyzed. The latter, instead, uses actuators to perturb the system and then analyzes the response. The former is more common because it is less expensive and less invasive. The latter, instead, is more accurate but its application is limited to special applications. The most common field of application of active PdM is electrical systems, where the perturbation can be applied by injecting a current or a voltage [21].

In [21], the author proposes also, as an example of active PdM, the use of the Loop Current Step Response (LCSR) technique. In this test, an electrical signal in the form of a step change is sent to the sensor using a Wheatstone bridge, causing heating in the RTD sensing element. The resulting exponential transient at the bridge output is analyzed to determine the RTD's response time. Beyond measuring response time, the LCSR test can serve other purposes, such as detecting water levels in a pipe and ensuring the proper installation of temperature sensors in thermowells. Moreover, it aids in verifying timely responses to temperature changes and identifying potential degradation due to ageing.

Models of degradation In [22], the authors propose a decision model that optimizes the inspection schedule and replacement time to minimize the cost of failure and unavailability. This procedure is based on two variables: the *replacement threshold* and the *inspection schedule*. Most of the non CBM policy can be emulated with specific values of these two variables. This is applied to gradually deteriorating single-unit systems. The degradation is simulated with a random model that also considers the time to perform maintenance for an arbitrary period.

Another approach for characterizing the degradation of a system is to use a stochastic model hypothesizing the use of an imperfect monitoring system. The data from the sensors are used to update the model with a Bayesian approach. The study is tested on simulated data that emulate a decaying system using Markov chains [23][24].

Cloud based PdM A relatively new structure for PdM is proposed in [25]. The authors investigate a low-cost cloud-based paradigm based on the concept of *mobile agents*, implemented in embedded Linux OS with open-source libraries. Compared to the traditional client-server paradigm, this approach enhances the scalability and flexibility of the system, reducing also the need for transmission of heavy raw data.

The concept of mobile agents used in this implementation can be resumed as autonomous software entities that can migrate from one host to another, carrying their data and state [26].

The authors of [25] tested the mobile agent implementation with induction motors that exhibited different failure modes. For example, a motor with a broken rotor bar defect is analyzed, collecting raw current measurements, envelope analysis, and spectrum analysis. Spectrum analysis poses challenges in distinguishing healthy and faulty motor signals. However, a comparison of current envelopes reveals marked differences in energy concentration associated with broken rotor bar-related frequencies. The defects analyzed by the system are: broken bar, bowed rotor, unbalanced rotor, stator winding defect, and

defective bearing.

In the study [27], a cloud-based PdM system is proposed and tested on a gearbox in a bench test. This study performs anomaly detection, fault detection, and RUL prediction. Thw RUL predictions are made by selecting a health indicator that is strongly correlated with the remaining life of the component.

Thermal imaging Yet another tool for detecting anomalies, mostly used for electrical devices, is gathering images of the device using an infrared camera. This method has the advantage of being noninvasive. The process of images is a whole discipline, in [28], the authors use a multilayered perceptron MLP to classify 11 features of the images. They achieved 78% accuracy using the MLP alone, which has been enhanced to 84% performing a graph cut.

Algorithms for PdM To continue the overview of state-of-the-art PdM techniques, we will now focus on the algorithms used to analyze the data. The two main categories of algorithms are Traditional ML and Deep Learning (DL). The survey [29] provides a comprehensive overview of the most common algorithms used in PdM, that we summarized in **table 2.2**, that is a merge of [29],[30],[31],[32] and [33]. ANN, DT, SVM, k-NN, PF, ART and SOM are ML algorithms, whlile AE, CNN, RNN, DBN, GAN, TL and DLR are deep learnign algorithms. The most common field of application of each algorithm is also reported in the table.

Table 2.2: ML and DL algorithms used in PdM [29]

Algorithm	**Acronym**	**Typical application**
Artificial Neural Network	ANN	• fault diagnostic in bearings • RUL predictions fo bearings
Decision Tree	DT	• fault diagnostic - grids - rail vehicles - bearings - hydraulics etc. • fault prognosis - turbofans - batteries - mechanical systems etc.
Support Vector Machines	SVM	• fault diagnostic - rotation machinery - bearings - wind turbines etc. • RUL predictions - batteries - bearings etc.

Continued on next page

Table 2.2: ML and DL algorithms used in PdM [29] (Continued)

k-Nearest Neighbor	k-NN	• fault diagnostic • RUL prediction • Early fault warning
Particle Filter [30]	PF	• RUL in turbine application [31]
Adaptive resonance theory	ART	• anomaly detection metal oxide surge arrester [33]
Self-Organizing Maps	SOM	• anomaly detection [32]
Auto-Encoder	AE	• feature extraction • data fusion • fault diagnostic • degradation estimation • RUL predictions
Convolutional Neural Network	CNN	• (joint) fault diagnostic • degradation estimation • RUL predictions
Recurrent Neural Network	RNN	• fault diagnostic • RUL predictions • health indicator
Deep Belief Network	DBN	• feature extraction • fault classification • RUL predictions
Generative Adversarial Network	GAN	• class imbalance • fault identification • RUL predictions
Transfer Learning	TL	• fault diagnosis • RUL predications
Deep Reinforcement Learning	DLR	• decision making • fault diagnosis • health indicator

Fault / Novelty detection As anticipated in the **section 1.1**, another distinction in the PdM techniques arises from the data available to build a model and/or to train it.

Fault detection If there is a knowledge of the peculiar features of most faults, the algorithms can be trained to detect them. As anticipated, this is called *fault detection* (FD). For example, if the monitored system is a ball bearing, it is well known in the literature that there are four distinct fault modes, each of which has a specific frequency signature illustrated in **figure 2.4** [34]:

$$\begin{aligned}
\text{Ballpass frequency, outer race (BPFO)} &= \frac{n \cdot f_r}{2}\left\{1 - \frac{d}{D}\cos\phi\right\} \\
\text{Ballpass frequency, inner race (BPFI)} &= \frac{n \cdot f_r}{2}\left\{1 + \frac{d}{D}\cos\phi\right\} \\
\text{Fundamental train frequency (FTF)} &= \frac{f_r}{2}\left\{1 - \frac{d}{D}\cos\phi\right\} \\
\text{Ball (roller) spin frequency (BSF)} &= \frac{D}{2 \cdot d}\left\{1 - \left(\frac{d}{D}\cos\phi\right)^2\right\}
\end{aligned}$$

Where f_r is the shaft speed, n is the number of rolling elements, and ϕ is the angle of the load from the radial plane.

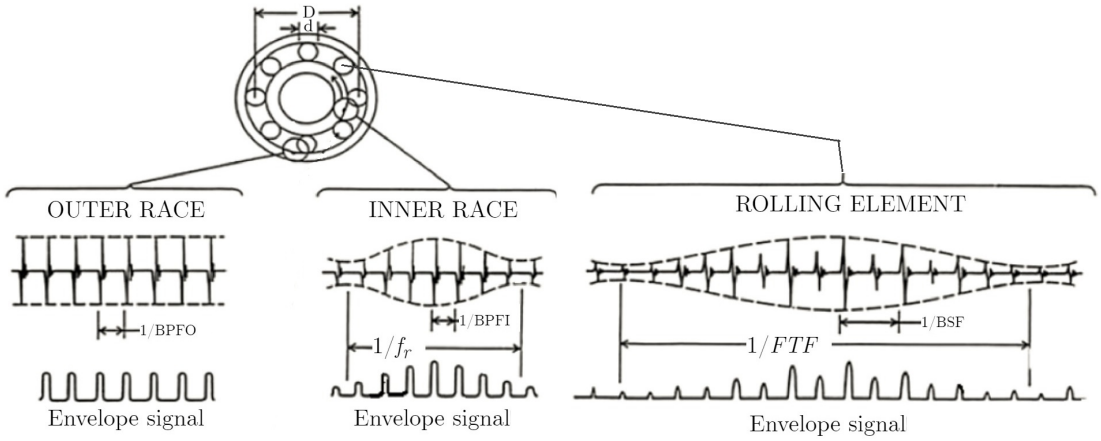


Figure 2.4: *Typycal bearing fault signals [34]*

An automated method for bearing diagnosis has been developed by [35]. The method is parametric and can be adapted to a large variety of cases. In the study, it has been tested on a helicopter gearbox, a high speed ($\approx$ 12000rpm) test bench application and a low speed ($\approx$ 1800rpm) radar tower.

The automated procedure [35] has been extended by a more recent study [36] where the authors applied a Cepstral Editing Procedure (CEP) based signal Pre-Whitening (PW). The framework has been tested on data collected from seventeen wind turbines. The procedure was successful in this case study, the preprocessing flow applied to the time-series, and the resulting spectral in which the BPFI is exploited to detect the fault, are shown in **figure 2.5**.

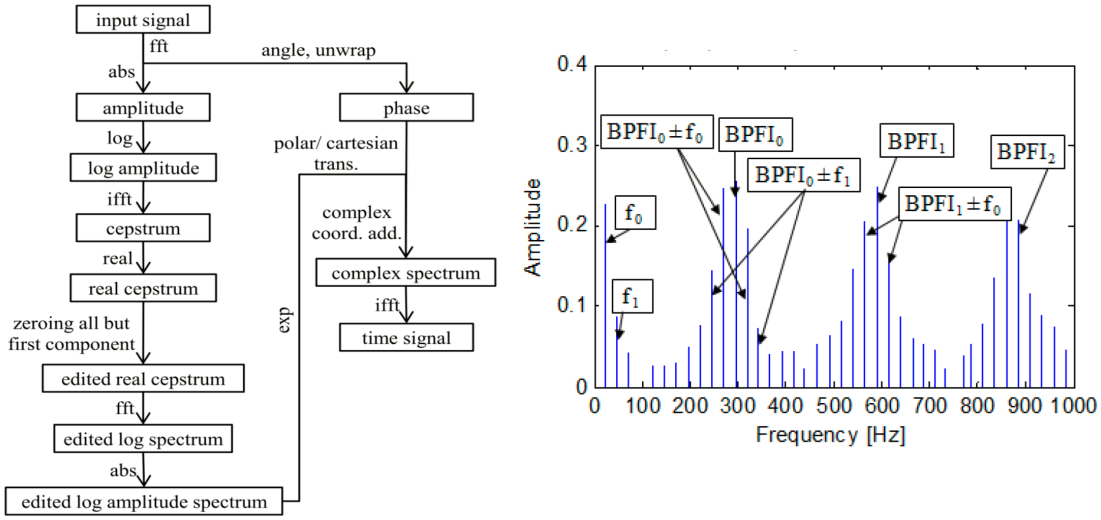


Figure 2.5: *Preprocessing schematic and spectrum of a bearing fault signal [36]*

Novelty detection As anticipated, most of the time there is almost no precise knowledge about the physics of the system and data collections about faults are not available. In this case, *novelty detection* (ND) can be used. The task of detecting if a condition is "novel" can be seen as a classification problem with only one class (the data collected on the healthy system). The general idea is that if the one-class classifier is not able to classify a new observation as "healthy", it means that the observation is "novel".

Once the novelty detection algorithm is trained, it can be used to give an estimate of "how novel" the current behaviour of the system is. One of the major issues with ND is to set the threshold value to decide if the observation is novel or not [37]. This is because the value of the metric is hardly linkable to a physical property, and the span of the metric is not known a priori.

In the **table 2.3**, the novelty detection techniques described in the comprehensive review [37] are summarized. The review makes clear that in the field of ND, both supervised and unsupervised techniques are used. It divides the categories on the theory behind:

- **Probabilistic** - involve a density estimation of the data;
- **Distance-based** - are the class of clustering techniques used traditionally for classification;
- **Reconstruction-based** - use a regression model to reconstruct the data, then the error is used to detect the novelty;
- **Domain-based** - try to define a boundary that contains all the normal data;
- **Information-theoretic** - is based on the idea that novel data significantly alter the information content of the dataset.

Table 2.3: State of the Art techniques for ND [37]

Model	**Type**
Mixture models	probabilistic, parametric
State-space models	probabilistic, parametric
Kernel density estimators	probabilistic
Nearest neighbour	distance-based
Clustering	distance-based
Neural networs	reconstruction-based
Subspace-based approaches	reconstruction-based
Support vector descriptors	domain-based
One-class support vectors	domain-based

The first two terminologies are adopted also in the technical review on methods [38]. This study also categorizes the pattern to be identified in the following classes:

- **Point pattern** - are single instances that are anomalous w.r.t. the rest of the data;
- **Contextual patter** - are anomalous w.r.t. a specific context;
- **Collective pattern** - is a collection of data instances that are anomalous if considered together.

The three distinct concepts are illustrated in **figure 2.6**.

The task of detecting the novelty is often associated with the task of predicting the Remaining Useful Life (RUL), before the fault becomes fatal for the component.

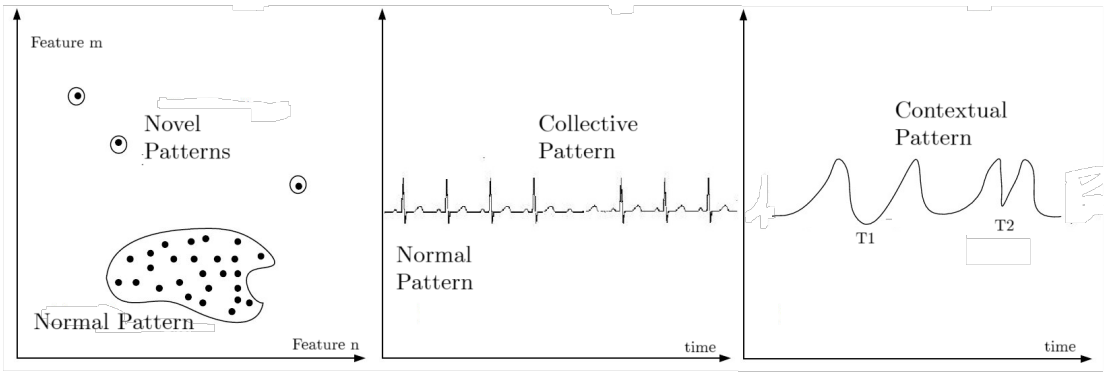


Figure 2.6: *Types of patterns [38]*

A novel framework for performing ND, FD and RUL predictions has been proposed

by researchers at PIC4SeR[1] in [39]. It is based on several autonomous agents working together on a database. The authors aimed to perform PdM in a scenario in which there is no physical knowledge and no prior data collections about the maintained system. The framework is meant to be set in a *training phase* on a new machine, to collect the *normal* data and train the ML model. After that, it will continuously work in *testing mode*: the framework will compute a prediction error on the current data that is used as a novelty metric.

The features are pre-processed using a windowing function and a cumulative absolute sum. Three regression models are used to perform ND and FD: a Linear Regressor LR, a Decision Rree (DT) and a Random Forest (RF). The user of the framework can decide which regressor to use in each specific case.

The model can be retrained after a novelty has been detected, to update the model with the new data. Even if the framework is meant to be trained on a new machine, it can be used also on a machine that has been in service for years: the faults already present will be part of the training database, but the predictions will still be useful because of the tendency of the faults to worsen over time.

The RUL predictions are made by averaging the prediction error in two intervals and then performing a linear regression on the two points.

This framework has been successfully tested on:

- a synthetic dataset that the authors created to emulate a bearing fault (using the definitions of the 4 typical faults [34]).
- a real dataset of bearing faults provided by the Center for Intelligent Maintenance Systems [15]
- a laboratory test on spring probes

In the first two cases, the framework was able to detect the novelty and predict the RUL, in the laboratory test, it was able to recognise the new data as *healthy* because the probes were not yet damaged. The graphical results of one test on real-world data are shown in **figure 2.7**.

Clustering The most common unsupervised task is clustering. In recent years, the volume of data collected in a typical factory has increased dramatically. Clustering is a collection of tools to extract information from huge amounts of unlabeled data.

These algorithms can be divided into: *partitioning-based* where the task is to define the boundaries between the clusters; *hierarchical-based* that shows the relation between each pair of clusters depending on the medium of similarity or dissimilarity; *density-based* that describes the clusters as a dense region of data points separated by low-density regions; *grid-based* that apply the transformation of the feature space into a grid before proceeding

[1] `https://pic4ser.polito.it/`

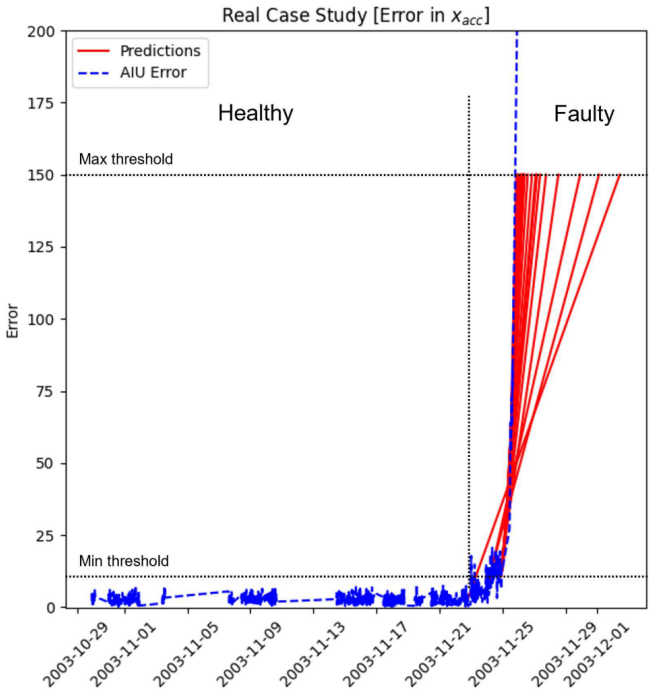


Figure 2.7: *Results provided by [39] for the test n°1 of IMS dataset.*

with the clustering and *model-based* that use a statistical or deep-learning model to describe the data.

Recently, the survey [40] provides a comprehensive overview of the most common clustering algorithms used in an industrial context, with reference studies. The comparison of the study is reported in **table 2.4**.

Table 2.4: Clustering algorithms comparison [40]. n = number of samples, k = number of clusters, d = number of features.

Algorithm	**Volume**	**High dim.**	**Cluster shape**	**Complexity**	**n. param.**
K-means	any	no	non-convex	$\mathcal{O}(nkd)$	1
K-modes	large	yes	non-convex	$\mathcal{O}(n)$	1
K-medioids	small	yes	non-convex	$\mathcal{O}(n^2dt)$	1
PAM	small	no	non-convex	$\mathcal{O}(k(n-k)^2)$	1
CLARA	large	no	non-convex	$\mathcal{O}(k(40+k)^2+$ $+k(n-k))$	1
Ward	any	no	non-convex	$\mathcal{O}(n)$	1
BIRCH	large	no	non-convex	$\mathcal{O}(n)$	2

Continued on next page

Table 2.4: Clustering algorithms comparison [40]. n = number of samples, k = number of clusters, d = number of features. (Continued)

CURE	large	yes	any	$\mathcal{O}(n^2 \log n)$	2
ROCK	large	no	any	$\mathcal{O}(n^2 + n^2 \log n)$	1
Chamelon	large	yes	any	$\mathcal{O}(n^2)$	3
DBSCAN	large	no	any	$\mathcal{O}(n \log n)$	2
OPTICS	large	no	any	$\mathcal{O}(n \log n)$	2
DENCLUE	large	yes	any	$\mathcal{O}(D)$	2
Wavecluster	large	no	any	$\mathcal{O}(n)$	3
STING	large	no	any	$\mathcal{O}(k)$	1
CLIQUE	large	yes	any	$\mathcal{O}(ck + mk)$	2
OPTGRID	large	yes	any	$\mathcal{O}(nd \log n)$	3
EM	large	yes	non-convex	$\mathcal{O}(knp)$	3
COBWEB	small	no	non-convex	$\mathcal{O}(n^2)$	1
SOM	small	yes	non-convex	$\mathcal{O}(n^2 m)$	2

Chapter 3

Machine Learning

Before diving into the description of the unsupervised algorithms used for the development of this thesis work presented in **chapter 4**, this chapter aims to be an introduction of *Machine Learning* (ML) in general.

An early but useful definition of Machine Learning was given by Arthur Samuel in 1959: "*Machine learning is the field of study that gives computers the ability to learn without being explicitly programmed.*" A more recent definition is the following, from Tom Mitchell: "*A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.*" [41, p. 4]

So, in general, the ingredients of ML are:

- some data linked to some task
- a task to be performed
- an algorithm that learns how to perform the task on specific data

The data are usually preprocessed before giving them to the algorithm. The processed data are called *features*. This is a generic term that refers to the information content of the data. For example, if the data are recordings of temperatures over time, the features could be the mean, the standard deviation, the minimum, and the maximum of the temperature or, in some cases if the algorithm is able to learn directly from them, the raw data themself.

The tasks can be divided into main categories:

- regression: the algorithm is trained to measure the relation between the value of output variables and corresponding values of other input variables;
- classification: the algorithm is trained to assign a label to a new instance, based on the training dataset of labelled instances;
- clustering: the algorithm is trained to group similar instances into clusters.
- anomaly detection: the algorithm is trained to identify instances that are different from known previous instances.

3.1 Regression

3.1.1 Least Squares

Lets consider a set of m observations of a variable $y \in \mathbb{R}^{n_y}$ (output features) that depends on a variable $x \in \mathbb{R}^{n_x}$ (input features) and a set of $n_f \cdot n_y$ parameters $\theta \in \mathbb{R}^{n_f \times n_y}$.

Suppose to know that the output features are linked to the input features with n_f functions linear in the parameters θ, so that:

$$\begin{bmatrix} y_1 & y_2 & \dots & y_{n_y} \end{bmatrix} =$$

$$\begin{bmatrix} f_1(x_1, \dots, x_{n_x}) & f_2(x_1, \dots, x_{n_x}) & \dots & f_{n_f}(x_1, \dots, x_{n_x}) \end{bmatrix} \cdot \begin{bmatrix} \theta_{1,1} & \dots & \theta_{1,n_y} \\ \theta_{2,1} & \dots & \theta_{2,n_y} \\ \vdots & \ddots & \vdots \\ \theta_{n_f,1} & \dots & \theta_{n_f,n_y} \end{bmatrix}$$

Where all the f_i are any known functions, y_i and x_i are known data and $\theta_{i,j}$ are the parameters to be found.

Considering the m observations, the previous equation can be extended as:

$$\begin{bmatrix} y_{1,1} & y_{1,2} & \dots & y_{1,n_y} \\ y_{2,1} & y_{2,2} & \dots & y_{2,n_y} \\ \vdots & \ddots & \vdots & \\ y_{m,1} & y_{m,2} & \dots & y_{m,n_y} \end{bmatrix} =$$

$$\begin{bmatrix} f_1(x_{1,1}, \dots, x_{1,n_x}) & \dots & f_{n_f}(x_{1,1}, \dots, x_{1,n_x}) \\ f_1(x_{2,1}, \dots, x_{2,n_x}) & \dots & f_{n_f}(x_{2,1}, \dots, x_{2,n_x}) \\ \vdots & \ddots & \vdots \\ f_1(x_{m,1}, \dots, x_{m,n_x}) & \dots & f_{n_f}(x_{m,1}, \dots, x_{m,n_x}) \end{bmatrix} \cdot \begin{bmatrix} \theta_{1,1} & \dots & \theta_{1,n_y} \\ \theta_{2,1} & \dots & \theta_{2,n_y} \\ \vdots & \ddots & \vdots \\ \theta_{n_f,1} & \dots & \theta_{n_f,n_y} \end{bmatrix}$$

Rewriting the previous equation in a more compact form:

$$\begin{bmatrix} \boldsymbol{y}_1 \\ \boldsymbol{y}_2 \\ \vdots \\ \boldsymbol{y}_m \end{bmatrix} = \begin{bmatrix} f_1(\boldsymbol{x}_1) & f_2(\boldsymbol{x}_1) & \dots & f_{n_f}(\boldsymbol{x}_1) \\ f_1(\boldsymbol{x}_2) & f_2(\boldsymbol{x}_2) & \dots & f_{n_f}(\boldsymbol{x}_2) \\ \vdots & \ddots & \vdots & \\ f_1(\boldsymbol{x}_m) & f_2(\boldsymbol{x}_m) & \dots & f_{n_f}(\boldsymbol{x}_m) \end{bmatrix} \cdot \begin{bmatrix} \theta_{1,1} & \dots & \theta_{1,n_y} \\ \theta_{2,1} & \dots & \theta_{2,n_y} \\ \vdots & \ddots & \vdots \\ \theta_{n_f,1} & \dots & \theta_{n_f,n_y} \end{bmatrix} \tag{3.1}$$

That, in the most compact form, becomes:

$$\boldsymbol{Y} = \boldsymbol{\Phi}(\boldsymbol{X}) \cdot \boldsymbol{\Theta} \tag{3.2}$$

In close form, there is a solution $\boldsymbol{\Theta_{LS}}$, for estimating the parameters that minimize the error between the estimated output $\boldsymbol{Y_{LS}} = \boldsymbol{\Phi(X)\Theta_{LS}}$ and the real output $\boldsymbol{Y}$, that is

known. Let's see, in an intuitive way:

$$\boldsymbol{Y} = \boldsymbol{\Phi}(\boldsymbol{X}) \cdot \boldsymbol{\Theta_{LS}} \tag{3.3}$$

$$\boldsymbol{\Phi}(\boldsymbol{X})^T \boldsymbol{Y} = \underbrace{\boldsymbol{\Phi}(\boldsymbol{X})^T \boldsymbol{\Phi}(\boldsymbol{X})}_{\text{square}} \cdot \boldsymbol{\Theta_{LS}} \tag{3.4}$$

$$(\boldsymbol{\Phi}(\boldsymbol{X})^T \boldsymbol{\Phi}(\boldsymbol{X}))^{-1} \boldsymbol{\Phi}(\boldsymbol{X})^T \boldsymbol{Y} = \boldsymbol{\Theta_{LS}} \tag{3.5}$$

$$\text{pinv}(\boldsymbol{\Phi}(\boldsymbol{X})) \boldsymbol{Y} = \boldsymbol{\Theta_{LS}} \tag{3.6}$$

In fact, it is known that $\boldsymbol{\Theta_{LS}} = \text{pinv}(\boldsymbol{\Phi}(\boldsymbol{X}))\boldsymbol{Y}$ is the solution of the following minimization problem:

$$\boldsymbol{\Theta_{LS}} = \arg \min_{\boldsymbol{\Theta} \in \mathbb{R}^{n_f \times n_y}} \|\boldsymbol{Y} - \boldsymbol{\Phi}(\boldsymbol{X})\boldsymbol{\Theta}\|_2^2 \tag{3.7}$$

That is why this method is called *Least Squares* (LS). It is proven that if the data $\boldsymbol{Y}$ affected my white noise, and the data $\boldsymbol{X}$ are known precisely, the solution converges to the real parameters $\boldsymbol{\Theta}_{\text{true}}$ when the number of observations m goes to infinity.

$$\lim_{m \to \infty} \boldsymbol{\Theta_{LS}} = \boldsymbol{\Theta}_{\text{true}} \tag{3.8}$$

Is this considered machine learning? Yes, even being just a simple implementation of linear algebra, once programmed in a computer, it qualifies as the (simplest) machine learning algorithm because fitting new data does not require any human intervention. Let's see an example. Suppose to have 400 data points, shown in **figure 3.1a**, of the variable x, y_1 and y_2 sampled with noise, that we call Feature 1, Feature 2 and Feature 3, respectively. Suppose that it is known that the output features are linked to the input feature with a linear combination of the functions e^x, x^3, $\cos(x)$, $\sin(x)$ and $\cos^3(x)$, but the parameters θ are unknown:

$$y_1 = \theta_{1,1} e^x + \theta_{2,1} x^3 + \theta_{3,1} \cos(x) + \theta_{4,1} \sin(x) + \theta_{5,1} \cos^3(x) \tag{3.9}$$

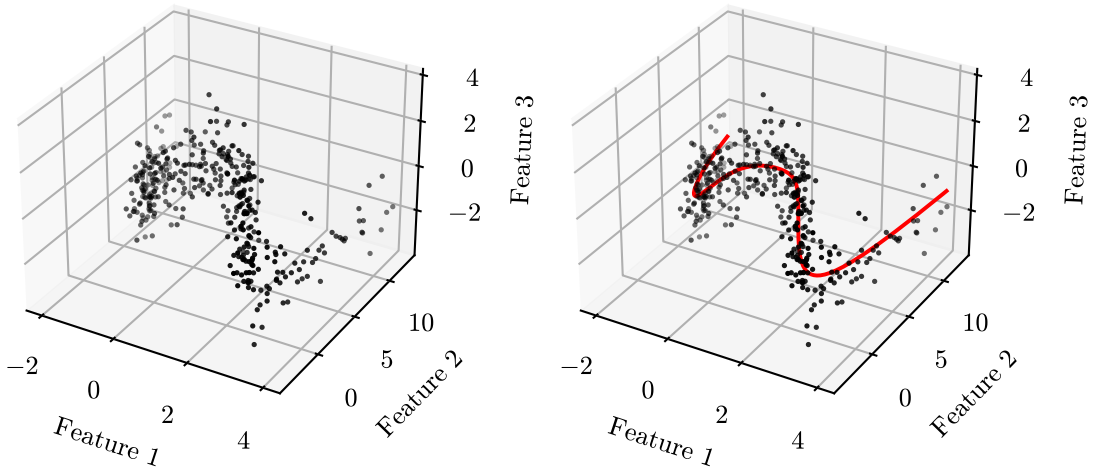
$$y_2 = \theta_{1,2} e^x + \theta_{2,2} x^3 + \theta_{3,2} \cos(x) + \theta_{4,2} \sin(x) + \theta_{5,2} \cos^3(x) \tag{3.10}$$

rearranging in matrix form:

$$\underbrace{\begin{bmatrix} y_{1,1} & y_{1,2} \\ y_{2,1} & y_{2,2} \\ \vdots & \vdots \\ y_{m,1} & y_{m,2} \end{bmatrix}}_{\boldsymbol{Y}} = \underbrace{\begin{bmatrix} e^{x_1} & x_1^3 & \cos(x_1) & \sin(x_1) & \cos^3(x_1) \\ e^{x_2} & x_2^3 & \cos(x_2) & \sin(x_2) & \cos^3(x_2) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ e^{x_m} & x_m^3 & \cos(x_m) & \sin(x_m) & \cos^3(x_m) \end{bmatrix}}_{\boldsymbol{\Phi}(\boldsymbol{X})} \cdot \underbrace{\begin{bmatrix} \theta_{1,1} & \theta_{1,2} \\ \theta_{2,1} & \theta_{2,2} \\ \theta_{3,1} & \theta_{3,2} \\ \theta_{4,1} & \theta_{4,2} \\ \theta_{5,1} & \theta_{5,2} \end{bmatrix}}_{\boldsymbol{\Theta}} \tag{3.11}$$

applying the LS solution from **equation 3.6**, we obtain:

$$\boldsymbol{\Theta_{LS}} = \text{pinv}(\boldsymbol{\Phi}(\boldsymbol{X}))\boldsymbol{Y} = \begin{bmatrix} +1.997 & -0.004 \\ -1.498 & +0.003 \\ +1.332 & -0.018 \\ -0.005 & +0.999 \\ -0.032 & +1.035 \end{bmatrix}$$



(a) *400 data points* **(b)** *data points and the fitted curve*

Figure 3.1: *Least square regression example*

that is quite close to the real parameters used to generate the data:

$$\boldsymbol{\Theta}_{\text{true}} = \begin{bmatrix} +2.0 & +0 \\ -1.5 & +0 \\ +1.3 & +0 \\ +0.0 & +1 \\ +0.0 & +1 \end{bmatrix}$$

Using the estimated parameters, it is possible to estimate the output features for new input features, the regression line is shown in **figure 3.1b**.

Applicability

This is an elegant closed-form solution for a regression problem, however, it has some limitations:

- if the noise is not white, or it is present also in the input features, the solution is not guaranteed to converge to the real parameters;
- if there are nonlinearities in the parameters (for example $\sin(\theta_{1,1}x)$), the solution is not applicable;

3.1.2 Gradient Descent GD

To overcome these limitations, another way to estimate the parameters is to use an iterative algorithm that minimizes a cost function over the parameters space. The iterations aim to update the parameters in the direction of the steepest descent of the cost function. This can be done even with nonlinearities in the data, and even if the noise is not white, but has the drawback of the risk of getting stuck in a local minimum of the cost function, starting from a random initialization. Another limitation is the fact that a learning rate η has to be defined, that is a parameter that defines how much the parameters are updated at each iteration. If the learning rate is too small, the algorithm will take a lot of time to converge, if it is too large, the algorithm may overshoot the minimum and avoid convergence.

In the previous closed form solution (**subsection 3.1.1**), the hypotesis function was linear in the parameters $\boldsymbol{Y} = \boldsymbol{\Phi}(\boldsymbol{X}) \cdot \boldsymbol{\Theta}$, so we can call this prediction $\hat{\boldsymbol{y}} = \boldsymbol{h}_{\boldsymbol{\Theta}}(\boldsymbol{x})$.

The cost function to be minimized is usually defined as the mean squared error between the prediction and the real data:

$$\text{MSE}(\boldsymbol{X}, h_{\boldsymbol{\Theta}}) = \frac{1}{m} \sum_{i=1}^{m} (\hat{\boldsymbol{y}}_i - \boldsymbol{y}_i)^2 \tag{3.12}$$

The gradient of the cost function, used by all gradient descent algorithms, is defined as:

$$\nabla_{\boldsymbol{\Theta}} \text{MSE}(\boldsymbol{X}, h_{\boldsymbol{\Theta}}) = \begin{bmatrix} \frac{\partial}{\partial \theta_1} \text{MSE}(\boldsymbol{X}, h_{\boldsymbol{\Theta}}) \\ \frac{\partial}{\partial \theta_2} \text{MSE}(\boldsymbol{X}, h_{\boldsymbol{\Theta}}) \\ \vdots \\ \frac{\partial}{\partial \theta_{n_f \times n_y}} \text{MSE}(\boldsymbol{X}, h_{\boldsymbol{\Theta}}) \end{bmatrix} \tag{3.13}$$

The algorithm then updates the parameters at each iteration as:

$$\boldsymbol{\Theta}^{(i+1)} = \boldsymbol{\Theta}^{(i)} - \eta \nabla_{\boldsymbol{\Theta}} \text{MSE}(\boldsymbol{X}, h_{\boldsymbol{\Theta}}) \tag{3.14}$$

3.1.3 Stochastic Gradient Descent

The *Stochastic Gradient Descent* (SGD) is a variant of the GD algorithm that computes the gradient only on one instance at each iteration, instead of on the whole dataset. This makes the algorithm much faster, but the cost function will be much more noisy, and theta will not reach a steady value but instead will oscillate around the minimum. This has the advantage of being more robust to local minimum entrapment, but the disadvantage of never reaching the minimum. To overcome this, the learning rate η can be reduced at each iteration, but this will slow down the convergence.

In the **figure 3.2** it is visualized graphically what has been said about Gradient Descent.

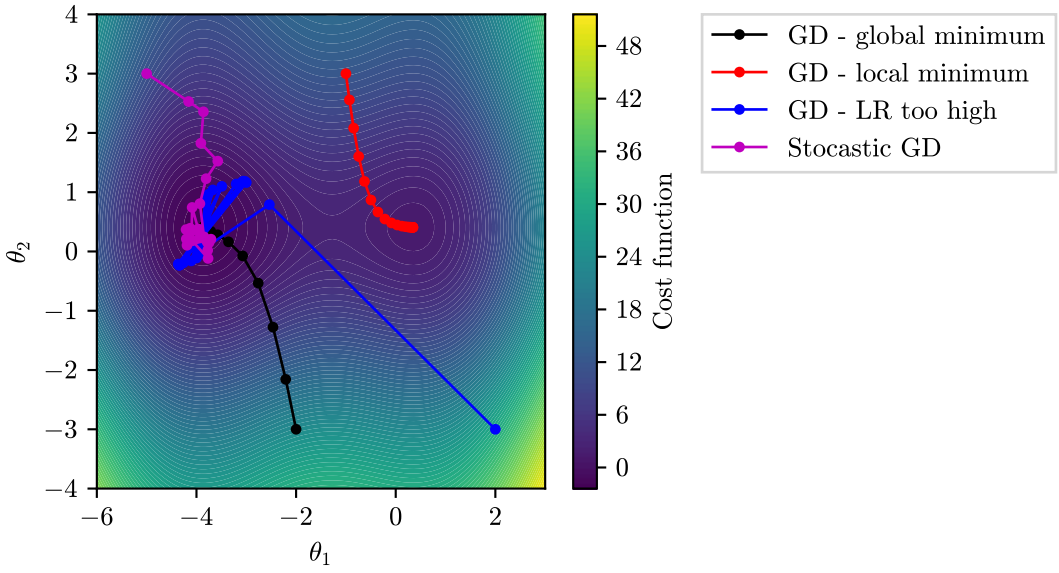


Figure 3.2: *Gradient Descent comparison*

3.1.4 Avoid overfitting

The GD algorithm is very powerful, but it can overfit the data. To avoid that, the problem of when to stop the iterations has to be addressed. A common way to do that is to split the dataset into a training set and a validation set. The training set is used to train the algorithm, and the validation set is used to evaluate the performance of the algorithm on new data. The training is stopped when the performance on the validation set starts to degrade, even if the performance on the training set is still improving. This is called *early stopping*. In the **figure 3.3** it is shown an example of early stopping using as metric the *Root Mean Square Error* (RMSE), that is just the square root of MSE, on the validation set.

3.2 Classification

Another common task in ML is classification. In this case, the algorithm is trained to assign a label to a new instance, based on the training dataset of labelled instances. Naively, it aims to define a set of rules that divide the space of the input features in regions, each one associated with a label. The two main approaches are *hard* and *soft* classification. In the former, the algorithm is trained to assign a single label to each instance, while in the latter, the algorithm is trained to output a probability for each label, and the label with the highest probability is assigned to the instance.

