Università degli Studi di Napoli
Federico II

Scuola Politecnica e delle Scienze di Base
Dipartimento di Ingegneria Elettrica e Tecnologie dell'Informazione

Doctoral Thesis

Failure Predictions and Anomaly Detections by Artificial Intelligence with Heterogeneous Measures

Author:
Federico Gargiulo

Tutor:
Prof. Pasquale Arpaia

Submitted in fulfilment of the requirements for the degree of Doctor of Philosophy in Information Technology and Electrical Engineering, XXXIV Cycle.
Coordinator: Prof. Daniele Riccio.

08 March 2022

Resourcefulness drives the world.

Abstract

The changes in the manufacturing world driven by the revolution of industry 4.0 entail a growing volume of information and the demand for planning in production's tasks and activities. The techniques of measurement, transport, analysis and processing of data are constantly evolving. The technological growth driven by the evolution of industry 4.0 opens up new frontiers of research in the field of predictive maintenance, a theme that is deeply felt in highly competitive production contexts. In this PhD thesis, the topic of predictive maintenance is addressed by means of a research work on measurement techniques, data processing and Machine Learning (ML) algorithms. The research works presented in this thesis have focused on 3 important issues in modern industry. The first work explores the techniques for predictive maintenance on machines for the compression of cryogenic fluids and a solution based on unsupervised machine learning techniques for diagnostics is proposed; the second work concerns techniques for the identification of faults and diagnostics of magic hard disks and a method for the prediction of failures based on decision forests is proposed; the third work focuses on measurement, processing and machine learning techniques for the prevention of failure on three-phase asynchronous electric motors by means of fault detection approach.

As for the first research work, a fault detection method exploiting Hidden Markov Models (HMMs) is proposed for fluid machinery without adequate a-priori information about faulty conditions. The method was tested and validated at CERN on screw compressors for cryogenic cooling.

As second research work a method to facilitate automated proactive disk replacement is proposed. The method identifies disks with media failures in a production environment and adopts an application of supervised machine learning, based on Regularized Greedy Forest, to predict disk failures. In particular, a proper stage to automatically label (healthy/at-risk) the disks during the training and validation stage is presented along with tuning strategy to optimize the hyperparameters of the associated machine learning classifier. The machine learning model is trained and validated against a large set of 65,000 hard drives in the CERN computer center, and the remarkable achieved results are discussed.

Finally a method to identify electrical and mechanical faults in three-phase asynchronous electric is proposed in order to prevent device failures. A mea-

surement strategy and machine learning algorithm, based on Artificial Neural Network, is proposed to properly classify failures. The method is validated on a set of 28 electric motors. The method's performance is compared with common state of art machine learning techniques.

Acknowledgements

I would like to firstly thank my PhD Tutor Prof. Pasquale Arpaia and all teachears helped me along the way: Prof. Rosario Schiano Lo Moriello, Prof. Annalisa Liccardo and Prof. Luca De Vito.
I would sincerly like to thank Dr. Marco Pezzetti and Dr. Dirk Duellmann for the pleasant work environment I found.
Finally, I thank Eng. Umberto Cesaro, Eng. Federico De Vitiis, Eng. Andrea Rocco and Meridiana Aspiratori's staff for the technical support in the research work on three-phase asynchronous motors.

Contents

1 Introduction

Digital growth in manufacturing involves an evolution of production models. The fourth industrial revolution (Industry 4.0) is bringing, on the one hand, a more advanced automation thanks to the interconnection of robots and machinery, and on the other hand, the maintenance of systems and industrial plants have the opportunity to make a qualitative leap, adopting a new approach to maintenance. This new approach aims to mitigate extraordinary interventions in terms of importance and unpredictability. This new type of maintenance is referred to as predictive and has been developing in the last decade thanks to the new technologies described above.

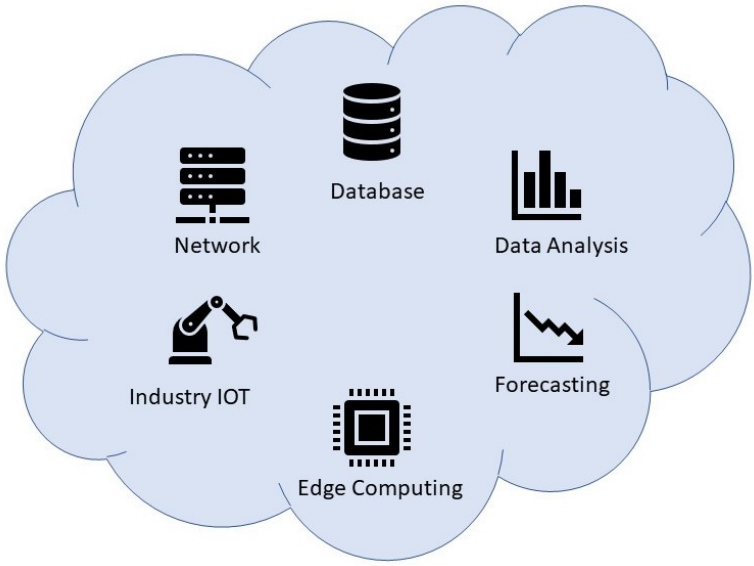

Figure 1.1: Industry 4.0

Thus, Industry 4.0 is the new frontier of manufacturing. The evolution of

production systems involves the entire industrial structure thanks to a greater use of data and information coming from totally digitized and connected components and systems.
Modern industry is focusing on new maintenance procedures because production efficiency is becoming crucial in order to decrease costs related to quality production and machine downtime[47].

The downtime production costs lead to a significant increase in demand for reliability and availability in modern industrial manufacturing. In the last decade the literature on diagnostic techniques has been populated with new methodologies for the improvement of availability and reliability in a plethora of different contexts. The growing complexity of electromechanical systems and the integration of electronics on control units makes the analytical modeling of systems increasingly difficult for the prevention and mitigation of failures. The measures-based inferential approaches overcome the modeling difficulties such as heterogeneity of models, design complexity, in-depth knowledge, etc. Furthermore, with inference measures-based modeling it is possible to obtain a highly representative model of the case study device.

The modeling of a system, its component or its behavior can be done analytically, but this requires a deep knowledge of what is being modeled. The alternative may be measure-driven modeling. In the second case, Machine Learning (ML) algorithms are well suited for making inference and modelling systems. Machine learning techniques allow you to have a model of the system while having little or no previous knowledge of the system itself. Moreover, this feature is highly valid and useful in contexts where the devices on which you want to make the inference for various reasons cannot be removed from the work location or it is not possible to obtain an analytical model. The frontiers of edge computing open new horizons for technological progress. The computational capacity of microprocessors and Microcontrollers (MCU) has greatly advanced in the last decade. This technological advancement has made available considerable resources for the implementation of increasingly complex software. Advances in hardware and software, in addition to new network and communication technologies, nowadays allow access to a volume of data that has completely changed the structure of research on maintenance issues.

As described above, the wide availability of data is possible thanks to measurement, network and storage technologies. A failure is a deviation from the specifications of a system, it is caused by the activation of an error that becomes visible to the outside. An error, in turn, is the activation of a fault, that is, the deviation from the specifications of a component of the system. Moreover, an internal failure of a system can be seen as the failure of one of its components or subsystems[4]. This cascade propagation of a fault is schematized and summarized in the figure 1.2. The analysis of the conditions of a system and its components together with the growth of acquisition and analysis techniques

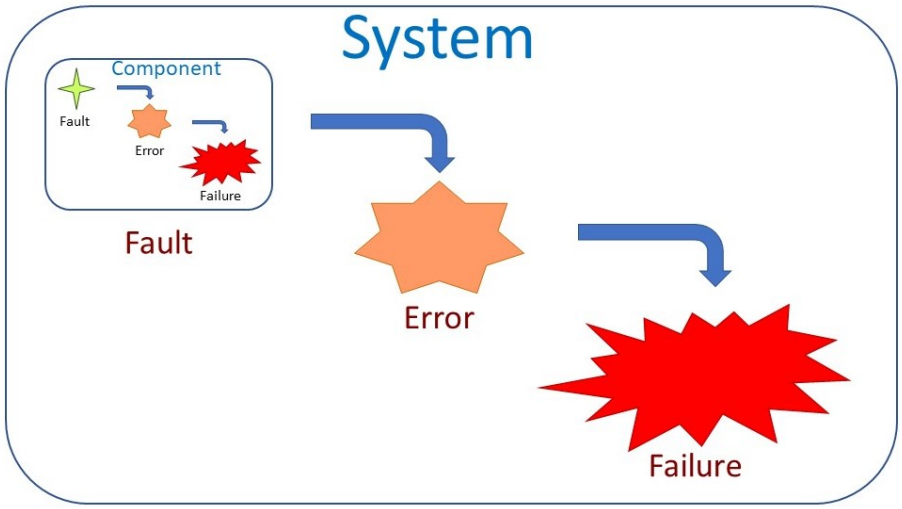

Figure 1.2: Component Failure

pushes the lines of research on diagnostics to integrate Artificial Intelligence (AI) techniques for predictive maintenance.

Machine learning-based predictive maintenance techniques follow 4 distinct phases (fig. 1.3) that all works of this type have in common. The works presented in this thesis adhere to these 4 phases. The first phase is the Data Acquisition in which all the necessary steps are carried out so that measurements and useful information from the system are collected (i.e. Acquisition Sensor Setup,Label Data, Store Data).

The second phase is that of Preprocessing in which the data acquired in the data acquisition phase are turned into a suitable way for the modelling input constraints. Preprocessing's steps usually include data cleaning tasks, normalization and features extraction.

Modelling phase is than executed as the third main passage. First of all, it is necessary to select a modeling method and an adequate configuration, then it is possible to carry out the actual modeling step. In a machine learning context, first of all it is necessary to identify whether you prefer a supervised or unsupervised modeling method, after which all the selections and optimizations of parameters and hyperparameters are made in order to achieve the desired results in terms of efficiency. Finally, it is necessary to deploy the prediction system and integrate it into the system on which the predictive maintenance is to be carried out. Of course, the deployment and integration phase requires a series of improvements and optimization to strengthen the system.

Figure 1.3: Steps of Predictive Maintenance based on Artificial Intelligence

1.1 Outline of the Thesis

This thesis deals with the issue of predictive maintenance by means of Machine Learning techniques. The issue is dealt with in the context of three very widespread problems in the industrial field. Each of these problems has in common the need to prevent and, as far as possible, mitigate the activation of faults and the occurrence of failures. This PhD thesis is conceptually divided into three parts: the first part is focused on Fault Detection on Fluid Machinery; the second is concerned with Hard Disk Failure Prediction; Failure Predictions in Injunction Motors is covered in the third part.

Chapter 2 illustrates the state of the art of predictive maintenance with and without machine learning in the three fields addressed in the following chapters.

In Chapter 3 a first predictive maintenance problem is investigated. The chapter explores the problem of preventing failures on fluid compression machinery. The result of the research activity presented in the third chapter demonstrates how the need to perform a manual analysis on screw compressors can be overcome, this is possible thanks to the use of algorithms for non-evident state modeling known as Hidden Markov Models (HMM).

The focus of Chapter 4 is on the prevention of failures in data storage systems, as cloud computing is a growing technology in industry 4.0. The chapter addresses the issue of preventing failures on magnetic hard disks (HDDs) as they represent the most common cause of failure in data storage systems. The problem of HDD failures is approached algorithmically both from the standpoint of failure detection and recognition, and from the standpoint of failure prevention. The machine learning technique adopted is based on the means of decision trees, this technique recently proposed in the literature is named Regularized Greedy Forest (RGF).

Chapter 5 deals with a third typical problem in production realities which concerns predictive maintenance in electric motors. The chapter explores the problem of failures in three-phase asynchronous motors, the measurement techniques, features extraction methods and the machine learning algorithms for modeling classifiers for the identification of imminent failures.

Conclusions and final remarks about Predictive Maintenance based on Machine Learning are drawn in Chapter 6.

2 State of the Art

In this Chapter the state of the art on failure prediction techniques is reviewed in order to explore the solutions of the literature with and without the aid of Artificial Intelligence (AI). Machine learning has opened new frontiers for measure based modeling. The range of AI has grown enormously since the intelligence of machines with machine learning capabilities has created profound impacts on business, governments, and society. [10]

Authors of the work [46] have classified maintenance management techniques into three classes, based on complexity level and efficiency. Higher the efficiency, higher the complexity. Following the three definitions are reported.

- Run-to-Failure (R2F) is the first maintenance approach in terms of complexity but this procedure entails less efficiency of the maintenance procedures. The maintenance procedures are performed after a failure have occurred, consequently, a maintenance plan can hardly be made;
- Preventive Maintenance (PvM) consists of carrying out a scheduled maintenance plan. This approach reduces the probability of failure occurring but also involves unnecessary action with increased costs;
- Predictive Maintenance (PdM) is an approach based on a statistical assessment of the system health status. Maintenance procedures are triggered by the Predictive Maintenance system. The Predictive Maintenance procedures are usually measure driven approach driven by statistical and inferential techniques[46].

The opportunities of Industry 4.0 create the conditions for greater connection between machines. Predictive maintenance is one of the main challenges in this area. Machine downtime, reduction of management and maintenance costs, process control and production quality are at the heart of technological evolution. Furthermore, in Industry 4.0, data analysis and machine learning methods to change production procedures do not always include predictive maintenance methods and their organization as well[63].

Maintenance costs in industrial production generally have an impact greater than 15% and which in some cases can reach 60% of the total manufacturing costs[21].

Predictive maintenance takes on greater importance in industrial contexts

with the introduction of machine learning and measurement-driven modeling. Machine Learning applications are increasing in the industrial sector as a result of the rapid growth of data availability and increasing computing capabilities. Currently in the machine learning sector, supervised algorithms represent the vast majority. However, unsupervised machine learning algorithms are assuming more important roles in industrial applications thanks to the evolution of sensors and growing storage and computing technologies. Moreover, it is possible to introduce hybrid approaches of the two techniques (supervised / unsupervised). Machine Learning is therefore opening many frontiers in the field of industry 4.0 and many opportunities for technological development[52].

2.1 Diagnostics in Screw Compressors

Fluid machines are devices exploiting the energy stored in fluids, by transforming it in mechanical energy or vice versa. These machines are widely employed in several industrial plants. Although these are highly reliable machines, a fault occurrence is a deleterious event in terms of correct operation, time, and costs. Fault detection mechanisms are in charge of identifying whether the monitored component or process is properly working or not [24]. In literature, a wide variety of fault detection mechanisms have been proposed. In particular, in [11], a review on fault detection methods for pipelines highlights that, usually, a single quantity is measured. Among the analyzed techniques and approaches, it is shown that procedures are usually complex when good fault detection performance is required. In [31], a review about wind turbines derives that vibration analysis has been widely used in fault diagnosis and feature extraction. However, gathering data about faulty conditions in actual working conditions turns out difficult, especially for complex machines like energy turbines or high-power compressors. Several specific signals and often custom invasive monitoring based on multiple specific sensors have to be installed directly on field. Potential faulty conditions of whatever type have to be forecast, modeled, and experimented. Finally, in [62, 38, 36], fault detection methods are proposed for the monitoring of compressors operation. The methods usually rely on the analysis of vibration signals, and it is shown that the training of the fault detection models exploits both measures related to normal and faulty conditions. This is also confirmed in [37], where faults are classified with a Support Vector Machine by taking into account a big amount of data (300 TB) to identify and validate the model.

A promising trend research in fault detection, with exceptional ability in modeling 1D sensor signals, is related to Hidden Markov Models (HMMs). A simple Markov Model is a "Markov Chain", i.e. a finite state machine governed by a stochastic process. This process is described with a probability matrix, in which each element a_{ij} is the probability of the transition from the state i to the state j (a_{ii} is the probability to remain in the same state). A Markov chain allows computing the probability for a sequence of states to be observed. In many cases, however, the states of interest may not be directly observable, and thus the state

sequence is 'hidden'. In such a case, it is possible to calculate the probability of the sequence according to the Markov model, as well as the state sequence that most likely produced the observations. Such a model is an HMM.

Firstly introduced in 1989 [40] to solve speech recognition problems, recently these models have been extended to pattern identification [56, 28], health monitoring, human action recognition, biological sequence analysis, economics [7], and fault detection. In particular, HMM-based approaches were used to monitor fault detection in mechanical devices, such as bearings [35, 57, 30, 58], or in industrial processes [19, 3, 50]. The wide use of this methodology may be attributed to the following reasons:

- applicability to embedded stochastic processes, whose states are not directly observable, but they can be deduced through another process;
- training easiness, due to the easiness in estimating the model parameters, even when training samples are abundant;
- independence of multiple models, i.e. the possibility to use a number of independent HMM to describe different conditions of a process.

By relying on the peculiar features of the HMM, a novel fault detection method is here proposed. The abovementioned works exemplify that most researches considered acoustic vibrations or employed accelerometers to measure the state of the monitored process. Moreover, measures were often obtained with a dense sampling. Instead, in actual applications, measures are sparse and asynchronous. Then, in general, one single physical quantity may not be sufficient to detect an incipient failure condition. For instance, in case of an operating anomaly, the machine could not have abnormal vibrations, but it may overheat. Finally, in fluid machinery with constraining requirements of reliability, the dynamics of the fault detection model is such that it will remain in a fault-free state for long stretches of time.

2.2 Failure Prediction in Hard Disks

An automated and standardized hard disk replacement process can be defined based on the results from automatic testing framework, which is integrated by disk manufacturers in their disk firmware: the SMART system. The hard disk models used in storage centers are often numerous and change over time. Due to the heterogeneity of hard disks of which datacenters are often composed, it is not possible to perform analytical modeling. So modeling by means of machine learning is the best solution in this context. Multiple recent approaches based on SMART attributes have been proposed and show improved prediction performance in general but are usually highly dependent on the specific device they were applied to.

A Gaussian Mixture based the fault detection approach has been proposed in 2017, which is able to minimize the False Alarm Rate (FAR) of 0%, but its performance drops by a few dozen percentage points in the hours before the last

24 h, and there is no information on the rate of false alarms in the days before the last. In addition, the method has been tested on a single hard disk model [39].

The authors of [53] propose a technique based on Online Random Forests. Their method reaches an accuracy (defined as the fraction of true positives and true negatives in the whole population) above 93% while maintaining a low rate of false positives (i.e., failure predicted for an actually healthy disk). To achieve this result, this technique considers hard disk metrics over the period of one week. It predicts a disk failure in case any of the smart metrics in the period is positive and is hence sensitive to transient phenomena.

The authors of [55] propose a method to proactively predict disk errors before they cause more serious damages. Failure prediction is done by combining both SMART metrics and system-level signals. This method has the limit of not being applicable in all those contexts in which it is not possible to access system-level signals and therefore it would be inapplicable.

An interesting work of study and comparison of several recurrent neural models is done in this work. The authors have done an excellent study work and have proposed an interesting method as it returns information on the health conditions of hard drives. However, the proposed methodology does not solve 2 common problems in contexts of this kind. In fact, the authors do not suggest a method for the construction of the dataset on which to train automatic learning models, nor do they explain whether the proposed methodology is able to address the heterogeneity of hard disk models and the differences between sensor technologies[54].

Good results have been reached using an offline machine learning technique based on a decision forest named Regularized Greedy Forest (RGF) [8]. Only two cases of hard disk models were explored for which a recall (defined as the fraction of correctly detected failures among all positives) value as high as approximately 98% was achieved. The corresponding results for the false positives rate are not mentioned explicitly but can be derived as approximately 2% [8].

An interesting approach based on the Long Short Term Memory model has been proposed in [12] that achieves good results in terms of the false alarms rate. This paper does not present a labeling method for broken disks; thus, it is not clear how to create the dataset for the machine learning models. Moreover, it does not introduce a solution on how to deal with heterogeneous sets of hard drive models.

The authors of [41] focus on the wide-spread heterogeneity of data-centers. They addressed the issue of the manufacturer dependencies of the implementation technology, which the SMART monitoring system is integrated with. The authors compare Decision Trees, Neural Networks and Logistic Regression and suggest Decision Tree as the best solution for the problem at hand. The method proposed is able to predict about 52% of all hard disk failures among the truly failed drives.

2.3 Fault Detection and Failure Prediction in Injection Motors

In recent years, diagnostics of electric motors and health monitoring techniques have become a task of great relevance, as timely maintenance through early detection of irregularities during the normal machine, the operation can be achieved[60]. Due to the importance of this field, many methods have emerged to carry out diagnostics and predictive maintenance.

Lingxin Li and Chris Mechefske in the article *Induction motor fault detection & diagnosis using artificial neural networks* report a statistics about failure causes. About 50% of failures are caused by bearing failures, around 40% are due to winding failures of the stator, the remaining 10% is due to failures of the rotor or the shafts[29].

An interesting method for online diagnostics has been proposed in [9] where authors proposed the measurement of the frequency response in motor windings in order to perform detection of both mechanical and electrical damage (i.e.deformation of the windings and degradation of conductors and materials for isolation). This method is very interesting as it can be easily integrated into systems based on asynchronous motors and allows for real-time diagnostics. The fact that it doesn't interfere with normal engine operations is a big plus. Unfortunately, the proposed method presupposes a thorough knowledge of the type of engine being observed, consequently the methodology is difficult to implement on a large scale. Furthermore, the authors do not introduce a quantitative measure of the real ability of the proposed method to identify failures. Finally, the case studies are limited to cases of simulated and not real faults, there are not enough data to suggest a real applicability.

The methodology proposed in [26] addresses the problem of rotor manufacturing inaccuracies caused during the die casting process. Depending on the importance, this type of problem can present itself immediately or remain hidden until it manifests itself in critical moments of the use of motors. However, this method can only be used offline and is invasive as it is necessary to disassemble the motor.

A smart-sensor has been proposed in [60], it is based on low cost compact triaxial stray flux sensors, the setup is easy and is non-invasive. Despite this advantages, sensors need to be applied to specific locations on the motors, consequently an offline intervention on the motor need to be performed.

An interesting comparison between vibrations and current monitoring is presented in [16]. The authors pointed out that methods based on current and vibration analysis are the most widely used techniques for motor diagnostics. This is explained by the advantages of reliability, non-invasiveness, and ease of installation of the measuring sensors. The authors applied the Support Vector Machine algorithm to the different cases of failures and measured signals. With Support Vector Machine it is shown that mechanical failures are better identified by means of vibration signals and electrical failures are well identified by current

measurements. It is also pointed out that the accuracy of the diagnostics varies according to the engine speed.Although it is possible to achieve satisfactory accuracy in some contexts, a valid method is not proposed to diagnose every type of fault by means of current signals alone. Furthermore, the authors do not propose a method for the choice of measurement and diagnostic instruments. Finally, the applied method does not seem to be applicable in real time on an embedded device but rather requires an offline computer analysis.

A graph-based semi-supervised learning has been proposed in [59] in order to develop a comprehensive fault diagnosis method for an online diagnostics on induction motors. The proposed method is an approach based on semi-supervised learning which requires a smaller amount of labeled data. In particular, the authors adopted the greedy-gradient max cut algorithm (GGMC). The authors note that a large labeled dataset is required for supervised machine learning methods which is not always available in real settings. The method is interesting because it responds to the need to have a consistent training dataset since it is not always easy to collect failed engines for multiple reasons. However, the proposed method has been validated on the same engines used for training. Although the data on which the approach has been validated are not the same, there is no information on the validity of the approach in a real context. Finally, the authors do not specify the system requirements to be able to build a diagnostic system for a large-scale application.

3 Anomaly Detection in Fluid Machinery

3.1 Introduction

In this work, a fault detection method for fluid machinery without adequate a priori information about faulty conditions is proposed by exploiting HMM. The procedure takes into account multiple physical quantities and it deals with the sparsity and non-synchronization of these measures. The core of the proposed method is the ability to train a Hidden Markov Model by merely exploiting data related to normal machine operation. This extends its employability to reliable machines, where the fault is a rare, though deleterious, event. The proposed method, by taking into account multiple factors, can be generalized and extended to complex systems where various physical quantities can be monitored.

According to state of the art mentioned above, fault detection is a preliminary and essential step for fault diagnostics. Moreover, many applications simply require the detection of a fault, such as in the case study of the present work. Therefore, the aim of this work is to build a fault detection model, while further extension to diagnostics or prognostics will be addressed in future works.

The proposed fault detection method based on Hidden Markov Models, taking sparse measures of multiple physical quantities is sketched in Fig. 3.1. The problem of sparsity and non-synchronization of data is faced, first, by filling the input data on a coherent time-base and, then, by applying a decimation. Next, a Principal Component Analysis (PCA) is employed to project the data in a new space, and to select the principal components. Finally, data are clustered. After this data processing, the HMM is trained by means of the Baum-Welch algorithm. The trained model is then employed for fault detection, by means of a goodness–of–fit test, as shown in Fig. 3.1b.

In the following subsections, the blocks in Fig. 3.1a and Fig. 3.1b are detailed. Then, test and validation of the model are reported in the next sections, by referring to a case study at CERN.

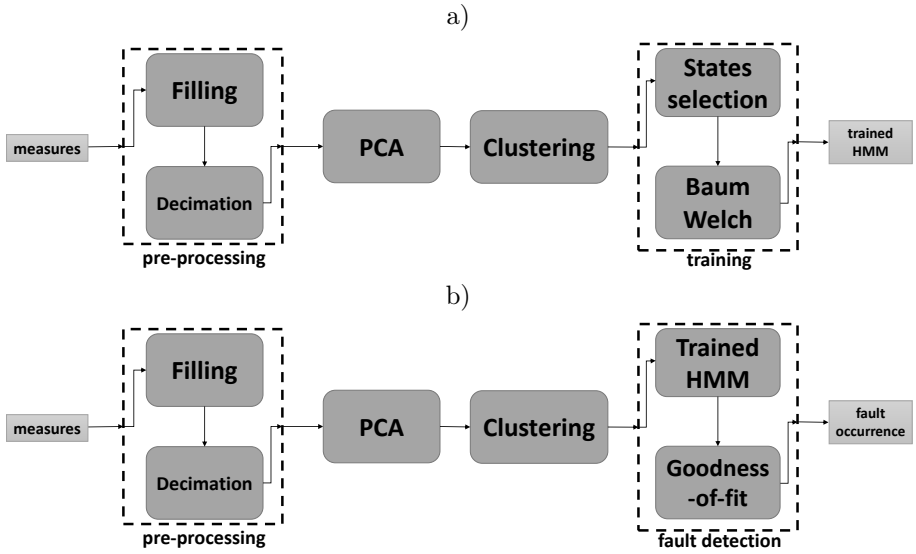

Figure 3.1: Block diagram of the proposed method for a) the Hidden Markov Model training and b) the fault detection (PCA: Principal Component Analysis).

3.1.1 Data processing

The first step in data processing is the *filling*. In general, sensors installed on the monitored fluid machine can return non-uniformly acquired data with data acquisition not synchronous. Indeed, the sampling rate depends on the expected variation of each physical measurand. However, for the next steps of training and fault detection, structured data are necessary. To this aim, a time base is considered with equally-spaced time instants, so that every measure can be referred to a reference instant. Then, the value of each quantity is replicated for each time instant up to the next measured value. It is assumed that a data is obtained in a stationary operating condition, namely, that measures are not acquired during transitions of the machine to a regime.

After that, data are *decimated* in order to diminish the computational burden and allow further processing. The optimal decimation factor is found balancing the trade-off between the computational burden and the final accuracy of the model. This factor does depend on the available data, and a determination example is reported in the following section within the discussion of the case study.

As mentioned above, the *Principal Component Analysis (PCA)* follows. The purpose in using PCA is to represent the variation of measured quantities with a smaller number of "factors", or "main components". The samples are represented in a new space, by redefining the axes through the main components instead of the original variables. PCA captures the variance of data, and accounts for cor-

relations among variables. The resulting lower-dimensional representations of data can improve the proficiency of detecting and diagnosing faults using multivariate statistics. The number of principal components to take into account depends again on the available data. The proposed criterion is that the first discarded component would add an amount of data variance which is two-order of magnitude less than the variance already taken into account with the selected (principal) components (lower than 1%). This step can be executed with the MATLAB function *pca* by deriving the scores associated to each component.

Data processing is concluded with a *cluster analysis*. Its purpose is to group data according to some criteria of similarity, As an example, the clusters are determined in such a way that the observations are as homogeneous as possible within the same cluster, and as uneven as possible between the different clusters. The clustering technique employed in the proposed method exploits the Euclidean metric. According to that, the data are clustered so that the Euclidean intra-cluster distance is minimized, and the extra-cluster Euclidean distance is maximized.For instance, this can be done with the *kmeans* function available in MATLAB, by exploiting the default distance measure *sqeuclidean*. The success of this modeling depends on the choice of the number of clusters. Too many clusters would lead to ambiguities, while choosing few clusters means flattening observations and merging observations that are actually associated to different states. The number of clusters is chosen with an iterative procedure aiming to find the combination of clusters and states of the Markov model that maximized the accuracy of the trained model. This is better described below in discussing the training algorithm.

3.1.2 Training

An HMM is usually characterized by five elements:

1. **N**, the number of states;
2. **M**, the number of distinct observation symbols per state;
3. **A**, the state transition probability distribution;
4. **B**, the observation symbol probability distribution;
5. π an initial state distribution.

Training refers to the process of obtaining the HMM parameter set. In particular, the matrices **A** and **B** are to find. Each element a_{ij} of the matrix **A** represents the probability of transition from the state i to the state j. Instead, each element b_{ij} of the matrix **B** represents the probability to observe the symbol i given a state j. The training of an HMM consists in finding these two matrices from measured data. The dimension of the matrix **A**, as well as one dimension of matrix **B**, is related to the number of hidden states **N**, which is not known a-priori. Thus, an iterative procedure is employed in order to find the number of states that maximizes a likelihood function. The iterative algorithm that constitutes the core of training in the proposed method is the “Baum-Welch algorithm”.

Let $\theta = (A, B, \pi)$ be the model to train. The algorithm is made of three steps: (i) a forward procedure, (ii) a backward procedure, and (iii) an update procedure. The forward procedure calculates the probability of a partial sequence of states α_i, finishing in state i, given a model θ. To this aim, an iterative procedure is implemented (eq.3.3). The probability of observing a symbol $y(t+1)$ in state i is multiplied by the probability of entering the state i.

$$\alpha_i(t) = P(Y_1 = y_1, \ldots, Y_t = y_t, S_t = i|\theta) \tag{3.1}$$

The iterative procedure is initialized with $\alpha_i(1)$ equal to the product between the probability of observing the symbol y_1 in state i and the probability that the initial state is i (namely π_i) (eq. 3.2).

$$\alpha_i(1) = \pi_i b_i(y_1) \tag{3.2}$$

Eq. 3.1 resumes this first step, where Y is the random variable representing the observed measure, and S is the random variable representing the state.

$$\alpha_i(t+1) = b_i(y_{t+1}) \sum_{j=1}^{n} \alpha_j(t) a_{ij} \tag{3.3}$$

On the contrary, the backward procedure corresponds to find the probability (eq.3.4) of generating the sub-sequence from $t+1$ to T after entering the state i.

$$\beta_i(t) = P(Y_{t+1} = y_{t+1}, \ldots, Y_T = y_T, S_t = i|\theta) \tag{3.4}$$

The iteration now considers the sum of all probabilities of entering the states after i and observe the symbol $y(t+1)$ multiplied by the probability of having the sub-sequence β_j. Eq. 3.6 resumes this second step. The iteration is initialized considering that this probability is 1 (eq. 3.5) in the last observation T, i.e. $\beta_i(T) = 1$.

$$\beta_i(T) = 1 \tag{3.5}$$

Hence, the initialization here considers the final observation, while the rest is obtained going backward.

$$\beta_i(t) = \sum_{j=1}^{N} \beta_j(t+1) a_{ij} b_j(y_{t+1}) \tag{3.6}$$

Finally, it is possible to calculate the probability γ of being in state i at time t given the observed sequence Y and the parameters of model θ, and the probability ξ of being in state j at time t and $t+1$ given the observations Y and the parameters of θ.

$$\begin{aligned} \gamma_i(t) &= \frac{\alpha_i(t)\beta_i(t)}{\sum_{j=1}^{N} \alpha_j(t)\beta_j(t)} \\ \xi_{ij}(t) &= \frac{\alpha_i(t) a_{ij} \beta_j(t+1) b_j(y_{t+1})}{\sum_{j=1}^{N} \sum_{j=1}^{N} \alpha_i(t) a_{ij} \beta_j(t+1) b_j(y_{t+1})} \end{aligned} \tag{3.7}$$

This allows to update the model parameters because π_i equals $\gamma(1)$, namely the probability of being in state i equals the initial state probability, the matrix $\mathbf{A}$ is updated with the ratio between the expected number of transitions from state i to state j and the number of transitions from state i to any other state, while the matrix $\mathbf{B}$ is updated with the ration between the number of times that the output is k when the state is i divided by the total number of times that the state is i. This update procedure is resumed in eq. 3.7 and 3.8.

$$\begin{aligned} \pi_i^* &= \gamma_i(1) \\ a_{ij}^* &= \frac{\sum_{t=1}^{T-1}}{\xi_{ij}(t)} \sum_{t=1}^{T-1} \gamma_i(t) \\ b_i^*(v_k) &= \frac{\sum_{t=1}^{T} 1_{y_t=v_k} \gamma_i(t)}{\sum_{t=1}^{T} \gamma_i(t)} \end{aligned} \tag{3.8}$$

where $1_{y_t=v_k}$ equals "1" when $y_t = v_k$ and "0" otherwise.

3.1.3 Fault detection

Once the model is trained, it can be employed for the fault detection. The data processing steps necessary in this phase and reported in Fig. 3.1b comply with the corresponding ones of the training phase, already detailed in the previous section. Then, the clusters are compared to the model and the likelihood function is returned. The maximum likelihood is found with an Expectation–Maximization (EM) algorithm, which estimates the parameters of the statistical model [1]. Finally, a goodness–of–fit test is executed between the likelihood sample obtained in the training phase and that obtained by the application of the test sequence. A fault is detected when a different distribution is found for the two samples. An example of goodness–of–fit test is the Wilcoxon one [20]. It is a non-parametric test employed in case of non-Gaussian distribution of data. Alternative statistical tests can also be used. As an example, the likelihood distribution could be analyzed with a preliminary statistical test and if a compliance is found with a Gaussian distribution, specific tests can be used, such as the chi square test.

In the following, the training and validation steps described above are resumed in an algorithmic form in order to clarify the method implementation.

3.2 Experimental Case Study

The methodology presented above was validated on a case study at CERN. Failure prediction techniques can address maintenance planning needs. This is especially useful in contexts such as those of CERN where an attempt is made to condense maintenance activities to certain periods of the year.

Algorithm 1 Fault Detection with Hidden Markov Models: Training

Collection of sequences S_i in the data set S;
procedure PRE-PROCESSING (S) ▷ $\forall\ S_i \in S$
    `Filtering`: discard all sequences out of ON period;
    `Filling`: replication of the last available measurement;
    `Decimation`: selecting 1 over N observations, with N the decimation factor;
    `PCA`: Extraction of the principal components
    return Pre-processed Sequences: S^1
end procedure
procedure TRAINING OF MODEL(80% of S^1) ▷ multiple training needed
    for all $S_i \in S^1$ **do**
        `Clustering`: $(S^1_{c_i}) \leftarrow EuclideanClusterization(S^1)$
        `Training`: $(M_i, L_i) \leftarrow BaumWelch(S^1_{c_i}, A)$ ▷ A states number
    end for
    return (M_i) whose (L_i) is Maximum
end procedure
procedure TESTING OF MODEL(M_i, 20% of S^1)
    `Check` $(L_i) \leftarrow ExpectationMaximization(M_i, S^1_{c_i})$
    return number of sequences rejected
end procedure

Algorithm 2 Fault Detection with Hidden Markov Models: Validation

procedure VALIDATION(M_i, 20% of S^1)
    `Anomaly injection`: Alteration of few values in the nominal observations $S^a \leftarrow$ Alteration(S^1)
    $S^{a_1} \leftarrow$ *Pre-Processing*(S^a)
    $L^a \leftarrow ExpectationMaximization(M_i, S_i^{a_1})\ \forall\ S_i^{a_1} \in S^{a_1}$
    `Validation`: $P_w \leftarrow Wilcoxon(L_i^a, L_j^a)$ with $L_i^a, L_j^a \in L^a$ and $i \neq j$
    return P_w
end procedure

3.2.1 Description and test strategy

At CERN, liquid helium is used to cool superconducting magnets that accelerate the particle beam in the Large Hadron Collider (LHC). Moreover, within the High-Luminosity LHC upgrade, superconducting links are under construction, and this requires an important economic investment also in the cryogenic plants [5]. To process liquid helium, screw compressors are employed (Fig. 3.2).

Figure 3.2: Screw Compressor Cutaway.

A complex refrigeration system based on this kind of compressors guarantees the distribution of liquid helium in the accelerator. In such a system (Fig. 3.3), it is essential to prevent malfunctions of screw compressors, especially when the accelerator is in operation. Even though a fault is a rare event because of the reliability of these machines, it still poses a challenge for long accelerator runs.

As mentioned above, though the case study takes into account screw compressors, the proposed method was conceived for fluid machines in general. The involved quantities are typical for fluid machines, namely 3-axis acceleration, power consumption, temperature, inlet pressure, and outlet pressure. In Fig. 3.4, the physical quantities that compose the data set taken into account are plot for the sake of clarity.

To verify the validity of the method, the model was built for compressor 7 in cryogenic system QSCA installed at the point 2 of LHC. To test the algorithm, it was necessary to conduct a series of experiments based on the administration of well-known nominal and abnormal sequences, in order to verify if the algorithm correctly recognizes the incoming sequence. The basic idea was to verify how many anomalous and nominal sequences were correctly recognized and how many were not. Data were divided into 80% employed for training, and 20% employed for a preliminary evaluation of the model accuracy. A further validation step followed. However, since the whole data set contained measures acquired during nominal operation, faulty data were obtained by artificially corrupting the original data and these were given as input to the algorithm to verify if they are properly recognized. In simulating the faulty data, it was considered that, as a result of a fault, a significant increase in the vibrations is expected. Screw compressors are complex machines, composed of several components, such as gear-box, rotors, bearings, each of them subject to faults. It is very difficult

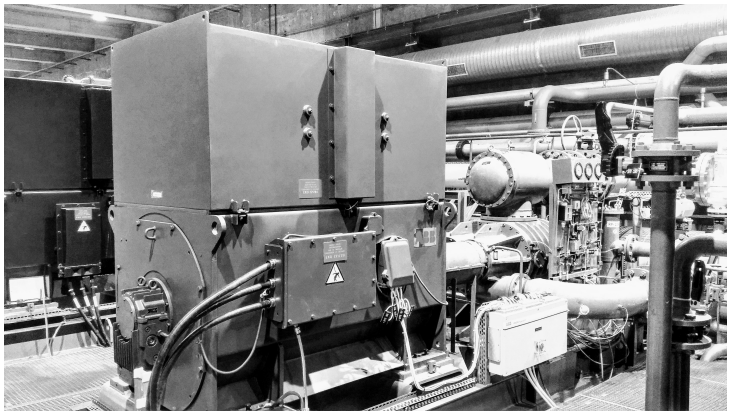

Figure 3.3: Screw Compressor N.7 in LHCA-P2 CERN.

to model the system as a whole, and, therefore, to develop a model that simulates a faulty condition. However, historical fault data indicate that in many faults or anomalous conditions, a significant vibration increase was observed before the machine failure [44]. Often, a fault of a subsystem (such as a scuffed rotor tooth [15] or hydrodynamic unbalance [61]), although causing a significant vibration increase, and probably a consequent variation of the other process quantities, do not cause an immediate failure of the machine. However, if suddenly discovered, it can be fixed without causing unrepairable damages to the machine. Therefore, seven parameters, namely the vibration velocity on the three axes, the inlet and outlet pressure, the outlet temperature, the delta oil pressure and the position of the slide valve, were selected and collected in sequences of 24 hours each one. Anomalous conditions were simulated by altering the vibration velocity on the axis Z in each sequence of observation for about the 15% of its duration. The amount of the alteration was varied in the set {20%, 30%,50%,100%, 150%} of the original value. The results of these experiments are reported at the end of this section. Unfortunately, it is not possible to perform controlled experiments on the compressors considered in the test case, because they are installed and in operation. However, the simulated amount of vibration was chosen coherently with the documented faults for the screw compressors.

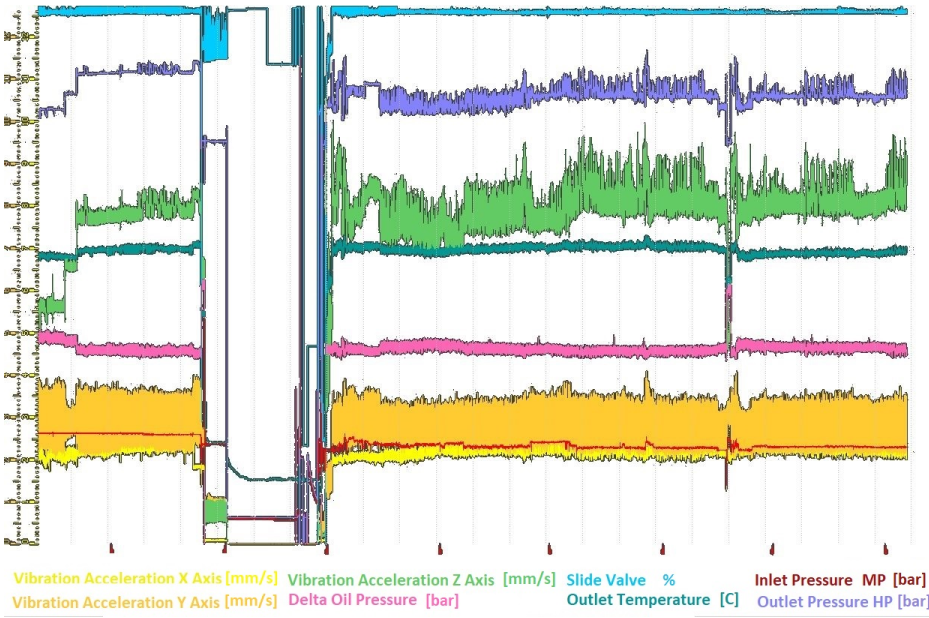

Figure 3.4: Trends of physical quantities measured over a period of 3 months. Light Yellow:Vibration X Axis, Dark Yellow:Vibration Y Axis, Light Green:Vibration Z Axis, Pink:Delta Oil Pressure, Light Blue:Slide Valve, Dark green: Outlet Temperature, Red:Inlet Pressure MP, Violet:Outlet Pressure HP.

3.2.2 Data processing

As already described in the proposal, data were processed with filling, decimation, principal component analysis (PCA), and clustering. In the study, the first 3 main components were taken into account when executing the PCA. They were able to explain 99.93% of data variance. The new dataset, obtained from the projection of the measurements on the new orthonormal base, was then composed by only 3 main features.

Euclidean distances were hence calculated between points in a 3D space for clustering. The centroids for the number of desired clusters were found with the *kmeans* function available in MATLAB. The number of clusters was chosen with an iterative procedure, aimed at finding the combination of states and clusters that maximized the quality of the trained Markov model. Quality of model is defined in terms of likelihood probability carried out at the end of training.

The system is to be modeled in its state transitions, therefore the same length of the sequence and the same distribution of the observations have to be maintained. Thus, the sequences were reconstructed by associating the centroid of the corresponding cluster to each observation. Each data point was assigned to

the cluster from which the Euclidean distance was minimum. Then, for each sequence, a vector of the same size was constituted.

3.2.3 Training

Using the iterative algorithm of Baum–Welch, described in the previous Section, the HMM was trained exploiting 80% of processed data. In order to achieve the best combination of states and clusters, a number of states ranging from 2 to 7 was tested. The best result was obtained with a configuration of 3 states and 3 clusters. The results of this procedure are plotted in Fig. 3.5 versus the iteration of the algorithm. It is shown that the likelihood increases quickly in the first part, and then it reaches a steady state as no further significant improvement is found. Although the maximum number of iterations was set to 100, less than 50 were enough to reach the steady-state value.

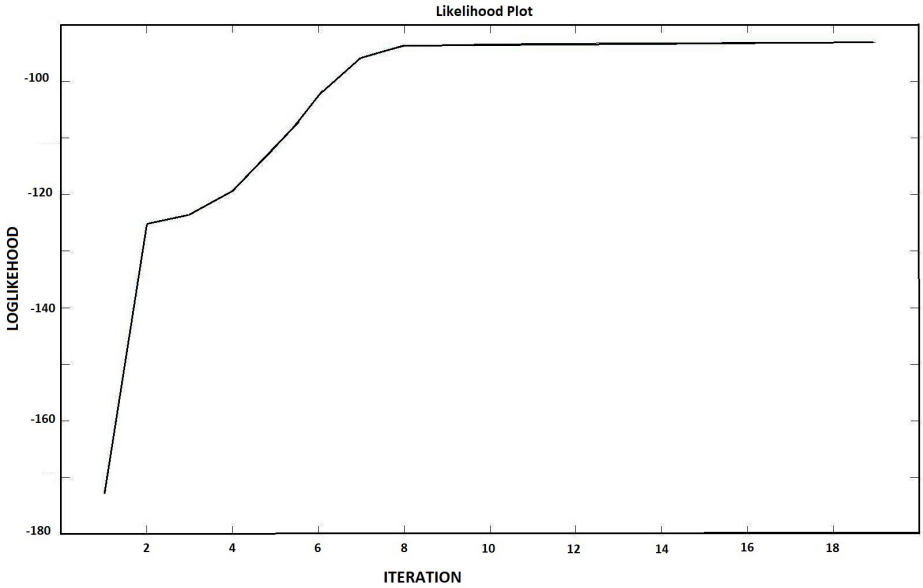

Figure 3.5: Model Log-likelihood per iteration.

The resulting model is synthesized in Fig. 3.6, which graphically represents the matrices **A** and **B**, describing the Markov model. In the present case study, a totally connected graph with 3 nodes was obtained. In each state, it is possible to obtain one of the outputs, previously clustered into 3 classes. For each state, the probability of issuing one of the possible outputs and the probability of passing in another of the states or remaining in the same state are reported, respectively.

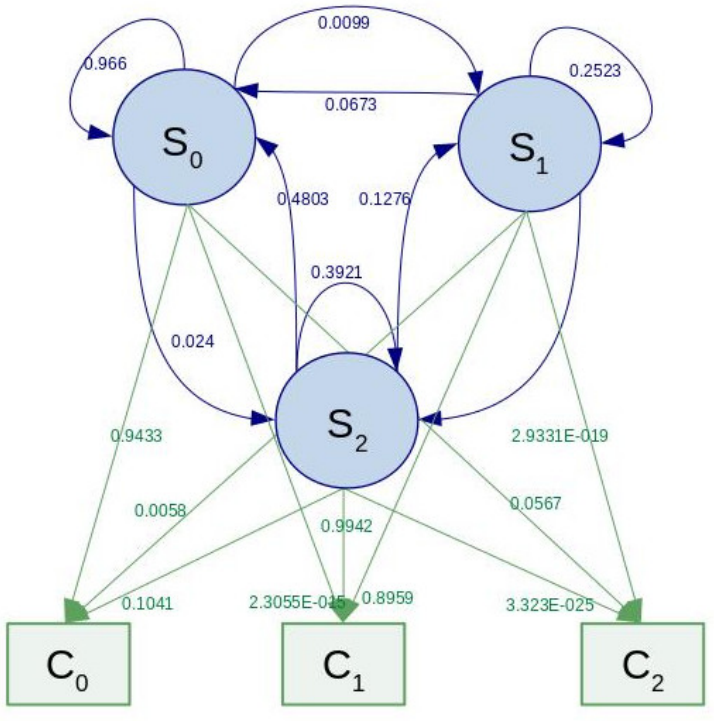

Figure 3.6: Trained Hidden Markov model.

3.2.4 Validation

Once the model was built, it was necessary to move on to the fault detection phase. Part of the measured data was manipulated to create the test set. For this reason, samples are dependent. The Wilcoxon test was employed to assess the goodness–of–fit because data distributions resulted non-Gaussian by executing a Shapiro–Wilk test and were supposed to be dependent. Referring to Fig. 3.1b, it is clear that the testing data is processed, prior to the statistical test, following the same processing of the training data up to the PCA and clustering. The algorithm carried on this processing with the same number of principal components and clusters already identified in the train phase, which both equal 3, as already discussed. Then, clustered data enters the trained model, and the log-likelihood is calculated to verify if each sequence of data fits the model.

The distribution of likelihoods has been collected for each test sequence and Wilcoxon Test was performed on those distributions. The results of the validation procedure are reported in Tab. 3.1. The first column reports the percentage increment adopted to corrupt test data and create anomalous sequences. The

Increment	Probability
0%	0.9283
20%	0.0375
30%	0.0243
50%	0.0202
100%	0.0160
150%	0.0188

Table 3.1: Wilcoxon test results for validation.

second column, instead, reports the Wilcoxon test results performed on the distributions of likelihood vales collected. Wilcoxon test null hypothesis is "the two populations have the same continuous distribution". The null hypnosis is confirmed or rejected with a probability value, and the confidence level associated to the statistical test is 95%. The test confirms that the dissimilarity is statistically significant in the presence of alteration. This means that the proposed methodology provides statistically different likelihood values in conditions of minimal alterations on the sequences of observations ergo it is able to recognize small differences on nominal conditions.

3.3 Remarks and Discussion

In this work, a method for fault detection based on a Hidden Markov model (HMM) has been treated. The method was conceived for fluid machines, and the results of the application to screw compressors have been reported. In particular, the method was validated experimentally on a case study at CERN with reference to a cryogenic plant.

Seven physical quantities have been taken into account. They constitute a dataset built with several months of continuous acquisition. However, the measures were sparse and asynchronous, so that a filling was employed at the beginning of data processing. The method then carries on Principal Component Analysis and clustering. The clusters are employed to train the HMM in the first phase, or for a fault detection exploiting the trained HMM.

A testing procedure and a validation procedure with artificially altered data were built to assess the capability of the method to detect a fault. A Wilcoxon test was performed in order to assess the adherence of data sequences to the trained model. Results point out that every sample of likelihood calculated from altered sequences is statistically different from the likelihood sample calculated on nominal sequences. Future work will be devoted to considering another machinery facility, by performing some tests characterized by several 'damage levels', in order to further explore the algorithm performance.

4 Failure Prediction in Hard Disks

4.1 Introduction

The new frontiers of industrial production and scientific research are reflected in a growing demand for computing and, consequently, in an evolution of storage technologies. The increased demand for data storage and processing pushes the data-centers to provide increasingly efficient services [48]. This efficiency requirement can be compromised by the devices that underpin storage: hard drives. Magnetic hard disks are currently among the most widely used and devices for storing data.

Magnetic hard disks are currently among the most widely used and among the most frequently failing components of storage systems and disk failures, resulting in about 78% of cloud server system failures [49].