Classification is a *supervised* learning task because the training dataset is labelled. The labels can be provided by a human or can be generated by another algorithm. Some

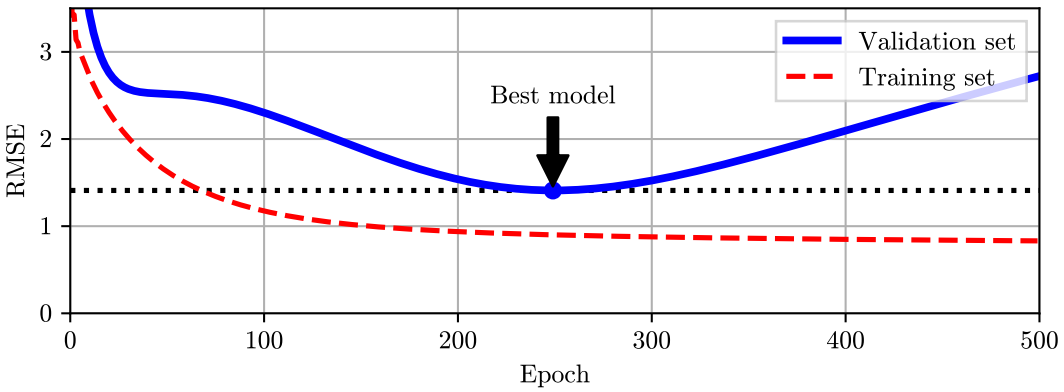


Figure 3.3: *Overfitting example [41, p. 162]*

classification algorithms are available also in the unsupervised version, where the labels are not provided, and the task is usually novelty detection.

3.2.1 Support Vector Machines SVM

Support Vector Machines are simple but powerful classification algorithms that can be used both for hard and soft classification, with medium size datasets. They are based on the idea of finding the hyperplane that best divides the space of the input features into two regions, each one associated with a label.

The main drawback is that, natively, they can only be used for binary classification (two classes), but there are some extensions that allow to use of them for multiclass classification. Furthermore, as will be explained in **section 4.6**, they can be used also for novelty detection (one class). Another limitation is that, being a linear classifier, they can only be used for linearly separable data, but using the *kernel trick*, they can be used also for nonlinearly separable data.

Linear SVM

Looking at **figure 3.4**, it is possible to visualize what the algorithm does: it finds the plane that separates one class from the other, and vice-versa for the second class. In other words, it finds the most distant parallel hyperplanes that separate the two classes. As evident from the figure, the distance between the hyperplanes (called *margin*) is sensitive to the features scaling. The term "support" derives from the fact that only the instances that are on the margin, define (support) the two planes. Those instances are called *support vectors*, and in the figure are highlighted with a grey circle.

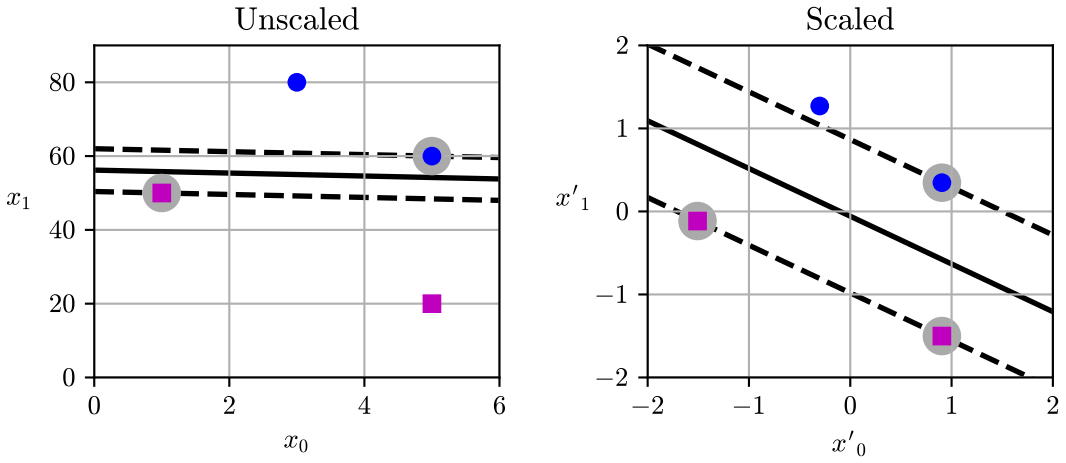


Figure 3.4: *Linear SVM example [41, p. 176]*

Noninear SVM

As said before, the SVM algorithm can be used also for nonlinearly separable data, using the *kernel trick*. The idea is to project the data into a higher dimensional space, where they are linearly separable, and then use the linear SVM algorithm. The projection is done using a *kernel mapping*.

Let's have a look at what is the function for classifying an instance $\boldsymbol{x}^{(i)}$:

$$t^{(i)} = \begin{cases} -1 & \text{if } \boldsymbol{w}^T \boldsymbol{x}^{(i)} + b < 0 \\ 1 & \text{if } \boldsymbol{w}^T \boldsymbol{x}^{(i)} + b \geq 0 \end{cases} \tag{3.15}$$

The model is trainded to find the parameters $\boldsymbol{w}$ and b that:

$$\underset{\boldsymbol{w},b}{\text{minimize}} \ \frac{1}{2}\boldsymbol{w}^T \boldsymbol{w} \tag{3.16}$$

$$\text{subject to } t^{(i)}(\boldsymbol{w}^T \boldsymbol{x}^{(i)} + b) \geq 1 \quad \forall i = 1, \ldots, m \tag{3.17}$$

Since the objective function is convex, and the inequality constraints are differentiable and convex, the solution is the same as the solution of the dual problem [41, p. 188]:

$$\underset{\boldsymbol{\alpha}}{\text{minimize}} \ \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha^{(i)} \alpha^{(j)} t^{(i)} t^{(j)} \boldsymbol{x}^{(i)T} \boldsymbol{x}^{(j)} - \sum_{i=1}^{m} \alpha^{(i)} \tag{3.18}$$

$$\text{subject to } \alpha^{(i)} \geq 0 \quad \forall i = 1, \ldots, m \quad \text{and} \quad \sum_{i=1}^{m} \alpha^{(i)} t^{(i)} = 0 \tag{3.19}$$

Kernel Trick Suppose needing to use a second-degree polynomial mapping, the mapping function is defined as:

$$\phi(\boldsymbol{x}) = \phi(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}) = \begin{bmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{bmatrix} \tag{3.20}$$

Trasforming two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ with the mapping function, to be inserted in **equation 3.18**:

$$\phi(\boldsymbol{a})^T\phi(\boldsymbol{b}) = \begin{bmatrix} a_1^2 \\ \sqrt{2}a_1a_2 \\ a_2^2 \end{bmatrix}^T \begin{bmatrix} b_1^2 \\ \sqrt{2}b_1b_2 \\ b_2^2 \end{bmatrix} = a_1^2b_1^2 + 2a_1b_1a_2b_2 + a_2^2b_2^2 = (\boldsymbol{a}^T\boldsymbol{b})^2 \tag{3.21}$$

So, transforming with a polynomial mapping of degree d, does not require computing the mapping function, but just computing the dot product of the two vectors and elevating it to the degree d, in the dual problem. There also are other kinds of kernels, resumed in the following:

$$\begin{aligned} \text{Linear: } & K(\boldsymbol{a}, \boldsymbol{b}) = \boldsymbol{a}^T\boldsymbol{b} \\ \text{Polynomial: } & K(\boldsymbol{a}, \boldsymbol{b}) = (\gamma\boldsymbol{a}^T\boldsymbol{b} + r)^d \\ \text{Gaussian RBF: } & K(\boldsymbol{a}, \boldsymbol{b}) = \exp(-\gamma\|\boldsymbol{a} - \boldsymbol{b}\|^2) \\ \text{Sigmoid: } & K(\boldsymbol{a}, \boldsymbol{b}) = \tanh(\gamma\boldsymbol{a}^T\boldsymbol{b} + r) \end{aligned}$$

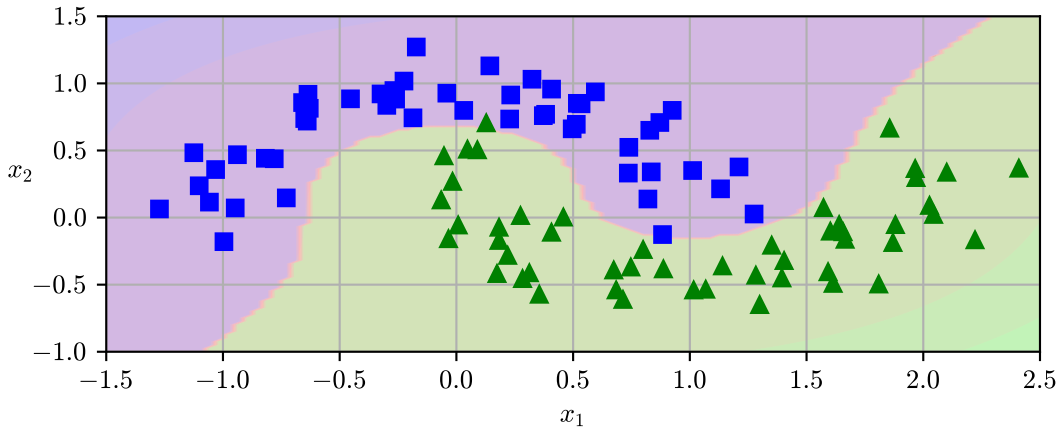


Figure 3.5: *Kernel Trick example [41, p. 180]*

The **figure 3.5** shows an example of SVM classification of data that are not linearly separable.

This topic seems unrelated to the scope of this thesis, but in **section 4.6** we will see how to use the SVM algorithm for novelty detection, as a one-class classifier.

3.2.2 Decision Trees DT

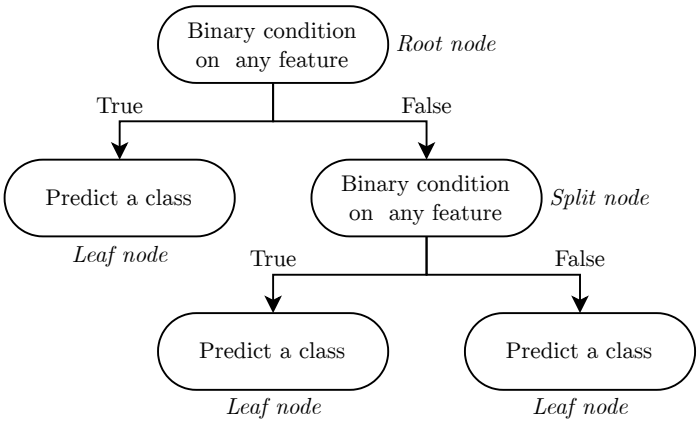


Figure 3.6: *Decision Tree structure*

Decision Trees are very powerful classification algorithms that can also be used for regression, thinking of feature values as classes. The classification process is based on a tree structure, where each sample starts from the root node, and is filtered through a bunch of *if - then* statements until it reaches a leaf node, that outputs the predicted class. In **figure 3.6**, it is illustrated the structure of a very simple binary tree with only a split node and three leaf nodes.

Gini impurity The classification algorithm is hence very simple, the ML part of is the training process. Let's consider a leaf node, and imagine passing all the training samples through the tree. Ideally, all the samples that reach the leaf node (and any other leaf node) should have the same class. This is possible, but a tree that does that is most likely very overfitted to the training dataset and will not perform well on future data. Anyway, the aim of training is to obtain a tree close enough to the ideal one, without overfitting. To do that, there exists a metric called *Gini impurity* that assumes a value of zero if the leaf node is pure (all the samples that reach it have the same class), or a positive value $\in$ (0,0.5] that measure how different the classes in the node are, 0.5 being the maximum value that means that all the classes are present in the node with equal frequency. The mathematical definition is the following:

$$G_i = 1 - \sum_{k=1}^{n} p_{i,k}^2 \tag{3.22}$$

where $p_{i,k}$ is the ratio of class k instances among the training instances in the i^{th} node.

Then the training procedure tries to grow a tree defining the binary conditions that minimize the weighted average of the Gini impurity of the two child nodes, so the cost function is:

$$J(k, t_k) = \frac{m_{\text{left}}}{m} G_{\text{left}} + \frac{m_{\text{right}}}{m} G_{\text{right}} \tag{3.23}$$

where k is the feature index, t_k is the threshold value, m_{left} and m_{right} are the number of instances in the left and right child nodes, and G_{left} and G_{right} are the Gini impurity of the left and right child nodes.

A common way for minimization of the cost function is to use the *Classification and Regression Tree* (CART) algorithm, which is a greedy algorithm that searches for the optimal split at each node, but not for the global optimal tree. The algorithm complexity is $\mathcal{O}(n \times m \log_2(m))$.

Entropy Another metric that can be used instead of Gini impurity inside the same cost function is the *entropy* of the node, which is defined as:

$$H_i = -\sum_{k=1}^{n} p_{i,k} \log_2(p_{i,k}) \tag{3.24}$$

This renders trees very similar to the ones obtained using Gini impurity, but the entropy is slightly slower to compute, due to the logarithm. however, it tends to produce slightly more balanced trees [42].

Avoid overfitting To avoid overfitting the data, the CART algorithm implementation in `sklearn` has some hyperparameters that can be tuned:

- `max_depth`: the maximum depth of the tree;
- `min_samples_split`: the minimum number of samples a node must have before it can be split;
- `min_samples_leaf`: the minimum number of samples a leaf node must have;
- `max_leaf_nodes`: the maximum number of leaf nodes;
- `max_features`: the maximum number of features that are evaluated for splitting at each node.

Increasing the `min` bound, or decreasing the `max` bound, will regularize the model, and reduce the risk of overfitting. In **figure 3.7** it is shown an example of overfitting, where the left plot shows the decision boundaries of a tree with no regularization, and the right plot shows the decision boundaries of a tree with regularization.

Regression As anticipated, the DTs can also be used for regression, in this case, the cost function is the MSE of the predicted value in the leaf node:

$$J(k, t_k) = \frac{m_{\text{left}}}{m} \text{MSE}_{\text{left}} + \frac{m_{\text{right}}}{m} \text{MSE}_{\text{right}} \tag{3.25}$$

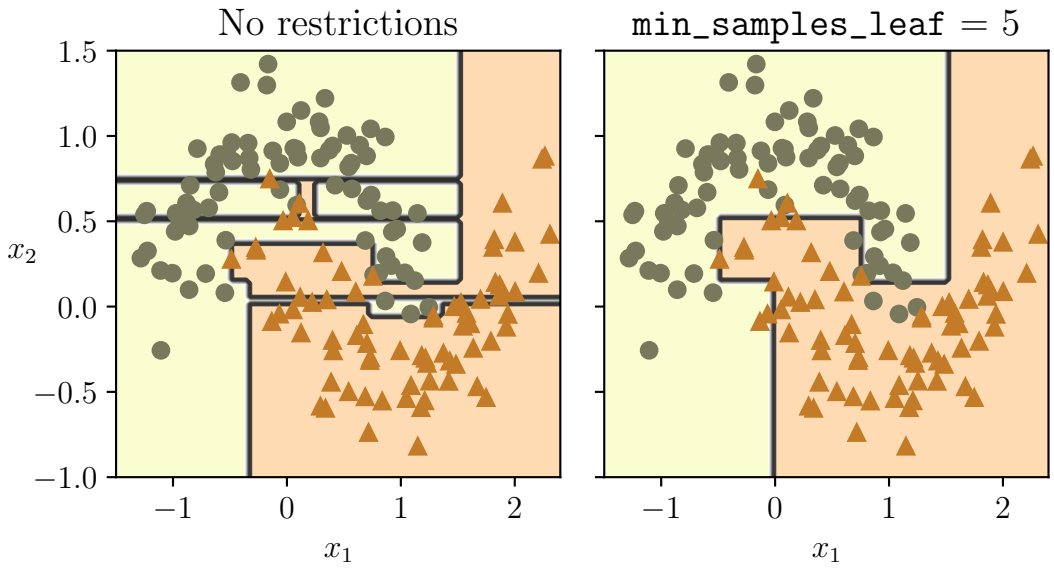


Figure 3.7: *Decision Tree overfitting example [41, p. 203]*

Advantages and limitations The main disadvantages of DTs are that the classification procedure uses thresholds on the value of the features (cutting the hyperspace in orthogonal hyperplanes), so they are sensitive to axis orientation, and they are very sensitive to small variations in the training data. They are also very sensitive to the hyperparameters, so a small variation in constraints leads to very different trees. The main advantages are that they are very fast to train, and the resulting model is very fast to make predictions, they are very easy to understand and visualize and they do not require any feature scaling or centring.

3.2.3 Random Forests RF

The high sensitivity of the DTs to small variations in the training data, can be reduced using the *Random Forests* (RF) algorithm. The idea is to train a bunch of DTs on different random subsets of the training data, and then to average their predictions. The subsets of the training set are usually picked randomly with replacement, this technique is called *bagging* (short for *bootstrap aggregating*).

The benefits of using more threes on subsets of the training data are shown in the **figure 3.8**. The left plot shows the decision boundaries of a single DT, and the right plot shows the decision boundaries of a RF with 500 trees.

Again, this topic seems unrelated to the scope of this thesis, but in **section 4.4** we will see how to use the RF algorithm for novelty detection, exploiting the fact that outliers are usually more isolated (require more split nodes to be reached) than the normal instances.

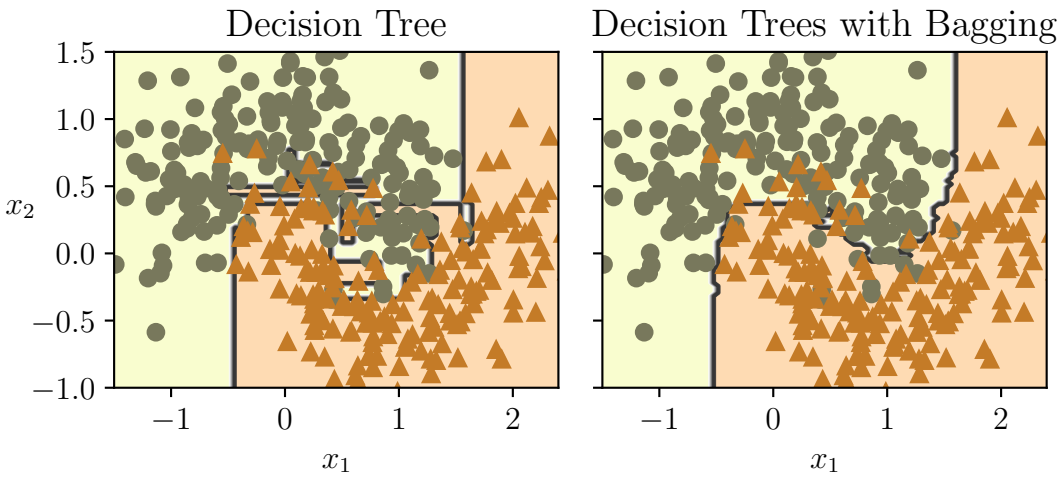


Figure 3.8: *Random Forest example [41, p. 218]*

Chapter 4

Unsupervised Learning

In the previous **chapter 3**, an overview of the most common supervised learning algorithms has been provided, but all these techniques require a labelled dataset. As anticipated in the introduction, most of the time a model of the machine to be monitored is not available. Furthermore, usually a prior knowledge of the behaviour of the machine in the *healthy* or a *faulty* state is not available either. Even in the best-case scenario, where some data collection has been done, the data will be unlabeled.

To address this scenario, two approaches are possible: either label the data or use an unsupervised learning algorithm. The former would be tedious and time-consuming in the case the dataset contains both healthy and faulty data. This is because the faulty data would have to be labelled by hand. If the dataset contains only healthy data, it would be trivial to automatically label all the instances as healthy and use a supervised learning algorithm, but this would be a stretch of the definition of supervised learning. The latter is a more linear approach since unsupervised learning algorithms are designed to work with unlabeled data.

The most common unsupervised task is dimensionality reduction [41, p. 260] but, in this thesis, the main focus will be on novelty detection, fault detection and predictive maintenance, so the considered UML algorithms will be clustering and density estimation.

Clustering Clustering is the task of grouping together similar instances. The definition of *similar* depends on the algorithm used. The most common algorithms are k-means and DBSCAN. The former is a centroid-based algorithm, it is fast to evaluate a new instance and produce a lightweight model but performs poorly in some conditions that will be described in detail. The latter is a density-based algorithm, it performs better in the condition where k-means fails but has the drawbacks of being much slower to evaluate a new instance, and to perform novelty detection, the proposed solution has to keep all the train data in memory. Both will be described in detail in the following **section 4.1** and **section 4.2**.

Gaussian mixture models The second approach to novelty detection is the use of Gaussian Mixture Models (GMM). This approach is based on the assumption that the data is generated from a mixture of several Gaussian distributions with unknown parameters [41, p. 283]. Then the distribution model can be used for novelty detection. This approach will be described in detail in the following **section 4.3**.

Other approaches At the end of the chapter, some other approaches will be briefly described and tested on the same dataset used for demonstrating clustering and GMM. These approaches are: iForest, LOF and ν-SVM. The first two are based on the assumption that outliers are instances that are isolated from the rest of the data, while the latter is based on a kernelized SVM algorithm.

4.1 K-means

This problem is called k-means because it aims to describe the "clustering" by separating the data into clusters ($\boldsymbol{\mathcal{C}}$), and define each cluster with its mean ($\boldsymbol{c}$). Note that the mean of the cluster, from a physical point of view, is the centre of mass of the cluster itself as if it is composed of unitary point masses located at the positions of the data points. The mean is not necessarily a point belonging to the cluster.

Let's assume to have extrapolated F features from each of our signals, to produce a set $\mathbf{S}$ of n snapshots $\boldsymbol{\mathcal{S}}_i, i \in [1, n], \boldsymbol{\mathcal{S}}_i \in \mathbf{S}$ (every snapshot is a vector of features $\in \mathbb{R}^F$). The task is to define a set $\boldsymbol{\mathcal{C}}$ of k clusters ($k \leq n$) $\boldsymbol{\mathcal{C}}_i, i \in [1, k], \boldsymbol{\mathcal{C}}_i \in \boldsymbol{\mathcal{C}}$ that minimize the squared sum of the distances between the snapshots and the centroids $\boldsymbol{c}_i$ of the clusters they belong to. This is equivalent to finding the centroids that minimize the **variance** of the clusters themselves, so the problem can be formulated as in the **equation 4.1**.

$$\arg\min_{\boldsymbol{\mathcal{C}}} \sum_{i=1}^{k} \sum_{\boldsymbol{\mathcal{S}}_j \in \boldsymbol{\mathcal{C}}_i} \|\boldsymbol{\mathcal{S}}_j - \boldsymbol{c}_i\|^2 = \arg\min_{\boldsymbol{\mathcal{C}}} \sum_{i=1}^{k} |\boldsymbol{\mathcal{C}}_i| \mathrm{Var}\boldsymbol{\mathcal{C}}_i \tag{4.1}$$

Unfortunately, this problem is NP-hard, even for as little as $F = 2$ features considered [43], so it is not possible to guarantee to find the global optimum in a reasonable time.

Anyway, heuristic clustering algorithms were already developed in the 1950s. The first appearance of the term "K-means" was used in 1957 by MacQueen [44], and the algorithm settled to a "standard" version, published by Stuart Lloyd in 1982 [45] (but developed at Bell Labs in 1957).

Among all the unsupervised clustering algorithms, a survey from 2002 [46] stated that K-means "is by far the most popular clustering algorithm used in scientific and industrial applications". A more recent survey from 2019 [40], cited this algorithm as first in the group of four most popular algorithms.

Nowadays, the K-means algorithm is implemented in many libraries, such as `scikit-learn` for `Python`, and others for `C`, `R`, `MATLAB`, etc. However, the runtime performances vary widely depending on the implementation [47]. The problem of the algorithm returning a local minima instead of the global one is still present. Most implementations try to

minimize the probability of returning this sub-optimal result by running the algorithm multiple times with different initializations and then selecting the best result, so the problem of getting a sub-optimal result is not a common issue in practice.

4.1.1 Training

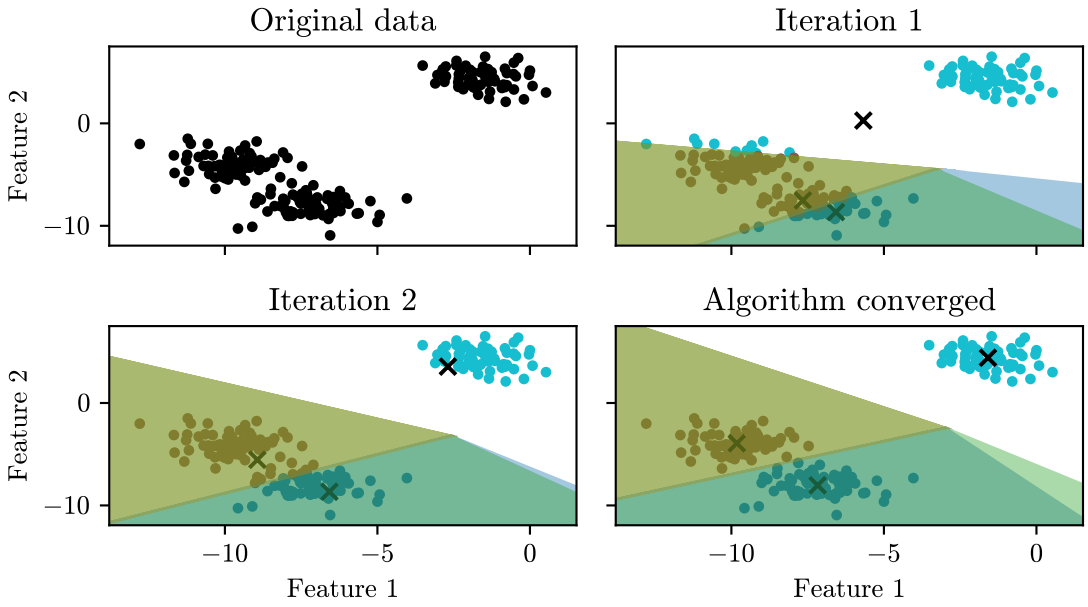


Figure 4.1: *K-means algorithm in the 2-dimensional space*

The naive k-means algorithm consists of a series of iterations. First, the centroids $\boldsymbol{c}_i$ are initialized randomly, then the snapshots are assigned to the nearest centroid, and finally, the centroids are updated as the mean of the snapshots assigned to them. These steps are repeated until the position of the centroids does not change anymore, or a defined maximum number of iterations is reached. This naive algorithm is summarized in the **algorithm 1**.

As an example, we can consider $F = 2$ features, and generate some test points shaped like three separated clusters. In the **figure 4.1** are shown the original data, the first two iterations of the algorithm, and the final result. The K-means algorithm had $n = 200$ snapshots, and $k = 3$ clusters as input. The colours of the dots and the shaded areas represent the clusters and the decision boundaries. The centroids are represented as black crosses. The decision boundaries are a Voronoi tessellation of the space, and they are defined as the set of points that are equidistant from the centroids of two different clusters. The algorithm itself does not compute the boundaries, but it is useful to plot them for visualization purposes.

Algorithm 1 Training of the K-means model

function K-MEANS.TRAIN($\mathbf{S}, k$)
    ▷ $\mathbf{S}$ is the set of snapshots to be clustered
    ▷ k is the number of clusters to be obtained
    $\boldsymbol{c}_i \leftarrow$ random initialization, $\forall i \in [1, k], \boldsymbol{c}_i \in$ Domain of $\mathbf{S}$
    repeat
        ▷ Every snapshot is assigned to the nearest centroid. Every centroid defines a cluster containing the assigned snapshots
        $\boldsymbol{\mathcal{C}}_i \leftarrow \left\{ \boldsymbol{\mathcal{S}}_p : \|\boldsymbol{\mathcal{S}}_p - \boldsymbol{c}_i\|^2 \leq \|\boldsymbol{\mathcal{S}}_p - \boldsymbol{c}_j\|^2 \forall j \in [1, k] \right\} \forall i \in [1, k]$
        ▷ The centroids are updated as the mean of their snapshots
        $\boldsymbol{c}_i \leftarrow \frac{1}{|\boldsymbol{\mathcal{C}}_i|} \Sigma_{\boldsymbol{\mathcal{S}}_j \in \boldsymbol{\mathcal{C}}_i} \boldsymbol{\mathcal{S}}_j, \forall i \in [1, k],$ ▷ $|\boldsymbol{\mathcal{C}}_i|$ is the cluster size
    until All the centroids do not change anymore, or max iterations reached
    $r_i \leftarrow \max \|\boldsymbol{\mathcal{S}}_j - \boldsymbol{c}_i\|, \forall \boldsymbol{\mathcal{S}}_j \in \boldsymbol{\mathcal{C}}_i, \forall i \in [1, k]$
    return $\mathcal{M}_{\texttt{k-means}}$ ▷ The model contains the centroids $\boldsymbol{c}_i$, the radii r_i of the clusters, and the labels of the snapshots
end function

4.1.2 Variations of the K-means algorithm

Kmeans Finding the optimal solution to the k-means problem (**equation 4.1**), as said, is NP-hard. To address this problem, some other heuristics algorithms have been proposed in the last two decades. Simple implementations have time complexity $\mathcal{O}(n^2)$ [48]. This means that most algorithms do not scale well as the number n of snapshot increases. With respect to the number of clusters, the problem has a linear complexity $\mathcal{O}(k)$.

It is worth noticing that an exact solution to the problem has been published [49] with a time complexity $\mathcal{O}(n^{k \cdot F})$. This is still impractical for actual applications, as it is exponential with respect to both the number of clusters and the number of features F.

Lloyd's algorithm The classic Lloyd algorithm [45] has a complexity $\mathcal{O}(n \cdot k \cdot F)$.

Various improvements to Lloyd's algorithm Keeping the same basic idea, various modifications of the Lloyd algorithm have been proposed to improve the performances. For example, Kanugo [50] proposed a local search algorithm that has a complexity $\mathcal{O}(n^3)$. Another result by Malay [48] has a linear complexity $\mathcal{O}(n)$.

Another improvement has been developed by Elkan [51], by keeping track of the bounds from the instances (snapshots) and the centroids. This algorithm becomes convenient when the number k of clusters is large (≥ 20), and up to a dimensionality of $F = 1000$ features.

A variant of the Lloyd algorithm regarding both the speed of execution and the memory consumption has been proposed by Sculley [52]. This solution achieves a reduction of the execution time by orders of magnitude and enables performing the classification even for datasets that don't fit in the memory of the machine. This is achieved by using a *mini-batch* approach, where the centroids are updated after each batch of snapshots.

Concerning the problem of converging to a local minimum, the most common approach is to run the algorithm multiple times with different initializations and then select the best result, this is avoided by Reddy [53], by using the Voronoi tessellation of the hyperspace using the data points to generate the initialization for the centroid positions, this algorithm perform better in the sense that is less likely to get trapped in a local minimum.

K-means++ The last improved algorithm reported in this section has its own paragraph because it is the one used in this thesis. It was developed and named Kmeans++ by Arthur and Vassilvitskii in 2007 [54]. The difference from the Lloyd algorithm is only in the first initialization of the centroids $\boldsymbol{c}_i \forall i \in [1, n]$. In this case, instead of a random initialization for all the centroids, the first centroid $\boldsymbol{c}_1$ is chosen randomly from the snapshots, and then the other centroids are chosen from the remaining snapshots with a probability that depends from the distance of the candidate snapshot to the closest already chosen centroid. This approach is summarized in the **algorithm 2**.

For the development of the framework of this thesis, the K-means++ algorithm has been implemented in `Python`, using the `scikit-learn` library. The library function has been modified, adding a method that returns the radii of the clusters, this information

is crucial for our scope, as it will be needed for evaluating if a new snapshot is a novelty, normal or fault, as it will be explained in the **subsection 4.1.6** and **subsection 4.1.7**.

Algorithm 2 K-means++ algorithm

function K-MEANS++.TRAIN($\mathbf{S}, k$)
    ▷ $\mathbf{S} = \{\boldsymbol{\mathcal{S}}_1, \boldsymbol{\mathcal{S}}_2, \ldots, \boldsymbol{\mathcal{S}}_n\}$ is the set of snapshots to be clustered
    ▷ k is the number of clusters to be obtained
    $\boldsymbol{c}_1 \leftarrow$ random initialization, $\boldsymbol{c}_1 \in \mathbf{S}$
    for $i \leftarrow 2$ to k **do**
        ▷ $D(\boldsymbol{\mathcal{S}})$ is the distance of the snapshot $\boldsymbol{\mathcal{S}}$ from the closest centroid already chosen
        $c_i \leftarrow \boldsymbol{\mathcal{S}}' \in \mathbf{S}$ with probability $\dfrac{D(\boldsymbol{\mathcal{S}}')^2}{\Sigma_{\boldsymbol{\mathcal{S}} \in \mathbf{S}} D(\boldsymbol{\mathcal{S}})^2}$
    end for
    perform the Lloyd algorithm using the calculated $\boldsymbol{c}_i, \forall i \in [1, k]$ as initialization, get the model.
    return $\mathcal{M}_{\texttt{k-means}}$ ▷ The model contains the centroids $\boldsymbol{c}_i$, the radii r_i of the clusters, and the labels of the snapshots
end function

4.1.3 Selecting the number of clusters

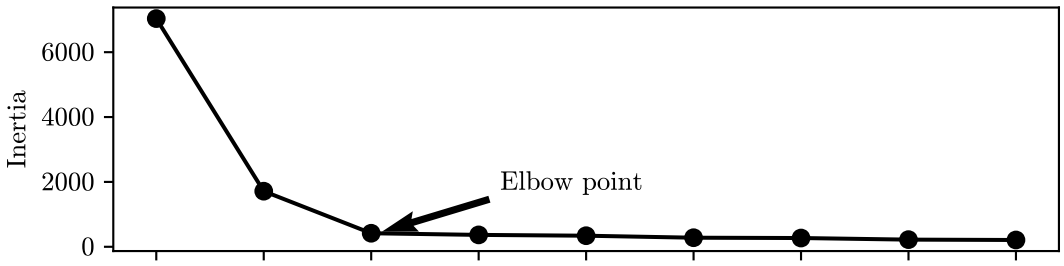
It is important to notice that, even being an *unsupervised* learning algorithm, the K-means algorithm needs to know the number of clusters k in advance. There are some methods to decide what is the best number of clusters, but they usually need to perform more iterations of the algorithm with different values of k, and then compare the results. This task is automatable so, during the training phase, the user can decide the number of clusters to be used, or leave the selection to the algorithm itself.

To compare the results of the different iterations, it is possible to use some metrics on the data and the centroids. The most common metrics are the *inertia* and the *silhouette score*, described in the following paragraphs.

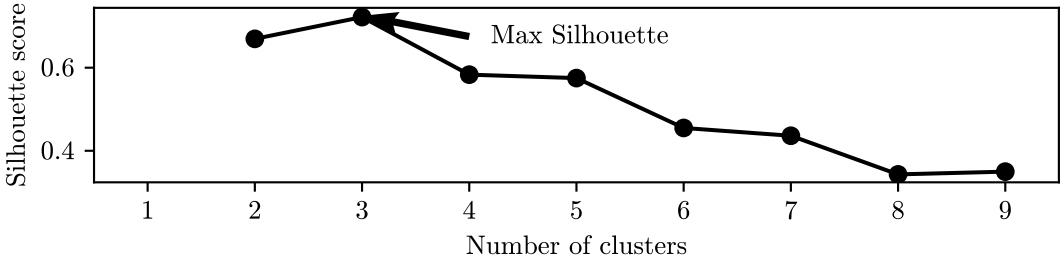
Inertia The inertia metric measures the total (sum) distance of each point belonging to a cluster from the centroid of the cluster itself, as shown in the **equation 4.2**. This is called inertia because, in the physical sense, it is the sum of the moment of inertia of each cluster if all the snapshots were considered as point masses (with unitary mass). This analogy is useful to understand that the lower the inertia, the more compact the clusters are.

Let's span $k \in$ [1,9] and plot the inertia of the clusters for each value of k, on the previous dataset. The result is shown in the **figure 4.2a**. As expected, the inertia decreases as the number of clusters increases. This is not desirable behaviour, if the aim is selecting the number of clusters, the best guess is to select (by eye or with some automatism) the Pareto optimal point (POF) of the curve [55].

$$I = \sum_{i=1}^{k} \sum_{\boldsymbol{\mathcal{S}}_j \in \boldsymbol{\mathcal{C}}_i} \|\boldsymbol{\mathcal{S}}_j - \boldsymbol{c}_i\|^2 \tag{4.2}$$



(a) *Inertia of the clusters for different values of k*



(b) *Silhouette score of the clusters for different values of k*

Figure 4.2: *Metrics for selecting the number of clusters*

Silhouette score A better metric that can be used to select the number of clusters is the silhouette score. The silhouette score is defined for each snapshot as in **equation 4.3**, where a is the mean distance of the snapshot from the other snapshots in the same cluster, and b is the mean distance of the snapshot from the snapshots in the nearest cluster. The resulting silhouette S_i of a snapshot $\boldsymbol{\mathcal{S}}_i$ is a scalar: $S_i \in [-1,1]$. The three relevant cases are:

- a value close to 1 means that the snapshot is far inside its own cluster and far from snapshots of other clusters;
- a value close to 0 means that the snapshot is on the boundary between two clusters;
- a value close to -1 means that the snapshot is far from its own cluster and close to another cluster, so it may have been misassigned.

$$S_i = \frac{b_i - a_i}{\max(a_i, b_i)} \tag{4.3}$$

At this point, the global silhouette score S_g can be computed as the mean of the silhouette scores of all the snapshots (**equation 4.4**). The global silhouette score, for the same example dataset, is shown as a function of the number of clusters k in the **figure 4.2b**. Note that this time $k \in [2,9]$, because the silhouette score is not defined for a single cluster.