In addition to the impact of a service outage for the user, these events also result in significant costs for the service provider. As unscheduled interventions, they may require rapid (and hence costly) human intervention or a resource intensive consistency check and re-validation of recent processing steps by a combination of the service provider and user. In the worst case, disk failures can lead to permanent data loss and hence economical damage to the user and service provider due to a breach of service level agreements [34, 49, 51]. While fault tolerance techniques can reduce the risk of loss, they also often impact negatively on system performance or price per usable volume.

In order to facilitate the automated interpretation of the disk operational status, the disk vendors introduced Self-Monitoring, Analysis and Reporting technology (SMART), a standardized monitoring system with the goal to warn about an increasing likelihood of disk failures. The SMART subsystem is a firmware component running on the disk processor and continuously collects operational metrics in order to generate a warning or error messages before disk failures that affect the integrity of application data.

The characteristics of SMART technology include a number of attributes for diagnostics, the SMART attribute implementation technology varies from model

to model, even for different disk models of the same manufacturer [42].

The goal of the SMART system is to warn a user at least 24 h before device failure [23]. Each manufacturing company implements SMART sensors according to their proprietary technology choices and sets alarm thresholds with the aim of an acceptable rate of false-alarms and its potential impact on drive warranty obligations. Generally, the false-alarm rate is around 0.1% per year [32, 39].

Several methods based on modeling techniques have been proposed to recognize hard drives that need to be replaced using SMART attributes [39, 17, 53, 41], but none of them meet all the needs of adapting to heterogeneous sets of hard drives, low false positive and false negative rates, automatic preliminary dataset labeling.

To overcome the limitations considered before, an inference method to retrieve information about failed disks and a supervised machine learning approach to build a predictor of near-failure disks is proposed in this work. The method presented in this work does not need a human intervention for dataset labeling and is capable of operating with heterogeneous sets of magnetic hard disks.

The proposal has been validated in a case study at the European Organization for Nuclear Research (CERN), where the High Energy Physics (HEP) challenges have led storage services (fig. 4.1) to become a crucial part of the cloud services [13]

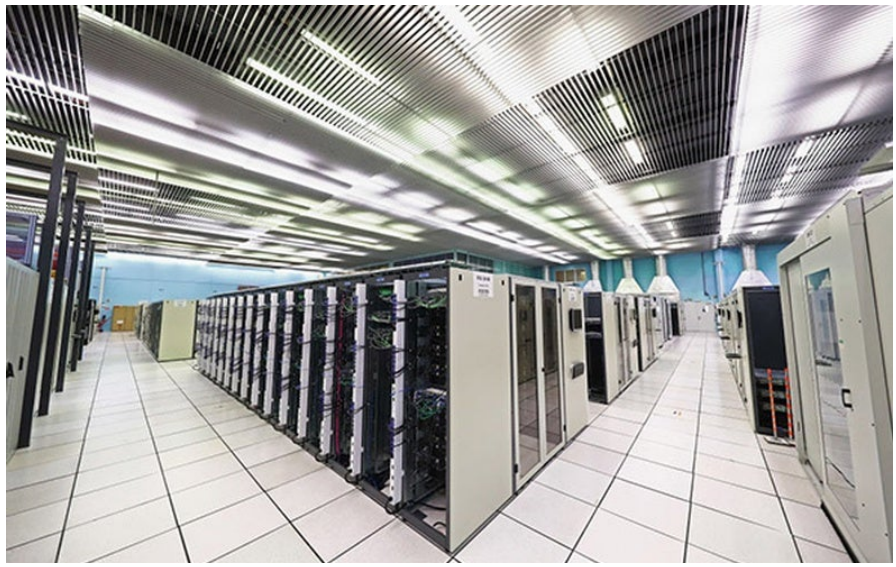

Figure 4.1: Cern Data Center

CERN uses a variety of hard disk models [14], and an effective prediction of a likely drive malfunction would allow increasing the user's perceived service quality and, at the same time, decrease the resources to operate the service.

4.2 Regularized Greedy Forest Model

The Regularized Greedy Forest model proposed in [25] is a variant of a Gradient Boosted Decision Tree with an explicit regularization (introduced with a regularization term in the loss function) and with a repeated full-correction of the coefficients (a repeated optimization). The RGF model and the algorithm for its training are briefly described to clarify the steps that follow in the paper. The RGF is a forest (fig. 4.2), or more formally, a non-linear function class

$$h : X \longrightarrow \{0, 1\} \quad , \tag{4.1}$$

where X is a set of input vector

$$\boldsymbol{x} = [x[1], x[2], \ldots, x[d]] \in \mathbb{R}^d \quad . \tag{4.2}$$

The vector $y \in \{0, 1\}$ is a response vector used for the supervised learning. Each node is a non-linear decision rule ν associated with the pair (β_ν, α_ν). Here, β_ν refers to a basis function and α_ν to the weight assigned to the internal node ν. The model of the forest is

$$h_F(\boldsymbol{x}) = \sum_{\nu \in F} \alpha_\nu \beta_\nu(\boldsymbol{x}) \quad , \tag{4.3}$$

where $\alpha_\nu = 0$ for any internal node. This forest model is accompanied with a loss function $L(h(x); y)$ and a regularization term $R(h)$. The RGF goal is to find a non-linear function $\hat{h(\boldsymbol{x})}$ from a function class H that minimizes the following risk function:

$$\hat{h} = argmin_{h \in H}[L(h(x), y) + R(h)] \quad . \tag{4.4}$$

The algorithm greedily selects the basis functions and optimizes the weights. The two structural operations managing the forest are (1) the splitting of nodes and (2) the introduction of additional trees. At the k-th iteration, the algorithm, in modifying the structure, operates one of the two options in order to reduce the regularized loss functions defined as:

$$Q(F) = L(h_F(X), Y) + R(h_F) \quad . \tag{4.5}$$

The addition of a new node (fig. 4.3), from which to start a new tree, consists of the sum of the node pair to the existing tree, formally:

$$h_{F_k}(\boldsymbol{x}) = h_{F_{k-1}}(\boldsymbol{x}) + \alpha\beta(\boldsymbol{x}) \quad . \tag{4.6}$$

In figure 4.3 the possible changes to the forest (fig. 4.2) are shown. In particular, the example of creating a new tree in the forest or splitting a node is schematized. The splitting of a node (fig. 4.3) consists of the removal of the node that must be divided and the addition of the two new nodes. Assuming a generic node associated with (β, α), (β, α) needs to be split into (β_1, α_1) and

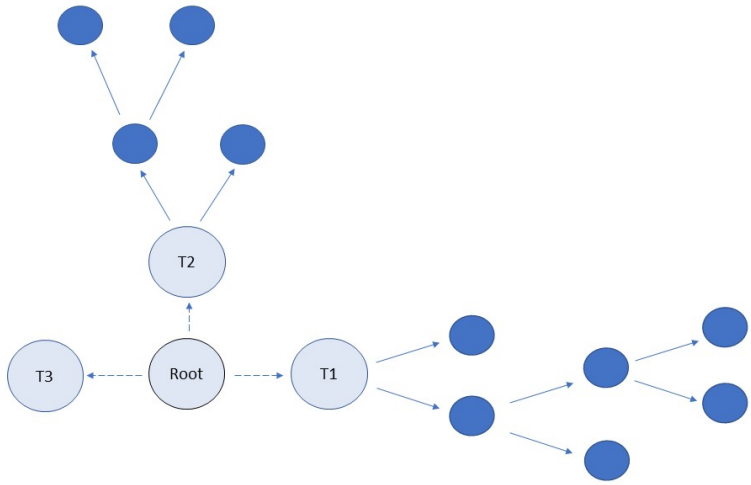

Figure 4.2: RGF Schema

(β_2, α_2) because it is the node split that minimize the loss function, the forest h at the iteration k becomes:

$$h_{F_k}(\boldsymbol{x}) = h_{F_{k-1}}(\boldsymbol{x}) - \alpha\beta(\boldsymbol{x}) + \alpha_1\beta_1(\boldsymbol{x}) + \alpha_2\beta_2(\boldsymbol{x}) \quad . \tag{4.7}$$

After a series of changes in the structure, the algorithm proceeds to an optimization of the weights. The algorithm then fixes the structure and adjusts the weights by evaluating all possible combinations in order to minimize the loss function. Carrying out a frequent optimization of the weights affects the calculation times.

To make RGF work with success, several parameters involved in the model training have to be suitably tuned. Three different versions of the *Loss Function* $L(h_F(X), Y)$ can be exploited:

- the Square Loss $LS := (f(x) - y)^2/2$;
- the Exponential Loss $Expo := e^{(yf(x))}$;
- the Logistic Loss $Log := \log(1 + e^{(yf(x))})$ function .

The regularization term $R(h_F)$ can assume three different models[25]. In all of the three cases, the coefficient λ weighs the importance for regularization. The first regularizer is the L_2 *Regularization on Leaf-only Models* defined as

$$R(h_T) = \lambda \cdot \sum_{v \in T} \alpha_v^2/2 \quad . \tag{4.8}$$

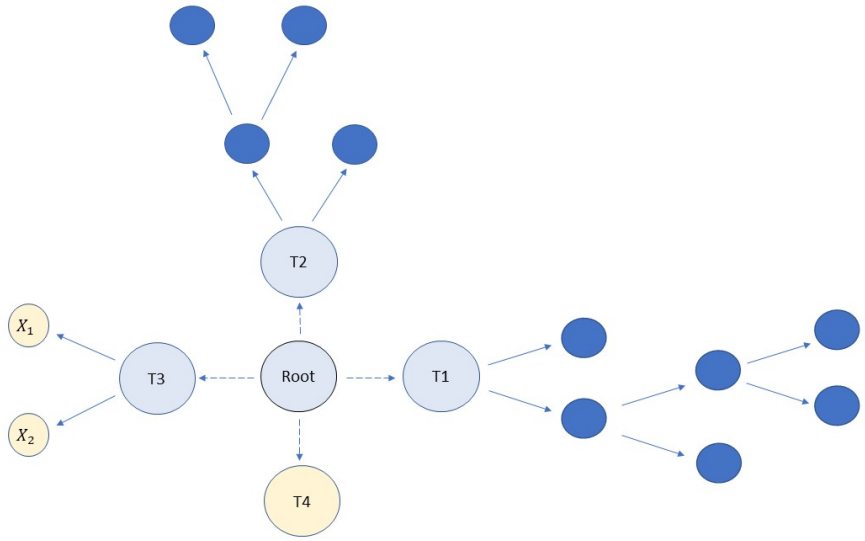

Figure 4.3: RGF Schema Structure Changes

The second regularizer is the *Minimum-Penalty Regularizer* defined as

$$R(h_T) = \lambda \cdot \{ \sum_{\nu \in T} \gamma^{d_\nu} \beta_\nu^2 / 2 : h_{T(\beta)}(\boldsymbol{x}) \equiv h_T(\boldsymbol{x}) \} \quad . \tag{4.9}$$

The third regularizer is similar to the second, but the condition changes: the sum of the sibling pair need to be zero. The third regularizer is the *Minimum-Penalty Regularizer with Sum-to-zero Sibling Constraints*; the regularizer is

$$R(h_T) = \lambda \cdot \{ \sum_{\nu \in T} \gamma^{d_\nu} \beta_\nu^2 / 2 : h_{T(\beta)}(\boldsymbol{x}) \equiv h_T(\boldsymbol{x}) \; ; \forall \nu \notin L_T \cdot [\sum_{p(\omega)=\nu} \beta_\omega = 0] \} \quad . \tag{4.10}$$

4.3 Proposed Approach

The novel method presented here is for inferencing failed hard disks (Algorithm 3) and to predict when a drive needs to be replaced due to impending failure (Algorithm 4; the method addresses the issue of the hard disk models' heterogeneity of which the storage systems are generally characterized.)[18].

From the collected data, a first step (fig.4.4) of knowledge extraction is made to label the disks that have been replaced due to a failure and distinguish them from all other disks (replaced or not). After that, a cleaning of the inconsistent data is carried out, then, hyperparameters tuning, training and testing phases of an RGF model are performed consequently (fig. 4.5).

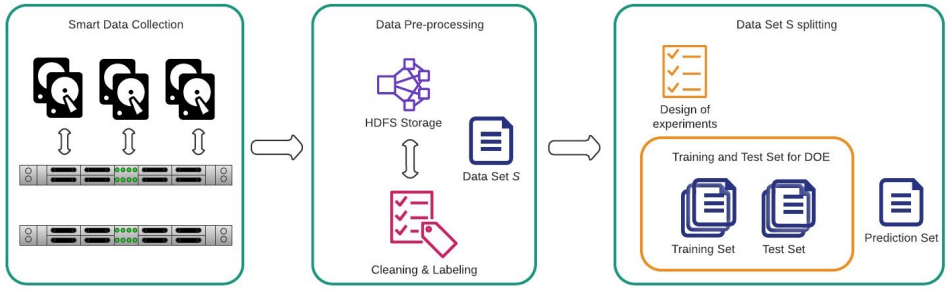

Figure 4.4: Data Collection and pre-processing phases.

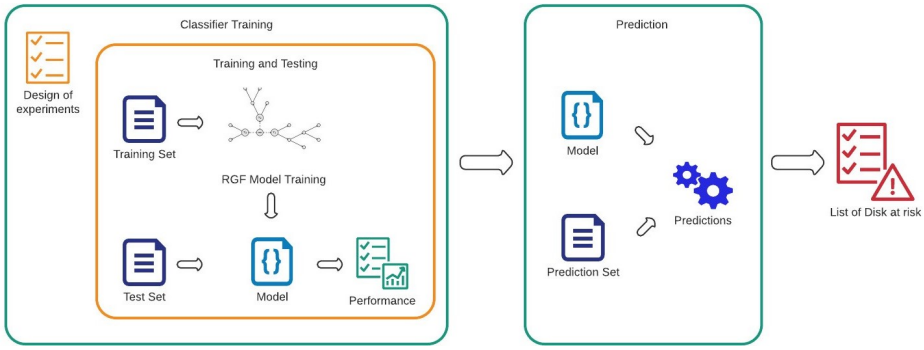

Figure 4.5: DOE, modeling and prediction phases.

4.3.1 Automatized Labeling Stage

Labeling Training Data

In many environments, including the CERN storage deployment, a precise definition or label for "failed" disks is not a priori available but needs be derived from other existing information. At CERN, a probe has been put in place to determine, several times per day, the presence of all disk drives (identified by their serial numbers) in all server or compute nodes. Changes between these configuration snapshots are used to track interventions by the data center operations team, which usually take place for one of the following reasons: A disk is:

- removed/disabled after repeated I/O or self-test errors;
- retired/replaced at the end of its planned lifetime;
- relocated within the data center(s) (e.g., due to floorspace reorganization).

In Figure 4.6, the hard disk d_2 is an example of temporary suspension during the interval $I_g = [t_{g_1}; t_{g_2}]$. A typical scenario of a host decommissioning is also shown in the Figure 4.6. All drives belonging to a host that has to be shut down are drained and removed. Due to the high disk reliability, it is quite unlikely that

more than one disk would fail at the same day on the same host. Thus, whenever multiple shutdowns are observed (e.g., d_1, d_2, d_3, d_4) in a small day interval, it is necessary not to classify these disks as replaced disks due to a failure.

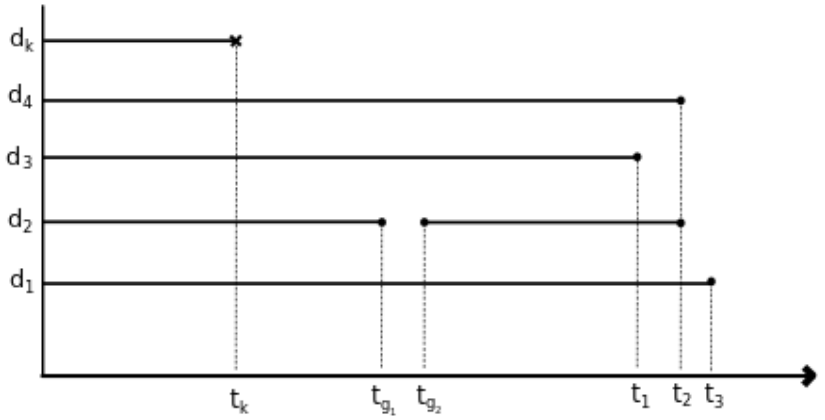

Figure 4.6: Hard disks' lifetimes in a decommissioned host at time t_3.

In Fig.4.7 it is presented a real case, in which the number of disks measured each day is reported; the figure well illustrates a typical scenario where there is an inconsistency between the number of disks that a probe measures daily and the number of disks actually running. The obtained outliers are a consequence of multiple occurrences of gaps.

As there is no available label classifying a disk as broken, we propose a heuristic approach based on the following definition for "disks at risk", which is aimed to identify disk replacements due to individual device failures from the larger amount of disk replacements due to other data center operations (e.g., host or disk model replacements or retirement campaigns).

Definition. *A disk is classified as broken if it has been removed and all other hard disk belonging to the same host machine continue to operate nominally.*

In other words, replaced hard disks are classified as broken if and only if neither if other HDDs are removed in the same day nor if the host machine is decommissioned within 30 days (e.g., d_k at t_k instant in Figure 4.6).

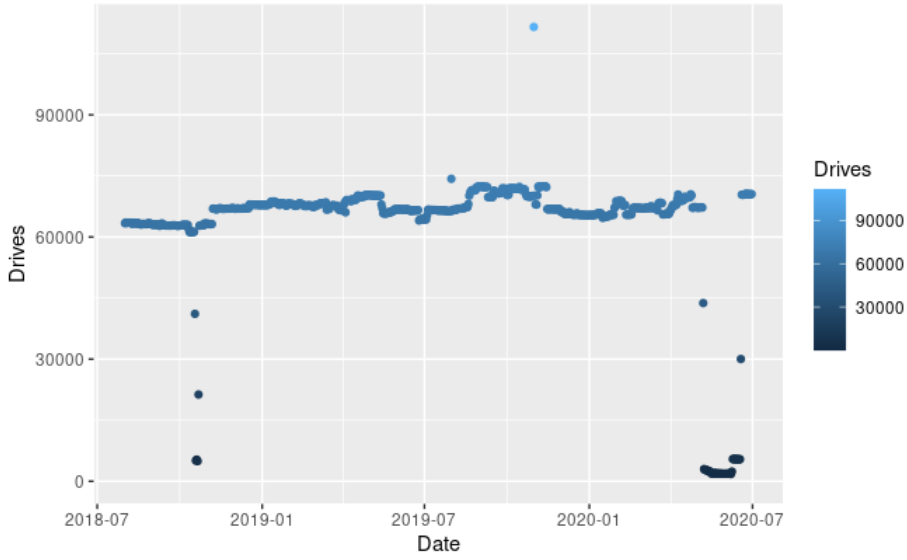

Figure 4.7: Number of disks over days

SMART Data

The number of hard disks required to perform good training must be reasonably large. Hard drives fail even after several years from their installation, which leads to a small number of failures per year in a storage system. To obtain sufficient SMART tuples (a complete set of SMART measures defining the health status of a disk) to train and test the proposed machine learning method, some tens of replaced hard disks have to be taken into account; this way, the SMART tuples of all hard disks must be necessarily collected in a database daily. The collected data must be accessible in order to be evaluated in their temporal sequence, and each SMART attribute tuple must be associated with the serial number, model, host machine and timestamp of the drive. The job that collects and stores this information must be run daily.

Pre-Processing

To suitably train the proposed method, all incorrect measures associated with errors, missing values, exceeding values, etc., must be preliminarily removed. The collected measurements may be corrupted as errors caused by the probe, in the internal network, in the file system, etc., may occur. Another typical example of a corrupt measurement can be an out-of-range temperature measurement, which can happen because the temperature sensor of a hard disk broke, but this has not affected the correct functioning of the storage device. Moreover, a disk can be turned off for several days before being switched on again. Disks disappeared

from the list of monitored devices and belonging to multiple removals (due to maintenance, dismantling of hosts machine or similar) have to be excluded from the set labeled as broken units. In addition, entire measures may be missing in some days because of software errors, busy firmware, database problems, etc.

An example of the considered conditions are schematically presented in Figure 4.6; in particular, the hard disk d_2 (more specifically, its SMART tuples) are not present during a time interval, a gap.

Let us define S as the whole dataset of available hard disks in the storage system, and let $I_{\text{g}} := [t_{g_1}; t_{g_2}]$, with $t_{g_2} > t_{g_1} + 1$ be a generic days gap experienced during SMART tuples acquisitions. This way, the subset $S_r \in I_{\text{g}}$ of hard disks replaced during the gap has to be removed from the dataset because no assumptions can be made about their possible failure and the classification rule for broken disks is not applicable; a specific hard disk is dropped from the list of monitored devices since its host machine has been completely removed, and hard disks possibly operating have also been dismissed. On the other hand, disks normally running after the gaps are kept in the dataset. Disks replaced during an interval $S_{r,k} \in I_{\text{g+k}} := [t_{g_2}; t_{g_2+k}]$ need to be excluded as well because a replacement $\hat{S}_{r,k}$ that happened in the interval $I_{\text{g+k}}$ does not have the k days of measures needed for a consistent train dataset. Thus, the dataset becomes $S_d = S - \{S_r \in I_{\text{g}}, I_{\text{g+k}}\}$.

Afterward, a classification needs to be performed on the dataset according to Definition 4.3.1. The system must be able to recognize and process the typical measures of the last K useful days before disk replacement. To this aim, a preliminary analysis has to be carried out to understand how many days before the replacement, the SMART measures of a disk, typically, begin to significantly change.

4.3.2 RGF-Based Machine Learning Approach

Training of Models

The considered dataset S_d needs to be split into training, S_L, and test, S_T, sets. As there is a need to classify each hard disk according to its state of health, a binary classification distinguishing between "at-risk" s_b and "not at risk" s_h disks must be performed. The classification phase is performed by using the RGF model. The training algorithm requires a tuning of the hyperparameters.

The goal is to learn a single nonlinear function $h(\boldsymbol{x})$ on some input vectors $\boldsymbol{x} \in S_L$ with labels Y, minimizing the argument of a loss function $L(h(X), Y)$. The algorithms used for learning can be RGF with L_2 regularization on leaf-only models (referred to as "*RGF*" in the following), or the RGF with Minimum-Penalty Regularization ("*RGF_ Opt*") or RGF with min-penalty regularization with Sum-to-zero Sibling constraints ("*RGF_ Sib*").

The *number of leaves* is a data-dependent hyperparameter, and its tuning affects the training time. The number of leaves can be chosen in the range of (1000, 10,000). The *degree of regularization* λ can be adjusted choosing a value as small

as needed $\{1, ...10^{-20}\}$. A lower value of λ reduces the importance of the regularization in the regularized loss functions. The hyperparameter *mazimum depth* γ is a parameter used only with the two Minimum-Penalty Regularization models and indirectly tunes the importance of nodes because a smaller value ensures a lesser penalty for deeper nodes. In the two min-penalty regularizers, the d_v represents the distance from the root of the generic node v, and the constant hyperparameter is elevated as γ^{d_v}. Thus, higher values of γ penalize deeper nodes. The last hyperparameter of interest is the *Test Interval*, i.e., the number of leaves added per each iteration. Besides the ones just listed, there are others hyperparameters whose configurations, if not defined, do not prevent model training but can help improving its efficiency and effectiveness. Their contribution will not be discussed in this paper, and the amplitudes exploited during the performance assessment have been set to their default value.

The hyperparameter tuning, together with the choice of the number of observation days, create a number of possible configurations that can rapidly reach a few thousands. A drastic reduction of the experiments number without losing significance can be made by means of a Design Of Experiments (DOE) using the Taguchi Orthogonal Array Designs [33]; in particular, each combination of factor levels exploited to carry out a specific experiment is referred to as plan configuration. Some of the considered parameters, such as the width of the observation window, lambda, the number of leaves, etc., can assume values within large intervals. The purpose of the design of experiments is to evaluate the effect of all parameters in some significant configurations for training purposes; this way, for each parameter, a suitable, limited number of values has been established according to both their typical interval of variation and authors' knowledge and experience.

The choice of levels for each factor is limited to the number of configurations of the experiment plan used. The risk of overfitting is averted. Due to the limited number of possible combinations, it is necessary to execute the experiment plan within intervals that reasonably already give good performance. The optimal configuration is therefore identified between levels of factors that do not cause overfitting. It is also possible to make a comparison between the performances obtained in the optimal case and in all cases foreseen by the experiment plan. This further comparison allows the system to obtain the certainty of having achieved the best performance among all the tested combinations.

Usually, the goal is generally to minimize false negatives to avoid risk targets being incorrectly predicted, thus assuring a conservative behavior from an operating point of view. The accuracy index is often used in contexts of methodologies based on predictors to give a quantization of the goodness of the model. Accuracy is defined as

$$Accuracy = \frac{True\ Positives + True\ Negatives}{Total\ Population} \quad , \tag{4.11}$$

where True Positives and True Negatives are the number of disks well predicted as healthy or non-healthy, respectively. Vice-versa, the number or disk wrongly

predicted are False Positives (in case of an healthy disk predicted as to be replaced) and False Negatives (for disks that need to be replaced and actually predicted as healthy). The accuracy is usually used to have a first feeling on the effectiveness of the predictor since it returns the fraction of correctly predicted test cases out of the totality of all test cases. Although this index shows correctly executed predictions, it does not measure the predictor's sensitivity to the distinctions between positive and negative cases. The method proposed in this work uses *Recall*, *False Positive Rate* (FPR) and *Positive Likelihood Ratio* (LR+) to better evaluate the predictor's performance.

The Recall index, defined as

$$Recall = \frac{True\ Positives}{True\ Positive + False\ Negatives} \tag{4.12}$$

is often preferred in this context; the higher the recall value, the better the reduction of false negatives.

As regards data-centers storage systems, characterized by tens of thousands of hard disks, a single percentage point of false positives corresponds, instead, to several hundred hard disks wrongly classified as "to be replaced" even if they are not. This way, in the considered application field, minimizing the false positive occurrences turns out to be as fundamental as maximizing the recall. The index associated with the risk of false positives is the False Positive Rate defined as:

$$FPR = \frac{False\ Positives}{False\ Positive + True\ Negatives} \quad . \tag{4.13}$$

The Taguchi DOE approach can be exploited to assess the factors' impact on one performance index; to this aim, the *Positive Likelihood Ratio* (*LR+*), defined as:

$$LR+ = \frac{Recall}{FPR} \quad , \tag{4.14}$$

and capable of simultaneously taking into account Recall and FPR, has been exploited. Before carrying out the training stage of the RGF model, input data have to suitably shuffled in order to guarantee independence from the temporal distribution of the measures and prevent biases related to the dataset S_d. Both training and test experiments are required for each of the plan configurations; to this aim, the whole dataset S_d is split according to a ratio of 70%–30%, for training and testing, respectively.

The Regularized Greedy Forest is trained using the the training data set S_{L} and tested using the testing data set S_{T}.

Authors suggest at least 30 runs for each plan configuration in order to simultaneously assure statistical significance and feasible execution times. For each plan configuration, the values of Accuracy, Recall and FPR, expressed in percentage terms, are calculated as the median of the Accuracy, Recall and FPR achieved in the various runs. The final index of LP+ is calculated as the ratio of the median Recall and FPR.

Algorithm 3 Classifier Training.

procedure DATASETCOLLECTION(D)
  for each disk $d_i \in D$ **do**
    $S \longleftarrow$ Get SMART tuples daily of d_i
  end for
  Return S
end procedure
procedure PRE-PROCESSING(S)
  for each SMART tuple $S_i \in S$ **do**
    if S_i is corrupted OR missed OR out-of-range **then**
      Remove S_i from S
    end if
    Label(S_i) according to Definition 4.3.1
    if S_i is replaced in $[g_1; g_2 + k]$ **then**
      Remove S_i from S
    end if
  end for
  Return S
end procedure
procedure DESIGN OF EXPERIMENT(S)
  T = TaguchiDesign(L_9, replicates = 30)
  for each design $t_i \in T$ **do**
    $Shuffle(S)$
    $S_L, S_T = Split(S, 70 - 30\%)$
    $Model = RgfTrain(t_i, S_L)$
    $LR+ = RgfTest(Model, S_T)$
  end for
  Select configuration C using $EffectPlot(T, LR+)$
  Return C
end procedure
procedure THRESHOLD ASSESSMENT(S, C)
  $Shuffle(S)$
  $S_L, S_T = Split(S)$
  Get Definitive Model $Model = RgfTrain(C, S_L)$
  Get Probabilities $P = RgfTest(Model, S_T)$
  $RC \longleftarrow RocCurve(P, S_T)$
  Select Threshold T_c using RC
  Return $T_c, Model$
end procedure

The RGF classifier returns a probability for each tuple of SMART measures of likelihood with respect to the two classes of healthy or broken. The last step for the user is the choice of a proper threshold beyond which discriminating whether the considered tuple identifies a Hard Disk that needs to be replaced or not. The choice of the optimal threshold turns out to be a trade-off between false and true positive rates. To drive the choice of the right threshold, the Receiver Operating Characteristic (ROC) curve has been exploited. The ROC curve, a graphic tool for the evaluation of FPR and TPR, helps the developer, in fact, to identify the best trade-off threshold, which the authors experienced close to the beginning of the curve knee.

Validation

The system configured and trained according to the above method can be put into production. It is useful to perform a further final validation and evaluate the trend of the probability with which each hard disk is classified as healthy or not. The tuples of SMART attributes collected daily are processed by the trained and

Algorithm 4 Prediction.

```
1: procedure PREDICTION(T_c, D, Model)
2:     for each disk d_i ∈ D do
3:         D_p ⟵ Get SMART tuples of d_i
4:         Prediction = RgfPrediction(D_p, Model, T_c)
5:         Return (D_p, Prediction)
6:     end for
7:     Return the list of disks at risk with probabilities
8: end procedure
```

tested RGF model. By reporting the evolution of probabilities versus time on a plot, samples corresponding to a change of the operating condition (from healthy to broken) can be identified a few days before the replacement of the hard disk.

4.4 Experimental Results

The proposed method has been validated on a case study at CERN; CERN has a computer center where a large volume of data generated by the complex accelerator system and experiments are stored and processed. The collected data are stored in a set of Magnetic, Solid State or Tape Hard Disks through a distributed file systems service called EOS and internally developed. There are currently roughly 65,000 magnetic drives, including 89 different models. Approximately 15,000 disk replacements occur annually due to different reasons; replacements due to bankruptcies are roughly a dozen per week. Data collected from August 2018 and October 2020 have been used for the case study experiments in Section 4.4.1. A remaining fraction of data collected in the months between

Model	Healthy	Broken	Total
Ischia	11,962	81	12,043
Capri	11,238	160	11,398
Ventotene	9172	18	9190
Procida	7948	20	7968
Ponza	4695	378	5073

Table 4.1: The composition of the dataset.

November 2020 and January 2021 were reserved for validation in Section 4.4.2. An example of the algorithm implemented in R is available as supplementary material "R-Scripts" in order to encourage the reproducibility of the method (A.2 A.3).

4.4.1 Case Study

For the case study, the five models among those used at the CERN Data Center were examined. Selected models are those characterized by the higher cardinality in the CERN Data Center; this is a key issue since a large number of hard drives are necessary to grant a sufficient amount of information on broken disks to train the RGF model. The examined datasets are shown in Table 4.1; it is worth noting that all considered datasets are unbalanced in terms of healthy versus replaced hard disks in favor of the healthy ones. For the sake of privacy, the vendors and models of the examined Hard Disks have been replaced with names of some Tyrrhenian islands.

Before collecting the data to train and test the RGF model, a study on the temporal distribution of the SMART measures has been carried out; as stated above, reasons why there may be gaps between the measures are manifold. Since the proposed method requires that for RGF training, tuples of hard disks classified according to Definition 1 are necessary, establishing after how many days of absence a hard disk can be considered as permanently replaced and its missing measures that are not attributable to transient conditions is a fundamental step. To this aim, the duration of gaps between the tuples greater than 1 day have been collected and organized as a histogram. The corresponding results are reported in Figure 4.8, where it can be noticed that the maximum number of occurrences is associated with a 5-day gap. On the contrary, the number of occurrences for gaps longer than 20 days rapidly decreases.

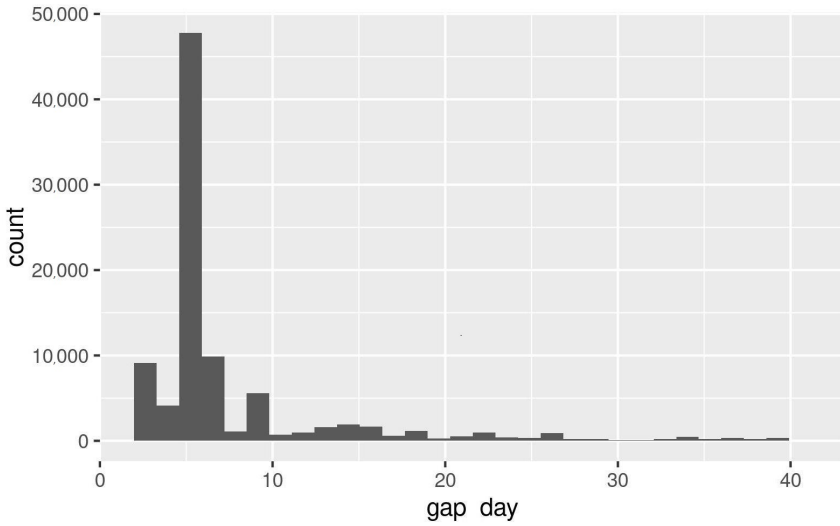

Figure 4.8: A histogram of the temporal gap between SMART tuples by drive.

It has been decided to adopt a conservative approach and to remove all the measures of the 30 days prior to the last available from the dataset. The measures of the last 30 days have however been used for the purpose of labeling hard drives. In particular, the SMART attributes considered to provide measure tuples are reported in Table 4.2.

After extrapolating only the data relating to the hard disks shown in Table 4.1 from the entire dataset S, the dataset S_d has been pre-processed to clean from corrupt, incomplete and error data according to what was stated in Section 4.3.1.

Once the dataset was cleaned, each hard disk's serial (the drive's unique identifier) was classified and labeled according to Definition 4.3.1; successively, the labeled dataset has been divided into train, S_l, and test, S_t, datasets. The training set of SMART measures of healthy disks has been randomly decimated in such a way as to assure the same cardinality of the broken drives for each disk model.

A Resolution III plan L_9 has been exploited for all five hard drive models, whose plan configurations are reported in Table 4.3.

Each configuration is characterized by at least one parameter level different from one another, thus assuring a complete and efficient investigation of the whole experimental space.

The main output of the Taguchi approach is the so-called effects diagram, i.e., the evolution of the performance index versus each parameter level; the effects diagram allows the developer to determine which level of each factor corresponds to a maximization (or minimization) of the chosen performance index. For the considered method, the goal is the maximization the $LR+$ index. As an example, the corresponding results of the DOE for the *Ponza* model is reported in Figure 4.9, in which, for each factor, the levels were coded with the numbers 1,

SMART	Attribute Name	Description
01	Read Error Rate	Hardware read errors that occurred when reading data from a disk surface
03	Spin-Up Time	Average time of spindle spin up
04	Start/Stop Count	Number of spindle start/stop cycles
05	Reallocated Sectors Count	Quantity of remapped sectors
07	Seek Error Rate	Frequency of errors while positioning
09	Power-On Hours	Number of hours elapsed in the power-on state
10	Spin Retry Count	Number of retry attempts to spin up
12	Device Power Cycle Count	Number of power-on events
192	Power-off Retract Count	Number of power-off or emergency retract cycles
193	Load/Unload Cycle	Number of cycles into landing zone position
194	HDA temperature	Temperature of a hard disk assembly
197	Current Pending Sector Count	Number of unstable sectors (waiting for remapping)
198	Offline Uncorrectable Sector Count	Number of uncorrected errors
199	Ultra DMA CRC Error Count	Number of CRC errors during UDMA mode

Table 4.2: Selected SMART Attributes[22].

Experiment	Algorithm	Loss Function	Leaf	Days
1	RGF	LS	1000	6
2	RGF	Expo	5000	7
3	RGF	Log	10,000	8
4	RGF_Sib	LS	5000	8
5	RGF_Sib	Expo	10,000	6
6	RGF_Sib	Log	1000	7
7	RGF_Opt	LS	10,000	7
8	RGF_Opt	Expo	1000	8
9	RGF_Opt	Log	5000	6

Table 4.3: The design of the experiment.

2 and 3. As regards the algorithms and the loss functions, the three coded levels are in the order: *RGF*, *RGF_Opt* and *RGF_Sib* for algorithms and *LS*, *Expo* and *Log* for the loss function. As for the levels of the number of leaves and the number of days of observation, the levels encode the values of 1000, 5000, 10,000 and 6, 7, 8, respectively. From the reported effects diagrams, it can be stated that for Ponza models, the proposed method gives better results using the RGF with L_2 regularization on leaf-only models ("*RGF*") codified as the first level of the first factor. In a similar way, *Expo*, 10,000 and 6 *days* levels are chosen for the Loss Function, maximum number of leaves and observation interval for training, respectively.

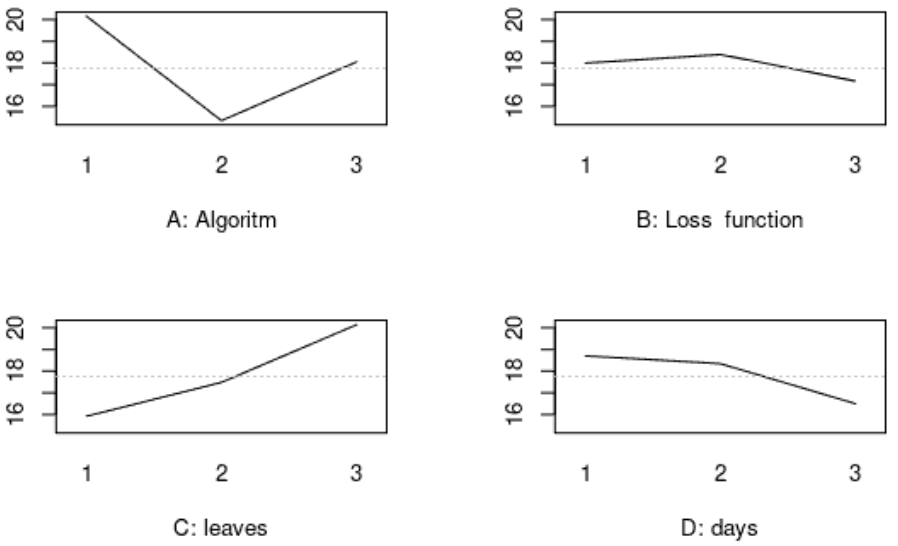

Figure 4.9: Ponza's Effect Diagram.

Similar behaviors have also been obtained for the other disk models; for the sake of brevity, they have not been reported. Choosing the levels associated with the maximum value in the effects diagrams for each parameter, the operating configuration capable of assuring the best prediction performance can be determined; the corresponding values are reported in Table 4.4 for the different disk models.

Model	Algorithm	Loss Function	Leaf	Days
Ischia	RGF_Opt	Log	10,000	7
Capri	RGF	Log	1000	8
Ventotene	RGF_Opt	Expo	1000	8
Procida	RGF_Opt	LS	10,000	7
Ponza	RGF	Expo	10,000	6

Table 4.4: Optimum Configuration.

Due to the discretization of the variation ranges, the DOE allows the developer to single out a sub-optimal operating configuration. This way, a manual tuning of the hyperparameters in a small neighborhood of the obtained configurations has been carried out in order to further improve the performance in the prediction stage; in particular, the performance index *LR+* has increased for some points. The first draft of performances are reported in Table 4.5. Using the sub-optimal configuration following the DOE, quite good performance values were obtained, but it is necessary to further reduce the False Positive Rate for the system in order to support maintenance.

Model	Recall	FPR	LR+
Ischia	95.2%	1.6%	59.5
Capri	92.7%	3.6%	25.8
Ventotene	95.0%	8.7%	113.1
Procida	97.6%	6.1%	15.9
Ponza	100%	5.1%	19.5

Table 4.5: The results with a threshold at 50%.

Therefore, it is necessary to increase the threshold beyond which a hard disk is considered close to failure. Finally, the optimal thresholds have been carried out by means of the ROC method (Figure 4.10). The associated final results are reported in Table 4.6; the remarkable results in FPR reduction can be appreciated.

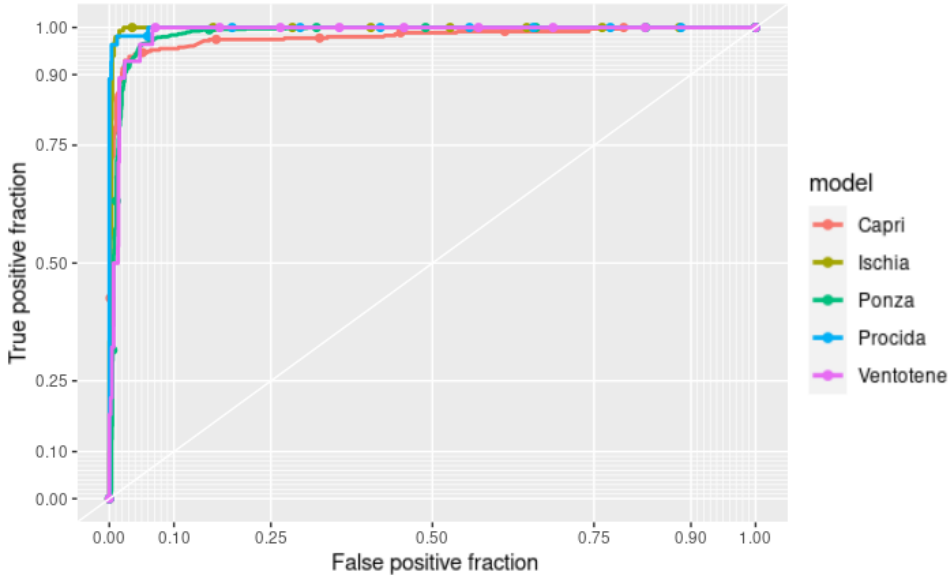

Figure 4.10: ROC Curves.

Model	Threshold	Recall	FPR	LR+
Ischia	85%	98.4%	0.2%	659
Capri	67%	92.2%	0.8%	115
Ventotene	74%	95.8%	0.6%	160
Procida	62%	99.5%	0.4%	284
Ponza	88%	78.1%	0.9%	86

Table 4.6: The final results with the optimized Threshold

4.4.2 Validation

The performance of the model was evaluated on a small fraction of disks not used in the previous phases. From the validation set, the hard drives replaced due to failure have been extracted, according to the heuristic approach proposed in this paper. Thus, the predictor's ability to recognize these hard drives at risk in the days preceding the replacement has been assessed. The serial number of the replaced hard disks have been masked using the name of German cities and French rivers. SMART data from the last 60 days prior to the replacement day were collected from these hard drives, and the predictor was evaluated on the tuples of each day. The most significant results are shown in Figure 4.11a,b for Ischia and Ventotene models, respectively. It can be noted that some hard disks showed a significant change in the range of a week before the replacement day,

and other hard disks were to be considered at risk even tens of days before replacement.

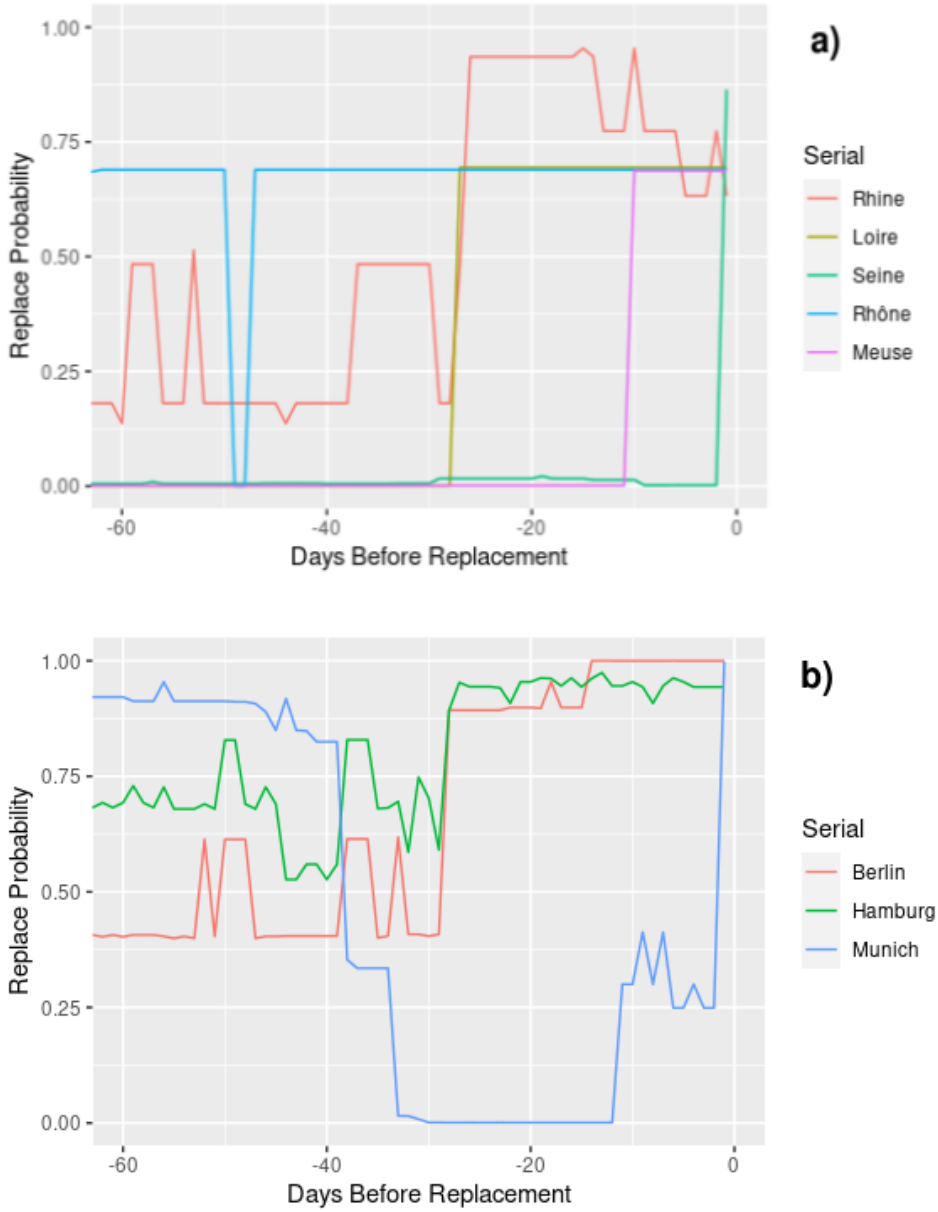

Figure 4.11: Validation of the Ischia (**a**) and Ventotene (**b**) Hard Disk Models.

4.5 Final Remarks and Discussion

In this paper, a method for predicting hard disk media failures and hence increasing the availability of large-scale storage services has been presented. The main contribution is a proper stage to automatically label (healthy/at-risk) the disks during the training and validation stage along with the tuning strategy to optimize the hyperparameters of the associated machine learning classifier. This way, the described classification model is fully automated and avoids any repeated human intervention or judgment from a storage deployment team. The proposed method is based on an inferential modeling approach, designed to adhere to the hard drive model it is trained on. Consequently, a forest is trained for each hard disk model. This feature allows the prediction system to overcome the heterogeneity problems of hard disk models typical of datacenters. It is hence applicable in large data-centers faced with a heterogeneous population of storage devices and storage deployments. The presented method is based on a practical identification heuristics for failed disks and the application of supervised machine learning (Regularised Greedy Forest), exploiting the full set of available SMART metrics for failure predictions. A data preparation algorithm has been presented, it takes care of handling operational problems, such as gaps in SMART sensor collection. The method allows the system to reliably identify hard disks that have been replaced due to failures, in contrast to other frequent deployment operations, such as disk retirement or relocation, and to operate without the requirement of keeping consistent replacement logs, e.g., by a data center operations team. The trained model with our method achieves a promising level of accuracy, in excess of 95%, and a False Positive Rate typically below 5%, even reaching below 1% in some cases. The additional information from the model allows storage service providers to reduce the risk of service unavailability, e.g., by proactive re-replication or via the determination of "at-risk" failure groups with multiple devices with an increased failure likelihood. This method has the disadvantage of the need to archive a continuous flow of data from the probe that collects data from the hard disks. In order to carry out adequate training and achieve satisfactory performance values, it is necessary to collect a set of measurements from tens of failed hard drives. In future works, there is the intention to extend this method also to solid state drives whose quantity in our case study is growing day by day and will soon be sufficient to validate the methodology. In future works, two main tasks will be carried out. The first task will be to validate the current methodology, especially the pre-processing phase described above, on larger datasets that are continuously collected and tested with other machine learning classifiers. The second task will be to validate the methodology presented above on other contexts in which the dataset is very unbalanced and for which the False Positive Rate assumes an important equal (or greater) of false negatives, such as asynchronous electric motors and power supplies.

5 Fault diagnostics and Failure Prediction in Induction Motors

5.1 Introduction

The three-phase asynchronous motor is widely used as an electric drive due to its design simplicity, low production cost, sturdiness and reliability. Furthermore, the asynchronous motor is characterized by a high efficiency. It can also be simply connected directly to the distribution network with constant voltage and frequency if it is not necessary to control it in speed with an inverter. The inrush current, at start-up, may be greater than the current absorbed at full load (4 -10 times higher), although the torque may be small due to the initial phase shift. In the three-phase asynchronous motor there is a fixed part called the stator and a moving component called the rotor. The asynchronous motor (also called induction motor) is an alternating current motor whose angular speed of the rotor is lower than the rotation speed of the magnetic field generated by the stator windings, hence the asynchronous[2].