In this case, the best value for k is $k = 3$, because it is the value that maximizes the silhouette score. This approach is simpler and easier to automate than the inertia one.

$$S_g = \frac{1}{n} \sum_{i=1}^{n} S_i \tag{4.4}$$

4.1.4 Assignation of the new instance to a cluster

The procedure for assigning the new snapshot $\boldsymbol{\mathcal{S}}_n$ to a cluster is quite simple, it is sufficient to compute the distance between $\boldsymbol{\mathcal{S}}_n$ and the centroids $\boldsymbol{c}_m$, $\forall m \in [1, \ldots, k]$. The distance is defined as the l^2-norm in the feature space, it can be computed using the **equation 4.5**, and assign $\boldsymbol{\mathcal{S}}_n$ to the cluster with the minimum distance.

$$\boldsymbol{d}_{n,m} = ||\boldsymbol{\mathcal{S}}_{n,f} - \boldsymbol{c}_{m,f}||_2 = \sqrt{\sum_{f=1}^{F} (\boldsymbol{\mathcal{S}}_{n,f} - \boldsymbol{c}_{m,f})^2} \tag{4.5}$$

4.1.5 Evaluation of a new instance

At this point, with a model trained on the data, a generic nth new snapshot instance $\boldsymbol{\mathcal{S}}_n$ can be evaluated using the K-means algorithm. From a geometric point of view, the snapshot $\boldsymbol{\mathcal{S}}_n$ is a point in the F-dimensional space, where F is the number of features used to train the model.

For demonstration purposes, in this section, since it is still feasible to show 3D plots, it is considered an example with $F = 3$ features.

In the **figure 4.3**, the training data are represented in the 3-dimensional space, where the axis are the features used to train the model. The K-means model has been ideally trained with an arbitrary number k of clusters but, for display purposes, only two clusters ($\boldsymbol{\mathcal{C}}_i$ and $\boldsymbol{\mathcal{C}}_j$) are plotted.

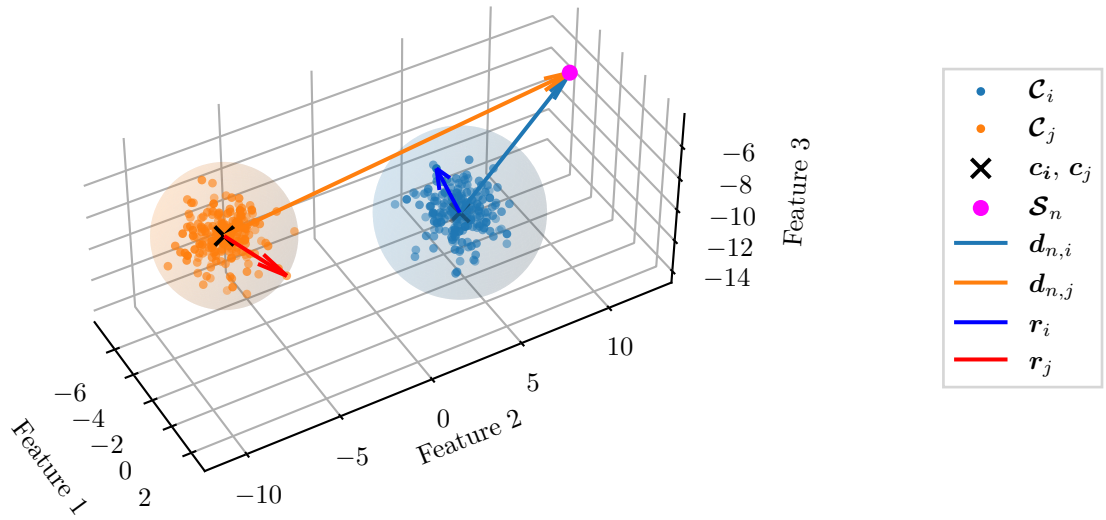


Figure 4.3: *Cluster model in the 3-dimensional space, with new snapshot* $\mathcal{S}_n$

The entities shown in the **figure 4.3** are:

- $\boldsymbol{c}_{i(j)}$ is the centroid of the ith (jth) cluster;
- $\boldsymbol{r}_{i(j)}$ is the radius of the ith (jth) cluster, it is defined as the distance between the centroid $\boldsymbol{c}_{i(j)}$ and the farthest point belonging to the cluster itself;
- $\boldsymbol{\mathcal{C}}_{i(j)}$ is the set of training snapshots belonging to the ith (jth) cluster, it has a centroid $\boldsymbol{c}_{i(j)}$ and a radius $\boldsymbol{r}_{i(j)}$;
- $\boldsymbol{\mathcal{S}}_n$ is the new snapshot to be evaluated;
- $\boldsymbol{d}_{n,i}$ is the vector between $\boldsymbol{\mathcal{S}}_n$ and $\boldsymbol{c}_i$;
- $\boldsymbol{d}_{n,j}$ is the vector between $\boldsymbol{\mathcal{S}}_n$ and $\boldsymbol{c}_j$;
- the semi-transparent spheres represent the cluster sizes, the radius of the spheres is the radius of the cluster itself, and the centre is the centroid of the cluster;

4.1.6 Metric for the new instance evaluation

Once the new snapshot $\boldsymbol{\mathcal{S}}_n$ has been assigned to the right cluster $\boldsymbol{\mathcal{C}}_i$ using **equation 4.5**, some kind of measure (a.k.a. metric) linked to how novel this snapshot is needs to be computed. In this document, this measure, referred to the n-th cluster, will be called e_n, in order to remind some sort of error, even if it is not an error in the strict sense. One simple approach could be to compute the difference between the distance of $\boldsymbol{\mathcal{S}}_n$ from the centroid $\boldsymbol{c}_i$ and the radius $\boldsymbol{r}_i$ of the cluster itself. With this approach, the measure defined in the **equation 4.6** is relative to the current snapshot, so it is possible to use that as a novelty measure.

$$e_n = ||\boldsymbol{d}_{n,i}||_2 - ||\boldsymbol{r}_i||_2, \text{ where } i \text{ is the of the assigned cluster} \tag{4.6}$$

Few consideration about the result of the **equation 4.6**:

- if $e_n > 0$, the new snapshot $\boldsymbol{\mathcal{S}}_n$ is outside the sphere of radius $\boldsymbol{r}_i$ centered in $\boldsymbol{c}_i$, so it is probably a novel snapshot;
- if $e_n < 0$, the new snapshot $\boldsymbol{\mathcal{S}}_n$ is inside the sphere of radius $\boldsymbol{r}_i$, so it is probably a normal snapshot. In this case, it is worth noticing that this assumption is reasonable only if the shape of the point cloud resembles a sphere, otherwise, the radius $\boldsymbol{r}_i$ is not a good measure of the cluster size, and use it for novelty detection would not be reasonable. **This emphasizes the importance of the standardization procedure applied to the features before the training phase**;

Using this metric it is possible to define as *novelty* all the snapshots with $e_n > 0$ and as *normal* all the snapshots with $e_n < 0$. This approach is not very robust because s snapshot that is even slightly outside the sphere of radius $\boldsymbol{r}_i$ will be considered a novelty, but since the sphere is tuned the training *measured* data, that have an aleatory component, this approach will probably detect some novelty even in normal snapshots.

4.1.7 Introducing a threshold for the metric evaluation

In order to improve the robustness of the novelty detection algorithm, it is possible to define a threshold t_i for each cluster $\boldsymbol{\mathcal{C}}_i$ and use it to detect if a snapshot is a novelty or not. Once the threshold t_i is defined, the detection of the novelty can be triggered by the condition $e_n > t_i$.

At this point, the problem is that the user would have to define a threshold for each cluster, and this is not a trivial task. This is because it is likely that the clusters have different sizes, and so one threshold for all the clusters would be more conservative for the smaller clusters and less conservative for the bigger ones. Moreover, most of the times, the clusters' shape and size will not have a physical meaning, and the act of manually defining a threshold for each cluster would go against our goal of designing a fully unsupervised framework.

To address this problem, it is possible to change the definition of the metric itself, so that is not dependent on the cluster size. This can be done by normalizing the already defined metric e_n with the radius $\boldsymbol{r}_i$ of the cluster itself, as shown in the **equation 4.7**. In this way, t_i can be defined as a percentage of the cluster size, so that the user can define a single threshold for all the clusters, and selecting the number to assign to t_i has a more intuitive meaning. From now on if not otherwise specified, the metric e_n will be this normalized version. Obviously, the metric can be easily displayed as a percentage: $e_{n,\%} = e_n \cdot 100$. This value can be evaluated in real time and plotted in a graph so that the user can see the novelty metric behaviour over time.

$$e_n = \frac{\|\boldsymbol{d}_{n,i}\| - \|\boldsymbol{r}_{n,i}\|}{\|\boldsymbol{r}_{n,i}\|} = \frac{\|\boldsymbol{d}_{n,i}\|}{\|\boldsymbol{r}_i\|} - 1, \text{ where } i \text{ is the of the assigned cluster} \tag{4.7}$$

After applying this scaling, the metric now follows this rule of thumb:

- $e_n \in [-1,0] \implies \boldsymbol{\mathcal{S}}_n$ is a normal snapshot;
- $e_n \in (0, +\infty) \implies \boldsymbol{\mathcal{S}}_n$ is a novelty;

4.1.8 Transformation of the metric for the fault detection

In the previous **subsection 4.1.7** a metric has been proposed to detect how novel a snapshot is.

Let's assume now to have trained the model on a dataset of snapshots collected in a period in which the system was having a known malfunction. In this case, the metric applied to any future snapshot will carry the information of "how faulty" the system is, or better "how similarly the system is behaving w.r.t. a known malfunction".

Since the metric already defined is the normalized distance of the current snapshot from the centroid of the cluster, if the clustered data are indicative of a malfunction, the

metric e_n will be in the range $[-1,0]$ when the system has a known malfunction present in the training data and e_n will be in the range $(0,+\infty)$ when the system is not behaving as a known malfunction.

To maintain the same behavior of the metric as in **subsection 4.1.7**, a trasformation that maps the range $[-1,0]$ into $(0,+\infty)$, and the range $(0,+\infty)$ into $[-1,0]$ is needed. This can be done by applying a logarithmic transformation to the metric, as shown in the **equation 4.8**. This transformation is applied only if the model is working in fault detection mode, otherwise, the metric has to remain the same as in **equation 4.7**.

$$e'_n = -\ln(e_n + 1) = -\ln\left(\frac{\|\boldsymbol{d}_{n,i}\|}{\|\boldsymbol{r}_i\|}\right), \text{ where } i \text{ is the the assigned cluster} \tag{4.8}$$

In practice, to avoid numerical representation problems, it is mandatory to avoid mapping -1 (when the new snapshot happens to be perfectly in the centre of a cluster) into $+\infty$. To do that, a constant slightly smaller than 1 is multiplied to the metric before applying the logarithmic transformation, so that the function will have the vertical asymptote slightly before -1, and all the inputs $\in [-1,0]$ will be mapped into real values. The plot of the settled function is shown in the **figure 4.4**.

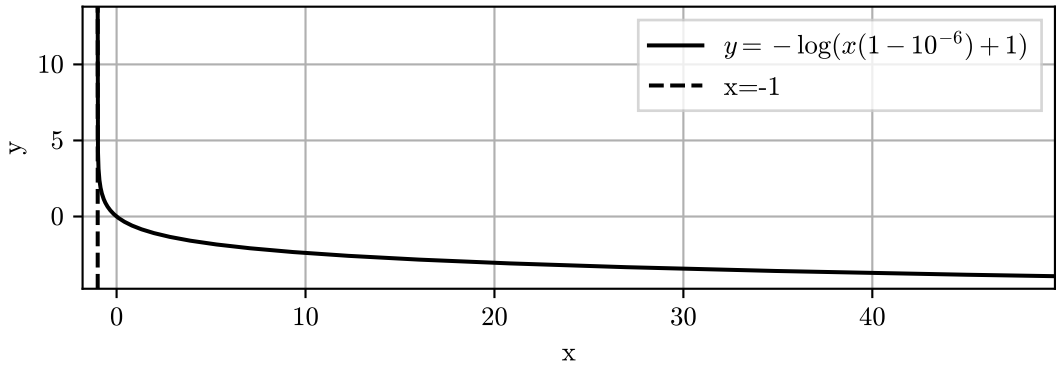


Figure 4.4: *Logarithmic Transformation applied to the metric in case the model is working in fault detection mode*

4.1.9 Evaluation procedure

The evaluation procedure developed to address the novelty/fault detection scope of this thesis can be summarized in the **algorithm 3**.

Algorithm 3 Evaluation of a new snapshot with a K-means model

procedure EVAL($\mathcal{M}_{\texttt{k-means}}, \boldsymbol{\mathcal{S}}, t$)
  ▷ $\mathcal{M}_{\texttt{k-means}}$ is the trained K-means model
  ▷ the model contain the centroids $\boldsymbol{c}_i$ and the radii $\boldsymbol{r}_i$ of the clusters
  ▷ $\boldsymbol{\mathcal{S}}$ is the new snapshot to be evaluated
  ▷ t is the threshold for the novelty detection
  $k \leftarrow$ number of clusters in $\mathcal{M}_{\texttt{k-means}}$
  min $\leftarrow \infty$ ▷ initialize the minimum distance
  for $i \leftarrow 1$ to k **do**
    $\boldsymbol{d}_i \leftarrow \boldsymbol{\mathcal{S}} - \boldsymbol{c}_i$
    if $\|\boldsymbol{d}_i\| <$ min **then**
      min $\leftarrow \|\boldsymbol{d}_i\|$
      $i_{\text{min}} \leftarrow i$
    end if
  end for
  $e \leftarrow \dfrac{\|\boldsymbol{d}_{i_{\text{min}}}\|}{\|\boldsymbol{r}_{i_{\text{min}}}\|} - 1$ ▷ compute the novelty metric
  if fault detection mode **then**
    $e \leftarrow -\ln(e \cdot (1 - 10^{-}6) + 1)$ ▷ apply the logarithmic trasformation
  end if
  if $e > t$ **then**
    return novelty flag, e ▷ the snapshot is novelty
  else
    return normal flag, e ▷ the snapshot is normal
  end if
end procedure

4.1.10 Comment about selecting the wrong value of k

A brief comment about what has been done in the experimental phase, in the cases where the number of clusters k was difficult to define based on the silhouette plots. This happened when the maximum value of k was shared between more than one value of k (multiple peaks with similar values or a flat shape of the peak).

In this case, the best choice is to select the maximum value of k that is still compliant with the silhouette criterion. This is because the in scope of this thesis the final goal is to detect novelty.

For display purposes, let's consider an example in the plane ($F = 2$), with a dataset that clearly has three distinct clusters. Looking at **figure 4.5** it is clear why it is better to select k larger than the actual number of clusters, rather than selecting it too small. This is because, if k is larger than the actual number of clusters, the algorithm will still be able to detect the novelty, but if k is too small, somewhere the algorithm will be forced to join two clusters together, and a new snapshot that happens to be in the middle of the two clusters will not be detected as novelty.

In the case of the **figure 4.5**, the best actual number of clusters was $k = 3$, so selecting $k = 2$ the algorithm would have joined the two clusters on the right, and the new snapshot would not have been detected as a novelty. On the other hand, by selecting $k = 4$, the algorithm would have correctly detected the novelty, even if one cluster had been split in two.

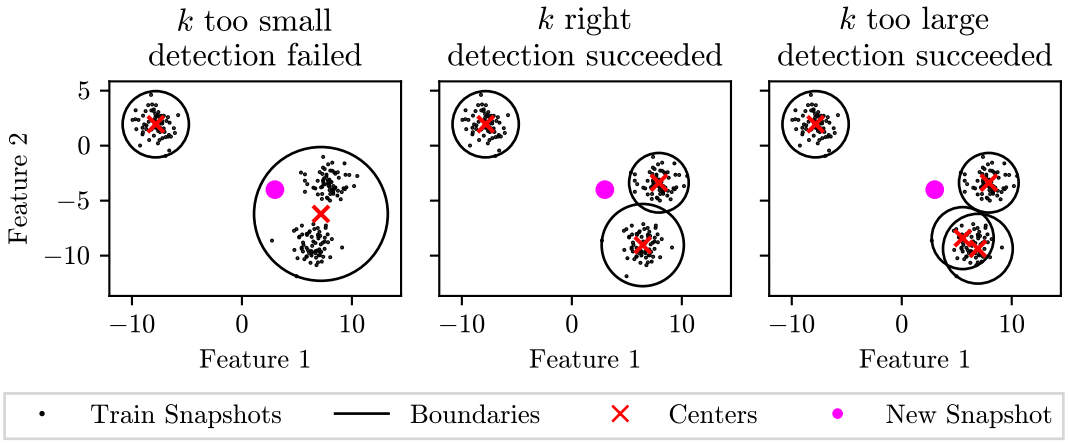


Figure 4.5: *Novelty detection of a new $\boldsymbol{\mathcal{S}}_j$ with different values of k*

4.1.11 Limits of the k-means algorithm

This algorithm has many advantages: it's simple, popular (easy to find in libraries), it's fast, produces a model that does not need to store the original data in order to perform the classification of a new instance and is well scalable. But it has also some limits, the most important are [41, p. 273]:

- it performs poorly if the clusters are not spherical;
- it performs poorly if the clusters have very different sizes;
- it performs poorly if the clusters have different densities.

For this reason, it is very important to perform a good preconditioning of the data. In this thesis, the data are standardized before the training phase, this makes the clusters more spherical and with similar sizes.

4.2 DBSCAN

4.2.1 Overview

Looking at **figure 4.6a** we can see that the data are clearly divided into 3 clusters, but a simple K-means will fail to cluster correctly the data because the clusters are very stretched. But how do we humans see that there are 3 clusters? We instinctively look at the density of the points, and we can see that there are 3 areas with a high density of points, and the rest of the space is empty. DBSCAN is a clustering algorithm that tries to mimic this behaviour.

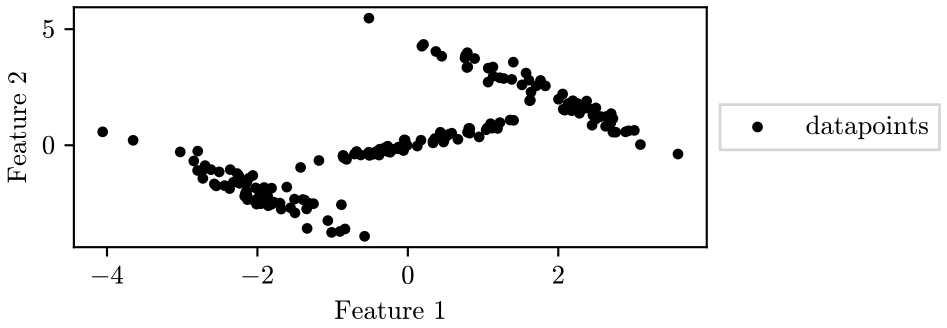
DBSCAN is a density-based clustering designed by Martin Ester, Hans-Peter Kriegel, Jörg Sander and Xiaowei Xu [56]. The algorithm is based on the definition of what a *core point* is, and what a *density-reachable* point is.

The algorithm inputs are:

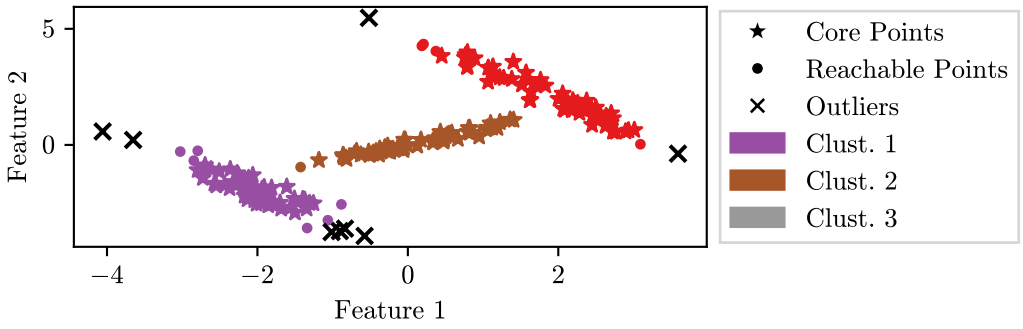
- D: the dataset
- ε: the radius of the neighbourhood
- $MinPts$: the minimum number of points to form a cluster

The basic idea is that if a point has a sufficient number of other points ($MinPts$) in its neighbourhood of radius ε, then it is a *core point*. Chosen a core point, all the other core points in its neighbourhood are assigned to the same cluster, and the process is repeated for all the core points in the neighbourhood of the just assigned ones and so on. At a certain point there will be a cluster of core points that are not near any other core point, when this happens the idea is to include in the cluster also the points that are not core points but are in the neighbourhood of the core points of the cluster.

To better visualize the process, **figure 4.7** shows some data points. Let's assume $MinPts = 5$ The point p_1 is a core point because, in its neighbourhood of radius ε, there



(a) *training data*



(b) *training result*

Figure 4.6: *DBSCAN clustering*

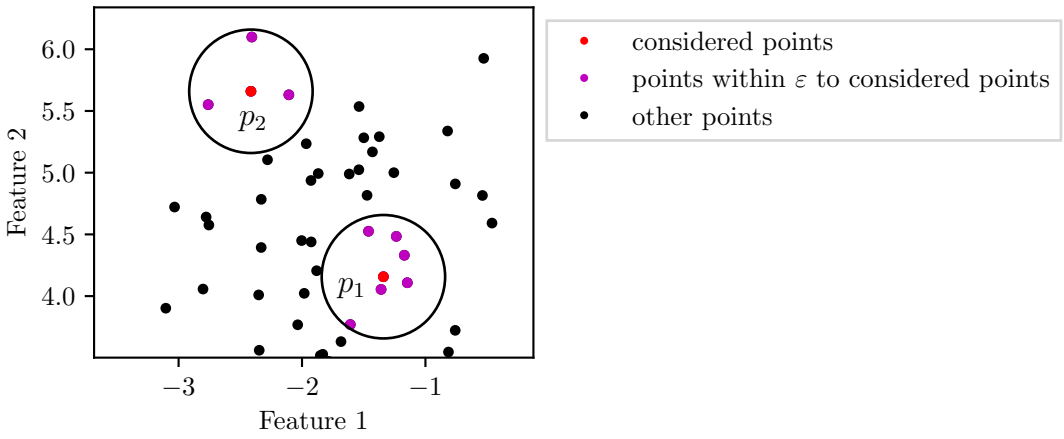


Figure 4.7: *Example of core and border points*

are $7 > MinPts$ points. The point p_2 is not a core point because in its neighbourhood there are only $3 < MinPts$ points. even if it is not a core point, it may be assigned to a cluster, depending on if there is a core point in its neighbourhood. If a point is not a core point and there is not a core point in its neighbourhood, then it is a noise point and it is not assigned to any cluster.

Some of the definitions presented in [56] can be resumed for our purpose as follows:

- $N_\varepsilon(p) = q \in D : \|\boldsymbol{d} \leq \varepsilon\|$ is the set of points in the ε-neighbourhood of p;
- a point p is a *core point* if there are at least $MinPts$ points in the ε-neighbourhood of p;
- a point p is *directly density-reachable* from q if p is in the ε-neighbourhood of q and q is a core point;
- a point p is *density-reachable* from q if there is a chain of points $p_1, \ldots, p_n$ such that $p_1 = q$ and $p_n = p$ and p_{i+1} is directly density-reachable from p_i;
- a point p is *density-connected* to q if there is a point o such that both p and q are density-reachable from o;

In the paper [56] the authors propose a detailed pseudocode of the algorithm, **algorithm 4** is a more abstract version of it.

Algorithm 4 Train DBSCAN

procedure DBSCAN($D, \varepsilon, MinPts$)
  for $p \in D$ **do**
    if $absN_{\varepsilon}(p) \geq MinPts$ **then**
      mark p as a core point
    end if
  end for
  $i \leftarrow 0$ ▷ cluster index
  while there are unassigned points **do**
    $cluster_i \leftarrow \emptyset$
    choose an unassigned point p
    $cluster_i \leftarrow cluster_i \cup p$ ▷ add p to the cluster
    for $q \in$ all reachable points from p **do**
      ▷ all density-reachable points from p are added to the cluster
      $cluster_i \leftarrow cluster_i \cup q$ ▷ add q to the cluster
    end for
    if there are no unassigned core points **then**
      drop all unassigned points from D ▷ they are noise
    end if
    $i \leftarrow i + 1$
  end while
end procedure

4.2.2 Example of DBSCAN clustering

Running the algorithm implemented in the `sklearn` library on the dataset of **figure 4.6a** with $\varepsilon = 0.78$ and $MinPts = 10$ we obtain the result of **figure 4.6b**. The algorithm correctly identifies the 3 clusters, also the noise points number is affected by the choice of ε and $MinPts$.

Note that the DBSCAN is able to identify clusters with arbitrary shapes, even if they are not convex and are not linearly separable. This is a big advantage over the K-means, which is not able to identify clusters with arbitrary shapes.

4.2.3 Choosing the parameters

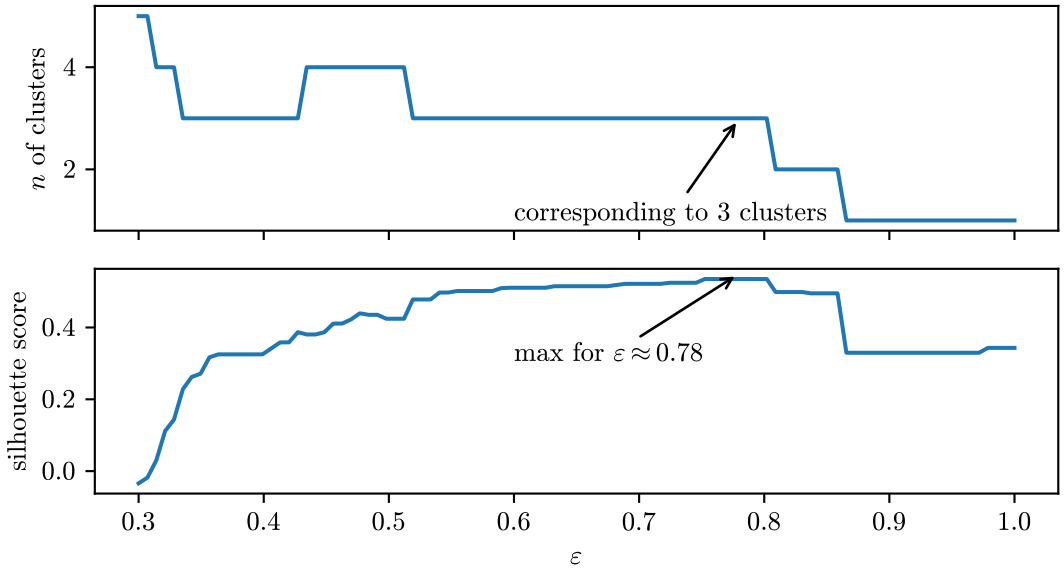


Figure 4.8: *Silhouette score for different values of ε*

Even being the DBSCAN an unsupervised algorithm, it has some parameters that need to be set by the user. The first parameter is ε, the radius of the neighbourhood. This gives a measure of the density we expect from the clusters. As seen for the K-means in **section 4.1**, we can try to find the best ε by using the silhouette score.

Let's plot the silhouette score for different values of ε to confirm that using $\varepsilon = 0.78$ is a good choice. In the **figure 4.8** are shown the results that link the best ε value to 3 cluster being generated.

4.2.4 Evaluation of a new instance

Assume now that the training of the DBSCAN is complete, and a new snapshot $\boldsymbol{\mathcal{S}}_n$ is generated from the sensor data. How can we evaluate if the new snapshot is novel or not? The DBSCAN algorithm is not able to predict the cluster of a new instance. However, the task of this project is novelty detection, not classification. To do that, as has been done for the K-means, we need to provide a metric linked to how novel a new snapshot is.

The idea used previously to compute the distance to a centroid and normalizing it by the radius of the cluster is not applicable here, because the DBSCAN do not use centroids to define clusters. A naive idea for our purpose is to compute the distance of the new snapshot from the closest snapshot in the training dataset. If the distance is greater than a threshold, then the new snapshot is considered novel.

And now the question: if we use the distance of a new $\boldsymbol{\mathcal{S}}_n$ from the closest $\boldsymbol{\mathcal{S}}_i$ in the training dataset (spanning the whole dataset to compute it) why did we train a model in the first place? Wouldn't it be simpler to just apply this metric to the training dataset without clustering it? As far as concerns scope of this thesis, performing the clustering first has two main advantages:

- ε has been chosen running the DBSCAN, so nobody had to guess it. This is a big advantage because the choice of ε would be very difficult to do with limited knowledge of the monitored system. Normalizing the computed distance by ε we can obtain a measure of how novel a snapshot is for which we can set a threshold without tuning it on the specific system;

- during the training phase, the DBSCAN has discarded the noise points. This improves the quality of the metric because the distance from a noise point is not meaningful.

4.2.5 Limitations of DBSCAN

The first limitation of DBSCAN is about cluster density: since ε is fixed, the algorithm is not able to identify clusters with different densities. In this case, it will split low-density clusters into multiple clusters. For our purpose, this is not a problem, because we are interested in novelty detection, so even if a cluster is misidentified as multiple clusters, the new snapshot will be considered novel. To increase the sensitivity of the novelty detection ε can be decreased, but this will increase the number of noise points.

The other main limitation of DBSCAN is that it has a complexity of roughly $\mathcal{O}(m^2)$, where m is the number of points in the dataset. This means that the algorithm is not scalable to large datasets [41, p. 281]. There exist improvements to the algorithm that reduce the complexity to $\mathcal{O}(m \log m)$, like the FDBSCAN developed by Bing Liu [57]. The complexity remains linear w.r.t. the number of features.

4.3 Gaussian Mixture Model

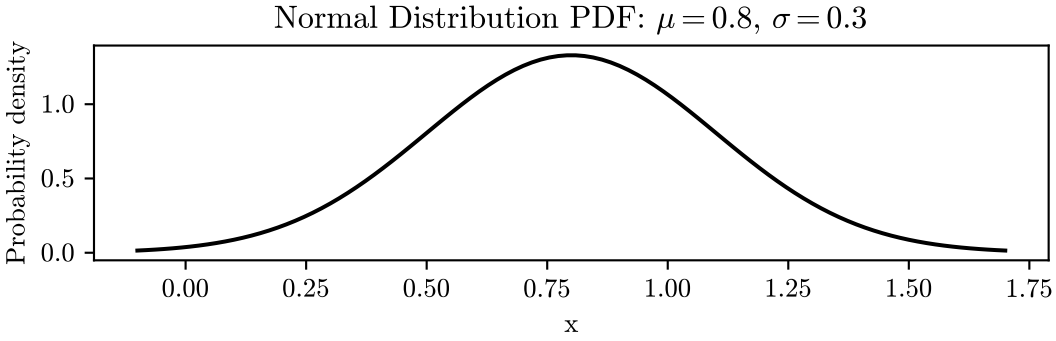


Figure 4.9: *Gaussian distribution probability density function*

The **figure 4.9** illustrates a Gaussian distribution probability density function (PDF), also known as normal distribution. The peak represents the mean μ, while the spread is determined by the standard deviation σ. The equation of the PDF is the following:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

A Gaussian Mixture Model is a probabilistic model that assumes that the data are generated from a mixture of several Gaussian distributions. Depending on the parameters of a Gaussian distribution (mean and variance) on each axis (Features), the data generated from it in the F-dimensional space will have a shape that resembles an ellipsoid with a centre and an orientation.

4.3.1 Training

As anticipated, the assumption is that every cluster has a normal PDF on each axis. So, for each feature, the total PDF is a superposition of k normal PDF. The algorithm used to train the model is the Expectation Maximization (EM) algorithm. This is a generalization of the k-means algorithm that allows to find the centroids (means), the covariance matrices of the clusters, and their weights. Unlike the k-means, the EM is a soft clustering algorithm, meaning that each point is assigned to each cluster with a probability, instead of being assigned to a single cluster. The EM algorithm is an iterative algorithm that aims to find the parameters that maximize the likelihood function.

4.3.2 Selecting the number of clusters

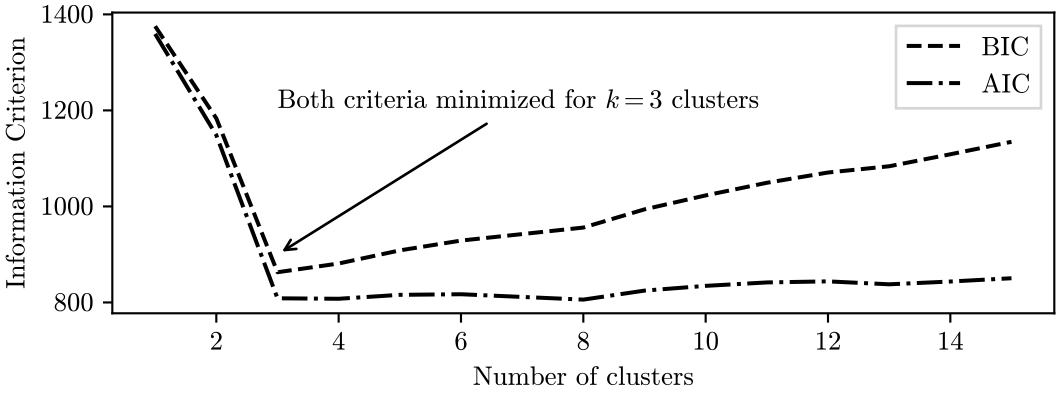


Figure 4.10: *Criteria for selecting the number of clusters*

In **subsection 4.3.1** we have seen that the EM algorithm needs to know in advance the number of clusters k to train the model. To select the best k we could use the same criteria used for the k-means, that is the elbow method or the silhouette score. But, since the EM is most suitable for ellipsoids-shaped clusters of very different sizes, it is better to avoid those metrics that work best with spherical clusters. Instead, we can use the AIC or the BIC criteria, which are functions of the size m of the training dataset, the number p of parameters, and the max value of likelihood function $\hat{\mathcal{L}}$.

$$\begin{aligned} AIC &= 2p - 2\ln(\hat{\mathcal{L}}) \\ BIC &= p\ln(m) - 2\ln(\hat{\mathcal{L}}) \end{aligned} \tag{4.9}$$

Both the criteria in **equation 4.9** penalize the models with a large number of parameters, they usually select the same model, but if they select different models, the AIC tends to select the most accurate model.

Considering, as an example, the same dataset used in **figure 4.6a**, and running the EM algorithm with different values of k, we obtain the results shown in **figure 4.10**. As expected the AIC and the BIC criteria select the same model, that is the one with $k = 3$.

4.3.3 Evaluation of a new instance

Once the model is well-trained, we can use it to evaluate a new instance. The **figure 4.11** shows the result of the training of the GMM on the example dataset. The ellipsoids represent the clusters, the size of the ellipsoids is proportional to the weight of the cluster, while the orientation and the size of the ellipsoids are determined by the covariance matrix of the cluster.

The `pyhton` implementation of the GMM used in this project is the one provided by the `sklearn` library. It has an attribute called `score_samples` that returns the log value

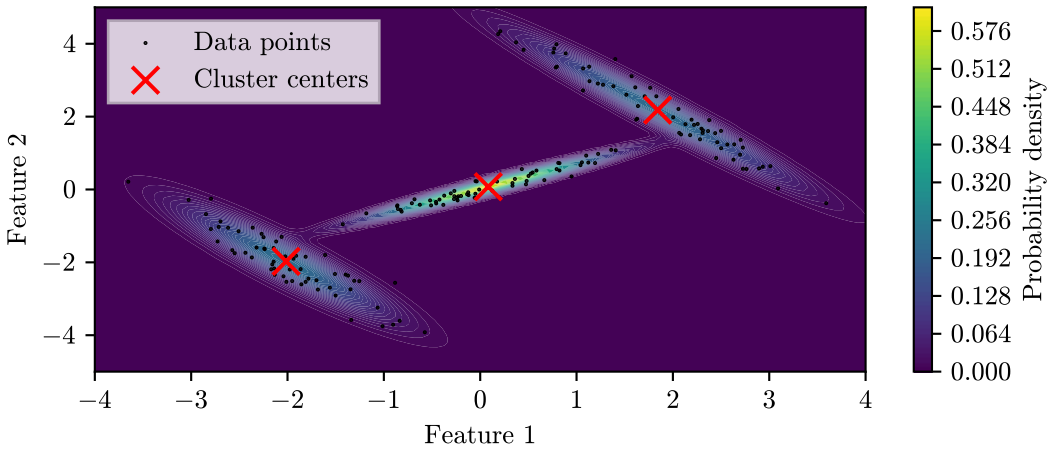


Figure 4.11: *Trained Gaussian Mixture Model*

of the PDF of the samples. The bigger the value, the more likely it is to be inside a cluster. To maintain the same philosophy of the other algorithms, we can take the negative value of `score_samples` as a metric to evaluate the novelty of a new instance.

4.3.4 Selecting the threshold

As said for K-means, and DBSCAN, a threshold will be needed to decide if the new instance is novel or not. If, as supposed for now, the scenario is that the training dataset is composed only of normal instances, then the threshold can be selected by looking at the distribution of the novelty metric on the training dataset and selecting a value slightly higher than the maximum value for the training dataset.

If the scenario is that the training dataset is composed of both normal and anomalous instances sampled at constant intervals, and the ratio between normal and anomalous instances is Known, then GMM enables us to select the threshold in a more sophisticated way.

Suppose for example that 1% of the training dataset is anomalous, since the model gives us the PDF of the samples, we can compute the threshold as the 1% percentile of the PDF of the training dataset. This will ensure that the 1% of the training instances will be classified as anomalous. If the model is correct, also future evaluations will have the same ratio of anomalous instances.

4.3.5 Totally unsupervised approach

Another advantage of the GMM is that it can be used in a totally automated way. This can be done using the variation called Bayesian Gaussian Mixture Model BGMM. This variation is able to assign 0 as weight to some clusters, meaning that those clusters are not used to generate the PDF of the samples. In this case, we can set the number of clusters as high as the size of the training dataset, and the BGMM will select the best number of clusters to use to generate the PDF of the samples.

4.3.6 Limitations of GMM

The main limitation of the GMM is that it is not able to identify clusters with arbitrary shapes.

If the data are not shaped like some ellipsoids, the model can approximate the data anyway, splitting the clusters into smaller ellipsoids. For our purposes of novelty detection, this is not a limitation.

The complexity of the algorithm is $\mathcal{O}(kmF^2 + kF^3)$, where k is the number of clusters, m is the number of samples, and F is the number of features [41, p. 281]. This means that the algorithm is not scalable to a large number of features, but it is scalable to a large number of samples. For our purposes, it's more important to have an algorithm that is fast to evaluate a new instance, than an algorithm that is fast to train.

4.4 Isolation Forest

Similarly to the Random Forest, which is used for classification, the Isolation Forest is an ensemble method that is used for anomaly detection. The idea is to train the decision trees to isolate the data from each other, rather than profiling the normal data, and then provide a score that represents how isolated the data is. The more isolated the data is, the more likely it is to be an anomaly. This algorithm was introduced in 2008 by [58], the paper also provides a way of computing the anomaly score.

This algorithm has the advantage of being very fast to train and requiring very little memory. For hour purposes, as previously said, it is more important to have an algorithm that is fast to evaluate a new instance, than an algorithm that is fast to train.

4.4.1 Training

The algorithm is available in `sklearn` library, and it is very easy to use. This is a truly unsupervised algorithm, the function only takes the dataset as input and manages to automatically select all the rest of the parameters. As an example, I used the same dataset used in the previous sections for the DBSCAN and GMM algorithms. The result is shown in **figure 4.12**.

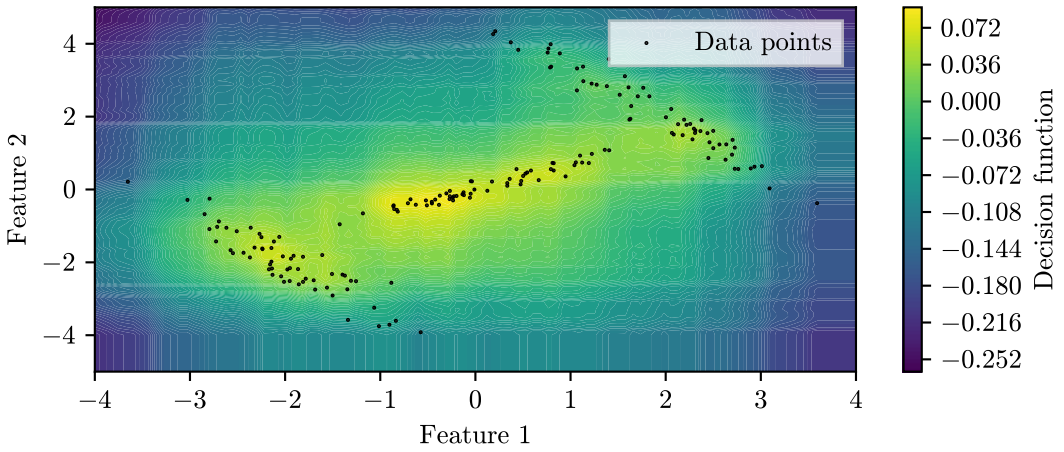


Figure 4.12: *Isolation Forest decision function.*

4.4.2 Evaluation of a new instance

The implementation in `sklearn` provides a function called `decision_function`, plotted as a heatmap in **figure 4.12**. the higher the value, the more likely it is to be a normal instance. To maintain coherence with the work done up to now, we can take the negative value of `decision_function` as a metric to evaluate the novelty of a new instance.

4.4.3 Selecting the threshold

With this algorithm is it difficult to geometrically interpret the decision function, so it is not possible to select the threshold a priori. The approach could be to look at the values of the decision function on the training dataset and select a value slightly higher than the maximum value for the training dataset.

4.4.4 Limitations of Isolation Forest

The main limitation is that since the algorithm is based on decision trees, the decision thresholds are defined on the features axis, so the decision boundaries are highly dependent on the alignment of clusters with the axis. looking back to **figure 4.12**, we can see that the decision function has roughly the same value in the lower right corner as in the lower left corner, even if the lower right corner is clearly more isolated than the lower left corner. This is because the decision boundaries are aligned with the axis, and the lower right corner is more isolated only in the diagonal direction.

The complexity of the training phase is $\mathcal{O}(t\psi \log \psi)$ where t is the number of trees used, and ψ is the size of the subsampling size. The complexity of the evaluation is instead $\mathcal{O}(t \log \psi)$ [58].

4.5 Local Outlier Factor

The Local Outlier Factor (LOF) algorithm is effective in identifying outliers by assessing the density of instances surrounding a particular data point in comparison to the density around its neighbouring points. Typically, an anomaly is considered more isolated than its k nearest neighbours [41, p. 293].

Formally, the authors that designed this algorithm define the LOF of an instance $\boldsymbol{\mathcal{S}}$ as the average of the ratio of the local reachability density of $\boldsymbol{\mathcal{S}}$ and the local reachability density of its $MinPts$ k-nearest neighbours [59]. This is a measure of how isolated the instance is with respect to the surrounding neighbourhood.

Using this approach, this algorithm is able to identify outliers in a dataset with arbitrary shapes. Moreover, it can identify outliers in a dataset with different densities, as a point very near to a very dense cluster can be declared as an outlier, while a point even more distant from a less dense cluster could be declared as normal.

4.5.1 Training

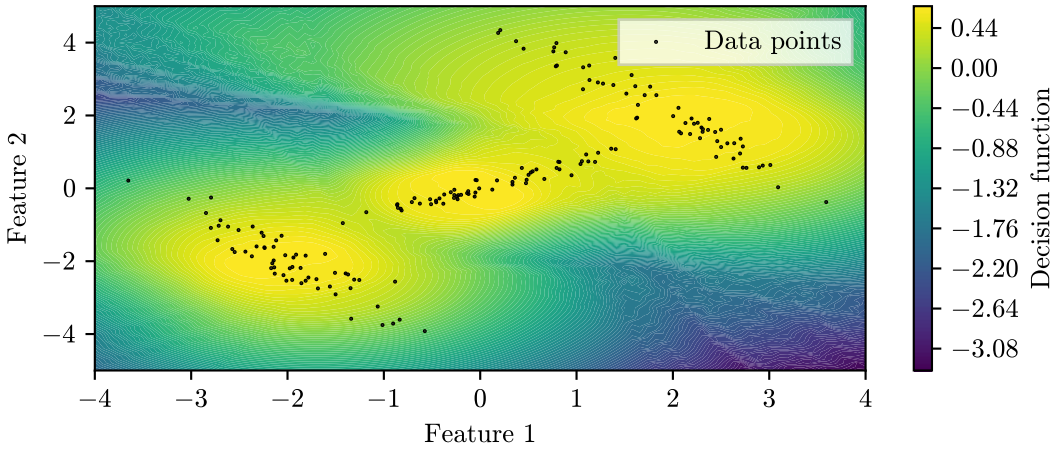


Figure 4.13: *Local Outlier Factor decision function.*

This algorithm is implemented in `sklearn` library and requires the number $MinPts$ of k-nearest neighbours to be specified. According to the authors, the value of LOF of a particular snapshot is neither increasing nor decreasing with the value of $MinPts$. They also suggest an heuristic to select the value of $MinPts$ that is hardly automatable. For this reason, it's difficult to use this algorithm in a real-time scenario in an unsupervised way.

Anyway, as an example, using the same dataset used in **figure 4.6a**, and running the LOF algorithm with the default value of $MinPts = 20$, we obtain the results shown in **figure 4.13**.

4.5.2 Evaluation of a new instance

The `sklearn` implementation of the LOF algorithm has a method that returns the LOF of a new instance. Since the bigger the value is, the more likely it is that the new snapshot is an outlier, we can take directly this value as a metric to evaluate the novelty of a new instance.

To select a threshold value, it holds what has been said in **subsection 4.4.3** about iForest.

4.5.3 Limitations of Local Outlier Factor

The main limitations are the difficulty of defining a threshold value to declare some instances as outliers and using this algorithm in a completely unsupervised way.

4.6 One-Class Support Vector Machine

This algorithm is based on the kernelized SVM algorithm. While the standard application is to find the hyperplane that separates two classes, in this case, the aim is to separate the instances from the origin. If a new instance is too close to the origin (of an augmented hyperspace), then it is considered an outlier. It was introduced in 2001 by Müller [60].

4.6.1 Training

As an example, we refer to the same dataset used in the previous sections for the DBSCAN and GMM algorithms. The training of the version implemented in `sklearn` requires specifying the kernel to use, and the parameter ν that is the upper bound on the fraction of outlier. Training the algorithm on the dataset, with Gaussian kernel and $\nu = 0.002$, we obtain the result shown in **figure 4.14**.

4.6.2 Evaluation of a new instance

The decision function is the relative distance from the separation hyperplane, if it is positive, then the instance is considered normal, if it is negative, then the instance is considered an outlier.

This allows us to take directly the negative of this value as a metric to evaluate the novelty of a new instance (to maintain coherence with previous sections).

Furthermore, similarly to the GMM case (**section 4.3**), if the training dataset contains only normal instances, we have to guess a positive value for the threshold by looking at the value of the metric on the training dataset, but if we have a dataset with a known fraction of outliers, we can use the fact that ν is the upper bound on the fraction of outliers to select ν correctly in the training phase, and then use a threshold of zero for the decision function. This should make at most a ν fraction of future evaluation to be considered outliers.

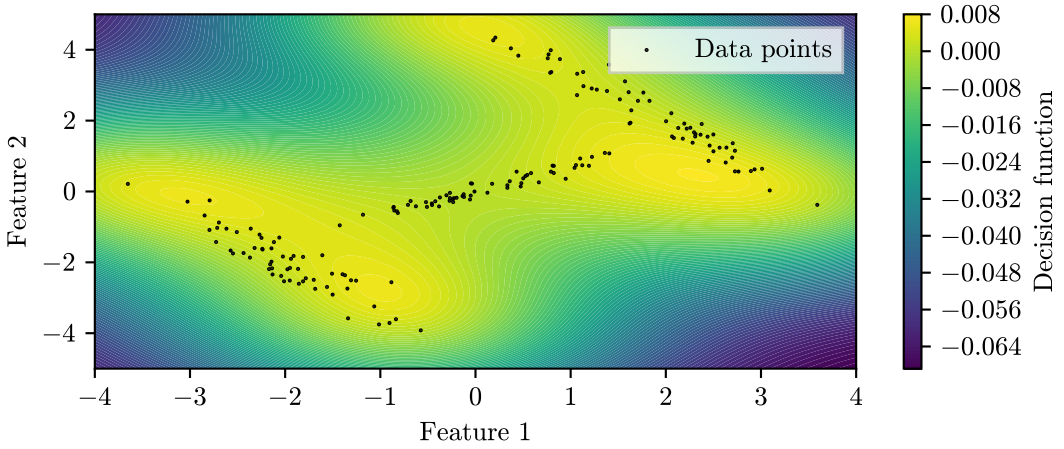


Figure 4.14: *One-Class Support Vector Machine decision function.*

4.6.3 Limitations of ν-SVM

The limitations are about the sensitivity to the choice of the kernel and the parameter ν, making it difficult to use in a completely unsupervised manner.

It works well in high dimensional spaces, but it is not suited for large datasets [41, p. 294]

Chapter 5

Feature Extraction

Before diving into the novelty detection framework itself, the features to be used need to be defined and extracted from the data. Since our goal is to detect a novel behaviour, we are interested in both "time-domain" and "frequency-domain" features. The former are used to capture the temporal evolution of the signal, while the latter are used to capture the spectral content of the signal. In this chapter, we will first introduce the reference dataset that will be used to test the framework and then we will describe the features that will be used in the framework. The ND framework will work on the complete collection of features $\boldsymbol{F} = \{\boldsymbol{F}_t, \boldsymbol{F}_f\}$, where $\boldsymbol{F}_t$ is the collection of time-domain features and $\boldsymbol{F}_f$ is the collection of frequency-domain features. Some of the features will be extracted for all the signals available, while some others only on a subset of the signals, depending on the settings of the framework, as it will be explained in **chapter 6**.

5.1 Reference dataset

In the field of ND, a famous bearing vibration dataset has been collected from the Center for Intelligent Maintenance Systems (IMS) of the University of Cincinnati and made available online on the NASA website [61]. Let's take this as a starting point for our work. The dataset contains the vibration measurements collected at a sampling frequency $f_s = 20$kHz of four forced-lubricated bearings. The shaft was kept constant at 2000rpm during the data collection. The test rig is shown in **figure 5.1** and the test parameters are summarized in **table 5.1**, which also shows the type of faults that happened in each repetition of the test.

To visualize what happens to the vibration signal as the bearing degrades over time, let's consider the "Bearing 3 x" signal from the IMS dataset, shown in **figure 5.2**. The top plot shows the vibration of the new bearing, while the bottom plot shows the vibration of the bearing just before the test was stopped. It's evident that the degraded signal reaches greater peaks in vibration and has a greater variance. The narrow peaks in the degraded signal are due to the presence of fault frequencies, that are not present in the new bearing signal.

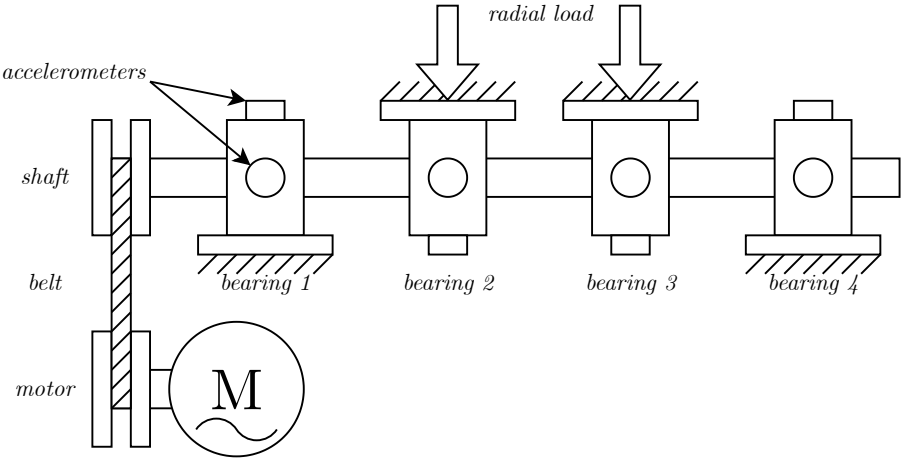


Figure 5.1: *The test rig used by [61]*

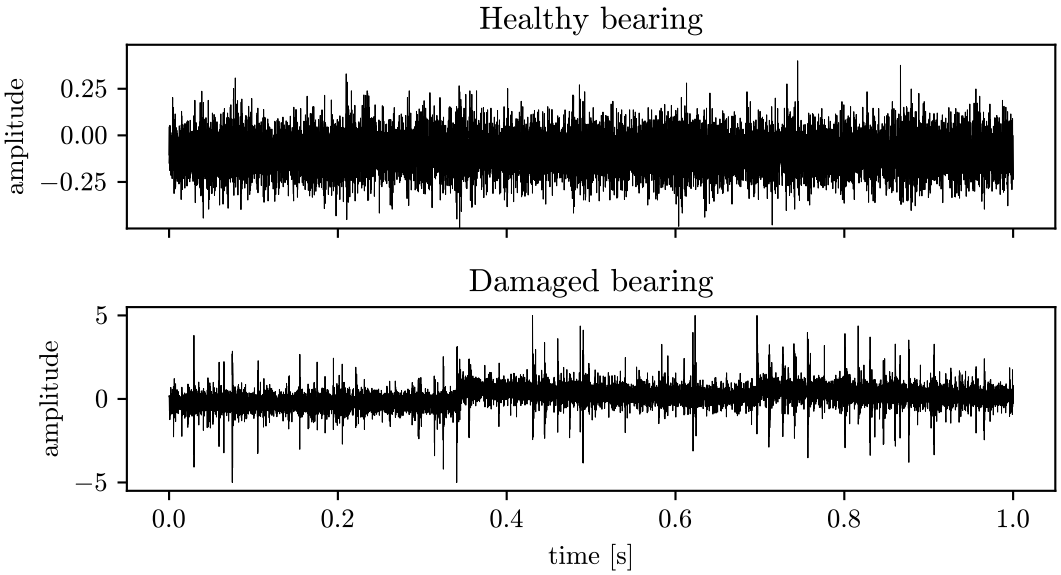


Figure 5.2: *"Bearing 3 x" vibration signal from the IMS dataset*

5.2 Time-domain features

Let's consider a timeserie $\boldsymbol{x}$ containing n samples $x_1, x_2, \ldots, x_n$. The goal of the feature extraction process is to extract a set of features $\boldsymbol{F}_t = \{F_1, F_2, \ldots, F_m\} \in \mathbb{R}^F$ that can be used to describe the signal. In this section, we will describe the features that will be used in the developed framework.

Table 5.1: *IMS Test setup [61]*

	Set No. 1	**Set No. 2**	**Set No. 3**
Recording Duration	22/10/2003 - 25/11/2003	12/02/2004 - 19/02/2004	04/03/2004 - 04/04/2004
No. of Files	2156	984	4448
No. of Channels	8	4	4
Channel Arrangement	Bearing 1 ch 1 & 2 Bearing 2 ch 3 & 4 Bearing 3 ch 5 & 6 Bearing 4 ch 7 & 8	Bearing 1 ch 1 Bearing 2 ch 2 Bearing 3 ch 3 Bearing 4 ch 4	Bearing 1 ch 1 Bearing 2 ch 2 Bearing 3 ch 3 Bearing 4 ch 4
File Recording Interval	5 or 10 min	10 min	10 min
Fault type	Bearing 3: inner race defect Bearing 4: roller element defect	Bearing 1: outer race failure	Bearing 3: outer race failure

Mean The first feature to be considered is simply the mean of the signal. It is defined as

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{5.1}$$

RMS The Root mean square of the signal RMS is related to the power and is defined as

$$\text{RMS} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2} \tag{5.2}$$

Peack-to-peak The peak-to-peak value of the signal is defined as

$$\text{P2P} = \max(\boldsymbol{x}) - \min(\boldsymbol{x}) \tag{5.3}$$

Standard deviation The standard deviation is a measure of the dispersion of the signal and is defined w.r.t. a known distribution with knowledge of the true mean μ_{true} as

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_{true})^2} \tag{5.4}$$

but since in our case we don't know the true mean, we will use the sampled standard deviation, which is the best estimate, defined as

$$\hat{\sigma} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu)^2} \tag{5.5}$$

Skewness The skewness is a measure of the asymmetry of the signal and is defined as

$$\gamma = \mathbb{E}\left[\left(\frac{x-\mu}{\sigma}\right)^3\right] \tag{5.6}$$

and can be estimated on sampled data as

$$\hat{\gamma} = \frac{\sqrt{n(n-1)}}{n-2} \frac{\frac{1}{n}\sum_{i=1}^{n}(x_i-\overline{x})^3}{\left[\frac{1}{n}\sum_{i=1}^{n}(x_i-\overline{x})^2\right]^{3/2}} \tag{5.7}$$

Kurtosis The kurtosis is a measure of the "peakedness" of the signal and is defined as

$$\kappa = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{x_i-\mu_{true}}{\sigma}\right)^4 \tag{5.8}$$

and can be estimated on sampled data as

$$\hat{\kappa} = \frac{(n+1)\,n}{(n-1)\,(n-2)\,(n-3)} \frac{\sum_{i=1}^{n}(x_i-\mu)^4}{k_2^2} - 3\,\frac{(n-1)^2}{(n-2)(n-3)} \tag{5.9}$$

To visualize the evolution of the features over time, let's consider all the snapshots of the "Bearing 3 x" signal from the IMS dataset and plot all the time-domain features described above. The result is shown in **figure 5.3**. Having seen also **figure 5.2**, it's not surprising that the P2P and the kurtosis are the time domain features that capture the degradation of the bearing the earliest.

5.3 Frequency-domain features

To capture also the frequency behaviour of the signal, other tools are needed. In this section, we will describe the frequency-domain features $\boldsymbol{F}_f = \{F_{m+1}, F_{m+2}, \ldots, F_l\}$ that will be used in the developed framework.

5.3.1 Fourier Transform

One powerful tool to analyze the frequency content of a signal is the Fourier transform or, in the case of a discrete signal, the Discrete Fourier Transform(DFT), which has a very efficient implementation in the Fast Fourier Transform (FFT) algorithm [62]. This algorithm is implemented in many programming languages and libraries, including `python` and `C`. The theory behind this topic is briefly described in **Appendix A**.

To have a better understanding of the capability of the FFT to capture the frequency content of a signal (and the presence of a disturbance in a portion of the signal), let's consider the following signal, composed by the sum of four sinusoids with different frequencies that is plotted with its DFT in **figure 5.4**:

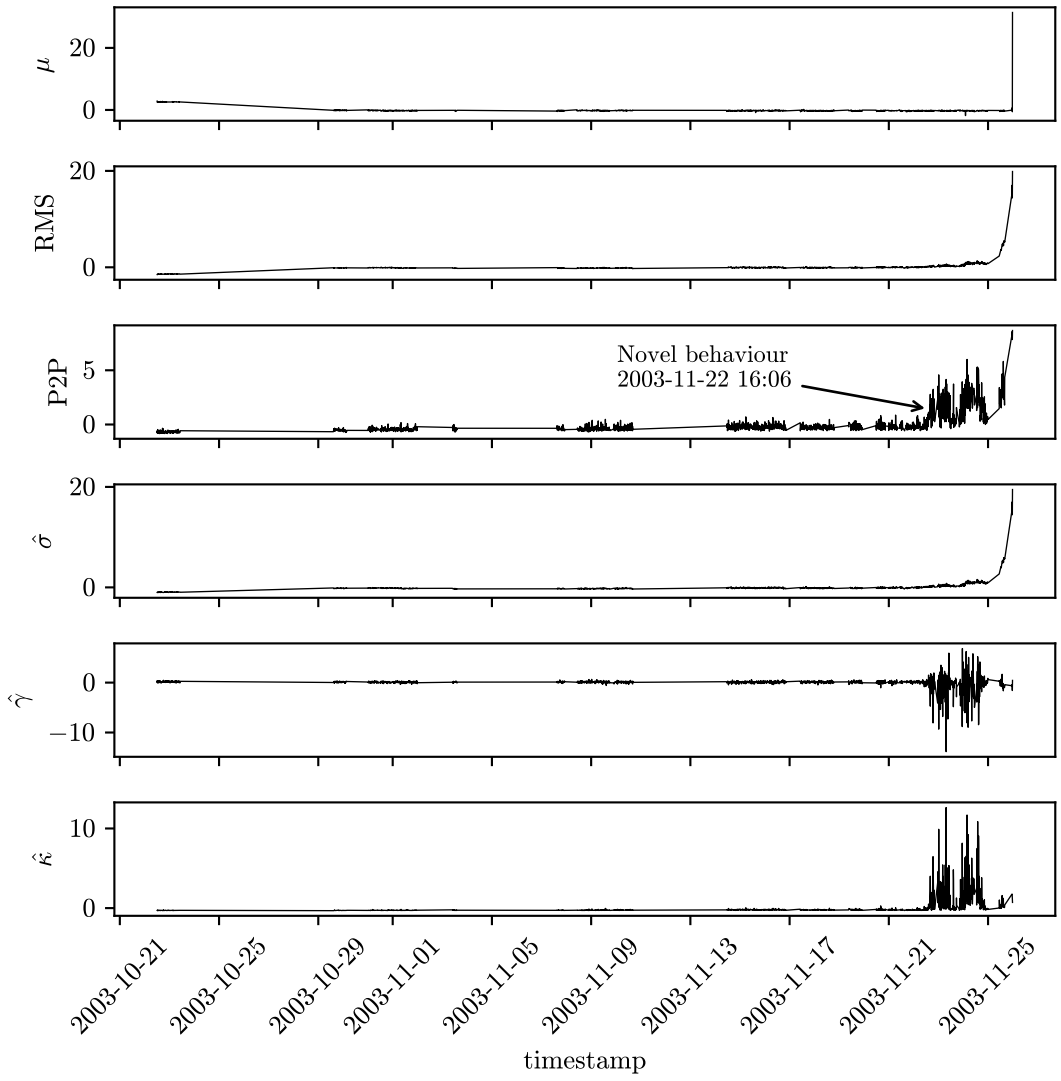


Figure 5.3: *All time-domain features for the "Bearing 3 x" vibration signal from the IMS dataset*

$$x(t) = \sin(2\pi f_1 t) + \sin(2\pi f_2 t) + \sin(2\pi f_3 t) + \sin(2\pi f_4 t),$$
$$t \in [0, 1], \quad \{f_1, f_2, f_3, f_4\} = \{2, 5, 7, 15\}\text{Hz} \quad (5.10)$$

In the **figure 5.4** it is possible to see the four peaks in the frequency domain, corresponding to the four frequencies of the sinusoids. Now, let's consider the same signal, but with an additive disturbance in a small portion of the signal, as shown in **figure 5.5**. The disturbance is a period of the signal $x(t) = \sin(2\pi f_5 t)$, with $f_5 = 50$Hz, that is added to the signal in the interval $t \in [0.4, 0.6]$. The DFT of this signal is shown in **figure 5.5**. It

is possible to see that the disturbance is captured by the spectrum as a broad frequency additional components, instead of a peak at $f_5 = 50\text{Hz}$. This is because the disturbance is not acting on the whole signal. For our purpose, this is still good because we can detect the variation of all the frequencies involved.

Consider the period of the undisturbed signal, that is the LCM of the periods of the four sinusoids composing the signal.

$$T = \text{LCM}\left(\frac{1}{2}\text{s}, \frac{1}{5}\text{s}, \frac{1}{7}\text{s}, \frac{1}{15}\text{s}\right) = 1\text{s}$$

So, we were processing a signal that has a period of one second, sampled for exactly one second. Let's see what happens if we sample the same signal for a time that is not an integer multiple of the period. In **figure 5.6** it is shown the DFT of the same signal as before, but sampled for $t \in [0, 0.9]$. It is possible to see that the peaks are still at the frequencies of the sinusoids composing the signal, but they are not single points as it appears that also frequencies near the true frequencies are present. This phenomenon is called *spectral leackage* and happens because the signal is not sampled for an integer number of periods [63].

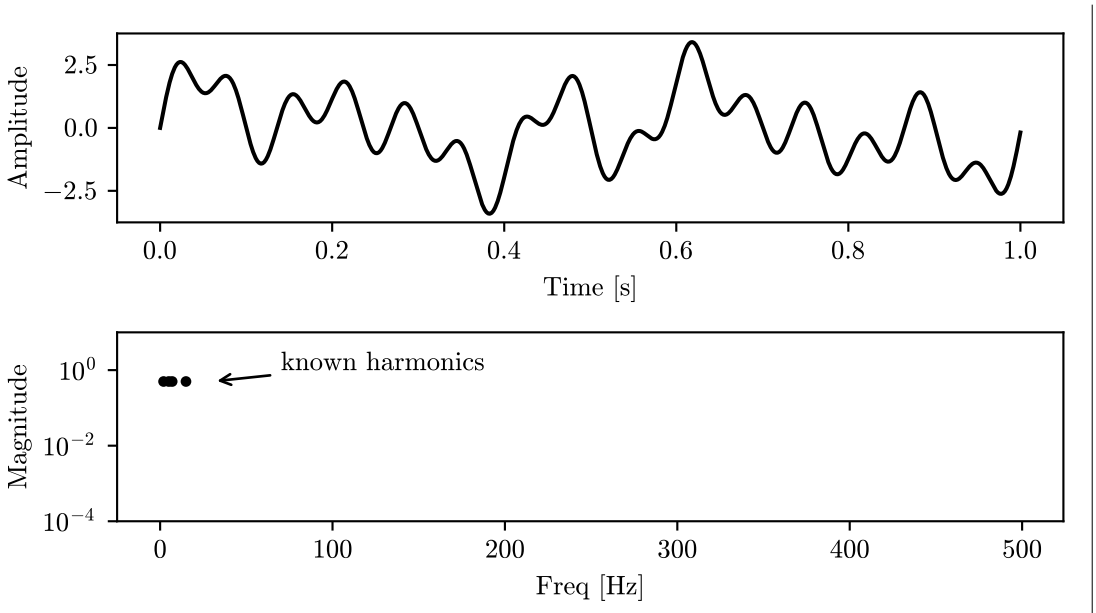


Figure 5.4: *FFT of the signal with known frequency components*

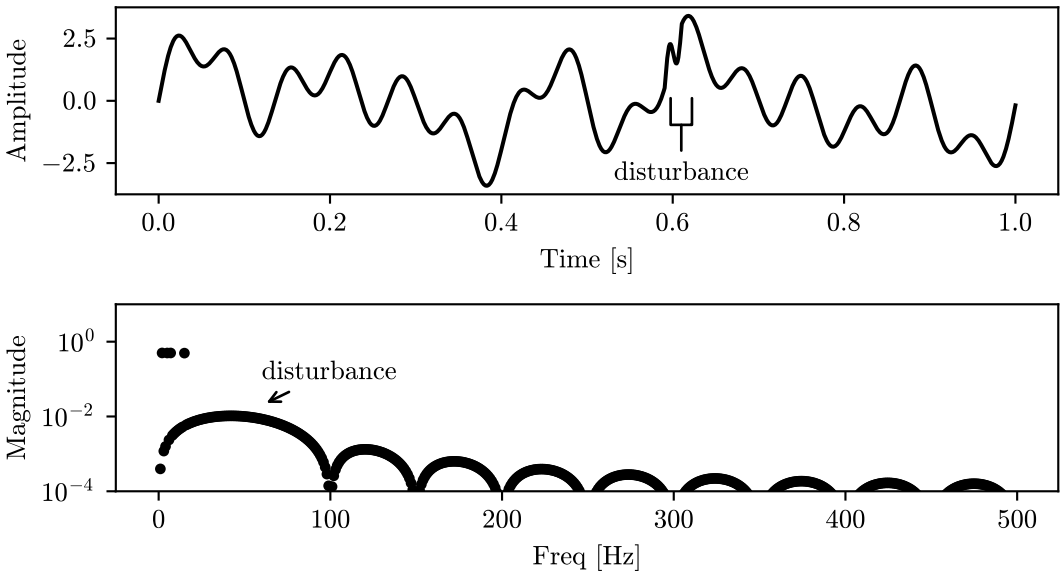


Figure 5.5: *FFT of the signal with known frequency components, and an additive disturbance*

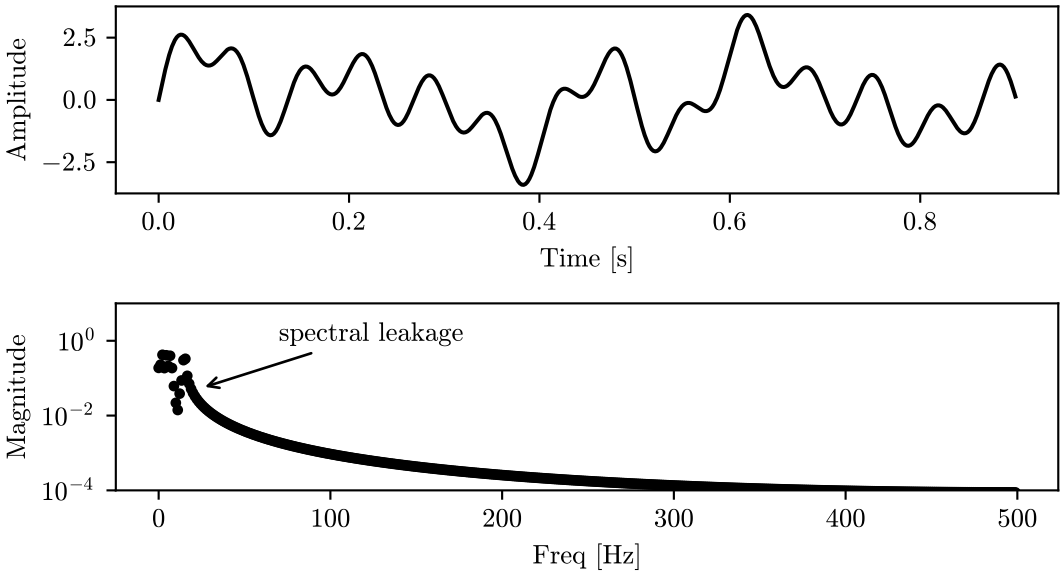


Figure 5.6: *FFT of the signal with known frequency components, with a domain that is not an integer multiple of the period*

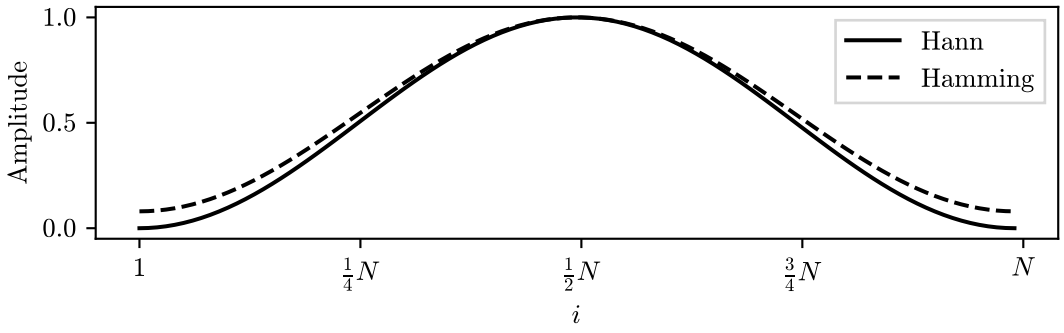


Figure 5.7: *Hann and Hamming windows*

Preprocessing

Up to now, it has been shown that the DFT will translate the presence of a disturbance in the time domain into several periodic components in the frequency domain. However, it has the spectral leakage weakness if the sampling interval is not an integer multiple of the period of the signal which is the most likely scenario in a real application. To compensate for this phenomenon, it is better to use a transformation to the signal that makes it artificially start and end with the same value. This can be done with a *windowing* function, or with the *flip and reverse* method [64].

The most common windowing functions are the *Hann* and the *Hamming* windows, defined as:

$$w_{hann}(i) = 0.5 \left[1 - \cos\left(\frac{2\pi i}{n}\right)\right] \tag{5.11}$$

$$w_{hamming}(i) = \frac{25}{46} \left[1 - \cos\left(\frac{2\pi i}{n}\right)\right] \tag{5.12}$$

where i is the considered sample as independent variable and n is the total number of samples. Those functions are plotted in **figure 5.7**. The preprocessing operation is simply to multiply the signal by the windowing function. this will make the first and last point of the signal exactly zero in the case of the Hann window, and very close to zero in the case of the Hamming window but with the advantage of having a narrower peak in the spectrum.

The "flip and reverse" method is simply to concatenate a flipped copy of the signal to itself and downsampling it to preserve the original length of the array, as follows [64]:

$$x_{new}(i) = \begin{cases} x(2i) & \text{if } i \leq \frac{n}{2} \\ x(2(n-i)) & \text{if } i > \frac{n}{2} \end{cases} \qquad \forall i \in [1, n] \tag{5.13}$$

A comparison of those preprocessing techniques is shown in **figure 5.8** and **figure 5.9**. In both figures, it is evident that the "flip and reverse" method generates a fake peak at half the sampling frequency, while the windowing function doesn't. Moreover, looking

at **figure 5.9**, it is possible to see that the Hann window better suppresses the spectral leakage. In figure **figure 5.10** it is shown the effect of different preprocessing techniques on a real-world signal from the IMS dataset. The Hann window will be implicitly used in the rest of this section.

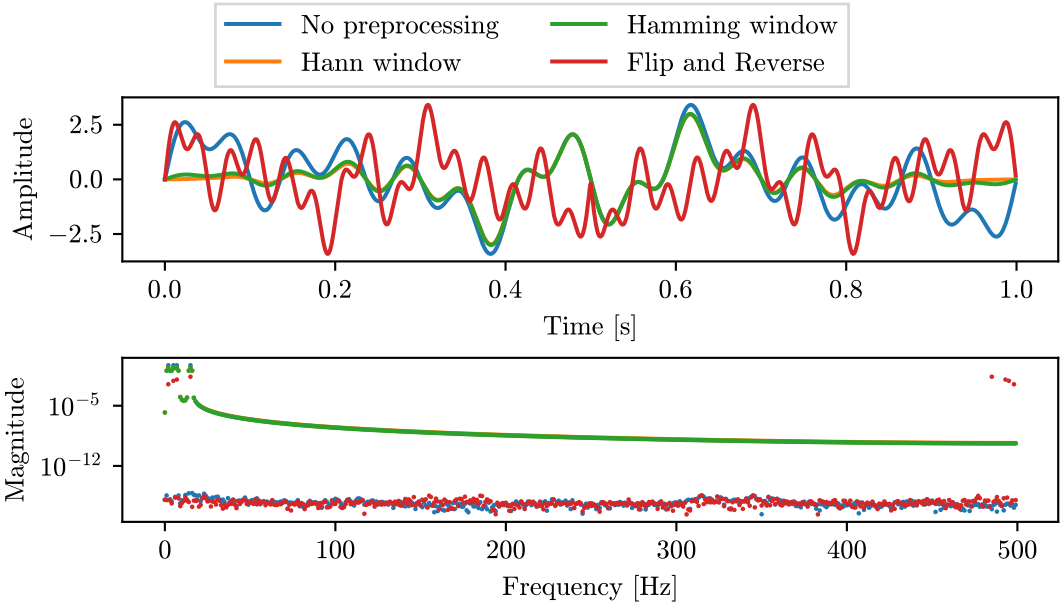


Figure 5.8: *FFT of the signal with a domain that is an integer multiple of the period, and preprocessing techniques applied*

Application to the reference dataset

At this point, to see if the spectrum is representative of a fault, as it has been done for the time-domain features, we can plot all the spectrum for all the snapshots of the "Bearing 3 x" signal from the IMS dataset, as shown in **figure 5.11**. Looking at the picture, we can see that the signature frequencies of faults are present in the spectrum, and they become more and more prominent as the bearing degrades. This is a good sign justifying to use of the spectrum to detect the novel behaviour.