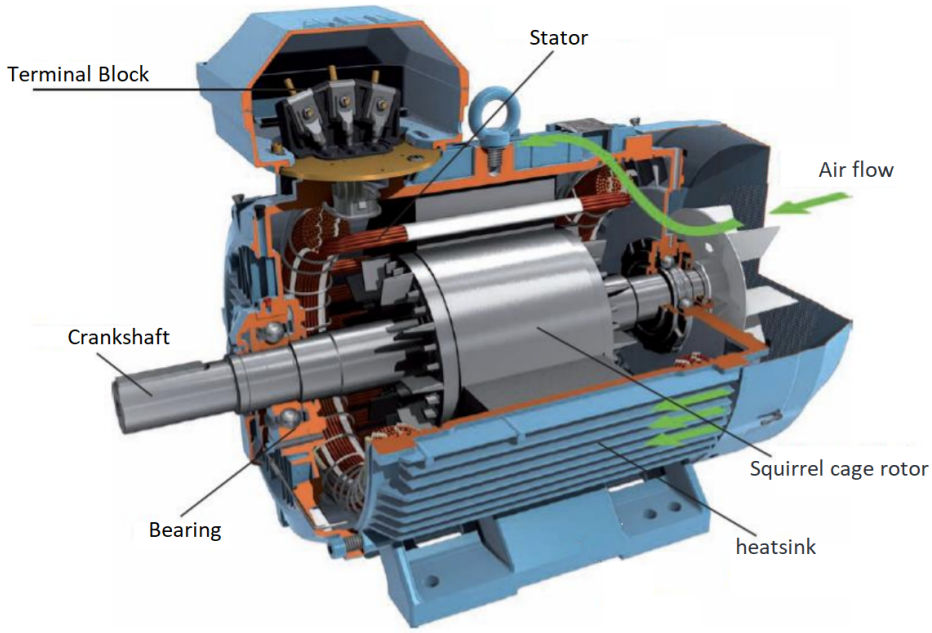

Figure 5.1: Induction Motor

A set of laminations makes up the stator, it is in the shape of a circular crown. Between these laminations there are internal grooves in which the enamelled copper conductors of the three-phase stator winding are wound. Inside the stator there is the rotor which also consists of a set of laminations, with an internal hole in which the rotation shaft is inserted. The stator contains external grooves in which the rotor winding is inserted.There is a thickness of air or dielectric between the rotor and the stator called an air gap, typically less than a millimeter large to allow the rotor to spin freely. The most common rotor is the squirrel cage rotor which is made by inserting die-cast aluminum or copper bars into the channels. The bars are directly connected to each other on the part that protrudes beyond the reed valve, on both sides. Induction motors are widely used and appreciated in industry as they have a very low rotor resistance; voltages on the rotor are not high, while the currents are high due to the low value of the resistance. Finally the rotor does not have a number of its own poles, but it depends on the number of poles of the stator.

Induction motors play an important role in industrial manufacturing. To get a sense about the impact of the induction motors in industrial field suffice it to note that these devices account for 29% of global and 69% of industrial electricity consumption[43]. As seen in the chapter 2 on the state of the art, predictive maintenance of electric motors is a hotly debated topic. Signal processing and artificial intelligence recent advancement have attracted renewed interest in induction motor diagnostics thus model-based approaches can be overcome thanks to the

machine learning-based fault diagnosis methods[59]. Induction motor faults are mainly diagnosed by using characteristic signals of the motors, such as vibration signals, thermal images, acoustic signals, and motor currents[27].

It is well documented in the literature that mechanical faults are generally identifiable with appropriate motor vibration analysis. On the other hand, faults of an electrical nature can be adequately identified by means of analysis of the current signals. The diagnostic techniques by means of vibrations are however invasive as it is necessary to have complete access to the motors in order to install the sensors. Vibration-based diagnostics can be performed using sensors installed on the system or portable sensors that require human intervention for positioning. For fixed sensors, there are techniques that provide triaxial sensors to be able to analyze vibrations in space and there are sensors to be applied in several points of the engine in order to distinguish the various types of faults. In the case of sensors to be used manually, an operator must intervene to be able to measure the vibrations in particular points of the engine. It is evident that the vibration analysis is invasive as it requires complete access to the engine, this is not always possible. Generally, in real contexts, asynchronous motors are located in places where access is limited or space restrictions are present. This constraints prevent a diagnosis guided by an operator. At the same time, in many cases the cost of installing vibration sensors is not economically advantageous. Finally, it is important to remember that vibration-based techniques are able to achieve satisfactory accuracy only in the diagnostics of mechanical faults and it is not yet possible to identify, from the vibration signals, faults of a nature other than mechanical. An approach based on current measurements can solve all these problems and has the advantage of being easily used in many contexts where cost and invasiveness are critical requirements. The method proposed below is based precisely on the collection and treatment of current signals coming from the power supply circuit of the three-phase asynchronous electric motors.

5.2 Current signals

The prediction maintenance system presented in this work requires particular focus on the measurement sensors which assume a critical importance for the purpose of the performance. It is necessary to pay attention to the signal-to-noise ratio (SNR), the sampling frequency, the size of the measurement buffer, etc. On the other hand, the processing speed of these signals is not a very important requirement as the dynamics of the asynchronous motors and the evolution of the failures leave the designer enough time to process the prediction algorithms.

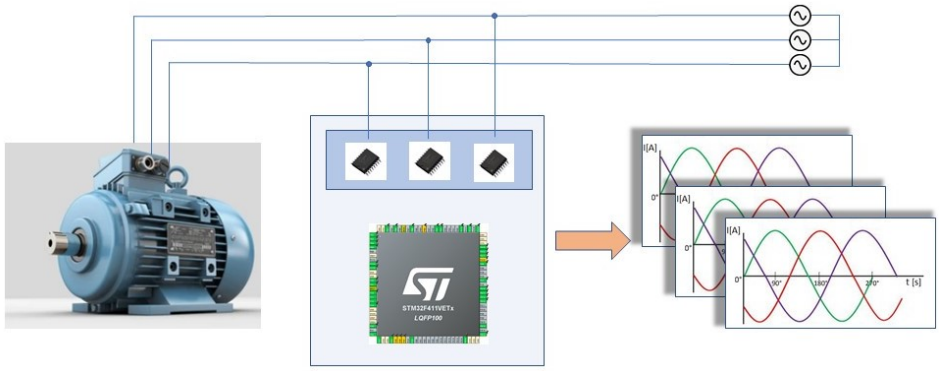

Figure 5.2: Measurement System

The choice of sensors to be adopted depends on the type of measures to be made. Three-phase asynchronous electrical motors are generally powered by alternating current at 400V at 50Hz. From the study of the state of the art, it is known that some failure phenomena have an impact on the supply current in a band centered on 1kHz. It is therefore necessary to acquire with a frequency greater than double the frequency in which the phenomenon generally occurs but which does not cause an excessive consumption of resources.

5.3 Prediction Algorithms

The prevention of failures by means of artificial intelligence is operable by adopting a classifying algorithm trained to recognize an error state before its activation causes the failure.

Therefore the issue of performing fault detection must be addressed by adopting a classifier that, on the one hand, ensures a satisfactory level of accuracy in the prediction of failures and, on the other hand, allows the diagnostic system to be easily integrated into edge computing solutions.

The goal is therefore to train an algorithm represented by a function $f(X)$ that, given the current signals measured on the power supply line, returns a symbol that identifies the class to which the signals belong (healthy / broken). The function (5.1) to be constructed is therefore a relationship between a set of cardinality m to a set of cardinality 1. The function $f(X)$ is therefore the following:

$$Y = f(X) : R^m \rightarrow R^1 \tag{5.1}$$

where m is the number of features and $R^1$is the label set. The function, therefore represent the classification algorithm and the output is the result of the classification.

The features extracted from the signal are derived from the spectral analysis of the measured signals. In particular, the largest spectral components of the

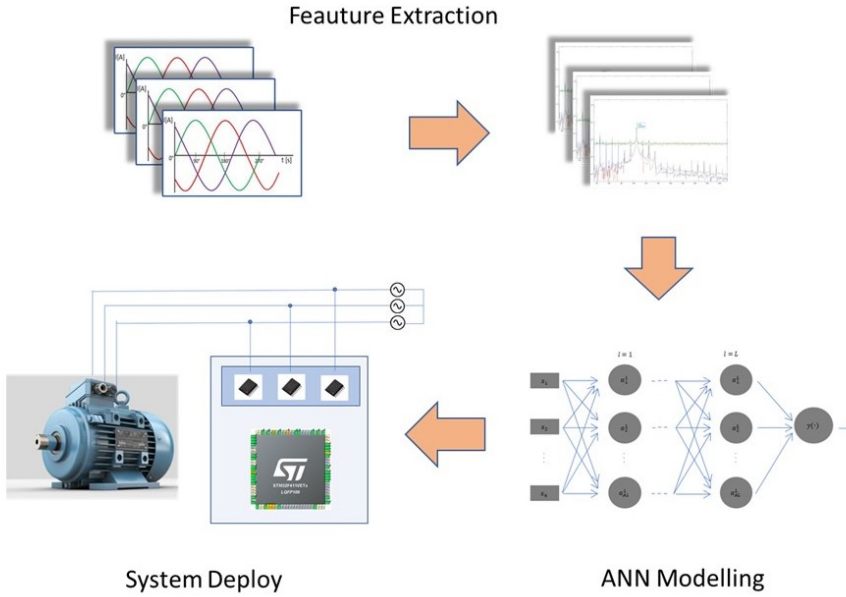

Figure 5.3: Predictor System

Fast Fourier Transform (FFT) and the largest components of the power spectral density (PSD) are collected from each signal. For each component, amplitude-frequency pairs are selected. The method proposed in this work adopts both FFT and PSD because the aim is to extract characteristics from the signals that are representative of the failure phenomena. A failure cannot be identified on the basis of thresholds in the FFT or PSD components alone. This is the reason why the method proposed in this work adopts a more complex inferential modeling by adopting a large number of feautures extracted from both the FFT and the PSD.

In consideration of the inputs taken, the function of the classifier therefore becomes:

$$Y = f(\boldsymbol{F}^{FFT}, \boldsymbol{A}^{FFT}, \boldsymbol{F}^{PSD}, \boldsymbol{A}^{PSD}) \tag{5.2}$$

The number of components to be selected must be large enough to represent the reconstructed signal with good approximation. At the same time it is not advisable to select too large a number of components for reasons of computational complexity. It should be noted that an excessive number of components (FFT and PSD) risks being representative not only of the signal but also of the noise of the measures.

The model therefore takes as input the features described above extracted from the observations collected from the supply line.

The method proposed in this work adopts *Feed-Forward* Artificial Neural Network (ANN) as machine learning model. Feed-Forward ANN's connections between the units do not form cycles. In the case of feed-forward networks, function 5.1 is transformed consequently and it is necessary to introduce the concept of *Neuron* (hence the name) to explain it.

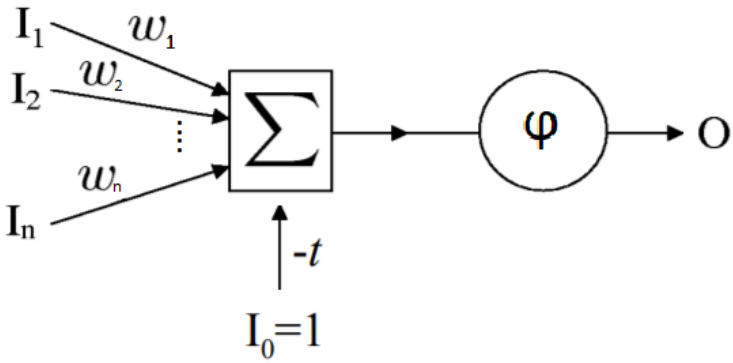

Figure 5.4: Neuron

Given an input vector I, a *Neuron* consists of an activation function ϕ that takes as input the weighted sum of the elements of the vector I (fig. 5.4). This simple neuron is called *Perceptron*, introduced by Rosenblatt in 1962. Often the threshold is included in the neuron model, adding a fictitious input with value on this input fixed at 1, the weight of the connection is given by $-t$ that can be imagined as an additional neuron with no input values. Formally the function of the neuron becomes:

$$O = \phi(\sum_{i=0}^{n} \omega_i I_i) \tag{5.3}$$

where n is the total number of neuron's inputs.

The activation function ϕ can be of different types, the ones that are adopted in this proposed methodology are listed below.

- Rectified Linear Unit (ReLU) which is an activation function defined as the positive part of its argument(fig.5.5), the function ϕ is $\phi(x) = max(0, x)$;
- Hyperbolic tangent (Tanh) which is an activation function defined as $\tanh(x)$ (fig.5.6), thus the function ϕ becomes $\phi(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- Sigmoid, sometimes named as Logistic or Soft Step (fig.5.7), whose function is $\phi(x) = \frac{1}{1+e^{-x}}$

It is important to note that in the configuration of the neural network architecture, the activation function is a very important hyperparameter in order to

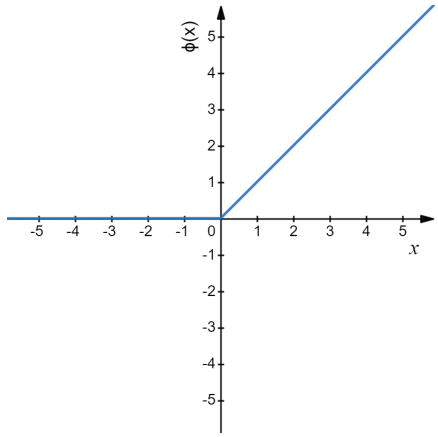

Figure 5.5: ReLU

achieve high performance. All neurons therefore contribute to form the network. If the output of a neuron is linked in input to a new neuron, the latter contributes to form a level that is defined as a deep level. Hence, the network is organized in levels: each neuron of a level receives input only from the neurons of the previous level; it propagates the outputs only towards the neurons of the following levels. The layers between the inputs and the output are called *Hidden Layers*. Self-connections are not allowed in this type of network, nor are connections between neurons belonging to the same level.Each neuron, therefore, has the function of propagating the signal through the network, with a flow of information that goes from the previous level to the next level (the levels could coincide with the input or output of the network). It follows that the first level of the Neural Networks takes argument of 5.2 as input.

In the fig.5.8 a generic architecture of a Feed-forward neural network is reported, where $l \in \{2, ..., L\}$ is the layer index, the total number of hidden levels is L and the A_l is the generic number of neurons per level l. The number of intermediate levels $(L-1)$, the number of neurons for each level (A_l) and the activation functions $(\phi(x))$ are hyperparameters that compose the architecture configuration and should be established at the first stage, in order to get the topology, the number and type of neurons, the connections, etc. Further hyperparameters are added that do not describe the architecture in the strict sense but contribute to the performance of the model: Regularization strength (Lambda) and The Standardize data option. The Regularization strength (Lambda) hyperparameter specifies the regularization penalty term and the Standardize Data binary hyperparamenter specifies whether to standardize the numeric predictors must be standardized or not, in the case of predictors with widely different scales. The term of the regularization penalty affects the weight of the regularization, depending on the value chosen, the risk of overfitting is prevented or increased.

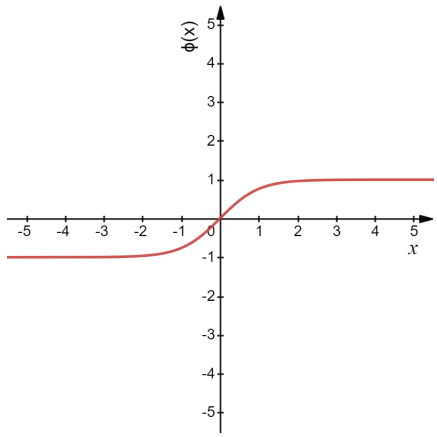

Figure 5.6: Tanh

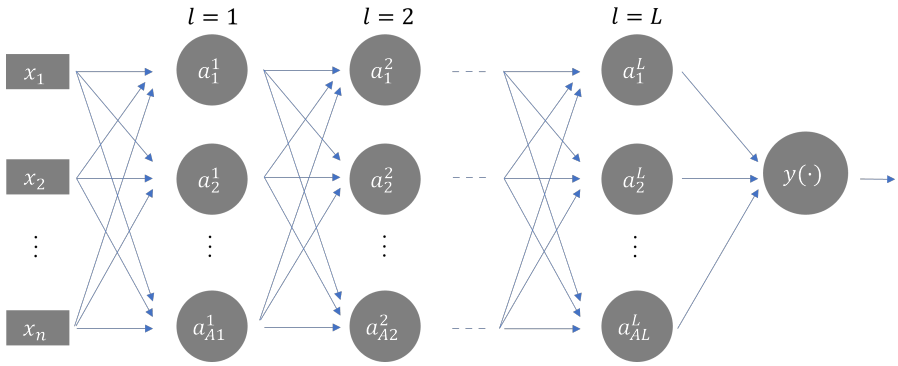

Figure 5.8: Feed-forward neural network architecture

Once the architecture is ready, second stage is to determine the weights of the connections in order to build a classifier, based on the training data set made available to the network (placed in input), this phase is named Training.

However, the choice of the hyperparameters that describe the architecture must be guided by technical approach because the possible hyperparameters combinations can be infinite. Of course, the goal is to maximize the classifier performances while trying to minimize the complexity of the architecture.

Hyperparameter configuration can be chosen via grid search based approach, which consists of an exhaustive search in a limited range of possible configurations, or by a random search based approach, which consists of a non-exhaustive and random search for configurations in a range of possible combinations. In case of large number of hyperparameters, the random search technique is to be preferred over the grid search from the computational times' viewpoint, because it has been demonstrated that random search is able to preserve good

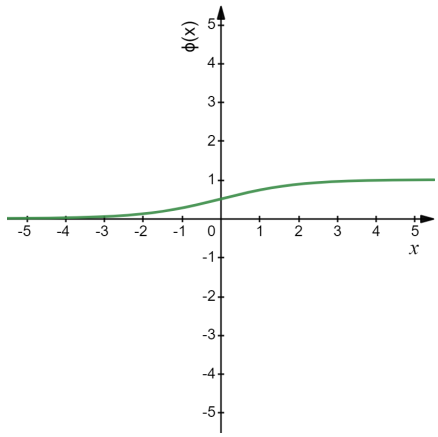

Figure 5.7: Sigmoid

performance[6].

As mentioned, the training phase is used, in an iterative way, to do the tuning of the weights which affect the connections between neurons. The weights are updated in each iteration taking into account the error between the result obtained and the desired result. It is therefore necessary to introduce an error function that must be minimized at each iteration.

In order to evaluate the effectiveness of the training operations and of the resulting model, it is necessary to preserve a fraction of the dataset on which to test the model. The simplest division technique to evaluate a Machine Learning algorithm is that of static division (ie 70% -30%), but it can involve drawbacks and is more appropriate in contexts where the dataset is very large and it is not possible to perform multiple trainings. The method presented in this paper for the prevention of failures in electric motors, on the other hand, recommends using cross-validation. This technique operates a division of the dynamic dataset, it is in fact a statistical technique that allows the data to be used alternately both for the train and for the test. It is often called k-fold cross validation because the dataset is initially divided into a series of equal portions and uses a fraction for the train and a fraction for the test at each of the k-iterations. In this work the cross validation technique is adopted as it is more suitable when the datasets are not large. In contexts such as that of electrical motor diagnostics it is not easy to collect large datasets of data for the model training. In fact, it is evident from the state of the art that the methods proposed for failure prediction on electric motors, in literature, have also been validated on a limited number of faulty motors. Finally, the optimization of the model can be guided by means of the Receiver Operating Characteristic (ROC) curve. The ROC curve is a graph that relates the sensitivity and specificity of a diagnostic test to the variation of the cut-off value (threshold value). Sensitivity and specificity are two fundamental indicators to be able to correctly interpret the result. Sensitivity is the propor-

tion of positive cases that are correctly classified, the specificity of a diagnostic test is the proportion of negatives that are correctly classified. Increasing the cut-off value increases the number of false negatives, while the number of false positives decreases. Consequently, there is a highly specific but not very sensitive test. Lowering the cut-off value, the number of false positives increases, while the number of false negatives decreases, therefore we have a highly sensitive but not very specific test. The larger the area under the curve, the higher the quality of the model.

5.4 Maintenance Prediction Device: a case study

The case study on which the method was validated includes a set of motors of various nature and different working conditions.
Current sensor chosen for the acquisition is the MCR1101-20-5. The sensor's characteristics are reported in table 5.1.

Figure 5.9: MCR1101-20-5 Package

It was decided to adopt this sensor due to its full scale, passband and limited magnetic hysteresis characteristics. The sensor has been validated in laboratory tests using the Fluke 5720A Calibrator and 5725A Amplifier. The evaluation of the magnetic hysteresis was performed by subjecting the sensor to increasing and decreasing current flows and acquiring 10 thousand samples for each current step.

Parameter	Typical Value for $VCC = 5V$ and $T_A = 25°C$
Input Range	$\pm 20\,A$
Sensitivity	$100\,mV/A$
Zero Current Offset	$\pm 20\,mA$
Sensitivity Error	$\pm 0.3\,\%$
Linearity Error	$\pm 0.3\,\%FS$
Total Error	$\pm 0.6\,\%RD$
Zero Current Offset Drift	$\pm 60\,mA$

Sensitivity Drift	$\pm 0.3\,\%$
Total Error Drift	$\pm 0.4\,\% FS$

Table 5.1: MCR1101-20-5 sensor'scharacteristics

The sensors were evaluated in order to ensure the validity of the measurements. A measurement campaign was performed in order to asses sensor performances.

Input Current [A]	Mean [A]	Standard Deviation [A]	Samples
-10,000	-9,872	0,007	10000
-9,000	-8,878	0,005	10000
-8,000	-7,886	0,006	10000
-7,000	-6,926	0,007	10000
-6,000	-5,913	0,008	10000
-5,000	-4,926	0,007	10000
-4,000	-3,939	0,007	10000
-3,000	-2,954	0,007	10000
-2,000	-1,972	0,008	10000
-1,000	-0,978	0,007	10000
0,000	0,010	0,008	10000
1,000	1,009	0,008	10000
2,000	1,990	0,008	10000
3,000	2,983	0,007	10000
4,000	3,986	0,006	10000
5,000	4,992	0,006	10000
6,000	5,982	0,006	10000
7,000	6,984	0,005	10000
8,000	7,983	0,006	10000
9,000	8,961	0,006	10000
10,000	9,990	0,007	10000
9,000	8,957	0,006	10000
8,000	7,973	0,006	10000
7,000	6,993	0,007	10000
6,000	5,970	0,007	10000
5,000	4,974	0,007	10000
4,000	3,983	0,007	10000
3,000	3,018	0,008	10000
2,000	2,009	0,007	10000
1,000	1,010	0,007	10000
0,000	0,009	0,008	10000
-1,000	-0,980	0,008	10000

-2,000	-1,983	0,008	10000
-3,000	-2,953	0,007	10000
-4,000	-3,940	0,007	10000
-5,000	-4,942	0,007	10000
-6,000	-5,912	0,007	10000
-7,000	-6,911	0,006	10000
-8,000	-7,901	0,006	10000
-9,000	-8,889	0,007	10000
-10,000	-9,879	0,007	10000

Table 5.2: Magnetic Hysteresis

In the figure 5.10 the average values of 10 thousand samples per each step are plotted. It is evident from the graph that the averages of the measurements overlap, so it can be deduced that there are no relevant magnetic hysteresis effects.

The Microcontroller (MCU) chosen for the setup is the *STM32F4V11VET*. The MCU's charachteristics are written below, for the sake of brevity, just information relevant for the case study have been reported below.

- Arm® 32-bit Cortex®-M4 CPU with FPU;
- 512 Kbytes of Flash memory;
- 128 Kbytes of SRAM;
- General-purpose DMA;
- Up to 11 timers;
- A 12-bit A/D converter 2.4 MSPS with 16 channels;
- Up to 3 USARTs.

The current sensors output a voltage proportional to the measured current. Since it is necessary to acquire samples coming from 3 motor phases, 3 ADC channels have been used on which the voltage signals coming from the MCR1101-20-5 sensors are input. It is necessary to reach a sampling rate in order to collect from the three channels measurements at 10000 samples per second. This is possible by using the DMA and setting it so that as soon as the ADC produces a valid value, the DMA takes it to a buffer in RAM. Of course it is necessary to reach a trade off between sample size and available RAM resources.

In this case study it was possible to acquire 20 whole periods with 20000 samples for each phase, for a total buffer of 60000 samples.

The device including sensors and wiring to operate the acquisitions is shown in the figure 5.11.

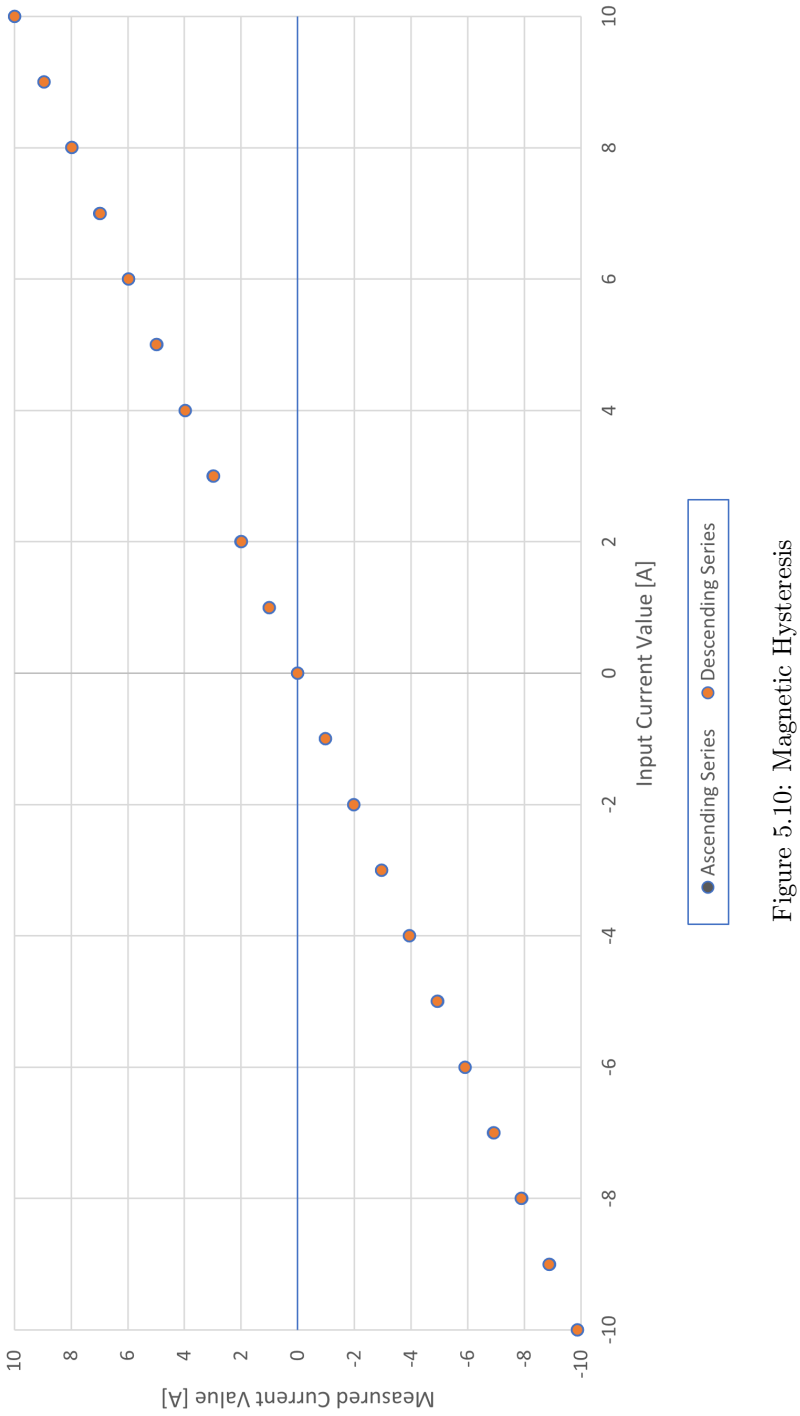

Figure 5.10: Magnetic Hysteresis

Figure 5.11: Data Acquisition System

To operate the measurement campaign, it was necessary to acquire samples on a large number of engines in different health conditions. Samples from the 3 power supply streams were collected for each motor. The dataset used for the case study is shown in the table ??.

Classes	Number of Motors	Class'dimension
Healthy	7	21
Faulty	21	63

Table 5.3: Dataset

For each sample, the 10 largest frequency components of the Fast Fourier Transform (FFT) and the 10 largest components of the Power Spectral Density (PSD) were selected. The dataset is then created by taking the frequencies and amplitudes of the largest 10 components of the FFT and PSD.

$$\boldsymbol{X} = \{f_1, A_1^{FFT}, f_2, A_2^{FFT}, ..., f_{10}, A_{10}^{FFT}, f_1, A_1^{PSD}, f_2, A_2^{PSD}, ..., f_{10}, A_{10}^{PSD}\} \quad (5.4)$$

The Dataset is therefore composed of 40 features that describe in a synthetic way, and with a good approximation, the nature of the acquired signal. For the training and testing of the model, the *k-fold* technique was adopted. In this case, 5 folds were selected.

It is necessary to make a choice of hyperparameters before starting the training. As illustrated above, the random search technique can lead to satisfactory results by reducing development times. The table 5.4 shows the ranges of all the hyperparameters, it is natural that the number of all the possible configurations is very high, this entails a great deal of difficulty in operating a grid search technique (exhaustive evaluation of all configurations).

Hyperparameter	Range
Number of Fully connected Layer	$\{1-3\}$
First Layer Size	$\{1-300\}$
Second Layer Size	$\{1-300\}$
Third Layer Size	$\{1-300\}$
Activation	$\{ReLU; Tanh; Sigmoid\}$
Regularization Strength (Lambda)	$\{1.1905e^{-07} - 1190.4762\}$
Standardize Data	$\{Yes - No\}$

Table 5.4: Hyperparameter configuration

The training phase is therefore carried out iteratively and it is necessary to introduce criteria with which to terminate it. The iterations can be limited in number, in time or on the basis of an index and the achievement of its threshold value.

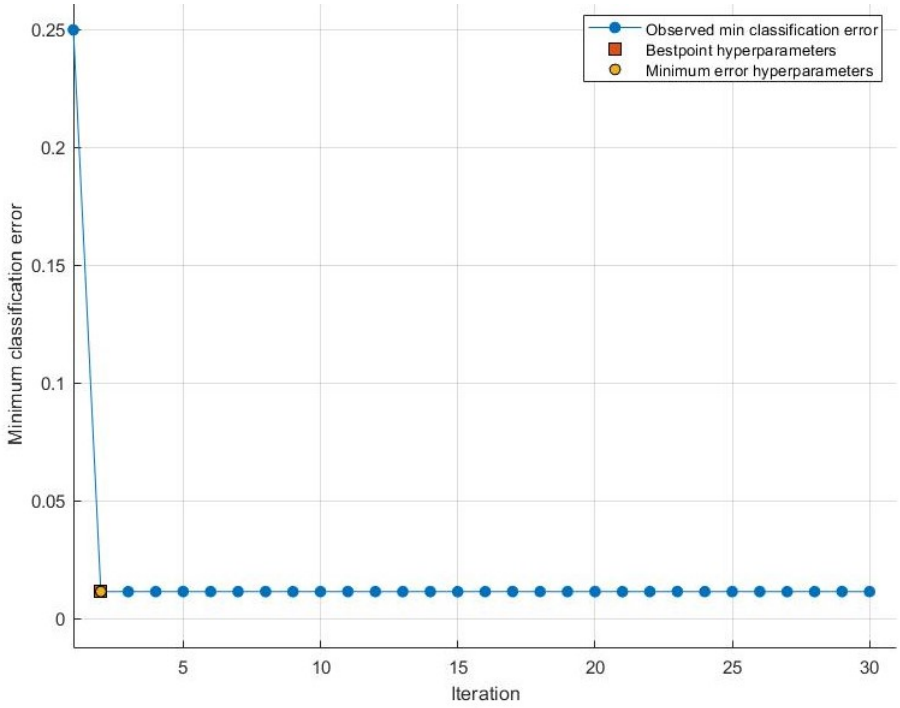

Figure 5.12: Neural Network Minimum Classification Error

The graph in the plot represents the estimate of the Minimum Classification Error (MCE). The MCE is calculated considering the sets of hyperparameter values for each iteration (blue points). The yellow dot and the red square respectively represent the Minimum Error Hyperparameters and the Bestpoint Hyperparameters. In the figure 5.12 the Minimum Error Hyperparameters and the Bestpoint Hyperparameters coincide.

The results are summarized at the end of the cross validations in the confusion matrix (fig. 5.13) where known and predicted classes are reported. The false classes (labeled as 0) correspond to observations labeled as healthy, that is, observations collected from the engines in good condition. The true classes are the classes labeled as faulty, i.e. observations corresponding to motors in anomalous conditions (broken bearings, misalignments, etc.).

The graph 5.14 shows the ROC Curve which represents the relationship between Sensitivity (True Positive Rate) and Specificity (True Negative Rate). The Sensitivity is the probability of positives cases correctly classified and the Specificity is the probability of negative cases correctly classified as negatives.

The Sensitivity is calculated by taking the ratio between the cases belonging to

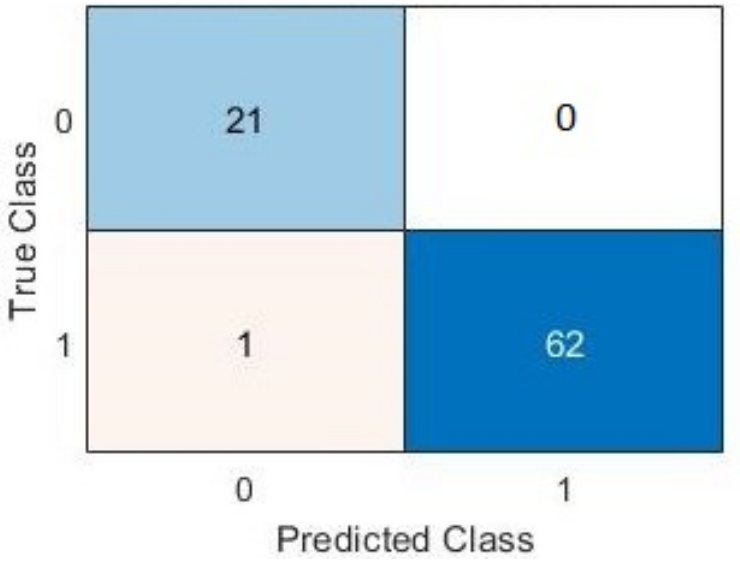

Figure 5.13: Neural Network Confusion Matrix

the class of fault signals correctly classified as positive (the true positives) divided by the sum between true positives and the faulty cases erroneously classified as negative (the false negatives) (5.5). This index represents the probability with which a classifier correctly identifies a faulty case as positive.

$$Sensitivity = \frac{True\ Positives}{True\ Positives + False\ Negatives} \quad , \tag{5.5}$$

The Specificity is calculated by taking the ratio between the cases belonging to the class of nominal device signals correctly classified as negative (the true negatives) divided by the sum between true negatives and the healthy cases erroneously classified as positive (the false positives). This index represents the likelihood with which a classifier correctly identifies a healthy case as negative.

$$Specificity = \frac{True\ Negatives}{True\ Negatives + False\ Positives} \quad , \tag{5.6}$$

Both sensibility and Specificity indices are calculated from the results presented in the confusion matrix 5.13.

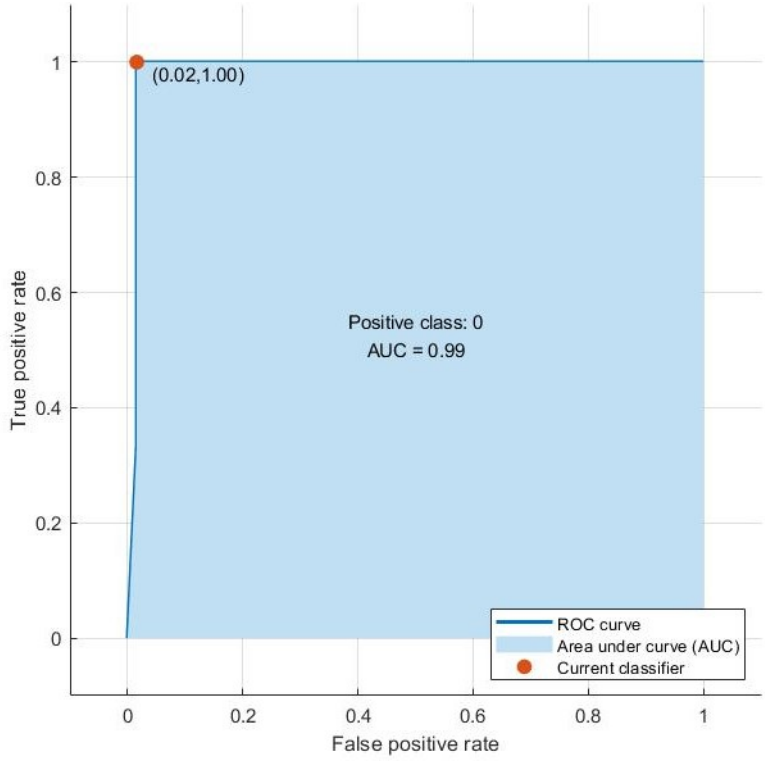

Figure 5.14: Neural Network ROC Curve

5.5 Performance Discussion

A comparative analysis was carried out between the proposed method and a modified version of it. The comparison was made by replacing the machine learning core with two common classifiers. The selected algorithms for this comparison were chosen for their characteristic of being widely used in diagnostic applications based on the machine learning approach. The solution based on a feed forward neural network has been compared with Support Vector Machine (SVM) and Decision Tree (DT).

Support Vector Machine is a binary classifier trained on a set of labeled patterns[45]. A Training set can be defined as:

$$(\boldsymbol{x_i}, y_i) \in R^l \times \{\pm 1\} \quad i = 1, ..., N \tag{5.7}$$

where $\boldsymbol{x_i} \in R^l$ is the input data set and $y_i \in \{\pm 1\}$ is the target. The goal of the support vector machine is to divide the samples by a hyperplane so that the

Hyperparameter	Range
Box Constraint level	$\{0.001 - 1000\}$
Kernel scale	$\{0.001 - 1000\}$
Kernel Function	$\{Gaussian, Linear, Quadratic, Cubic\}$
Standardize Data	$\{True, False\}$

Table 5.5: Hyperparameter configuration - SVM

division coincides with the targets y_i.
The classification function is defined as:

$$f(\boldsymbol{x}) = sgn(\boldsymbol{w} \cdot \boldsymbol{x} + b) \tag{5.8}$$

The function sgn is the bipolar sign function, the vector w is the vector of coefficient and b stands for the bias of the hyperplane.

The classifier's hyperplane must be idenified in order to satisfy the condition that y_i is greater than or equal to one:

$$y_i[\boldsymbol{w} \cdot \boldsymbol{x} + b] \geq 1, \quad i = 1, ..., N \tag{5.9}$$

The 5.9 can be modified as reported in 5.11 in order to introduce a slack variable to identify a hyperplane that does not fully satisfy the The 5.9 but maximizes the result.

$$y_i[\boldsymbol{w} \cdot \boldsymbol{x} + b] \geq 1 - e_i, \quad i = 1, ..., N \tag{5.10}$$

The goal of the algorithm is to minimize the following:

$$min\boldsymbol{J}(\boldsymbol{w}, e, b) = \frac{1}{2}\boldsymbol{w} \cdot \boldsymbol{w} + \frac{1}{2}C\sum_{i=1}^{N} e_i^2 \tag{5.11}$$

As performed in the case of the neural networks described above, also in this case, for comparative purposes, we proceeded with the evaluation of the performances with the cross validation technique. The random search technique was also applied to the support vector machine for the configuration of the hyperparameters. The possible values that the hyperparameters can assume are shown in the table 5.5.
Following the application of the random search, the optimal configuration obtained consists of the kernel function such as the Gaussian, the Kernel scale 12.6062, the Box constraint level 167.007, and Standardize data true. This optimal configuration allowed to reach an accuracy of 97.6 %, true positive rate of 100% and true negative rate of 90,4%.

The Support Vector Machine ROC Curve is shown in Figure 5.20. The curve is quite similar to that obtained with the feed forward neural network proposed

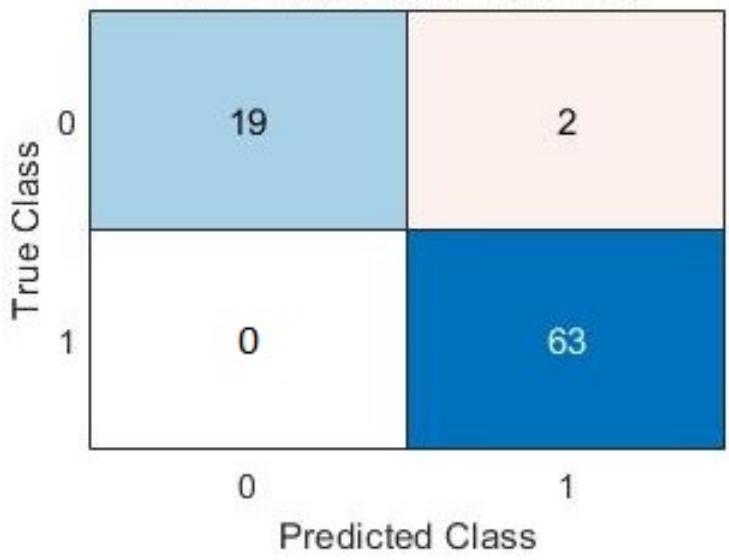

Figure 5.15: SVM Confusion Matrix

in the method, in fact the accuracy level achieved is not much lower. This does not mean that the feed forward neural network is the best choice for predictive maintenance purposes in asynchronous three-phase electric motors.

The Decision Trees are Machine Learning algorithms that can be used for both regression and classification problems. A decision tree is a tree-like model of decisions and it is usually build upside down with its leaves at the bottom.

In decision trees, decisions are represented by the path taken from the root to the leaf node. Tree construction occurs iteratively through leaf splitting or pruning. The random search and cross validation techniques have also been applied in the case of decision trees. The possible values that the hyperparameters can assume are shown in the table 5.6. It is necessary to establish the optimal split criterion for this case. This choice will also be made by means of the random search technique.Two split criteria are considered: Gini Diversity Index and Maximum Deviance Reduction function. Gini Index (G) is defined according to the formula 5.12:

$$G = 1 - \sum_{k} P_k^2 \quad , \tag{5.12}$$

where the percentage inside a group of elements is defined as P_k and the group of elements must belong to class k. The value G represent the purity and it is equal to 0 if all the elements (inside the group) are part of the same class. Thus, from the branches the node returns as output observation of just one class, given all the elements belong to that specific class, the classification error is null.

The other split criterion is the Maximum Deviance Reduction function (some-

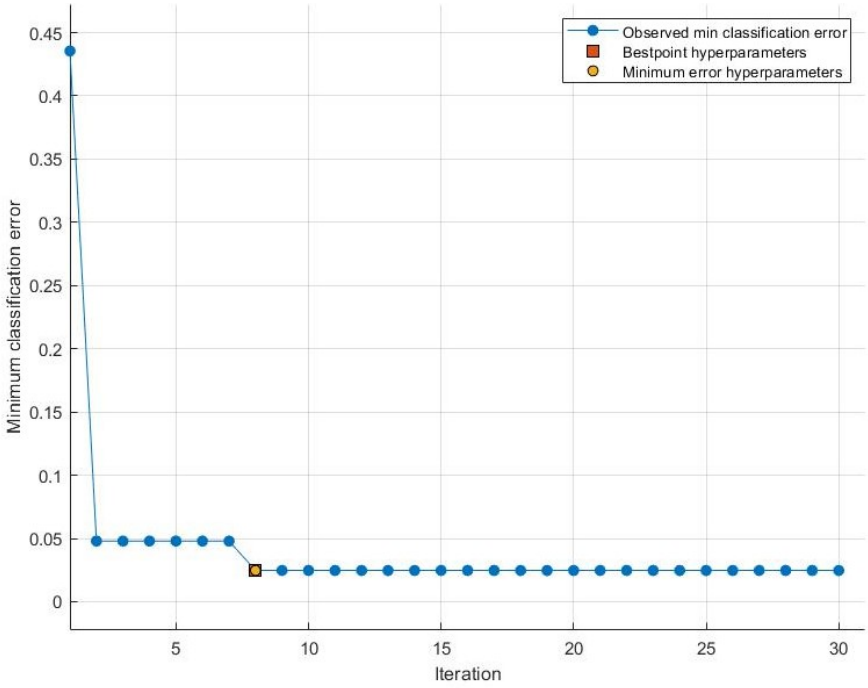

Figure 5.16: SVM Classification Error

times called cross entrpoy). The function is defined as:

$$-\sum_{k} p_k * log_2(p_k) \quad . \tag{5.13}$$

Also in Maximum Deviance Reduction function, the elements that are part of the class k are represented by the variable p_k which stands for the percentage inside a group.

The procedure of splitting keeps going if the conditions are still valid. Of course, a too complex decision tree is not advisable and must be avoided, otherwise there could be risks of overfitting, interpretability and unreliability of predictions.

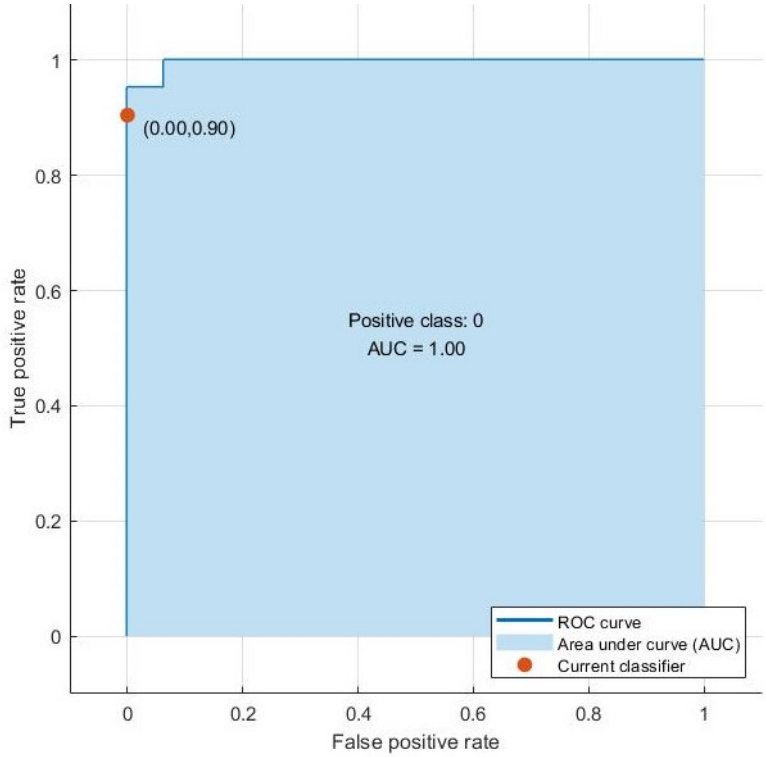

Figure 5.17: SVM ROC Curve

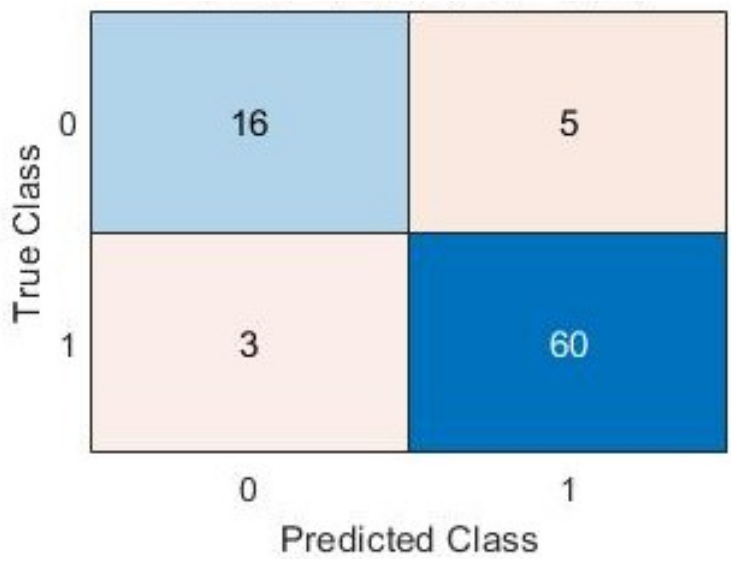

Figure 5.18: Decision Tree Confusion Matrix

Hyperparameter	Range
Maximum Number of Splits	$\{1 - 83\}$
Split Criterion	$\{Gini's\ diversity\ index;$ $Maximum\ Deviance\ Reduction\}$

Table 5.6: Hyperparameter configuration - Decision Tree

Following the application of the random search in Decision Tree algorithm, the optimal configuration obtained consists of 6 number of splits and Gini's diversity index as Split Criterion. This optimal configuration allowed to reach an accuracy of 90.5 %, true positive rate (Sensitivity) of 95.2% and true negative rate (Specificity) of 76,2%.

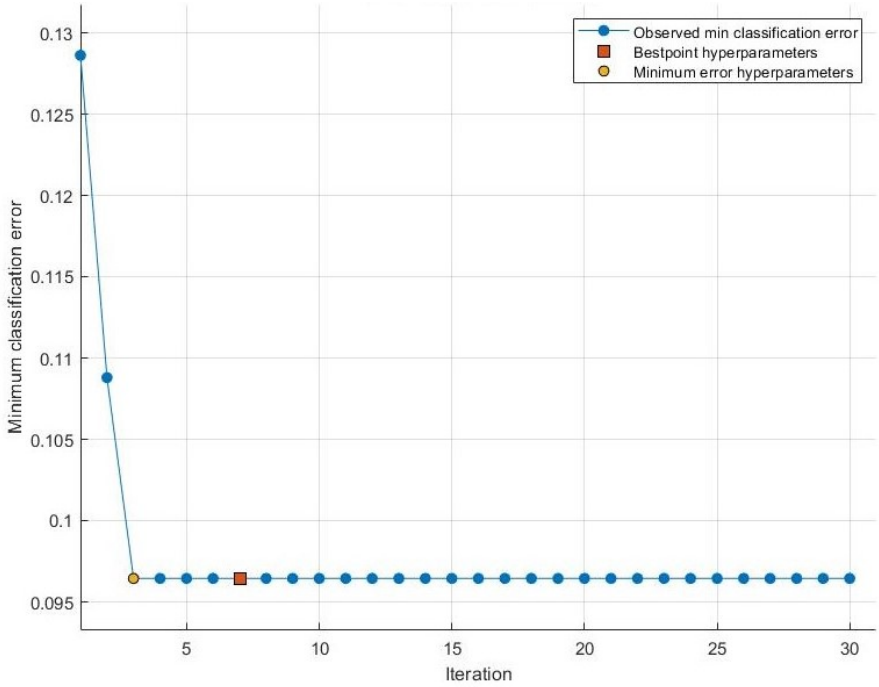

Figure 5.19: Decision Tree Classification Error

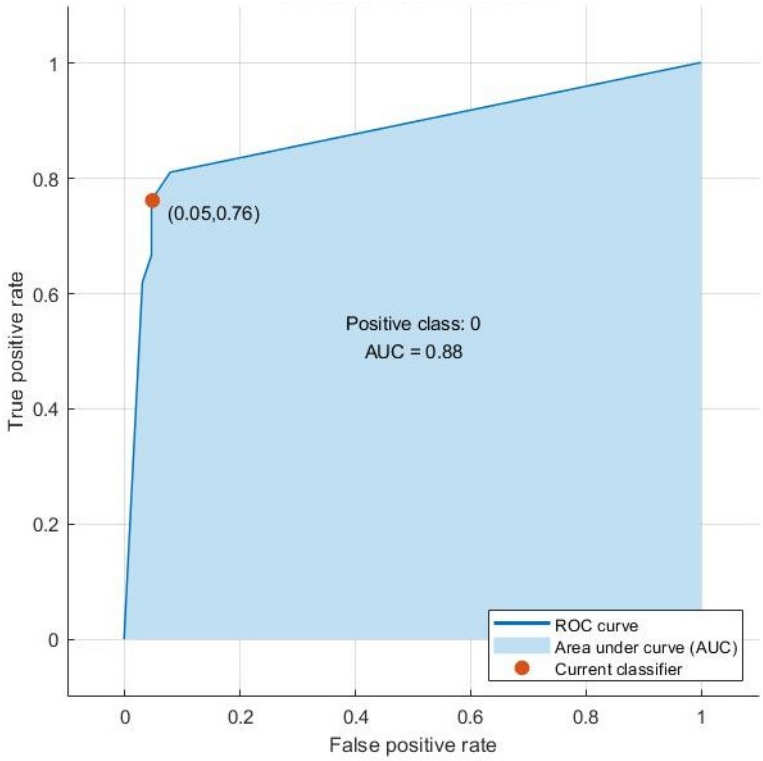

Figure 5.20: Decision Tree ROC Curve

The figure 5.20 shows the ROC curves of the Decision Tree model. Already graphically it is possible to note that the area under the curve is considerably lower than that of the curves in the two previous models. This suggests that the performances cannot be superior or equal to those of the other two algorithms previously explored.