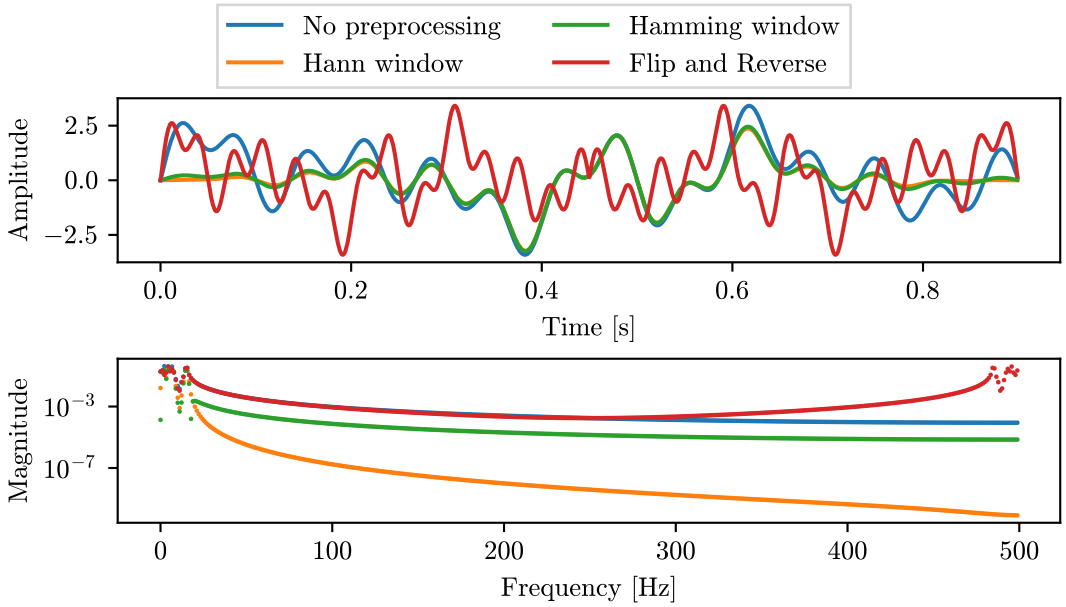


Figure 5.9: *FFT of the signal with a domain that is not an integer multiple of the period, and preprocessing techniques applied*

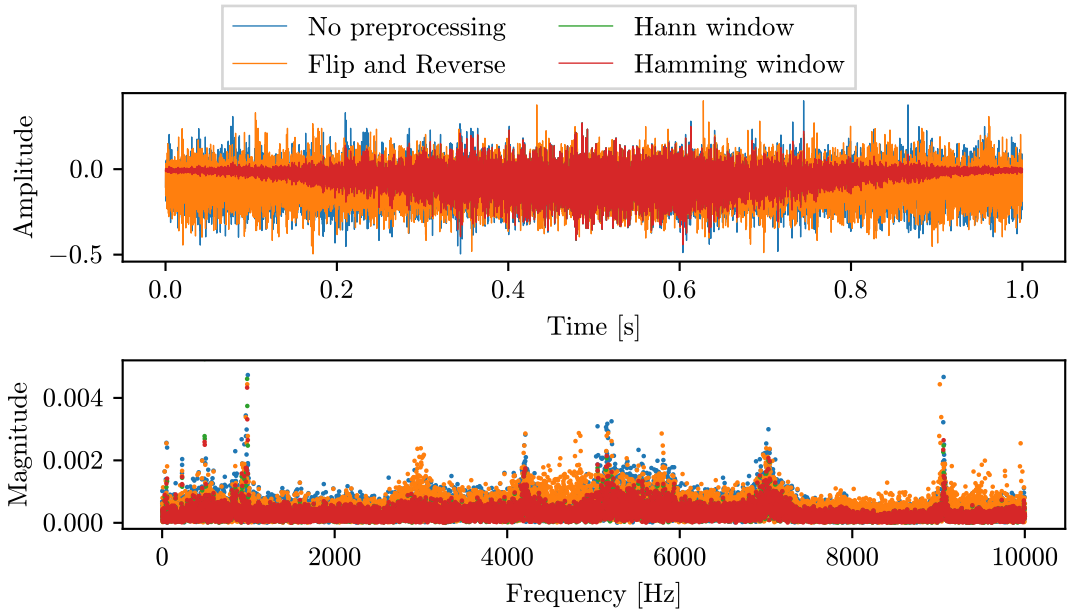


Figure 5.10: *FFT of the "Bearing 3 x" vibration signal from the IMS dataset, in normal conditions, with preprocessing techniques applied*

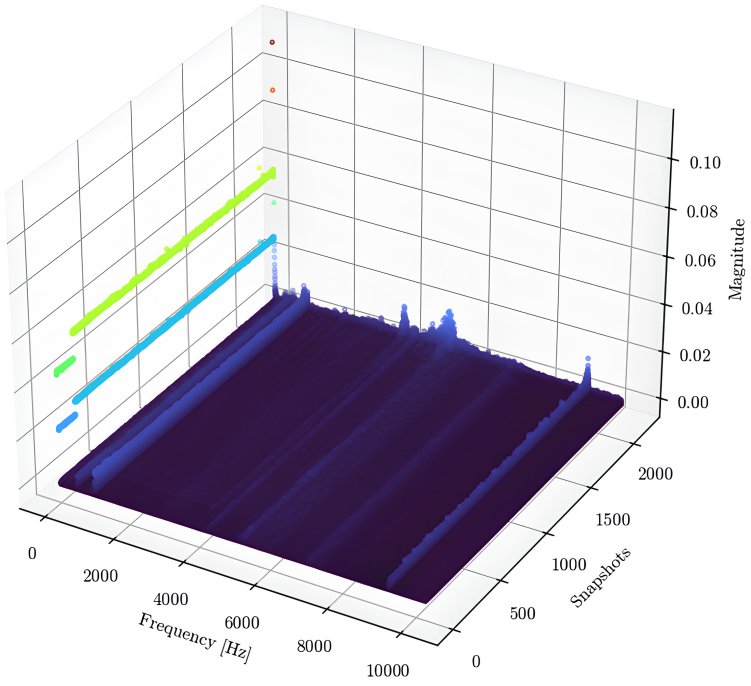


Figure 5.11: *FFT of the "Bearing 3 x" vibration signal from the IMS dataset, in normal conditions with Hann window preprocessing applied*

5.3.2 Wavelet Packet Decomposition

Motivation

In the previous **subsection 5.3.1**, it has been shown that the spectrum of the signal contains useful information for performing ND. Remembering that the ultimate scope of this work is to develop a framework that runs in edge computing, it is important to consider the computational cost of the features extraction, and the memory usage. Considering only the FFT of all signal of the dataset shown in **figure 5.11** and the fact that a floating point value uses 64 bit, the memory usage is $64\text{bit} \times 10\text{kFeatures} \times 2156\text{Snapshots} \div 8\text{bit/byte} \approx 172.5\text{Mb}$. This is not a big problem for a desktop computer, but it may be a problem for an IOT device. Moreover, the FFT is a very computationally expensive operation, with a complexity of $\mathcal{O}(n \log n)$, where n is the number of samples.

To overcome this problem it is necessary to find a method that exploits the same information in the frequency domain, but using much less computational power and memory. Other tools for the frequency analysis are the Wavelet Transform, and in particular the Wavelet Packet Decomposition (WPD) has been used in the developed framework. The theory behind this topic is briefly described in **Appendix B**.

Application to the reference dataset

At this point, let's consider a WPD with a dept of 6, that outputs $2^6 = 64$ coefficients. We are going to take the norm of each coefficient as a feature so, in the end, we will have 64 features (from "aaaaaa" that is lead from all the approximation coefficients to "dddddd" that that is lead from all the detail coefficients instead, passing for all the combinations in between). The WPD of the "Bearing 3 x" signal from the IMS dataset is shown in **figure 5.12** as the bearing is new, and in **figure 5.13** as the bearing is degraded. It is possible to see that the WPD is able to capture the presence of the fault frequencies.

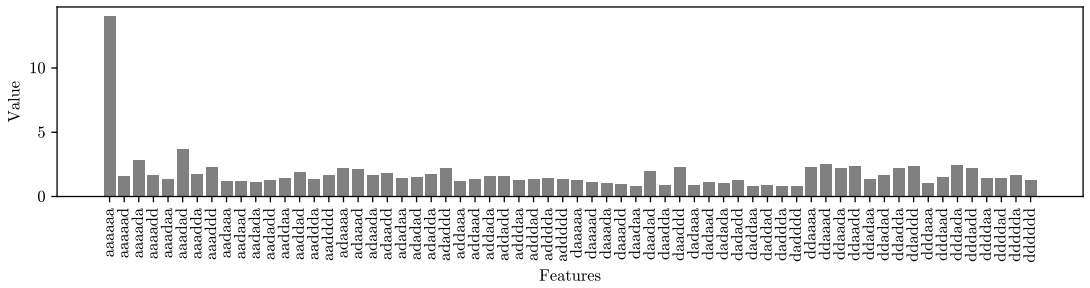


Figure 5.12: *Wavelet Packet Decomposition of the "Bearing 3 x" vibration signal from the IMS dataset, in normal conditions*

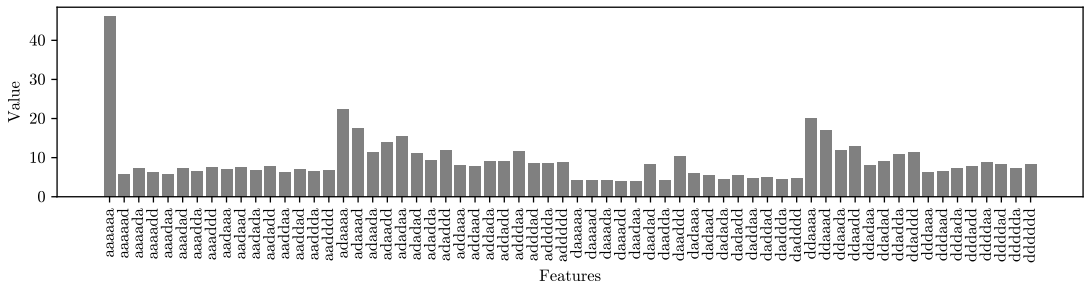


Figure 5.13: *Wavelet Packet Decomposition of the "Bearing 3 x" vibration signal from the IMS dataset, in abnormal conditions*

At this point, it is possible to extract the features with the WPD and plot them over time, for the whole dataset, as it has been done for the FFT in **figure 5.11**. The result is shown in **figure 5.14**. The memory consumption of the data in the second figure is 64bit × 64Features × 2156Snapshots ÷ 8bit/byte ≈ 1.1Mb, that means that the memory saving obtained using WPD instead of FFT exceeds 99%. Despite this huge memory saving, it is possible to see that the WPD is able to retain the information about the fault frequencies, and it is possible to see that the fault frequencies become more and more prominent as the bearing degrades over time. For this reason, the WPD will be used in the developed framework, along with the time-domain features.

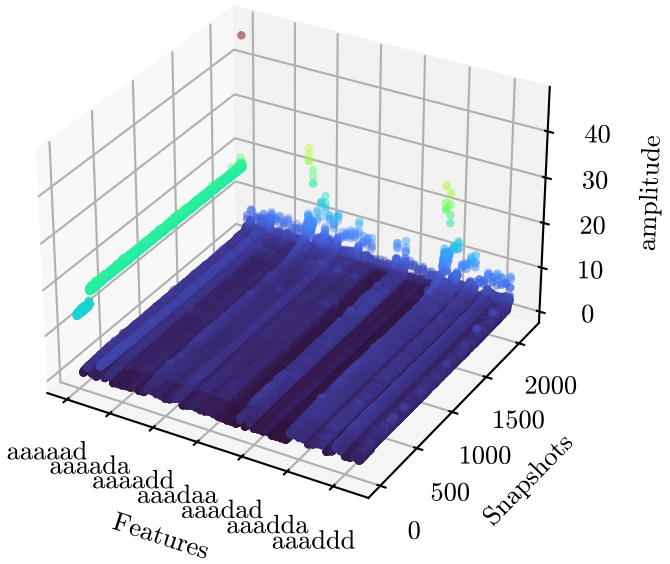


Figure 5.14: *Wavelet Packet Decomposition of the "Bearing 3 x" vibration signal from the IMS dataset, in normal conditions*

5.4 Conclusions

Even just considering very simple time-domain features, we can already detect the degradation of the bearing by comparing these feature values to a threshold. On the other hand, also the frequency-domain features could be just compared with a threshold to detect novel behaviours. But, as we will see in **chapter 6**, a general purpose framework based upon a more sophisticated approach based on the clustering of the data, it will be possible to condense all the features from all the signals into a single metric that will be used to detect the novel behaviour. This will have several advantages. Some of them are being able to detect a novel behaviour even if it is not possible to define a threshold for each feature, and the single threshold to be defined on the metric will be easier to interpret and tune. Moreover, the framework will be able to detect the novelty even earlier and even if the degradation of the bearing is not captured by the time-domain features, but only by the frequency-domain features.

Chapter 6

Proposed Framework

In the **chapter 5** the features extraction process has been described. Before approaching the problem of performing ND in edge computing, let's build a framework in `python` that runs on a PC that is *configurable*, *modular*, *expandable* and *general purpose*. This framework will be used to test the features extraction process and to test the ML algorithms before selecting one of them to be implemented in edge computing framework, which will be harder to configure.

A real case application would probably have several signals of several physical quantities, so a general approach that can manage different types of features, and extract from each of them the most relevant information, is needed.

The proposed framework is thought to be set up on any type and combination of sensors. The framework is thought to manage data that are correlated to a specific fault. For example, think about a CNC machine like the one in **figure 6.1**. It has five axes, so a solution would be to instance the framework five times, one for each axis, linked to vibration sensors, temperature sensors etc. of the considered axis. This would allow us to pinpoint the fault to a specific axis. Another concern is, what if the single axis is seeing a normal condition, but the machine as a whole is not? This may happen if the tool has a problem: the vibration registered in the spindle would be normal in general, but would not be normal *related* to the feed rate that another axis is imposing. To address this scenario, other instances of the framework can be set up, that also receive the speeds from the machine controller and the feed rate from the CNC program as well as data from the sensors used in the other instances. This would allow us to detect a more complex fault, that is not characteristic of any part of the machine, but of the machine as a whole. The former kinds of instances would allow specific faults to be detected, giving also an idea of what the fault is, the latter would allow complex faults to be detected, but would not give a precise idea of what the fault is. The framework is thought to be able to manage both of these scenarios and to be able to manage them together.

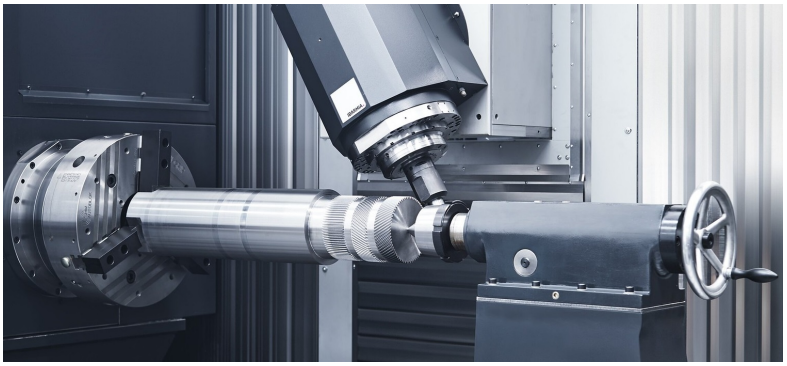


Figure 6.1: *A 5-axis CNC milling machine. [65]*

6.1 Commissioning

To adapt the framework to a specific machine, the commissioning of the ML system would have to be done in steps. Starting from the data acquisition and ending with the predictions of RUL and model updates, the steps are described in this section.

6.1.1 Data structure

The first phase of adaptation of the framework to a machine is to define what data to sample and how to sample them. This includes the decision of which sensors to use, the sampling frequencies, the data acquisition system and which features are needed to be extracted from each sensor data. At this point, if more than one instance of the framework is needed, the sets of sensors and features to be used in each instance are defined. For example, in a shaft with two bearings, each with two accelerometers, the first instance of the framework would be linked to the first bearing, and would use the data from the two accelerometers to extract the features that are needed to detect the fault in the first bearing. The second instance of the framework would be linked to the second bearing, and would use the data from the other two accelerometers to extract the features that are needed to detect the fault in the second bearing. Optionally, a third instance of the framework would use the data from all four accelerometers to detect a generic fault in the shaft. Those decisions influence the structure of the database, which will be described in **section 6.2**.

6.1.2 Data acquisition

Once the structure of the data is defined, the first phase of the commissioning procedure is to set up the data acquisition. This has to be done when the machine is new or, at least, someone guarantees that the machine is in a healthy condition.

During the previous phase, the number of instances of the framework is defined. Each instance would have its own database. This phase is just a matter of storing the data that will be used to train the models the first time. A software agent, which we call Field Agent (FiA), is responsible for this task. This phase lasts until the database is filled with enough data to train the models.

6.1.3 Training

The second phase of the commissioning procedure is to train the models. Once the healthy data are enough to characterize all the normal conditions of the maintained system, all the recorded data are elaborated by another software agent that we call Feature Agent (FA). This agent extracts all the features from the time-series and stores them in a structured way.

Once all the features are available in the database, another agent called Machine Learning Agent (MLA) is responsible for training the models. All the models considered are UML models. The models are trained on a standardized version of the feature matrix. The standardization is done w.r.t. the time evolution, i.e. all the features used for training have a time evolution with zero mean and unit variance. This is done because most ML algorithms are sensitive to the scale of the features.

All that has been said is valid for a single instance of the framework. If more than one instance is needed, the training phase has to be done for each instance.

6.1.4 Evaluation

At this point, a model that represents the normal condition of the system is available (actually a model for each instance of the framework). The next step is to evaluate the model.

In this phase, the machine continues to perform its normal operations. The Field Agent provides the sampled data, the Feature Agent extracts the features and the Machine Learning Agent evaluates the health of the system. The proposed novelty/fault metric and procedure are specific to the model used, as described in the dedicated chapter about UML models (**subsection 4.1.6** for the k-means, **subsection 4.2.4** for DBSCAN, **subsection 4.3.3** for GMM, **subsection 4.6.2** for ν-SVM, **subsection 4.4.2** and **subsection 4.5.2** for LOF).

Now the ND is up and running. The novelty metric is used to decide if the system is healthy or not. The metric is plotted for the user to see. Note that in a classic ML approach, the dataset is split into a training set and a test set. In this case, the test set is the data that are sampled during the evaluation phase. It's equivalent to saying that the model is trained on the past data and evaluated on the future data, or that the

framework works in testing phase for an undetermined amount of time, until the user decides to update the models. This is equivalent to a test phase because if during this phase the framework outputs too many false positives, the user will decide to update the models. Otherwise, it means that the models are working properly and this phase can last indefinitely.

6.1.5 Model update

Once the metric overshoots the threshold, the MLA warns the user about the novelty detected, and starts to perform PdM predicting the future evolution of the metric and the RUL of the system. Again, this condition can last indefinitely, once new data are sampled, the MLA evaluates the health of the system and updates the predictions.

It's up to the user to determine if the warning is a false positive, a novel but healthy behaviour, or a real fault. In the first two cases, the user can decide to command the MLA to update the models. The snapshots that generated the warning are incorporated into the training set, and the models are retrained, returning to the evaluation phase. In the latter case, the user can simply perform the repairs/maintenance needed and restore the system to a healthy condition or can use the snapshots that generated the warning to train a new model that represents the fault condition.

If the user decides to train also the second model with the snapshot declared faulty, the system returns into the evaluation phase, but now the MLA outputs two metrics: one estimates the health of the system and the other how similar the behaviour of the system is compared to any known fault condition.

6.1.6 Predictions

The metric generated by the MLA is useful to detect novelties, with a CBM approach. To actually perform PdM, the MLA has also to predict the future evolution of the metric. Since, as anticipated in the introduction, this framework aims to output *degradation based* predictions, a suitable fitting curve has to be used. Degradation-based failures are led by an initial defect that worsens over time. Often, the presence of an early defect further increases the worsening rate of the system. For this reason, and by observing the features evolution on publically available datasets, it seems reasonable to model the degradation with an exponential curve. This approach has been used in [66] and [67].

The candidate function to fit would then be:

$$f(t) = a \cdot e^{b \cdot t} \tag{6.1}$$

where a and b are the parameters to be estimated. This may work in the case depicted in **figure 6.2a**, in which the novelty metric starts from zero and then increases exponentially. This is a problem for a general case in which the metric can have a plateau, or start as a step that signals the early defect, and then start an exponentially decay, as the example in **figure 6.2b**. In our case, it **chapter 4**, the set of defined metrics to be used are usually negative, to depict a normal behaviour, and not zero.

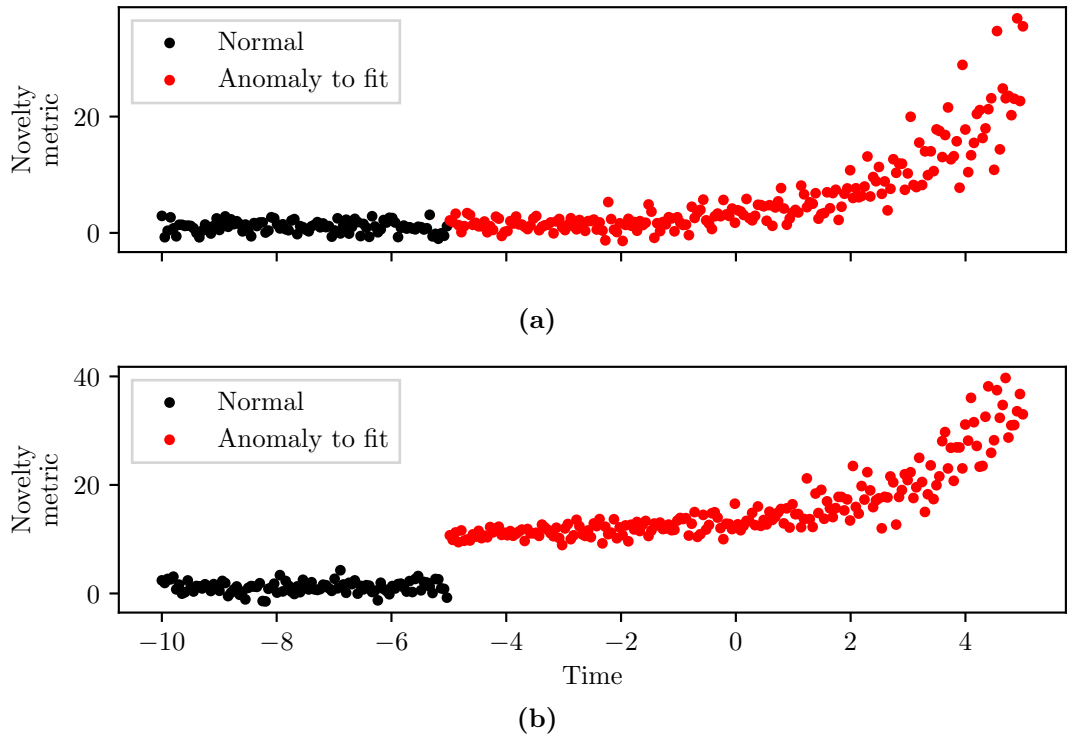


Figure 6.2: *Novelty metric data to fit with an exponential curve.*

The **equation 6.1** is nice, because by extracting the logarithm from both sides (and calling $y = \log(f(t))$) we obtain a linear equation that can be fitted easily with the least squares method described in **subsection 3.1.1**, but for the reasons explained above, it's not suitable for our case. A better candidate is:

$$f(t) = a \cdot e^{b \cdot t} + c \tag{6.2}$$

This is a better candidate because it can depict the case in which the metric starts from a value different from zero, and then increases exponentially.

`Scipy` fit

Unfortunately, **equation 6.2** cannot be arranged in linear form, so the least squares solution has to be found with some other procedure. The `python` library `scipy.optimize` has a function called `curve_fit` that can be used to fit a generic function to a dataset. The problem with this recursive solution is that is very sensitive to outliers and can't really estimate the parameter c correctly, as is shown in **figure 6.3**.

Closed form fit with least squares

Fortunately, in [68], the author provides a *nonrecursive* solution to the problem that minimizes the LS error w.r.t. the parameters a, b and c. The solution has been converted into the following **algorithm 5** and implemented in the framework. The fitting with this closed-form solution is shown in **figure 6.3**. This optimal solution is used as default in the framework to predict the future evolution of the metric, the `scipy` implementation can still be used by changing a parameter in the configuration file because it has the advantage of fitting any function, so it may be useful in particular cases.

Algorithm 5 Exponential regression of the novelty metric.

function ExpRegression(x, y)
  ▷ $x = \{x_1, \ldots, x_n\}$ is the array of x-coordinates of the data points.
  ▷ $y = \{y_1, \ldots, y_n\}$ is the array of y-coordinates of the data points.
  ▷ Returns the parameters a, b and c of the exponential function $y(x) = a \cdot e^{b \cdot x} + c$ that minimize the least squares error.
  $S_1 \leftarrow 0$
  $S_k \leftarrow S_{k-1} + \frac{1}{2}(y_k + y_{k-1})(x_k - x_{k-1}) \quad \text{for} \quad k \in [2, n] \cap \mathbb{N}$
  $\begin{bmatrix} A_1 \\ B_1 \end{bmatrix} \leftarrow \begin{bmatrix} \Sigma(x_k - x_1)^2 & \Sigma(x_k - x_1) \cdot S_k \\ \Sigma(x_k - x_1) \cdot S_k & \Sigma S_k^2 \end{bmatrix}^{-1} \begin{bmatrix} \Sigma(y_k - y_1)(x_k - x_1) \\ \Sigma(y_k - y_1) \cdot S_k \end{bmatrix}$
  $a_1 \leftarrow -\frac{A_1}{B_1}$
  $c_1 \leftarrow B_1$
  $c_2 \leftarrow c_1$
  $\theta_k \leftarrow e^{c_2 \cdot x_k} \quad \forall k \in [1, n] \cap \mathbb{N}$
  $\begin{bmatrix} a_2 \\ b_2 \end{bmatrix} \leftarrow \begin{bmatrix} n & \Sigma\theta_k \\ \Sigma\theta_k & \Sigma\theta_k^2 \end{bmatrix}^{-1} \begin{bmatrix} \Sigma y_k \\ \Sigma y_k \cdot \theta_k \end{bmatrix}$
  $a \leftarrow b_2$
  $b \leftarrow c_2$
  $c \leftarrow a_2$
  return a, b, c
end function

6.1.7 Instance structure

As anticipated, in order to address different kinds of malfunctions, the framework is thought to be instanced several times on the same system. To visualize the structure of the instances, consider the general case described in **figure 6.4**. Any instance can rely on data gathered from any sensor subset and can perform ND and FD. In the figure, the first instance reads the first three sensors and has both the ND and FD algorithms active (this means that a fault dataset has been gathered in the past). The second instance reads a shared sensor with the first one, plus two new sensors. The dashed line surrounding the FD algorithm means that is not active (no fault dataset has been gathered in the past, or it has not been trained). This instance performs only ND. The last instance is just a

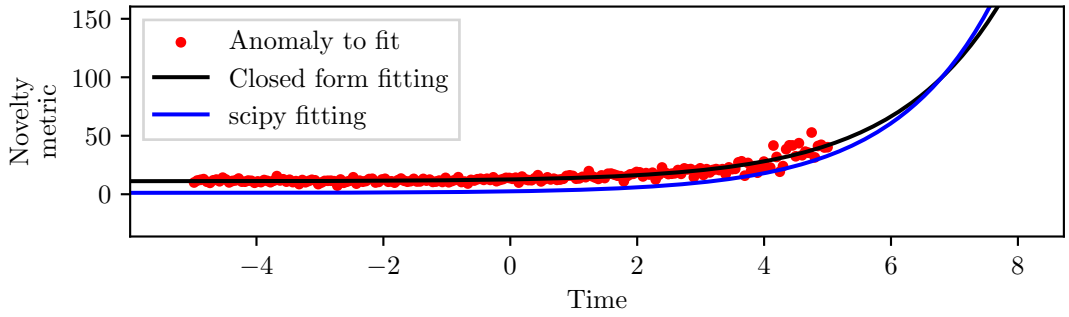


Figure 6.3: *Fitted curve for RUL prediction. The* `scipy` *library fit fails to estimate the parameter c. The closed-form solution actually minimizes the error.*

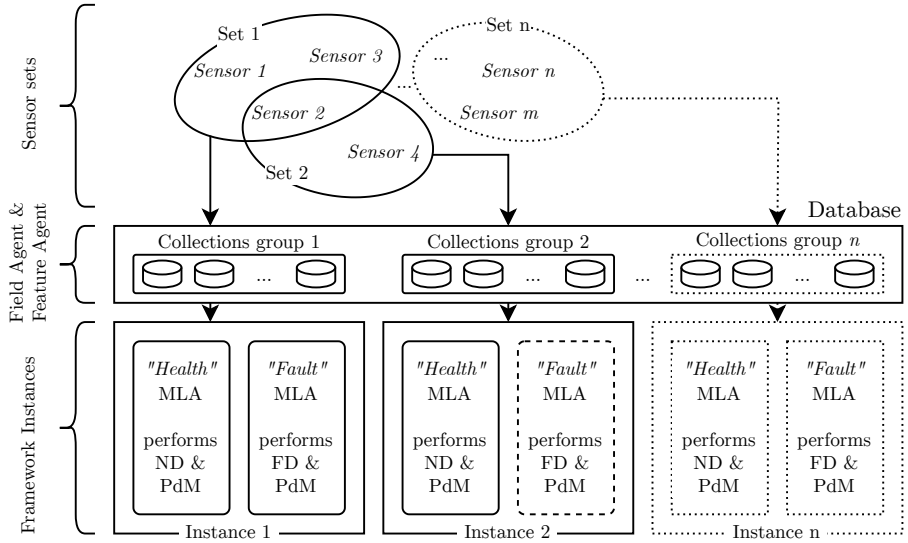


Figure 6.4: *The structure of the instances of the framework.*

general prosecution of the structure, that can be scaled indefinitely.

6.2 Database

In the previous **section 6.1**, the setup behaviour phases of the framework have been described, referring to a generic "database", without specifying the structure of the database. Let's now address the problem of storing the data efficiently and effectively. Instead of relying on `python` data structures, it is better to use a dedicated database manager.

The proposed framework uses MongoDB for the following reasons. It is a widely-used, open-source NoSQL database that is designed to handle unstructured or semi-structured data. It utilizes a document-oriented data model, storing data in flexible, JSON-like BSON

format. MongoDB is suitable for implementation in a ND framework due to its scalability, flexibility, and real-time data processing capabilities. In novelty detection, the system often deals with diverse and dynamic data sources, making MongoDB's "unstructureness" advantageous for handling varying data formats and evolving data requirements. It has the ability to handle large volumes of data and support scaling allowing for efficient storage and retrieval of information in real-time, crucial for real-time applications. Moreover, MongoDB has a rich query language and secondary indexes that allow for fast and efficient querying of data and a library for `python` that makes it easy to use. The JSON format is also human-readable, which makes it easy to understand the data stored in the database, and "mongoDB Compass" is a graphical user interface that allows one to easily explore the database.

6.2.1 Collections

Table 6.1: Collections contained in the MongoDB database

Collection	**Content**
raw	time-series and information about them
unconsumed	snapshots to be evaluated
quarantined	snapshots detected as novelty waiting to be declared healthy, faulty or be discarded
healthy	snapshots declared as normal behavior
healthy train	training dataset (scaled, processed, packet) for the ND UML model
faulty	snapshots declared as faulty behavior
faulty train	training dataset (scaled, processed, packet) for the ND UML model
models	models trained on healthy and faulty data the metrics and predictions to be shown
backup	time-series, features, models, etc.

MongoDB structure is based on collections, that are groups of (JSON) documents. A document is a set of key-value pairs that can be nested in several layers. Documents have a dynamic schema, which means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data. To store the data needed by the framework the collections reported in **table 6.1** are used. In the following paragraphs, the structure and purposes of each collection are described.

Raw Thinking about the data flow, the first interface between the hardware and the software would be the sensor readings. Every sensor should have a name and be sampled at a constant frequency (or, at least, the sensors that provide data for frequency-domain feature extraction should have a constant sampling frequency). This data is stored in the raw collection, with the JSON structure summarized in **??**.

Table 6.2: *Structure of the "raw" collection JSON configuration file.*

Field	Sub-Field	Type
_id		string
timestamp		ISO date
sensor 1	sampling frequency	float
	time-serie	list[float]
sensor 2	sampling frequency	float
	time-serie	list[float]
...	...	...
sensor n	sampling frequency	float
	time-serie	list[float]

where `_id` is the unique identifier of the document, `timestamp` is the time at which the data was acquired, in ISO format, and `Sensor 1` to `Sensor n` are the names of the sensors. Each sensor has a `sampFreq` field that contains the sampling frequency of that particular sensor, and a `timeSerie` field that contains the data acquired by the sensor, as a list. The `timeSerie` field is a list of floating point numbers, that can be of any length. Note that the sampling frequencies of different sensors can be different, for example, if a timestamp contains 1s period of data, a vibration sensor would be linked to an array with several thousands of samples, while a temperature sensor would be linked to only one sample.

Unconsumed Once defined the structure that the time-series will have in the database, let's define the structure of the snapshots. The features extracted from the time-series are stored in the unconsumed collection, with the JSON structure described in **??**.

Table 6.3: *Structure of the "unconsumed" collection JSON configuration file.*

Field	Sub-Field	Type
_id	-	string
timestamp	-	ISO date
sensor 1	mean	float
	root mean square	float
	peak to peak	float
	standard deviation	float
	skewness	float

	kurtosis	float
	wavelet coefficient 1	float
	wavelet coefficient 2	float
	⋮	⋮
	wavelet coefficient $2^{\text{three dept}}$	float
sensor 2	mean	float
	root mean square	float
	peak to peak	float
	standard deviation	float
	skewness	float
	kurtosis	float
	wavelet coefficient 1	float
	wavelet coefficient 2	float
	⋮	⋮
	wavelet coefficient $2^{\text{three dept}}$	float
⋮	⋮	⋮
sensor n	mean	float
	root mean square	float
	⋮	⋮
	wavelet coefficient $2^{\text{three dept}}$	float
novelty evaluated flag	-	boolean

Notice that different sensors can have different features. The "novelty evaluated" field is a boolean that is set to `false` when the snapshot is created, and is set to `true` when the ND algorithm evaluates the snapshot. This field is used to avoid evaluating the same snapshot multiple times while leaving it in the collection until also the FD algorithm is performed. At this point, the snapshot will be moved either to the backup collection, discarded or to the quarantine collection if either the ND or the FD flag it.

Quarantined The "quarantined" collection is used to store the snapshots that were flagged as "novelty" by the ND algorithm or as "faulty" by the FD algorithm (or were flagged by both of them). The structure is the same as the "unconsumed" collection, but the "novelty evaluated" field is not present since, at this point, the snapshots are guaranteed to have been evaluated. The snapshots in this collection are waiting to be declared as "healthy" or "faulty" by the user or to be discarded.

Healthy The idea behind the "healthy" collection is to store the snapshots that are acquired during the first work phase of the framework, before training, or the snapshots that were in the "quarantine" collection and were declared as healthy by the user. The documents in this collection have the same structure as the documents in the "quarantined" collection.

Healthy train In this collection the healthy snapshots are packed together in different documents, each of them useful in a different phase of the training process.

The first document hase the `id` `training_set`, that contains all the `N` training snapshots, each of them with `n` sensors signals, characterized by `F` features. For ease of accessibility, every bottom-nested field is a list of `N` elements. The structure is resumed in **??**.

Table 6.4: *Structure of the "healthy train" collection JSON configuration file.*

Field	Sub-Field	Type
_id	-	string
timestamp	-	list[ISO date]
sensor 1	feature 1	list[float]
	feature 2	list[float]
	⋮	⋮
	feature F	list[float]
sensor 2	feature 1	list[float]
	feature 2	list[float]
	⋮	⋮
	feature F	llist[loat]
⋮	⋮	⋮
sensor n	feature 1	list[float]
	feature 2	llist[loat]
	⋮	⋮
	feature F	list[float]

This collection contains other three documents:

- `training set scaled`, that contains the scaled training set, having the same structure as the `training set` document;

- `training set MIN MAX`, that contains the minimum and maximum values of the features of the training set, useful to plot the features with a reference of the bounds of the training set. It has the same structure of the `training set` document, but the bottom-nested fields are lists of two elements (the minimum and the maximum value);

- `StandardScaler_pickled`. It contains the `StandardScaler` object that was used to scale the training set. This object is encoded in Pickle format, and it is used during the evaluation phase to scale the snapshots before evaluating them.