Model	Accuracy	Sensitivity	Specificity
Neural Network Feed-Forward	98,8	98,4	100
SVM	97,6	100	90,5
Decision Tree	90,4	95,2	76,2

Table 5.7: Performances comparison

5.6 Final Remarks

In this chapter a method for the predictive maintenance of three-phase asynchronous electric motors has been proposed. The proposed method explores the acquisition techniques for samples of current measures on the supply power lines. The proposed method is based on an analysis carried out on each single phase. For each motor the three phases are used independently in this method. This approach increases the robustness of the method as a problem that can occur on a single phase can be identified by the algorithm. Furthermore, treating the phases separately allows you to triple the size of the dataset, reducing the risk of overfitting.

The preprocessing for feature extraction is also easily implemented in edge computing devices, this allows the implementation and deployment of the proposed method even in real-world contexts where access to external resources is limited. The machine learning algorithm adopted in this method is a classifier based on a Feed-Forward Neural Network. The simplicity of this model is advantageous for edge-computing deployment.

Finally, in this work a further comparison was made between the performance of the feed-forward network-based classifier and other common classifiers in order to demonstrate the highest performance that a feed-forward-based classifier is capable of achieving. It is evident from the experimental data that the proposed method therefore achieves high levels of accuracy (higher than 98%).

6 Conclusions

In this thesis the issue of predictive maintenance was addressed by means of machine learning algorithms with heterogeneous measures. The fourth industrial revolution of these years is transforming production processes. The connection of devices and the growing computing capacity (cloud computing/edge computing) create new frontiers in the technological transformation of manufacturing companies. The interconnection between industrial devices, the growth of edge computing and the advancement of cloud computing produce large volumes of data. The availability of this growing amount of data has opened up scientific approaches in the design and organization of industrial processes.

The literature on predictive maintenance amply highlights the importance of model-driven approaches in the management of ordinary and extraordinary maintenance tasks, as well as highlighting the impact that downtime times have on production chains. The need for diagnostics of the health status of the systems is justified by the costs caused by the failures, which are given by the sum of the extraordinary maintenance costs and the costs of interruption of the processes dependent on the interrupted systems.

Among the many fields in which the issue of predictive maintenance is felt, three contexts commonly recognized as important in modern industry have been selected: screw compressors, electric motors and magnetic hard disks. These devices all have in common two extremely important factors for which it was decided to shift the focus of the research: the wide diffusion and the high risk (given by the product of the frequency of failures by the cost of failure).

First, a method for fault detection based on a hidden Markov model (HMM), initially conceived for fluid machines and validated on a screw compressor case study, was discussed. The experimental case study at CERN is inserted in a cryogenic plant for which the risk of long periods of downtime raises the attention of the problem. Following a campaign to collect historical measures several months long, seven physical quantities that make up the dataset were taken into consideration. The measures were sparse and asynchronous, so at the beginning of the data processing techniques were proposed and adopted to overcome this

typical problem, after which PCA and clustering were carried out. The clusters thus constituted were used to train the HMM in the first phase, or for fault detection using the trained HMM test procedure. Subsequently, a test procedure and a validation procedure with artificially altered data to evaluate the ability of the method to detect a fault were proposed and adopted. The Wilcoxon test was adopted to evaluate the adherence of the data sequences to the trained model. The test demonstrated that there is statistical significance in validation as the likelihood samples calculated from altered sequences are statistically different from those calculated from nominal sequences.

The second work is a method of predicting failures on hard drives and thereby increasing the availability of large-scale storage services. The method takes advantage of the dataset composed of SMART metrics and presents a technique for automatically labeling the disks and an optimization technique for choosing the hyperparameters of the machine learning classifier. The method is fully automated and is also based on the application of the Regularized Greedy Forest machine learning algorithm. The method allows the system to reliably identify hard drives that have been replaced due to failures by distinguishing them from other maintenance operations that may be required. Experimental tests have shown that the method achieves an accuracy of over 95%, and a false positive rate of less than 5% (with peaks of 1%), a much better result than those proposed in the state-of-the-art literature. Finally, the proposed method poses a solution to the problem of heterogeneity typical of large scale storage centers.

The third work dealt with a new method for predictive maintenance in three-phase asynchronous electric motors. A method is proposed for current measurements, the extraction of features from the acquired samples and for the modeling of a machine learning classifier for the identification of motors close to failure. The method is robust to the risks of overfitting as it treats the current measurements of the phases independently of each other, thus avoiding excessive learning on the individual motors. The optimal configuration of the hyperparameters was achieved by adopting the random search technique and the validation of the efficiency was performed by means of cross validation. The proposed methodology has achieved levels of accuracy above 98%, sensitivity of 98.4% and specificity of 100%.

The entire research work reported in this thesis explored common machine learning techniques for predictive maintenance and proposed new methods for innovative solutions. The methods presented in the previous chapters reach performance levels higher than those of the state of the art, demonstrating that machine learning techniques validly solve the most common needs of the manufacturing world in the field of predictive maintenance.

A Appendix

In support of the thesis, some information is provided to improve understanding and support the reproducibility of the proposed methodologies. In particular, the codes concerning the work on the diagnostics of screw compressors (A.1), on the failure prediction of magnetic hard disks(A.2 and A.3) and on the acquisition device for the diagnostics of three-phase asynchronous motors (A.4) are reported below.

Listings

Listing A.1: Hidden Markov Model Code Example

```
% Select Folder with training data
dir_name = ['C:\Users\fgargiul\Documents\HMM_Gargiulo_v2\datasets\'];

% Select the model name. model_name.mat and model_name.json will be
    created
model_name = 'Model_A_and_B';
file = '..';
repetitions_cluster ='..';

% SET MINOR PARAMETERS ****************************************
% HMM type (3 options)
% 'left to right', 'ergodic-random', 'ergodic-uniform'
hmm_type = 'ergodic-random';

% Filtering thresholds
% either: 0.0 and 0.0, 0.05 and 0.05, 0.1 and 0.1
idle_th = 0.0;
dir_th = 0.0;

% Number of cluster
% either 5, 8, 14.
no_of_clusters =3;

% Number of states
% either 8, 14, 20.
Q =3; % no of states

% Maximum iterations for Baum Welch Algorithm.
% It stops earlier if diff. between two consecutive ll
% is smaller than 1e-4
max_iter = 500;

% PARAMETERS for online recognition
% They are not used in the MATLAB script but by the online recognition
    that
% runs in the mobile brower. Those can be changed in the .json file
% afterwards, too.
% CREATE .mat MODEL  ****************************************
type = hmm_type; % model type
O = no_of_clusters; % no of output symbols
```

```
%initial guess of parameters
if strcmp(type,'ergodic-uniform')
    prior1 = ones(Q,1) * (1/Q);
    transmat1 = ones(Q, Q)* (1/Q);
    obsmat1 = ones(Q, O)* (1/O);
elseif strcmp(type,'ergodic-random')
    prior1 = mk_stochastic(rand(Q,1));
    transmat1 = mk_stochastic(rand(Q,Q));
    obsmat1 = mk_stochastic(rand(Q,O));
else
    % left to right model
    prior1 = zeros(Q,1);
    prior1(1)= 1;
    % transistion limited to current and next state (one step l2r)
    transmat1 = zeros(Q);
    for i = 1:(Q-1)
        transmat1(i,i) = 0.5;
        transmat1(i,i+1) = 0.5;
    end
    transmat1(Q, Q) = 1;
    obsmat1 = ones(Q, O)* (1/O);
end

% Variable will contain all model parameters.
all_models = cell(size(1));

    % Read in single gesture and cluster the data.
    % f contains info about every single sample
%     regex = ['media_1hour.csv'];
    regex = ['TIMBER_Media_' file '.csv'];
    [cl_values, no_of_files, f] = prepare_cluster_v2(regex, dir_name, ...
        idle_th , dir_th);

    % Apply kmeans algorithm to find cluster centers.
    [~,C, sumd] = kmeans(cl_values, no_of_clusters, ...
        'display', 'final', 'replicates', 5);

    % Dendrogram plot
    tree = linkage(cl_values,'centroid');
    figure(1)
    dendrogram(tree)
    hold on;

%     Assign samples to cluster values
    [f, ~] = assign_to_cluster(C, cl_values, f);

    % Select 80% of data for training and 20% to find average likelihood.
    k = 5; % how many folds i want
    N = size(f,1); % total number of observations

    % No cross validation is performed here. Used to create random
    % indices onyly
    indices = crossvalind('Kfold',N,k);

    % Split orignial set
    test = (indices == 1) ; % which points are in the test set
    train = ~test; % all points that are NOT in the test set
    test_set = f(test,:);
    train_set = f(train,:);

    % in this context, models contains only one model
    [ model ] = train_data(train_set, '1', prior1, transmat1, obsmat1,
        max_iter);

    model(1).cluster_centers = C;
```

```
    model(1).idle_th = idle_th;
    model(1).dir_th = dir_th;
    model(1).hmm_type = hmm_type;

%     model(1).f_sample = f_sample;
    [~, ~, prob_table] = test_data( test_set,  model(1));

    % check that only the correct gesture was tested.
     assert(all([prob_table{:, 3}] == '1'));
    % Compute the tested_ll_mean
    % need to exclude the -Inf, as it will get biased o/w.
     tmp = [prob_table{:, 4}];
     model(1).tested_ll_mean = mean(tmp([prob_table{:, 4}] ~= -Inf));
     model;
save(['C:\Users\fgargiul\Documents\HMM_Gargiulo_v2\Output\' model_name],
    'all_models');

% CREATE .json MODEL  ***************************************
%%
% Seperate filter
convert_model_to_json(model_name, model, '1');
```

Listing A.2: Automatized Labeling and Features Extraction

```
######################################################################## This
    script loads the SMART data from your folder and assigns a value
    Broken {0,1} to each measure
# and it splits the dataset in measures set for the train/test and a set
    for the prediction
# Before run this script, make sure you run it in the correct folder with
     getwd() and setwd()
########################################################################

dataset_initialization <- function(hd_type,num_models, num_months,
    lastdays, pathscript, pathdata)
{


  library(data.table)
  library(ggplot2)
  library(lubridate)
  library(fpc)
  library(dplyr)
  cat("last_days_measures_for_trainingset", lastdays, "\n")



  ##############################################
  setwd(pathscript)
  ##############################################
  #missing parameters are substituted
  if(missing(hd_type)) hd_type<-"HDD"
  if(missing(num_models)) num_models<-5
  if(missing(num_months)) num_months<-28
  if(missing(pathdata)) pathdata<-"/mypath/data"
  if(missing(pathscript)) pathscript<-"/mypath/scripts"
  ##############################################
  source("./renaming_values_func.R")
  source("./removing_columns_func.R")
  add.months= function(date,n) seq(date, by = paste (n, "months"), length
       = 2)[2]

  ##############################################
  #setwd(pathdata)
  unlink("/mypath/dataset_initialized/", recursive=TRUE)
```

```
dir.create(file.path(paste0(pathdata,"/dataset_initialized")),
    showWarnings = FALSE)
load(paste0(pathdata,"/disk/disk-all.rda"))

##########################################

sd<-sd[order(mtime)]
sd<-sd[,dyear:=NULL]

if (hd_type=="HDD"){

  sd<-sd[type=="HDD"]
  invalid_vendors <- c("####","####")
  sd <- sd[ !(vendor %in% invalid_vendors)]

} else if (hd_type=="SSD"){

  sd<-sd[type=="SSD"]
  invalid_vendors <- c("####","####")
  sd <- sd[ !(vendor %in% invalid_vendors)]

}

cols_na_omit<-c("mtime","serial","host", "mtime")
sd<-na.omit(sd, cols= cols_na_omit, invert=FALSE)


#attaching last date per serial
a<-sd[, .(lastdate_serial=max(mtime)), by=serial]

setkey(sd,serial)
setkey(a,serial)
sd<-merge(sd,a, all=TRUE)
remove(a)
colnames(sd)[colnames(sd)=="i"] <- "lastdate_serial"
sd$lastdate_serial<-as.Date(sd$lastdate_serial, "CET")

#attaching last date per serial completed

#### tagging brokens

sd<-na.omit(sd, cols= "mtime", invert=FALSE)

d1<-as.Date("2018-08-01")
d2<-as.Date("2018-07-01")

i=0
numhdbroken<-c()
numdisk<-c()
list_broken_serial<-data.table()

for(i in 1:num_months){

  d2 <- add.months(d2,1)
  d1<-add.months(d2,1)
  day(d1) <- day(d1) - 1

  lhsd<-sd[, .(lastdate=max(mtime)), by=host]
  lhsd<-lhsd[as.Date(lastdate, "CET")<=d1 & as.Date(lastdate, "CET")>=
      d2]

  lhd<-sd[!(host %in% lhsd$host)]
  ld<-lhd[, .(lastdate=max(mtime)), by=serial]
  ld<-ld[as.Date(lastdate, "CET")<=d1 & as.Date(lastdate, "CET")>=d2]

  list_broken_serial<-rbind(list_broken_serial,ld)
```

```
}
#################################################################
sd<-sd[order(mtime)]
#################################################################
### splitting in (last day and second last day ) & all the rest
sd_predict<-sd[as.Date(mtime,"CET")==max(sd$lastdate_serial)]
sd<-sd[as.Date(mtime,"CET")<max(sd$lastdate_serial)-20]

#################################################################
sd$broken<-0
sd[serial %in% list_broken_serial$serial]$broken<-1 #to be postponet at
      line 121

remove(lhd, lhsd, list_broken_serial,ld, d1, d2, i, numdisk,
     numhdbroken)

########Tagging completed ########################

####### removing fake replacements ##############
#
#a<-sd[broken==1]

a<-sd[,.SD[.N], by=serial]
a<-a[,.N, by=c("host","lastdate_serial")]
colnames(a)[colnames(a)=="i"] <- "num_deseappering_hd"

host_lastdate_serial<-c("host","lastdate_serial")
setkeyv(sd,host_lastdate_serial)
setkeyv(a,host_lastdate_serial)
sd<-merge(sd,a, all=TRUE)

remove(a)

print(nrow(sd[broken==1]))
sd[N>1]$broken<-0
print(nrow(sd[broken==1]))
#####################################################


sd<-sd[,N:=NULL]
sd<-sd[,lastdate_serial:=NULL]
sd_predict<-sd_predict[,lastdate_serial:=NULL]
sd<-sd[order(mtime)]
#####################################################

a<-sd[, .(lastdate_serial=max(mtime)), by=serial]
setkey(sd,serial)
setkey(a,serial)
sd<-merge(sd,a, all=TRUE)
remove(a)
colnames(sd)[colnames(sd)=="i"] <- "lastdate_serial"
sd$lastdate_serial<-as.Date(sd$lastdate_serial, "CET")
sd<-sd[order(mtime)]

#let's keep last days we need
sd<-sd[as.Date(mtime,"CET")> (lastdate_serial-lastdays)] #da verificare
sd<-sd[,lastdate_serial:=NULL]

#################################################################
sd<-sd[,lastdate_serial:=NULL]
print(nrow(sd[broken==1][,.N,by=serial]))

#################################
#Let's extract sets
model_list<-sd[,.N,by=model]
```

```
  model_list<-model_list[order(-N)]
  model_list<-na.omit(model_list, cols="model", invert=FALSE)
  model_list<-model_list[1:num_models,]

  num_broken_by_model<-sd[broken==1,.N,by=c("model","serial")]
  # print(num_broken_by_model)
  a<-num_broken_by_model[, .N, by=model]
  colnames(a)[colnames(a)=="N"] <- "num_broken"
  setkey(model_list,model)
  setkey(a,model)
  model_list<-merge(model_list,a, all=TRUE)
  model_list<-model_list[order(-N)]
  remove(a)


  # model_list$broken<-nrow(as.data.table(num_broken_by_model))


  for (i in 1:num_models) {
    m<-model_list[i]$model
    m<-as.character(m)
    sub_sd<-sd[model==m]
    sub_sd_predict<-sd_predict[model==m]

    ##############################
    #renaming & removing column in training set has been performed
        remotely because of dimensions
    sub_sd<-renaming_values(sub_sd, TRUE)
    sub_sd<-removing_columns(sub_sd)
    sub_sd[,serial:=NULL]
    sub_sd<-sub_sd[base::sample(nrow(sub_sd))]
    ##############################

    dir.create(file.path(paste0(pathdata,"/dataset_initialized/", m)),
        showWarnings = FALSE)
    write.csv(sub_sd,paste0(pathdata,"/dataset_initialized/",m,"/train_",
        m, ".csv"), row.names = FALSE)
    write.csv(sub_sd_predict,paste0(pathdata,"/dataset_initialized/",m,"/
        prediction_",m, ".csv"), row.names = FALSE)

  }

  write.csv(model_list[1:num_models,], paste0(pathdata,"/production_
      dataset_initialized/models_list.csv"), row.names = FALSE)
}
```

Listing A.3: Training Testing RGF

```
reticulate::use_python("/mypath_to_conda/anaconda3/bin/python", required
    = T)
library(data.table)
library(reticulate)
library(doParallel)

## make sure the python packages are actually installed
stopifnot(reticulate::py_module_available("rgf.sklearn"))

## before initialising RGF
library(RGF)


train_test_pred_precision_v2 <- function(num_model,iteration){

  list_models          <- fread("./production_dataset_initialized/models_
      list.csv")
```

```
list_of_candidates <- data.table()
results            <- data.table()

for(i in 1:num_model) {

  m <- list_models[i]$model
  pred_dir <- file.path("prediction" , iteration ,m)

  if (!dir.exists(pred_dir)){
    dir.create(pred_dir, recursive = TRUE) # file.path should not have
        a trailing "/"
  }

  path <- paste0("dataset_finalized/", m,"_",iteration ,"/")
  HD_trainset_data          <- data.matrix(fread(paste0(path, "trainset
      _feature.csv" )))
  HD_trainset_target        <- fread(paste0(path, "trainset_target.csv"
      ))[[1]]
  HD_trainset_broken_X      <- data.matrix(fread(paste0(path, "testset_
      feature_broken.csv" )))
  HD_trainset_broken_Y      <- fread(paste0(path, "testset_target_
      broken.csv" ))[[1]]
  HD_trainset_healthy_X     <- data.matrix(fread(paste0(path, "testset_
      feature_healthy.csv" )))
  HD_trainset_healthy_Y     <- fread(paste0(path, "testset_target_
      healthy.csv" ))[[1]]
  HD_trainset_tobepredicted <- data.matrix(fread(paste0(path, "testset_
      predict.csv")))

  rgf <- RGF_Classifier$new(max_leaf=5000,
                            algorithm="RGF_Opt",
                            test_interval=100,
                            #reg_depth=200,
                            l2=0.1,
                            loss="Expo")


  rgf$fit(HD_trainset_data, HD_trainset_target)

  score_train   <-  rgf$score(HD_trainset_data, HD_trainset_target)
  # score_healthy <-  rgf$score(HD_trainset_healthy_X, HD_trainset_
      healthy_Y)
  # score_broken  <-  rgf$score(HD_trainset_broken_X, HD_trainset_
      broken_Y)
  score_proba_healthy  <-  rgf$predict_proba(HD_trainset_healthy_X)
  score_proba_broken   <-  rgf$predict_proba(HD_trainset_broken_X)

  score_proba_healthy <- as.data.table(score_proba_healthy)
  score_healthy<- nrow(score_proba_healthy[V1>0.25]) / nrow(score_proba
      _healthy)
  score_proba_broken <- as.data.table(score_proba_broken)
  score_broken <- nrow(score_proba_broken[V2>0.75]) / nrow(score_proba_
      broken)


  write.csv(score_proba_healthy,paste0("prediction/",iteration ,"/",m,"/
      score_proba_healthy",".csv"), row.names = FALSE)
  write.csv(score_proba_broken,paste0("prediction/",iteration ,"/",m,"/
      score_proba_broken",".csv"), row.names = FALSE)
  a<- rgf$dump_model()
  b<- rgf$get_params()
  c<- rgf$feature_importances()
  rgf$save_model(paste0("prediction/",iteration ,"/",m,"/model.json"))

  ###################################
```

```
    # prediction
    ##################################

    guess_set_serial              <- fread(paste0("dataset_finalized/",m
        ,"_",iteration,"/serial_predict.csv"))

    guess_set_serial$model        <- m
    colnames(guess_set_serial)    <- c("serial", "model")
    guess_set_serial$candidates   <- rgf$predict(HD_trainset_
        tobepredicted)
    a                             <- rgf$predict_proba(HD_trainset_
        tobepredicted)
    guess_set_serial$healthy_prob <- a[,1]
    guess_set_serial$broken_prob  <- a[,2]

    ##################################
    # summaries
    ##################################

    list_of_candidates <- rbind(list_of_candidates, guess_set_serial[
        candidates==1 ])
    print(list_of_candidates[,.N, by=model])

    summary <- fread(paste0("dataset_finalized/", m,"_",iteration, "/
        summary.csv"))

    tp <- score_broken * summary$N_Test_B
    tn <- score_healthy * summary$N_Test_H
    fp <- summary$N_Test_H*(1-score_healthy)
    fn <- summary$N_Test_B*(1-score_broken)
    pr <- tp/(tp+fp)
    rc <- tp/(tp+fn)
    fpr<- fp/(fp+tn)
    plr<- rc/fpr

    results<-rbind(results,
                   data.table(model              = as.character(m),
                              ratio              = summary$ratio,
                              N_Train_H          = summary$N_Train_H,
                              N_Train_B          = summary$N_Train_B,
                              Score_H            = score_healthy,
                              Score_B            = score_broken,
                              N_Test_H           = summary$N_Test_H,
                              N_Test_B           = summary$N_Test_B,
                              N_pred_set         = summary$N_pred_set,
                              # N_candid           = nrow(guess_set_
                                  serial[ candidates==1 ]),
                              Expected_false_neg = (1 - score_broken) * (
                                  summary$N_pred_set),
                              Expected_false_pos = (1 - score_healthy) *
                                  (summary$N_pred_set),
                              Precision          = pr,
                              Recall             = rc,
                              F_Score            = (2 * pr * rc) / (pr +
                                  rc),
                              FPR                = fpr,
                              PLR                = plr
                   ))

    write.csv(guess_set_serial,paste0("prediction/",iteration,"/",m,"/
        predicted",m,".csv"), row.names = FALSE)

  }
  write.csv(results,paste0("prediction","/",iteration,"/results.csv"),
      row.names = FALSE)
}
```

```
num_model<-7
pathscript<-"/mypath/scripts"
source(paste0(pathscript, "/production_dataset_finalization.R")) #in the
    same folder there is the the script called in the line below
pathdata<-"/mypath/data"
numCores <- detectCores()-2
registerDoParallel(numCores)
unlink(paste0(pathdata, "/dataset_finalized/"), recursive=TRUE)
unlink(paste0(pathdata, "/prediction/"), recursive=TRUE)
unlink(paste0(pathdata, "/tent_all/"), recursive=TRUE)

foreach (i= 1:9) %dopar% {
 # for(i in 1:5) {
  print(i)
  iteration<-i
  dataset_finalization(num_model,0.95,28, pathdata,iteration)
  setwd(pathdata)
  train_test_pred_precision_v2(num_model,iteration)

  dest_dir <- file.path("tent_all") # file.path should not have a
      trailing "/"

  if (!dir.exists(dest_dir)) {
    dir.create(dest_dir, recursive = TRUE)
  }

  file.copy(from = paste0( "prediction/",iteration), to = dest_dir,
            recursive = TRUE, copy.date = TRUE)

}

unlink("/tmp/rgf", recursive=TRUE)
```

Listing A.4: Data Acquisition Device - Main.C

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************

  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************

  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2021 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under Ultimate Liberty
      license
  * SLA0044, the "License"; You may not use this file except in
      compliance with
  * the License. You may obtain a copy of the License at:
  *                             www.st.com/SLA0044
  *
  ******************************************************************************

  */
/* USER CODE END Header */
/* Includes
    ------------------------------------------------------------------*/
#include "main.h"
#include "usb_device.h"
```

```
/* Private includes
----------------------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef
-----------------------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define
------------------------------------------------------------------------*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */

/* Private macro
-------------------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables
---------------------------------------------------------------------*/
ADC_HandleTypeDef hadc1;
DMA_HandleTypeDef hdma_adc1;