Faulty This collection serves the same exact purpose as the "healthy" collection, but for the faulty snapshots. Faulty snapshots are not discarded because they can be used to train the FD UML algorithm.

Faulty train This collection serves the same exact purpose as the "healthy train" collection, but for the faulty snapshots.

Models This collection contains the models trained on the healthy and faulty data and a buffer of the predictions and metrics to be displayed to the user.

The structure of the models documents is just an identifier and the `python` object of the model, encoded in Pickle format. The structure of the predictions and metrics documents is the **??**:

Table 6.5: *Structure of the "models" collection JSON configuration file.*

Field	Sub-Field	Type
_id	-	string
timestamp	-	list[ISO date]
values	-	list[float]
assigned cluster	-	list[int]
anomaly flag	-	list[bool]
prediction curve parameters	-	pickle format

Backup The backup collection is a general-purpose container for any document that needs to be stored for backup purposes. It can contain time-series, features, models, etc. The structure of the documents in this collection is the same as the structure of the documents in the other collections.

6.3 Software Agents

In the previous sections, the software agents were mentioned as the main actors in the framework. This section will provide a more detailed description of the software agents, their role and their interaction with the environment and the data, following the flow from the hardware through the time-series, the feature domain to the ML algorithms. The reference layout is the one in **figure 6.5**.

Software agents are autonomous programs that perform a specific task in a cycle. In the proposed `python` implementation, the agents are classes that are instanced and run in a loop.

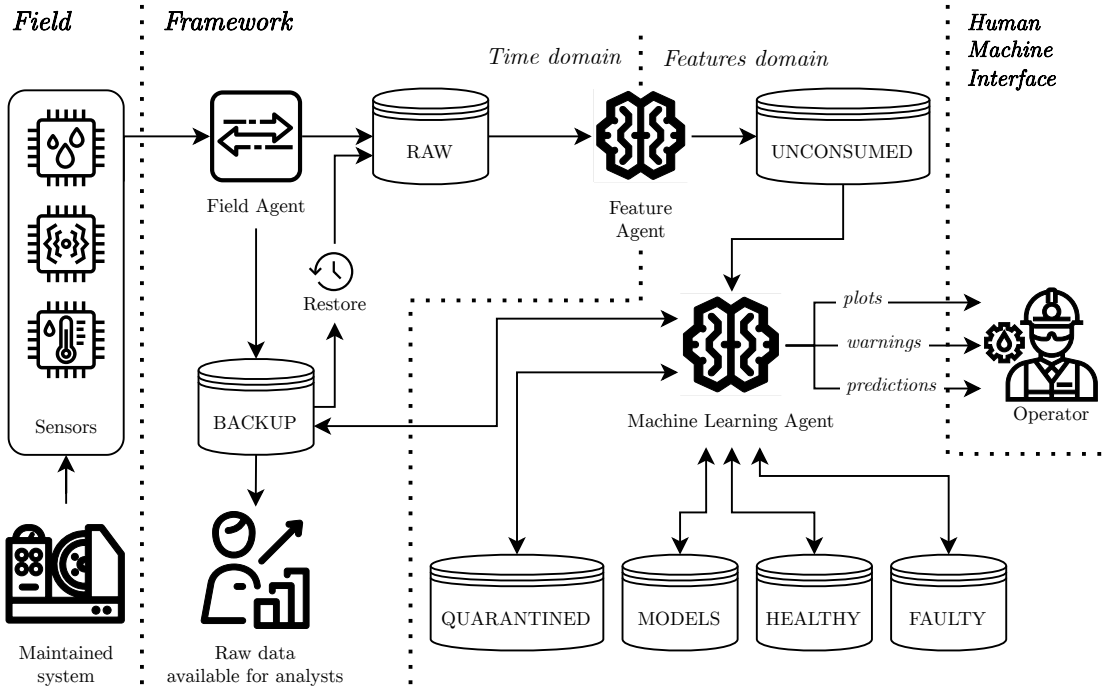


Figure 6.5: *Framework logical structure*

6.3.1 Field Agent

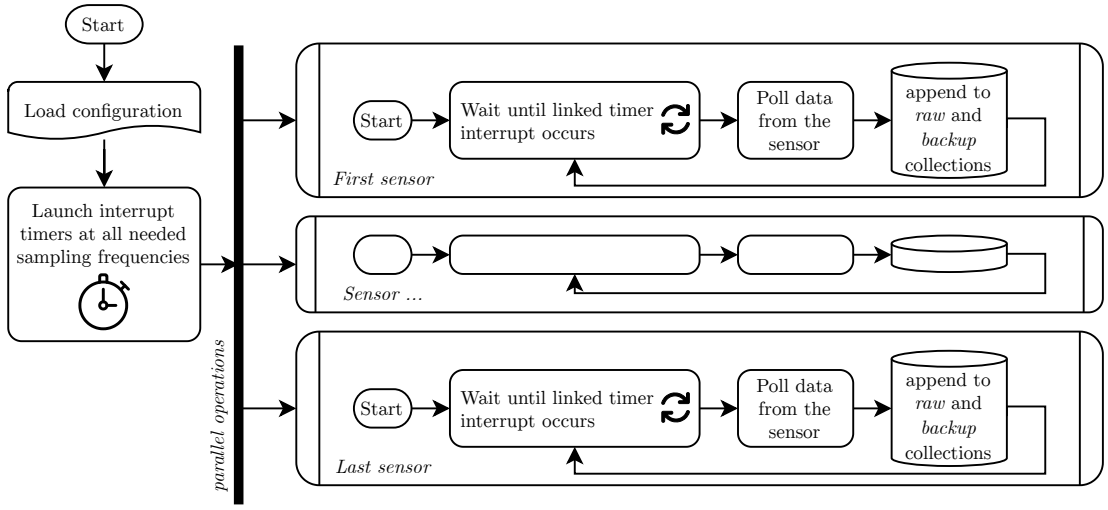


Figure 6.6: *Field Agent flowchart*

The Field Agent is the interface between the hardware and the software. It is responsible

for the acquisition of the data from the sensors and the communication with the database. Since some features are related to the spectrum of the data, a precise and fixed sampling frequency is needed. Hence, the FiA must incorporate a synchronization with the ADC. It stores data in the *raw* collection and the *backup* collection. In **figure 6.6**, the flow of operations is shown as a flowchart, emphasizing the importance of the synchronization with the ADC. This software agent has not been implemented in `python`, because the experimental validation of this work, as it will be described in **chapter 8**, has been performed directly on the edge computing platform. During the tests on the publicly available datasets, an abstract version of the FiA has been used, that reads the data from the CSV files.

6.3.2 Feature Agent (FA)

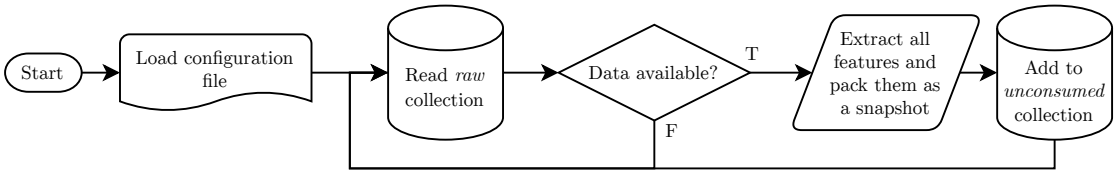


Figure 6.7: *Feature Agent flowchart*

The FA is responsible for the feature extraction from the raw data. It reads the data from the *raw* collection, extracts the features and stores them in the *unconsumed* collection. The flow of operation is shown in **figure 6.7**. The FA is implemented in `python` and it is a class that has been designed to be easily expandable and configurable. The methods implemented in the FA class are shown in **table 6.6**.

Table 6.6: *FA class implemented methods*

Method	**Description**
readFromRaw	reads a snapshot from the raw collection and stores it in the instance self
extractFeatures	extract all the features from the current snapshots, for all the sensors
extractTimeFeautures	extract mean, rms, P2P, std, skewness and kurtosis, based on the config file for the specified sensor
extractFreqFeautures	extract the wavelet coefficients for the specified sensor, up to the configured depth
deleteFromraw	delete current snap record from the *raw* collection
writeToUnconsumed	write the extracted features to the *unconsumed* collection

initialize_barPlotFeatures	initializes the bar plot of the features that is shown to the user
barPlotFeatures	updates the bar plot with new features
run	perform the agent operations in a loop. idle until new data are available in *raw* collection

6.3.3 Machine Learning Agent (MLA)

The MLA is responsible for the training and the evaluation of the ML models. It reads the data from the *unconsumed* collection, evaluates the snapshot and stores the result in the *models* collection, it also constantly updates the information about the novelty or fault metric to the user. The flow of operation is shown in **figure 6.8**. The methods implemented in the MLA class are shown in **table 6.7**.

This agent is designed to be configured as a ND or FD agent with just one hyperparameter. If it is instanced for ND, it uses the healthy collection as a training dataset, if it is instanced for FD it uses the faulty collection. The metric used to evaluate the snapshots is the novelty metric for ND and the fault metric for FD. According to the procedure defined in **algorithm 3**.

Table 6.7: *MLA class implemented methods*

Method	**Description**
run	run the MLA according to its current state
evaluate	evaluate the current snapshot based on the novelty or the fault metric, according to the type of instance
predict	fits the novelty metric with a degradation curve to predict the future evolution
mark_snap_evaluated	set the evaluated flag to true for the current snapshot
delete_evaluated_snap	remove the evaluated snapshots from the *unconsumed* collection
scale_features	scales the features of the current snapshot according to the standard scaler used during the training procedure
evaluate_error	compute the novelty or fault metric for the current snapshot, according to the type of instance
calculate_train_cluster_dist	compute the radiuses of the clusters during the training procedure
prepare_train_data	performs the preprocessing of the data before training the model
pack_train_data	pack the training snapshot in a matrix

___move_to_train	move an entire collection of snapshots to the training collection
standardize_features	make all the features in the training matrix have zero mean and unitary variance
save_features_limits	save the unscaled bounds of the training features
save_StdScaler	store the standard scaler instance in Pickle format into the database
retrieve_StdScaler	restore the standard scaler instance in Pickle format from the database
save_KMeans	store the model instance in Pickle format
retrieve_KMeans	restore the model instance in Pickle format
_append_features	append the current features in a document
train	performs the training of the clustering models
evaluate_silhouette	compute the silhouette score of the training set snapshots
___plot_silhouette	plots the silhouette score against the number of clusters
evaluate_inertia	compute the inertia score of the training set snapshots
___plot_inertia	plots the inertia score against the number of clusters
packFeaturesMatrix	format all the training features as a matrix
retrain	perform a new training of the models

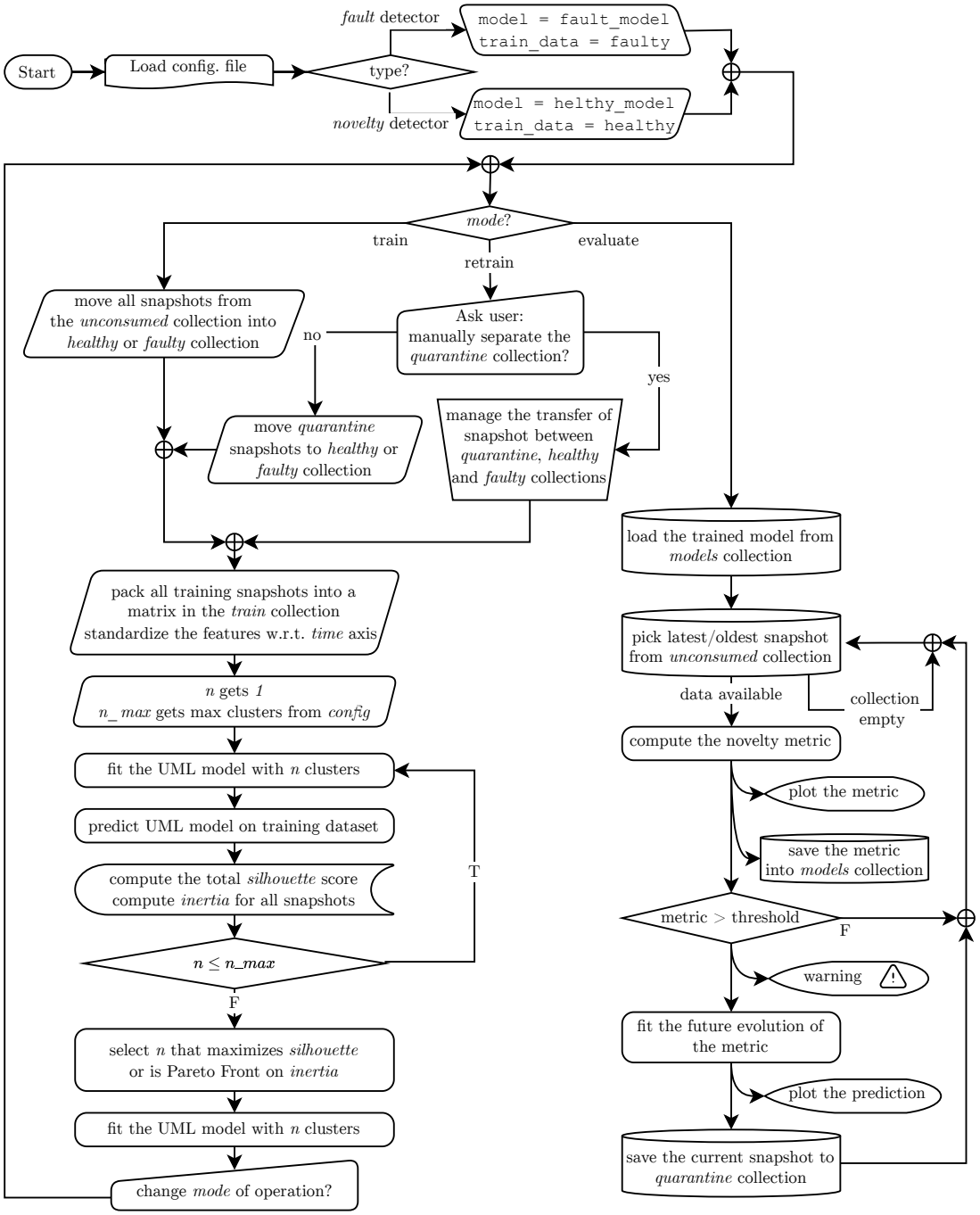


Figure 6.8: *Machine Learning Agent flowchart. When it is instanced for ND, the MLA uses the healthy collection as a training dataset, when it is instanced for FD it uses the faulty collection.*

6.3.4 Configuration of the framework

All the configurations described in this chapter are stored in the `config.yaml` file. This file is read by the agents at the beginning of their execution. The configuration file is divided into sections by topic: database, models etc. The **table 6.8** shows the structure of the configuration file.

Table 6.8: *Structure of the framework configuration file.*

Field	Type	Description
Database	structure	Database configuration
.../URI	string	MongoDB URI
.../DB	string	Database name
.../Collection	structure	Collections configuration
.../.../Backup	string	Backup collection name
.../.../Raw	string	Raw data collection name
.../.../Unconsumed	string	Unconsumed data collection name
.../.../Healthy	string	Healthy data collection name
.../.../Healthy_train	string	Healthy data collection packed for training (some healthy data are not used if not novelty)
.../.../Quarantined	string	Quarantined data collection name
.../.../Faulty	string	Faulty data collection name
.../.../Faulty_train	string	Faulty data collection packed for training (some faulty data are not used if not novelty)
.../.../Models	string	Models collection name
Sensors	structure	Sensors configuration
.../Sensor 1/Name	string	Sensor 1 name
.../.../Features	structure	Features configuration of this sensor
.../.../.../Wavelet	bool	Enable or disable wavelet decomposition for the considered sensor
.../.../.../Mean	bool	Enable or disable mean feature for the considered sensor
.../.../.../RMS	bool	Enable or disable RMS feature for the considered sensor
.../.../.../P2P	bool	Enable or disable P2P feature for the considered sensor
.../.../.../Std	bool	Enable or disable Standard deviation feature for the considered sensor
.../.../.../Skew	bool	Enable or disable skewness feature for the considered sensor
.../.../.../Kurt	bool	Enable or disable kurtosis feature for the considered sensor
.../...	...	...
.../Sensor n/Name	string	Sensor n name

.../Sensor_n/Features	structure	Features configuration of this sensor
.../Sensor n/...	...	...
Wavelet	structure	Wavelet Packet Decomposition configuration
.../Type	string	Type of wavelet to be used (ex. db10, Moore, etc.)
.../Mode	string	Mode of decomposition (ex. Symmetric)
.../Level	int	Depth of the decomposition tree (No. of features = 2^{Level})
Model	structure	Models configuration
.../Max clusters	int	Maximum number of clusters to attempt during clustering
.../Max iterations	int	Maximum iterations of the UML algorithm
.../queue	int	Number of RUL predictions to keep in memory
.../Plot size	int	Number of Novelty Metric values to be kept in memory for plotting
Novelty-Fault	structure	Configuration of the novelty or fault detection
.../Threshold	float	Novelty - Fault detection threshold
.../RUL Threshold	float	Novelty - Fault threshold for RUL predictions
.../Fit points	int	Number of samples used for fitting the prediction curve
.../Outlier filter	int	Number of consecutive outliers to filter
.../Regressor	string	Type of curve to fit (exp or scipy to select the closed form solution or the iterative solution)
Log	string	Path where to store the logs of the framework

6.3.5 Command Line Interface (CLI)

To ease the interaction with the user, a CLI has been implemented. It relies on the `click` and `typer` libraries for `python`. The CLI allows the user to instance the agents, to configure the framework, to monitor the agents and to interact with the database. All the commands are provided with a help message that can be accessed by typing `-help` after the command, as shown in **figure 6.9** for the command `run-machine-learning-agent`. The commands implemented in the CLI are shown in **table 6.9**.

Table 6.9: *CLI implemented commands*

Command	Description
copy-collection	Move all the documents from one collection to another
create-empty-db	Create an empty database in MongoDB. The database should not exist already. It is configured according with "config.yaml" file.
ims-converter	Transfer the data from the glsims textual files into the MongoDB database in a suitable way.

fault-indicator	This function plots the fault metric.
novelty-indicator	This function plots the novelty metric.
move-collection	Move all the documents from one collection to another
plot-features	Plot the features of the last snapshot in the UNCONSUMED collection
run-feature-agent	Run the Feature Agent - takes the last snapshot from RAW collection, extracts features and writes them to UNCONSUMED collection
run-machine-learning-agent	Run the Machine Learning Agent

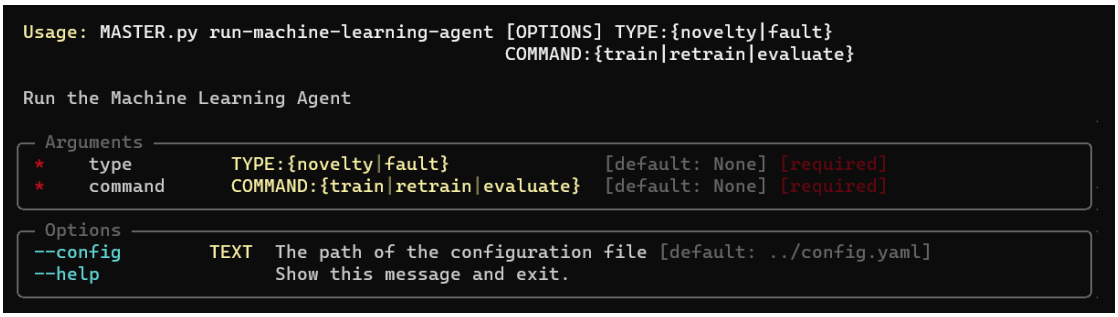


Figure 6.9: *Command Line Interface help message*

Chapter 7

Embedded implementation

In **chapter 6**, an overview of the framework developed in `python` was given, relying on the MongoDB database. This chapter will focus on the implementation of the embedded system. The first big difference is that the embedded system is written in `C`, which is not an object-oriented language. The second big difference is that the embedded system does not use a database, but it relies only on the variables stored in the RAM. Because of the memory constraints, the training phase relies upon the communication with a PC for storing the heavy data. Once the model has been trained, the model is stored in the embedded program and the novelty detection is performed in real time. The general structure is shown in **figure 7.1**.

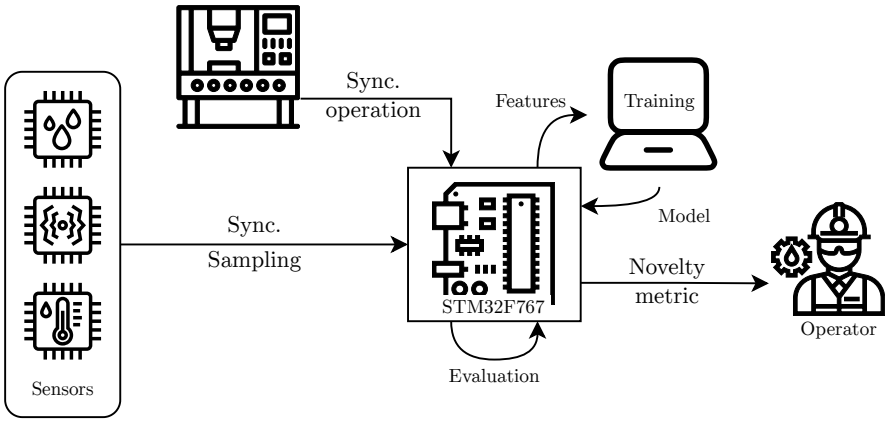


Figure 7.1: *Embedded system overview*

7.1 Hardware

The hardware used for the implementation is the STM32F767ZI board. The characteristics of the board are resumed in **table 7.1**.

Table 7.1: *Hardware characteristics of STM32F767ZI board*

Feature	**Description**
Microcontroller	STM32F767ZI
Architecture	ARM Cortex-M7
Clock Speed	Up to 216 MHz
Flash Memory	2 MB
SRAM	512 KB
EEPROM	No
GPIO	Up to 176
Timers	3 x 12-bit, 12 x 16-bit, 2 x 32-bit
ADC	3 x 12-bit
DAC	3 x 12-bit
Communication Interfaces	USART, UART, SPI, I2C, CAN, Ethernet, USB
Operating Voltage	1.7V - 3.6V
Operating Temperature	$-40\,^{\circ}\mathrm{C}$ to $150\,^{\circ}\mathrm{C}$

Similarly to what has been done for the `python` implementation, the parameters of the algorithm are configurable. To avoid the reading of files during the operation, the configuration is held in global variables defined in a header file. The configurable parameters are the usual: depth of the wavelet three, number of features, sampling frequency, time-series length etc.

7.2 Software

The code consists of a main loop, that is continuously running. It is responsible for executing the state machine behaviour, that manages the different phases of operation. The phases of operation are the same as described for the `python` implementation, except for the training phase, in which the microcontroller performs the sensor polling and the feature extraction and then sends the data to the PC using serial communication. The PC is responsible for the training phase. This part is developed again in `python`, but the final model is then formatted as a `model.h` file that can be directly included in the embedded code. The model is then stored in the flash memory of the microcontroller, together with the rest of the program.

The hardware configuration has been done using the IDE (STM32cubeIDE tool), which is a graphical interface that allows the configuration of the microcontroller and generates the initialization code.

7.2.1 Sensor polling

The microcontroller comes with a Hardware Abstraction Layer (HAL) which acts as an intermediary layer between the hardware and software. It simplifies interaction with the microcontroller's peripherals, such as GPIO, UART, and timers, by providing standardized functions and APIs. The HAL library enhances code reusability across different STM32 microcontroller families, streamlining the development process and enhancing the scalability of embedded systems projects.

To sample the data at a precise sampling frequency, two options are available:

- Use the Direct Memory Acces (DMA) capability of the microcontroller. This approach allows sampling the GPIO and storing the result in the memory accessible by the CPU, without using CPU time. The DMA is then configured to trigger an interrupt at the end of the transfer, and the interrupt is used to signal the end of the sampling and to start the feature extraction. It is suitable for high sampling frequencies and in fact, even downscaling the clock frequency linked to the DMA there is a lower bound of obtainable sampling frequencies.

- Use the Timer peripheral of the microcontroller. The timer is configured to trigger an interrupt at a precise frequency, and the interrupt causes the CPU to poll the sensor data. This approach is suitable for sampling frequencies that are not too high, and it is the one used in this work (for frequency in the order of kHz). If too many interrupts are generated, the CPU may not be able to execute them instantly, so the actual sampling may shift from the desired frequency, however, in this implementation the only interrupt used is the one for the sampling, so the CPU is not overloaded and the sampling frequency is precise.

7.2.2 Feature extraction

The features available to be extracted are the same as the ones described in **chapter 5**. The time-domain features are coded directly in a function that is responsible for extracting them. The frequency-domain features are computed by another function that relies on the `C` library `wavelib` for the wavelet transform [69]. The power of the wavelet coefficients is then computed and appended to the feature vector. The feature vector is then stored in the RAM, and it is used for novelty detection. The features are then standardized using the same mean and standard deviation used for the training phase.

7.2.3 Evaluation

When the microcontroller is in the evaluation phase, the feature vector is processed to compute the novelty metric. The model cluster centroids and radiuses were saved in the code in the training phase, so now it is possible to run the **algorithm 3**.

7.2.4 Custom `C` functions

The `C` main loop, which executes all the behaviours that in the `python` implementation were executed by the various agents, relies on the library functions as well as on the custom functions resumed in **table 7.2**.

Table 7.2: *Custom function implemented in `C`*

Method	**Description**
setRTCclock	Set the clock of the microcontroller to the current time
get_time	Get the current time from the RTC clock
acquireSnapshot	Acquire a snapshot from the sensor
calcSnapDistanceError	Calculate the novelty metric based on the model centroids, radiuses and the current features vector
std_sclr	Standardize the features vector
snapReadyHandler	Handle the snapshot ready event (the interrupt of the Timer)
norm2	Compute the norm of a vector
packetCoeff	Perform the Wavelet packed decomposition and compute the norm of the coefficients, it relies on [69]
featureExtractor	Extract the features from the snapshot (both time domain and frequency domain)
eucDist	Compute the Euclidean distance between two vectors

Chapter 8

Validation

This chapter is dedicated to the validation of the framework on real-world data. In **chapter 5**, the reference dataset [61] 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 behaviour only if two consecutive snapshots are labelled 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.

8.1 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 **chapter 5**. The training dataset is used to train the MLA to recognize the normal behaviour of the bearing, and the testing dataset is used to validate the trained model. The **table 5.1** 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 8.1** in normal and abnormal

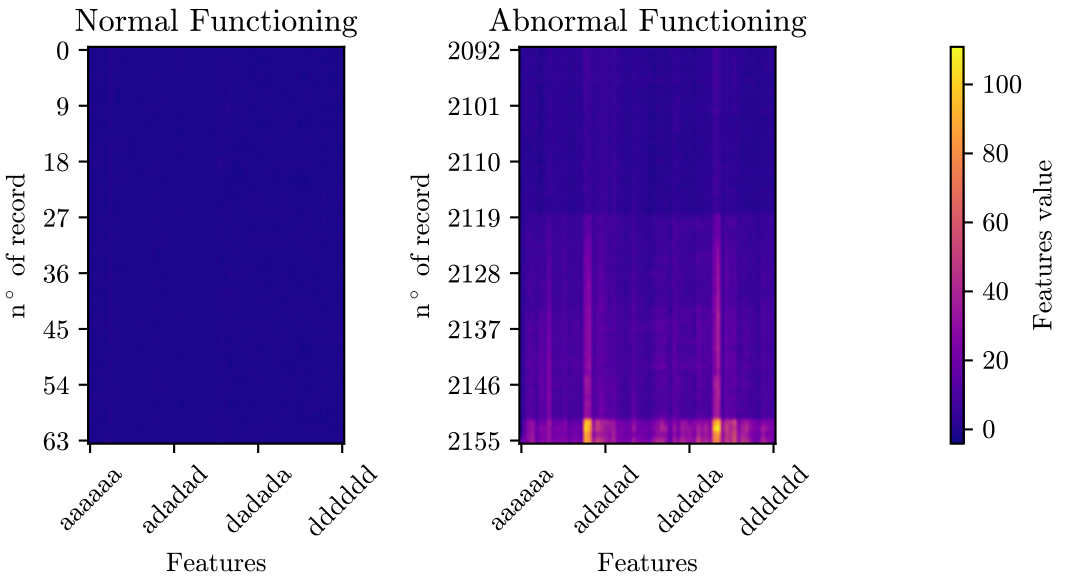


Figure 8.1: *Heatmap of the standardized features value for the test n°1 of IMS dataset*

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.

8.1.1 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 8.2** and **figure 8.3**. 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 **subsection 4.1.10**.

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 8.4**, where the points are the snapshots, the crosses are the centroids and the colours 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 **subsection 4.1.11**.

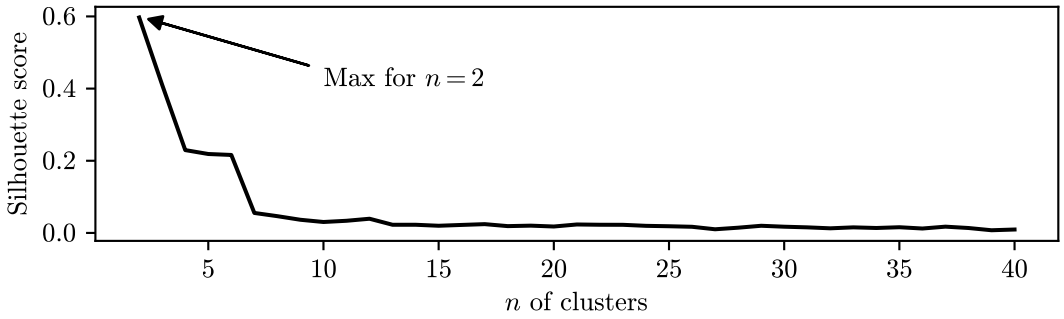


Figure 8.2: *Silhouette score for clustering the test n°1 of IMS dataset (K-means)*

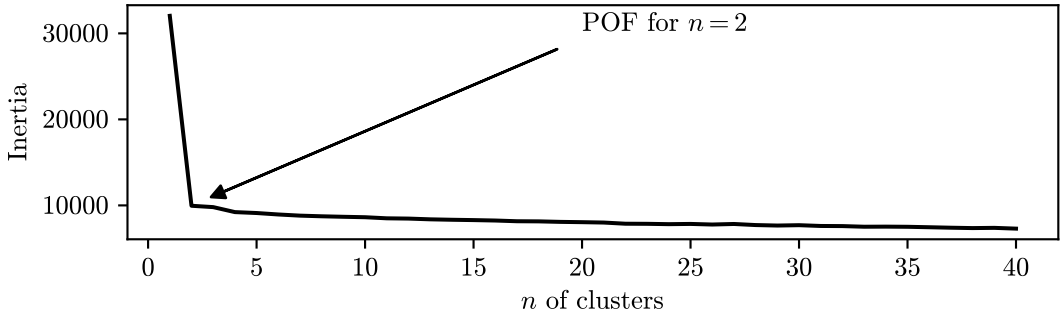


Figure 8.3: *Inertia score for clustering the test n°1 of IMS dataset (K-means)*

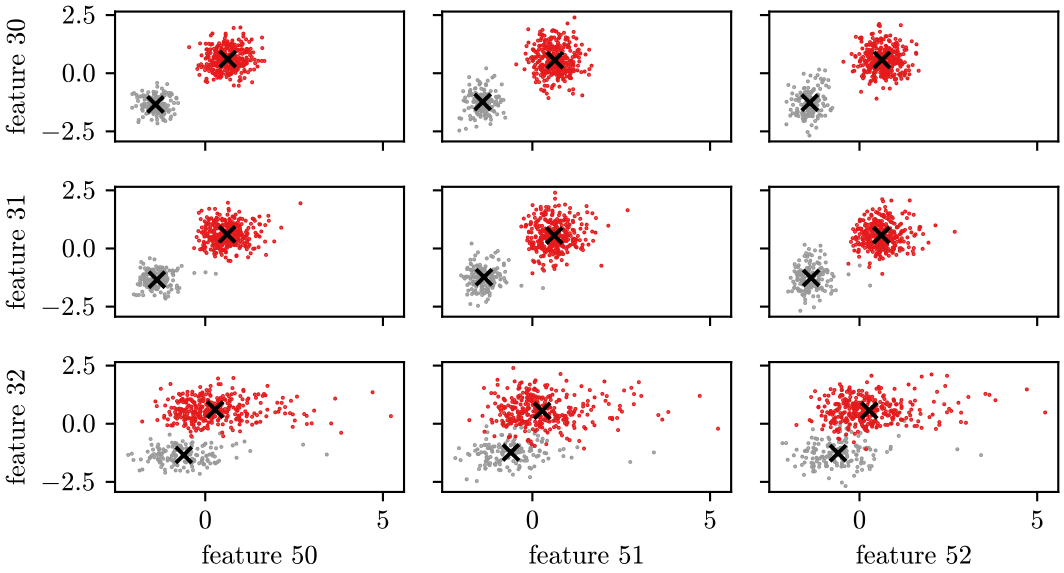


Figure 8.4: *Scatterplot of training snapshot for the test n°1 of IMS dataset*

8.1.2 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 **algorithm 3** 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 8.5**, 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 8.6**, a detailed view of the ND metric becoming consistently greater than the threshold is shown.

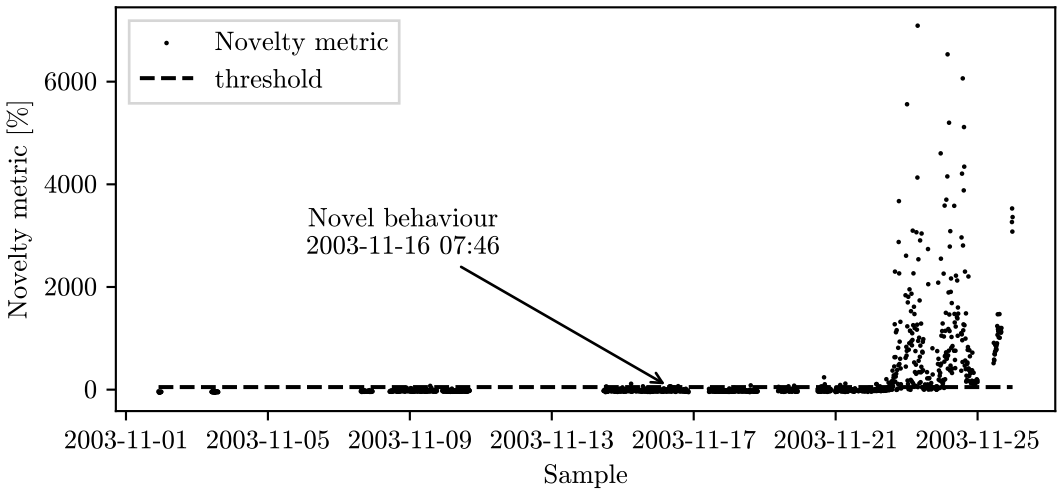


Figure 8.5: *Results of ND for the test n°1 of IMS dataset (K-means)*

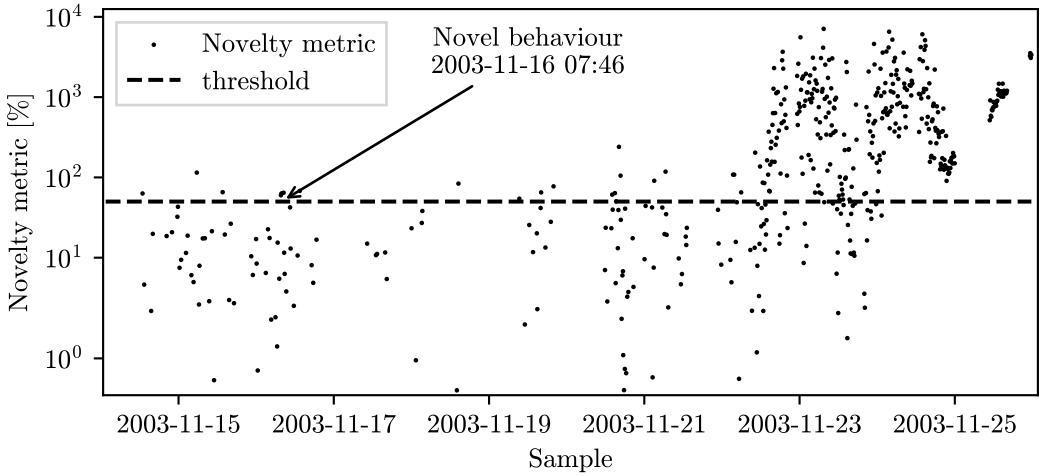


Figure 8.6: *Results of ND for the test n°1 of IMS dataset (K-means) - detailed view*

8.1.3 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 8.7**, the optimal value is 8, which corresponds correctly to the generation of two clusters.