I2C_HandleTypeDef hi2c1;

I2S_HandleTypeDef hi2s2;
I2S_HandleTypeDef hi2s3;

TIM_HandleTypeDef htim3;

UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */

uint16_t measure_buffer[60000];
uint8_t command[1];
/* USER CODE END PV */

/* Private function prototypes
-----------------------------------------------------------------*/
void SystemClock_Config(void);
void PeriphCommonClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_ADC1_Init(void);
static void MX_I2C1_Init(void);
static void MX_I2S2_Init(void);
static void MX_I2S3_Init(void);
static void MX_TIM3_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_DMA_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code
---------------------------------------------------------------------*/
/* USER CODE BEGIN 0 */

void get_samples(void){
```

```
        ADC1->CR2 |= 1|(1<<8);
        DMA2_Stream0->M0AR=measure_buffer;
        DMA2_Stream0->PAR=&ADC1->DR;
        DMA2_Stream0->NDTR=60000;
        DMA2_Stream0->CR|=1;
        HAL_GPIO_TogglePin(LD4_GPIO_Port, LD4_Pin);
        TIM3->CR1 |= 1;

        //        HAL_ADC_Start_DMA(&hadc1, (uint32_t*)measure_buffer,
            40000);
        while((DMA2->LISR &(0x00000020))==0);
        DMA2->LIFCR |= 0x00000020 ;
        TIM3->CR1 = 0;
        HAL_GPIO_TogglePin(LD4_GPIO_Port, LD4_Pin);

}

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration
       --------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the
      Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

/* Configure the peripherals common clocks */
  PeriphCommonClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_ADC1_Init();
  MX_I2C1_Init();
  MX_I2S2_Init();
  MX_I2S3_Init();
  MX_TIM3_Init();
  MX_USART2_UART_Init();

  MX_USB_DEVICE_Init();
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

  /* Infinite loop */
```

```
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */


            if(command[0] =='a'){
                                    CDC_Transmit_FS((uint8_t*)measure_buffer,
                                        sizeof(measure_buffer));
                    }
            command[0]='0';

            uint32_t dimension[1];
            char token[1]="#";
            int count=0;
            dimension[0]= (uint32_t)(sizeof(measure_buffer));

            uint8_t ack[4];
            ack[0]=0;
            ack[1]=0;
            ack[2]=0;
            ack[3]=0;

            while ((uint32_t)((ack[3]<<24) |(ack[2]<<16) |(ack[1]<<8) | (
                ack[0])) !=dimension[0]){

                HAL_UART_Transmit(&huart2, (uint8_t*)(token),1, 1000);
                HAL_Delay(10);
                HAL_UART_Transmit(&huart2, (uint8_t*)(dimension),sizeof(
                    dimension), 1000);
                HAL_UART_Receive(&huart2, (uint8_t*)(ack), 4, 1000);
                HAL_Delay(100);
            }
            ack[0]=1;
            ack[1]=1;
            get_samples();
            while (count<dimension[0]) {
                HAL_UART_Transmit(&huart2, (uint8_t*)measure_buffer +
                    count, 100, 1000);
                count+=100;
                HAL_Delay(50);
            }


            HAL_Delay(2000);

  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
  /** Initializes the RCC Oscillators according to the specified
```

```
      parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 4;
  RCC_OscInitStruct.PLL.PLLN = 96;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 4;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_3) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief Peripherals Common Clock Configuration
  * @retval None
  */
void PeriphCommonClock_Config(void)
{
  RCC_PeriphCLKInitTypeDef PeriphClkInitStruct = {0};

  /** Initializes the peripherals clock
  */
  PeriphClkInitStruct.PeriphClockSelection = RCC_PERIPHCLK_I2S;
  PeriphClkInitStruct.PLLI2S.PLLI2SN = 200;
  PeriphClkInitStruct.PLLI2S.PLLI2SM = 5;
  PeriphClkInitStruct.PLLI2S.PLLI2SR = 2;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief ADC1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_ADC1_Init(void)
{

  /* USER CODE BEGIN ADC1_Init 0 */

  /* USER CODE END ADC1_Init 0 */

  ADC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN ADC1_Init 1 */
```

```
  /* USER CODE END ADC1_Init 1 */
  /** Configure the global features of the ADC (Clock, Resolution, Data
      Alignment and number of conversion)
  */
  hadc1.Instance = ADC1;
  hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
  hadc1.Init.Resolution = ADC_RESOLUTION_12B;
  hadc1.Init.ScanConvMode = ENABLE;
  hadc1.Init.ContinuousConvMode = DISABLE;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_RISING;
  hadc1.Init.ExternalTrigConv = ADC_EXTERNALTRIGCONV_T3_TRGO;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 3;
  hadc1.Init.DMAContinuousRequests = ENABLE;
  hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
  }
  /** Configure for the selected ADC regular channel its corresponding
      rank in the sequencer and its sample time.
  */
  sConfig.Channel = ADC_CHANNEL_11;
  sConfig.Rank = 1;
  sConfig.SamplingTime = ADC_SAMPLETIME_28CYCLES;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /** Configure for the selected ADC regular channel its corresponding
      rank in the sequencer and its sample time.
  */
  sConfig.Channel = ADC_CHANNEL_12;
  sConfig.Rank = 2;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /** Configure for the selected ADC regular channel its corresponding
      rank in the sequencer and its sample time.
  */
  sConfig.Channel = ADC_CHANNEL_15;
  sConfig.Rank = 3;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC1_Init 2 */

  /* USER CODE END ADC1_Init 2 */

}

/**
  * @brief I2C1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_I2C1_Init(void)
{

  /* USER CODE BEGIN I2C1_Init 0 */

  /* USER CODE END I2C1_Init 0 */
```

```
  /* USER CODE BEGIN I2C1_Init 1 */

  /* USER CODE END I2C1_Init 1 */
  hi2c1.Instance = I2C1;
  hi2c1.Init.ClockSpeed = 100000;
  hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
  hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN I2C1_Init 2 */

  /* USER CODE END I2C1_Init 2 */

}

/**
  * @brief I2S2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_I2S2_Init(void)
{

  /* USER CODE BEGIN I2S2_Init 0 */

  /* USER CODE END I2S2_Init 0 */

  /* USER CODE BEGIN I2S2_Init 1 */

  /* USER CODE END I2S2_Init 1 */
  hi2s2.Instance = SPI2;
  hi2s2.Init.Mode = I2S_MODE_MASTER_TX;
  hi2s2.Init.Standard = I2S_STANDARD_PHILIPS;
  hi2s2.Init.DataFormat = I2S_DATAFORMAT_16B;
  hi2s2.Init.MCLKOutput = I2S_MCLKOUTPUT_DISABLE;
  hi2s2.Init.AudioFreq = I2S_AUDIOFREQ_96K;
  hi2s2.Init.CPOL = I2S_CPOL_LOW;
  hi2s2.Init.ClockSource = I2S_CLOCK_PLL;
  hi2s2.Init.FullDuplexMode = I2S_FULLDUPLEXMODE_ENABLE;
  if (HAL_I2S_Init(&hi2s2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN I2S2_Init 2 */

  /* USER CODE END I2S2_Init 2 */

}

/**
  * @brief I2S3 Initialization Function
  * @param None
  * @retval None
  */
static void MX_I2S3_Init(void)
{

  /* USER CODE BEGIN I2S3_Init 0 */
```

```
  /* USER CODE END I2S3_Init 0 */

  /* USER CODE BEGIN I2S3_Init 1 */

  /* USER CODE END I2S3_Init 1 */
  hi2s3.Instance = SPI3;
  hi2s3.Init.Mode = I2S_MODE_MASTER_TX;
  hi2s3.Init.Standard = I2S_STANDARD_PHILIPS;
  hi2s3.Init.DataFormat = I2S_DATAFORMAT_16B;
  hi2s3.Init.MCLKOutput = I2S_MCLKOUTPUT_ENABLE;
  hi2s3.Init.AudioFreq = I2S_AUDIOFREQ_96K;
  hi2s3.Init.CPOL = I2S_CPOL_LOW;
  hi2s3.Init.ClockSource = I2S_CLOCK_PLL;
  hi2s3.Init.FullDuplexMode = I2S_FULLDUPLEXMODE_DISABLE;
  if (HAL_I2S_Init(&hi2s3) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN I2S3_Init 2 */

  /* USER CODE END I2S3_Init 2 */

}

/**
  * @brief TIM3 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM3_Init(void)
{

  /* USER CODE BEGIN TIM3_Init 0 */

  /* USER CODE END TIM3_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};

  /* USER CODE BEGIN TIM3_Init 1 */

  /* USER CODE END TIM3_Init 1 */
  htim3.Instance = TIM3;
  htim3.Init.Prescaler = 0;
  htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim3.Init.Period = 1919;
  htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_UPDATE;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_ENABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) !=
      HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM3_Init 2 */
```

```
  /* USER CODE END TIM3_Init 2 */

}

/**
  * @brief USART2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART2_UART_Init(void)
{

  /* USER CODE BEGIN USART2_Init 0 */

  /* USER CODE END USART2_Init 0 */

  /* USER CODE BEGIN USART2_Init 1 */

  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */

  /* USER CODE END USART2_Init 2 */

}

/**
  * Enable DMA controller clock
  */
static void MX_DMA_Init(void)
{

  /* DMA controller clock enable */
  __HAL_RCC_DMA2_CLK_ENABLE();

  /* DMA interrupt init */
  /* DMA2_Stream0_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA2_Stream0_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA2_Stream0_IRQn);

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOE_CLK_ENABLE();
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
```

```
__HAL_RCC_GPIOA_CLK_ENABLE();
__HAL_RCC_GPIOB_CLK_ENABLE();
__HAL_RCC_GPIOD_CLK_ENABLE();

/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_RESET);

/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(OTG_FS_PowerSwitchOn_GPIO_Port,
      OTG_FS_PowerSwitchOn_Pin, GPIO_PIN_SET);

/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(GPIOD, LD4_Pin|LD3_Pin|LD5_Pin|LD6_Pin
                         |Audio_RST_Pin, GPIO_PIN_RESET);

/*Configure GPIO pin : DATA_Ready_Pin */
GPIO_InitStruct.Pin = DATA_Ready_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(DATA_Ready_GPIO_Port, &GPIO_InitStruct);

/*Configure GPIO pin : PE3 */
GPIO_InitStruct.Pin = GPIO_PIN_3;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOE, &GPIO_InitStruct);

/*Configure GPIO pins : INT1_Pin INT2_Pin MEMS_INT2_Pin */
GPIO_InitStruct.Pin = INT1_Pin|INT2_Pin|MEMS_INT2_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_EVT_RISING;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(GPIOE, &GPIO_InitStruct);

/*Configure GPIO pin : OTG_FS_PowerSwitchOn_Pin */
GPIO_InitStruct.Pin = OTG_FS_PowerSwitchOn_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(OTG_FS_PowerSwitchOn_GPIO_Port, &GPIO_InitStruct);

/*Configure GPIO pin : PA0 */
GPIO_InitStruct.Pin = GPIO_PIN_0;
GPIO_InitStruct.Mode = GPIO_MODE_EVT_RISING;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

/*Configure GPIO pins : SPI1_SCK_Pin SPI1_MISO_Pin SPI1_MOSI_Pin */
GPIO_InitStruct.Pin = SPI1_SCK_Pin|SPI1_MISO_Pin|SPI1_MOSI_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
GPIO_InitStruct.Alternate = GPIO_AF5_SPI1;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

/*Configure GPIO pins : LD4_Pin LD3_Pin LD5_Pin LD6_Pin
                        Audio_RST_Pin */
GPIO_InitStruct.Pin = LD4_Pin|LD3_Pin|LD5_Pin|LD6_Pin
                      |Audio_RST_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

/*Configure GPIO pin : OTG_FS_OverCurrent_Pin */
GPIO_InitStruct.Pin = OTG_FS_OverCurrent_Pin;
```

```
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(OTG_FS_OverCurrent_GPIO_Port, &GPIO_InitStruct);

}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return
      state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line
      number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line
      number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file,
         line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF
    FILE****/
```

Bibliography

[1] Expectation-maximization. https://github.com/rezaahmadzadeh/Expectation-Maximization.

[2] Motore asincrono trifase. https://www.mcurie.edu.it/files/gargano.pierangelo/SISTEMI/MOTORE_ASINCRONO.pdf.

[3] Muhammad Shahzad Afzal, Wen Tan, and Tongwen Chen. Process monitoring for multimodal processes with mode-reachability constraints. *IEEE Transactions on Industrial Electronics*, 64(5):4325–4335, 2017.

[4] Eric Alata, João Antunes, Mohamed Kaâniche, Nuno Neves, Vincent Nicomette, and Paulo Veríssimo. Critical utility infrastructural resilience project acronym: Crutial. 01 2022.

[5] A. Ballarino. Development of superconducting links for the Large Hadron Collider machine. *Superconductor Science and Technology*, 27(4), 2014.

[6] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

[7] Ramaprasad Bhar and Shigeyuki Hamori. *Hidden Markov models: applications to financial economics*, volume 40. Springer Science & Business Media, 2004.

[8] Mirela Madalina Botezatu, Ioana Giurgiu, Jasmina Bogojeska, and Dorothea Wiesmann. Predicting disk replacement towards reliable data centers. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 39–48, 2016.

[9] Giovanni Bucci, Fabrizio Ciancetta, and Edoardo Fiorucci. Apparatus for online continuous diagnosis of induction motors based on the sfra technique. *IEEE Transactions on Instrumentation and Measurement*, 69(7):4134–4144, 2019.

[10] Raffaele Cioffi, Marta Travaglioni, Giuseppina Piscitelli, Antonella Petrillo, and Fabio De Felice. Artificial intelligence and machine learning applications

in smart production: Progress, trends, and directions. *Sustainability*, 12(2), 2020.

[11] Shantanu Datta and Shibayan Sarkar. A review on different pipeline fault detection methods. *Journal of Loss Prevention in the Process Industries*, 41:97–106, 2016.

[12] Aniello De Santo, Antonio Galli, Michela Gravina, Vincenzo Moscato, and Giancarlo Sperlì. Deep learning for hdd health assessment: An application based on lstm. *IEEE Transactions on Computers*, 71(1):69–80, 2020.

[13] D Duellmann. Big data: Challenges and perspectives. *Grid and Cloud Computing: Concepts and Practical Applications*, 192:153, 2016.

[14] Duellmann, Dirk and Portabales, Alfonso. Disk failures in the eos setup at cern - a first systematic look at 1 year of collected data. *EPJ Web Conf.*, 214:04046, 2019.

[15] A. Fujiwara and N. Sakurai. Experimental analysis of screw compressor noise and vibration. In *Proc. of International Compressor Engineering Conference*, 1986.

[16] Purushottam Gangsar and Rajiv Tiwari. Comparative investigation of vibration and current monitoring for prediction of mechanical and electrical faults in induction motor based on multiclass-support vector machine algorithms. *Mechanical Systems and Signal Processing*, 94:464–481, 2017.

[17] S. Ganguly, A. Consul, A. Khan, B. Bussone, J. Richards, and A. Miguel. A practical approach to hard disk failure prediction in cloud platforms: Big data model for failure management in datacenters. In *2016 IEEE Second International Conference on Big Data Computing Service and Applications (BigDataService)*, pages 105–116, March 2016.

[18] Federico Gargiulo, Dirk Duellmann, Pasquale Arpaia, and Rosario Schiano Lo Moriello. Predicting hard disk failure by means of automatized labeling and machine learning approach. *Applied Sciences*, 11(18):8293, 2021.

[19] M Ge, R Du, and Y Xu. Hidden markov model based fault diagnosis for stamping processes. *Mechanical Systems and Signal Processing*, 18(2):391–408, 2004.

[20] Edmund A Gehan. A generalized wilcoxon test for comparing arbitrarily singly-censored samples. *Biometrika*, 52(1-2):203–224, 1965.

[21] Mark Haarman, Michel Mulders, and Costas Vassiliadis. Predictive maintenance 4.0: predict the unpredictable. *PwC and Mainnovation*, 4, 2017.

[22] Jim Hatfield. Smart attribute annex. *Seagate Technology*, 2005.

[23] Gordon F Hughes, Joseph F Murray, Kenneth Kreutz-Delgado, and Charles Elkan. Improved disk-drive failure warnings. *IEEE transactions on reliability*, 51(3):350–357, 2002.

[24] Rolf Isermann. Supervision, fault-detection and fault-diagnosis methods—an introduction. *Control engineering practice*, 5(5):639–652, 1997.

[25] Rie Johnson and Tong Zhang. Learning nonlinear functions using regularized greedy forest. *IEEE transactions on pattern analysis and machine intelligence*, 36(5):942–954, 2013.

[26] Petr Kacor, Petr Bernat, and Petr Moldrik. Utilization of two sensors in offline diagnosis of squirrel-cage rotors of asynchronous motors. *Energies*, 14(20), 2021.

[27] Jong-Hyun Lee, Jae-Hyung Pack, and In-Soo Lee. Fault diagnosis of induction motor using convolutional neural network. *Applied Sciences*, 9(15):2950, 2019.

[28] Jinbo Li, Witold Pedrycz, and Iqbal Jamal. Multivariate time series anomaly detection: A framework of hidden markov models. *Applied Soft Computing*, 2017.

[29] Lingxin Li and Chris K Mechefske. Induction motor fault detection & diagnosis using artificial neural networks. *International Journal of COMADEM*, 9(3):15, 2006.

[30] Jie Liu, Youmin Hu, Bo Wu, Yan Wang, and Fengyun Xie. A hybrid generalized hidden markov model-based condition monitoring approach for rolling bearings. *Sensors*, 17(5):1143, 2017.

[31] WY Liu, BP Tang, JG Han, XN Lu, NN Hu, and ZZ He. The structure healthy condition monitoring and fault diagnosis methods in wind turbines: A review. *Renewable and Sustainable Energy Reviews*, 44:466–472, 2015.

[32] Joseph F Murray, Gordon F Hughes, and Kenneth Kreutz-Delgado. Machine learning methods for predicting failures in hard drives: A multiple-instance application. *Journal of Machine Learning Research*, 6(May):783–816, 2005.

[33] Kacker' Raghu N, Eric S Lagergren, and James J Filliben. Taguchi's orthogonal arrays are classical designs of experiments. *Journal of research of the National Institute of Standards and Technology*, 96(5):577, 1991.

[34] Rekha Nachiappan, Bahman Javadi, Rodrigo N Calheiros, and Kenan M Matawie. Cloud storage reliability for big data applications: A state of the art survey. *Journal of Network and Computer Applications*, 97:35–47, 2017.

[35] Hasan Ocak and Kenneth A Loparo. A new bearing fault detection and diagnosis scheme based on hidden markov modeling of vibration signals. In *Acoustics, Speech, and Signal Processing, 2001. Proceedings.(ICASSP'01). 2001 IEEE International Conference on*, volume 5, pages 3141–3144. IEEE, 2001.

[36] Kurt Pichler, Edwin Lughofer, Markus Pichler, Thomas Buchegger, Erich Peter Klement, and Matthias Huschenbett. Fault detection in reciprocating compressor valves under varying load conditions. *Mechanical Systems and Signal Processing*, 70:104–119, 2016.

[37] Guanqiu Qi, Zhiqin Zhu, Ke Erqinhu, Yinong Chen, Yi Chai, and Jian Sun. Fault-diagnosis for reciprocating compressors using big data and machine learning. *Simulation Modelling Practice and Theory*, 80:104–127, 2018.

[38] Qiang Qin, Zhi-Nong Jiang, Kun Feng, and Wei He. A novel scheme for fault detection of reciprocating compressor valves based on basis pursuit, wave matching and support vector machine. *Measurement*, 45(5):897–908, 2012.

[39] Lucas P Queiroz, Francisco Caio M Rodrigues, Joao Paulo P Gomes, Felipe T Brito, Iago C Chaves, Manoel Rui P Paula, Marcos R Salvador, and Javam C Machado. A fault detection method for hard disk drives based on mixture of gaussians and nonparametric statistics. *IEEE Transactions on Industrial Informatics*, 13(2):542–550, 2016.

[40] Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[41] C. A. C. Rincón, J. Pâris, R. Vilalta, A. M. K. Cheng, and D. D. E. Long. Disk failure prediction in heterogeneous environments. In *2017 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, pages 1–7, 2017.

[42] Inc Seagate Technology. Get s.m.a.r.t. for reliability. (TP-67D), July 1999.

[43] Maciej Skowron, Teresa Orlowska-Kowalska, Marcin Wolkiewicz, and Czeslaw T Kowalski. Convolutional neural network-based stator current data-driven incipient stator fault diagnosis of inverter-fed induction motor. *Energies*, 13(6):1475, 2020.

[44] Donald R. Smith. Pulsation, vibration, and noise issues with wet and dry screw compressors. In *Proc. of the Forthieth Turbomachinery Symposium*, pages 170–201, Sept. 2011.

[45] Bing-yu Sun and Moon-chuen Lee. Support vector machine for multiple feature classifcation. In *2006 IEEE International Conference on Multimedia and Expo*, pages 501–504. IEEE, 2006.

[46] Gian Antonio Susto, Andrea Schirru, Simone Pampuri, Seán McLoone, and Alessandro Beghi. Machine learning for predictive maintenance: A multiple classifier approach. *IEEE transactions on industrial informatics*, 11(3):812–820, 2014.

[47] Gian Antonio Susto, Andrea Schirru, Simone Pampuri, Daniele Pagano, Sean McLoone, and Alessandro Beghi. A predictive maintenance system for integral type faults based on support vector machines: An application to ion implantation. In *2013 IEEE international conference on automation science and engineering (CASE)*, pages 195–200. IEEE, 2013.

[48] S. Toor, R. Toebbicke, M. Z. Resines, and S. Holmgren. Investigating an open source cloud storage infrastructure for cern-specific data analysis. In *2012 IEEE Seventh International Conference on Networking, Architecture, and Storage*, pages 84–88, 2012.

[49] Kashi Venkatesh Vishwanath and Nachiappan Nagappan. Characterizing cloud computing hardware reliability. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 193–204, 2010.

[50] Fan Wang, Shuai Tan, Yawei Yang, and Hongbo Shi. Hidden markov model-based fault detection approach for a multimode process. *Industrial & Engineering Chemistry Research*, 55(16):4613–4621, 2016.

[51] Y. Wang, E. W. M. Ma, T. W. S. Chow, and K. Tsui. A two-step parametric method for failure prediction in hard disk drives. *IEEE Transactions on Industrial Informatics*, 10(1):419–430, 2014.

[52] Thorsten Wuest, Daniel Weimer, Christopher Irgens, and Klaus-Dieter Thoben. Machine learning in manufacturing: advantages, challenges, and applications. *Production & Manufacturing Research*, 4(1):23–45, 2016.

[53] Jiang Xiao, Zhuang Xiong, Song Wu, Yusheng Yi, Hai Jin, and Kan Hu. Disk failure prediction in data centers via online learning. In *Proceedings of the 47th International Conference on Parallel Processing*, pages 1–10, 2018.

[54] Chang Xu, Gang Wang, Xiaoguang Liu, Dongdong Guo, and Tie-Yan Liu. Health status assessment and failure prediction for hard drives with recurrent neural networks. *IEEE Transactions on Computers*, 65(11):3502–3508, 2016.

[55] Yong Xu, Kaixin Sui, Randolph Yao, Hongyu Zhang, Qingwei Lin, Yingnong Dang, Peng Li, Keceng Jiang, Wenchi Zhang, Jian-Guang Lou, et al. Improving service availability of cloud systems by predicting disk error. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 481–494, 2018.

[56] Jie Ying, Thia Kirubarajan, Krishna R Pattipati, and Ann Patterson-Hine. A hidden markov model-based algorithm for fault diagnosis with partial and

imperfect tests. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 30(4):463–473, 2000.

[57] Xu Yong et al. Bearings fault diagnosis based on hmm and fractal dimensions spectrum. In *Mechatronics and Automation, 2007. ICMA 2007. International Conference on*, pages 1671–1676. IEEE, 2007.

[58] Jianbo Yu. Adaptive hidden markov model-based online learning framework for bearing faulty detection and performance degradation monitoring. *Mechanical Systems and Signal Processing*, 83:149–162, 2017.

[59] Shafi Md Kawsar Zaman and Xiaodong Liang. An effective induction motor fault diagnosis approach using graph-based semi-supervised learning. *IEEE Access*, 9:7471–7482, 2021.

[60] Israel Zamudio-RamÃrez, Roque Alfredo Osornio-RÃos, Jose Alfonso Antonino-Daviu, and Alfredo Quijano-Lopez. Smart-sensor for the automatic detection of electromechanical faults in induction motors based on the transient stray flux analysis. *Sensors*, 20(5), 2020.

[61] Omid A. Zargar. Hydraulic unbalance in oil injected twin rotary screw compressor vibration analysis (a case history related to iran oil industries). *Int. journal of Mechanical and Mechatronics Engineering*, 7:2371–2377, 2013.

[62] Youmin Zhang, Jin Jiang, Martin Flatley, and Bryan Hill. Condition monitoring and fault detection of a compressor using signal processing techniques. In *Proceedings of the 2001 American Control Conference.(Cat. No. 01CH37148)*, volume 6, pages 4460–4465. IEEE, 2001.

[63] Tiago Zonta, Cristiano André da Costa, Rodrigo da Rosa Righi, Miromar Jose de Lima, Eduardo Silveira da Trindade, and Guann Pyng Li. Predictive maintenance in the industry 4.0: A systematic literature review. *Computers & Industrial Engineering*, 150:106889, 2020.