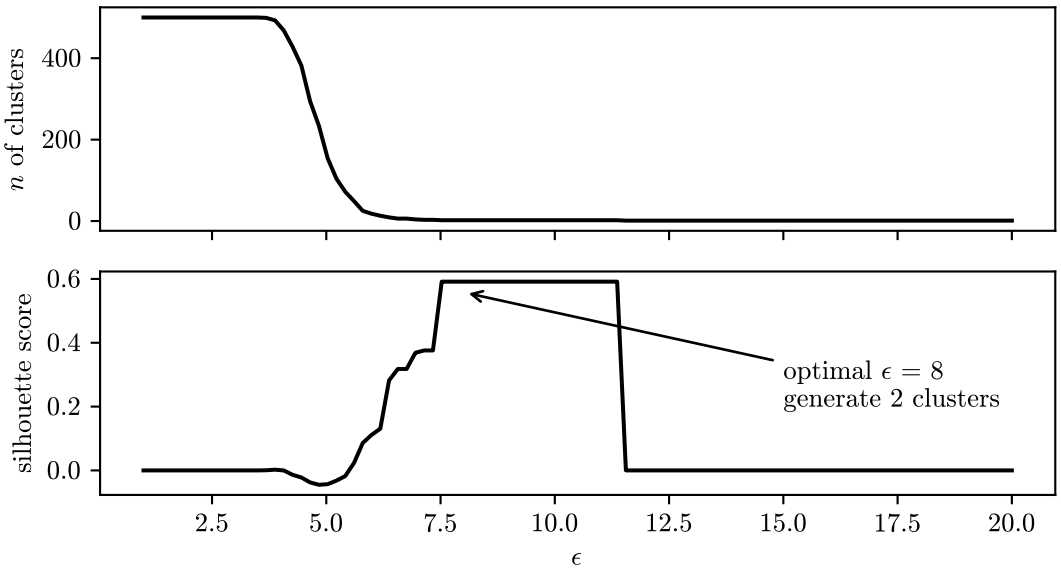


Figure 8.7: *Silhouette score for clustering the test n°1 of IMS dataset (DBSCAN)*

8.1.4 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 **subsection 4.2.4**. The result is shown in **figure 8.8**, 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.

8.1.5 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 8.9**. The two metrics diverge but, as discussed in **subsection 4.3.1**, 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.

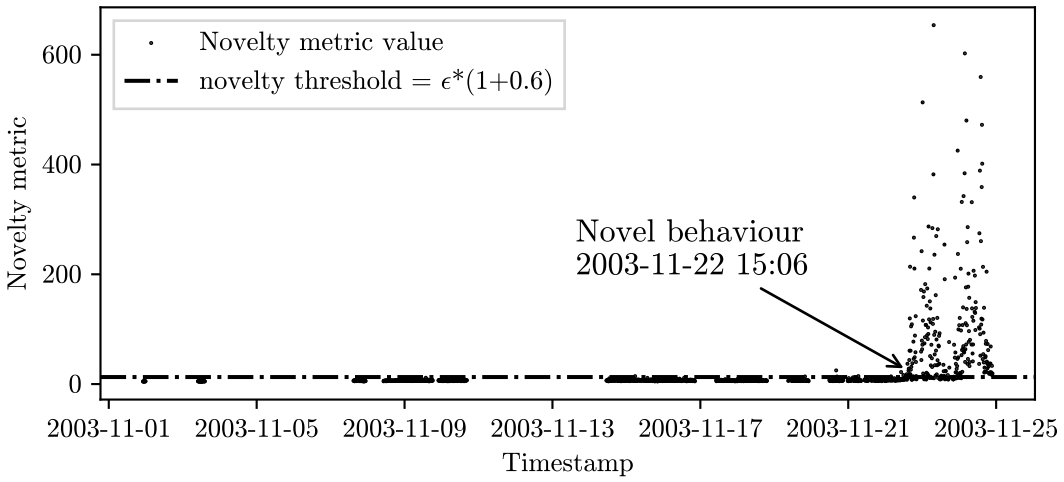


Figure 8.8: *Results of ND for the test n°1 of IMS dataset (DBSCAN)*

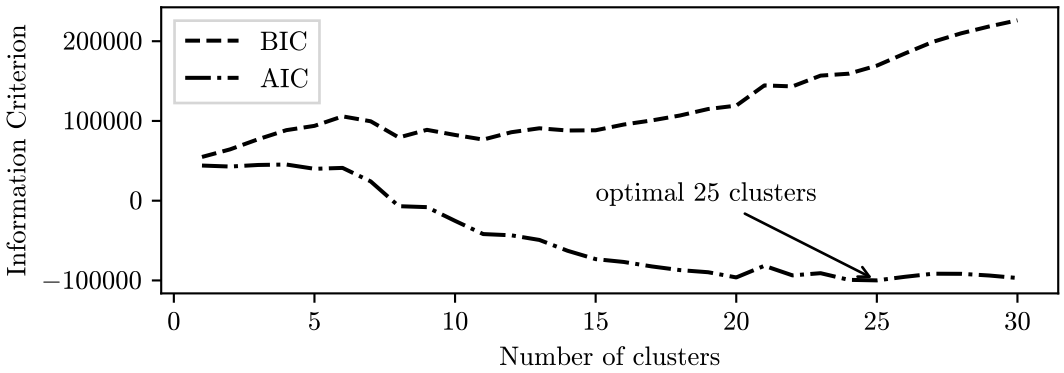


Figure 8.9: *BIC and AIC for clustering the test n°1 of IMS dataset (GMM)*

8.1.6 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 8.10**, 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 8.10**, the metric (density value) appears in coloured dots, as each colour represents the cluster to which the snapshot has been assigned.

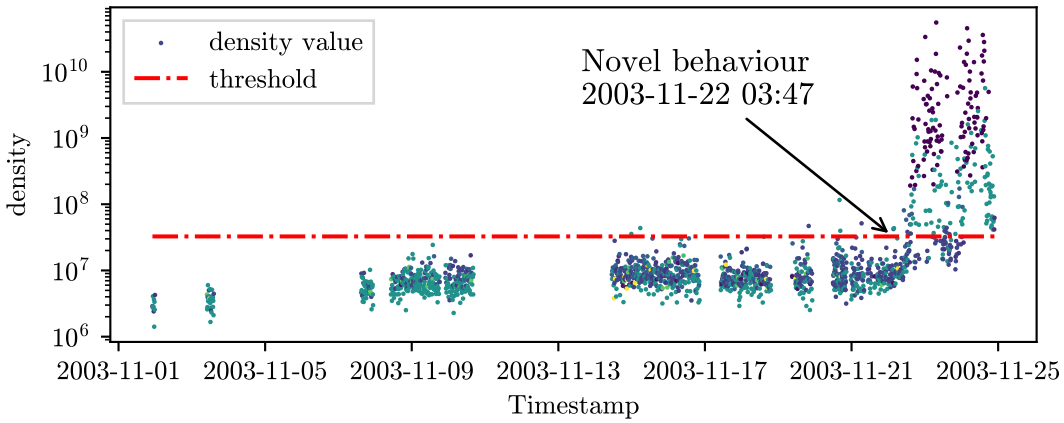


Figure 8.10: *Results of ND for the test n°1 of IMS dataset (GMM)*

8.1.7 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 8.11**, 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.

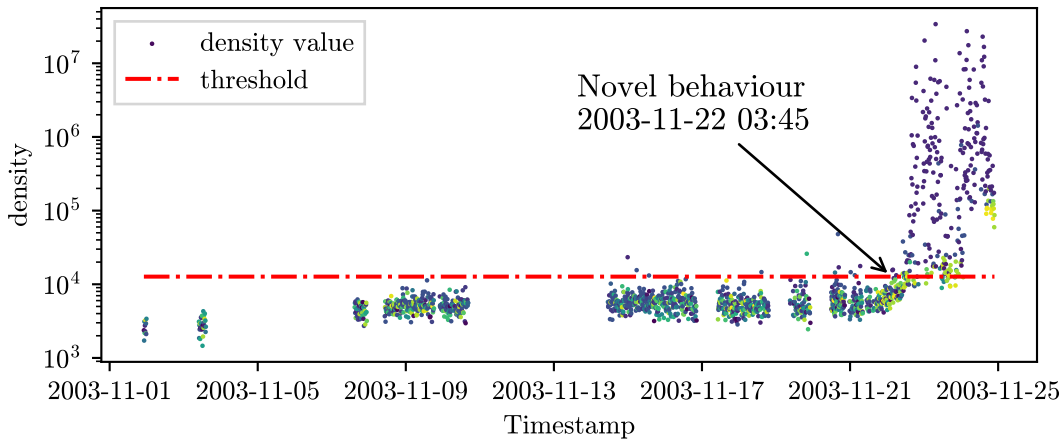


Figure 8.11: *Results of ND for the test n°1 of IMS dataset (BGMM)*

8.1.8 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 8.12**, 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.

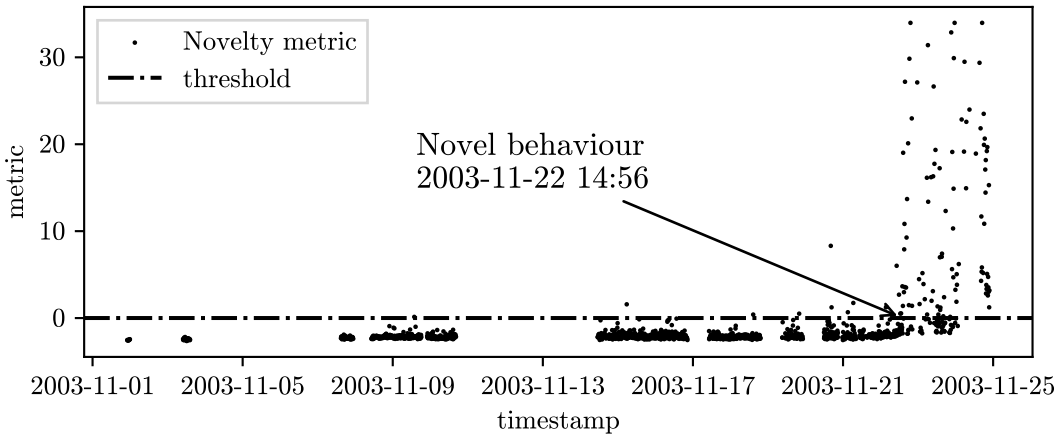


Figure 8.12: *Results of ND for the test n°1 of IMS dataset (ν-SVM)*

8.1.9 ND Validation - iForest

The second last technique to test is the one based on the iForest model. The result is shown in **figure 8.13**, where we can see that the iForest model detects the novelty at 2003-11-16 10:08:46, which 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.

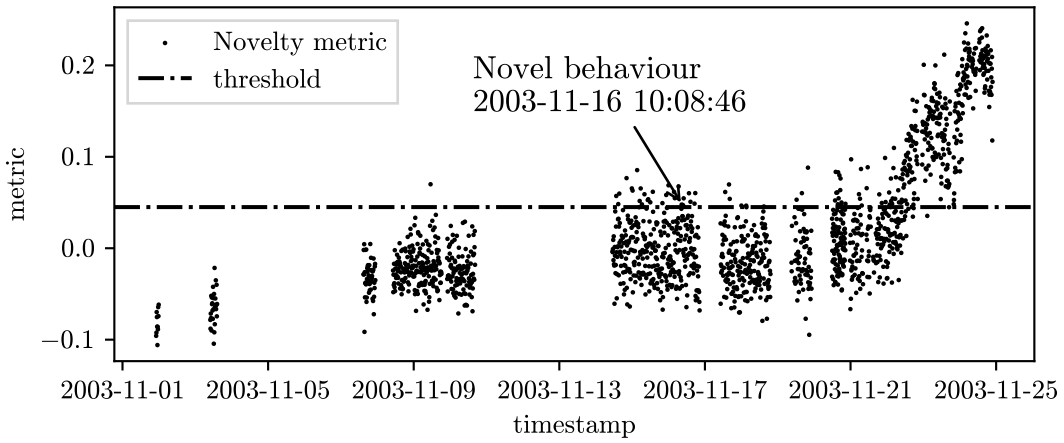


Figure 8.13: *Results of ND for the test n°1 of IMS dataset (iForest)*

8.1.10 ND Validation - LOF

The last algorithm to test is the LOF. The result is shown in **figure 8.14**, 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.

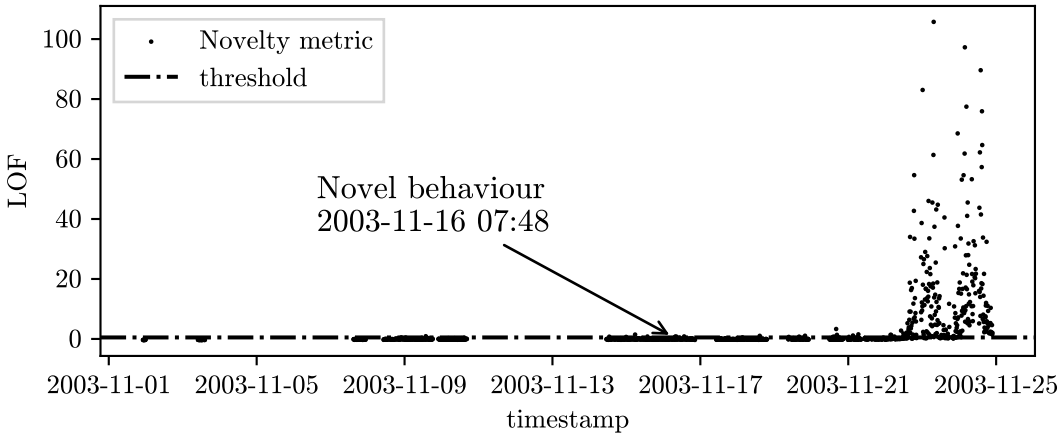


Figure 8.14: *Results of ND for the test n°1 of IMS dataset (LOF)*

8.1.11 Comparison of the results

Comparison between the models

Table 8.1: *Comparison 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 8.1**, 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 **figure 5.3**. 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 **chapter 2**, about the State of the Art, the signal of the same bearing (Bearing 3x) of this same test has been used in [39]. In their research, the authors used a different approach, based on regression, and obtained the result reported in **figure 2.7**

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. A high variance, on this dataset, means that the system is very sensitive while evaluating quite similar signals.

8.1.12 RUL Predictions validation - K-means

After the ND event, the MLA starts predicting the future evolution of the novelty metric, and it superimposes 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 **equation 6.2**, as described in **subsection 6.1.6**. 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 8.15**. 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 8.16**.

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 [15]). If this behaviour 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.

8.1.13 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).

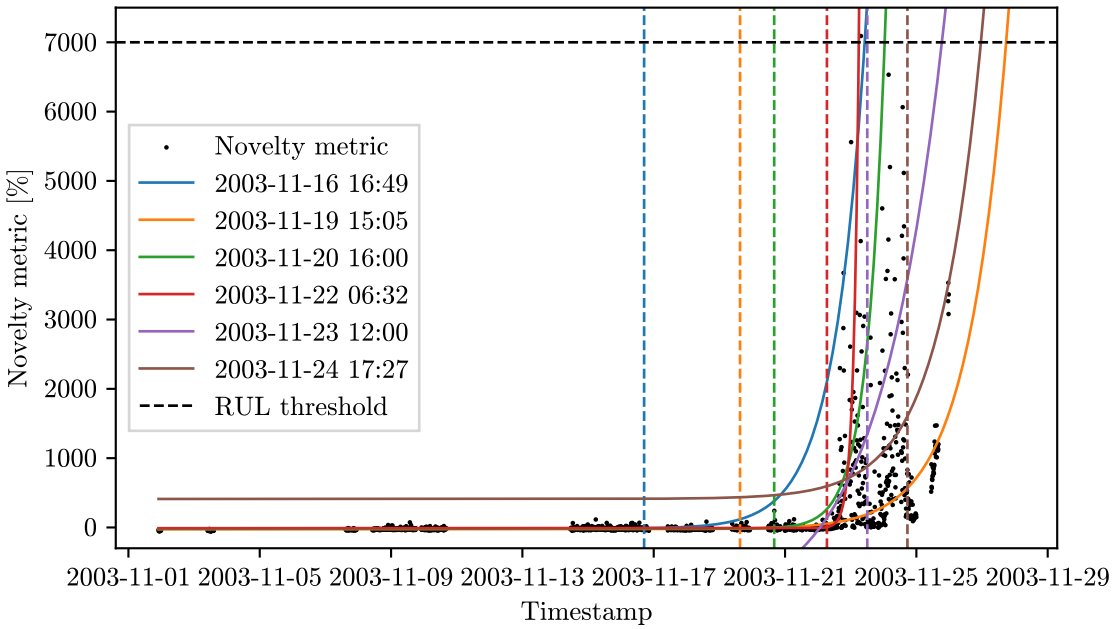


Figure 8.15: *RUL prediction at different instants after the ND event. The dashed lines are the instants of the predictions corresponding to the same colour solid line prediction.*

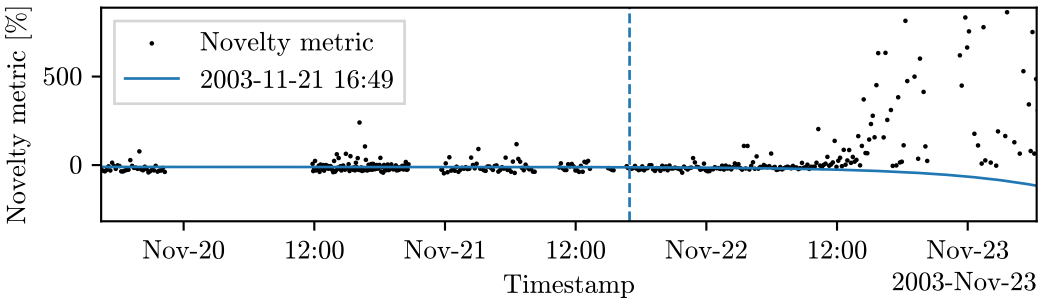


Figure 8.16: *Failed RUL prediction.*

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 8.17**. The predictions of the RUL are shown in **figure 8.18**.

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 8.19**, 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 8.4** 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.

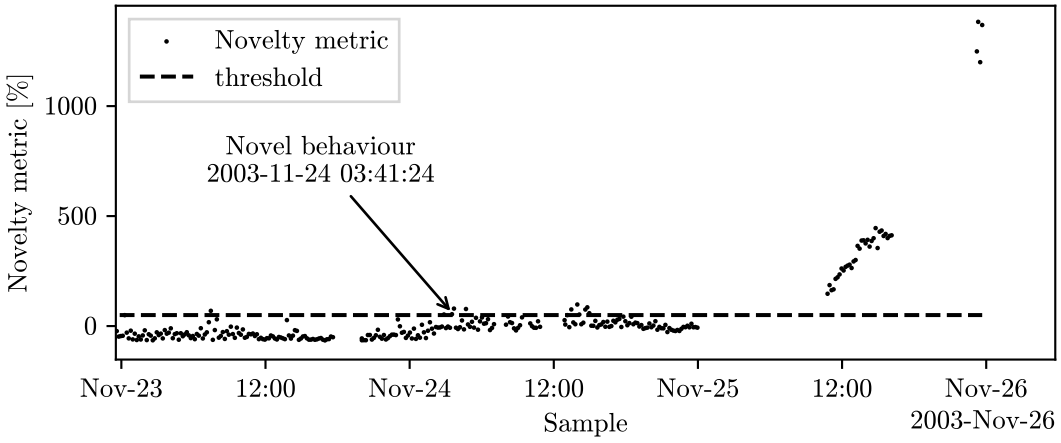


Figure 8.17: *Results of ND for the test n°1 of IMS dataset (K-means) - retrained model*

8.2 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.

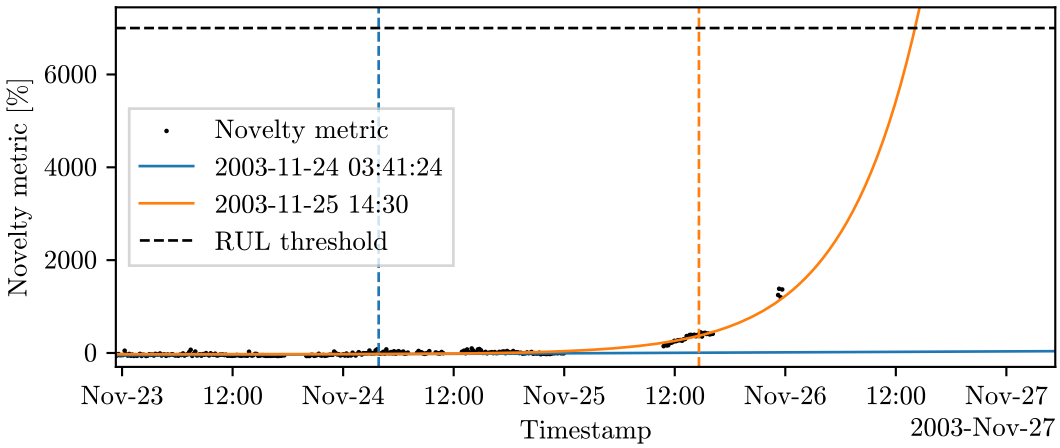


Figure 8.18: *RUL prediction at different instants after the ND event. The dashed lines are the instants of the predictions corresponding to the same colour solid line prediction.*

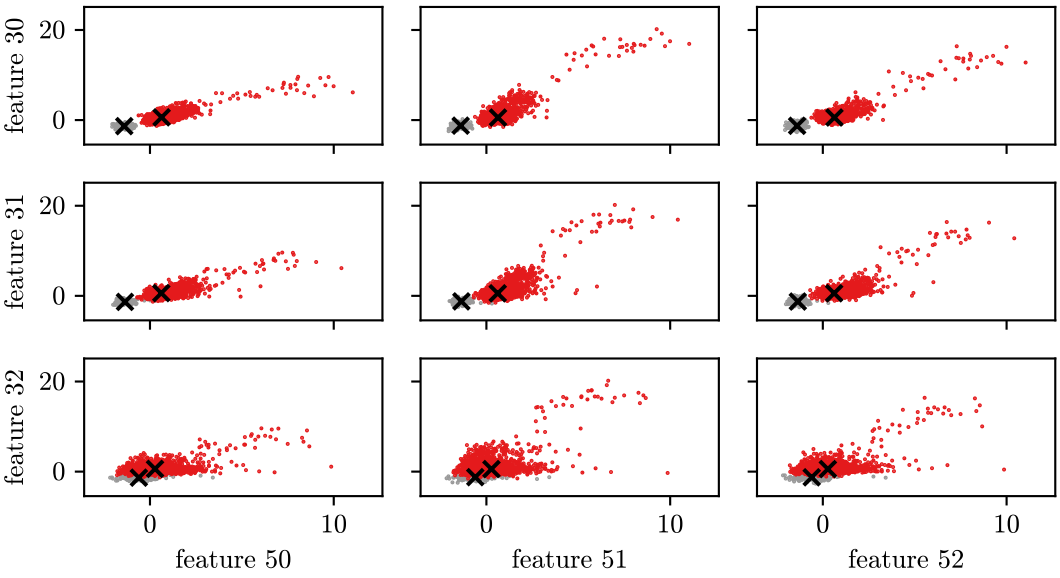


Figure 8.19: *Scatterplot of all the snapshot for the test n°1 of IMS dataset*

8.3 IMS dataset No.2 - Bearing 3x sensor

8.4 IMS dataset No.3 - Bearing 3x sensor

8.5 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 8.2**, 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 8.20**.

Table 8.2: *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

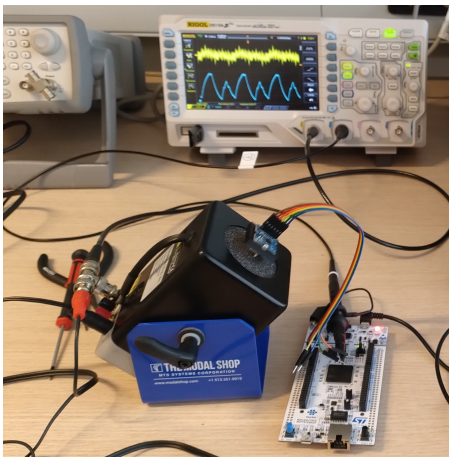


Figure 8.20: *Setup of the shaker tests.*

Table 8.3: *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

8.5.1 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 8.3**. 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 8.21**, 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.

8.5.2 Results

The result of the novelty detection is shown in **figure 8.22**. 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 8.23**.

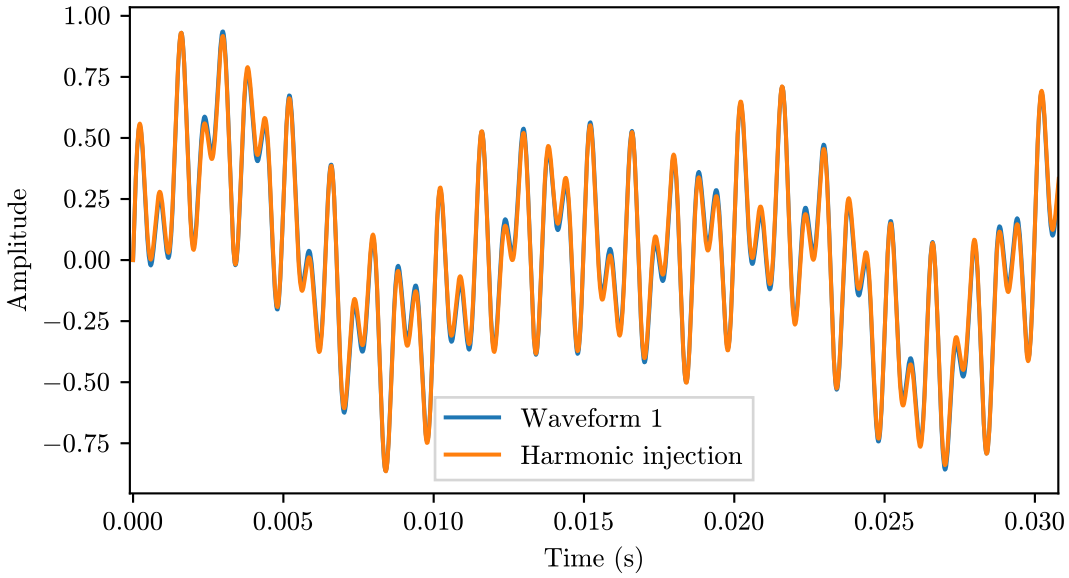


Figure 8.21: *Waveform comparison of the shaker test.*

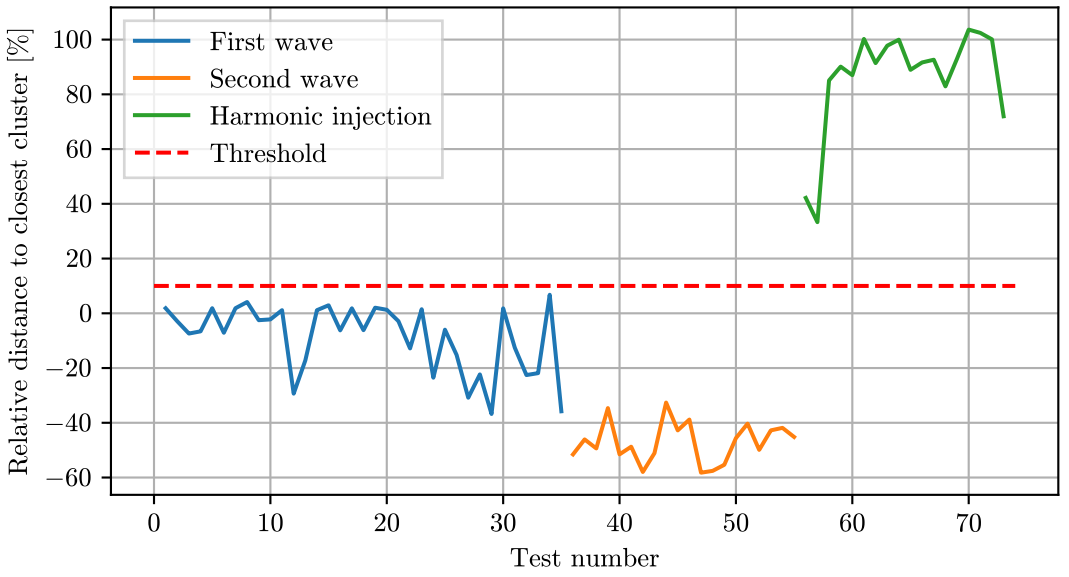


Figure 8.22: *Novelty detection result*

8.6 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

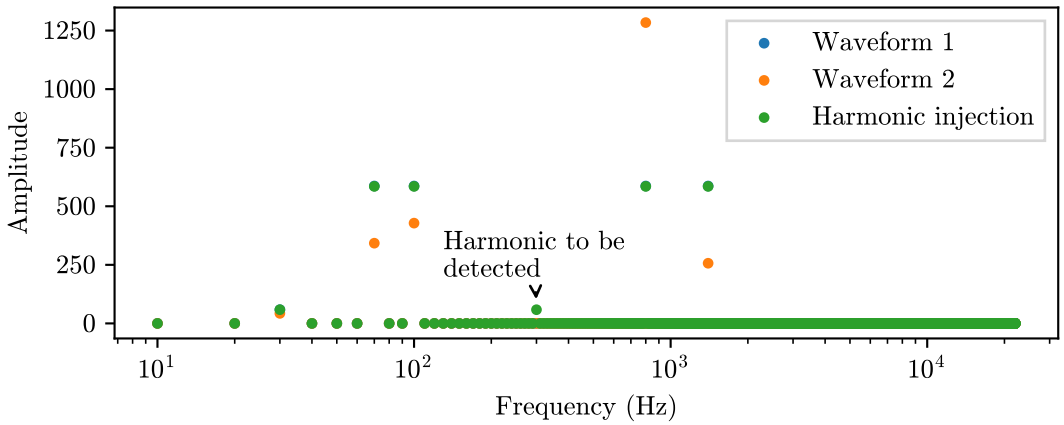


Figure 8.23: *Spectrum of the waveforms.*

to evaluate the capability to detect time-domain variations and the effect of reducing the frequency resolution.

8.6.1 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 8.4**. 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 8.4: *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

8.6.2 Results

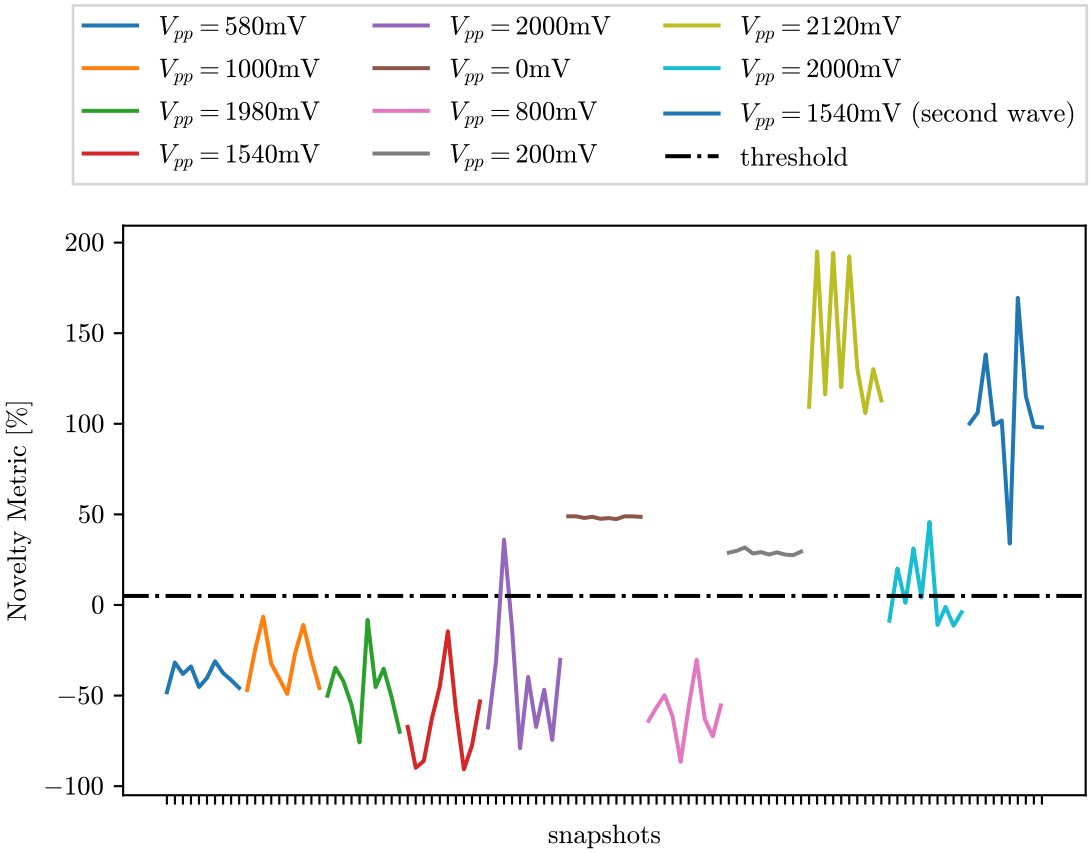


Figure 8.24: *Novelty detection result*

The result of the novelty detection is shown in **figure 8.24**. 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 decomposition is still generating features that are informative.

The pink line is the test signal with an 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 8.25** 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 grey 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 modelled 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 **subsection 4.1.11**. For completness, in **figure 8.25**, also the true positive results are shown, as magenta dots.

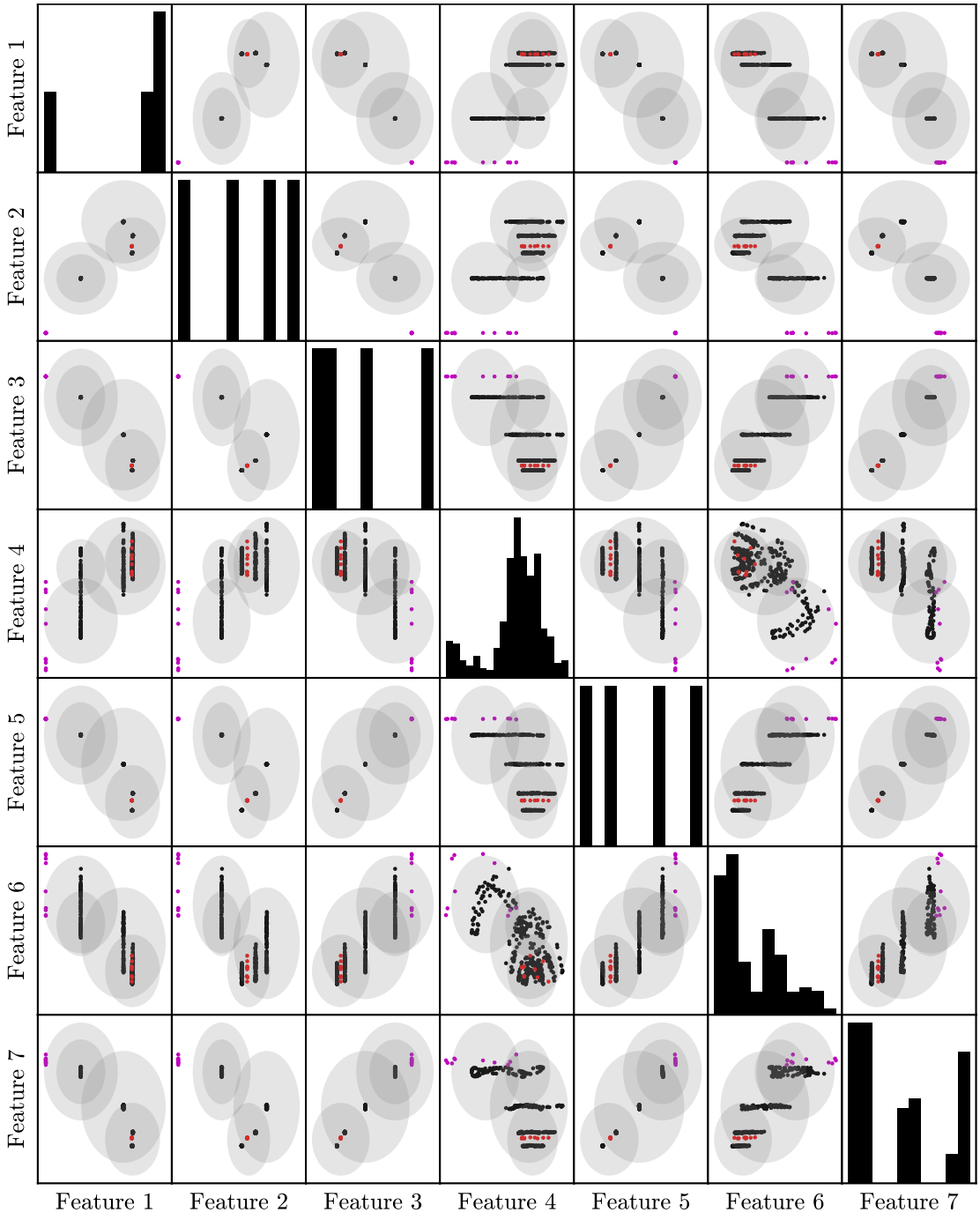


Figure 8.25: *False Negative and True Positive results. On the diagonal, there is a histogram of the feature values. The off-diagonal plots are the scatter plots of the features. The shades are the projection of the clusters on the considered plane. (Red: False Negative, Magenta: True Positive, Black: training data)*

8.6.3 Possible improvements

The environment of this test is very challenging for the k-means algorithm. As discussed in **chapter 4**, 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. 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.

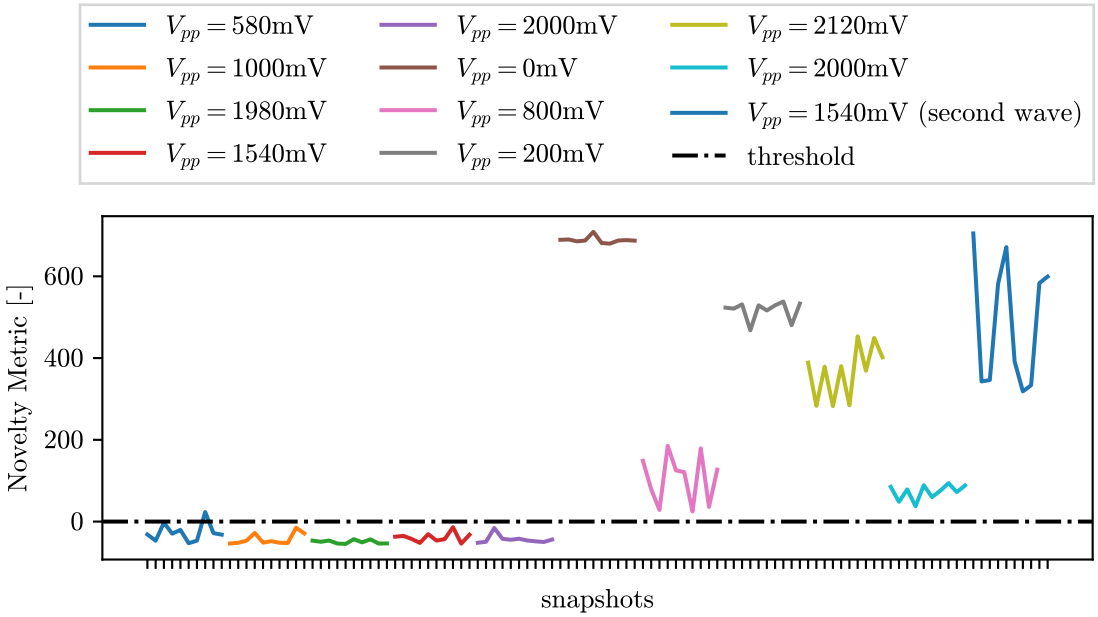


Figure 8.26: *LOF novelty detection result*

8.7 Experimental validation on a linear axis

The experimental validation reported in the previous **section 8.5** and **section 8.6** 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 8.2** 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 8.27**, and the parameters of the profiles are resumed in **table 8.5**.

Figure 8.27: *Position reference for the linear axis test.*

Table 8.5: *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

8.7.1 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 the 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 8.28**, and the features are shown in **figure 8.29**.

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 outliers are caused by the axis control, and the investigation about why it happens 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.

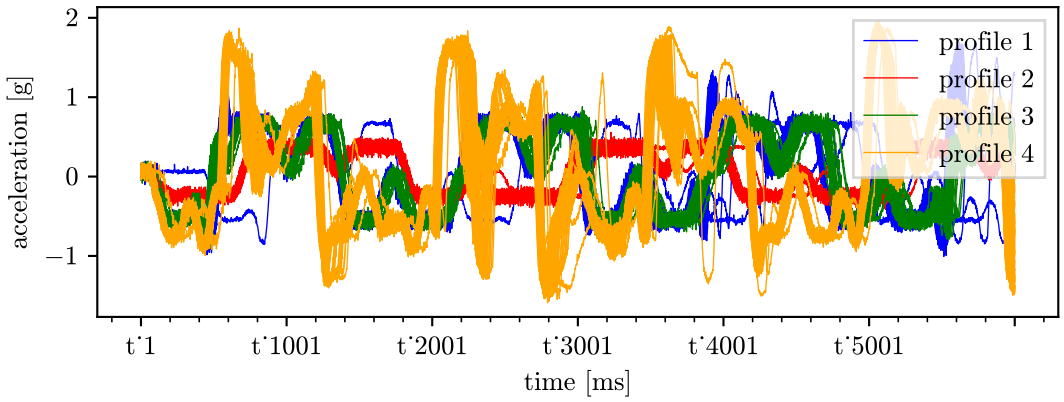


Figure 8.28: *Timeseries of the training set*

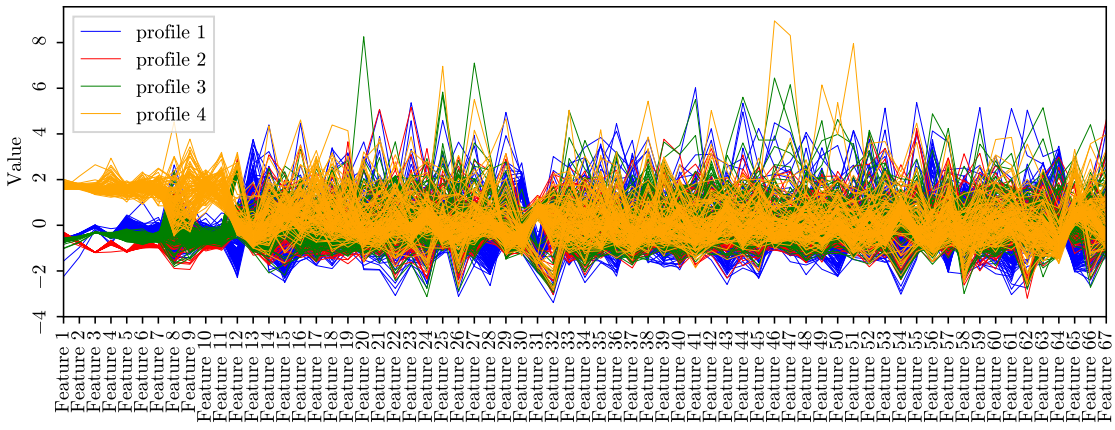


Figure 8.29: *features of the training set*

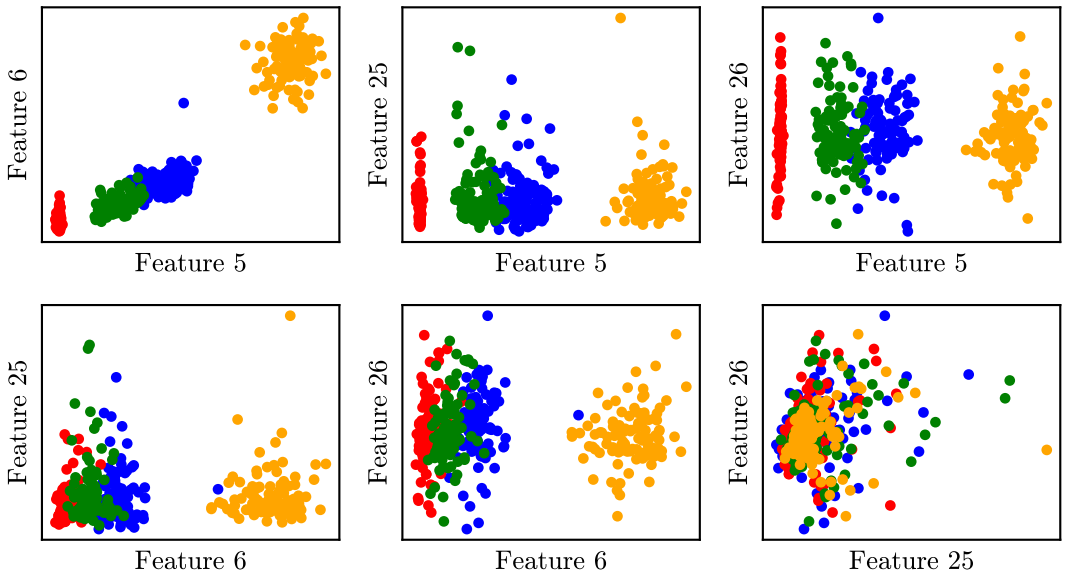


Figure 8.30: *Visualization of the separation between profiles in the feature space*

8.7.2 Testing on a known profile

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 8.31**. 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 8.29**), 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 8.30**, 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. Features 25 and 26, instead, are not significative, as the data points are not grouped by profile.

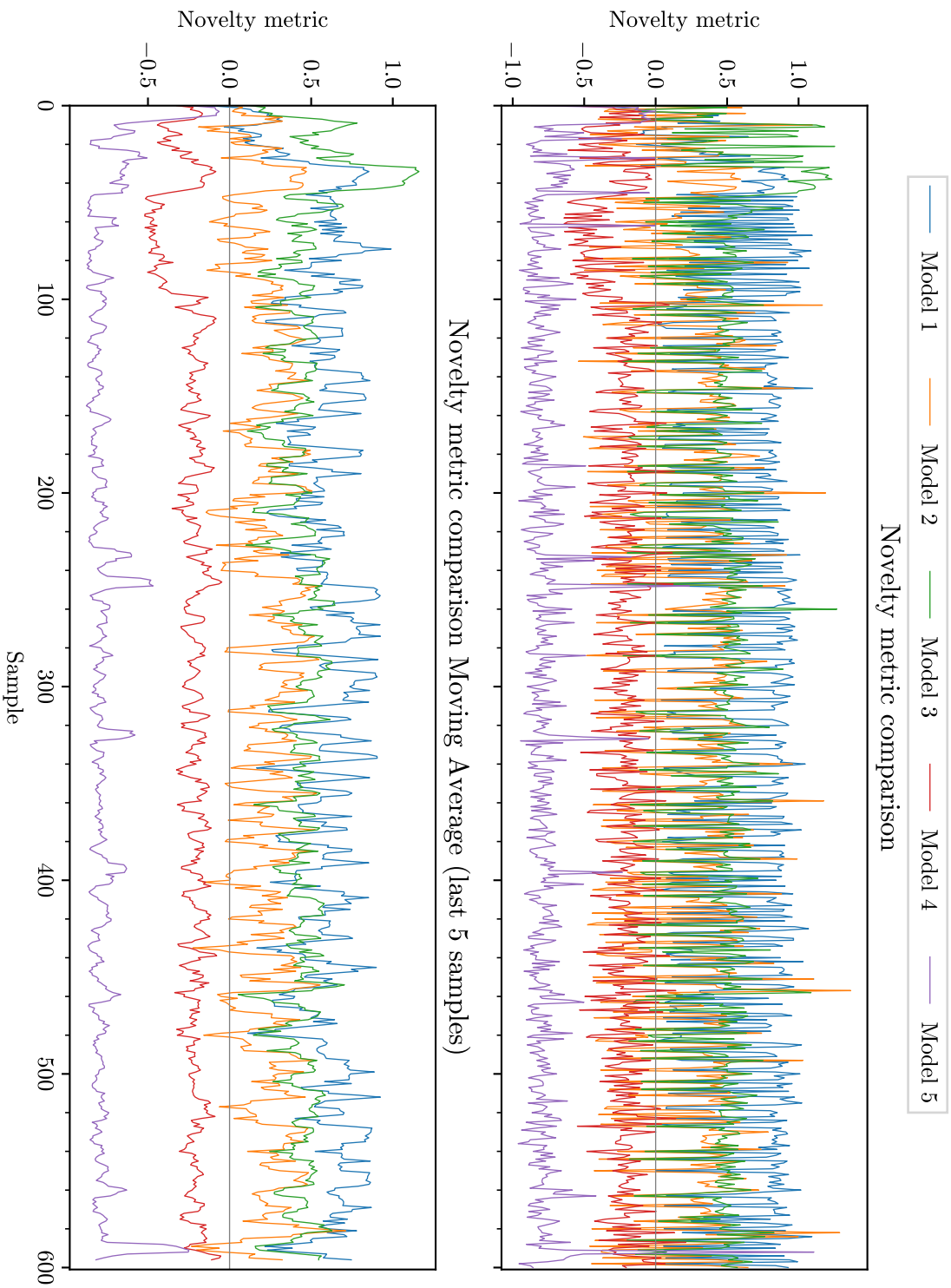


Figure 8.31: *Novelty detection on profile 2.*

8.7.3 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 of applying 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 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 8.32**.

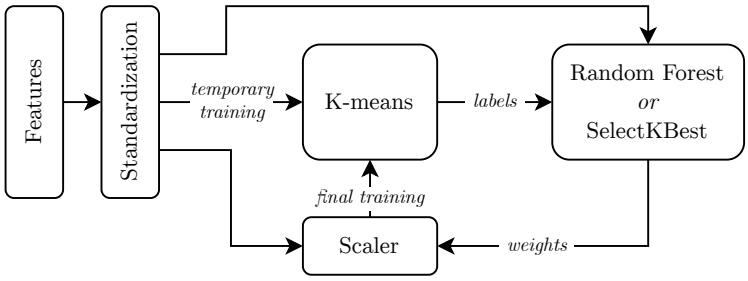


Figure 8.32: *Feature scaling procedure.*

This scaling technique is used with several configurations for both scaling the feature and reducing the number of features, by discarding the unnecessary ones. The settings of each tuned model are resumed in **table 8.6**. These models are described later in this section.

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 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 [70]. This concept, averaged over all splits of all the trees, gives a measure of the importance of each feature.

Table 8.6: *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	100	100	100	6

For our purpose, the importance 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

At this point, both the training of the RF and SelectKBest is performed. The weights obtained with both methods are shown in **figure 8.33**. The two methods gave very similar results. Now it is possible to train the MLA with the scaled features, Model 2 and Model 3 are trained with the RF and SelectKBest weights, respectively. Both models perform better than the original model because they give a smaller novelty score for the testing set. However, the problem of eliminating the false positive results is not yet solved, because the novelty score is still greater than zero.

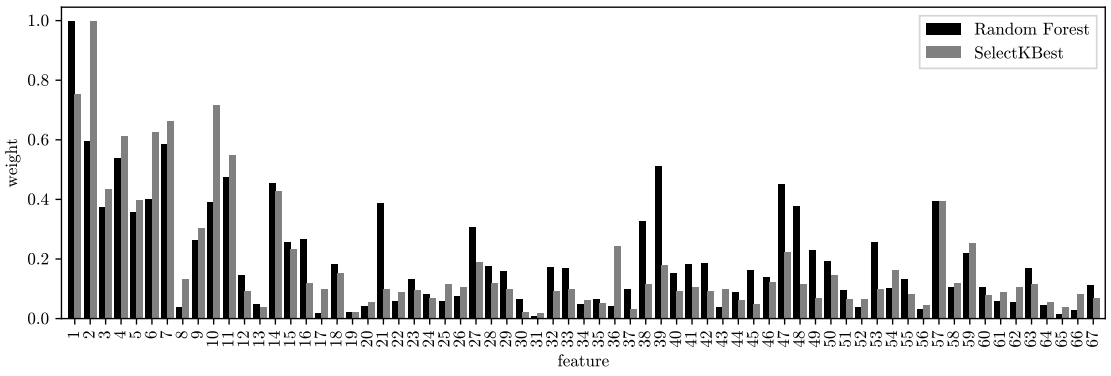


Figure 8.33: *Feature weights obtained with the RF and SelectKBest algorithms*

To further refine the model, a fourth version is provided, this time extending the training set to include the first 100 samples of this testing set. So the Model 4 will be trained on a total of 200 snapshots for profile 2. In the **figure 8.31**, of course, the novelty score is always negative for the first 100 samples, because they are part of the training set. Then, the novelty score becomes positive only in isolated cases, and even then it remains quite low. In this scenario, a novelty threshold of 20% is enough to run all the dataset without any false positive result, even without the activation of the outlier filter.

8.7.4 Testing the ND

The previous experiment was done to test the capability of the framework to identify a known profile. Now the framework is tested to identify a profile that is not part of the training set, so it should trigger a ND event. The test consists of actuating the axis alternating between unknown profiles and known profiles (10 snapshots each repetition). For reference, also Model 1, Model 2, and Model 3 are used, even if they were discarded in the previous section.

The results are shown in **figure 8.34**. Model 4 performs poorly because, even if it avoids completely giving false positive results, it gives a lot of false negative results. Unknown profiles are not detected as novelties, or they are detected only in isolated cases, with a very low novelty score that is not enough to trigger the ND event, considering the threshold discussed in the previous section.

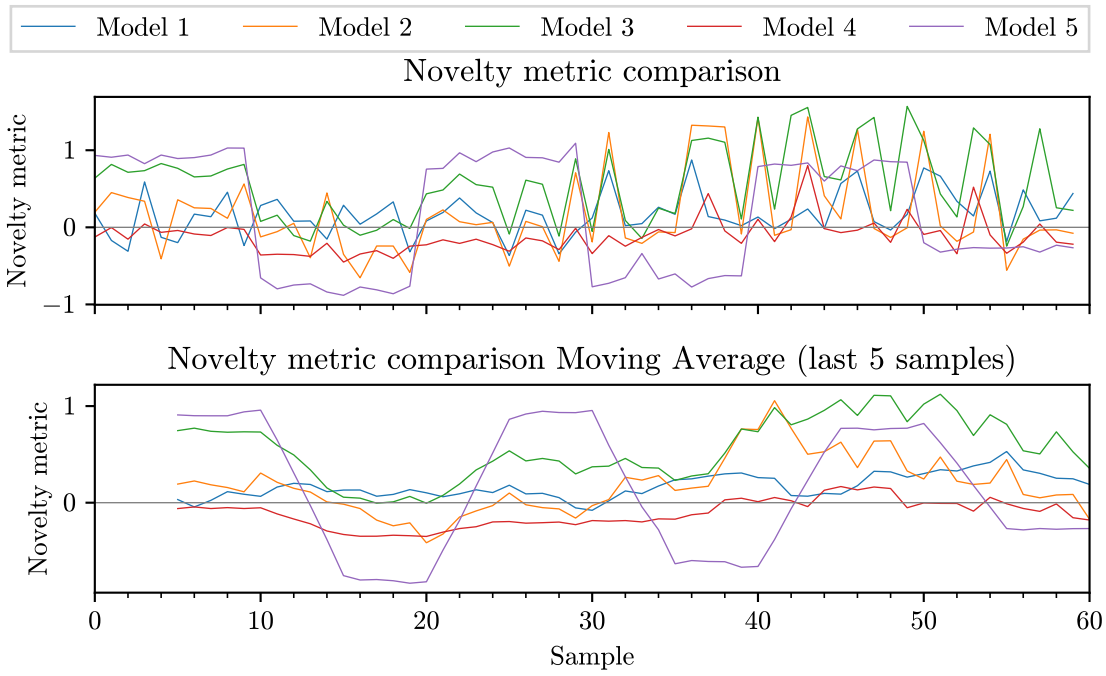


Figure 8.34: *Novelty detection on known and unknown profiles*

Chapter 9

Conclusion

Appendix A

Fast Fourier Transform

Appendix B

Wavelet Transform

Bibliography

[1] Vladimir Zwass. *Software Agent*. Category: Science & Tech. URL: https://www.britannica.com/technology/software-agent (cit. on p. xi).

[2] CEN/TC 319. *Maintenance - Maintenance terminology*. CEN Standard EN 13306:2018. English, French, and German language. European Committee for Standardization (CEN), 2018, p. 93 (cit. on pp. xi–xiii, 9).

[3] Vijay Kotu and Bala Deshpande. «Chapter 10 - Deep Learning». In: *Data Science (Second Edition)*. Ed. by Vijay Kotu and Bala Deshpande. Second Edition. Morgan Kaufmann, 2019, pp. 307–342. ISBN: 978-0-12-814761-0. DOI: https://doi.org/10.1016/B978-0-12-814761-0.00010-1. URL: https://www.sciencedirect.com/science/article/pii/B9780128147610000101 (cit. on p. xi).

[4] Accenture. *What is edge computing?* URL: https://www.accenture.com/us-en/insights/cloud/edge-computing-index#:~:text=Edge%20computing%20is%20an%20emerging,led%20results%20in%20real%20time. (cit. on p. xi).

[5] Marc HJ Romanycia and Francis Jeffry Pelletier. «What is a heuristic?» In: *Computational intelligence* 1.1 (1985), pp. 47–58 (cit. on p. xii).

[6] Alina Lazar and Bradley A. Shellito. *Linearly Separable Data*. IGI Global, 2009. Chap. 14, p. 7. DOI: 10.4018/978-1-59140-995-3.ch014 (cit. on p. xii).

[7] Craig S. Mullins, Jack Vaughan, and Barney Beal. «NoSQL (Not Only SQL database)». In: *TechTarget Network* (2021). Last updated in April 2021. URL: https://www.techtarget.com/searchdatamanagement/definition/NoSQL-Not-Only-SQL#:~:text=NoSQL%20is%20an%20approach%20to,%2C%20distributed%2C%20flexible%20and%20scalable. (cit. on p. xii).

[8] ISO/TC 108/SC 5. «Condition monitoring and diagnostics of machines - Prognostics - Part 1: General guidelines». In: *ISO 13381-1* (Sept. 2015). Status: Published, Edition: 2, ICS: 17.160, Corrected version (fr): 2021-12, Stage: International Standard to be revised [90.92], p. 21. URL: https://www.iso.org/standard/51436.html (cit. on p. xiii).

[9] Elizabeth Peterson. «Who Invented the Steam Engine?» In: *Live Science* (Mar. 2014). URL: https://www.livescience.com/44144-who-invented-the-steam-engine.html (cit. on p. 3).

[10] Ferguson Brothers Ltd. *Triple Expansion Reciprocating Steam Engine of Engine Grab Hopper Dredger Anadrian*. Photograph. Malta Maritime Museum Collection, Wikimedia Commons. Wikimedia Commons, 1951. URL: https://commons.wikimedia.org/wiki/File:Triple_expansion_reciprocating_steam_engine_Anadrian_MMM_n03.jpg (cit. on p. 3).

[11] *Evosite Control Rooms, Control Consoles & Ergonomic Chairs*. Accessed: February 26, 2024. Evosite, Inc. 2023. URL: https://www.evosite.net/ (cit. on p. 3).

[12] Brian A. Weiss Douglas S. Thomas. *Economics of Manufacturing Machinery Maintenance*. Tech. rep. U.S. Department of Commerce, 2020. URL: https://doi.org/10.6028/NIST.AMS.100-34 (cit. on pp. 3, 9).

[13] Wieger Tiddens, Jan Braaksma, and Tiedo Tinga. «Exploring predictive maintenance applications in industry». In: *Journal of quality in maintenance engineering*. 28.1 (2022). ISSN: 1355-2511. DOI: 10.1108/JQME-05-2020-0029. URL: https://doi.org/10.1108/JQME-05-2020-0029 (cit. on pp. 3, 4).

[14] Moamin A. Mahmoud, Naziffa Raha Md Nasir, Mathuri Gurunathan, Preveena Raj, and Salama A. Mostafa. «The Current State of the Art in Research on Predictive Maintenance in Smart Grid Distribution Network: Fault's Types, Causes, and Prediction Methods—A Systematic Review». In: *Energies* 14.16 (2021). ISSN: 1996-1073. DOI: 10.3390/en14165078. URL: https://www.mdpi.com/1996-1073/14/16/5078 (cit. on p. 4).

[15] 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 pp. 5, 17, 110).

[16] International Organization for Standardization. «ISO 8601 — Date and time format». In: (2019). Date and time format, Representations for information interchange. URL: https://www.iso.org/iso-8601-date-and-time-format.html (cit. on p. 6).

[17] P Coandă, M Avram, and V Constantin. «A state of the art of predictive maintenance techniques». In: *IOP Conference Series: Materials Science and Engineering* 997.1 (Dec. 2020), p. 012039. DOI: 10.1088/1757-899X/997/1/012039. URL: https://dx.doi.org/10.1088/1757-899X/997/1/012039 (cit. on p. 8).

[18] Ali Rastegari. «Condition Based Maintenance in the Manufacturing Industry: From Strategy to Implementation». ISBN 978-91-7485-355-1, ISSN 1651-4238. PhD Thesis. School of Innovation, Design and Engineering: Mälardalen University, 2017. URL: https://www.researchgate.net/publication/321883047 (cit. on p. 9).

[19] J. Moubray. *Reliability-Centered Maintenance*. Revised. Industrial Press, Inc., 1997 (cit. on p. 9).

[20] Daniel Sillivant. «Reliability centered maintenance cost modeling: Lost opportunity cost». In: Jan. 2015, pp. 1–5. DOI: 10.1109/RAMS.2015.7105111 (cit. on pp. 9, 10).

[21] H. M. Hashemian. «State-of-the-Art Predictive Maintenance Techniques». In: *IEEE Transactions on Instrumentation and Measurement* 60.1 (2011), pp. 226–236. DOI: 10.1109/TIM.2010.2047662 (cit. on p. 11).

[22] A. Grall, L. Dieulle, C. Berenguer, and M. Roussignol. «Continuous-time predictive-maintenance scheduling for a deteriorating system». In: *IEEE Transactions on Reliability* 51.2 (2002), pp. 141–150. DOI: 10.1109/TR.2002.1011518 (cit. on p. 11).

[23] Giuseppe Curcurù, Giacomo Galante, and Alberto Lombardo. «A predictive maintenance policy with imperfect monitoring». In: *Reliability Engineering & System Safety* 95.9 (2010), pp. 989–997. ISSN: 0951-8320. DOI: https://doi.org/10.1016/j.ress.2010.04.010. URL: https://www.sciencedirect.com/science/article/pii/S0951832010000955 (cit. on p. 11).

[24] G. Galante, A. Lombardo, and A. Passannanti. «Tool-life modelling as a stochastic process». In: *International Journal of Machine Tools and Manufacture* 38.10 (1998), pp. 1361–1369. ISSN: 0890-6955. DOI: https://doi.org/10.1016/S0890-6955(98)00019-4. URL: https://www.sciencedirect.com/science/article/pii/S0890695598000194 (cit. on p. 11).

[25] Jinjiang Wang, Laibin Zhang, Lixiang Duan, and Robert Gao. «A new paradigm of cloud-based predictive maintenance for intelligent manufacturing». In: *Journal of Intelligent Manufacturing* 28 (June 2017), pp. 1125–1137. DOI: 10.1007/s10845-015-1066-0 (cit. on p. 11).

[26] J. Cucurull, R. Martí, G. Navarro-Arribas, S. Robles, B. Overeinder, and J. Borrell. «Agent mobility architecture based on IEEE-FIPA standards». In: *Computer Communications* 32.4 (2009), pp. 712–729. ISSN: 0140-3664. DOI: https://doi.org/10.1016/j.comcom.2008.11.038. URL: https://www.sciencedirect.com/science/article/pii/S014036640800618X (cit. on p. 11).

[27] Francesca Calabrese, Alberto Regattieri, Marco Bortolini, Mauro Gamberi, and Francesco Pilati. «Predictive Maintenance: A Novel Framework for a Data-Driven, Semi-Supervised, and Partially Online Prognostic Health Management Application in Industries». In: *Applied Sciences* 11.8 (2021). ISSN: 2076-3417. DOI: 10.3390/app11083380. URL: https://www.mdpi.com/2076-3417/11/8/3380 (cit. on p. 12).

[28] Irfan Ullah, Fan Yang, Rehanullah Khan, Ling Liu, Haisheng Yang, Bing Gao, and Kai Sun. «Predictive Maintenance of Power Substation Equipment by Infrared Thermography Using a Machine-Learning Approach». In: *Energies* 10.12 (2017). ISSN: 1996-1073. DOI: 10.3390/en10121987. URL: https://www.mdpi.com/1996-1073/10/12/1987 (cit. on p. 12).

[29] Yongyi Ran, Xin Zhou, Pengfeng Lin, Yonggang Wen, and Ruilong Deng. *A Survey of Predictive Maintenance: Systems, Purposes and Approaches*. 2019. arXiv: 1912.07383 [eess.SP] (cit. on pp. 12, 13).

[30] Andrea Schirru, Simone Pampuri, and Giuseppe De Nicolao. «Particle filtering of hidden Gamma processes for robust Predictive Maintenance in semiconductor manufacturing». In: *2010 IEEE International Conference on Automation Science and Engineering*. 2010, pp. 51–56. DOI: 10.1109/COASE.2010.5584518 (cit. on pp. 12, 13).

[31] C. Yang, Q. Lou, J. Liu, and et al. «Particle filtering-based methods for time to failure estimation with a real-world prognostic application». In: *Applied Intelligence* 48 (Aug. 2018), pp. 2516–2526. DOI: 10.1007/s10489-017-1083-0. URL: https://doi.org/10.1007/s10489-017-1083-0 (cit. on pp. 12, 13).

[32] Alexander von Birgelen, Davide Buratti, Jens Mager, and Oliver Niggemann. «Self-Organizing Maps for Anomaly Localization and Predictive Maintenance in Cyber-Physical Production Systems». In: *Procedia CIRP* 72 (2018). 51st CIRP Conference on Manufacturing Systems, pp. 480–485. ISSN: 2212-8271. DOI: https://doi.org/10.1016/j.procir.2018.03.150. URL: https://www.sciencedirect.com/science/article/pii/S221282711830307X (cit. on pp. 12, 13).

[33] GRS Lira, EG Costa, VS Brito, and LAMM NOBREGA. «Adaptive Resonance Theory Applied to MOSA Monitoring». In: *Proceedings of the XVII international symposium on high voltage engineering. Hannover, Germany*. 2011 (cit. on pp. 12, 13).

[34] Robert B. Randall and Jérôme Antoni. «Rolling element bearing diagnostics—A tutorial». In: *Mechanical Systems and Signal Processing* 25.2 (2011), pp. 485–520. ISSN: 0888-3270. DOI: https://doi.org/10.1016/j.ymssp.2010.07.017. URL: https://www.sciencedirect.com/science/article/pii/S0888327010002530 (cit. on pp. 14, 17).

[35] Nader Sawalhi and Robert B Randall. «Semi-automated bearing diagnostics-Three case studies». In: *Non Destructive Testing Australia* 45.2 (2008), p. 59 (cit. on p. 14).

[36] Michael Schlechtingen and Ivan Santos. «Automated wind turbine gearbox bearing diagnosis algorithm based on vibration data analysis and signal pre-whitening». In: *Proceedings of 13th SIRM: The 13th International Conference on Dynamics of Rotating Machinery*. Technical University of Denmark. 2019, pp. 88–114 (cit. on pp. 14, 15).

[37] Marco A.F. Pimentel, David A. Clifton, Lei Clifton, and Lionel Tarassenko. «A review of novelty detection». In: *Signal Processing* 99 (2014), pp. 215–249. ISSN: 0165-1684. DOI: https://doi.org/10.1016/j.sigpro.2013.12.026. URL: https://www.sciencedirect.com/science/article/pii/S016516841300515X (cit. on pp. 15, 16).

[38] Dubravko Miljković. «Review of novelty detection methods». In: May 2010, pp. 593–598. ISBN: 978-1-4244-7763-0 (cit. on p. 16).

[39] 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 pp. 17, 18, 109).

[40] Abla Chouni Benabdellah, Asmaa Benghabrit, and Imane Bouhaddou. «A survey of clustering algorithms for an industrial context». In: *Procedia Computer Science* 148 (2019). The second international conference on intelligent computing in data sciences, ICDS2018, pp. 291–302. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2019.01.022. URL: https://www.sciencedirect.com/science/article/pii/S1877050919300225 (cit. on pp. 18, 19, 34).

[41] Aurelien Geron. *Hands-On Machine Learning With Scikit-Learn, Keras & TensorFlow*. English. O'Reilly Media, 2022, p. 850. ISBN: 9781098125974 (cit. on pp. 20, 26–28, 31–34, 47, 52, 56, 58, 60).

[42] Sebastian Raschka. «Why are implementations of decision tree algorithms usually binary and what are the advantages of the different impurity metrics?» In: *Machine Learning FAQ* (2013). Available at: http://sebastianraschka.com/faq/docs/decision-tree-binary.html (cit. on p. 30).

[43] Meena Mahajan, Prajakta Nimbhorkar, and Kasturi Varadarajan. «The planar k-means problem is NP-hard». In: *Theoretical Computer Science* 442 (2012). Special Issue on the Workshop on Algorithms and Computation (WALCOM 2009), pp. 13–21. ISSN: 0304-3975. DOI: https://doi.org/10.1016/j.tcs.2010.05.034. URL: https://www.sciencedirect.com/science/article/pii/S0304397510003269 (cit. on p. 34).

[44] James MacQueen et al. «Some methods for classification and analysis of multivariate observations». In: *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*. Vol. 1. 14. Oakland, CA, USA. 1967, pp. 281–297 (cit. on p. 34).

[45] S. Lloyd. «Least squares quantization in PCM». In: *IEEE Transactions on Information Theory* 28.2 (1982), pp. 129–137. DOI: 10.1109/TIT.1982.1056489 (cit. on pp. 34, 37).

[46] Pavel Berkhin. *Survey of clustering data mining techniques*. Tech. rep. San Jose, CA: Accrue Software, 2002 (cit. on p. 34).

[47] Hans-Peter Kriegel, Erich Schubert, and Arthur Zimek. «The (black) art of runtime evaluation: Are we comparing algorithms or implementations?» In: *Knowledge and Information Systems* 52.2 (Oct. 2017), pp. 341–378. ISSN: 0219-3116. DOI: 10.1007/s10115-016-1004-2. URL: https://doi.org/10.1007/s10115-016-1004-2 (cit. on p. 34).

[48] Malay K. Pakhira. «A Linear Time-Complexity k-Means Algorithm Using Cluster Shifting». In: *2014 International Conference on Computational Intelligence and Communication Networks*. 2014, pp. 1047–1051. DOI: 10.1109/CICN.2014.220 (cit. on p. 37).

[49] Mary Inaba, Naoki Katoh, and Hiroshi Imai. «Applications of Weighted Voronoi Diagrams and Randomization to Variance-Based k-Clustering: (Extended Abstract)». In: *Proceedings of the Tenth Annual Symposium on Computational Geometry*. SCG '94. Stony Brook, New York, USA: Association for Computing Machinery, 1994, pp. 332–339. ISBN: 0897916484. DOI: 10.1145/177424.178042. URL: https://doi.org/10.1145/177424.178042 (cit. on p. 37).

[50] Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. «A local search approximation algorithm for k-means clustering». In: *Comput. Geom.* 28.2-3 (2004), pp. 89–112 (cit. on p. 37).

[51] Charles Elkan. «Using the Triangle Inequality to Accelerate K-Means». In: *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*. ICML'03. Washington, DC, USA: AAAI Press, 2003, pp. 147–153. ISBN: 1577351894 (cit. on p. 37).

[52] D. Sculley. «Web-Scale k-Means Clustering». In: *Proceedings of the 19th International Conference on World Wide Web*. WWW '10. Raleigh, North Carolina, USA: Association for Computing Machinery, 2010, pp. 1177–1178. ISBN: 9781605587998. DOI: 10.1145/1772690.1772862. URL: https://doi.org/10.1145/1772690.1772862 (cit. on p. 37).

[53] Damodar Reddy and Prasanta K. Jana. «Initialization for K-means Clustering using Voronoi Diagram». In: *Procedia Technology* 4 (2012). 2nd International Conference on Computer, Communication, Control and Information Technology(C3IT-2012) on February 25 - 26, 2012, pp. 395–400. ISSN: 2212-0173. DOI: https://doi.org/10.1016/j.protcy.2012.05.061. URL: https://www.sciencedirect.com/science/article/pii/S2212017312003404 (cit. on p. 37).

[54] David Arthur and Sergei Vassilvitskii. «K-Means++: The Advantages of Careful Seeding». In: vol. 8. Jan. 2007, pp. 1027–1035. DOI: 10.1145/1283383.1283494 (cit. on p. 37).

[55] Lilian Bejarano, Helbert Espitia, and Carlos Montenegro. «Clustering Analysis for the Pareto Optimal Front in Multi-Objective Optimization». In: *Computation* 10 (Mar. 2022), p. 37. DOI: 10.3390/computation10030037 (cit. on p. 38).

[56] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. «A density-based algorithm for discovering clusters in large spatial databases with noise». In: *kdd*. Vol. 96. 34. 1996, pp. 226–231 (cit. on pp. 47, 49).

[57] Bing Liu. «A Fast Density-Based Clustering Algorithm for Large Databases». In: *2006 International Conference on Machine Learning and Cybernetics*. 2006, pp. 996–1000. DOI: 10.1109/ICMLC.2006.258531 (cit. on p. 52).

[58] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. «Isolation Forest». In: *2008 Eighth IEEE International Conference on Data Mining*. 2008, pp. 413–422. DOI: 10.1109/ICDM.2008.17 (cit. on pp. 56, 57).

[59] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. «LOF: Identifying Density-Based Local Outliers». In: *SIGMOD Rec.* 29.2 (May 2000), pp. 93–104. ISSN: 0163-5808. DOI: 10.1145/335191.335388. URL: https://doi.org/10.1145/335191.335388 (cit. on p. 58).

[60] Klaus-Robert Müller, Sebastian Mika, Gunnar Rätsch, Koji Tsuda, and Bernhard Schölkopf. «An Introduction to Kernel-Based Learning Algorithms». In: *IEEE TRANSACTIONS ON NEURAL NETWORKS* 12.2 (2001), p. 181 (cit. on p. 59).

[61] 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 pp. 61–63, 99).

[62] James W Cooley and John W Tukey. «An algorithm for the machine calculation of complex Fourier series». In: *Mathematics of computation* 19.90 (1965), pp. 297–301 (cit. on p. 64).

[63] Arvid Breitenbach. «Against spectral leakage». In: *Measurement* 25.2 (1999), pp. 135–142. ISSN: 0263-2241. DOI: https://doi.org/10.1016/S0263-2241(98)00074-8. URL: https://www.sciencedirect.com/science/article/pii/S0263224198000748 (cit. on p. 66).

[64] Chao-Yen Wu and A.Terry Bahill. «Preprocessing methods in the computation of the fast fourier transform». In: *Computers & Industrial Engineering* 21.1 (1991), pp. 653–657. ISSN: 0360-8352. DOI: https://doi.org/10.1016/0360-8352(91)90168-6. URL: https://www.sciencedirect.com/science/article/pii/0360835291901686 (cit. on p. 68).

[65] Fagor Automation. *IBARMIA - 5 AXES MACHINING CNC 8065*. Sept. 2017. URL: https://www.flickr.com/photos/fagorautomation/35762600461/ (cit. on p. 75).

[66] Victorchan An and William Meeker. «Estimation of Degradation-Based Reliability in Outdoor Environments». In: (Oct. 2001) (cit. on p. 77).

[67] Mingming Yan, Xingang Wang, Bingxiang Wang, Miaoxin Chang, and Isyaku Muhammad. «Bearing remaining useful life prediction using support vector machine and hybrid degradation tracking model». In: *ISA transactions* 98 (2020), pp. 471–482 (cit. on p. 77).

[68] Jean Jacquelin. *Régressions Et Équations Intégrales*. Première édition : 14 janvier 2009 - Mise à jour : 3 janvier 2014. 2009, pp. 16–18. URL: https://www.scribd.com/doc/14674814/Regressions-et-equations-integrales (cit. on p. 79).

[69] Rafat Hussain. *wavelib*. https://github.com/rafat/wavelib.git. Dec. 2014 (cit. on pp. 97, 98).

[70] 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. 126).